Abari Kálmán

Adatkezelés és egyváltozós elemzések R-ben

Adatkezelés és egyváltozós elemzések R-ben

Abari Kálmán

A könyvet Máth János lektorálta,
és Friss Kinga illusztrálta.

ISBN 978-963-615-275-8 (pdf)

Kiadta: a Debreceni Egyetemi Kiadó, az 1795-ben alapított
Magyar Könyvkiadók és Könyvterjesztők Egyesülésének a tagja
dupress.unideb.hu
Felelős kiadó: Karácsony Gyöngyi

Tartalomjegyzék

Előszó

Kedves Olvasó!

Köszönjük, hogy bizalmat szavaz könyvünknek, és az R megismeréséhez ezt az utat választja. Az első lépésektől a komplett adatelemzési feladatok megoldásáig vezetjük az Olvasót, és főként kezdő vagy újrakezdő felhasználókhoz szólunk. Utunk során áttekintjük az adatfeldolgozás minden lépését: az adatok beolvasását, előkészítését, elemzését és az eredmények publikálását is.

Könyvünk összesen 11 fejezetet tartalmaz. Az egyes fejezeteket alkotó alfejezeteket három különböző ikon egyikével jelöltük meg, amelyek jelzőtáblaként szolgálnak az R megismerésének útján. Az egyes ikonok jelentése a következő:

> 🙂 Könnyű szint
>
> Egy enyhén mosolygó arc, amely a nyugodt és könnyen érthető tananyagrészeket jelképezi. Az így jelölt fejezet az R alaptudás része, megismerése feltétlenül javasolt. A könyvben megfogalmazott célok ezen fejezetek áttekintésével is elérhetők, azaz komplett adatelemzéseket hajthatunk végre csupán ezek végig olvasásával is.

> 🤔 Közepes szint
>
> A gondolkodó arc tartozik a közepes nehézségű részekhez, amelyek már némi figyelmet és elemzést igényelnek. Ezek olyan kiegészítő tudást tartalmazó fejezetek, amelyek újabb eszközök megismerését teszik lehetővé, és/vagy hozzájárulnak az 🙂 fejezetek mélyebb megértéséhez.

> 🤯 Nehéz szint
>
> A fejrobbanás emoji jól érzékelteti az összetett, kihívást jelentő tananyagrészeket, amelyekért komolyan meg kell dolgozni. Az R ismeretek további részletezése, a meglévő

eszközök finomabb kezelése, vagy további beállítási lehetőségek olvashatók ezekben a fejezetekben. Elképzelhető, hogy ritkábban felmerülő problémák megoldásához kapunk itt segítséget.

A fejezetek hármas tagolása azt a célt szolgálja, hogy minél hamarabb örömet és sikert okozhasson az R használata, ugyanakkor további olvasással a részletesebb ismeretek utáni vágyunkat is kielégíthessük. Könyvünk olvasását tehát az 1. fejezet 🙂 alfejezetével (1.1.) érdemes kezdeni, ott kapunk ajánlást a folytatásra. A további fejezetek olvasási sorrendje teljes mértékben az elvégzendő feladattól, tudásunktól és kíváncsiságunktól függ.

A fejezetek végén összefoglaljuk a tanultakat. Megismételjük a legfontosabb fogalmakat és felsoroljuk a megismert függvényeket.

🗂️ Összefoglalás

A fejezet végén található összefoglalások segítenek a tanultak áttekintésében, és a fejezetben szereplő függvények, parancsok, beállítások gyors visszakeresésében. Érdemes időről időre visszatérni ezekhez az összefoglalókhoz, így ellenőrizhetjük mennyire naprakész az R tudásunk, és eldönthetjük, hogy szükséges-e ismételten átolvasni a fejezet tartalmát.

Az R tanulmányozása kitartást és némi időt igényel. Nagyon fontos szerepet kap a gyakorlás, ezért minden fejezet végén találunk feladatokat.

🎯 Feladatok

A fejezet végi feladatok megoldásával jelentősen hozzájárulunk a magabiztos R tudás megszerzéséhez. Találunk szórakoztató és érdekes feladatokat is.

A könyv teljes forrásanyaga, az adatbázisok és az R-kódok elérhetők a következő címen: https://github.com/abarik/aeer. A feladatok megoldásai a https://abarik.github.io/aeer_m/ oldalon találhatók.

Kívánunk sok türelmet és kitartást az olvasáshoz, reméljük, hamarosan meggyőződik róla, milyen nagyszerű lehetőségeket kínál az R! Örömmel fogadjuk az észrevételeket az abari.kalman@gmail.com címen. Az ábrák és táblázatok elkészítéséhez az *alap R* mellett a következő csomagokat használtuk: `{ggplot2}`, `{summarytools}`, `{gt}`, `{knitr}`, `{kableExtra}`, `{RColorBrewer}` és `{quarto}` (Allaire és Dervieux, 2024; Comtois, 2025; Iannone és mtsai., 2024; Neuwirth, 2022; R Core Team, 2025; Wickham, 2016; Xie, 2024; Zhu, 2024).

Abari Kálmán
2025. április 30.

I. rész
Az R nyelv és környezet

1. Itt kezdődik

– Szóval azt mondod, hogy publikációkész elemzést tudok készíteni, ha elolvasom ezt a könyvet?

1.1. Elindulás 🙂

> **i** Miről lesz szó? Ebben a fejezetben
>
> - áttekintünk egy konkrét adatelemzési példát és a könyv tartalmát,
> - lehetőséget adunk az előzetes R ismeretek felmérésére,
> - és segítünk a megfelelő fejezet kiválasztásában a folytatáshoz.

Könyvünk elsődleges célja az R bemutatása kezdő felhasználók számára, de minden bizonnyal azok is találni fognak hasznos részeket, akik már rendelkeznek R ismeretekkel. Bevezetést nyújtunk az R által lefedett három nagy terület mindegyikébe: az adatkezelésbe, a grafikus megjelenítésbe és az adatelemzésbe is. A leírtak megértéséhez némi statisztikai alapismereten túl semmilyen előzetes tudás nem szükséges.

Most egy konkrét adatelemzési példa segítségével bemutatjuk, hogy mit nyújt e könyv az Olvasó számára. A bevezető példa megoldása során az előismeretekkel rendelkező Olvasó a saját R tudását is felmérheti, és ezzel egyben segítséget kaphat a tudásához és céljaihoz legjobban illeszkedő fejezet kiválasztására, amellyel tovább folytathatja az olvasást.

Példa 1.1 (Két tanítási módszer összehasonlítása)**.** Egy 2020-as kutatásunkban 7. osztályos tanulóknak Excel ismereteket oktattunk két különböző megközelítésben. Az egyik csoportban hagyományos, míg a másikban modern (Sprego) tanítási módszert használtunk. A tanulási időszak az Excel ismeretek felmérésével zárult. Az összegyűjtött adatok az `excel_2020.xlsx` állományban állnak rendelkezésre.
Forrás: Csapó és mtsai. (2020)

Nézzük az adatelemzés lépéseit és egyben könyvünk felépítését!

2. fejezet: Mi az R?
A fenti 1.1. példa megoldását R-ben fogjuk elvégezni (és nem más eszközben, mint például az SPSS, jamovi, JASP, SAS stb.). Érdemes tehát ismerni az R céljait és lehetőségeit, jó ha van egy összképünk a használt statisztikai programcsomagról. Ezt az áttekintést nyújtja a 2. fejezet.

3. fejezet: Az R telepítése.
Adatelemzésünk konkrét lépéseinek elvégzéséhez telepített *Alap R* és *RStudio* szükséges. Ha ezek nem állnak rendelkezésre, vagy még nem is találkoztunk ezekkel az eszközökkel, akkor a 3. fejezet nekünk szól.

4. fejezet: Munka az R-ben.
Az adatelemzés végrehajtásához az *RStudio*-t ajánljuk, és azon belül pedig a projektek hasz-

nálatát szorgalmazzuk. A 4. fejezetben megismerjük az *RStudio* legalapvetőbb funkcióit, a parancsállományok létrehozását és futtatását.

A fenti előzmények után elkezdhetjük a bevezető példa megoldását:

1. indítsuk el az *RStudio*-t,
2. hozzunk létre egy új projektet,
3. hozzunk létre egy új *Quarto* állományt,
4. helyezzük el a lentebb szereplő R parancsokat az állomány egyes R csonkjaiban.

5. fejezet: Az R nyelv.
Az R parancsok létrehozásának vannak szabályai, amelyeket a munka során be kell tartanunk. Ismernünk kell jó néhány függvényt, és általában el kell tudnunk igazodni az R nyelvben. Az 5. fejezet ezért kulcsfontosságú, tanulmányozzuk alaposan, és lehetőleg minden kitűzött feladatát oldjuk meg.

6. fejezet: Beolvasás.
Minden adatelemzés első lépése az adatállomány beolvasása. Adataink változatos formában állhatnak rendelkezésre, a 6. fejezetben ezek beolvasására kapunk receptet.

A bevezető példa megoldásához a *Quarto* állomány egyik csonkját bővítsük a lenti sorokkal.

```
# install.packages('rio') # rio csomag telepítése
library(rio)  # rio csomag betöltése
felmeres <- import(file = "adat/excel_2020.xlsx")  # beolvasás
```

7. fejezet: Adatkezelés.
A statisztikai elemzés elkezdése előtt számos adatkezelési tevékenységre lehet szükség. Ezt a sokszor rendkívül időigényes folyamatot a 7. fejezetben részletezzük.

A bevezető példa megoldásához a *Quarto* állomány egyik csonkját bővítsük a lenti sorokkal. Az adatkezelés legtöbbször a beolvasott állomány jellemzőinek lekérésével kezdődik.

```
str(felmeres)              # az adatmátrix szerkezete
#> 'data.frame':    26 obs. of  2 variables:
#>  $ modszer : chr  "modern" "modern" "modern" "modern" ...
#>  $ eredmeny: num  0.662 0.326 0.645 0.888 0.823 ...
names(felmeres)            # változónevek
#> [1] "modszer"  "eredmeny"
unique(felmeres$modszer)   # különböző értékek
#> [1] "modern"      "hagyományos"
```

A karakteres vagy numerikus vektorok faktorrá konvertálása az egyik leggyakoribb előkészítő parancs.

```
felmeres$modszer <- factor(felmeres$modszer)
```

A táblázatok és ábrák megfelelő megjelenéséhez, végezzük el a faktorszintek sorrendbe állítását.

```
felmeres$modszer <- factor(felmeres$modszer,
                           levels=c("modern", "hagyományos"))
```

8. fejezet: Mutatók és táblázatok.
Ha az adatainkat már megfelelő formába hoztuk, akkor továbbléphetünk az elemzés felé. A 8. fejezet a leíró statisztikai elemzésekből a mutatók és a táblázatok létrehozását mutatja be.

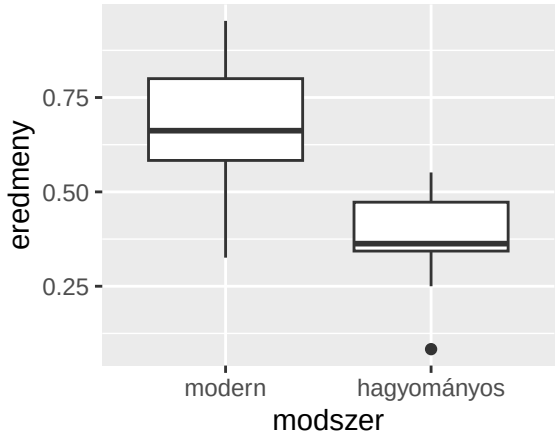
Most a felmérés eredményeinek statisztikai mutatóit íratjuk ki a két tanítási módszert használó csoportban.

```
# install.packages("psych") # psych csomag telepítése
psych::describeBy(x = felmeres$eredmeny, group = felmeres$modszer,
                  mat=T, fast=T, digits = 2)
#>     item      group1 vars  n mean   sd median  min  max range  skew
#> X11    1      modern    1 13 0.65 0.20   0.66 0.33 0.95  0.63 -0.29
#> X12    2 hagyományos    1 13 0.38 0.13   0.36 0.08 0.55  0.47 -0.64
#>     kurtosis   se
#> X11    -1.19 0.05
#> X12    -0.11 0.04
```

9. fejezet: Grafika.
A grafikus megjelenítés is a leíró statisztikai elemzés része. A 9. fejezetben részletesebben olvashatunk a publikációkész ábrák létrehozásáról.

Most a numerikus változók esetén használt egyik elterjedt ábrázolási formát, a dobozdiagramot használjuk két tanítási csoport eredményének grafikus összehasonlítására.

```
library(ggplot2)
ggplot(data = felmeres, mapping = aes(x=modszer, y=eredmeny)) +
  geom_boxplot()
```



10. fejezet: Hipotézisvizsgálatok.
A statisztikai hipotézisvizsgálat minden adatelemzés központi része, a gyűjtött adatokból a populációra nézve következtetést vonhatunk le. A 10. fejezetben a leggyakoribb egyváltozós elemzéseket mutatjuk be.

Most Mann–Whitney-próbát hajtunk végre a két tanítási módszer eredményességének összehasonlítására.

```
# Mann--Whitney-próba
wilcox.test(eredmeny ~ modszer, data = felmeres)
#>
#>  Wilcoxon rank sum exact test
#>
#> data:  eredmeny by modszer
#> W = 145, p-value = 0.001248
#> alternative hypothesis: true location shift is not equal to 0
```

11. fejezet: Publikálás.
Az adatelemzési folyamat utolsó lépése, az elemzés eredményének publikációkész formába öntése. A 11. fejezetben megismerjük azokat a legegyszerűbb folyamatokat, amelyekkel többnyire formanyelvtől függetlenül, publikációkész eredményközlést végezhetünk.

Most a bevezető példában kapott eredmények publikálását végezzük el. A korábban használt `psych::describeBy()` függvény hívását úgy módosítjuk, hogy az bármely formanyelven (PDF, HTML, DOCX) megfelelő eredményt adjon. Ehhez mindössze egészítsük ki a következő sorokkal a leíró statisztikai elemzést, majd a *Render* nyomógomb segítségével fordítsuk le a *Quarto* állományt. A leíró statisztikai mutatók máris táblázatos, könnyen áttekinthető formában jelennek meg.

```
options(OutDec = ",")  # a tizedesjel beállítása
st <- psych::describeBy(x = felmeres$eredmeny,
                        group = felmeres$modszer,
                        mat=T, fast=T, digits = 2)
st |>
  dplyr::select(Csoport=group1,
                N=n, Átlag=mean, SD=sd,
                Min.=min, Max.=max) |>
  gt::gt() |>
  gt::tab_header(
    title = gt::md("Leíró statisztikai adatok"),
    subtitle = gt::md("A **Sprego adatok** elemzése")
    ) |>
  gt::tab_options(latex.use_longtable = TRUE)
```

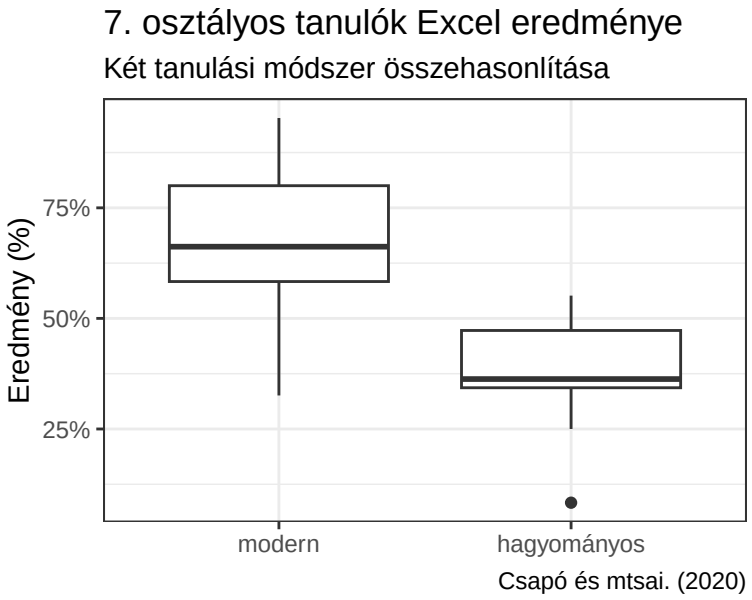
Leíró statisztikai adatok

A **Sprego adatok** elemzése

Csoport	N	Átlag	SD	Min.	Max.
modern	13	0,65	0,20	0,33	0,95
hagyományos	13	0,38	0,13	0,08	0,55

Publikációnk szerves része a magyarázó ábra. A korábban rajzolt dobozdiagramunkat csinosítsuk ki a következő sorok R csonkba helyezésével. A `ggsave()` függvény a háttértárra rögzítésről is gondoskodik.

```
library(ggplot2)
p1 <- ggplot(data = felmeres, mapping = aes(x=modszer, y=eredmeny)) +
  geom_boxplot() +
  labs(x=NULL, y="Eredmény (%)",
       title="7. osztályos tanulók Excel eredménye",
       subtitle = "Két tanulási módszer összehasonlítása",
       caption = "Csapó és mtsai. (2020)") +
  scale_y_continuous(labels = scales::percent) + theme_bw()
ggsave(filename = "output/kep/tanulasi_modszer.png", plot = p1)
p1
```



A bevezető példa megoldásához természetesen a hipotézisvizsgálat szöveges értékelése is hozzátartozik, de ezt most az alfejezet végén szereplő egyik kitűzött feladatra halasztjuk. A hangsúly a könyv vázlatos tartalomjegyzékének bemutatásán volt, részletesebb, de felsorolásszerű tartalomjegyzéket a következő két alfejezetben találunk.

Összefoglalás

Ebben az alfejezetben egy adatelemzési példát oldottunk meg, melynek segítségével illusztrálni tudtuk a további fejezetek tartalmát. A 2. fejezetben áttekintést adunk az R-ről, a 3.-ban az *Alap R* és *RStudio* telepítését, a 4.-ben az *RStudio* használatát mutatjuk be. Az 5. fejezetben kellő részletességgel ismertetjük az R nyelvet. A további fejezetekben az adatelemzés szokásos lépéseit vesszük sorra, a 6. fejezetben a beolvasást, a 7. fejezetben az adatok előkészítését, a 8. és 9. fejezetben a leíró statisztikai műveleteket mutatjuk be. A 10. fejezet az egyváltozós hipotézisvizsgálatoké, az utolsó, 11. fejezet az eredmények publikálását foglalja össze.

Feladatok

1. Milyen online vagy nyomtat könyvek segítik az R elsajátítását? Próbáljuk összegyűjteni a magyar nyelvű könyveket is!
2. Térképezzük fel az online videókurzusokat is az R tanulásához!
3. Az 1.1. példa (*Két tanítási módszer összehasonlítása*) hipotézisvizsgálatának eredménye alapján adjunk szöveges értékelést!

1.2. A könyv felépítése 🤔

> ℹ️ Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük a könyv részletes felépítését,
> - és ezzel tovább segítjük a választást a folytatáshoz.

A könyv 11 fejezetből áll, és fejezetenként 3 vagy több alfejezetből. Most röviden bemutatjuk az egyes alfejezetek tartalmát. A szimbólumokkal utalunk az egyes fejezetek tartalmára, nehézségi szintjére.

1. Itt kezdődik

- **1.1. Elindulás** 🙂
  A könyv fejezeteinek bemutatása egy konkrét adatelemzésen keresztül (1.1).
- **1.2. A könyv felépítése** 🤔
  Jelen alfejezet, amelyben a könyv egyes alfejezeteit mutatjuk be röviden.
- **1.3. Próbák listája** 🤯
  A könyvben szereplő egyváltozós statisztikai eljárások listája (1.3).

2. Mi az R?

- **2.1. Az R bemutatása** 🙂
  A parancssoros R jellemzői, az R nyelv, az *Alap R* és a csomag fogalma (2.1).
- **2.2. A modern R** 🤔
  Megtanuljuk a *Tidyverse R* fogalmát, megtudjuk mi a modern R (2.2).
- **2.3. Múlt és jelen** 🤯
  Az R rövid története, alapelvek az R tanulásához, és az R alaptudás elemei (2.3).

3. Az R telepítése

- **3.1. Az *Alap R* és az *RStudio* telepítése** 🙂
  Megismerjük az *Alap R* és az *RStudio* telepítését (3.1).
- **3.2. A *Tidyverse R* telepítése** 🤔
  A `{tidyverse}` csomag(gyűjtemény) telepítése (3.2).
- **3.3. Az R frissítése** 🤯
  Az *Alap R*, az *RStudio* és a csomagok frissítésének módszerei (3.3).

4. Munka az R-ben

8. Mutatók és táblázatok

- **8.1. Az *Alap R* lehetőségei** 🙂
  Az alapértelmezett mutató és táblázat kiírató függvények az R-ben (8.1).
- **8.2. A *Tidyverse R* lehetőségei** 🤔
  Mutatók és táblázatok a *Tidyverse R*-ben (8.2).
- **8.3. Haladó lehetőségek** 🤯
  Speciális csomagok mutatókra és táblázatokra (8.3).

9. Grafika

- **9.1. A ggplot2 alapjai** 🙂
  A ggplot2 alapelve és alapvető ábratípusai (9.1).
- **9.2. Számítások az ábrán** 🤔
  Automatikus és direkt összesítések megjelenítése ábrán (9.2).
- **9.3. Ábrák testreszabása** 🤯
  Publikációkész ggplot2 ábrák beállításai (9.3).

10. Hipotézisvizsgálatok

- **10.1. Paraméteres próbák** 🙂
  T-próbák, varianciaelemzés, korreláció- és regressziószámítás (10.2).
- **10.2. Nemparaméteres próbák** 🙂
  Az ismert rangpróbák (10.3).
- **10.3. Normalitás vizsgálata** 🙂
  Próbák a normális eloszlás vizsgálatára (10.4).
- **10.4. Varianciára vonatkozó próbák** 🙂
  A szóráshomogenitásra vonatkozó klasszikus próbák (10.5).
- **10.5. Valószínűség próbái** 🙂
  Próbák populációbeli arányokra (10.6).
- **10.6. Hatásméret** 🤔
  Hatásméret meghatározásának esetei (10.7).
- **10.7. Statisztikai erő és mintanagyság** 🤔
  A statisztikai erő és mintanagyság kiszámítása (10.8).
- **10.8. Jamovi az R-ben** 🤯
  A jamovi funkció R-ben a `{jmv}` csomagon keresztül (10.9).

11. Publikáció

- **11.1. Quarto HTML dokumentum** 🙂
  A *Quarto* publikációs rendszer megismerése, HTML dokumentumok létrehozása (11.1).
- **11.2. Hivatkozások** 🤔
  Hivatkozás képekre, ábrákra és más tudományos írásokra a *Quarto* doumentumon belül (11.2).
- **11.3. Más Quarto formátumok** 🤯
  A *Quarto* lehetőségei PDF, Word és könyv formátumú tudományos írások létrehozására (11.3).

🗂️ Összefoglalás

Ebben a részben röviden bemutattuk a könyv összes alfejezetét. A későbbiekben térképként használhatja az Olvasó az itt ismertetett felsorolást.

🎯 Feladatok

1. Az adatfeldolgozás 4 lépése a következő: (1) adatok beolvasása, (2) adatok előkészítése elemzésre, (3) adatok elemzése és (4) az eredmények publikálása. A könyv mely fejezetei tartoznak az adatfeldolgozás fenti lépéseihez?
2. Az R-rel való munka általunk javasolt módja: *RStudio*-ban, projektmódban, R vagy *Quarto* állományokat szerkesztünk és hajtunk végre. Mely fejezetekben találunk hasznos információkat az R ezen használatával kapcsolatban?

1.3. Próbák listája 🤯

ℹ️ Miről lesz szó? Ebben a fejezetben

- áttekintjük a könyvben szereplő egy- és kétváltozós hipotézisvizsgálatokat.

A 10. fejezetben bemutatjuk az egy- és kétváltozós hipotézisvizsgálatok végrehajtását. Ebben a fejezetben felsoroljuk a legfontosabb próbákat:

- egy mintát vizsgáló próbák (1.2. táblázat),
- páros mintát vizsgáló próbák (1.3. táblázat),
- két független mintát vizsgáló próbák (1.4. táblázat),
- több összetartozó mintát vizsgáló próbák (1.5. táblázat),
- több független mintát vizsgáló próbák (1.6. táblázat),

- korreláció- és regressziószámítás (1.7. táblázat).

A táblázatokban megadjuk, hogy a vizsgálatnak mi a célja, vagyis a populációbeli változó(k) melyik paraméterére vonatkoznak a próbák, a várható értékre, a mediánra, a varianciára vagy a valószínűségre. A 10. fejezetben foglalkozunk az eloszlásvizsgálatok közül a normalitást ellenőrző próbákkal is, így az 1.2. táblázat ezeket is számba veszi.

1.2. táblázat: Egy minta vizsgálata *(saját szerkesztés)*

Cél	Próba neve	R függvény
várható érték	egymintás t-próba	`t.test()`
medián	egymintás előjel-próba	`DescTools::SignTest()`
	egymintás Wilcoxon-próba	`wilcox.test()`
variancia	egymintás khí-négyzet próba	`TeachingDemos::sigma.test()`
valószínűség	egzakt binomiális próba	`binom.test()`
	aránypróba	`prop.test()`
	egzakt multinominális-próba	`RVAideMemoire::multinomial.test()`
	khí-négyzet próba	`chisq.test()`
	G-próba	`RVAideMemoire::G.test()`
normalitás	Shapiro-Wilk próba	`shapiro.test()`
	Kolmogorov-Smirnov próba	`DescTools::LillieTest()`
	D'Agostino-próba	`fBasics::dagoTest()`

1.3. táblázat: Páros minta vizsgálata *(saját szerkesztés)*

Cél	Próba neve	R függvény
várható érték	páros t-próba	`t.test(paired=T)`
medián	páros előjel-próba	`DescTools::SignTest()`
	páros Wilcoxon-próba	`wilcox.test(paired=T)`
variancia	Pitman–Morgan próba	`PairedData::Var.test()`
valószínűség	McNemar-próba	`mcnemar.test()`
	McNemar-–Bowker-próba	`mcnemar.test()`

1.4. táblázat: Két független minta vizsgálata *(saját szerkesztés)*

Cél	Próba neve	R függvény
várható érték	kétmintás t-próba	`t.test(var.equal=T)`
	Welch-féle d-próba	`t.test(var.equal=F)`

Cél	Próba neve	R függvény
medián	Mann–Whitney-próba	`wilcox.test()`
	kétmintás Mood medián-próba	`RVAideMemoire::mood.medtest()`
variancia	F-próba	`var.test()`
valószínűség	khí-négyzet próba	`chisq.test()`
	Fisher-féle egzakt próba	`fisher.test()`
	G-próba	`RVAideMemoire::G.test()`

1.5. táblázat: Több összetartozó minta vizsgálata (saját szerkesztés)

Cél	Próba neve	R függvény
várható érték	egyszempontos összetartozó mintás varianciaelemzés	`afex::aov_ez()`
medián	Friedman-próba	`friedman.test()`
valószínűség	Cohran-Q próba	`RVAideMemoire::cochran.qtest()`

1.6. táblázat: Több független minta vizsgálata *(saját szerkesztés)*

Cél	Próba neve	R függvény
várható érték	egyszempontos varianciaelemzés	`aov()`
	Welch-féle egyszempontos varianciaelemzés	`oneway.test()`
medián	Kruskal–Wallis-próba	`kruskal.test()`
	többmintás Mood medián-próba	`RVAideMemoire::mood.medtest()`
variancia	Levene-próba	`DescTools::LeveneTest()`
	Bartlett-próba	`bartlett.test()`
	Fligner–Killeen	`fligner.test()`

1.7. táblázat: Korreláció- és regressziószámítás *(saját szerkesztés)*

Cél	Próba/Eljárás neve	R függvény
Pearson r	Korrelációs hipotézisvizsgálat	`cor.test(method ="pearson")`
Kendall TauB	Korrelációs hipotézisvizsgálat	`cor.test(method ="kendall")`
Spearman rho	Korrelációs hipotézisvizsgálat	`cor.test(method ="spearman")`
függés	Egyszerű lineáris regresszió	`summary(lm())`

Összefoglalás

Ebben a részben rövid áttekintést adtunk a könyv 10. fejezetében sorra kerülő statisztikai próbákról. Megneveztük a próbákat, R parancsokkal szemléltettük használatukat, valamint jeleztük a céljukat. A táblázatok áttekintésével képet kaphatunk arról, hogy a későbbiekben milyen jellegű statisztikai következtetéseket tudunk levonni az R használatával.

Feladatok

1. Minden statisztikai próba esetében négy dolgot érdemes tudni: (1) a statisztikai próba neve, (2) null- és ellenhipotézise, (3) alkalmazási feltételei, és (4) a próba végrehajtásának módja valamely statisztikai programcsomagban. A 10. fejezetben a statisztikai próbák végrehajtását természetesen R-beli eszközökkel mutatjuk be. Ismerjük a fenti táblázatokban megnevezett próbák null- és ellenhipotézisét, valamint az alkalmazási feltételeit? Próbáljuk ezeket felidézni! Hol találunk ezekről információt?
2. Mely próbák maradtak ki ebből a könyvből? Hol találunk ezek R-beli végrehajtására példát?

2. Mi az R?

2.1. Az R bemutatása 🙂

Miről lesz szó? Ebben a fejezetben

- megismerjük az R jellemzőit,
- megtudjuk, hogy melyek a parancssoros interfész előnyei,
- megismerjük az *Alap R* fogalmát,
- körülhatároljuk az R nyelv, az *Alap R* és a csomag fogalmát.

2.1.1. Az R jellemzői

Az R egy magas szintű programozási nyelv és környezet, amelynek legfontosabb felhasználása az adatelemzés és az ahhoz kapcsolódó grafikus megjelenítés. Három alapvető jellemzője kiemeli a többi statisztikai programcsomag közül: (1) az R ingyenesen telepíthető és használható; (2) az R nyílt forrású, így bárki hozzájárulhat az R fejlesztéséhez, azaz létrehozhat új csomagokat, és ezzel kiegészítheti az R tudását; és (3) az R felhasználók rendkívül aktív és befogadó online közösséget alkotnak, szinte minden felmerülő kérdésünkre azonnal választ kaphatunk.

Álljon itt egy bővített lista azokról a jellemzőkről, amelyek vonzóvá tehetik számunkra az R statisztikai programcsomagot.

- Az R szabad szoftver, bárki ingyenesen letöltheti és használhatja. Ez egyfelől megkönnyíti az oktatási intézmények, tanszékek és oktatók munkáját, hiszen nincs szükség a kereskedelmi programok licenceléséből adódó pénzügyi vagy más természetű nehézségek kezelésére. Másrészt a hallgatók a statisztika kurzusok során tanultakat otthon vagy később a munkájukban is felhasználhatják.
- Az R platform-független, azaz Windows, Linux és macOS környezetben is használható. Nem kell lemondanunk a kedvenc operációs rendszerünkről, ha az R-t szeretnénk használni.
- Az R nemcsak egy statisztikai programcsomag önmagában, hanem egy teljes értékű programozási nyelv.
- Az R statisztikai módszerek szinte végtelen választékát kínálja. A R-ben felhasználható statisztikai eljárásokat statisztikusok fejlesztik folyamatosan és csomagok formájában teszik elérhetővé. Valószínű, hogy egy új statisztikai módszer leghamarabb az R-ben válik elérhetővé.
- Az R rendkívül gazdag grafikus lehetőségekkel rendelkezik.
- A statisztikai szakirodalomban és az egyetemi oktatók körében egyre elterjedtebb az R – mint közös (statisztikai program)nyelv – használata. Ha valamilyen statisztikai

problémára keressük a megoldást, vagy csak konzultálunk egy statisztikussal, az R ismerete (akár csak olvasási szinten) rendkívüli előnyt jelenthet.
- Az R igen jól dokumentált, a beépített súgón kívül számos könyv és leírás érhető el.
- Az R parancssoros interfésszel rendelkezik, amely számos előnnyel jár. Egyrészt a szkript állományok létrehozása és végrehajtása a statisztikai elemzések megismételhetőségét biztosítják, másrészt ez az oktatók és a hallgatók könnyebb kommunikációját is lehetővé teszi.
- Az R az adatelemzés eredményének sokszínű publikálását is biztosítja. Az Quarto formanyelv segítségével HTML, PDF és MS Word dokumentumot, illetve prezentációs diákat vagy akár kész cikkeket hozhatunk létre. A Shiny csomag interaktív Webes alkalmazások építését teszi lehetővé.
- Mára az R használata szinte egyet jelent az ingyenesen elérhető *RStudio* használatával, amely egy kényelmes integrált fejlesztői környezetet biztosít a parancsállományok létrehozásához.

Érdemes bepillantani az R árnyékosabb oldalába is. Az R egyik gyengesége, hogy nagy adatbázisok kezeléséhez erős hardverre van szüksége, de a legtöbb felhasználás során ez semmilyen problémát nem okoz. A másik gyengeség, hogy az R elsajátításához nem kevés idő és kitartás szükséges. Jelen könyv éppen ezt a folyamatot kívánja megkönnyíteni és lerövidíteni.

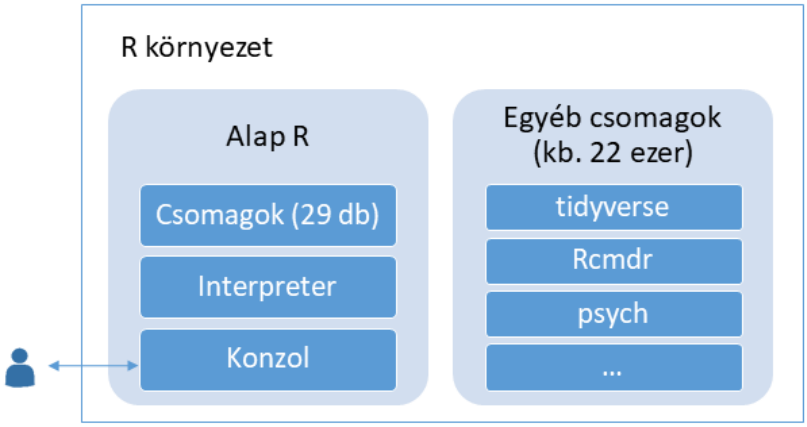
2.1.2. A R parancssoros

Az R alapvető használata során parancsokat gépelünk be és hajtunk végre. Ez lényegesen eltér a ma megszokott felhasználói programok világától, ahol egy grafikus felhasználói felületen egérrel vagy az ujjunkkal mutogatjuk el a kívánt tevékenységet. Az R egészen más megközelítést vall, használata a kezdeti lépésektől nagyfokú figyelmet és pontosságot követel. Parancsokban kell gondolkodnunk, ám ezt végig áthatja a *tudom mit csinálok* elv, így némi idő elteltével érezni fogjuk, hogy az R megszelídül, már nem köt bele minden szavunkba, egyre több dologra tudjuk rávenni, és végül egy rendkívül értékes társsá válik. Jelen könyv ezen az úton szeretné végigvezetni az Olvasót.

Már a tanulás elején szeretnénk tisztázni, hogy az R elsajátításához nem szükséges programozói alaptudás. Az R felhasználók többsége egyáltalán nem programozó, és a mindennapi adatelemző munka sem igényli az R nyelv programozói fokú ismeretét. Természetesen, ha rendelkezünk ilyen irányú előtanulmányokkal a tanulási folyamat néhány szakasza lerövidíthető, de könyvünk elsősorban azok számra íródott, akik programozási nyelvekkel korábban nem találkoztak, és nem is vágynak az R ilyen mélységű ismeretére. Az R nyelv elsajátítása során bevezetjük azokat az egyszerű fogalmakat, amelyeket nem nélkülözhetünk az adatelemzés során, azonban az R programozásához más szakkönyveket javaslunk olvasásra.

2.1.3. Mi valójában az R?

Az R nyelv fejlesztője az R Core Team. Az R nyelv egy rendkívül népszerű szkriptnyelv, több millióan használják világszerte. Elsősorban adatelemzésre, adatmodellezésre és grafikus megjelenítésre, vagyis arra, amit ma adattudományok (data science) alatt értünk. Azonban az R nyelv még önmagában nem szoftver, hanem egy rendkívül rugalmas szkriptnyelv, amely például előírja, hogy milyen szintaktikai szabályok mentén fogalmazhatjuk meg az utasításainkat. Ahhoz, hogy az R nyelvet használni tudjuk, vagyis, hogy a számítógép valóban végre is hajtsa a szintaktikailag helyes utasításainkat, szükség van egy szoftveres környezetre, egy olyan futtató rendszerre, amely a kódunkat értelmezi és végrehajtja.



2.1. ábra: Az R környezet: *Alap R* és az egyéb csomagok *(saját szerkesztés)*

Az R környezet három fő összetevőt tartalmaz (2.1. ábra): (1) egy *konzol*t, ahová a parancsainkat begépelhetjük; (2) a parancsok végrehajtásáért felelős *R interpreter*t; (3) a *csomagok*at. A konzol és az interpreter biztosítja az R nyelven írt parancsok tényleges végrehajtását. Így tudunk adatokat beolvasni, átlagot számolni, varianciaelemzést futtatni, vagy publikációkész ábrákat létrehozni. A csomagok adatokat és függvényeket tartalmaznak, például a `{MASS}` csomag 88 adatobjektumot és 78 függvényt tartalmaz. A függvények valamilyen tevékenységet hajtanak végre, és valójában ezeket a csomag-függvényeket használjuk fel a konzolban, ha *bármilyen tevékenységet* szeretnénk végrehajtani (például adatokat beolvasni, átlagot számolni stb.). A könyv írásának időpontjában kb. 22 ezer csomag volt érhető el az R-hez. Csomagok 3 csoportját különböztetjük meg: *standard csomagok* (14 db), *ajánlott csomagok* (15 db) és *egyéb csomagok* (kb. 22 ezer db). A standard csomagok fejlesztője az R Core Team. A standard csomagok: `{base}`, `{compiler}`, `{datasets}`, `{grDevices}`, `{graphics}`, `{grid}`, `{methods}`, `{parallel}`, `{splines}`, `{stats}`, `{stats4}`, `{tcltk}`, `{tools}`, `{utils}`. Az ajánlott csomagok: `{KernSmooth}`, `{MASS}`, `{Matrix}`, `{boot}`, `{class}`, `{cluster}`, `{codetools}`,

`{foreign}`, `{lattice}`, `{mgcv}`, `{nlme}`, `{nnet}`, `{rpart}`, `{spatial}`, `{survival}`. Az ajánlott csomagok közül a `{foreign}` és az `{nlme}` fejlesztője az R Core Team, a többit más felhasználók fejlesztették, például a már említett `{MASS}` csomag fejlesztője Brian Ripley. Csomagot bárki szabadon fejleszthet és terjeszthet, az *egyéb csomagok* csoportját akár mi is gyarapíthatjuk.

A R környezet már igazi szoftver, terjesztésének koordinálását az R Foundation végzi a CRAN infrastruktúráján keresztül. Ez biztosítja, hogy számítógépünkre telepíthessük az R környezetet. Ezt a CRAN-ról elérhető R futtatási környezetet *Alap R*-nek nevezzük. Fő komponensei a már említett konzol a parancsok begépelésére, az R értelmező a begépelt parancsok végrehajtására és a csomagok közül a standard és ajánlott csomagok. Az *Alap R* telepítése után már tudunk R parancsokat végrehajtani, és nagyon sok adatelemzési probléma megoldására nyílik módunk, sőt azt mondhatjuk, hogy tetszőleges problémát megoldhatunk kisebb-nagyobb erőfeszítéssel, mert az R egy teljes értékű nyelv. Azonban sokszor érdemesebb az *egyéb csomagok* közül választani, hiszen könnyen elképzelhető, hogy a számtalan csomag között találunk olyat, amely segítségünkre lehet speciális feladataink megoldása során. Valószínű, hogy létezik olyan csomag és benne olyan függvény, amely adatkezelési, adatelemzési, grafikai vagy publikálási feladatunkat jelentősen megkönnyíti. Az *egyéb csomagok* csoportjába tartozó csomagok forrása több tárhely is lehet, ezek közül legjelentősebb az R Foundation által karbantartott CRAN (22457 csomaggal), a Bioconductor (2341 csomaggal) és a GitHub.

Az R tehát egyszerre több dolgot jelent. Az R egyrészt egy magas szintű programozási nyelv, hamarosan megtanuljuk, hogyan írjunk ezen a nyelven értelmes utasításokat. Másrészt a nyelv körüli környezetet is jelenti, amely magába foglalja a konzolt, a parancsaink értelmezésért felelős R interpretert, valamint azokat a csomagokat, amelyekkel az R tudása kiegészíthető.

Összefoglalás

Minden statisztikai programcsomag, így az R is, alapvetően a számításigényes statisztikai eljárások kézi végrehajtásától kímél meg minket. Az R nagyon gazdag adatmanipulációs és grafikus funkciókban is, támogatja a reprodukálható adatelemzés végrehajtását. Az R ingyenes, többplatformos és egyik legfontosabb jellemzője, hogy parancsok útján bírhatjuk működésre. Az *Alap R* biztosítja a konzolt a parancsok begépelésére, az R interpretert a parancsok tényleges végrehajtására, és jó néhány csomagba szervezett eljárást az adatelemzési feladatok elvégzéséhez. Az *Alap R* mindössze néhány tucat csomagot tartalmaz, a *standard csomagokat* és az *ajánlott csomagokat*, de több tízezer további csomaggal bővíthetjük az R tudását. Az adatelemzési munka során egy R környezet vesz minket körül, amely az R nyelven megírt parancsok értelmezésére és végrehajtására képes *Alap R*-ből, és az ún. *egyéb csomagokból* áll.

🎯 Feladatok

1. Keressünk weboldalakat, amelyek az R előnyeit és hátrányait listázzák!
2. Keressük meg, hogy az R optimális futtatásához, milyen hardver követelmények szükségesek!
3. Nézzünk utána, hogy ma kb. hány csomag érhető el az R-hez? Keressünk ábrát, amely bemutatja, hogy az évek során hány csomag volt elérhető az R-hez?
4. Hol áll az R népszerűsége a többi programozási nyelvhez, illetve statisztikai programcsomaghoz képest?
5. Milyen ingyenesen elérhető, grafikus felhasználói felülettel rendelkező statisztikai programcsomagok építenek az R-re?
6. Említettük, hogy az adatelemezési munka nem igényli az R programozói fokú ismeretét, de soroljunk fel néhány könyvet, amelyből az R programozása is megtanulható!

2.2. A modern R 🤔

ℹ Miről lesz szó? Ebben a fejezetben

- megismerjük a *Tidyverse R* fogalmát,
- és megtudjuk mit értünk modern R alatt.

Az R egy nem túl fiatal, a funkcionális programnyelvekhez hasonlóan építkező programozási nyelv. A probléma megoldása ilyen nyelvekben tipikusan sokszorosan egymásba ágyazott függvényhívások segítségével történik. Ez sok-sok nyitó és záró kerek zárójellel jár együtt, így a parancsaink áttekintése és karbantartása sokszor nehézségekbe ütközik. Ezt kiküszöbölendő az R-ben előszeretettel használnak procedurális eszközöket (például `for` ciklusokat), de a kód olvashatóságát és karbantartását igazán ez sem könnyíti meg.

A 2014-es év az R nyelv életében meghatározó változást hozott. Egyrészt megjelent a `{magrittr}` csomagban a pipe operátor (`%>%`), amellyel olvashatóbb kódok írására nyílt lehetőség, másrészt a pipe operátorra alapozva Hadley Wickham bemutatta a `{dplyr}` és `{tidyr}` csomagokat. Ezzel az R funkcionális oldalát úgy erősítették meg, hogy a sokszoros egymásba ágyazás során kiküszöbölték a kerek zárójelek írásának problémáját.

Az ebben a szellemben készült csomagok listája bővült az idők folyamán, és a *Tidyverse* nevet kapta ez a csomaggyűjtemény. Jelenleg a következő csomagok alkotják: `{ggplot2}`, `{purrr}`, `{tibble}`, `{dplyr}`, `{tidyr}`, `{stringr}`, `{readr}`, `{forcats}` és `{lubridate}`. Ezek a csomagok nem egyszerűen új funkciókkal ruházzák fel az *Alap R* tudását, mint általában

az *egyéb csomagok*. A *Tidyverse* csomagjai konzisztens módon együttműködnek, és egy új megközelítést hoznak az adatelemzési folyamatok végrehajtásában és a kódok írásában. Rövidebb idő alatt hozhatunk létre könnyebben karbantartható kódokat, és a műveleteink végrehajtása is rendszerint gyorsabb. Amikor ebben a megközelítésben hozzuk létre és hajtjuk végre utasításainkat, akkor azt mondjuk hogy a *Tidyverse R*-t használjuk. A *Tidyverse R* nem helyettesíti az *Alap R*-t, és csak bizonyos feladatokra használható. Lássunk tisztán, amit elvégezhetünk *Tidyverse R*-ben, azt az *Alap R*-ben is meg tudnánk tenni, de valószínűleg több gépeléssel, lassabb és rosszabbul karbantartható kóddal.

Eddig láttuk, hogy az R használatához szükséges az *Alap R* telepítése, majd a speciális problémánknak megfelelően kiegészíthetjük az R tudását úgy, hogy telepítünk egyet vagy többet az *egyéb csomagok* kategóriájából. Választhatjuk akár a *Tidyverse* csomagjait is telepítésre, ugyanis így lehetőségünk nyílik a *Tidyverse R* használatára. Utasításaink megfogalmazásának ma ez a legmodernebb módja. A *Tidyverse R*-nek nagy hatása volt magára az R nyelvre is, mára a pipe operátor (`|>` formában) az R nyelv beépített részévé vált.

A modern R alatt lényegében azokat a funkciókat értjük, amelyek a *Tidyverse* gyűjteményben található csomagokhoz kötődnek. Ezekkel a csomagokkal, gyorsabb, olvashatóbb és könnyebben karbantartható kódokat hozhatunk létre. A *Tidyverse* használata tehát erősen javasolt, de ebben a könyvben a "hagyományos", *Tidyverse R* előtti lehetőségeket is bemutatjuk.

Összefoglalás

A *Tidyverse R* egy csomaggyűjtemény az *egyéb csomagok* csoportjából, amely újabb szemléletű R parancsok írására ad lehetőséget. Az így készült kódjaink rendszerint gyorsabban futnak és könnyebben karbantarthatók. A modern R a *Tidyverse R* csomagjaival kiegészített *Alap R*, de legfőképp egy új lehetőség parancsaink megfogalmazására.

Feladatok

1. Ki Hadley Wickham?
2. Mikor történt az egyik legjobb dolog az R-rel?

2.3. Múlt és jelen 🤯

> Miről lesz szó? Ebben a fejezetben
>
> - megismerjük az R rövid történetét és annak szereplőit,
> - majd egy szubjektív listával segítjük az R tanulását,
> - illetve megismerjük az R alaptudás elemeit.

2.3.1. Szereplők és fogalmak

Érdemes néhány szereplőt és fogalmat tisztázni az R világán belül. Az R nyelvet 1992-ben kezdte fejleszteni Ross Ihaka és Robert Gentleman, 1997-től pedig egy nagyobb csapat, az R Development Core Team vezeti a fejlesztést (rövidebben *R Core Team*). Ettől az évtől az R hivatalosan a GNU projekt része. Az *R Core Team* tagjai 2002-ben létrehozták a The R Foundation for Statistical Computing (rövidebben *The R Foundation*) közhasznú, nonprofit szervezetet, amelynek célja (1) az R folyamatos fejlesztésének biztosítása, és ehhez kapcsolódóan a nyílt forráskódú számítógépes statisztikai innovációk támogatása, (2) az R fejlesztői közösség (*R Core Team*) hivatalos hangjaként a felhasználók, intézmények és üzleti vállalkozások számára a kommunikáció biztosítása, és (3) az R program és dokumentációk szerzői jogainak kezelése. A szervezet rendszeresen konferenciákat, találkozókat szervez, referált folyóiratot, kézikönyveket és technikai leírásokat ad ki, valamint fenntart egy számítógépes infrastruktúrát (ez a CRAN, amely levelező listákat, FTP- és Webszervereket üzemeltet). Az R Foundation hivatalos oldala – egyben az R hivatalos oldala – a *https://www.r-project.org/*. Az R Foundation (és más önkéntesek) által üzemeltetett számítógépes hálózat neve a CRAN (Comprehensive R Archive Network), amely szabad hozzáférést nyújt az R legfrissebb verziójához, az R kiterjesztéseihez (a csomagokhoz) és a részletes dokumentációkhoz. A CRAN fő számítógépe Ausztriában található *https://CRAN.R-project.org/*, azonban nagyon sok naponta frissülő tükörszerver érhető el világszerte.

2.3.2. Alapelvek

Az R elsődleges célja, hasonlóan más statisztikai programcsomagokhoz, a statisztikai adatelemzés, amelyet négy lépésre bonthatunk:

1. adatok beolvasása,
2. adatok előkészítése elemzésre,
3. adatelemzés,

4. eredmények publikálása.

Az R mára a fenti 4 tevékenység elvégzését teljes körűen támogatja. A könyv célja ezek bemutatása. Mielőtt elkezdjük ezt az izgalmas utat – az R tanulmányozását – néhány alapelvet szeretnék megemlíteni, ami segíthet minket az utazásunk során:

- *Magabiztosság* - Az R nagyon nagy, így a teljes megismerése nem lehet célunk. Mindig lesz valaki, aki az R egyik vagy másik részét jobban, vagy kevésbé ismeri nálunk. Ez természetes, ezen soha ne csodálkozzunk. Az eltérő ismeretek azonban az R speciális területeire vonatkoznak, az *R alaptudás* (2.3.3. fejezet) minden R-ben jártas felhasználó számára közös. E könyv célja ennek az alaptudásnak az átadása, melynek birtokában már kellő magabiztossággal vághatunk neki az R azon részeinek elsajátításába, amelyek az éppen elénk kerülő speciális feladat megoldásához szükségesek. Hisszük, hogy e könyv elolvasásával, mind az R alaptudás, mind a kellő magabiztosság elérhetővé válik számunkra.
- *Gyakorlás* - Az R alaptudásának megszerzése némi időbe telik, ez tagadhatatlan. A motiváció megtartásához viszonylag jól kell éreznünk magunkat a tanulás és a gyakorlás során. A könyvben ezért minden fejezet végén találunk megoldandó feladatokat, amelyek között szórakoztató, érdekes és kihívást jelentő gyakorlatok is szerepelnek.
- *Svájci bicska* - A R nagyon sokféle statisztikai és nem-statisztikai probléma megoldására képes, sőt ugyanarra a problémára nagyon sok különböző eszközt kínál. Ha elsőre nem a legszebb, legoptimálisabb megoldás jut az eszünkbe, ne csüggedjünk, ez a legtöbb esetben nem jelent gondot. Azon se csodálkozzunk, ha korábban megoldott problémánkra idővel újabb és újabb megoldási lehetőségeket találunk.

2.3.3. Az R alaptudás

Melyek az R-ben való munkavégzéshez nélkülözhetetlen alapismeretek? Meggyőződésünk, ha a lentebb felsorolt témakörökkel tisztában vagyunk, akkor már magabiztos R tudással rendelkezünk, és bármilyen további R témakör könnyen elsajátítható lesz. Ezekre az ismeretekre úgy gondolhatunk, mint egy ablakra, amelyen keresztül az R szinte végtelen lehetőségeinek tárháza nyílik meg előttünk. Később visszatérhetünk ehhez a listához, és ellenőrizhetjük, hány elemet tudunk már kipipálni.

Az R alaptudás elemei:

- Az R környezet alapszintű ismerete
  - ☐ az *Alap R*, az *RStudio* és a csomagok telepítése
  - ☐ projektek használata és R parancsok futtatása az *RStudio*-ban
- Az R nyelv alapszintű ismerete

- ☐ konstansok írása
- ☐ objektumok kezelése
- ☐ egyszerű adattípusok
- ☐ alapvető operátorok
- ☐ kifejezés fogalma
- ☐ a függvényhívás lehetőségei
- ☐ összetett adattípusok,
- ☐ a vektoraritmetika szabályai

- Az alapvető függvények ismerete
  - ☐ csomagkezelő függvények
  - ☐ a munkaterület függvényei
  - ☐ matematikai függvények
  - ☐ input/output függvények
  - ☐ indexelés, szűrés, rendezés
  - ☐ információ kérés az objektumokról
  - ☐ egyszerű típuskonverzió
  - ☐ transzformáció
  - ☐ ismétlő és összesítő függvények
  - ☐ a hagyományos grafika néhány eleme
  - ☐ a ggplot2 alapszintű ismerete

- Egyéb ismeretek
  - ☐ szövegszerkesztési és állománykezelési ismeretek
  - ☐ a tagolt szöveges állomány fogalma
  - ☐ reprodukálható kutatás a *Quarto* segítségével

Összefoglalás

Az R fejlesztését Ross Ihaka és Robert Gentleman kezdte, majd 1997-től egy nagyobb csapat, az *R Development Core Team* vezeti a fejlesztést. Az *R Core Team* tagjai 2002-ben létrehozták a *The R Foundation for Statistical Computing* közhasznú, nonprofit szervezetet, amelynek fő célja az R folyamatos fejlesztésének biztosítása. A szervezet fenntart egy CRAN nevű számítógépes hálózatot, amely szabad hozzáférést biztosít az R legfrissebb verziójához, a csomagokhoz és a részletes dokumentációkhoz.
Az R alaptudás megszerzése elegendő magabiztosságot fog nyújtani az adatelemzési munka során, azonban vegyük figyelembe, hogy ezt csak kellő gyakorlással érhetjük el. Az R sokféle megoldást biztosít ugyanarra a problémára, legyen az statisztikai vagy bármilyen más jellegű feladat.

🎯 Feladatok

1. Keressünk olyan statisztikai jellegű témaköröket, amelyekben az R segítségünkre lehet?
2. Keressünk olyan nem-statisztikai jellegű témaköröket, amelyekben az R segítségünkre lehet?
3. Nézzünk át néhány online elérhető R könyvet, és hasonlítsuk össze az R alaptudás egyes elemeivel! Melyek az átfedő részek, és hol vannak különbségek?
4. Melyek a fontosabb lépcsőfokok az R fejlődésében?

3. Az R telepítése

"ha a kéket veszed be, a játéknak vége... felébredsz az ágyadban, azt hiszel, amit hinni akarsz. De ha a pirosat, maradsz Csodaországban, és én megmutatom, hogy milyen mély a nyúl ürege."

3.1. A fő komponensek telepítése 🙂

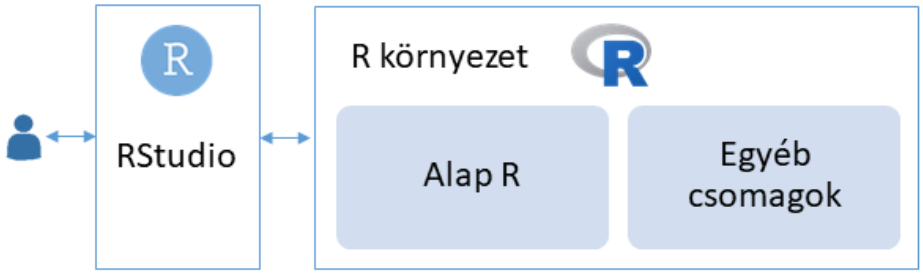
> **Miről lesz szó? Ebben a fejezetben**
>
> - megismerjük az *Alap R*,
> - az *RStudio*,
> - és a csomagok telepítését.

A korábbi fejezetekben megismertük az R világának néhány fogalmát és szereplőjét. Tudjuk, hogy az R nyelv használatához megfelelő szoftveres környezetre van szükség, amely magába foglalja az *Alap R*-t és az *egyéb csomagok* kategóriájából esetlegesen telepített csomagokat is. Az R már ezen eszközök birtokában is teljes körűen használható, azonban egy újabb ingyenes eszköz, az *RStudio*, kényelmessé és hatékonnyá teszi az adatelemzési munkát.

Könyvünk legfontosabb gondolata: **ma akkor tudjuk a legjobban kihasználni az R lehetőségeit, és ezzel egyidőben a legkényelmesebb módon elvégezni az adatelemzési feladatunkat, ha**

1. **az *Rstudio*-t használjuk,**
2. **projekt üzemmódban dolgozunk, és**
3. ***Quarto* állományokban rögzítjük az R parancsainkat.**

Ezt a szemléletet következetesen képviseljük az egyes fejezetekben, és a későbbiekben részletesebben bemutatjuk, hogyan tudjuk mindezt megvalósítani (3.1. ábra).



3.1. ábra: Az R kényelmes használata *(saját szerkesztés)*

A R kényelmes használatához a legelső lépés a szoftveres környezet egyes elemeinek telepítése. Három fő komponens telepítésére lesz szükségünk:

1. *Alap R*, amely tartalmazza a konzolt, az R interpretert, illetve a *standard csomagokat* és az *ajánlott csomagokat*,

2. *RStudio*, amely egy új konzollal "eltakarja" az *Alap R*-t, és kényelmesebb hozzáférést biztosít az *Alap R* interpreteréhez és a csomagjaihoz.
3. Csomagok, amelyek az *egyéb csomagok* nagy halmazából származnak, és telepítésükkel újabb és újabb képességekkel ruházzuk fel az *Alap R*-t.

3.1.1. Az *Alap R* telepítése

Az *Alap R* telepítéséhez látogassunk el az R hivatalos letöltő oldalára: https://cran.r-project.org/. Az operációs rendszerünknek megfelelő link kiválasztásával folytassuk a navigálást.

- A Windows felhasználók a `Download R for Windows` linken, majd a `base` linken kattintva jutnak el a telepítőprogram linkjéhez: `Download R X.X.X for Windows`. A sikeres letöltés után indítsuk el a telepítőt, és az alapértelmezetten felajánlott opciók nyugtázásával végezzük el a telepítést. A telepítést lehetőleg olyan Windows felhasználó alatt végezzük el, amelynek a neve sem ékezetes karaktert, sem szóközt, sem egyéb írásjelet nem tartalmaz.
- A macOS felhasználók a `Download R for (Mac) OS X` linken kattintva jutnak a telepítőhöz: `R-X.X.X.pkg`. A letöltés után indítsuk el a telepítőt, és a `Next` gombok segítségével végezzük el a telepítést.
- A Linux felhasználók az aktuális R verzió telepítéséhez a `Download R for Linux` linken keresztül jutnak el, ahol a megfelelő disztribúció (Debian, Redhat, Suse, Ubuntu) kiválasztása után konkrét információkat kapnak a telepítésről.

3.1.2. Az *RStudio* telepítése

Az *RStudio* telepítéséhez az operációs rendszerünknek megfelelő telepítőt kell letöltenünk a https://posit.co/download/rstudio-desktop/ oldalról. Az *RStudio Desktop* változatra lesz szükségünk, töltsük le és telepítsük ezt a számítógépünkre. A telepítés során fogadjuk el az alapértelmezett opciókat. Az *RStudio* automatikusan megtalálja és használja a korábban telepített *Alap R* példányunkat, így a későbbiekben elegendő lesz az *RStudio*-t használni, azon keresztül elérhetjük az *Alap R* minden funkcióját (3.1. ábra).

3.1.3. Csomagok telepítése

A csomagok telepítésére az *Alap R* vagy az *RStudio* elindítása után van módunk. Érdemes a telepítéseket az *RStudio*-ból végezni. A csomag fellelési helye alapján, három különböző tárhelyről mutatjuk be a csomagok telepítését. Látni fogjuk, hogy a csomagok telepítéséhez R parancsokat fogunk használni. Ha még nem vagyunk jártasak R parancsok futtatásban,

akkor a 4.1. fejezet fellapozásával segítséget kaphatunk a lenti parancsok kipróbálásához, de úgy is eljárhatunk, hogy most kihagyjuk ennek a résznek a kipróbálását, és később térünk vissza, amikor valóban felmerül az igény csomagok telepítésére.

Az R csomagok hivatalos helye a CRAN (The Comprehensive R Archive Network). A CRAN számítógépei tárolják a nyílt forráskódú R nyelv és környezet különböző verzióinak kódjait és dokumentációit, így az összes R csomag forráskódját is. Egy bírálási folyamat után bármely felhasználó csomagja a CRAN-ból is elérhető lehet.

Az *Alap R* vagy az *RStudio* elindítása után az `install.packages()` függvénnyel tölthetünk le és telepíthetünk csomagot a CRAN-ról. Tetszőleges csomag telepítéséhez írjuk a csomag nevét idézőjelekben a függvény argumentumába:

```
install.packages("csomag_neve")
```

A `{psych}` csomagot, amely a pszichológia kutatások adatainak elemzéséhez nyújt segítséget, például így telepíthetjük:

```
install.packages("psych")        # psych csomag telepítése
```

A csomagok másik fontos forrása a Bioconductor, ahol alaposan tesztelt és igen jól dokumentált bioinformatikai témájú csomagokat találunk. Az innen elérhető csomagokat – például most a `{DESeq2}` csomagot az RNS-szekvenálási elemzésekhez – a következő parancsokkal telepíthetjük:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("DESeq2")
```

A csomagok harmadik fő forrása a GitHub. A felhasználók a saját fejlesztésű csomagjaikat rendszerint először a GitHub-on keresztül teszik elérhetővé. Ha ezeket a csomagokat szeretnénk kipróbálni, akkor a felhasználó és a csomag nevének birtokában a következő parancsot kell kiadnunk:

```
devtools::install_github("felhasznalo_neve/csomag_neve")
```

Például a GitHub-ról telepíthető `{emo}` csomag segítségével hangulatjeleket szúrhatunk be a *Quarto* állományainkba. Ezzel a sorral telepíthetjük a csomagot:

```
devtools::install_github("hadley/emo")
```

Fontos tudnunk, hogy a csomagok telepítésére egy számítógépen egy adott R verzión belül csak egyszer van szükség. A telepítő parancsainkat azonban érdemes megőrizni, ugyanis egy új R verzióban könnyebben tudjuk így telepíteni a korábban használt csomagjainkat. Nagyon fontos, hogy a telepítő parancsok futtatása után, tegyük azokat megjegyzésbe, vagyis írjunk eléjük kettős kereszt (#) karaktert (részletesebb információkat a megjegyzésekről a 5.1.3. fejezetben olvashatunk). Ezzel tudjuk megvédeni ezeket a telepítő parancsokat az újbóli, véletlen, felesleges végrehajtástól. Ennek megfelelően a telepítő parancsainkat ilyen formában kell őriznünk:

```
# install.packages("psych")        # psych csomag telepítése
# if (!requireNamespace("BiocManager", quietly = TRUE))
#     install.packages("BiocManager")
# BiocManager::install("DESeq2")
# devtools::install_github("hadley/emo")
```

Vegyük figyelembe, hogy egy csomag telepítése során más, egyéb csomagok telepítése automatikusan is megtörténhet, tehát egy helyett valójában több csomag is felkerülhet a gépünkre. Az is előfordulhat, hogy egy csomag telepítése csak akkor lesz sikeres, ha más csomagok frissítését engedélyezzük az adott csomag telepítése során. Végül előfordulhat olyan eset is, amikor egy csomag telepítése valamilyen oknál fogva meghiúsul. Erről minden esetben hibaüzenet tájékoztat minket, és ez szinte minden esetben jó kiindulásul szolgál a hibát okozó körülmény elhárításában. A legtöbbször egy másik csomag hiánya okozza a sikertelen telepítést, ezért olvassuk ki a hibaüzenetből a hiányolt csomag nevét, és először ennek a telepítését végezzük el. Nagyon ritka esetben az is előfordulhat, hogy egy csomag telepítését az *RStudio* helyett az *Alap R*-ben kell elvégeznünk.

Összefoglalás

Az R kényelmes használatához először telepítsük az operációs rendszerünknek megfelelő *Alap R*, majd az *RStudio* legújabb verzióját. Az R képességeit csomagok segítségével bővíthetjük, melyek három különböző tárhelyről származhatnak. A legtöbb csomagot a CRAN-ről telepíthetjük az `install.packages()` parancs használatával. A Bioconductor-ról vagy a GitHub-ról származó csomagok telepítéséhez más parancsokat kell használnunk.

🎯 Feladatok

1. Melyik az R legfrissebb változata, és milyen újdonságokat tartalmaz az előző változathoz képest?
2. Melyik az *RStudio* legfrissebb változata, és milyen újdonságokat tartalmaz az előző változathoz képest?
3. Hogyan deríthető ki, hogy egy csomagban (például a `{MASS}`) csomagban, hány adatobjektum, és hány függvény található?

3.2. A *Tidyverse R* telepítése 😉

ℹ Miről lesz szó? Ebben a fejezetben

- megismerjük a *Tidyverse R* telepítését.

A *Tidyverse R* az R meglévő funkcióinak új szemléletű használatát jelenti. A modern R jelenleg egyet jelent a *Tidyverse R*-rel, az ebben a szemléleteben készült parancsaink gyorsak, jól olvashatók és könnyen módosíthatók. A *Tidyverse R* funkciói összesen több csomagba (például `{ggplot2}` és `{dplyr}`) vannak szétosztva, mindegyik csomag egy-egy témakört fed le. A fenti csomagok telepítése egyetlen gyűjtőcsomag a `{tidyverse}` nevű csomag telepítésével is elvégezhető:

```
install.packages("tidyverse")  # a Tidyverse R telepítése
```

A *Tidyverse R* telepítését követően a csomagokban lévő függvények használatához a *Tidyverse R* betöltésére is szükség van. Hívjuk meg a `library()` függvényt, amely ebben az esetben igen részletes tájékoztatást ad az újonnan elérhető csomagokról.

```
library(tidyverse)
#> ── Attaching core tidyverse packages ──────────────── tidyverse 2.0.0 ──
#> ✔ dplyr     1.1.4     ✔ readr     2.1.5
#> ✔ forcats   1.0.0     ✔ stringr   1.5.1
#> ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
#> ✔ lubridate 1.9.4     ✔ tidyr     1.3.1
#> ✔ purrr     1.0.4
#> ── Conflicts ────────────────────────────────── tidyverse_conflicts() ──
#> ✖ dplyr::filter() masks stats::filter()
```

```
#> ■ dplyr::lag()    masks stats::lag()
#> ■ Use the conflicted package to force all conflicts to become errors
```

A *Tidyverse R* csomagjait jelenleg is intenzíven fejlesztik, így gyakran jelenik meg újabb és újabb verzió. Érdemes ellenőrizni, hogy a *Tidyverse R* csomagjai közül a legfrissebbeket használjuk-e. Ehhez a `{tidyverse}` csomag `tidyverse_update()` függvényét használjuk.

```
tidyverse::tidyverse_update()  # a Tidyverse R frissítése
#> The following packages are out of date:

#> • broom    (1.0.7 -> 1.0.8)
#> • ggplot2  (3.5.1 -> 3.5.2)
#> • jsonlite (1.8.9 -> 2.0.0)
#> • pillar   (1.10.1 -> 1.10.2)
#> • ragg     (1.3.3 -> 1.4.0)
#> • readxl   (1.4.3 -> 1.4.5)
#> • rlang    (1.1.5 -> 1.1.6)
#> • xml2     (1.3.6 -> 1.3.8)
#> Start a clean R session then run:
#> install.packages(c("broom", "ggplot2", "jsonlite", "pillar", "ragg",
#> "readxl", "rlang", "xml2"))
```

Például a fenti esetben több csomag frissítését is javasolja a `tidyverse_update()` függvény, és segítséget is ad a telepítőparancs listázásával. A javaslatban szereplő munkamenet törlés (`Start a clean R session`) az *RStudio*-ban az `.rs.restartR()` parancs vagy a `Ctrl-Shift-F10` billentyűkombináció kiadásával valósítható meg.

Összefoglalás

A *Tidyverse R* használatához elegendő telepítenünk a `{tidyverse}` csomagot, amely a többi (legalább 8) csomag telepítését automatikusan elvégzi. A telepítést a `install.packages("tidyverse")` paranccsal végezzük. Időnként ellenőrizzük a `tidyverse::tidyverse_update()` segítségével, hogy a legfrissebb változatát használjuk-e a *Tidyverse R*-t alkotó csomagoknak.

Feladatok

1. Keressünk rá a *Tidyverse R* csomagjaira, és próbáljuk kideríteni az egyes csomagok fő célját, alkalmazási területeit!
2. Derítsük ki, hogy az *R Core Team* vagy Hadley Wickham több R csomag szerzője!

3.3. Az R frissítése 🤯

> ℹ Miről lesz szó? Ebben a fejezetben
>
> - bemutatjuk az *Alap R*,
> - az *RStudio*,
> - és a csomagok frissítését.

A R ideális használata során az *RStudio*-ban dolgozunk, és így érjük el az *Alap R* és az egyes csomagok szolgáltatásait. A mai napig mindhárom komponenst intenzíven fejlesztik, újabb és újabb funkciókat építenek be, és az esetleges hibákat rendre javítják a frissebb változatokban. Az *Alap R* évente kb. négyszer frissül, az *RStudio* háromszor, és érdemes időnként azt is ellenőrizni, hogy a gyakran használt csomagjainkból nincs-e frissebb példány.

3.3.1. Az *Alap R* frissítése

A telepített *Alap R* verzióját az `R.version.string` végrehajtásával ellenőrizhetjük. Amennyiben az R hivatalos oldalán találunk frissebb példányt, akkor legalább két módszer segítségével frissíthetjük az *Alap R*-t. Megjegyezzük, hogy az *Alap R* sikeres frissítése után az *RStudio* automatikusan az új példányt fogja használni.

1. módszer (csak Windows alatt) Windows operációs rendszer alatt rendelkezésre áll az `{installr}` csomag, amelynek pontosan az a feladata, hogy kényelmesen telepíthessük számítógépünkre az *Alap R* legfrissebb verzióját. Az `{installr}` a régebbi verzióban lévő csomagokat az új változatba is átmozgatja, és ott azok frissítését is elvégzi. A következő parancsok futtatására van szükség.

```
# install.packages("installr") # az installr csomag telepítése
library(installr)              # az installr csomag betöltése
updateR()                      # az Alap R és a csomagok frissítése
```

2. módszer (minden operációs rendszeren) Az *Alap R* frissítésének másik módja, hogy telepítünk egy új példányt a régi R mellé. Azaz a korábban látott módon letöltjük és telepítjük az *Alap R* legújabb változatát, pontosan úgy, mintha még nem lenne a gépünkön működő R. Ez az új verzió azonban félkarú óriás mindaddig, amíg a régi R verzióban használt összes csomagot nem telepítjük újra az új verzióban is. Ezt magunk is megtehetjük, ha korábban összegyűjtöttük a csomagtelepítő parancsainkat, legyen szó akár akár a CRAN, a Bioconductor vagy a GitHub oldaláról származó csomagokról. Ha ezek a parancsok nem állnak

rendelkezésre, akkor az *Alap R* frissítésének általános útját három lépésben foglalhatjuk össze.

1. Indítsuk el az *RStudio*-t még az új R verzió telepítése előtt, és futtassuk le a következő sorokat. A futtatás eredménye egy bináris állomány (`csomagok.rds`), amely a régi R összes telepített csomagjának nevét és más információkat tartalmaz. Lépjünk ki az *RStudio*-ból.

```
# futtatás telepítés előtt
telepitett.csomagok <- installed.packages(priority="NA")
saveRDS(object = telepitett.csomagok, file = "csomagok.rds")
```

2. Telepítsük az *Alap R* új verzióját.
3. Indítsuk el az *RStudio*-t és futtassuk le a lenti sorokat. A folyamat több percig is eltarthat. Az *RStudio* már az új R verziót használja, így a csomagok az új R tudását egészítik ki.

```
# futtatás telepítés után
telepitett.csomagok <- readRDS(file = "csomagok.rds")
install.packages(pkgs=telepitett.csomagok[,1])
```

Megjegyezzük, hogy a fenti módszer segítségével csak a CRAN csomagjait tudjuk telepíteni, a Bioconductor és a GitHub oldalakról származó csomagok telepítését magunknak kell megismételni. Tehát a nem CRAN-ről származó csomagok telepítő parancsait mindenképp érdemes megőrizni.

3.3.2. Az *RStudio* frissítése

A telepített *RStudio* példányunk verziószámát a `Help / About RStudio` menüpont segítségével, vagy az `rstudioapi::versionInfo()` parancs futtatásával ellenőrizhetjük. Frissebb verzió létezéséről a `Help / Check for Updates` menüpont ad tájékoztatást. Amennyiben találunk újabb verziót az *RStudio* hivatalos honlapján, töltsük le az operációs rendszerünknek megfelelő változatot és indítsuk el a telepítőt. Szerencsére a régi *RStudio* beállításait örökli az új példány, és a továbbiakban csak az új példány lesz elérhető.

3.3.3. Csomagok frissítése

A korábban telepített csomagokat az *RStudio* `Tools/Check for Package Updates` menüpontjával frissíthetjük. A frissíthető csomagok megjelennek egy dialógus dobozban, jelöljük ki az összes csomagot és indítsuk el a telepítési folyamatot. A következő R parancs végrehajtásával is frissíthetjük a csomagjainkat.

```
update.packages(ask = FALSE)
```

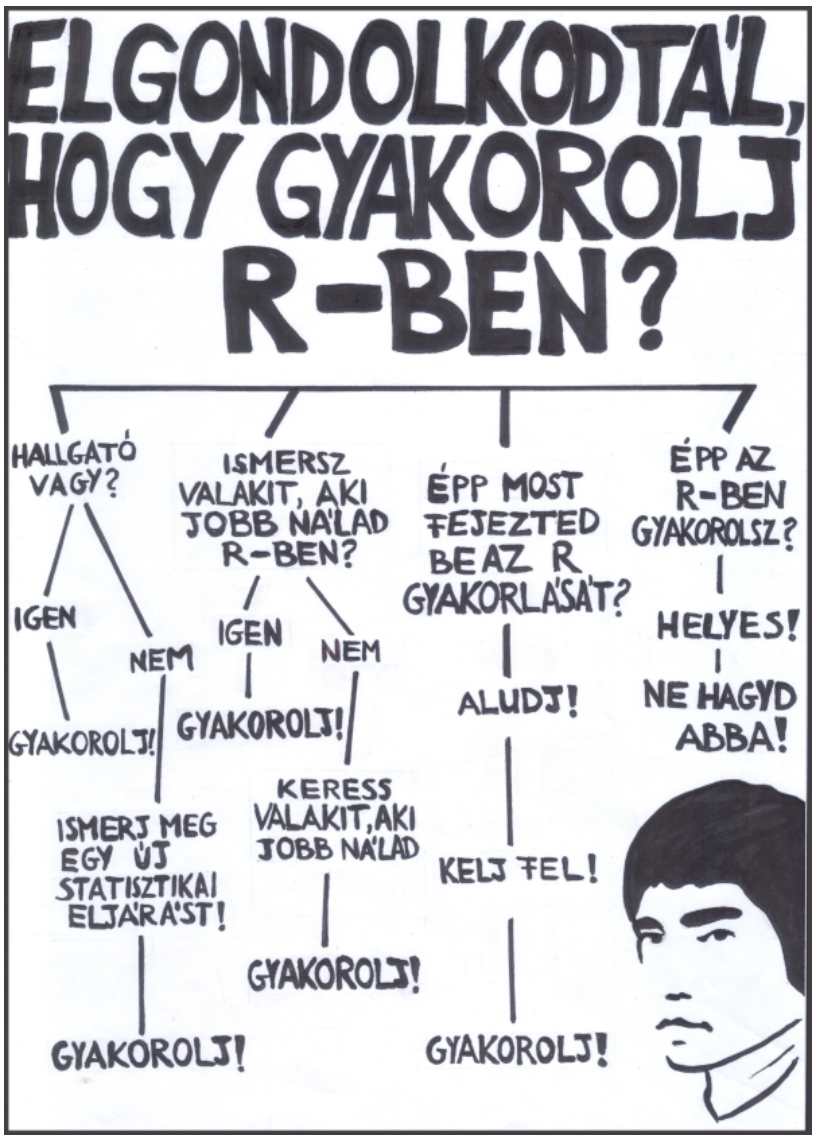
📚 Összefoglalás

Az *Alap R*, az *RStudio* és az egyes csomagok időről-időre megújulnak, érdemes évente legalább egy-két alkalommal elvégezni ezek frissítését. Az *Alap R* frissítése lényegében egy új verzió telepítését jelenti, a régi R továbbra is elérhető marad. Az *RStudio* frissítése után csak az új verziót használhatjuk. Az *Alap R* és az *RStudio* friss verziója a hivatalos honlapokról szerezhető be. A csomagok frissítéséhez használjuk az `update.packages(ask=FALSE)` parancsot.

🎯 Feladatok

1. Az *RStudio* `Tools/Check for Package Updates` menüpontjával tájékozódjunk a telepített csomagjaink állapotáról. Végezzük el a szükséges frissítéseket! Mit tegyünk, ha nem sikerül valamelyik csomag telepítése?
2. Ismerjük meg a telepített csomagjaink számát és forrását (CRAN vagy Bioconductor vagy GitHub)!

4. Munka az R-ben



4.1. Az *RStudio* használata 🙂

> **Miről lesz szó?** Ebben a fejezetben
>
> - megismerjük az *RStudio* jellemzőit és felépítését,
> - a konzolos és parancsállományos használat különbségeit,
> - a parancsállományok és a *Quarto* állományok lehetőségeit,
> - a projekt fogalmát és használatát,
> - és az *RStudio* billentyűparancsait.

Miután minden szükséges szoftverkomponenst feltelepítettünk, hogyan tudjuk működésre bírni az R-t?

Tegyük fel, hogy van egy nagyon egyszerű adatfeldolgozási problémánk, szeretnénk megtudni a *Csillagok háborúja* c. film karaktereinek átlagos testmagasságát a filmben szereplő egyes fajokra jellemzően. Ha rátalálunk egy alkalmas adatbázisra, amely tartalmazza a szereplők testmagasságait és azt, hogy melyik fajhoz tartoznak, akkor még két konkrét adatelemzési lépés vár ránk:

1. az adatbázis megnyitása,
2. az átlagos testmagasságok meghatározása fajonként.

Korábban láttuk, hogy az R parancssoros, tehát a fenti két lépést R parancsok formájában kell megfogalmaznunk. Azonban több kérdés is felmerül ezen a ponton:

1. hová írjuk a parancsainkat,
2. hogyan hajtsuk őket végre, és végül,
3. hol jelenik meg az eredmény.

Ebben a fejezetben a fenti három kérdésekre fókuszálunk, és azt a kérdést, hogy mely konkrét parancsokkal érhetjük el a célunkat a könyv további fejezeteire halasztjuk.

Máris megválaszoljuk a fenti kérdéseket. Korábban láttuk, hogy az *Alap R* telepítésével elérhetővé válik a *konzol*, ahová parancsainkat begépelve, majd `Enter`-t ütve utasításokat tudunk végrehajtani. Az *RStudio* telepítésével is kapunk egy konzolt, amelynek működése megegyezik az *Alap R* konzoljával: ide is gépelhetünk parancsokat, és `Enter`-rel végrehajthatjuk őket. A parancsok eredmény is itt, a konzolban fog megjelenni.

A kiinduló adatelemzési feladatunk megoldásához tehát vagy az *Alap R* vagy az *RStudio* konzoljába gépeljük be a következő parancsokat, sorról-sorra, és minden egyes sor végén üssünk `Enter`-t (a konzol használatához a 4.1.4. fejezetekben találunk segítséget). A `#`-el kezdődő részeket nem szükséges begépelnünk, azok nem az R-nek szólnak, hanem a megjegyzés szerepét töltik be.

```
install.packages("dplyr")      # a dplyr csomag telepítése
install.packages("psych")      # a psych csomag telepítése
data(starwars, package="dplyr")  # adatbázis beolvasása csomagból
# testmagasság átlagok fajonként
psych::describeBy(starwars$height, starwars$species, fast=T, mat=T)
```

A konzol azonban nem a legkényelmesebb módja az R parancsok végrehajtásának. Ezzel minden bizonnyal egyetértenek azok, akik a fenti sorok begépelését és végrehajtását valóban elvégezték a konzolban. A konzolba gépelés helyett érdemes egy szöveges állományban összegyűjteni az adatfeldolgozáshoz kapcsolódó R parancsainkat, ugyanis ezeket később kényelmesen elküldhetjük a konzolba végrehajtásra, pont úgy, mintha közvetlenül a konzolba gépeltük volna be őket. Ezeknek a szöveges állományoknak két fajtáját ismerjük meg ebben a könyvben: a *parancsállományokat* és a *Quarto* állományokat (a 11. fejezetben a *Quarto* állományokról többet olvashatunk).

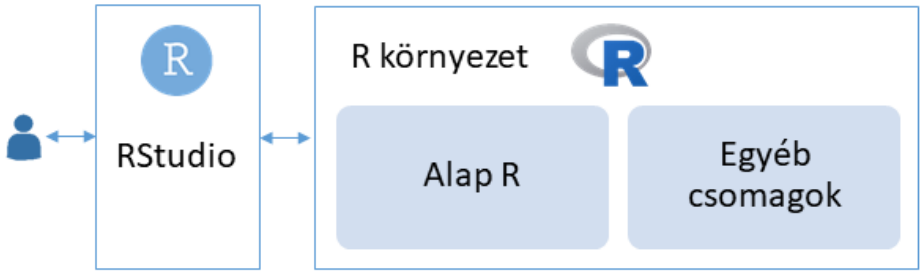
A parancsállományok és a *Quarto* állományok létrehozásához is a legtöbb segítséget az *RStudio* nyújtja: a parancsok begépelését drámaian leegyszerűsíti, és egyben számos más kényelmi funkciót is ajánl. Hová írjuk tehát az R parancsainkat? A legjobb válasz erre a kérdésre: az *RStudio* parancsállományaiba vagy *Quarto* állományaiba. Mielőtt valóban elvégeznénk ezen állományok létrehozását, ismerkedjünk meg az *RStudio* lehetőségeivel!

4.1.1. Az *RStudio* jellemzői

Fontos tisztázni, az *RStudio* használatához feltétlenül szükség van a telepített *Alap R*-re, nélküle nem tudunk R parancsokat futtatni. Jó gyakorlat, ha az *RStudio* telepítése előtt telepítjük fel az *Alap R*-t, de a fordított sorrend sem okoz problémát. Sőt, ha az *Alap R* egy új verzióját telepítjük fel, akkor a korábban telepített *RStudio* már az új verziójú R futtató környezetét fogja használni. Az *RStudio* tudása tehát a végrehajtható R parancsok tekintetében megegyezik az *Alap R* tudásával, hiszen minden utasítás, amelynek a végrehajtását az *RStudio*-ban kezdeményezzük, végső soron az *Alap R*-rel telepített interpreterhez kerül, és a végrehajtásáért ő felel (4.1. ábra).

Az *RStudio* elsősorban a parancsok írását könnyíti meg, segítségével a parancsok létrehozásához kapunk rendkívüli segítséget. Megjegyezzük, hogy az *RStudio* a *posit* nevű üzleti vállalkozáshoz tartozik, amely többféle terméket fejleszt. Ezek egyike az *RStudio*-nak nevezett integrált fejlesztőkörnyezet, kimondottan az R programozási nyelv számára. Foglaljuk össze, hogy melyek az *RStudio* erősségei:

- **Parancsok írásának könnyítése.** Az R parancsok begépelését számos eszköz segíti, például a kódkiegészítés, a szintaxisnak megfelelő kódszínezés és a tippek megjelenítése.



4.1. ábra: Az R kényelmes használata *(saját szerkesztés)*

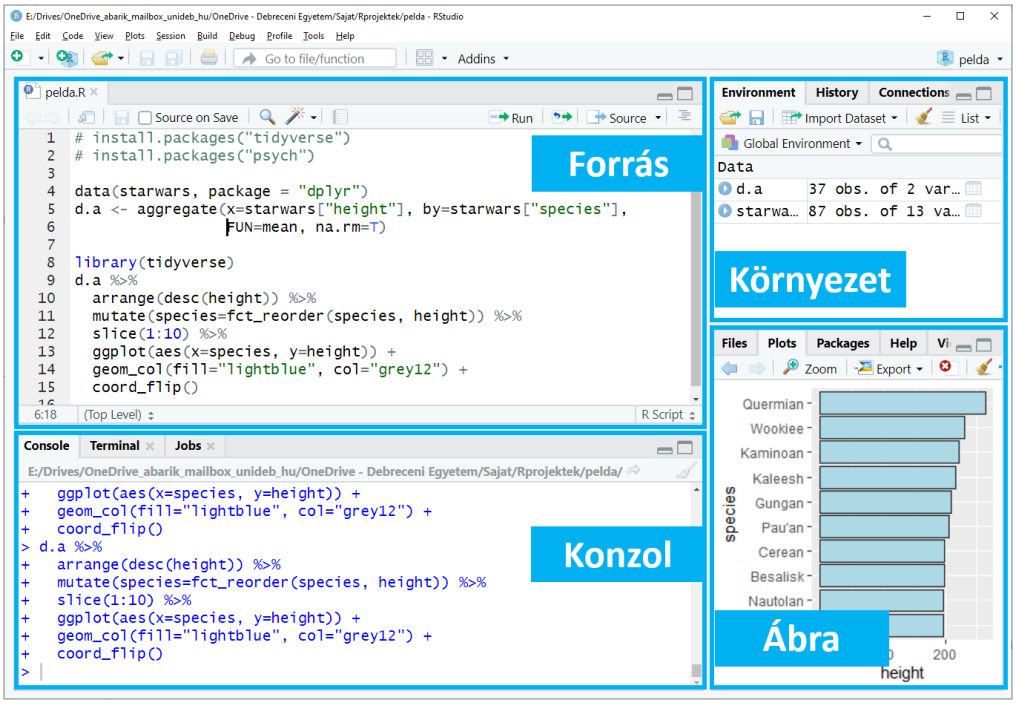
- **Integrált környezetben, egy felületen látjuk a munka során szükséges összes komponenst.** Az adatelemzési munka nem merül ki a parancsok begépelésében és végrehajtásában. Az R parancsokat jelentő forráskódon kívül kezelnünk kell az outputot, ami lehet szöveges és ábra jellegű is, valamint el kell igazodnunk a memóriában tárolt adatok között is. Sokszor a súgót is meg kell jelenítenünk, és információval kell rendelkeznünk a telepített csomagokról is. Az *RStudio* nagy előnye, hogy mindezt egyetlen integrált felületen láthatjuk és ezen keresztül vezérelhetjük.
- **Projektek használata.** Az *RStudio* támogatja a projektek használatát is, amellyel az adott adatfeldolgozási folyamat összetevőit – az adatállományokat, parancsállományokat, *Quarto* állományokat, képállományokat és dokumentációkat –, egyetlen könyvtárba foghatjuk össze, és a forráskódból relatívan hivatkozhatunk ezekre az állományokra.
- **Publikálás támogatása.** A *Quarto* segítségével kényelmesen és reprodukálható módon hozhatunk létre például PDF, HTML és Word formanyelvű dokumentumokat, vagy PDF, HTML és PowerPoint bemutatókat.
- **További lehetőségek.** Az *RStudio* támogatja a Shiny Webes alkalmazások fejlesztését, de saját csomagok létrehozásához is kapunk segítséget. Az *RStudio* támogatja a Git verziókezelő használatát is.

Az *RStudio* fenti lehetőségeinek bemutatása külön könyvet igényelne, de a mindennapi munkához szükséges ismereteket most bemutatjuk.

4.1.2. Az *RStudio* felépítése

Az *RStudio* indítása után egy több panelből álló alkalmazást látunk. Első indításnál három részre van osztva az alkalmazás, vagyis három panel látható, de a tipikus használat során négy panelünk van. Válasszuk ki először a `File / New file / R Script` menüpontot, amely egy

új parancsállomány létrehozását kezdeményezi. E lépés után már biztosan a négy-paneles, 4.2. ábrán látható elrendezést kapjuk. Az ábrán megneveztük az egyes részeket, a két bal oldali panel a *Forrás* és a *Konzol*, a jobb oldaliak a *Környezet* és az *Ábra*. Figyeljük meg, hogy a panelek tetején fülek láthatók, így az egyes paneleken különböző lapokat tudunk kiválasztani, egy panel tehát több lapot is tartalmazhat. A panelek szélessége és magassága állítható, egyrészt az elválasztó sávokat az egér segítségével mozgathatjuk, másrészt a panelek méretező gombjain (az egyes panelek jobb felső sarkában) is kattinthatunk. A méretezés során eltűnhetnek panelek, de a sávok mozgatásával vagy a `View / Panes / Show All Panes` menüponttal láthatóvá tehetjük az összes panelt.



4.2. ábra: Az *RStudio* tipikus képernyőképe (Posit team, 2025)

A legtöbb időt a *Forrás* nevű bal felső panelben töltjük, mert alapértelmezetten itt jelennek meg a parancsállományok és a *Quarto* állományok lapjai. Az R parancsainkat tehát ide írjuk. Az *RStudio* első indításánál ez a panel üres, de a további indításoknál a korábban szerkesztett, de be nem zárt lapok automatikusan megnyílnak. Itt helyeztünk el korábban egy parancsállomány lapot a `File / New file / R Script` segítségével. Ez a lap egy egyszerű szövegszerkesztő. Győződjünk meg erről, próbáljuk ki, mert a jövőben ebben a szövegszerkesztőben töltjük a legtöbb időt! A fejezet végi kitűzött feladatok között rákérdezünk a szövegszerkesztési ismeretekre. Oldjuk meg most azt a feladatot, majd térjünk vissza ide!

A bal alsó panel a *Konzol* nevet viseli, vagyis ez az *RStudio* konzolja, melynek használata és célja megegyezik az *Alap R* konzoljával. Vagyis begépelhetünk parancsokat, és az `Enter`-rel

végrehajtjuk őket. Azonban a konzol mindössze egysoros szövegszerkesztési lehetőséget kínál, lényegében egyszerre egy parancs begépelésére és végrehajtására van lehetőségünk. Ez lényegesen eltér a *Forrás* panel parancsállomány vagy *Quarto* lapján lévő teljes értékű szövegszerkesztőtől, ahol több sor begépelésére és végrehajtására van lehetőségünk. A konzol azonban mégis központi szerepet kap, mert alapesetben az R csak a konzolba kerülő parancsokat tudja végrehajtani. A parancsállományok és *Quarto* állományok R parancsait is valahogyan át kell ide irányítani, úgy mintha ide gépeltük volna be őket. De a konzol nem csak a parancsainkat, azaz az inputot, hanem azok eredményét, az outputot is tartalmazza.

A két jobb oldali panel többfunkciós. A jobb felső, *Környezet* panelben jelennek meg a munka során létrehozott objektumok nevei (*Environment* lap), valamint a parancsok története (*History* lap). Az *Environment* lapon megjelenő adatbázis nevén kattintva a *Forrás* panelben egy külön lapon megjelenik az adatbázis tartalma, így kapjuk az ún. adatbázis lapot. A jobb alsó *Ábra* panel tartalmazza a súgót (*Help* lap), a munka során rajzolt ábráinkat (*Plot* lap), a csomagjaink listáját (*Packages* lap) és a munkakönyvtárunk állományait, könyvtárait (*Files* lap). A két jobb oldali panel elnevezés önkényes volt, hiszen az *Environment* és a *Plot* csak egy-egy lap neve ezeken a többfunkciós paneleken.

4.1.3. Az *RStudio* beállításai

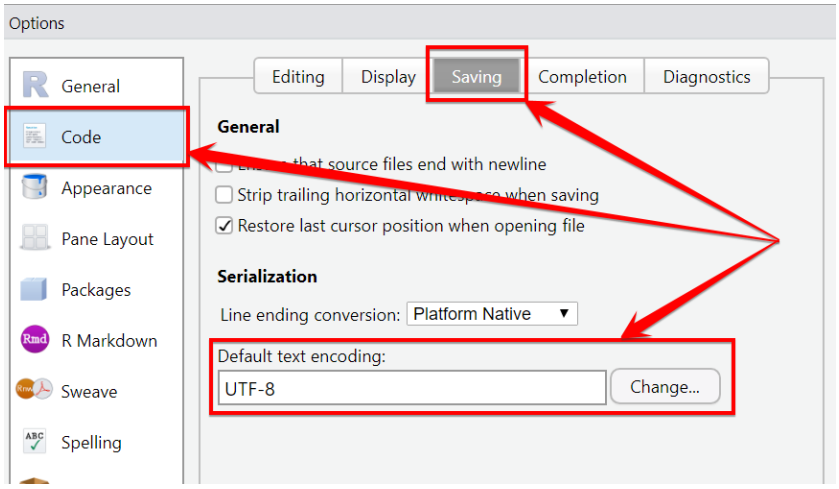
Mielőtt elkezdjük a munkát az *RStudio*-ban feltétlenül módosítsunk néhány alapbeállítást. Az *RStudio* működését az `Tools / Global Options` menüpont alatt változtathatjuk meg.

UTF-8 kódolás beállítása. A fenti menüpont kiválasztása után a bal oldali listából a `Code`, majd a fenti opciók közül a `Saving` opciót válasszuk. A 4.3. ábrán is látható módon, érjük el, hogy a `Default text encoding` alatt az `UTF-8` legyen kiválasztva. Fontos, hogy minden szöveges állományunk UTF-8 kódolású legyen.

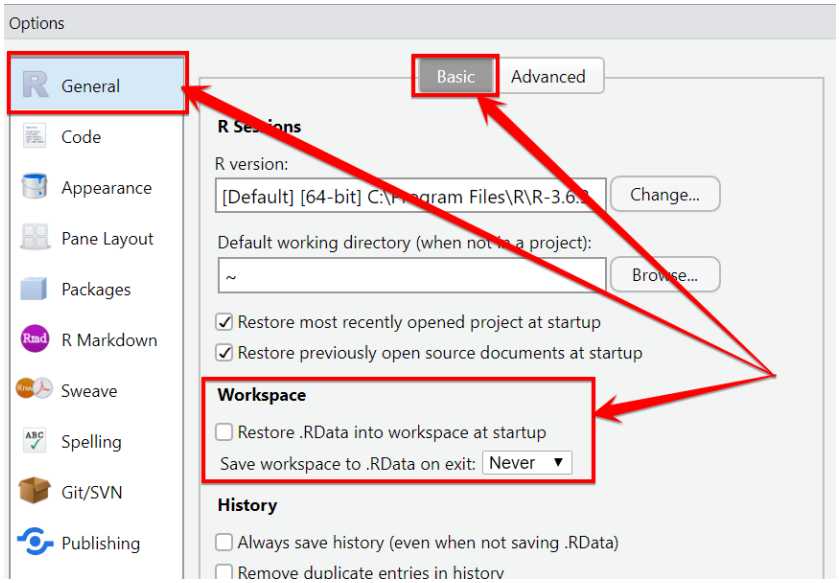
A munkaterület automatikus mentésének tiltása. A bal oldalon a `General` menüpont kiválasztása után a `Basic` opció alatt vegyük ki a pipát a `Restore .RData into workspace at startup` elől, valamint a `Save workspace to .RData on exit` választót állítsuk `Never`-re (4.4. ábra). Az *RStudio* projekt szemléletű használata mellett erre a mentési funkcióra nincs szükség.

Az output megjelenítésének tiltása a Quarto lapon. A bal oldalon az `R Markdown` menüpont kiválasztása után vegyük ki a pipát a `Show output inline for all R Markdown documents` elől (4.5. ábra). Ez a beállítás gördülékenyebb szerkesztést biztosít a *Quarto* lapokon.

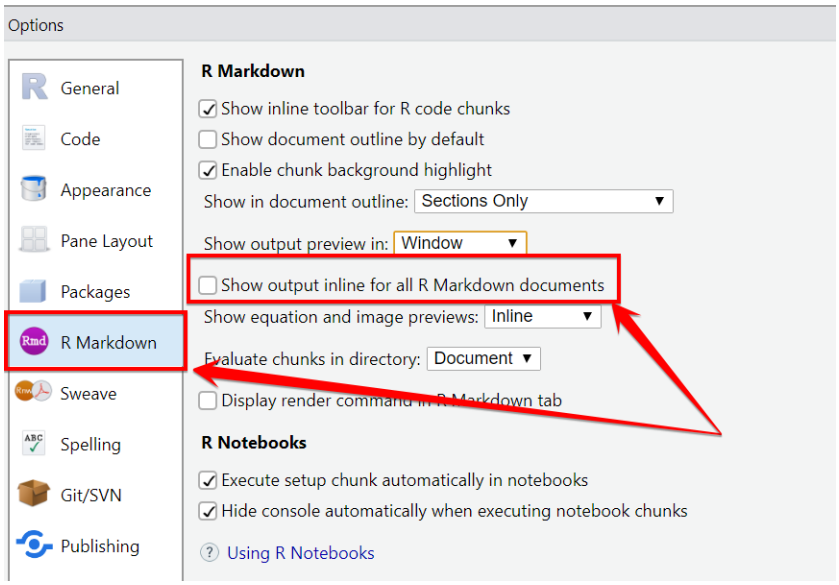
Opcionális lehetőségként a panelek tartalmán is változtathatunk a `Tools / Global Options / Pane Layout` menüpontban. Az *RStudio* színösszeállításán az `Appearance` menüpont `Editor theme` beállításával változtathatunk. Javasolt a `Tomorrow Night Bright` vagy más, sötétebb háttérszínnel rendelkező téma használata.



4.3. ábra: Az UTF-8 beállítása az *RStudio*-ban (Posit team, 2025)



4.4. ábra: A munkaterület automatikus mentésének tiltása az *RStudio*-ban (Posit team, 2025)



4.5. ábra: Az output megjelenítésének tiltása a *Quarto* lapon (Posit team, 2025)

4.1.4. Az *RStudio* konzol

Az *RStudio* konzolja a *Konzol* panel egyik lapján található (4.2. ábra). A konzol az *RStudio* kulcsfontosságú része, korábban láttuk, hogy minden R parancsot a végrehajtás előtt ide kell irányítani. Végrehajtása után a szöveges eredmények is itt jelennek meg, és a hibaüzeneteket is itt olvashatjuk. Láthatjuk tehát, hogy a konzol figyelmünk középpontjában áll a munka során.

Közvetlenül azonban nagyon ritkán gépelünk parancsot a konzolba, erre a *Forrás* panel parancsállomány vagy *Quarto* lapját fogjuk használni. Ebben a részben mégis a konzolt mutatjuk be, ugyanis meghatározó szerepe miatt értenünk kell működését.

A konzol működése nagyon egyszerű:

1. egysoros parancsokat gépelünk be a `>` prompt után,
2. `Enter`-t nyomunk,
3. az R interpreter értelmezi és végrehajtja a begépelt parancsot, és
4. megjelenik az eredmény vagy egy hibaüzenet.

Ezt követően egy újabb sor begépelésére van lehetőségünk, `Enter` után annak az értelmezése következhet, majd az eredmény megjelenítése jön, és így tovább.

Próbáljuk ki mi is a konzolt! Bátran gépeljünk be parancsokat. Például a `citation()` parancs outputja fontos lehet az R-el végzett munkáink publikálásánál, hiszen megmutatja hogyan hivatkozhatunk az R statisztikai programra, vagy valamelyik csomagjára.

```
> citation()
> citation(package = "ggplot2")
```

Fontos információ az *Alap R* és az *RStudio* pontos verziószáma, ezt a információt az `R.Version()` és a `RStudio.Version()` függvény szolgáltatja. Gépelésnél vigyázzunk a kis- és nagybetűk helyes bevitelére, mert az R megkülönbözteti ezeket.

```
> R.Version()
> RStudio.Version()
```

A konzol lehetőségeinek szisztematikus megismerését folytassuk egy egyszerű paranccsal:

```
> 1+2
[1] 3
```

A konzolban most is megjelent az eredmény, ahogy ezt az összes eddigi parancsunk esetében láthattuk. Azonban nem minden parancs után jelenik meg output a konzolban. Például a következő parancsnak nincs eredménysora a konzolban, de ez messze nem jelenti azt, hogy nem történt semmi (történt: létrehoztunk egy objektumot).

```
> x <- 3
```

Sőt, az is előfordulhat, hogy az R nem talált valamit rendben a parancsban. Ekkor természetesen nem hajtja/hajthatja végre a begépelt sort, helyette hibát jelez.

```
> Ez nem lesz jó.
```

A válasz a fenti "parancsra" az `Error: unexpected symbol in "Ez nem"` hibaüzenet lesz. Alapvető szabály, ha a válaszban megjelenik az `Error` szócska, akkor a parancsunkat valamilyen ok miatt nem tudta végrehajtani az R értelmező, és az `Error` utáni részből tájékozódhatunk a hiba okáról. Minden más esetben sikeres volt a végrehajtás.

Hosszabb, bonyolult parancsok gépelésénél gyakran előfordul, hogy valamiért nem sikerül "teljessé" tenni a begépelt parancsot, valami még hiányzik belőle (például egy záró kerek zárójel). Ezt az R értelmező észreveszi és az `Enter` megnyomása után egy `+` folytatás prompt megjelenítésével jelzi ezt számunkra. A `+` prompt után van lehetőségünk a hiányzó részek pótlására, majd ha készen vagyunk az `Enter` billentyűvel az összes eddig még végre nem hajtott sort elküldhetjük az értelmezőnek.

Gépeljük be a következő parancsot, három egymás utáni sorba, `Enter`-ekkel elválasztva.

```
> paste("Ez már",
+        "jó"
+       )
[1] "Ez már jó"
```

A `paste("Ez már",` kerüljön az első sorba, majd nyomjunk `Enter`-t. Az R nem hajtja végre a sort, de erre a nyilvánvalóan hibás, befejezetlen parancsra hibaüzenetet sem jelenít meg. Helyette felajánlja a parancs folytatását, befejezését egy új sorban, amely már a `+` prompttal kezdődik. A második sorba gépeljük be az `"jó"` karaktersorozatot, nyomjuk meg az `Enter`-t. Sajnos még ez sem tette teljessé a parancsunkat, így további folytatásra van lehetőségünk a `+` után a harmadik sorban. Ide gépeljük be a hiányzó `)` részt, és üssünk `Enter`-t. A parancsunk teljessé vált, megkapjuk az eredményt a konzolban, pontosan úgy, mintha a három sort egyetlen sorba gépeltük volna.

Legyünk nagyon óvatosak a konzol folytatás prompt funkciójával. Ha például az R nem találja a parancs hiányzó részét, akkor a konzol ezen kényelmi funkciója oda vezethet, hogy folyamatosan a `+` promptot kapjuk az `Enter` megnyomása után. Ezt a helyzetet hivatott megoldani az `ESC` billentyű, mellyel megszakíthatjuk az értelmező parancsfeldolgozási kísérletét. Az `ESC` megnyomása után visszakapjuk a `>` prompttal kezdődő (üres) sort, vagyis tiszta lappal, új, lehetőség szerint teljes parancs gépelésébe kezdhetünk. **A parancssorba mindig teljes parancsot gépeljünk, amint megjelenik a `+` folytatás prompt, azonnal szakítsuk meg az `ESC` megnyomásával az értelmezési folyamatot.**

Az R konzolos használatát két funkció valóban kényelmesebbé teszi. Egyrészt a korábban végrehajtott parancsainkat visszahívhatjuk, lapozhatunk bennük előre, hátra. Erre a `FEL` / `LE` billentyűkkel van lehetőségünk. Ezt *history*-nak is nevezzük, vagyis a parancsok történetének. Természetesen, az így visszahívott parancsot tetszőleges módon átszerkeszthetjük: navigálhatunk a sorban előre hátra, beszúrhatunk/törölhetünk karaktereket vagy használhatjuk a vágóasztal billentyűparancsait. A visszahívott és módosított parancsot az `Enter` segítségével újra végrehajthatjuk, és ehhez még a sor végére sem kell a szövegkurzort pozicionálni, az a sorban tetszőleges helyen állhat, az R mégis a teljes sort fogja értelmezni.

A másik kényelmi lehetőség a `TAB` billentyű használata, amellyel az elkezdett, de még be nem fejezett sorokat egészíthetjük ki. Ha egy sort többféleképpen is kiegészíthet az R, akkor egy listát kapunk a lehetőségekről, amelyet továbbgépeléssel szűkíthetünk, ha pedig csak egyetlen szóba jöhető befejezése van a begépelt karaktereknek, akkor a `TAB` megnyomása után ezzel a résszel kiegészül az elkezdett sorunk. Így nemcsak egyszerűen gépelést, illetve időt takaríthatunk meg, hanem például tájékozódhatunk a korábban létrehozott objektumok nevéről vagy az elérhető függvények névéről és paramétereiről is.

Az objektum, a függvények és az egyéb ebben a fejezetben homályosan hagyott fogalmak definícióit a könyv későbbi részeiben részletesen tárgyaljuk.

4.1.5. Parancsállományok

Láthattuk, hogy a konzolba egyszerre csak egy parancsot gépelhetünk be, úgy is gondolhatunk a konzolra, mint egy egysoros szövegszerkesztőre. Begépelünk egy sort és végrehajtjuk az `Enter`-rel. A problémáink többsége viszont nem oldható meg egyetlen paranccsal, csak több tízzel vagy százzal, ezért ez az interaktív, *konzolos használat* nem alkalmas hosszabb elemzésre.

Parancsainkat begépelhetjük egy `.R` kiterjesztésű, egyszerű, formázás nélküli szöveges állományba is. Az ilyen szöveges állományt *parancsállomány*nak vagy *szkriptállomány*nak nevezzük. Ilyen szöveges állományok létrehozására tetszőleges szövegszerkesztő alkalmas, de természetesen mi az *RStudio* segítségével fogjuk ezeket elkészíteni, ugyanis itt kapjuk a legnagyobb segítséget a parancsok gépeléséhez, majd végrehajtásához. A *Forrás* panel tartalmazza a parancsállomány lapokat, létrehozásuk a korábban látott `File / New file / R Script` menüponttal történik. Parancsállományok mentésére és már létező megnyitására is van lehetőségünk a megfelelő menüpont kiválasztásával (`File / Save` és `File / Open File`).

A parancsállományok használata lényegesen leegyszerűsíti az adatelemzés folyamatát, hiszen a konzol egysoros szövegszerkesztője helyett egy szinte végtelen sok parancssor begépelésére alkalmas szövegszerkesztő áll rendelkezésünkre. Mint minden szövegszerkesztőben, a különböző billentyűparancsok és a vágóasztal itt is megkönnyíti szerkesztés folyamatát. Az `Enter` jelentése parancsállományos környezetben a szövegszerkesztőkben megszokott speciális "újsor" karakter beszúrása, ami lényegesen különbözik a konzolos használat parancs végrehajtási funkciójától. A parancsaink interaktív végrehajtásáért az *RStudio*-ban a `Code / Run selected line(s)` menüpont, vagy még gyakrabban a `Ctrl-Enter` billentyűkombináció felel. Ezekkel a módszerekkel tudjuk a parancsainkat a konzolba irányítani és végrehajtani. De nézzük meg ezt a gyakorlatban!

4.1.6. Munka az *RStudio*-ban

Kezdjük a munkát! Nyissunk egy új parancsállományt (`File / New file / R Script`) és gépeljünk be néhány sort. Figyeljük meg, hogy milyen sokat segít az *RStudio* a lenti sorok begépelésében. Az értékadás (`<-`) operátort az `Alt--` billentyűkombináció segítségével vigyük be.

```
1+23
getwd()          # munkakönyvtár kiírása
x <- mean(1:100)
plot(1:10)
```

```
?mean
cat("- Vége -\n")
```

A szövegkurzorral álljunk az első sorra, és hajtsuk végre `Ctrl-Enter` billentyűparancsot. Láthatjuk, hogy (1) a sor átkerül a konzolba, (2) az *RStudio* végrehajtja a sort és az eredményt a konzolban megjeleníti, és (3) a szövegkurzor lejjebb lép a következő végrehajtható sorra. Egy újabb `Ctrl-Enter` így már ezt a sort hatja végre, és így tovább. Ha a sorok végrehajtása közben hibaüzenetet kapunk (`Error`), ne essünk kétségbe, a hibaüzenet a munka része. Nézzük át figyelmesen a begépelt sorainkat, javítsuk őket, és futtassuk újra az összes sort, fentről lefelé a `Ctrl-Enter`-ek segítségével.

A parancsok végrehajtása során láthatjuk mennyire kényelmes, integrált környezetben találtuk magunkat. Az `x <- mean(1:100)` hatására az *Environment* lapon megjelent az `x` objektum neve és értéke. A *Plot* lapon láthatunk egy ábrát, amit a `plot(1:10)` rajzolt meg, és a `?mean` a *Help* lapon mutatja meg a `mean()` átlagszámoló függvény beépített súgóját.

Mentsük el parancsállományunkat a `File / Save` vagy a `Ctrl-S` segítségével. Korábban létrehozott parancsállományokat a `File / Open` menüponttal nyithatunk meg.

A soronkénti végrehajtás mellett nagyon gyakori a kijelölt szövegrészek végrehajtása, amit szintén a `Ctrl-Enter`-rel tudunk kezdeményezni. A kijelölt rész lehet több sor, a teljes parancsállomány, vagy valamelyik sor egy része. Ez utóbbi próbáljuk ki úgy, hogy a parancsállomány első sorában csak az `1+2` részt jelöljük ki, és ezt hajtsuk végre a `Ctrl-Enter` segítségével. Az eredmény a konzolban a 3 lesz. A teljes szkriptállomány végrehajtásához jelöljük ki `Ctrl-A` segítségével a parancsállomány összes sorát, és nyomjuk meg a `Ctrl-Enter`-t. A konzolban tudjuk ellenőrizni, hogy minden sort újra végrehajtottunk.

Térjünk vissza a kiinduló adatelemzési problémánk megoldásához. Láttuk, hogy az R parancsok összegyűjtésére és végrehajtására a `.R` kiterjesztésű parancsállományok kiváló megoldást nyújtanak. Emlékezzünk vissza a fejezet eleji példára, amelyben a Csillagok háborúja c. film karaktereinek átlagos testmagasságát kerestük. Nyissunk egy új parancsállományt (`File / New file / R Script`) és gépeljük be a megoldást jelentő sorokat.

```
# A Csillagok háborúja c. film karaktereinek átlagos testmagassága
# Abari Kálmán
# 2025. 04. 30.

# install.packages("dplyr")      # a dplyr csomag telepítése
# install.packages("psych")      # a psych csomag telepítése
data(starwars, package="dplyr")  # adatbázis beolvasása csomagból
# testmagasság átlagok fajonként
psych::describeBy(starwars$height, starwars$species, fast=T, mat=T)
```

Látható, hogy a feladat tényleges megoldását jelentő két R parancs mellett megjegyzéseket is becsempésztünk, hogy később is tudjuk, ki, mikor és miért készítette ezt a parancsállományt (a `#` utáni részeket a sor végéig az R figyelmen kívül hagyja; részletesebb információkat a megjegyzésekről a 5.1.3. fejezetben olvashatunk). Futtassuk a sorokat a `Ctrl-Enter` segítségével, fussuk át a kiszámolt átlagos testmagasságokat az output `mean` oszlopában, majd mentsük el a szkriptállományt `Ctrl-S`-sel `starwars.R` néven. Később, napok, hetek vagy hónapok múlva, újra megnyithatjuk `starwars.R` állományunkat (`File / Open`), és újra lefuttathatjuk mini-elemzésünket. Ezzel a fejezet eleji adatelemzési feladatunkat megoldottuk. Vajon lehet ezt ennél jobban csinálni? Igen!

4.1.7. *Quarto* állományok

Az R parancsainkat olyan `.qmd` kiterjesztésű, egyszerű, szöveges állományokban is összegyűjthetjük, amelyek többet nyújtanak, mint a parancsállományok, de szerkezetük kicsit kötöttebb. Az ilyen szöveges állományok a *Quarto* állományok. Miben nyújtanak többet: ahogyan a 11. fejezetben részletesen áttekintjük, a *Quarto* állományok az eredmények publikálásához, például HTML, PDF vagy Word formanyelvű állományok létrehozását teszik lehetővé.

Hozzunk létre az *RStudio*-ban a `File / New File / Quarto Document` menüponttal egy új *Quarto* állományt. A megjelenő dialógusdobozban töltsük ki a `Title` és `Author` mezőket, azaz adjunk címet és szerzőt a dokumentumhoz, majd kattintsunk az `OK` gombon. A `Forrás` panelen megjelenik egy új *Quarto* lap, amely egy alapértelmezett tartalommal jön létre, és nem üresen, mint a parancsállományok esetében. Említettük, hogy a *Quarto* állományok szerkezete kötöttebb, ez az alapértelmezett tartalom az eligazodásban segít minket. Érjük el, hogy az új *Quarto* állomány ezeket a sorokat tartalmazza (a szerző neve a sajátunk legyen):

````
---
title: "A Csillagok háborúja c. film karaktereinek átlagos testmagassága"
author: Abari Kálmán
date: 2025-04-30
format: html
---

# Bevezetés

Ebben a dokumentumban a `tidyverse` csomag `starwars` adatkészletét fogjuk elemezni.

# Adatok betöltése és előkészítése

```{r}
````
````

````
#| label: setup
library(tidyverse)

# Az adatok áttekintése
glimpse(starwars)
```


# Alapvető leíró statisztikák

Az adatkészlet **`r nrow(starwars)`** karakter adatait tartalmazza.

```{r}
#| label: summary-stats

# install.packages("dplyr")      # a dplyr csomag telepítése
# install.packages("psych")      # a psych csomag telepítése
data(starwars, package="dplyr")  # adatbázis beolvasása csomagból
# testmagasság átlagok fajonként
psych::describeBy(starwars$height, starwars$species, fast=T, mat=T)
```
````

Minden *Quarto* állomány egy fejléccel kezdődik, amelyet a --- karakterek határolnak. A természetes nyelvű szöveget szabadon a fejléc alatti részben bárhová írhatjuk, az R parancsokat azonban ún. R csonkokban kell elhelyeznünk, amelyeket speciális kezdő és záró sorok határolnak. A 11. fejezetben részletesebben olvashatunk ezekről. Most elégedjünk meg annyival, hogy egy *Quarto* állományban tetszőlegesen sok R csonkot elhelyezhetünk, és egy R csonk tetszőlegesen sok R parancsot tartalmazhat. Egy R csonkon belül a parancsok végrehajtása ugyanúgy `Ctrl-Enter`-rel történik, mint a parancsállományok esetében. Próbáljuk ki! A most begépelt *Quarto* állományunk egyes csonkjában lévő R parancsokat hajtsuk végre `Ctrl-Enter`-ek segítségével. A mini-elemzés eredménye ismét a konzolban látható.

Hogyan foglalhatnánk össze a parancsállományok és a *Quarto* állományok közötti különbséget? A 4.1. táblázatban láthatjuk, hogy mindkét állományban összesen három különböző tartalmat szoktunk rögzíteni:

1. fejléc információt arról, hogy mi az elemzés célja, ki és mikor készítette az állományt,
2. magyarázó, természetes nyelvű szöveget (pl. magyar vagy angol nyelven), és
3. az adatelemző R parancsokat.

Az R parancsokat szabadon írhatjuk a parancsállományokba, viszont a fejléc információt és a magyarázó szövegeket megjegyzésbe kell tenni. A *Quarto* állományokba a magyarázó,
````

természetes nyelvű szövegek írhatók szabadon, míg az R parancsokat csonkokba, a fejléc információt pedig kötött módon, az állomány elejére kell írnunk.

4.1. táblázat: A parancsállomány és a *Quarto* állomány összehasonlítása *(saját szerkesztés)*

Tartalom	Parancsállomány (`.R`)	*Quarto* (`.qmd`)
Fejléc szöveg		
cím, szerző, dátum	megjegyzésbe	fejlécbe
Magyarázó szöveg		
természetes nyelvű szöveg	megjegyzésbe	bárhová
Adatelemzés		
R parancs	bárhová	R csonkba

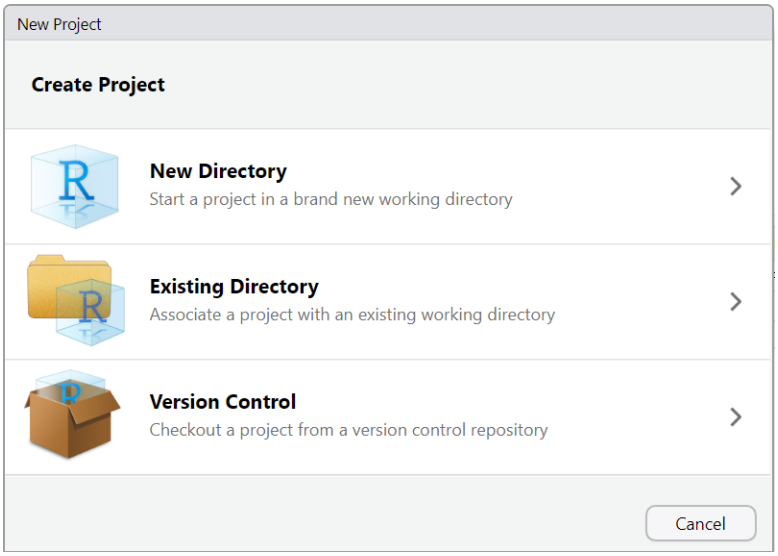
Valóban annyiban áll a különbség a két állománytípus között, hogy a máshová és máshogyan írjuk az R parancsokat és az egyéb magyarázó/fejléc szövegeinket? Nem. A 11. fejezetben részletesen bemutatjuk, hogy a *Quarto* állományok ereje abban van, hogy egy fordítási folyamat (*render*elés) során, olyan PDF, HTML vagy Word állományt tudunk előállítani, amely a magyarázó/fejléc szövegeken, és az R parancsokon kívül, az R parancsok outputját is tartalmazza, legyen az szöveges vagy ábra jellegű output.

4.1.8. Projektek használata

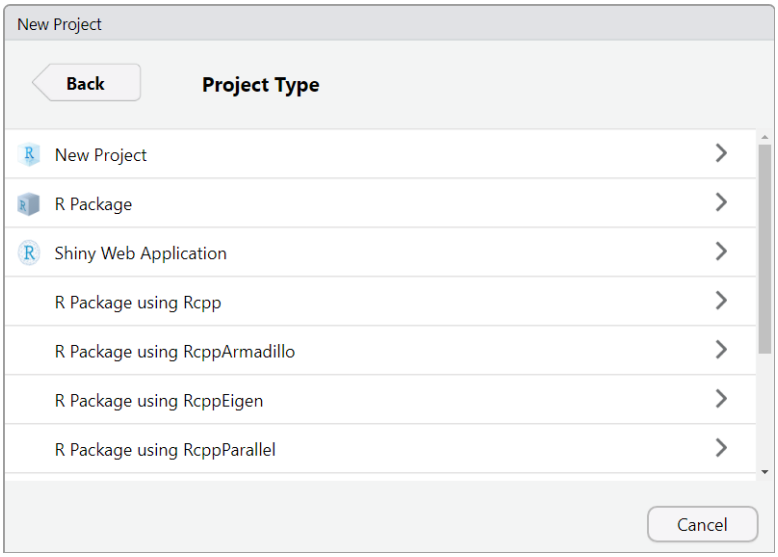
Mostanra nagyon közel kerültünk az általunk ajánlott adatelemzési munkaformához, ugyanis már tudunk az *RStudio*-n belül parancsállományokat és *Quarto* állományokat használni. Még egy összetevő azonban kulcsfontosságú a kényelmes munkához: az *RStudio*-ban minden esetben projektet kell használnunk.

Az *RStudio* lehetőséget ad arra, hogy minden egyes adatfeldolgozási feladatunkhoz egy projektet rendeljünk. Egy *RStudio* projekt minimálisan egy projekt könyvtárat és az ebben lévő lévő `.Rproj` kiterjesztésű projektállományt jelenti. Ezeket a következő módszerrel hozhatjuk létre. Először kattintsunk a `File / New Project` menüponton. Válasszuk ki a `New Directory` opciót (4.6. ábra), majd a `New Project` nyomógombon kattintsunk (4.7. ábra).

A `Directory name` szöveges mezőbe a projektünk nevét határozhatjuk meg, ami egyben az új projektünk könyvtárneve is lesz. Adjuk meg itt az `elso_projekt` nevet. A `Create project as subdirectory of` mezőben azt a szülő könyvtárat határozhatjuk meg, ahová a projekt könyvtárunkat el szeretnénk helyezni. Ezt szabadon megválaszthatjuk, lehet az adott felhasználó dokumentumok könyvtára is. A projekt létrehozását a `Create Project` nyomógombbal fejezhetjük be.

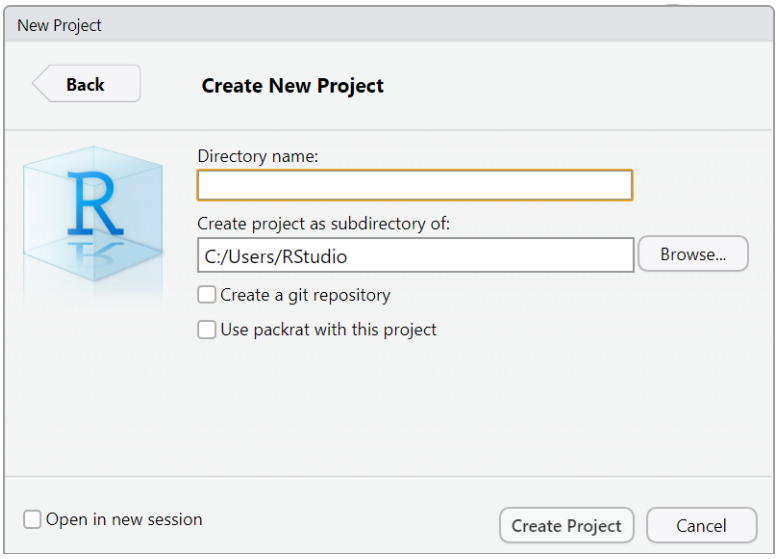


4.6. ábra: RStudio projekt létrehozása: 1. lépés (Posit team, 2025)



4.7. ábra: RStudio projekt létrehozása: 2. lépés (Posit team, 2025)

Két nagyon fontos dolog történt a fentiek hatására. Egyrészt a számítógépünkön létrejött az `elso_projekt` projektkönyvtár, és benne az `elso_projekt.Rproj` projektállomány, másrészt az *RStudio* ún. *projekt üzemmód*ba került, azaz az `elso_projekt` lesz az aktív projekt. Az *RStudio*-ban egyszerre egy projekt lehet aktív, de elképzelhető, hogy egyetlen projekt sem aktív. Az *RStudio* felületén a jobb felső sarokban tájékozódhatunk, ahol most az `elso_project` feliratot látjuk, de amennyiben nincs aktív projektünk, akkor a `Project: (none)` feliratot olvashatjuk. Kerüljük a projekt nélküli állapotot.



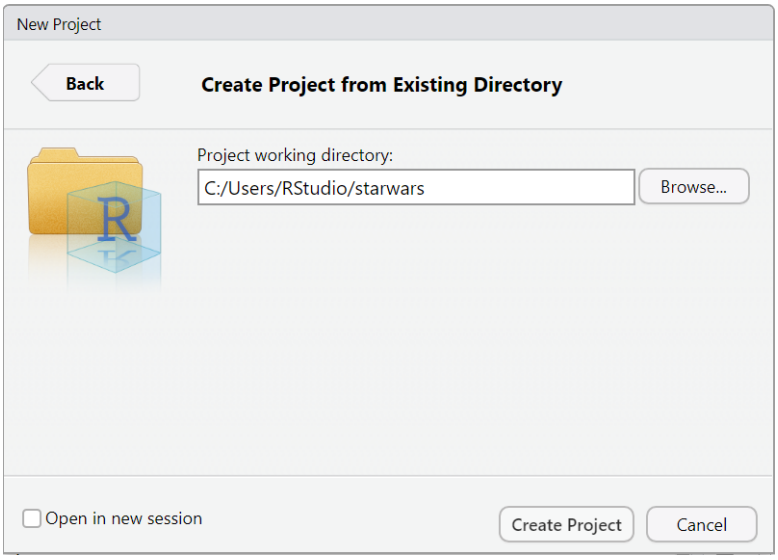
4.8. ábra: RStudio projekt létrehozása: 3. lépés (Posit team, 2025)

Minden adatfeldolgozási feladathoz – még a legkisebbhez is – hozzunk létre projektet. Minden állományt, amely a feladathoz tartozik a projektkönyvtáron belül helyezzünk el. Milyen állományok jöhetnek szóba: például parancsállományok, *Quarto* állományok, adatállományok, képállományok, dokumentációk és hivatkozásokat tartalmazó állományok. Érdemes ezeket rendezetten, ha szükséges, alkönyvtárakba szétosztva tárolni. Jó gyakorlat lehet, hogy a parancsállományokat és a *Quarto* állományokat közvetlenül a projektkönyvtárban (most ez az `elso_projekt`), az adatállományokat egy `adat` alkönyvtárban a projektkönyvtáron belül (most `elso_projekt/adat`) tároljuk, a képállományok és dokumentációk helye pedig lehet az `elso_project/kep`, illetve `elso_projekt/doku` alkönyvtár.

Válthatunk egy másik projektre is (`File / Open Project`), de be is zárhatjuk az aktív projektet (`File / Close project`). Később újra megnyithatjuk ezt is a `File / Open Project` segítségével. A megnyitás során természetesen az `.Rproj` kiterjesztésű projektállományt kell kiválasztanunk.

Ritkábban az is előfordulhat, hogy az adatfeldolgozási folyamatunkkal kapcsolatos állományok összegyűjtését korábban elkezdtük, és csak később szeretnénk ezt a könyvtárat egyben *RStudio*

projektkönyvtárként is felhasználni.



4.9. ábra: RStudio projekt létrehozása: létező könyvtár megadása (Posit team, 2025)

Korábban létrehozott könyvtárból szintén a `File / New Project` menüpont segítségével hozhatunk létre *RSudio* projektkönyvtárat. Itt azonban az `Existing Directory` opciót kell kiválasztani (4.6. ábra). Ezt követően ennek a létező könyvtárnak az elérési útját kell megadnunk az 4.9. ábrán látható beviteli mezőben.

Végül foglaljuk össze, milyen előnyökkel jár a projekt használata:

- a logikailag egy adatfeldolgozási folyamathoz tartozó állományainkat fizikailag is együtt tudjuk tartani.
- projekt üzemmódban az *RStudio* az aktuális könyvtárat a projektkönyvtárra állítja, így relatív hivatkozást használhatunk a kódunkban, amely a projekt hordozhatóságát biztosítja különböző számítógépek között.

4.1.9. Billentyűparancsok

Az *RStudio* legfontosabb billentyűparancsa a `Ctrl-Enter`, amely a parancsot a konzolba küldi végrehajtásra. Van még néhány további billentyűparancs, amelyet érdemes felsorolni, hiszen ezek használatával gyorsítani, egyszerűsíteni tudjuk a munkánkat.

- `Ctrl-Shift-N`: új parancsállomány létrehozása,
- `Ctrl-S`: állomány mentése,
- `Ctrl-W`: lap bezárása a `Forrás` panelen,

- `Ctrl-Tab` / `Ctrl-Shift-Tab`: aktív lap léptetése előre és hátra a `Forrás` panelen,
- `Ctrl-F`: szöveg keresése és cseréje,
- `Tab`: kód kiegészítése,
- `Ctrl-Shift-C`: kijelölt sorok be- vagy kikommentelése,
- `Ctrl-Alt-Fel` / `Ctrl-Alt-Le` (és `Shift+Jobbra` / `Shift+Balra`): a kurzor magasságának állítása (és az oszlopszélesség beállítása) több sor szerkesztésére,
- `Esc`: a `Konzol` panelen kilépés a folytatás promptból, a `Forrás` panelen kilépés a többsoros szerkesztésből,
- `Alt-Fel` / `Alt-Le`: sor mozgatása fel vagy le,
- `Alt-Shift-K`: billentyűparancsok módosítása,
- `Alt--`: értékadó operátor (`<-`) beszúrása,
- `Ctrl-Shift-M`: pipe operátor (`|>`) beszúrása,
- `Ctrl-Enter`: az aktuális sor vagy a kijelölt rész futtatása,
- `Ctrl-Alt-R`: a teljes parancsállomány futtatása,
- `Ctrl-Shift-P`: a kurzor feletti csonkok parancsainak futtatása,
- `Ctrl-+` / `Ctrl--`: betűméret nagyítása vagy kicsinyítése,
- `Ctrl-Shift-F10`: a munkamenet újraindítása.

Ha valamelyik kombináció nem működik a számítógépünkön, akkor a `Tools / Modify Keyboard Shortcuts` menüpont alatt új billentyűparancsot adhatunk az ott felsorolt funkciókhoz.

Összefoglalás

Az adatelemzési munka során az *RStudio* -t használjuk projekt üzemmódban, miközben *Quarto* állományba gyűjtjük az elemző R parancsokat és az egyéb magyarázó és fejléc szövegeket. Ebben a fejezetben ezt a tételmondatot töltöttük meg tartalommal. Megismertük az *RStudio* integrált környezetét. A *Forrás* panel lehetséges lapjai a parancsállomány, a *Quarto* állomány és az adatbázis. A *Konzol* panel legfontosabb lapja a *konzol*, amely központi szerepet játszik a munka során, hiszen a `Ctrl-Enter`-rel végrehajtott R parancsok eredménye és az esetleges hibaüzenetek is itt jelennek meg. A munka során a `.R` kiterjesztésű parancsállományok kiválóan alkalmasak a hosszabb elemzések R parancsainak tárolására, de ha a publikáláshoz is segítséget szeretnénk kapni, akkor inkább a a kötöttebb szerkezetű `.qmd` kiterjesztésű *Quarto* állományba rögzítsük parancsainkat. Az *RStudio* rutinszerű használatához a billentyűparancsok ismerete is hozzátartozik. A projektszemlélet az adatelemzéssel kapcsolatos állományok egyben tartásáról, és a hordozhatóság biztosításáról szól.

🎯 Feladatok

1. Bizonyosodjunk meg róla, hogy az alapvető szövegszerkesztési ismeretek birtokában vagyunk. Ismerjük az `Insert` billentyű funkcióját? Találjunk legalább 8 módszert, amely kizárólag a billentyű segítségével mozgatja a szövegkurzort! A szövegkijelölésnek milyen billentyűparancsait ismerjük? Milyen karaktertörlési lehetőségeket ismerünk? Ismerjük mindhárom vágóasztal-művelet billentyűparancsát?
2. Az *RStudio* mellett milyen más integrált fejlesztőeszközök léteznek az R-hez?
3. Az `Appearance` menüpont `Editor theme` beállításával változtassunk az *RStudio* színösszeállításán. Keressük meg a legjobban hozzánk illőt! Vegyük figyelembe, hogy hosszútávon a minél sötétebb háttér a jó választás.

4.2. Segítség az R használatához 🤔

ℹ Miről lesz szó? Ebben a fejezetben

- megismerjük az R hivatalos dokumentációit,
- az ún. cheet-sheet forrásokat,
- és a parancssorból elérhető súgó parancsokat.

Az R használatához számos segítséget találunk az interneten, a telepített *Alap R*-ben és az *RStudio*-ban egyaránt. Az online segítségek közül elsősorban a http://cran.r-project.org címen olvasható R dokumentációkat emeljük ki, ahol több tucat, elsősorban angol nyelvű leírást találunk az R megismeréséhez. A bal oldali `Documentation / Manuals` menüpont alatt találjuk például az R hivatalos bevezető dokumentumát (*An Introduction to R*), melynek tanulmányozása rendkívül nagy lépést jelenthet az R alaptudás megszerzéséhez. Az említett menüpont alatt találjuk még a *contributed documentation* linket is, amely számos rövidebb, és hosszabb dokumentációt tartalmaz, angol és más nyelveken. Itt találjuk Solymosi Norbert nagyszerű magyar nyelvű R bevezetőjét is.

Az R népszerűségének köszönhetően, nagyon sok további dokumentációt, tutoriált és példát találhatunk, ha az internetes keresőkhöz fordulunk. A fejezet végi egyik kitűzött feladatban összeállíthatjuk a saját listánkat.

Rendkívül népszerűek ma az ún. cheat-sheet-ek, amelyek néhány PDF oldalon sok ábrával, és a lényeg kiemelésével mutatják be egy-egy témakör legfontosabb tudnivalóit. Az *RStudio* `Help / Cheet Sheets` menüjéből, vagy közvetlenül a https://www.rstudio.com/resources/cheatsheets/ címről számos R téma cheet-sheet-jét érhetjük el.

Most tekintsük át azokat a súgókat, amelyek az R parancssorából indíthatók. Az R megismerését kezdhetjük a

```
help.start()
```

paranccsal, ahol számos, az R nyelvet részletesen tárgyaló dokumentum közül választhatunk.

Ha csak egyetlen függvénnyel kapcsolatban szeretnénk segítséget kérni, akkor használhatjuk a beépített súgórendszer parancsait. Adjuk ki a

```
help(t.test)
```

vagy a rövidebb

```
 ?t.test
```

parancsot, ha a `t.test()` függvényről szeretnénk részletes leírását kapni. A `?függvénynév` lehetőség, minden függvény esetében rendelkezésre áll a súgó kikérésére. Abban az esetben, ha nem ismerjük teljesen a függvény nevét, használhatjuk a

```
help.search("test")
```

parancsot, ekkor az összes olyan függvényt kilistázhatjuk, amelynek a nevében vagy a leírásában a `test` karaktersorozat előfordul.

Hasznos lehet továbbá a `find()` parancs, amely elárulja, hogy az illető függvény melyik már betöltött csomagban foglal helyet.

```
find("aov")
#> [1] "package:stats"
```

A fenti példából kiolvasható, hogy az `aov()` függvény a `{stats}` csomagban található.

Ugyancsak a betöltött csomagokban végez keresést az `apropos()` függvény, amellyel lehetőség van a parancssorból elérhető függvények vagy objektumok nevében keresni.

```
apropos("aov")
#> [1] "aov"         "eff.aovlist" "summary.aov"
```

Tovább segítheti az egyes függvények használatának elsajátítását az `example()` parancs, amely az egyes függvények használatára mutat példát.

```
example(t.test)
```

Utolsó lehetőségként ejtsünk szót a `demo()` függvényről, amellyel olyan beépített szkripteket futtathatunk, amelyek az R tudását, erejét hivatottak demonstrálni. Próbáljuk ki a következő parancsokat.

```
demo(graphics)
demo(persp)
demo(plotmath)
demo(Hershey)
```

🗂️ Összefoglalás

Az *RStudio* a parancsok gépelését számos módon könnyíti meg, de ha egy függvényről részletesebb leírást szeretnénk olvasni, akkor a `?függvénynév` parancsot is használjuk. Egy-egy témakör gyors megismeréséhez a puskákat (cheet-sheet) ajánljuk, amelyek az *RStudio* `Help / Cheet Sheets` menüjéből is elérhetők. Az R hivatalos honlapján hosszabb leírásokat is találunk.

🎯 Feladatok

1. Keressünk magyar nyelvű leírásokat az R-hez!
2. A közösségi médiában melyek az R legfontosabb fórumai?
3. Hogyan indíthatjuk el egy csomag beépített súgóját? Ismerjük meg így a `{fun}` csomagot!

4.3. Az *Alap R* használata 🤯

ℹ️ Miről lesz szó? Ebben a fejezetben

- megtanuljuk az *Alap R*-ben a konzol,
- és a parancsállományok használatát,
- az *R Commander* kezelését,
- valamint a kötegelt feldolgozás módszereit.

Amennyiben nagygépes környezetben dolgozunk, vagy valamilyen oknál fogva az *RStudio*-t nem tudjuk használni, akkor az *Alap R* lehet az egyetlen lehetőség R parancsok futtatására. Ebben az esetben sajnos le kell mondanunk a parancsok kényelmes bevitelét és végrehajtását támogató interaktív eszközökről, de természetesen az R teljes ereje, összes függvénye továbbra is rendelkezésünkre áll. Ebben a részben az *Alap R* lehetőségeit tekintjük át.

Az *Alap R* elindítása az adott platformon a megfelelő bináris állomány futtatását jelenti.

- Windows operációs rendszerekben az R indítása többnyire az Asztalon lévő R ikon segítségével lehetséges. Ez az `RGui.exe` grafikus felhasználói felülettel rendelkező alkalmazást indítja, amelynek legfontosabb része a külön ablakban (*R Console*) megjelenő konzol (4.10. ábra).
- MacOs környezetben indítsuk el az `R.app` alkalmazást, amely egyetlen konzolt tartalmaz.
- Linux környezetben az `R` parancs futtatásával szintén egy konzolt kapunk.

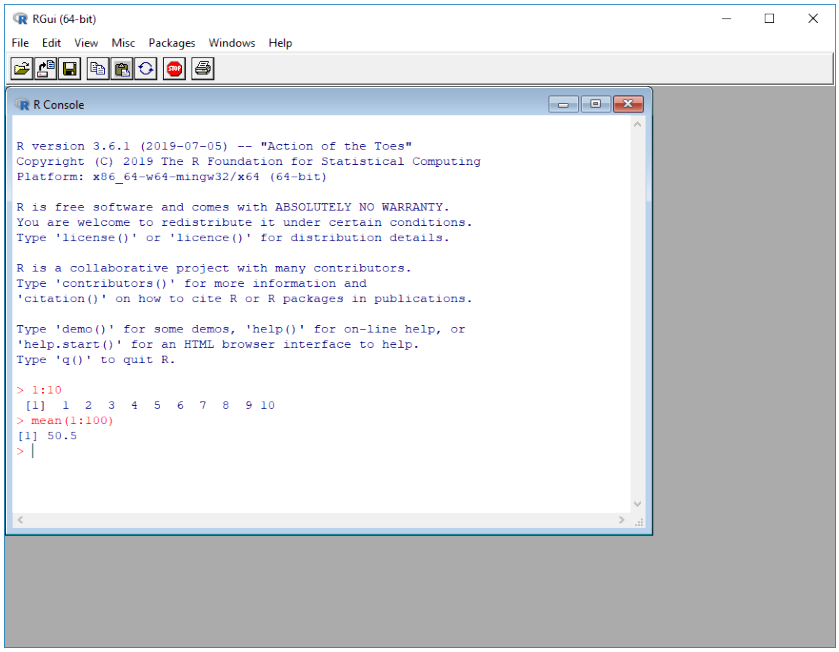
4.3.1. A konzol használata

A konzol az *Alap R* környezet központi része mindegyik platformon. A konzol működése lényegében megegyezik a korábban megismert *RStudio*-s konzol működésével: egysoros parancsokat gépelünk be a prompt (`>`) után, `Enter`-t nyomunk, majd az R interpreter értelmezi és végrehajtja a begépelt parancsot, és megjelenik az eredmény. A 4.10. ábrán a Windows környezetben használható *RGui* alkalmazás látható, miután a konzolba két parancsot gépeltünk be és hajtottunk végre.

Az *RStudio* konzoljának minden korábban említett alapfunkciója az *Alap R* konzoljában is elérhető, tudjuk használni a parancsok történetét, a kódkiegészítést a `TAB` billentyűvel, és a folytatás prompt (`+`) is megjelenik befejezetlen sorok esetén. Sőt a Windows alatt futó *RGui* ismeri a parancsállományokat is, bár a gépeléshez korántsem kapunk annyi támogatást mint az *RStudio*-ban.

4.3.2. Parancsállományok az *RGui*-ban

Az *RGui* a Windows-os *Alap R* része, és ahogyan láthattuk, egy nagyon egyszerű grafikus környezet, amelynek központjában a konzol található (*R Console* ablak a 4.10. ábrán). Az *RGui* nagyszerű tulajdonsága, hogy támogatja a parancsállományok használatát. Az *RGui*-ban találunk menüpontokat (`Fájl / Új szkript`, `Fájl / szkript megnyitása` és `Fájl / Ment`), amelyekkel létrehozhatunk, megnyithatunk, és elmenthetünk parancsállományokat. Tudjuk, hogy a parancsállományok használata lényegesen leegyszerűsíti az adatelemzés folyamatát, de fontos műveletként jelenik meg az átirányítás, amely a szövegszerkesztőben összegyűjtött



4.10. ábra: *RGui* alkalmazás a konzollal Windows környezetben (R Core Team, 2025)

parancsokat vezeti át a konzolba. Az *RGui*-ban ez a `Ctrl-R` billentyűkombinációval lehetséges – ez gyakorlatilag az *Rstudio*-beli `Ctrl-Enter` –, de a `Szerkesztés / Sor vagy kijelölés futtatása` vagy az `Szerkesztés / Minden futtatása` menüpontok is rendelkezésre állnak. A soronkénti végrehajtás mellett itt is lehetőség van kijelölt szövegrészek végrehajtására, de több sort, a teljes parancsállományt, vagy valamelyik sor egy részét is elküldhetjük a konzolba a `Ctrl-R` segítségével.

4.3.3. *R Commander*

Eddig az R használatának két lényegesen eltérő módját mutattuk be: a konzolos használatot és a parancsállományos használatot (a *Quarto* állományok használatát is ez utóbbi csoportba sorolhatjuk). Láttuk, hogy a konzol az *RStudio* és az *Alap R* központi része, de az *RStudio* és az *RGui* a parancsállományos használatot is támogatja. Mindegyik fenti használati mód parancsok gépelésével jár együtt.

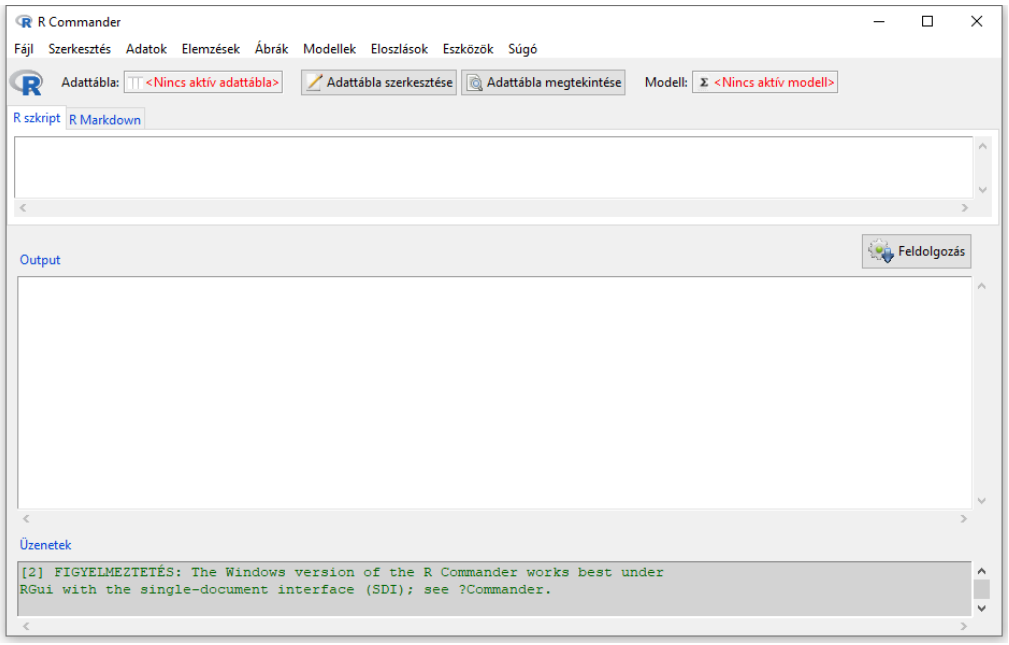
Azonban létezik egy harmadik, az eddigiektől lényegesen eltérő módja az R használatának. Parancsok gépelése nélkül, csupán egérkattintássokkal is végezhetünk statisztikai elemzést. Az R erre alkalmas beépített eszközét *R Commander*-nek nevezik, de külső eszközök is képesek az R parancssoros lényét elfedni előlünk. Ilyen külső eszköz például a *jamovi* és a *JASP*.

Mindhárom felsorolt eszközben közös, hogy grafikus felhasználói felületen mozgunk, és egérkattintással, menüben való navigálással, vezérlőelemek (rádiógombok, jelölőnégyzetek, listák, nyomógombok, beviteli mezők) használatával magyarázzuk el a kívánt tevékenységet.

A továbbiakban az *R Commander* lehetőségeit tekintjük át röviden. Az *R Commander* az `{Rcmdr}` nevű csomagban foglal helyet, így használatához ezt a csomagot telepítenünk kell. Ezt követően a `library()` függvény segítségével tudjuk elindítani az *R Commander*-t:

```
# install.packages("Rcmdr")  # R Commander telepítése
library(Rcmdr)               # R Commander indítása
```

Az indítás után egy külön *R Commander* ablak jelenik meg (4.11. ábra), melynek felépítése fentről lefelé a következő: (1) a gazdag menürendszer, (2) az eszköztár az aktuális adatbázis (`Adattábla`) mezővel és az `Adattábla megtekintése` gombokkal, (3) a parancsállomány vagy *R Markdown* lapok (a *Quarto* elődje), (4) az output számára fenntartott szöveges mező, és (5) az üzenetek helye. Megjegyezzük, hogy a 4.11. ábrán látható *R Commander*-t az *Alap R*-ből indítottuk. Amennyiben *RStudio*-ból adjuk ki a `library(Rcmdr)` parancsot, akkor a 4. és az 5. elem, azaz az output és az üzenetek rész nem lesz látható, mert az *RStudio* konzolja ezeket magába integrálja.



4.11. ábra: Az *R Commander* induló ablaka (Fox és mtsai., 2024)

A kilépést az *R Commander*-ből a `File / Kilépés` menüpont segítségével kezdeményezhetjük. Kilépés után az *R Commander* újraindításához a következő parancsokat kell használnunk:

```
# ha véletlenül bezártuk az R Commander-t
detach(package:Rcmdr)
library(Rcmdr)
```

Az *R Commander* lényegét a legkönnyebben úgy tudjuk szemléltetni, ha egérkattintásokkal is megoldjuk a *Csillagok háborúja* c. filmmel kapcsolatos adatelemzési feladatunkat. Első lépésként telepítsük a `{dplyr}` csomagot. Természetesen, ha már korábban a telepítést bármilyen apropóból elvégeztük, akkor ezt nem kell megismételni, de a teljesség kedvéért kezdjük azzal, hogyan tudunk interaktívan csomagot telepíteni. Az *RGui*-ban válasszuk ki a `Csomagok / Csomag(ok) telepítése` menüpontot, ha szükséges válasszunk a tükörszerverek közül, majd válasszuk ki a megjelenő listából a `{dplyr}` csomagot. Ezt követően már *R Commander*-ben töltsük be a `{dplyr}` csomagot az `Eszközök / Csomag(ok) betöltése` menüponttal. Keressük meg a listában a `{dplyr}` csomagnevet és kattintsunk az `OK` gombon. Ezt követően olvassuk be a `starwars` adatbázist a `{dplyr}` csomagból, az `Adatok / Csomagban lévő adatok / Adattábla beolvasása betöltött csomagból` menüpont segítségével. Kattintsunk duplán a `{dplyr}` csomagneven, majd a jobb oldali listában szintén duplán a `starwars` adatbázison, majd az `OK` gombbal fejezzük be a műveletet. Figyeljük meg, hogy az `R szkript` és `R Markdown` lapok tartalmazni fogják az egérrel elmutogatott tevékenységeinknek megfelelő R parancsokat, illetve az output és üzenetek részben ezek végrehajtásáról is értesítést kapunk.

Még egy rendkívül fontos dolog történt a `{dplyr}` csomag `starwars` adatbázisának beolvasása után. Az eszköztárban az `Adattábla` részben már nem a `Nincs aktív adattábla` szöveg szerepel, hanem a `starwars` adatbázis neve. Azt kell megjegyeznünk az *R Commander* használata során, hogy mindig van egy kitüntetett, aktív adattáblánk, és minden további tevékenység, amit a menüpontok segítségével el tudunk érni, az erre a kitüntetett, aktív adattáblára vonatkozik. Az aktív adattáblát le lehet cserélni. Amennyiben nyitnánk egy másik adatbázist, akkor a `starwars` feliratú gombon kattintva, egy listából kiválaszthatnánk, hogy melyik adatbázisunk legyen az *R Commander*-ben aktív.

Folytassuk az adatelemzést az `Elemzések / Összegzések / Numerikus változók összegzése` menüpont kiválasztásával. A megjelenő dialógusdobozból válasszuk ki a `height` változót, az `Összegzés csoportonként` gombon kattintva pedig a `species` változót. Az `OK` gombok megnyomása után az output részben látjuk az elemzés eredményét.

Az *R Commander* nagyon hatékony eszköz gyors elemzések, egyszerű adatbetekintések elvégzésére. Számos menüpontot kínál az adatok beolvasásához, előkészítéséhez és az elemzéséhez. Ráadásul az egyes menüpontokban elmutogatott tevékenységek R parancsait is szorgalmasan gyűjti, így azokat a `File / Szkript mentése` vagy `File / R Markdown mentése` kiválasztásával, el is tudjuk menteni magunk számára. Az *R Commander* vagy a *jamovi* és *JASP* ismerete nagyban hozzájárul a hatékony adatelemzéshez.

Végül megemlítjük, hogy az *R Commander* tudása kibővíthető beépülő modulok (plugin-ek) segítségével. Ezek új menüpontokat, dialógusdobozokat és természetesen új függvényeket tartalmaznak. A beépülő modulok csomagok formájában érhetők el. Például az `Easy R` beépülő modul telepítéséhez az `RcmdrPlugin.EZR` csomagra van szükség.

```
# egy beépülő modul telepítése
install.packages("RcmdrPlugin.EZR")
```

Telepítés után a beépülő modul betöltésére is szükség van, csak így tudjuk az új funkciókat elérni. Ezt az `Eszközök / Rcmdr plugin(ok) betöltése` menüpontban tehetjük meg. Az *R Commander* újraindulása után, már az új menüszerkezetet fogjuk látni. Az *R Commander hivatalos oldalán* részletesebb információkat olvashatunk.

4.3.4. Kötegelt futtatás

Ha felidézzük az eddig tanultakat az R használati módjairól, akkor világos, hogy mindegyik az interaktív használathoz kötődik. Egy tipikus adatelemzési munka során pontosan erre van szükség: kezdeményezzük egy művelet végrehajtását és várjuk az eredményt. Újabb művelet, újabb output. Ezt a fajta interaktív használatot láttuk a konzolban, a parancsállományok és *Quarto* állományok esetén, valamint az *R Commander*-ben is. Azonban az interaktív használat mellett beszélünk ún. kötegelt feldolgozásról is. Ez azt jelenti, hogy egy parancsállomány összes sorát egyetlen lépésben hatjuk végre. Nem vagyunk kíváncsiak a soronkénti eredményekre, a teljes szkriptállomány futtatása ad olyan eredményt, amelyre nekünk éppen szükségünk van. Kötegelt futtatásra a `source()` függvényt használhatjuk, valamint az *Alap R* egy külső alkalmazását, az `Rscript` programot.

Tegyük fel, hogy egy `netflix.R` parancsállományban összegyűjtöttük az összes olyan R sort, amely egyetlen ábra létrehozásához szükséges. Ez az ábra meglehetősen összetett, mert az egyes években megjelent filmek és sorozatok számát tartalmazza, és viszonylag sok adatelőkészítési műveletet előzte meg. Ezek nem mindig izgalmasak számunkra, annál inkább maga az ábra, amelynek létrehozása a `netflix.R` egyetlen célja.

A következő sort az *Alap R* vagy az *RStudio* konzoljába/parancsállományába, vagy az *RStudio Quarto* állományába is elhelyezhetjük. A `source()` függvény a `netflix.R` minden sorát végrehajtja és reményeink szerint előállítja a kívánt ábrát.

```
source("netflix.R", echo = T)
```

A `source()` függvény kicsit másként közelít a parancsainkhoz, mint amit megszoktunk az interaktív konzolos és parancsállományos használat során. A `source()` először a teljes állományban ellenőrzi a parancsok szintaktikai helyességét, és csak akkor kezdi el az első majd az azt követő parancsok végrehajtásához, ha mindent rendben talált.

Másik lehetőség parancsállomány kötegelt futtatására, az `RScript` program, amely ugyanúgy az *Alap R* része, mint a konzol vagy az interpreter. Az operációs rendszer parancssorából kell kiadnunk a következő parancsot:

```
Rscript --vanilla netflix.R > output.txt
```

A fenti sor hatására ugyanúgy létrejön a kívánt ábra, de az `output.txt`-ben megkapjuk a futás közben keletkező egyéb outputokat is.

Kötegelt feldolgozásra viszonylag ritkán van szükségünk, akkor is többnyire nagygépes környezetben. Az interaktív használat a legtöbb adatelemzési munka során elegendő rugalmasságot ad.

Összefoglalás

Amennyiben az *RStudio* használatára nincs lehetőségünk, akkor az *Alap R* eszközeivel is kiválóan megoldhatjuk adatelemzési feladatainkat. A konzol és az *RGui* parancsállományai interaktív parancsvégrehajtást biztosítanak, a `source()` függvény és az `RScript` alkalmazás pedig az `.R` kiterjesztésű parancsállományok kötegelt feldolgozását segíti. Az *R Commander* menüparancsai gépelése nélkül teszik lehetővé elemzések végrehajtását, mindössze a megfelelő almenüpontot kell kiválasztani, majd a dialógusdobozban elvégezni a szükséges beállításokat. Érdemes kipróbálni a *jamovi* és a *JASP* statisztikai programokat is, amelyek R-t használnak a háttérben, de szintén grafikus felhasználói felülettel rendelkeznek.

Feladatok

1. Foglaljuk össze az R használati módjait! Soroljuk fel mind a négy lehetőséget!
2. Hasonlítsuk össze a parancsállományok használatát *RGui*-ban és *RStudio*-ban!
3. Hasonlítsuk össze a parancsállományok és a *Quarto* / *R Markdown* használatát *R Commander*-ben és *RStudio*-ban!
4. Töltsük le és telepítsük az ingyenesen elérhető *jamovi* és a *JASP* statisztikai programokat, majd nyissuk meg a beépített adatbázisait, és végezzünk néhány egyszerűbb elemzést! Ha elakadunk, keressünk videótutoriált az eszközök használatáról. Melyik eszköz tetszik jobban? Miben hasonlítanak és miben térnek el?

5. Az R nyelv


Jaj,
nem tudom
beilleszteni az
R gyakorlását
a napirendembe...
R
GYAKOR-
LÁSA
ALVÁS
SULI
EVÉS
ALVÁS
SULI
EVÉS
R
GYAKOR-
LÁSA
ALVÁS
SULI
EVÉS
ALVÁS
Vagy mégis(?)...

Az előző fejezetekben megismertük az R környezetet, az *Alap R*, az *RStudio* és a csomagok telepítését, megtanultuk a projektek, parancsállományok és *Quarto* állományok létrehozását. Tudjuk, a különböző környezetekben eltérő módszerekkel hajthatjuk végre az R parancsokat: a konzolban az `Enter`, a Windows-os *RGui*-ban a `Ctrl-R`, míg az *RStudio*-ban a `Ctrl-Enter` billentyűkombinációt kell használnunk. A parancsok végrehajtása közben érdemes észben tartani, ha a folytatás prompt (`+`) feltűnik, akkor kattintsunk bele a konzolba, és nyomjuk meg az `ESC` billentyűt, így tudunk kilépni a befejezetlen sor szerkesztéséből.

E fejezet példáinak kipróbáláshoz hozzunk létre egy `gyakorlas` nevű új projektet az *RStudio*-ban (`File / New Project`), majd készítsünk egy `gyakorlas.qmd` *Quarto* állományt (`File / New File / Quarto Document`) és egy `gyakorlas.R` parancsállományt (`File / New File / R Script`). A fejezet példáit egyaránt gépeljük a *Quarto* állomány R csonkjaiba, illetve a parancsállomány egymást követő soraiba. A fejezet további részében az R nyelvre koncentrálunk, arra, hogy mit írunk, és nem arra, hogy hová írjuk a parancsokat.

5.1. Adatobjektumok 🙂

> Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük az egyszerű számolási lehetőségeket R-ben,
> - bevezetjük az aritmetikai operátor és a kifejezés fogalmát,
> - megismerjük az objektum létrehozását és elnevezését,
> - több parancs elhelyezését egy sorban,
> - és a megjegyzések használatát.

Az R nyelv megismerését számadatok írásával kezdjük. Az adatelemzés során a számszerű adatok kezelése a leggyakoribb, hiszen méréssel és számlálással is ilyen jellegű adatokhoz jutunk. Számszerű adat a testmagasságunk cm-ben kifejezve, az IQ-teszten elért pontszámunk, vagy a testvéreink és a Facebook ismerőseink száma is.

5.1.1. Számolás az R-ben

Kezdjük a számszerű adatok megismerését egy egyszerű sor begépelésével.

```
2 + 2
#> [1] 4
```

Végrehajtás után a konzolban láthatjuk az összeadás eredményét, a 4-et. Az eredmény előtt egy szögletes zárójelben lévő sorszámot is láthatunk (`[1]`), amely bonyolultabb outputokban segít eligazodni. Később az 5.3.3.2. fejezetben visszatérünk a `[1]` értelmezésére.

Látjuk, ebben az esetben az R úgy viselkedik, mint egy számológép. Kiszámolja a parancssorba gépelt algebrai kifejezés értékét, majd a képernyőn megjeleníti. Természetesen az összeadáson túl más műveletet is használhatunk.

```
4 + 6 * 2
#> [1] 16
```

A fenti példából látható, hogy az R követi a műveletek elvégzésének matematikában megszokott sorrendjét. Azaz a szorzás műveletre (`*`) hamarabb sor kerül, ennek eredménye 12. Ezt követi az összeadás (`+`), most már a 4 és a 12 között. Ennek az összeadás műveletnek az eredménye 16, ami egyben a kifejezés értéke is, tehát ez jelenik meg a konzolban.

Természetesen a matematikában megszokott módon változtathatunk a műveletek végrehajtásának alapértelmezett sorrendjén, azaz használhatunk kerek zárójeleket. Ezeket az R a megszokott módon értelmezi: a zárójelben szereplő műveletek végrehajtását előreveszi.

```
(4 + 6) * 2
#> [1] 20
```

A fenti példában az összeadás művelet lesz az első, amelynek az eredménye 10. Ezt követi a szorzás, így kapjuk a kifejezés értékeként a 20-at.

Ezeket a matematikában megszokott algebrai kifejezéseket, az R-ben egyszerűen kifejezésnek vagy – utalva arra, hogy a kifejezés értéke szám – *aritmetikai kifejezésnek* nevezünk. Az eddigiek alapján az aritmetikai kifejezések tehát a következő nyelvi elemeket tartalmazhatják:

- számok, amelyeket *numerikus konstans*oknak nevezünk,
- műveleti jelek, amelyeket *aritmetikai operátor*oknak nevezünk,
- és kerek zárójelek.

A fentiek alapján összetettebb aritmetikai kifejezéseket is formálhatunk. Az R minden esetben kiszámolja a kifejezések értékét – azaz *kiértékeli* a kifejezést –, és a kapott értéket megjeleníti a konzolban.

```
4^2 - 3 * 2 + 1
#> [1] 11
(104 - 20)/6 - 4 * 7 * 10/(5^2 - 5)
#> [1] 0
```

5.1. táblázat: Matematikai operátorok precedenciájuk csökkenő sorrendjében *(saját szerkesztés)*

Operátor	Művelet	Példa
`^ **`	hatványozás	`2^3;2**3`
`+ -`	előjelek	`+3.3;-.5`
`%% %/%`	maradékos osztás és egész osztás	`13%%4;15%/%4`
`* /`	szorzás és osztás	`2*3;4/2`
`+ -`	összeadás és kivonás	`2+3;2-3`

Az aritmetikai kifejezések használata során ne felejtkezzünk el a műveletek alapértelmezett végrehajtási sorrendjéről. A műveletek megjelenítését most az operátorok végzik, melyeknek fontos tulajdonsága, hogy mennyire szorosan kötik magukhoz az adatokat (vagy más néven az operandusokat). Az operátorok ezen fontos tulajdonságát *precedenciának* nevezzük. Az R-ben használható aritmetikai operátorokat a precedenciájuk csökkenő sorrendjében az 5.1. táblázat tartalmazza.

Például a hatványozás és az előjel operátor precedenciája eltér egymástól, a hatványozás nagyobb precedenciájú, azaz szorosabban köti magához az adatokat, így végrehajtása megelőzi az előjel operátort. Ha nem vagyunk elég óvatosak, és plusz zárójelek segítségével nem biztosítjuk a kívánt végrehajtási sorrendet, akkor "váratlan" eredményhez juthatunk. A lenti példában láthatjuk, hogy zárójelek nélkül a nagyobb precedenciájú hatványozás az elsőként végrehajtott művelet.

```
-2^2  # először hatványozás, majd előjel
#> [1] -4
(-2)^2  # először előjel, majd hatványozás
#> [1] 4
```

Eddig láthattuk, hogy kifejezéseinket operátorok, numerikus konstansok és zárójelek segítségével építettük fel. Ezek a kifejezések két alkotójukban is általánosíthatók:

- általánosítható a kifejezés adat része, amelyet eddig a numerikus konstansok képviseltek (ezekből lesznek az *objektum*ok),
- általánosítható a kifejezés művelet része, amelyet eddig az operátorok jelenítettek meg (ezek lesznek a *függvény*ek).

Az adatrész általánosítása tehát az adatobjektum (vagy röviden objektum), a műveleteké pedig a függvényobjektum (vagy röviden függvény). Ezeket tekintjük át a következőkben.

5.1.2. Objektumok

Ha egy kifejezés értéket nem egyszerűen a képernyőn szeretnénk megjeleníteni, hanem azt később is fel szeretnénk használni, akkor objektumot[1] kell létrehoznunk. Az objektumok révén a memóriába rögzíthetünk tetszőleges értékeket, később pedig elővehetjük és felhasználhatjuk ezeket az értékeket.

Tudjuk, ha a lenti aritmetikai kifejezést a parancssorba írjuk, az R miután kiértékelte a kifejezést, a kifejezés értékét megjeleníti a konzolban. Ez az érték azonban a megjelenítés után rögtön el is vész, többször nem használhatjuk fel.

```
1157/13 + 2^3
#> [1] 97
```

Ha létrehozunk egy x nevű objektumot, akkor ezt az értéket további kifejezésekben is szerepeltethetjük. Minden olyan helyen, ahol eddig számok jelentek meg a kifejezésekben, oda ez az x objektumnév is beírható.

```
x <- 1157/13 + 2^3
```

A fenti sor végrehajtása után írhatjuk a következőket, hiszen a kifejezések kiértékelése során az x objektum memóriában tárolt értékével fog számolni az R.

```
x+2       # mintha 97+2 lenne
#> [1] 99
2*x^3+5   # mintha 2*97^3+5 lenne
#> [1] 1825351
```

Minden objektumnak van neve és tartozik hozzá a memóriában egy terület, ahol a kérdéses érték tárolásra kerül. Esetünkben az objektum neve x, a hozzá tartozó memóriaterületen pedig a 97 értéket tárolja az R. Az objektum leegyszerűsítve tehát egy név-érték pár, ahol a nevet és a memóriában eltárolandó értéket is mi magunk választjuk meg.

Az objektumok kezeléséhez 3 művelet kapcsolódik:

- objektum létrehozása,
- objektum értékének lekérdezése, és
- objektum értékének megváltoztatása.

[1] Más programozási nyelvekben az "objektum" helyett a "változó" elnevezést használják, de a változó fogalma már foglalt a statisztikában, így szerencsésebb a memóriában tárolt adatokra objektumként hivatkozni.

5.1.2.1. Objektumok létrehozása

Objektumot értékadással hozhatunk létre. Az értékadás tartalmaz egy értékadás operátort, melynek alakja `<-` (balra nyíl), vagyis egy kisebb jel és egy mínusz előjel egymás után írva szóköz nélkül[2].

Az értékadás általános alakja:

```
objektumnév <- kifejezés    # értékadó utasítás
```

Ahol lehet a továbbiakban ezt a balra nyíl alakú értékadó operátort használjuk az értékadás során, és nem a szintén legális egyenlőségjelet (`=`). A balra nyíl írását az *RStudio* az `Alt+-` segítségével támogatja, így a bevitele nem okozhat nehézséget. Az egyenlőségjelet megtartjuk a függvényargumentumok elnevezésére. Az egyszerűség kedvéért a balra nyíl előtt lévő objektumnevet az értékadás bal oldalának, az utána lévő kifejezést az értékadás jobb oldalának nevezzük.

Ha olyan objektumnevet szerepeltetünk az értékadásban, amely még nem létezik, akkor az R létrehoz egy ilyen nevű új objektumot, és a hozzá tartozó memóriaterületen pedig az értékadás jobb oldalán lévő kifejezés kiértékelése után kapott értéket tárolja el.

```
a <- 1 + 2  # objektum létrehozása
```

A fenti sor végrehajtása után a konzolban nem jelenik meg eredmény, mégis egy nagyon fontos dolog történik, létrejön az `a` nevű objektum, amelynek értéke 3 lesz mindaddig, amíg ezen nem változtatunk. A munkánk során létrehozott objektumok a memória egy speciális területére, a *munkaterület*re (*workspace*) kerülnek.

Ha az értékadásban használt objektum már létezik, akkor a jobb oldali kifejezés kiértékelése után a kapott értékkel felülírja a bal oldali objektumhoz tartozó memóriaterületet. Ezzel a módszerrel tehát a korábban létrehozott objektum értékét módosíthatjuk.

A már létező `a` objektum értékét könnyen megváltoztathatjuk.

```
a <- 10/3  # objektum értékének megváltoztatása
```

[2]További értékadás operátorok a `->`, `<<-`, `->>` és az `=`. Ezeket nem használjuk ebben a könyvben.

5.1.2.2. Objektumok értékének lekérdezése

Az objektum memóriában tárolt értékét le is kérdezhetjük. A legegyszerűbb mód erre, ha az objektum nevét a parancssorba írjuk és végrehajtjuk a sort, máris megkapjuk az objektum memóriában tárolt értékét.

```
a  # vajon mi az 'a' objektum értéke
#> [1] 3.333333
```

Objektumok tetszőleges kifejezésben megjelenhetnek, akár egy értékadás jobb oldalán lévő kifejezésben is. A kifejezések kiértékelésében az objektum a memóriában tárolt értékével vesz részt.

```
a * 3            # a kifejezés értéke konzolba kerül
#> [1] 10
a <- 4 + a * 3   # megváltozik az objektum értéke, nincs output
a                # megtudjuk az objektum értékét
#> [1] 14
```

A fenti sorokból kiolvasható, hogy immár az `a` objektum értéke 14.

5.1.2.3. Objektumok elnevezése

Az objektumok elnevezésére eddig egyetlen betűt (karaktert) használtunk, de ez elég ritka eset a munka során. Helyes gyakorlat, ha az objektum neve utal az objektum tartalmára, céljára. Ha például testmagasságot tárolunk el egy objektumban, akkor írhatjuk a következőt:

```
magassag <- 179
```

A fenti sor létrehozza a munkaterületen a `magassag` nevű objektumot 179 értékkel.

Az objektumok elnevezésére

- betűket,
- számjegyeket,
- és az aláhúzás (_) vagy pont (.) szimbólumokat használhatjuk.

Az objektum neve csak betűvel vagy ponttal kezdődhet, számjeggyel vagy aláhúzással nem. Továbbá a név nem lehet az R-ben már lefoglalt kulcsszó, mint például `if`, `function` vagy `TRUE` (a kulcsszavak listáját a `?Reserved` paranccsal ismerhetjük meg). Hagyományosan a pont karaktert használjuk az objektumnevekben a tagolásra (például `magassag.peter` Péter magasságának tárolására), bár manapság az aláhúzás is népszerű. Az R a magyar ékezetes karakterek használatát is megengedi az objektumnevekben, de csakúgy mint az állományok és könyvtárak elnevezésében, érdemes ezek használatát mellőzni.

Az objektumoknak érdemes "beszédes" nevet választani, még ha ennek az ára némi extra gépelés is. Tudjuk, a `TAB` billentyű segíti a gépelést az *RStudio*-ban.

Az R kis- és nagybetű érzékeny, vagyis az `x` és a `X` különböző objektumoknak számítanak. Például a következő

```
pulzus.atlag <- 72
```

parancs után a

```
Pulzus.atlag
#> Error: object 'Pulzus.atlag' not found
```

sor hibát jelez (`Error: object 'Pulzus.atlag' not found`), azaz a `Pulzus.atlag` objektumot nem találja az R. Minden olyan esetben, ha nem létező objektumra hivatkozunk, a fenti hibaüzenet jelenik meg a konzolban.

Amennyiben gondoskodunk nagy `P`-vel kezdődő objektumról is, akkor lehetőségünk van hibaüzenet nélkül mindkét objektum értékének kiíratására.

```
Pulzus.atlag <- 69           # új objektumot hozunk létre
Pulzus.atlag; pulzus.atlag   # két parancs egy sorban
#> [1] 69
#> [1] 72
```

A gyakorlatban kerüljük el az olyan helyzeteket, amikor két objektumnév csak kis- nagybetűk használatában tér el.

A fenti példában egy további apró újdonság is szerepelt. Ha egy parancssorban több utasítást szeretnénk elhelyezni, akkor ezeket pontosvesszővel (`;`) kell elválasztanunk. A pontosvesszővel elválasztott utasításokat az R értelmező egymás után, balról jobbra haladva hajtja végre, mintha külön sorba írtuk volna őket. A lenti sor 3 kifejezést (parancsot) tartalmaz pontosvesszővel elválasztva, mindegyik eredménye külön-külön jelenik meg a konzolban, mintha 3 különböző sorba írtuk volna őket.

```
1+2; 3+4; 5+6     # három kifejezés egy sorban
#> [1] 3
#> [1] 7
#> [1] 11
```

5.1.3. Megjegyzések

Nagyon sok példában láttunk már magyar nyelvű, magyarázó, értelmező szövegrészeket az R parancsok körül. Ezek az R *megjegyzések*. Megjegyzést az R-ben a kettős kereszt (`#`) karakter használatával vezetünk be. Az R értelmező a kettős kereszttől a sor végéig tartó részt figyelmen kívül hagyja. Itt helyezhetjük el a paranccsal kapcsolatos magyarázatainkat magunk vagy a kódot később olvasók számára. Teljes sorokat, vagy a sorok végét tudjuk így kivonni a végrehajtás alól.

```
# Érdekes tény, ha a 153 számjegyeit köbre emeljük,
#   majd összeadjuk őket, pontosan 153-at kapunk
1^3+5^3+3^3       # hatványozás a ^ operátorral
#> [1] 153
1**3+5**3+3**3    # hatványozás a ** operátorral
#> [1] 153
```

Nem kizárólag magyarázó szövegek kerülhetnek megjegyzésbe, sokszor R parancsok végrehajtását akadályozzuk meg ezzel a módszerrel. Úgy kerülhetjük el egy parancs végrehajtását, hogy nem kell kitörölnünk a parancsállományból vagy a *Quarto* állományból, egyszerűen csak megjegyzésbe kell tennünk őket. Emlékezzünk vissza, hogy az 3.1.3. fejezetben a csomagok telepítésért felelős parancsok esetében kifejezetten javasoltuk a megjegyzések használatát:

```
# xkcd: Randall Munroe webképregényei
# install.packages("RXKCD")
library(RXKCD)              # csomag betöltése
searchXKCD("Star Wars")     # keresés címben vagy leírásban
getXKCD(1769)               # webképregény megjelenítése
```

Végül megemlítjük, hogy az *RStudio*-ban egyszerre több kijelölt sort tudunk megjegyzésbe tenni, vagy onnan kivenni a `Ctrl-Shift-C` segítségével.

Összefoglalás

Egyszerű kifejezéseket építhetünk numerikus konstansok (számok), operátorok és kerek zárójelek segítségével. A legfontosabb matematikai operátorok a négy alapművelet és a hatványozás. A kifejezés kiértékelése balról jobbra sorrendben történik, de ezt felülírhatja a kerek zárójelek használata és az operátorok precedenciája. Egy kifejezés értékét eltárolhatjuk a memória speciális területén, a munkaterületen. Ehhez az értékadó operátorral (`<-`) létre kell hoznunk egy új objektumot. Az objektum egy név-érték páros. Az objektum létrehozása után az objektum neve megjelenhet egy tetszőleges kifejezés adat részében. Több parancsot a pontosvesszővel (`;`) írhatunk egy sorba. Megjegyzéseket a kettős kereszt (`#`) segítségével helyezhetünk el.

Feladatok

1. Gondoljuk át, mi lehet a következő algebrai kifejezés eredménye, majd ellenőrizzük R-ben is: $8/2(2+2)$.
2. Gondoljuk át, hogy a `.342e1` név miért nem lehet érvényes objektumnév? Próbáljuk ki a `make.names(".342e1")` parancsot, majd tanulmányozzuk a `?make.names` leírást!
3. Magyarázzuk meg a `make.names(c("", "", ""))` és a `make.names(c("", "", ""), unique = T)` parancsok közötti különbséget!
4. Gondoljuk át, hogy egy parancsállomány mely pontjain érdemes feltétlenül megjegyzéseket használni!
5. Jelentősen segíthetjük a navigációt az *RStudio* parancsállományaiban, ha bizonyos megjegyzések végére ezt írjuk: `----` (szóköz és négy mínusz jel). Hogyan használhatjuk ezt a lehetőséget az *RStudio*-ban, és milyen előnyei vannak?
6. Az *RStudio*-ban parancsállomány (`.R`) szerkesztése közben próbáljuk ki a `Ctrl-Alt-R` billentyűparancsot, és a hozzá kapcsolódó `Shift-Alt-J` billentyűparancsot is. Mi a jelentése az `Alt-L`, `Shift-Alt-L`, `Alt-O` és `Shift-Alt-O` billentyűparancsoknak? A most megismert funkciók hogyan válthatók ki *Quarto* (`.qmd`) állomány szerkesztése közben?
7. Vizsgáljuk meg ennek a három kifejezésnek az értékét: `2**3**4`, `(2**3)**4` és `2**(3**4)`. Miért van különbség? Mi okozza a különbséget?

5.2. Függvények 🙂

> **Miről lesz szó? Ebben a fejezetben**
>
> - áttekintjük a függvényhívás lehetőségeit a nevesített argumentumokkal, az alapértelmezésekkel és az argumentumok sorrendjének megváltoztatásával,
> - megismerjük a legfontosabb matematikai függvényeket, és
> - pontosítjuk a kifejezés fogalmát.

Az aritmetikai kifejezéseinkben használható operátorok nem teszik lehetővé minden matematikai művelet elvégzését. Mit tegyünk, ha a 2 négyzetgyökét szeretnénk kiszámolni? A négyzetgyökvonás operátor nem létezik az R-ben, de ebben a speciális esetben a hatványozás operátor segítségével is elérhetjük a célunkat.

```
# 2 négyzetgyöke
2^0.5
#> [1] 1.414214
```

Az R azonban más lehetőséget is biztosít a négyzetgyök kiszámítására és ez az `sqrt()` függvény.

```
# 2 négyzetgyöke
sqrt(2)
#> [1] 1.414214
```

A függvények valamilyen utasítássorozatot hajtanak végre és a számítás eredményét szolgáltatják. Esetünkben az `sqrt()` függvény egy szám (pozitív) négyzetgyökét számolja ki, annak a számnak a négyzetgyökét, amely a kerek zárójelek között szerepel. Tehát az R a paraméterben megadott 2 értékre meghívja az `sqrt()` függvényt, ami visszatér a 2 négyzetgyökével.

5.2.1. A függvényhívás szabályai

A függvényhívás általános alakja:

```
függvénynév(argNév1=arg1, argNév2=arg2, ..., argNévN=argN)
```

5.2. táblázat: Az R alapvető matematikai függvényei *(saját szerkesztés)*

Függvény	Leírás	Példa
`abs(x)`	abszolútérték függvény	`abs(-1)`
`sign(x)`	előjel függvény	`sign(pi)`
`sqrt(x)`	négyzetgyök függvény	`sqrt(9+16)`
`exp(x)`	exponenciális függvény	`exp(1)`
`log(x,base=exp(1))`	logaritmus függvény	`log(exp(3));log(8,10)`
`log10(x);log2(x)`	10-es és 2-es alapú logaritmus	`log10(1000);log2(256)`
`cos(x);sin(x);tan(x)`	trigonometrikus függvények	`cos(pi);sin(0);tan(0)`
`round(x,digits=0)`	kerekítés adott tizedesre	`round(c(1.5,-1.5))`
`floor(x)`	x-nél kisebb, legnagyobb egész	`floor(c(1.5,-1.5))`
`ceiling(x)`	x-nél nagyobb, legkisebb egész	`ceiling(c(1.5,-1.5))`
`trunc(x)`	x-hez közelebbi egész x és 0 között	`trunc(c(1.5,-1.5))`

A függvény neve ugyanazoknak a szabályoknak engedelmeskedik, amelyeket az objektumok nevénél megtárgyaltunk (lévén a függvény is egy objektum). A függvény neve után kerek zárójelben következnek a függvény argumentumai, amelyek a függvény utasításainak a bemenő paraméterei. A függvény a bemenő paraméterek alapján az utasításainak megfelelően egy visszatérési értéket fog szolgáltatni.

Egy függvény különböző hívásainál az előforduló argumentumok száma és azok sorrendje igen változatos képet mutathat. Elöljáróban elmondhatjuk, hogy a függvények argumentumai alapértelmezett értékkel is rendelkezhetnek, így ezek az argumentumok elhagyhatók. Továbbá, a függvények argumentumai névvel is rendelkeznek, amelyeket ha a függvény hívásánál felhasználjuk, az argumentumok sorrendje tetszőleges lehet.

Először tekintsük át az R alapvető matematikai függvényeit 5.2. táblázat). Nézzük meg részletesebben a `log()` függvényt. Ha kikérjük a súgóját a `?log` parancs begépelésével, akkor megtudhatjuk, hogy ez a legáltalánosabb logaritmus függvény, tetszőleges alap esetén hívható. Számunkra most a legfontosabb a súgónak az a sora, amely a logaritmus függvény használatát mutatja: `log(x, base=exp(1))`.

Ebből kiolvasható, hogy a `log()` függvénynek 2 argumentuma (más néven paramétere) van. Az elsőt `x`-nek, a másodikat `base`-nek nevezik. A második paraméter alapértelmezett értékkel is rendelkezik, tehát ez a paraméter a hívásnál elhagyható, míg az `x=` argumentum megadása kötelező. A `base=` paraméter értéke könnyen kideríthető az

```
exp(1)  #  Euler-féle szám, a természetes logaritmus alapja
#> [1] 2.718282
```

parancsból. Ezt az irracionális számot a matematikában *e*-vel jelöljük, és Euler-féle számnak nevezzük, ez a természetes logaritmus alapja. Ha nem határozzuk meg a második paramétert, akkor a `log()` függvény ezzel a természetes alappal (`base=exp(1)`) számítja ki az `x` logaritmusát.

Ezek alapján 2 természetes alapú logaritmusát a

```
log(2)  # 2 természetes alapú logaritmusa
#> [1] 0.6931472
```

függvényhívás adja meg. Azt is megtehetjük, hogy felhasználjuk hívásnál az argumentum nevét (`x`), és egy egyenlőségjel (`=`) felhasználásával ezt a 2 elé szúrjuk be.

```
log(x = 2)  # 2 természetes alapú logaritmusa
#> [1] 0.6931472
```

A fenti sor természetesen ugyanúgy a 2 természetes alapú logaritmusát szolgáltatja, csak most explicit módon közöltük, hogy az aktuális paraméterben szereplő 2-es értéket az `x=` nevű formális paraméternek feleltetjük meg. Ez most felesleges gépelést jelentett és általában is elmondhatjuk, hogy matematikai függvények esetében az oly gyakori `x=` argumentumnevet szokás szerint nem írjuk ki a függvényhívás során.

Hívjuk most két argumentummal a `log()` függvényt. A 100 10-es alapú logaritmusát a

```
log(100, 10)  # 100 10-es alapú logaritmusa
#> [1] 2
```

paranccsal tudhatjuk meg. A függvényhívásnál az `x=` formális argumentum a 100, a `base=` pedig a 10 értéket kapja. Természetesen ezt a hívásnál mi is rögzíthetjük a világosabb értelmezés kedvéért saját magunk számára a

```
log(100, base = 10)  # 100 10-es alapú logaritmusa
#> [1] 2
```

vagy akár

```
log(x = 100, base = 10)  # 100 10-es alapú logaritmusa
#> [1] 2
```

formában is.

Arra is van lehetőség, hogy megcseréljük az aktuális paraméterek sorrendjét. A legbiztonságosabb ekkor az összes paraméter nevesítése,

```
log(base = 10, x = 100)  # 100 10-es alapú logaritmusa
#> [1] 2
```

de két argumentum esetén így is egyértelmű a hozzárendelés:

```
log(base=10, 100); log(10, x=100)  # 100 10-es alapú logaritmusa 2x
#> [1] 2
#> [1] 2
```

Ha az argumentumok nevesítése nélkül cseréljük fel az aktuális paramétereket, akkor természetesen nem a várt eredményt kapjuk, mert a 10 100-as alapú logaritmusa lesz az eredmény.

```
log(10, 100)  # 10 100-as alapú logaritmusa
#> [1] 0.5
```

Kényelmi lehetőség az aktuális paraméterek elnevezésénél, hogy rövidítéseket is használhatunk, addig csonkolhatjuk az argumentum nevét, amíg az argumentumok egyértelműen azonosíthatók maradnak. Így a példában akár a `b=`-vel is helyettesíthetjük a `base=` argumentumnevet:

```
log(b = 10, 100)  # 100 10-es alapú logaritmusa
#> [1] 2
```

Mint korábban említettük, az `x=` argumentum nem rendelkezik alapértelmezett értékkel, így paraméter nélkül nem hívható a `log()` függvény.

```
log()
#> Error: argument "x" is missing, with no default
```

A fenti hibaüzenethez hasonlót láthatunk, ha egy függvényt nem megfelelő számú paraméterrel hívunk.

Eddig a függvények aktuális paramétereiként csak numerikus konstansokat használtunk, pedig valójában tetszőleges kifejezéseket is megadhatunk. A függvény hívása előtt ezek kiértékelődnek és a hívás során ezek az értékek rendelődnek a formális paraméterekhez.

```
alap <- 10; log(exp(1)); log(exp(4),base=alap); log(2*exp(2),b=alap/2)
#> [1] 1
#> [1] 1.737178
#> [1] 1.673346
```

A fenti példa a következő numerikus konstansokkal történő hívásoknak felel meg:

```
log(2.718282); log(54.59815, base=10); log(14.77811, base=5)
#> [1] 1
#> [1] 1.737178
#> [1] 1.673346
```

A függvények sokféle csoportja létezik az R-ben, a most látott matematikai függvények osztálya csak egy a sok közül. A következő fejezetekben függvények más csoportjait is megismerjük.

5.2.2. A kifejezés fogalma

Elérkezett az idő, hogy a kifejezés fogalmát pontosítsuk: **egy konstans, egy objektum vagy egy függvényhívás önmagában kifejezés, de ezek operátorokkal és kerek zárójelekkel helyesen összefűzött sorozata is kifejezés**.

Az R nyelv parancsai, vagy más néven utasításai lényegében kifejezések. Az R nyelvben egy parancs végrehajtása lényegében egy kifejezés kiértékelését jelenti, és a legtöbb esetben a kifejezés értékének megjelenítését a konzolban.

A munka során az R értelmező az utasítások egymás utáni kiértékelését végzi. Az utasításokat "újsor" karakter vagy pontosvessző választhatja el. A szintaktikailag helyes utasítások kiértékelése mindig egy értéket eredményez, ez lesz az utasítás értéke. Még akkor is rendelkezik értékkel az utasításunk, ha az nem jelenik meg a parancssorban, például az értékadó utasítás értéke a jobb oldali kifejezés értéke. Ezért írhatjuk a következő parancsot:

```
y <- x <- 10
x; y
#> [1] 10
#> [1] 10
```

Amennyiben egy értékadás, mint kifejezés értékét szeretnénk megjeleníteni a konzolban, akkor tegyük kerek zárójelbe a teljes sort:

```
(x <- 20)
#> [1] 20
```

A kifejezés fogalmának gyakorlásához nézzünk egy példát. A másodfokú egyenlet megoldóképlete segítségével oldjuk meg az $x^2-5x+4=0$ egyenletet. Gépeljük be a következő sorokat:

```
egyutthato.a <- 1
egyutthato.b <- -5
egyutthato.c <- 4
D <- sqrt(egyutthato.b^2-4*egyutthato.a*egyutthato.c)
(-egyutthato.b+D)/(2*egyutthato.a)   # 1. gyök
#> [1] 4
(-egyutthato.b-D)/(2*egyutthato.a)   # 2. gyök
#> [1] 1
```

A fenti hat sor mindegyike egy-egy kifejezés. Az első három sorban lévő kifejezéseknek nincs outputja a konzolban, céljuk új objektumok létrehozása, és maguk a kifejezések csupán értékadó operátort, objektumnevet és konstanst tartalmaznak. A negyedik sor kifejezése szintén output nélkül hajtódik végre, és itt is új objektum jön létre, a kifejezés több összetevőt tartalmaz: objektumneveket, függvényhívást, matematikai operátorokat és konstansokat. Az ötödik és hatodik sorban lévő kifejezések értékei a kiértékelés után megjelennek az outputban, és objektumnevekből, matematikai operátorokból, kerek zárójelekből és konstansokból épülnek fel.

Összefoglalás

A függvényobjektumok (vagy röviden függvények) előre definiált utasítások sorozatát hajtják végre, és egy visszatérési értéket szolgáltatnak. A visszatérési érték meghatározását a függvény bemenő paraméterei, azaz az argumentumok is befolyásolják. Minden argumentumnak van neve, és rendelkezhetnek alapértelmezett értékkel is. Az R-rel való munka nem más, mint kifejezések létrehozása és végrehajtása, vagyis kiértékelése. A kifejezés fogalma: egy konstans, egy objektum vagy egy függvényhívás önmagában kifejezés, de ezek operátorokkal és kerek zárójelekkel helyesen összefűzött sorozata is kifejezés. A kifejezések kiértékelése során az eredmény megjelenhet a konzolban, de látható output nélkül is végbemehet a kifejezés végrehajtása.

🎯 Feladatok

1. Tekintsük át az 5.2. táblázat utolsó oszlopában szereplő R függvényeket. Próbáljuk megjósolni a függvények visszatérési értékét. Végezzünk ellenőrzést: gépeljük be, és hajtsuk végre a matematikai függvényeket! Egészítsük ki a begépelt matematikai függvényeket az argumentumok nevével, mindegyik argumentumnak adjunk nevet az 5.2. táblázat első oszlopa alapján!
2. Az előző feladatban a matematikai függvények gépelése során milyen *RStudio* kényelmi funkciókat fedeztünk fel. Soroljunk fel legalább hármat!
3. Az aranymetszés arányait tartalmazó épületek, képzőművészeti alkotások máig nagy esztétikai értékkel bírnak. Határozzuk meg ezt az arányt a $\phi = \frac{1+\sqrt{5}}{2}$ képlet segítségével! Egy A/4-es oldalra kb. 47 sort írhatunk 12-es betűmérettel, és kb. 35 sort 16-os betűmérettel. Egy üres lap hányadik sorába írnánk címet 12-es és 16-os betűméret esetén, ha esztétikailag harmonikusan szeretnénk elhelyezni? Próbáljuk ki mindezt egy szövegszerkesztőben is!
4. A trigonometrikus függvények argumentumában radiánban kell megadni a szög értékét, és nem fokban. Ezt figyelembe véve határozzuk meg a 0, 30, 45, 60, 90 és 180 fok szinuszát, koszinuszát és tangensét!

5.3. Adatszerkezetek 🙂

ℹ Miről lesz szó? Ebben a fejezetben

- áttekintjük a numerikus, karakteres és logikai konstansok írását,
- a vektor, mátrix, faktor, lista, és adattábla adatszerkezeteket,
- ezek kezelését, indexelését, tesztelését és konvertálását.

Kezdünk egyre mélyebbre ásni az R nyelvben. Megismertük már az adatobjektum, függvény és kifejezés fogalmát. Ezek birtokában már bátran belevághatunk könyvünk kulcsfontosságú fejezetébe, az adatszerkezetek tanulmányozásába. Legyünk alaposak az itt szereplő témakörök áttekintésében, és lehetőleg oldjunk meg minden kitűzött feladatot. Később ez sokszorosan megtérül.

Minden statisztikai programcsomag adatokkal dolgozik. Az R-ben nevekkel ellátott objektumokban tároljuk ezeket az adatokat. Lényegében minden tevékenység ezen objektumok létrehozása, módosítása és lekérdezése köré csoportosítható. Ezeket a műveleteket az R-ben az operátorok és a függvények végzik. Láttuk, adatokból (objektumokból), operátorokból és függvényekből kifejezéseket építünk, és hajtunk végre – így foglalható össze minden egyes tevékenység az R-ben.

Ebben a fejezetben a kifejezések adat részére összpontosítunk, hiszen minden adatelemzési munka kiinduló pontja maga az adat. Eddig csak számszerű (numerikus) adatokkal találkoztunk, és azok közül is csak az egész számok leírására fókuszáltunk. Adatfeldolgozási folyamatainkban a mért adatok azonban a numerikus mellett karakteres formában is előfordulnak, valamint az R-ben egy harmadik adattípus, a logikai is fontos szerepet kap. Összefoglalva, három R alaptípus lesz fontos számunkra az adatfeldolgozás során:

- *numerikus* típus, amely lehet *double* vagy *integer*, attól függően, hogy tizedes törteket vagy egész számokat szeretnénk tárolni,
- *karakteres* típus, amelyek nem egyetlen karaktert, hanem egy karaktersorozatot vagy más néven sztringet jelentenek,
- *logikai* típus, amely az adatszerkezetek manipulációja során jut nagyon fontos szerephez, és alapvetően igaz/hamis értékek tárolására szolgál.

A továbbiakban megismerjük, hogyan adhatjuk meg az R számára a fenti típusokba tartozó értékeket, illetve ezek felhasználásával, hogyan tudunk bonyolultabb adatszerkezeteket, összetett típusokat létrehozni.

5.3.1. Konstansok

Mért adatokat közvetlenül az R-be konstansok segítségével írhatunk be. A konstansok olyan objektumoknak is tekinthetők, amelyeknek nincs nevük, csak értékük, és azt nem is tudjuk megváltoztatni. Ha Péter 18 éves, akkor azt a `18` leírásával közölhetjük az R-rel, és ez nem is jelenthet mást (nem lehet más az értéke), mint 18. A már említett három egyszerű típusnak megfelelően tekintsük át a numerikus, karakteres és logikai konstansokat.

5.3.1.1. Numerikus konstansok

A numerikus konstansok többféle alakban is megjelenhetnek az R-ben. Az *integer* szóval az egész számok tárolását végző konstansra hivatkozunk, a *double* konstansok pedig törtrészt is tartalmazhatnak, de ez nem kötelező. Ha nem érdekes, hogy a szám *integer* vagy *double*, akkor egyszerűen a numerikus (R-ben *numeric*) elnevezést használjuk.

Az 5.3. táblázatban látható, hogy *integer* értékek írásához szükséges az `L` utótag használata, egyébként *double*-ként kezeli az R a számot, még akkor is ha nem adtunk meg törtrészt.

Fontos szabály, hogy a tizedesvessző alakja az R-ben a pont. A nulla egész részű tizedes törtek esetében az értéktelen nullát elhagyhatjuk.

5.3. táblázat: Numerikus konstansok írása *(saját szerkesztés)*

Numerikus konstans formája	Leírás
`1, -1, 2, 100, 3.5, .4`	pozitív és negatív double számok
`1L, -1L, 2L, 100L`	pozitív és negatív integer számok az 'L' utótaggal
`1.2e3, 3e+4, .6e-2, 4e1L`	exponenciális alakú double és integer számok
`0xef, 0XF01, -0xEf03, 0xd1L`	hexadecimális double és integer számok

```
0.04; .04; -.04 # utóbbi egy negatív szám, a nulla egész rész megadása nélkül
#> [1] 0.04
#> [1] 0.04
#> [1] -0.04
```

Használhatunk az R-ben exponenciális alakú és hexadecimális (16-os számrendszerbeli) számokat is.

```
12e3; 12E+3; 12e-3; 0xa2e; 0Xa2e
#> [1] 12000
#> [1] 12000
#> [1] 0.012
#> [1] 2606
#> [1] 2606
```

Az exponenciális alakú számokat `e` vagy `E` karakter vágja ketté, egy bal oldali és egy jobb oldali részre. Az exponenciális alakú szám értéke: a bal oldali rész szorozva 10 annyiadik hatványával, mint amennyi a jobb oldali rész. Érdemes időt szentelni az exponenciális alakú számok értelmezésére, mert az R outputokban gyakran előfordulnak: a szám előjelét a bal oldali rész előjele dönti el, viszont a nagyságrendjét a jobb oldali szám nagyságrendje és előjele együtt határozza meg.

Az exponenciális alakú számok nagy előnye, hogy a nagyon kis, illetve nagyon nagy számok nagyságát jobban meg tudjuk ítélni, és persze az ilyen alakú számok leírásánál helyet is megtakarítunk.

```
0.00000000000000000000000016726         # proton tömege (kg)
#> [1] 1.6726e-27
0.00000000000000000000000000091093822  # elektron tömege (kg)
#> [1] 9.109382e-31
100000000        # ennyi fele kell figyelni egy diáknak (százmillió)
```

```
#> [1] 1e+08
5970000000000000000000000                 # A Föld tömege (kg)
#> [1] 5.97e+24
```

Az R automatikusan exponenciális alakra vált túl kicsi vagy túl nagy számok konzolbeli megjelenítésénél. Ezt a viselkedést az R egyik globális opciójának beállításával tudjuk szabályozni. A globális opciókat az `options()` függvénnyel tudjuk állítani az R-ben (`?options`), amelyben most a `scipen=` paramétert kell megadnunk. Minél nagyobb pozitív értéket adunk meg, annál jobban törekszik az R a számok fix alakú megjelenítésére, negatív érték megadásánál pedig ugyanez igaz az exponenciális alakra.

```
options(scipen= 0)       # az alapértelmezés
0.0000001                # túl kicsi: exponenciális lesz
#> [1] 1e-07
123                      # marad fix alakú
#> [1] 123
100000000                # túl nagy: exponenciális lesz
#> [1] 1e+08
options(scipen=-8); 0.0000001; 123; 100000000 # exponenciális lesz mind
#> [1] 1e-07
#> [1] 1.23e+02
#> [1] 1e+08
options(scipen= 8); 0.0000001; 123; 100000000 # fix lesz mind
#> [1] 0.0000001
#> [1] 123
#> [1] 100000000
options(scipen= 0)       # az alapértelmezés visszaállítása
```

A 16-os számrendszerű számok írásához a `0-9` és a kis `a-f` vagy nagy `A-F` betűket használhatjuk fel. A hexadecimális számokat a `0x` vagy `0X` előtag vezeti be.

Aritmetikai műveleteinkben rendszerint *double* típusú számokat, 10-es számrendszerben és fix (nem exponenciális) alakban használunk. De ettől bármikor eltérhetünk:

```
12L + -3.04 + 3.4e2 + -0x1af  # számok 4 különböző formában
#> [1] -82.04
```

A számok megjelenését a konzolban még egy globális opció befolyásolja. A `digits` megszabja, hány értékes jegyre pontosan jelenjenek meg a számaink a konzolban. Lehetséges értékei az 1-22 tartományba esnek, alapértelmezés szerint 7 az értéke. A beállított érték csak egy ajánlás

az R számára, és főképp tizedes törtek esetén okozhat meglepetést, ha túl kicsire állítjuk a `digits` értékét.

```
options(digits = 1); 12.36
#> [1] 12
options(digits = 2); 12.36
#> [1] 12
options(digits = 3); 12.36
#> [1] 12.4
options(digits = 4); 12.36
#> [1] 12.36
options(digits = 7)        # alapértelmezés visszaállítása
```

Természetesen objektumokat is létrehozhatunk a numerikus értékek tárolására, ahogyan korábban már láttuk. Az objektum típusa a konstans típusával fog megegyezni:

```
peter.magassaga <- 181                                  # double objektum
peter.sulya     <- 72L                                  # integer objektum
peter.bmi       <- peter.sulya /(peter.magassaga/100)^2 # double objektum
```

5.3.1.2. Karakteres konstansok

Az R-ben a karakteres konstans (vagy más néven sztring vagy karaktersorozat) speciális karakterekkel határolt, tetszőleges karaktereket tartalmazó sorozat. A karakteres konstans tehát nem egyetlen karaktert jelent tipikusan, hanem többet. Legalább három módszerrel adhatunk meg karakteres konstanst:

```
"Látni távol kis falucska tornyát."
#> [1] "Látni távol kis falucska tornyát."
'Látni távol kis falucska tornyát.'
#> [1] "Látni távol kis falucska tornyát."
r"(Látni távol kis falucska tornyát.)"
#> [1] "Látni távol kis falucska tornyát."
```

Karakteres konstansok készítésekor a tetszőleges karaktersorozatunkat dupla (`"`) vagy egyszeres (`'`) idézőjellel kell körbevennünk, de az R 4.0.0-ás verziójától az `r"(tetszőleges_karaktersorozat)"` forma is elérhetővé vált. Láthatjuk, hogy az R a dupla idézőjelet részesíti előnyben az output megjelenítése során.

5.4. táblázat: Néhány escape szekvencia *(saját szerkesztés)*

Escape szekvencia	Jelentése
`\\t`	tabulátor
`\\r`	kocsi vissza karakter
`\\n`	új sor karakter
`\"`	dupla idézőjel
`\'`	szimpla idézőjel
`\\`	backslash karakter

Egy karakteres konstans tetszőleges karaktert (betűt, számjegyet, írásjeleket, szóközt stb.) tartalmazhat, de az első két megadási forma esetében azt a határolójelet el kell kerülnünk, amelyet az illető karakteres konstans létrehozásánál használtunk. Látjuk, hogy az `r"(tetszőleges_karaktersorozat)"` forma adja a legnagyobb szabadságot, de a legtöbbször a dupla (`"`) idézőjeles formával találkozunk.

A karakteres konstansok tartalmazhatnak ún. escape szekvenciákat, olyan backslash jellel (\, fordított perjel) kezdődő karaktersorozatokat, amelyeket speciálisan értelmez az R. A legfontosabb escape szekvenciákat és jelentésüket az 5.4. táblázat tartalmazza.

Természetesen, karakteres objektumokat is létrehozhatunk.

```
nev <- 'Zsolt'; foglalkozas <- "festő"; lakohely <- r"(Érd)"
nev; foglalkozas; lakohely
#> [1] "Zsolt"
#> [1] "festő"
#> [1] "Érd"
```

Karakteres operátor az R-ben nincs, de számos karakterkezelő függvény segíti a sztringek kezelését (5.5. táblázat).

5.3.1.3. Logikai konstansok

Az eddigiekben megismert numerikus és karakteres konstansok nagyon sokfélék lehetnek, de ugyanígy a numerikus és karakteres objektumokhoz nagyon sok lehetséges numerikus és karakteres érték rendelhető. A logikai adattípus ezektől lényegesen egyszerűbb típus, mivel itt összesen két érték tárolására van módunk. Ez a logikai *igaz* és a logikai *hamis* érték, amelyek az R nyelvben a `TRUE` és a `FALSE` logikai értékeket jelentik. Az R a logikai értékek

5.5. táblázat: Néhány karakterkezelő függvény *(saját szerkesztés)*

Függvény	Leírás	Példa
`paste();paste0(sep="")`	sztringek összefűzése	`paste("a","b",sep="=")`
`nchar(x)`	karakterszrting hossza	`nchar("alma")`
`substr(x,start,stop)`	sztring egy része	`substr("alma", 3, 5)`
`tolower(x)`	kisbetűsre konvertál	`tolower("Kiss Géza")`
`toupper(x)`	nagybetűsre konvertál	`toupper("Kiss Géza")`
`chartr(old,new,x)`	karakterek cseréje	`chartr("it","ál","titik")`
`cat(sep=" ")`	kiíratás	`cat("alma","fa\n",sep="")`
`grep();grepl();regexpr()`	részsztringek keresése	`grepl(pattern="lm",x="alma")`
`sub();gsub()`	részsztringek cseréje	`gsub("lm",repl="nyj",x="alma")`

5.6. táblázat: Relációs operátorok *(saját szerkesztés)*

Operátor formája	Művelet	Példa	Példa értéke
`<`	kisebb	`1<2;"alma"<"körte"`	`TRUE TRUE`
`>`	nagyobb	`3>(1+2);"abc">"ab"`	`FALSE TRUE`
`<=`	kisebb egyenlő	`1<=-.3;"él"<="elő"`	`FALSE TRUE`
`>=`	nagyobb egyenlő	`3/4>=7/9;"aki">="Ági"`	`FALSE TRUE`
`==`	egyenlő	`20==2e1;"Len"=="len"`	`TRUE FALSE`
`!=`	nem egyenlő	`exp(1)!=pi;"Len"!="len"`	`TRUE TRUE`
`%in%`	tartalmazás	`c(8, 12) %in% 1:10`	`TRUE FALSE`

írását a `T` és `F` globális változók bevezetésével segíti, ezek induló értéke a `TRUE` és a `FALSE` logikai érték.

Ezeket a logikai konstansokat értékadásban is szerepeltethetjük, így logikai objektumokat hozhatunk létre.

```
fiu <- TRUE; van.kocsija <- FALSE; hazas <- T
fiu; van.kocsija; hazas
#> [1] TRUE
#> [1] FALSE
#> [1] TRUE
```

Logikai értékeket vagy objektumokat relációs operátorok segítségével is létrehozhatunk (5.6. táblázat).

5.7. táblázat: Logikai operátorok *(saját szerkesztés)*

Operátor	Művelet	Példa	Példa értéke
`!`	logikai NEM	`!(1<2); !T; !F`	`FALSE FALSE TRUE`
`&`	logikai ÉS	`T & T; T & F; F & T; F & F`	`TRUE FALSE FALSE FALSE`
`|`	logikai VAGY	`T | T; T | F; F | T; F | F`	`TRUE TRUE TRUE FALSE`

Numerikus és karakteres adatok is lehetnek a relációs operátorok bemenő adatai. Numerikus adatok esetén a számok nagysága, karakteres adatok esetén az ábécében elfoglalt hely és a sztringek hossza (lexikografikus sorrend) alapján végzi az R az összehasonlítást. A sztringek lexikografikus összehasonlítása, magyar területi beállítások esetén, a magyar ékezetes karaktereket is helyesen kezeli.

A logikai értékkel visszatérő kifejezéseket (egyszerű) logikai kifejezéseknek nevezzük. Ezekből az egyszerű logikai kifejezésekből a logikai operátorok segítségével összetett logikai kifejezéseket hozhatunk létre (5.7. táblázat).

Összefoglalás

Az adatfeldolgozás során többnyire számokkal és szövegekkel dolgozunk. Az R a numerikus és a karakteres adatok írásának szabályait pontosan rögzíti. Numerikus konstansok írása a matematikában megszokott módon történik (például `12`, `-24`, `12e+3`, `0xabc3`), azonban fontos megjegyeznünk, hogy a tizedes törtek esetében pontot kell használnunk az egész és a törtrész elválasztására (például `12.34`, `-0.04`, `3.12e+12`). Karakteres konstansok esetében a következő formákat használhatjuk: `"tetszőleges karakterek"`, `'tetszőleges karakterek'`, és `r"(tetszőleges karakterek)"`. A logikai konstansok az adatmanipuláció során nyújtanak segítséget, két lehetséges értékük a logikai igaz és hamis: a `TRUE`, `FALSE` vagy rövidebben a `T`, `F`.

Feladatok

1. Mi a hasonlóság a következő három numerikus konstans között: `0xabc`, `2748`, `.2748e4`.
2. Az R öt előre definiált konstanssal rendelkezik (`?Constants`). Írassuk ki ezek értékeit, állapítsuk meg típusukat!
3. Az aranymetszés arányszámát ($\phi = \frac{1+\sqrt{5}}{2}$) írassuk a konzolba legalább 8 tizedes pontossággal!
4. Az `r"(tetszőleges karakterek)"` formájú karakteres konstans megadásnak több változata is létezik, soroljunk fel még legalább öt lehetőséget (`?Quotes`)! Milyen

előnyökkel rendelkezik ez a megadási forma az idézőjelek és a fordított perjel tekintetében?
5. Helyezzünk el idézőjeleket karakteres konstansokban, mindhárom megadási forma mellett!
6. Próbáljuk ki az 5.5. táblázat karakterkezelő függvényeit! Gépeljük be az utolsó oszlopban lévő példákat, és vizsgáljuk meg a függvények visszatérési értékét.
7. Próbáljuk ki az 5.6. táblázat relációs operátorait! Gépeljük be a példákat és ellenőrizzük az eredményeket.
8. A logikai operátorok működéséről teljes képet kaphatunk az 5.7. táblázatból. Próbáljuk ki ezeket a parancsokat is!

5.3.2. Adatszerkezetek áttekintése

Az előző fejezetben láttuk, hogy az R-ben leírható értékek alapvetően 4 típusba sorolhatók. Ezek a *double*, az *integer*, a *karakteres* és a *logikai* alaptípusok. Ezen értékek felhasználásával nagyon egyszerűen tudunk objektumokat létrehozni. Ezek az objektumok, mindjárt látjuk, az R legalapvetőbb adatszerkezetének, a *vektor*nak az egyelemű változatai.

```
obj.double     <- 12.03
obj.integer    <- 12L
obj.karakteres <- "Péter"
obj.logikai    <- TRUE
```

A fenti objektumok típusa rendre *double*, *integer*, *karakteres* és *logikai*. Ezt könnyen ellenőrizhetjük a `typeof()` vagy `class()` függvényekkel. A `typeof()` az objektum alaptípusát adja meg, a `class()` pedig inkább az R objektum-orientált lehetőségeihez kapcsolódó függvény, de a fenti objektumok esetében nagyon hasonló eredményt szolgáltat, és a későbbiek során is sokat fogjuk használni. Egyedül a *double* objektumokok esetén tér el a visszatérési értéke, `class()` ugyanis ekkor a `numeric` outputot adja.

```
typeof(obj.double);     class(obj.double)
#> [1] "double"
#> [1] "numeric"
typeof(obj.integer);    class(obj.integer)
#> [1] "integer"
#> [1] "integer"
typeof(obj.karakteres); class(obj.karakteres)
#> [1] "character"
```

```
#> [1] "character"
typeof(obj.logikai);    class(obj.logikai)
#> [1] "logical"
#> [1] "logical"
```

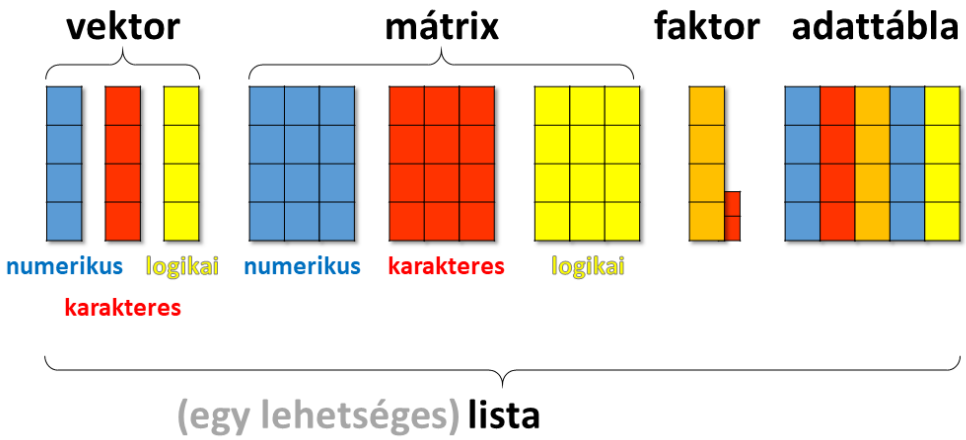
Az adatelemzési problémáink megoldásához egyszerre több adatérték feldolgozására van szükséges. Mivel az R nyelvet statisztikai adatfeldolgozásra tervezték, így nem csodálkozhatunk azon, ha több értéket is el tudunk tárolni egymás utáni memóriahelyeken a fenti 4 alaptípusból (*double*, *integer*, *karakteres* és *logikai*). Ezt többféleképp megtehetjük, például egy vagy több dimenzió mentén, illetve keverhetjük a típusokat vagy ragaszkodhatunk az azonos típusba tartozó értékek egymásutánjához. Ennek megfelelően több különböző R adatszerkezettel kell számolnunk. Ebben a fejezetben az R leggyakrabban használt adatszerkezetit tekintjük át. Most felsoroljuk és jellemezzük őket:

- *vektor* - azonos alaptípusú értékeket egymás után sorolunk fel, egy dimenzió mentén.
- *mátrix* - azonos alaptípusú értékekből egy kétdimenziós szerkezetet hozunk létre, amelynek vannak sorai és oszlopai.
- *faktor* - integer értékeket egymás után teszünk, egy dimenzió mentén, de megadjuk, hogy melyik szám milyen címkét jelöl.
- *lista* - tetszőleges típusú objektumokat egymás után sorolunk fel, egy dimenzió mentén.
- *adattábla* - tetszőleges típusú, de azonos elemszámú objektumokat egymás után sorolunk fel. Tipikusan azonos hosszúságú vektorokat vagy faktorokat teszünk egymás mellé, és így egy kétdimenziós szerkezetet kapunk, amelynek vannak sorai és oszlopai.

Az 5.1. ábra összefoglalja az adatszerkezetek fenti tulajdonságait. Beszélünk numerikus (*double* vagy *integer*), karakteres és logikai vektorokról, melyek egydimenziósak és homogének, azaz azonos típusú adatokat tartalmaznak. Ugyanez igaz a mátrixokra, csak két dimenzióban, sorokkal és oszlopokkal. A faktor egy integer vektor (azaz egydimenziós és homogén), azonban külön nyilvántartást vezet arról, hogy az egyes integer értékeknek milyen címke felel meg. Az adattábla lesz a legfontosabb adatszerkezet számunkra: kétdimenziós, de oszlopai homogének, hiszen ezek vektorok (numerikus, karakteres vagy logikai) vagy faktorok lehetnek. A lista a legszabadabb adatszerkezet, egydimenziós, de elemei bármilyen adatszerkezethez tartozhatnak. Például az 5.1. ábrán egy 8 elemű lista jelenik meg, amelynek első eleme egy numerikus vektor, utolsó eleme pedig egy adattábla.

Az 5.8. táblázatban más szempontból mutatjuk be az adatszerkezeteket: példát mutatunk adott típusú (adatszerkezetű) objektumok létrehozására, és közöljük, hogy a `typeof()` és a `class()` milyen outputot szolgáltat az így létrehozott objektumok esetében.

A következő alfejezetekben részletesen áttekintjük a *vektor*, a *mátrix*, a *faktor*, a *lista* és az *adattábla* adatszerkezeteket, ugyanis ezek töltik be a legfontosabb szerepet az adatelemzések során. Mindegyik esetben megvizsgáljuk:



5.1. ábra: Az R legfontosabb adatszerkezetei *(saját szerkesztés)*

5.8. táblázat: Adatszerkezetek *(saját szerkesztés)*

Adatszerkezet	Létrehozó parancs	typeof(x)	class(x)
double vektor	`c(12, 14)`	`double`	`numeric`
integer vektor	`c(12L, 14L)`	`integer`	`integer`
karakteres vektor	`c('a','az','egy')`	`character`	`character`
logikai vektor	`c(T, TRUE,FALSE,F)`	`logical`	`logical`
double mátrix	`matrix(1.3,nrow=2,ncol=3)`	`double`	`matrix array`
integer mátrix	`matrix(1L,nrow=2,ncol=3)`	`integer`	`matrix array`
karakteres mátrix	`matrix('az',nrow=2,ncol=3)`	`character`	`matrix array`
logikai mátrix	`matrix(F,nrow=2,ncol=3)`	`logical`	`matrix array`
faktor	`factor(c('D','D','ND'))`	`integer`	`factor`
lista	`list(A='Pék',B=1:2)`	`list`	`list`
adattábla	`data.frame(id=c('a','b'), pont=c(4,9))`	`list`	`data.frame`

- hogyan hozhatjuk létre az adott adatszerkezetű objektumot,
- hogyan tesztelhetjük, hogy az adott típusú objektumról van-e szó,
- hogyan konvertálhatunk más adatszerkezetekből ilyen típusú objektumot,
- milyen műveletekben vehet részt,
- hogyan érhetjük el az objektum részeit, azaz hogyan indexelhetjük az objektumokat.

Összefoglalás

A különböző típusú konstansokat objektumok létrehozására használhatjuk fel. A statisztikában egy objektumok értéke több konstans egymásutánja. A legegyszerűbb adatszerkezet az R-ben a *vektor*, amelyben tetszőlegesen sok, azonos típusú értéket helyezhetünk el egy dimenzió mentén. A *faktor* és a *lista* is egydimenziós, míg a *mátrix* és az *adattábla* kétdimenziós. A *faktor* integer vektor, amelyben a számoknak címkéket feleltetünk meg. A *lista* elemei tetszőleges típusúak lehetnek. A *mátrix* ugyanúgy homogén, mint a *vektor* és a *faktor*. Az *adattábla* felfogható azonos elemszámú vektorok/faktorok listájának.

Feladatok

1. Próbáljuk ki az 5.8. táblázatban szereplő példákat. Hozzuk létre a különböző típusú objektumokat és vizsgáljuk meg a `typeof()` és `class()` függvényekkel az objektumok típusát.
2. Könyvünkben az elemi adattípusok között a *double*, *integer*, *karakteres* és *logikai* típusokat tárgyaljuk. Az R nyelvben azonban léteznek más típusú objektumok is. Melyek ezek?

5.3.3. Vektor

Az R legalapvetőbb adatszerkezete a *vektor*. A vektort egymás melletti (vagy alatti) cellákban tárolt értékek sorozataként képzelhetjük el (5.1. ábra), mely értékek mindegyike azonos típusú. Így azt mondhatjuk, hogy a vektor azonos típusú (egynemű, homogén) adatok egydimenziós együttese. A vektor fontos jellemzője, hogy homogén, tehát a vektort alkotó értékek vagy kizárólag *integer*, vagy kizárólag *double*, vagy kizárólag *karakteres*, vagy kizárólag *logikai* típusúak lehetnek.

5.3.3.1. Vektor létrehozása

Vektort legegyszerűbben a `c()` függvénnyel hozhatunk létre, az argumentumlistában egymás után felsoroljuk a vektort alkotó értékeket. *Double* vektort hozhatunk létre például, ha a paraméterben numerikus konstansokat sorolunk fel:

```
v.d <- c(2, 4, 6, 8); v.d  # numerikus (double) vektor létrehozása
#> [1] 2 4 6 8
```

A `v.d` objektum egy 4 elemű *double* vektor. Az első eleme a 2, a második eleme a 4, a harmadik a 6 és a negyedik egyben utolsó eleme a 8. A vektor elemei szóközökkel elválasztva jelennek meg a konzolban.

Karakteres vektort hasonlóan hozhatunk létre, a `v.k` vektor 3 elemű lesz.

```
v.k <- c("erős", 'közepes', "gyenge"); v.k # karakteres vektor létrehozása
#> [1] "erős"    "közepes" "gyenge"
```

Egy logikai vektor csak logikai konstansokat tartalmazhat (`TRUE` vagy `FALSE`, illetve a `T` és `F` rövidebb változatot is használhatjuk):

```
v.l <- c(TRUE, FALSE, T); v.l  # logikai vektor létrehozása
#> [1]  TRUE FALSE  TRUE
```

A `v.d`, `v.k` és `v.l` objektum egy-egy példa az R különböző típusú vektoraira. Az objektumok fontos jellemzője az objektum hossza, ami vektorok esetén a vektort alkotó elemek számát jelenti. Ezt a `length()` függvénnyel kérdezhetjük le.

```
length(v.d); length(v.k); length(v.l)  # vektor hossza
#> [1] 4
#> [1] 3
#> [1] 3
```

A vektor hosszát létrehozása után is módosíthatjuk, szintén a `length()` függvényt használjuk, de az értékadás bal oldalán.

```
length(v.l) <- 5  # vektor hosszának módosítása
```

A `v.l` logikai vektor most már 5 elemű lesz:

```
v.l
#> [1]  TRUE FALSE  TRUE    NA    NA
```

Mivel nem adtuk meg a 4. és 5. elemét, így az `NA` lesz, ami a *hiányzó érték* jele az R-ben. Az `NA` minden vektornak eleme lehet, a vektor típusától függetlenül.

```
v.i <- c(12L, NA, 15L)  # 3 elemű integer vektor; a 2. eleme nem ismert
```

Térjünk vissza a vektorok létrehozásához. A `c()` függvény paraméterébe természetesen konstansok helyett tetszőleges kifejezéseket is írhatunk:

```
szamok <- c(1, (2+3)*4, 1/4, .5^3);        szamok
#> [1]  1.000 20.000  0.250  0.125
nevek  <- c("Péter", paste0('Zso', "lt")); nevek
#> [1] "Péter" "Zsolt"
iteletek <- c(T, 1<2, 2==3);               iteletek
#> [1]  TRUE  TRUE FALSE
```

A vektorok esetében a homogenitás központi szerepet játszik. Az R abban az esetben sem fog különböző típusú elemekből vektort létrehozni, ha ezeket egyetlen `c()` függvényhívásban szerepeltetjük. Ekkor automatikus típuskonverzió történik. Nézzük ezeknek az eseteit:

```
eset.1 <- c(2,4,"6",8);    eset.1
#> [1] "2" "4" "6" "8"
eset.2 <- c(T, FALSE,"6"); eset.2
#> [1] "TRUE"  "FALSE" "6"
eset.3 <- c(T, FALSE, 3);  eset.3
#> [1] 1 0 3
```

Amennyiben karakteres konstans szerepel az elemek között, a vektor karakteres típusú lesz. Ha numerikus és logikai értéket sorolunk fel, akkor a vektor numerikus lesz, azzal a kiegészítéssel, hogy a `TRUE` logikai érték 1-re, a `FALSE` pedig 0-ra konvertálódik.

További lehetőség a `c()` függvény használata során, hogy a paraméterben vektort is szerepeltethetünk. Ekkor ezek az elemek is szerepelni fognak az eredményvektorban:

```
regi.v.1 <- c(1, 2, 3)
regi.v.2 <- c(7, 8, 9)
uj.v <- c(0, regi.v.1, 4, 5, 6, regi.v.2, 10, c(11, 12)); uj.v
#>  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12
```

A fenti példában létrehozott `uj.v` 13 elemű numerikus vektor összerakásához felhasználtunk két 3 elemű vektort és egy kételemű vektort is.

Vektorok létrehozása során még egy érdekes lehetőségről érdemes szót ejteni. A `c()` függvényben a vektor egyes elemeit elnevezhetjük, és ezek a nevek az outputban is meg fognak

jelenni. Az elemek elnevezéséhez írjunk egy nevet és egy egyenlőségjelet az argumentumként használt elem elé. Ha a név nem egyetlen szó (vagyis tartalmaz szóközt), akkor a karakterkonstansok megadásánál látott három módszer valamelyikét használhatjuk (tehát a dupla és szimpla idézőjeleket és az `r"()"` konstrukciót), vagy a backtick (‘) szimbólumot. Ezzel a módszerrel például a naponta tanulással töltött időnket úgy rögzíthetjük, hogy az output “beszédesebb” lesz, több információt tartalmaz.

```
tan.ido <- c(Hétfő=35, Kedd=95); tan.ido
#> Hétfő  Kedd
#>    35    95
tan.ido <- c(Hétfő=35, "Kedd délelőtt"=50, `Kedd délután`=45); tan.ido
#>         Hétfő Kedd délelőtt  Kedd délután
#>            35            50            45
```

A vektorelemek nevei lekérdezhetők a `names()` függvénnyel. Amennyiben az értékadás bal oldalán szerepeltetjük, a vektor elemneveit tudjuk módosítani.

```
names(tan.ido)                          # elemnevek lekérdezése
#> [1] "Hétfő"         "Kedd délelőtt" "Kedd délután"
names(tan.ido) <- c("H", "K.1", "K.2") # elemnevek módosítása
tan.ido
#>   H K.1 K.2
#>  35  50  45
```

5.3.3.2. Szabályos vektorok létrehozása

Ha egy vektor elemei szabályos rendben követik egymást, akkor szabályos vektorokról beszélünk. Ilyen lehet például a következő három numerikus vektor és két karakteres vektor.

```
c(1, 2, 3, 4, 5); c(1, 3, 5, 7); c(1, 1, 1, 2, 2, 2)
c("férfi", "nő", "férfi", "nő"); c("f.1", "f.2", "f.3")
```

Szabályos numerikus vektorokat hozhatunk létre a kettőspont (`:`) operátorral vagy a `seq()` függvénnyel. Az így létrehozott vektorok ugyanis valamilyen számtani sorozat egymást követő elemei, vagyis az egymás mellett lévő elemek különbsége (a lépésköz) állandó.

5.3.3.3. A kettőspont operátor

A legegyszerűbb vektorlétrehozási mód a kettőspont (`:`) operátor, ahol az egymást követő elemek távolsága 1 vagy -1. Általános alakja: `start:stop`.

```
1:10    # a lépésköz +1, növekvő sorozat
#>  [1]  1  2  3  4  5  6  7  8  9 10
10:1    # a lépésköz −1, csökkenő sorozat
#>  [1] 10  9  8  7  6  5  4  3  2  1
-1.5:5  # a lépésköz +1, növekvő sorozat
#> [1] -1.5 -0.5  0.5  1.5  2.5  3.5  4.5
10.5:3  # a lépésköz −1, csökkenő sorozat
#> [1] 10.5  9.5  8.5  7.5  6.5  5.5  4.5  3.5
```

Látható, hogy az így létrehozott vektorok lehetnek csökkenő vagy növekvő rendezettségűek, valamint tört értékeket is használhatunk operandusként. A sorozat nem feltétlenül a kettőspont utáni értékig tart, mindössze annyi igaz, hogy a sorozat vége a `stop` értéknél mindig kisebb egyenlő (vagy nagyobb egyenlő, csökkenő sorozat esetén).

Hosszabb numerikus vektorokat is könnyűszerrel létrehozhatunk. A `101:140` parancs hatására 40 elemet hozunk létre. Hosszabb vektorok outputjában könnyebben el tudunk igazodni a sorok elején lévő `[x]` konstrukció segítségével: minden sorban a sor első eleme a vektor `x`. eleme. A lenti outputban szereplő `[17]` például azt mutatja, hogy a sor elején lévő 117 a 40 elemű vektor 17. eleme.

```
101:140  # a lépésköz +1, növekvő sorozat
#>  [1] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
#> [17] 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132
#> [33] 133 134 135 136 137 138 139 140
```

5.3.3.4. A `seq()` függvény

A `seq()` függvény nagyobb szabadságot ad a numerikus sorozatok generálására. Legegyszerűbb használata esetén a kettőspont (`:`) operátort kapjuk vissza:

```
seq(1, 10)  # a lépésköz +1, növekvő sorozat
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

A `seq()` függvény használatához négy argumentum nevét és jelentését kell megtanulnunk: a `from=` a sorozat első elemét határozza meg, a `to=` az utolsó elemet, a `by=` a lépésközt és a `length.out=` a létrehozandó vektor elemeinek a számát. A négy paraméterből három megadása már egyértelműen azonosítja a kívánt vektort:

```
seq(from=1, to=10, by=2)          # a lépésköz +2, növekvő sorozat
#> [1] 1 3 5 7 9
seq(from=1, to=10, length.out=5)   # a lépésköz +2.25, növekvő sorozat
#> [1]  1.00  3.25  5.50  7.75 10.00
seq(to=10, by=-1.3, length.out=5)  # a lépésköz -1.3, csökkenő sorozat
#> [1] 15.2 13.9 12.6 11.3 10.0
seq(from=1, by=1.3, length.out=5)  # a lépésköz +1.3, növekvő sorozat
#> [1] 1.0 2.3 3.6 4.9 6.2
```

A `seq_along()` függvénnyel szintén tudunk 1-től induló, +1-es lépésközű sorozatot alkotni, amelynek utolsó értéke, a paraméterben megadott vektor elemszáma.

```
x <- c("Hétfő", "Kedd", "Szerda"); y <- 11:20
seq_along(x) # numerikus vektor 1-től, +1-es lépésközzel, 3 elemű
#> [1] 1 2 3
seq_along(y) # numerikus vektor 1-től, +1-es lépésközzel, 10 elemű
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

5.3.3.5. A `rep()` függvény

Tetszőleges típusú vektor létrehozására használhatjuk a `rep()` függvényt, amely egy létező vektor értékeit ismétli meg. A `rep()` első paramétere az ismétlendő vektor, a `times=` pedig az ismétlések számát adja meg.

```
rep(2, times=3)            # számot ismétlünk 3-szor
#> [1] 2 2 2
rep(c(2, 0, -2), times=3)  # numerikus vektort ismétlünk 3-szor
#> [1]  2  0 -2  2  0 -2  2  0 -2
rep("nap", times=3)        # sztringet ismétlünk 3-szor
#> [1] "nap" "nap" "nap"
rep(c(F, T, T), times=3)   # logikai vektort ismétlünk 3-szor
#> [1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
```

A fenti példában mindenhol háromszor ismételtük meg az első paramétert, méghozzá úgy, hogy az R egymás után sorolta fel őket.

Egy vektor ismétlésének van egy másik esete is, amikor az elemeit sorban egyenként véve végezzük el az ismétlést (helyben ismétlés). Ekkor nem a `times=` paramétert, hanem az `each=` argumentumot kell használnunk a függvény hívásánál.

```
rep(2, each = 3)  # számot ismétlünk 3-szor
#> [1] 2 2 2
rep(c(2, 0, -2), each = 3)  # numerikus vektort elemeit ismételjük 3-szor
#> [1]  2  2  2  0  0  0 -2 -2 -2
rep("nap", each = 3)  # sztringet ismétlünk 3-szor
#> [1] "nap" "nap" "nap"
rep(c(F, T, T), each = 3)  # logikai vektor elemeit ismételjük 3-szor
#> [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Látjuk, hogy egyelemű vektorok ismétlése esetén nincs különbség a `times=` és az `each=` paraméterek használata között.

Utolsó esetként elemenként szeretnénk ismételni, de eltérő ismétlésszámmal. Ekkor a `times=` paraméterben a bemenő vektor elemszámával azonos hosszú vektort kell megadni. Ez a vektor tartalmazza az elemek ismétlés számát.

```
rep(c(2, 3, 4), times = c(1, 2, 3))  # numerikus vektort elemeit ismételjük
#> [1] 2 3 3 4 4 4
rep(c("nap", "part"), times = c(2, 3))  # karakteres vektort elemeit ismételjük
#> [1] "nap"  "nap"  "part" "part" "part"
rep(c(T, F, T), times = c(2, 3, 4))  # logikai vektort elemeit ismételjük
#> [1]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

Végezetül bemutatjuk, hogy az `each=` és az egyelemű értékkel rendelkező `times=` egyszerre is alkalmazható. Ekkor először a helyben ismétlés (`each=`), majd az így kapott vektor teljes ismétlése következik (`times=`).

```
rep(1:5, each = 2, times = 3)  # kombinált ismétlés
#>  [1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5
```

5.3.3.6. A `paste()` függvény

Szabályos karakteres vektor létrehozására használhatjuk a `paste()` függvényt. Egy előtaghoz (például `f`) hozzáfűzhetünk 10 különböző számot, amely így egy 10 elemű karakteres vektort eredményez.

```
paste("f", 1:10) # 10 elemű sztring vektor
#>  [1] "f 1"  "f 2"  "f 3"  "f 4"  "f 5"  "f 6"  "f 7"  "f 8"  "f 9"
#> [10] "f 10"
```

Láthatjuk, hogy az `f` karakter és a számok közé egy szóköz került, de ezt a `sep=` argumentummal megváltoztathatjuk:

```
paste("f", 1:10, sep="-") # gondolatjel az elválasztó
#>  [1] "f-1"  "f-2"  "f-3"  "f-4"  "f-5"  "f-6"  "f-7"  "f-8"  "f-9"
#> [10] "f-10"
paste("f", 1:10, sep="")  # nincs elválasztó
#>  [1] "f1"  "f2"  "f3"  "f4"  "f5"  "f6"  "f7"  "f8"  "f9"  "f10"
```

A `collapse=` argumentum használatával, akár egyetlen karakteres értékbe is összeolvaszthatjuk a fenti elemeket. Az argumentumban az összevonásnál használt elválasztó karaktert adjuk meg.

```
paste("f", 1:10, sep="-", collapse="_") # gondolatjel az elválasztó, egy sztring
#> [1] "f-1_f-2_f-3_f-4_f-5_f-6_f-7_f-8_f-9_f-10"
```

Az eddigiek összefoglalásaként nézzünk példát különböző típusú és elemhosszú vektorok létrehozására.

```
y <- 12L                        # 1 elemű integer vektor
y <- 12                         # 1 elemű double vektor
y <- "Bízz magadban!"           # 1 elemű karakteres vektor
y <- TRUE                       # 1 elemű logikai vektor
y <- c(23.8, -5)                # 2 elemű double vektor
y <- c("H", "K")                # 2 elemű karakteres vektor
y <- c(T, FALSE)                # 2 elemű logikai vektor
y <- c(1, 2, 3, 4, 5)           # 5 elemű double vektor
y <- 1:5                        # 5 elemű integer vektor
y <- seq(from=9, to=100, by=2)  # 46 elemű double vektor
y <- rep(c("H", "K"), times=10) # 20 elemű karakteres vektor
z <- seq_along(y)               # 20 elemű integer vektor
y <- paste("év", 2001:2020)     # 20 elemű karakteres vektor
```

5.3.3.7. A vektoraritmetika szabályai

Amint az előzőekben láttuk, az R rendszer legalapvetőbb adattárolási szerkezete a vektor. Az R egyik legnagyszerűbb tulajdonsága pedig az, ahogyan a vektorokkal műveleteket végezhetünk. Korábban már láttuk, hogyan tudunk összeadni két számot az R-ben. Próbáljunk meg összeadni két 2 elemű vektort:

```
c(1, 2) + c(3, 4) # két vektor összeadása
#> [1] 4 6
```

A két fenti vektort a parancssorban hoztuk létre a `c()` függvénnyel. Az összeadás eredménye egy 2 elemű vektor. Az eredményvektor az `1+3` és a `2+4` műveletek alapján jött létre, vagyis az összeadás operandusaiban szereplő vektor azonos sorszámú elemeire hajtotta végre a kijelölt műveletet az R.

Két vektor összeadásánál természetesen használhatunk objektumneveket is:

```
x <- c(1, 2, 3); y <- c(2, 3, 4)
x + y # két vektor összeadása
#> [1] 3 5 7
```

Itt az eredményvektor 3 elemű, és a komponensenkénti művelet végrehajtás szabályainak megfelelően az `1+2`, `2+3` és a `3+4` összeadások eredménye lesz a 3 új elem.

Az összeadás műveletet tetszőleges operátorral felcserélhetjük, használhatjuk az összes aritmetikai, relációs és logikai operátort.

```
c(1,2) - c(2,3) # két vektor összeadása
#> [1] -1 -1
x <- c(1, 2, 3); y <- c(2, 3, 4)
x - y           # két vektor különbsége
#> [1] -1 -1 -1
x * y           # két vektor szorzata
#> [1]  2  6 12
x / y           # két vektor hányadosa
#> [1] 0.5000000 0.6666667 0.7500000
x ^ y           # x az y-adikon
#> [1]  1  8 81
x == y          # x egyenlő y-nal?
#> [1] FALSE FALSE FALSE
x < y           # x kisebb, mint y?
#> [1] TRUE TRUE TRUE
```

A fenti műveletek közül a hatványozás végrehajtása tűnhet kicsit szokatlannak, itt ugyanis egy 3 elemű vektort, mint alapot egy 3 elemű másik vektorra, mint kitevőre emeljük. Ha azonban a komponensenkénti végrehajtás szabályát észben tartjuk, akkor világos, hogy az eredményvektor az `1^2`, `2^3` és a `3^4` eredménye.

A komponensenkénti végrehajtás szabálya a logikai operátorokra is érvényes.

```
!c(T, T, F, F)                # logikai NEM egy vektorra
#> [1] FALSE FALSE  TRUE  TRUE
c(T, T, F, F) & c(T, F, T, F)  # logikai ÉS két vektorral
#> [1]  TRUE FALSE FALSE FALSE
c(T, T, F, F) | c(T, F, T, F)  # logikai VAGY két vektorral
#> [1]  TRUE  TRUE  TRUE FALSE
```

A vektorok közötti műveletek legegyszerűbb esetét tekintettük át eddig, azaz azonos elemszámú vektorokat adtunk össze vagy vontunk ki egymásból. Ha az operátor két oldalán lévő vektorok elemszáma eltér, akkor az általános szabály az, hogy a rövidebbik vektort az R megismétli mindaddig, míg a hosszabbik vektor elemszámát el nem éri. Ha a rövidebbik vektort nem egész számszor megismételve kapjuk a hosszabb vektor hosszát, akkor figyelmeztetést kapunk az R-től, melyben erre a tényre felhívja a figyelmünket, de a kijelölt műveletet az R ennek ellenére végrehajtja.

```
c(1, 2) + 5  # két eltérő elemszámú vektor összeadása
#> [1] 6 7
```

A fenti példában egy 2 elemű és egy 1 elemű vektort adunk össze. A rövidebb vektort még egyszer megismételve már az `c(5, 5)` vektort kapjuk, így a kijelölt összeadás minden fennakadás nélkül végrehajtható. Az eredményvektor az `1+5` és a `2+5` összeadások eredménye lesz.

Most egy 2 elemű és egy 3 elemű vektort adunk össze.

```
c(1, 2) + c(3, 4, 5)  # két eltérő elemszámú vektor összeadása
#> Warning in c(1, 2) + c(3, 4, 5) :
#>   longer object length is not a multiple of shorter object length
#> [1] 4 6 6
```

A rövidebbik vektort még egyszer megismételve a `c(1, 2, 1, 2)` vektort kapjuk, de mivel nincs szükség minden elemre, ezért figyelmeztető üzenetet kapunk. Az eredményvektor az `1+3`, `2+4` és az `1+5` összeadások eredménye lesz.
A következő példában már nincs figyelmeztetés, hiszen a rövidebb vektort egész számszor, pontosan kétszer kellett megismételni a koordinátánkénti művelet végrehajtáshoz.

```
c(1, 2) + c(3, 4, 5, 6)  # két eltérő elemszámú vektor összeadása
#> [1] 4 6 6 8
```

Foglaljuk össze a vektoraritmetika szabályait:

- azonos elemszámú vektorok között az azonos pozícióban lévő vektorelemek között hajtódik végre a kijelölt művelet (vagyis koordinátánkénti végrehajtás történik),
- különböző elemszámú vektorok esetében pedig először a rövidebbik vektor ismétléssel kiegészül a hosszabbik vektor hosszára, és ezt követi a koordinátánkénti végrehajtás.

Az operátorokon túl az 5.2. táblázatban szereplő matematikai függvények is támogatják a vektor paramétert. Ekkor nem egyetlen értékkel térnek vissza, hanem a bemenő vektor minden elemére kiszámolt függvényértékek vektorával.

```
sqrt(c(4, 9, 16))              # 3 szám négyzetgyöke
#> [1] 2 3 4
log(x=c(1, 10, 100), base=10)  # 3 szám 10-es alapú logaritmusa
#> [1] 0 1 2
x <- 1.3:10; round(x)          # 9 szám egészre kerekítve
#> [1] 1 2 3 4 5 6 7 8 9
```

5.3.3.8. Függvények vektorokkal

Az előző fejezetben láttuk, hogy a matematikai függvények vektor argumentumot is elfogadnak, és a vektor minden elemére kiszámolják a függvényértéket. Míg a `log(x=16, base=2)` függvényhívás a matematikában megszokott módon egyetlen bemenő értékhez (16) egyetlen kimenő éréket szolgáltat (4), addig az R lehetőségeit jobban kihasználó `log(x = c(1, 2, 4, 8, 16), base=2)` függvényhívás négy bemenő értékből (`c(1, 2, 4, 8, 16)`) négy kimenő értéket `c(0, 1, 2, 3, 4)` állít elő. A függvények és a vektorok kapcsolatának azonban van egy másik aspektusa, amely szorosan kötődik a statisztikai műveletek végrehajtásához.

Az R függvények egy nagy csoportja eleve olyan vektort vár az argumentumába, amely több tíz vagy több száz elemet tartalmaz, és tipikusan egyetlen értékkel tér vissza. Ezeket a függvényeket vektor alapú függvényeknek nevezzük, és ebbe a csoportba tartoznak az R statisztikai mutatókat számoló függvényei is. A vektor alapú függvényekre az jellemző, hogy a bemenő vektor elemeivel egy előre definiált műveletsorozatot hajtanak végre, például összeadják a vektor elemeit, kiszámolják az elemek átlagát vagy szórását, és visszatérési értékként ezt az összeget, átlagot vagy szórást szolgáltatják. A legfontosabb vektor alapú függvényeket az 5.9. táblázat tartalmazza.

5.9. táblázat: Függvények vektorokkal *(saját szerkesztés)*

Függvény	Leírás	Példa	Példa értéke
`max(x)`	az x vektor legnagyobb eleme	`max(1:10)`	`10`
`min(x)`	az x vektor legkisebb eleme	`min(11:20)`	`11`
`sum(x)`	x elemeinek összege	`sum(1:5)`	`15`
`prod(x)`	x elemeinek szorzata	`prod(1:5)`	`120`
`mean(x)`	x számtani közepe (mintaátlag)	`mean(1:10)`	`5.5`
`median(x)`	x mediánja	`median(1:10)`	`5.5`
`range(x)`	x legkisebb és legnagyobb eleme	`range(1:10)`	`1 10`
`sd(x)`	az x minta szórása	`sd(1:10)`	`3.03`
`var(x)`	az x minta varianciája	`var(1:10)`	`9.17`
`cor(x,y)`	korreláció x és y között	`cor(1:10,11:20)`	`1`

5.3.3.9. Típusok kezelése

Minden R vektor típusa a négy alaptípus egyike lehet: *double*, *integer*, *karakteres* vagy *logikai*. Korábban láttuk, hogy a `class()` és a `typeof()` függvények pontos tájékoztatást adnak a vektorok típusáról. Létezik azonban egy függvénycsalád, amellyel megvizsgálhatjuk, hogy egy tetszőleges objektum az adott típushoz tartozik-e. Ez az `is.*()` függvénycsalád, amelynek eleme az `is.double()`, `is.integer()`, `is.logical()` és `is.character()` függvény. Nézzünk egy példát használatukra.

```
x.d <- c(3.5, 4.1, 9.2)  # új objektum – double vektor
is.double(x.d)           # x.d vajon double
#> [1] TRUE
is.integer(x.d)          # x.d vajon integer
#> [1] FALSE
is.character(x.d)        # x.d vajon karakteres
#> [1] FALSE
is.logical(x.d)          # x.d vajon logikai
#> [1] FALSE
```

Láttuk korábban, hogy a logikai értékek esetében, ha szükséges, automatikus típuskonverzió történik numerikus típusra (`TRUE` - 1, `FALSE` - 0). Sok esetben azonban explicit típuskonverzióra van szükség, amit az `as.*()` függvénycsaláddal hajthatunk végre. Vektorok esetében használhatjuk az `as.double()`, `as.integer()`, `as.logical()` vagy `as.character()` függvényeket. Nézzünk ezekre is néhány példát.

```
as.double(c(T, F))              # logikai vektorból double
#> [1] 1 0
as.integer(c("2.9", "a", "3"))  # karakteres vektorból integer
#> [1]  2 NA  3
as.character(1:5)               # integer vektorból karakteres
#> [1] "1" "2" "3" "4" "5"
as.logical(0:3)                 # integer vektorból logikai
#> [1] FALSE  TRUE  TRUE  TRUE
```

Karakteres értékből könnyen kaphatunk számot, például a `"2.9"` vagy `"3"` esetén, viszont az `"a"` karakter esetében `NA` érték kerül az integer vektorba, ahogyan ezt a fenti példában is láthatjuk.

5.3.3.10. Az `NA` hiányzó érték

Korábbi példáinkban már felbukkant a hiányzó érték, amelyet az R-ben az `NA` jelöl. Az adatelemzési munkánkat végigkísérik a hiányzó adatok. Első lépésként azt jegyezzük meg, hogy az `NA` hiányzó érték tetszőleges típusú vektorban lehet elem.

```
x <- c(2, NA, 4); x             # NA numerikus vektorban
#> [1]  2 NA  4
x <- c(NA, "erős", "gyenge"); x  # NA karakteres vektorban
#> [1] NA       "erős"   "gyenge"
x <- c(T, NA, NA); x            # NA logikai vektorban
#> [1] TRUE   NA   NA
```

Egy `NA` érték jelenlétét a vektorban az `is.na()` függvénnyel tudjuk kimutatni. Az `is.na()` argumentuma tetszőleges vektor lehet, visszatérési értéke pedig a bemenő vektor elemszámával megegyező logikai vektor. A visszatérő logikai vektor csak abban a pozícióban tartalmaz `TRUE` értéket, ahol bemenő vektorban hiányzó adatot találunk.

```
x <- c(1, NA, 3, 4, NA)    # két NA a numerikus vektorban
is.na(x)                   # két TRUE a logikai vektorban
#> [1] FALSE  TRUE FALSE FALSE  TRUE
```

Azt is leellenőrizhetjük, hogy egy vektorban van-e egyáltalán hiányzó érték. Az `anyNA()` függvény visszatérési értéke `TRUE`, ha a bemenő vektorban legalább egy `NA` érték található, és `FALSE`, ha a bemenő vektorban egyetlen `NA` sincs.

```
anyNA(x)  # van-e NA a numerikus vektorban
#> [1] TRUE
```

Hiányzó értékeket is tartalmazó vektor esetén néhány vektor alapú függvény meglepő eredményt adhat. A statisztikai mutatókat számoló függvények rendre `NA`-val térnek vissza, ha a bemenő vektorban van hiányzó érték.

```
mean(c(2, NA, 3, 4, 2, 5))  # NA-t tartalmazó vektor átlaga NA
#> [1] NA
```

Ha kíváncsiak vagyunk az `NA` értéken kívüli elemek átlagára, akkor egy második paramétert is szerepeltetnünk kell a `mean()` függvényben, és minden más statisztikai mutatót számoló függvényben. Az `na.rm=` argumentum `TRUE` értéke biztosítja, hogy az átlag számítása során a hiányzó értékeket figyelmen kívül hagyjuk.

```
mean(c(2, NA, 3, 4, 2, 5), na.rm=T)  # NA-t tartalmazó vektor átlaga már nem NA
#> [1] 3.2
```

5.3.3.11. Az `Inf` és a `NaN`

Az R-ben a numerikus műveletek eredménye – a matematikai értelmezéstől sokszor eltérően – vezethet pozitív vagy negatív végtelen eredményre. Ezeket az `Inf` és a `-Inf` szimbólumok jelölik, amelyeket különböző kifejezésekben akár mi is felhasználhatunk.

```
1/0  # ez a matematikában nem értelmes, de R-ben Inf
#> [1] Inf
log(0)
#> [1] -Inf
exp(Inf)
#> [1] Inf
mean(c(1, 2, Inf))
#> [1] Inf
```

Néhány esetben a numerikus kifejezések eredménye nem értelmezhető számként, ezt az R-ben a `NaN` (`Not a Number`) jelöli. Ilyen kifejezések például:

```
0/0
#> [1] NaN
Inf - Inf
#> [1] NaN
Inf/Inf
#> [1] NaN
```

Egy kifejezés véges vagy végtelen voltát az `is.finite()` vagy `is.infinite()` függvényekkel tesztelhetjük. A `NaN` értékre az `is.nan()` függvénnyel kérdezhetünk rá. Figyeljük meg, a `NaN` értékre, mind az `is.nan()`, mind az `is.na()` függvény `TRUE` értéket ad.

```
x <- c(1, NA, NaN, Inf, -Inf)
is.na(x)  # melyik elem hiányzó
#> [1] FALSE  TRUE  TRUE FALSE FALSE
is.nan(x)  # melyik elem nem szám
#> [1] FALSE FALSE  TRUE FALSE FALSE
is.infinite(x)  # melyik elem végtelen
#> [1] FALSE FALSE FALSE  TRUE  TRUE
is.finite(x)  # melyik elem véges
#> [1]  TRUE FALSE FALSE FALSE FALSE
```

5.3.3.12. Vektor indexelése

Fontos részhez érkeztünk, érdemes kicsit lassítanunk. Már nagyon sok mindent megtanultunk a vektorokról: egy vektorban egy dimenzió mentén azonos típusú értékeket sorolhatunk fel, amellyel a vektoraritmetika szabályai szerint műveleteket tudunk végezni. Például hozzunk létre egy 10 elemű vektort, növeljük meg minden egyes vektorelem értékét eggyel.

```
x <- 11:20         # x integer vektor létrehozása
x + 1              # kiíratjuk az 1-gyel megnövelt értékeket (x nem változik)
#>  [1] 12 13 14 15 16 17 18 19 20 21
x                  # x értékének kiírása
#>  [1] 11 12 13 14 15 16 17 18 19 20
```

A fenti sorok hatására a konzolban egy 10 elemű vektor elemei jelennek meg, minden elem eggyel nagyobb, mint az `x` adott eleme. Egyetlen összeadás (`+`) operátor segítségével valójában 10 összeadás végrehajtását írtuk elő. Vegyük észre, hogy maga az `x` vektor nem módosult, továbbra is az eredeti `11:20` elemeket tartalmazza. Egy objektum ugyanis addig őrzi az értékét, amíg értékadó operátor segítségével felül nem írjük.

Tekintsük most a következő sorokat.

```
y <- 11:20          # y integer vektor létrehozása
y <- y + 1          # megnöveljük 1-gyel y értékeit (y megváltozik)
y                   # y értékének kiírása
#>  [1] 12 13 14 15 16 17 18 19 20 21
```

Az `y` vektor 10 elemű, a `11:20` értékekkel hoztuk létre. A második sorban azonban megváltoztatjuk az `y` értékét, mert újra az értékadás bal oldalán szerepel az `y` objektum. Az `y` új értéke az értékadás jobb oldalán szereplő kifejezés értéke lesz, azaz a `y+1` összeadás eredménye, ami nem más, mint a `12:21`. Az `y` értékének megjelenítésével ellenőrizhetjük, hogy valóban a `12:21` elemek kerülnek a konzolba.

A fenti példában `y` minden értékét megváltoztattuk. Az eredeti `11:20` helyett az új érték `12:21`. Az `y` vektor minden egyes eleme megváltozott, például ahol 11 volt, ott most 12 van, ahol 12 volt ott most 13. Ha szükség van az eredeti és az új `y` értékekre akkor kicsit módosítanunk kell az eddigi sorokon.

```
z    <- 11:20         # z integer vektor létrehozása
z.uj <- z + 1         # z.uj double vektor létrehozása (z nem változik)
z                     # z értékének kiírása
#>  [1] 11 12 13 14 15 16 17 18 19 20
z.uj                  # z.uj értékének kiírása
#>  [1] 12 13 14 15 16 17 18 19 20 21
```

A `z` vektor is 10 elemű, a `11:20` a kezdőértéke, és jól látható, hogy a fenti sorok hatására ez nem is változik meg, hiszen a `z` újra már nem jelenik meg értékadás bal oldalán. Értékadás jobb oldalán viszont felbukkan, a második sorban a `z.uj` objektum létrehozásához használtuk fel `z` értékét. Az `z` és `z.uj` objektumok értékének kiírásával ellenőrizhetjük, hogy a `z` továbbra is biztonságosan tárolja a `11.20` értékeket, de a `z.uj`-ban a kívánt `12:21` módosított értékek is megtalálhatók. A további munkafázisokban így az eredeti és a módosított értékek is elérhetők lesznek, ami újdonság, mert az előző példákban ez a lehetőség nem volt elérhető. Az `x` objektumot használó példában csak az eredeti, az `y` vektoros példában csak a módosított értékeket tudnánk a későbbiekben használni.

Összefoglalva az eddigieket, két tanulságot vonhatunk le. Egyfelől, a vektorműveleteknek csak akkor lesz "maradandó" hatása, ha objektumban őrizzük a számítás eredményét, azaz értékadást használunk. Ez az objektum lehet a kiinduláskénti használt eredeti objektum (`y <- y + 1`), de biztonságosabb ha új objektumot hozunk létre az új értékek számára (`z.uj <- z + 1`), mert így az eredeti értékeket a jövőben is tudjuk használni. Másfelől, ezek a példák ráirányítják a figyelmet a vektoraritmetika egy nagyszerű jellemzőjére: a vektorműveletek

megadása független a vektor hosszától, nem lesz bonyolultabb egy vektorművelet, például az `x+1` összeadás ha `x` nem 10 elemű, hanem mondjuk 100 hosszú. Az összeadás művelet parancsa 100 elemű vektor esetén is csupán `x+1`, azonban a háttérben nem 10, hanem 100 összeadás történik. Akár 10, akár 100 elemű az `x`, az összes elemre az `x` segítségével hivatkozhatunk, és az `x+1` összeadás az `x` összes eleméhez hozzáad 1-et.

De mit tegyünk, ha nincs szükségünk `x` összes elemére, vagy nem szeretném `x` összes elemét megnövelni 1-gyel, csak néhányat. Ekkor *indexelést* kell használnunk.

Az adatfeldolgozás során gyakori, hogy a vektor egyes elemeit külön-külön szeretnénk elérni, lekérdezni vagy módosítani. A vektor egy tetszőleges részét, egy vagy több elemét az *indexelés* művelettel érhetjük el, melynek eredménye szintén vektor lesz. Az index operátor jele a szögletes zárójel (`[]`) az R-ben, amit a vektor neve után kell írnunk. Vektorok indexelésének általános alakja:

```
vektor[indexvektor]        # az eredmény egy vektor
```

Az indexvektor lehet numerikus, karakteres és logikai vektor is. Nézzük ezeket sorban.

5.3.3.13. Indexelés numerikus vektorokkal

Kezdjük egy 10 elemű `x` vektor létrehozásával.

```
x <- 11:20
x
#>  [1] 11 12 13 14 15 16 17 18 19 20
```

Megfigyelhetjük, hogy az `x` vektor 1. eleme 11, a 2. a 12, az utolsó, a 10. pedig éppen 20. Ebben a felsorolásban az elemek sorszámai (1., 2., 10.) pontosan a vektor indexeit jelentik. A vektor indexelése tehát 1-gyel kezdődik, ez az 1. elem indexe, a 2. elem indexe 2, az utolsó elemé pedig 10. Ha az index operátorba egy ilyen egyszerű sorszámot írunk, akkor a vektor adott indexű elemét érhetjük el.

```
x[1]     # x vektor 1. eleme
#> [1] 11
x[2]     # x vektor 2. eleme
#> [1] 12
x[10]    # x vektor 10. eleme
#> [1] 20
```

Nem csak lekérdezhetjük, hanem az értékadó operátor segítségével módosíthatjuk is valamelyik elemet.

```
x[2] <- 100       # x 2. elemének módosítása
x[3] <- 2*x[2]    # x 3. elemének módosítása
x
#>  [1]  11 100 200  14  15  16  17  18  19  20
```

Itt először a második elemet 100-ra cseréljük, majd a harmadikat a második kétszeresére. A változást ellenőrizhetjük a konzolban.

Ha az `x` vektort az elemszámánál nagyobb indexszel próbáljuk elérni, akkor `NA` értéket kapunk:

```
x[11]  # x csak 10 elemű, a 11. nem létező elem
#> [1] NA
```

Vektorokat azonban nem csak egy elemű indexvektorokkal indexelhetünk, hanem két vagy több elemű numerikus vektorokat is használhatunk. Ebben az esetben az indexvektorban felsorolt sorszámoknak megfelelő indexű elemeket érhetjük el.

```
x <- 11:20
x[c(1, 3, 5)]               # x vektor 1., 3. és 5. eleme
#> [1] 11 13 15
x[c(3, 5, 3, 1)]            # x vektor 3., 5., 3. és 1. eleme
#> [1] 13 15 13 11
x[3:6]                      # x vektor 3., 4., 5. és 6. eleme
#> [1] 13 14 15 16
y <- c(3,7)
x[y]                        # x vektor 3. és 7. eleme
#> [1] 13 17
x[seq(from=2, to=10, by=2)] # x vektor páros indexű elemei
#> [1] 12 14 16 18 20
```

A fenti példákban látható, hogy az indexelés során létrejött vektorok elemszáma az indexvektor elemszámával egyenlő. Egy indexet akár többször is felsorolhatunk, és tetszőleges sorrend megengedett. A szögletes zárójelben lévő indexvektort helyben is elkészíthetjük a `c()` és `seq()` függvénnyel (vagy bármilyen más vektorlétrehozó függvénnyel), vagy a kettőspont (`:`) operátorral, de korábban létrehozott objektumot is használhatunk indexelésre (`x[y]`).

Az indexelés során több vektorelemet egy lépésben is tudunk módosítani. Az indexelt elemek kaphatnak azonos vagy különböző értéket. Itt is a vektoraritmetika szabályai működnek.

```
x <- 11:20
x[c(1, 2, 3)] <- c(110, 120, 130) # x 1., 2. és 3. elemét módosítjuk
x[c(4, 5, 6)] <- 0                # x 4., 5. és 6. elemét módosítjuk
x[c(7, 8, 9)] <- c(170, 180)      # x 7., 8. és 9. elemét módosítjuk
x
#>  [1] 110 120 130   0   0   0 170 180 170  20
```

A fenti példában az `x` vektor három-három elemét módosítjuk az egyes értékadások során. Az értékadó operátor (`<-`) engedelmeskedik a vektoraritmetika szabályainak, azaz az értékadás bal és jobb oldalán szereplő vektorokat tekinthetjük két olyan vektornak, amelyek között műveletet szeretnénk végrehajtani. Az első értékadásban azonos elemszámú a két vektor, a koordinátánkénti értékadás azonnal megtörténik (`x[c(1, 2, 3)] <- c(110, 120, 130)`). A másik két értékadásban különbözik a két vektor elemszáma, így először ismétléssel kiegészül a jobb oldali, rövidebbik vektor, majd ezután következhet a koordinátánkénti végrehajtás.

Egy vektor indexe mindig egész szám, de az R megengedi, hogy tört értékeket tartalmazó indexvektort szerepeltessünk az index operátorban, ekkor az egész részét veszi az indexeknek, egyszerűen csonkolja őket.

```
x <- 11:20
x[2.3]       # x 2. eleme
#> [1] 12
x[2.8]       # x 2. eleme
#> [1] 12
```

Negatív értékeket tartalmazó numerikus vektorral is indexelhetünk, ekkor a negatív előjellel megadott sorszámokon kívül az összes többi elemet tudjuk elérni vagy módosítani.

```
x <- 11:15
x[-3]                       # minden x elem, kivéve a 3.
#> [1] 11 12 14 15
x[-c(1, 5)]                 # minden x elem, kivéve az 1. és az 5.
#> [1] 12 13 14
x[-(1:3)]                   # minden x elem, kivéve az első 3
#> [1] 14 15
x[-2] <- 0                  # minden x elem módosul, kivéve a 2.
x
#> [1]  0 12  0  0  0
```

5.3.3.14. Indexelés karaketeres vektorokkal

Amennyiben egy vektor elemei rendelkeznek névvel, akkor karakteres indexvektorokat is használhatunk az indexeléshez. Ez meglehetősen nagy könnyebbséget jelent, ugyanis nem kell ismernünk a kívánt elem pozícióját, azaz indexét, elegendő fejben tartanunk az elem nevét. Vegyük példaként a tanulók matematika versenyen elért pontszámait tartalmazó vektort.

```
x <- c('Peti'=35, 'Bori'=37, 'Éva'=33)
x["Bori"]                              # x "Bori" nevű eleme
#> Bori
#>   37
x[c("Peti", "Éva")]                    # x "Peti" és "Éva" nevű eleme
#> Peti  Éva
#>   35   33
x[c("Peti", "Éva")] <- c(36, 34)       # x fenti 2 elemének módosítása
x
#> Peti Bori  Éva
#>   36   37   34
```

Látható, hogy a kívánt elem eléréséhez, például Bori matematika teljesítményéhez nem kell ismernünk Bori pontszámának pozícióját, elegendő a névre emlékeznünk.

5.3.3.15. Indexelés logikai vektorokkal

Vektorok indexeléséhez logikai vektorokat is használhatunk. Első pillanatban kényelmetlennek, sőt feleslegesnek tűnik ez a lehetőség, de a következő fejezetben, a vektorok szűrésénél, magunk is meggyőződhetünk e módszer káprázatos erejéről.

A logikai indexvektor működése nagyon egyszerű. Hossza az indexelendő vektor hosszával egyenlő, és a `TRUE` logikai értékkel jelezzük, hogy az adott pozíción lévő elemet el akarjuk érni, a `FALSE` értékkel pedig azt, hogy nincs szükség arra az elemre.

```
x <- 11:15
x[c(T, F, T, T, F)]  # x vektor 1., 3., és 4. eleme
#> [1] 11 13 14
```

A fenti példában `TRUE` szerepel az 1., 3. és 4. pozícióban, így az `x` vektor 1., 3. és 4. elemeit érhetjük el.

Az indexelésre használt logikai vektor elemszáma kisebb is lehet, mint az indexelt vektor hossza, ekkor az R az indexvektor ismétlésével kapja meg a kívánt hosszt.

```
x <- 11:15
x[c(T, F)]      # x vektor 1., 3. és 5. eleme
#> [1] 11 13 15
x[T]            # x vektor összes eleme
#> [1] 11 12 13 14 15
x[F]            # x vektor egyik eleme sem
#> integer(0)
```

A `c(T, F)` vektor két elemű, az indexelendő `x` viszont 5 hosszú, így az R ismétléssel előállítja a `c(T, F, T, F, T)` öt elemű vektort, és ezt használja az `x` indexeléséhez. Ha a csupa `TRUE` értékű vektorral indexelünk, akkor az `x` vektor összes elemét megkapjuk, ha pedig a csupa `FALSE` értékkel, akkor az üres vektort kapjuk.

A logikai vektorral indexelt vektorelemeket ugyanúgy módosíthatjuk, mint korábban a numerikus és karakteres indexvektorok esetén.

```
x <- 11:15
x[c(T, F)] <- 0                    # x vektor 1., 3. és 5. elemét módosítjuk
x[c(F, T, F, T, F)] <- c(120, 140) # x vektor 2. és 4. elemét módosítjuk
x
#> [1]   0 120   0 140   0
```

5.3.3.16. Indexelés speciális értékekkel

Az indexelésnek van néhány speciális esete, amelyet érdemes ismernünk. Vektorok indexelése során az indexoperátor üresen is maradhat, ekkor a vektor összes elemét elérhetjük, vagyis az `x` és `x[]` kifejezések ugyanazt az outputot adják.

```
x <- 11:15
x[]  # x minden eleme
#> [1] 11 12 13 14 15
x[NaN]  # egyetlen NA
#> [1] NA
x[NA]  # x elemszámának megfelelő NA
#> [1] NA NA NA NA NA
```

A fenti példákból kiolvasható, hogy a `NaN` és `NA` indexként való használata egyetlen `NA`-t, vagy az `x` hosszának megfelelő számú hiányzó értéket szolgáltat.

Legyünk óvatosak, ha az indexvektor tartalmaz `NA` értéket, akkor az eredménybe azon a pozíción szintén `NA` fog bekerülni.

```
x <- 11:15
x[c(1, NA, 2)]          # x 1. eleme, NA és x 2. eleme
#> [1] 11 NA 12
x[c(1, NA, 2)] <- 100   # x 1. és 2. elemének módosítása
x
#> [1] 100 100  13  14  15
```

Kerüljük az értékadást `NA`-t tartalmazó indexvektor használata esetén. A fenti példában az értékadás ugyan nem jelez hibát, és ellenőrizhetjük, hogy valóban megtörtént az első két vektorelem módosítása. Azonban az értékadás jobb oldalán a több elemű vektor már nem engedélyezett, például az `x[c(1, NA, 2)] <- c(100, 200)` értékadás hibaüzenethez vezet. Összefoglalva, minden esetben ellenőrizzük, hogy az indexvektorunk tartalmaz-e `NA` hiányzó értéket.

5.3.3.17. Vektor szűrése

Eddig a vektorok elemeit pozíciójuk alapján értük el. Akár sorszámot, elemnevet vagy megfelelő pozícióban lévő logikai igaz/hamis értéket használtunk indexelésre, végső soron az számított, hogy az adott elem hol található a vektorelemek egydimenziós sorában. Ebben a fejezetben egy teljes más kiinduló pontot használunk a vektorelemek elérésére és ez a vektor tartalma lesz, vagyis a vektorelem konkrét értéke (és nem a pozíciója).

Bővítsük ki a matematika pontszámokat tartalmazó vektorunkat, rögzítsük hat tanuló eredményét.

```
# 6 tanuló matematika pontszáma
x <- c('Peti'=35, 'Bori'=37, 'Éva'=33, 'Pál'=21, 'Gergő'=34, 'Ili'=40)
x
#>  Peti  Bori   Éva   Pál Gergő   Ili
#>    35    37    33    21    34    40
```

Ha arra vagyunk kíváncsiak, hogy kik értek el 36 pontnál többet a versenyen és milyen pontszámokkal, akkor rövid áttekintés után megadhatjuk a választ, sőt a pozíció alapján könnyen elvégezhetjük az alábbi indexeléseket is.

```
x[c(2, 6)]               # indexelés numerikus vektorral
#> Bori  Ili
#>   37   40
x[c("Bori", "Ili")]      # indexelés karakteres vektorral
#> Bori  Ili
#>   37   40
x[c(F, T, F, F, F, T)]   # indexelés logikai vektorral
#> Bori  Ili
#>   37   40
```

A fenti sorok az eddigiekhez képest semmilyen újdonságot nem tartalmaznak, lényegében összefoglalják a pozíció alapú indexelésről tanultakat. Felmerülhet bennünk a kérdés, ha `x` nem hat elemű, hanem 60 vagy esetleg 600, akkor mennyi esélyünk lenne az indexelt kifejezések előállítására. Nem sok.

Adódik azonban egy másik lehetőség, amely közvetlenül abból indul ki, hogy a 36 pontnál nagyobb vektorelemeket keressük. Logikai művelettel ezt a következőképp fogalmazhatjuk meg.

```
x > 36        # relációs művelet, logikai vektort eredményez
#>  Peti  Bori   Éva   Pál Gergő   Ili
#> FALSE  TRUE FALSE FALSE FALSE  TRUE
```

Korábban láttuk, hogy ez a művelet a vektoraritmetikai szabályainak engedelmeskedve két lépésben értelmezhető: (1) mivel különböző elemhosszú a két vektor, `x` hat elemű, a 36 egy elemű, először a jobb oldal is hat elemű lesz (`c(36, 36, 36, 36, 36, 36)`), majd (2) koordinátánként a relációs művelet végrehajtásra kerül, azaz `x` minden eleméről döntés születik, hogy nagyobb-e, mint 36. A relációs művelet eredménye egy hat elemű logikai vektor, amely pontosan ott `TRUE`, ahol az illető `x` elem nagyobb 36-nál, minden más helyen pedig `FALSE`. Esetünkben a Bori és Ili elemeknél jelenik meg a `TRUE`, vagyis a 2. és 6. pozícióban. Vegyük észre, hogy ez pontosan az a logikai vektor, mint amit korábban hoztunk létre a pozíció alapú indexelés egyik példájaként (`x[c(F, T, F, F, F, T)]`).

A relációs művelet eredményét, mint logikai vektort, kiválóan fel tudjuk használni az indexelésben a 36 pontnál nagyobb vektorelemek eléréséhez.

```
x[x > 36]       # x vektor szűrése (36-nál nagyobb elemek leválogatása)
#> Bori  Ili
#>   37   40
```

A fenti sor az első példa szűrésre. A szűrés lényegében logikai vektorral való indexelés, ahol a logikai indexvektort egy olyan logikai kifejezés állítja elő, amely hivatkozik a vektor tartalmára. A definíciót értelmezve a példára: a logikai vektor, amely alapján az indexelés történik a `c(F, T, F, F, F, T)`, a logikai kifejezés, amely ezt előállítja az `x>36`, a vektor tartalmára pedig természetesen az `x` objektumnévvel utalunk a logikai kifejezésen belül.

A szűrés nagyszerűen kezeli a vektorhosszal kapcsolatban korábban felvetett problémánkat. Ha az `x` nem hat, hanem 60 vagy 600 elemű, akkor is az `x[x>36]` végzi a 36-nál nagyobb elemek leválogatását.

Próbáljuk ki a szűrést nagyobb elemszám esetén is. Generáljunk 60 véletlen értékeket a 0-40 értéktartományból, úgy mintha 60 tanuló matematika pontszáma állna rendelkezésre. A `sample()` függvény az `x=` argumentumában megadott értékekből, a `size=`-ban megadott darabszámnyit állít elő. A `replace=T` argumentummal gondoskodunk arról, hogy egy érték többször is szerepelhessen az eredményvektorban.

```
pontszamok <- sample(x = 0:40, size = 60, replace = T) # véletlen értékek
pontszamok[1:10]               # vektor első 10 eleme
#>  [1] 22 11 36 37 30  3 37  8  4  4
pontszamok[pontszamok > 36]    # vektor szűrése
#> [1] 37 37 37 38
```

A `pontszamok` vektor 60 elemű, az első 10 értékét a képernyőn láthatjuk. A 36-nál nagyobb elemek megjelenítését szűréssel végeztük. Látható, hogy a szűrés nem lett bonyolultabb a vektor hosszának növekedésével.

Más relációs operátorokat (5.6. táblázat) is használhatunk a szűrésben, sőt logikai operátorok (5.7. táblázat) segítségével tetszőleges természetes nyelven megfogalmazott feltételt át tudunk fordítani R logikai kifejezésbe. A logikai operátorokat tartalmazó logikai kifejezéseket *összetett logikai kifejezés*eknek nevezzük. Írassuk ki a pontszámokat 36 és 39 között, majd 3 és 6 között, és végül mindezeket együtt.

```
pontszamok[pontszamok>=36 & pontszamok<=39] # pontszámok 36-39 között
#> [1] 36 37 37 36 37 38
pontszamok[pontszamok>=3 & pontszamok<=6]   # pontszámok 3-6 között
#>  [1] 3 4 4 4 5 3 6 5 5 3 5
# mindkettő egyszerre, logikai vagy operátorral összefűzve
pontszamok[(pontszamok>=36 & pontszamok<=39) | (pontszamok>=3 & pontszamok<=6)]
#>  [1] 36 37  3 37  4  4  4  5  3  6  5  5 36 37  3 38  5
```

Időnként szükségünk lehet arra az információra, hogy a vektorban melyik pozícióban vannak a feltételnek eleget tevő vektorelemek. Erre a feladatra a `which()` függvényt használhatjuk. A

`which()` függvény bemenő paraméterként egy logikai vektort vár, visszatérési értéke pedig a `TRUE` logikai értékek indexe lesz.

Térjünk vissza a matematika pontszámokhoz.

```
x <- c('Peti'=35, 'Bori'=37, 'Éva'=33, 'Pál'=21, 'Gergő'=34, 'Ili'=40)
which(x > 36)             # hol vannak 36-nál nagyobb elemek
#> Bori  Ili
#>    2    6
which(36 <= x & x <= 39)  # hol vannak 36-39 közötti elemek
#> Bori
#>    2
which(x == 21)            # hol van a 21-es elem
#> Pál
#>   4
which(x != 21)            # hol nincs 21-es elem
#>  Peti  Bori   Éva Gergő   Ili
#>     1     2     3     5     6
```

Az outputokban nem látjuk a tanulók pontszámát, tehát nem a szűrés a `which()` célja, azoknak a vektorelemeknek az indexét látjuk, amelyek az egyszerű vagy összetett logikai kifejezéseknek eleget tesznek.

Végezetül tekintsük át a szűrés és az értékadás kapcsolatát. Az adatelemzés során előfordulhat, hogy bizonyos feltételnek eleget tevő elemeket módosítani szeretnénk. Például, ha egy vektorban előzetesen a hiányzó értékeket 99-cel jelöljük, akkor a későbbi hibamentes elemzéshez `NA`-ra kell módosítanunk ezeket az értékeket.

```
x <- c(11, 3, 99, 4, 99)   # nyers vektor, a 99 jelentése hiányzó érték
x[x == 99] <- NA           # 99 átírása NA-ra
x
#> [1] 11  3 NA  4 NA
```

Az `x` így már helyes módon tartalmazza a hiányzó értékeket. Ha esetleg később kiderül ezeknek az elemeknek a tényleges értéke, akkor az `NA`-t kell helyettesítenünk új értékekkel. Vigyázzunk, az `x == NA` kifejezés helytelen a hiányzó értékek tesztelésére, erre az `is.na()` függvényt kell használnunk.

```
x[is.na(x)] <- c(5, 7)    # hiányzó értékek módosítása
x
#> [1] 11  3  5  4  7
```

Az `x` vektorban két hiányzó érték volt, így a fenti értékadás jobb oldalán két elemű vektort használunk. Ha mindkét hiányzó értéket azonos számmal szeretnénk felülírni, akkor elegendő lenne a `x[is.na(x)] <- 7` kifejezés is.

Korábban már említettük a (5.3.3.16. alfejezetben, hogy kerüljük az értékadást `NA`-t tartalmazó indexvektor használata esetén. Azonban nem minden esetben tudunk kitérni az ilyen esetek elől. Növeljük meg a hiányzó értékeket tartalmazó `x` vektor azon elemeit 1-gyel, amelyek 36-nál kisebbek! A nyilvánvalónak látszó `x[x < 36] <- x[x < 36] + 1` parancs helytelen, hibaüzenetet ad. Az értékadás mindkét oldalán a logikai kifejezésekhez fűzzük hozzá a `& !is.na(x)` kifejezést, így tudjuk az `NA` értékeket eltávolítani az értékadás mindkét oldaláról.

```
x <- c(33, NA, 32, 38, NA, 37)
# biztonságos értékadás NA-t tartalmazó vektor esetén
x[x < 36 & !is.na(x)] <- x[x < 36 & !is.na(x)] + 1
```

5.3.3.18. Vektor rendezése

Egy vektor elemeit növekvő vagy csökkenő sorrendbe rendezhetjük. Az R-ben a vektor elemeit a `sort()` vagy az `order()` függvénnyel rendezhetjük.

```
x <- c(1:5, 5:3); x
#> [1] 1 2 3 4 5 5 4 3
sort(x)               # x elemei növekvő sorrendben
#> [1] 1 2 3 3 4 4 5 5
sort(x, decreasing=T) # x elemei csökkenő sorrendben, vagy: rev(sort(x))
#> [1] 5 5 4 4 3 3 2 1
```

A `sort()` függvény alapértelmezés szerint növekvő sorrendbe rendezi a bemeneti vektort, ha azonban a `decreasing=` paramétert `TRUE`-ra állítjuk, csökkenő rendezést kapunk. A `rev()` függvénnyel, amely a bementi vektor elemeit fordított sorrendben sorolja fel, szintén elérhetjük a csökkenő rendezettséget.

Ha a `sort()` függvénnyel átrendezett vektort a továbbiakban fel szeretnénk használni, akkor azt érdemes új objektumban tárolni (`x.2 <- sort(x)`).

A vektor rendezésének másik módja az `order()` függvényhez kapcsolódik. A visszatérési érték ekkor egy numerikus indexvektor, amellyel a bemenő vektort indexelve rendezett vektort kapunk.

```r
x <- c(1:5, 5:3)
x
#> [1] 1 2 3 4 5 5 4 3
order(x)  # indexekkel tér vissza
#> [1] 1 2 3 8 4 7 5 6
x[order(x)]  # azonos a sort(x)-szel
#> [1] 1 2 3 3 4 4 5 5
x[order(x, decreasing = T)]  # azonos a sort(x, decreasing=T)-val
#> [1] 5 5 4 4 3 3 2 1
```

Az `order()` függvény esetében is használhatjuk a `decreasing=` paramétert, amellyel csökkenő sorrendbe rendezhetjük a vektorunkat.

A numerikus vektorokon túl a karakteres és logikai vektorokat is sorba rendezhetjük a `sort()` és `order()` függvényekkel.

Összefoglalás

Gratulálunk! Maratoni alfejezetünk végigolvasásával jelentős lépést tett meg az Olvasó a magabiztos R ismeretek megszerzéséhez. A vektor minden adatelemzési munka alapja, biztos kezelése kulcsfontosságú. Tetszőleges vektor létrehozásához a `c()` függvényt használhatjuk, és az elemeket akár nevesíthetjük is. Szabályos vektort a `seq()`, `seq_along()`, `rep()` és a `paste()` függvénnyel, vagy a kettőspont (`:`) operátorral készíthetünk. Megbeszéltük a vektorok közötti műveletek végrehajtásának fő szabályát: ismétléssel hozzuk azonos hosszra a vektorokat ha szükséges, majd koordinátánként végezzük el a kívánt műveletet. A vektorokat támogatják a matematikai függvények is, minden vektorelemre meghívódik a függvény. A statisztikai függvények szintén vektort várnak, de többnyire egy értéket szolgáltatnak. A vektorok típusának tesztelése az `is.*()`, a konvertálása pedig az `as.*()` függvényekkel történik. A vektorok indexelésével (`vektor[indexvektor]`) a vektor elemeit pozíció alapján, a vektorok szűrésével (`vektor[logikai-indexvektor]`) a vektor elemeit érték alapján érhetjük el vagy módosíthatjuk. A vektorok rendezését a `sort()` és az `order()` függvénnyel végezzük.

Feladatok

1. Hozzuk létre a következő numerikus vektort: 12, 14, 17.
2. Hozzuk létre a következő karakteres vektort: “Vác”, “Eger”, “Pécs”.
3. Hozzuk létre a következő logikai vektort: TRUE, FALSE, FALSE.
4. Hozzuk létre egy számtani sorozat egymást követő elemeit, ahol az első elem 8, az utolsó 102 és a különbség 1.

5. Hozzuk létre egy számtani sorozat egymást követő elemeit, ahol az első elem 102, az utolsó 8 és a különbség -1.
6. Hozzuk létre egy számtani sorozat egymást követő elemeit, ahol az első elem 8, az utolsó 102 és a különbség 2.
7. Hozzuk létre egy számtani sorozat egymást követő elemeit, ahol az első elem 8, a különbség 3 és a vektor 25 elemű.
8. Hozzuk létre azt a numerikus vektort, amely 12 elemű, és minden elemének -2 az értéke!
9. Hozzuk létre azt a karakteres vektort, amely 7 elemű, és minden elemének "Péntek" az értéke!
10. Hozzuk létre azt a logikai vektort, amely 7 elemű, és minden elemének TRUE az értéke!
11. Hozzuk létre azt a numerikus vektort, amely a 2, 3, 5 elemeket háromszor egymás után megismétli! Hány elemű az így létrejött vektor?
12. Hozzuk létre azt a numerikus vektort, amely a 2, 3, 5 elemeket háromszor helyben megismétli! Hány elemű az így létrejött vektor?
13. Hozzuk létre azt a numerikus vektort, amely a 2, 3, 5 elemeket helyben megismétli úgy, hogy a 2-őt 4-szer, a 3-at 5-ször és az 5-öt 7-szer ismétli meg! Hány elemű az így létrejött vektor?
14. Szabályos vektorok létrehozásának van egy korábban még nem említett módja: a `sequence()` függvény. Ismerjük meg a súgóból ezt a függvényt, és értelmezzük a `sequence(4)` és `sequence(c(4,5))` függvényhívásokat!
15. Vektorok létrehozásának számos módját megismertük ebben a fejezetben, de elemek megadása nélkül, vagy akár nulla hosszúsággal is létrehozhatunk vektort. A `double()`, `integer()`, `character()` és `logical()` függvények közvetlenül az adott típusnak megfelelő vektort hozzák létre. A súgó tanulmányozásával állítsunk elő 0 és 10 elemű vektor objektumokat mind a négy típus esetén.
16. Próbáljuk ki az 5.9. táblázatban szereplő példákat.
17. Hozzuk létre a `'Peti'=5, 'Bori'=NA, 'Éva'=3, 'Pál'=NA, 'Gergő'=5, 'Ili'=4` adatokat tartalmazó vektort, majd rendezzük, indexeljük az első és az utolsó elemét, válogassuk le az 5-ös értékeket, csökkentsük mindegyik értéket 1-gyel, csak az 5-öket csökkentsük 1-gyel.
18. A fejezetben ismertetett `seq_along()` függvény mellett az `seq_len()` függvény is létezik, amely a megadott számú elemet tartalmazó vektort hoz létre. Milyen esetekben célszerű az `seq_len()` függvényt használni?

5.3.4. Mátrix

A mátrix adatszerkezet egyetlen lényeges dologban különbözik a vektortól: a mátrix kétdimenziós, sorokba és oszlopokba szervezi az elemeket, míg a vektor egydimenziós (érdemes visszalapozni a 5.1. ábrához). A mátrix ugyanúgy homogén, mint a vektor, ennek megfelelően beszélünk *double*, *integer*, *karakteres* és *logikai* mátrixokról.

5.3.4.1. Mátrix létrehozása

Mátrix létrehozásához a `matrix()` függvényt használjuk, amely egy kiinduló vektor elemeit használja fel a mátrix feltöltéséhez. A `data=` argumentumban kell megadnunk a kiinduló vektort, majd az `nrow=` és/vagy `ncol=` argumentumokban közöljük a sorok és oszlopok számát.

```
x <- matrix(data=1:20, nrow=4)        # 4x5-ös integer mátrix
x
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    5    9   13   17
#> [2,]    2    6   10   14   18
#> [3,]    3    7   11   15   19
#> [4,]    4    8   12   16   20
```

A fenti példában a 20 elemű vektort 4 sorban rendezi el a `matrix()` függvény, ennek megfelelően 5 oszlopos lesz az `x` mátrix. A `matrix()` függvényben az `ncol=` paraméter is használható.

```
x <- matrix(data=1:20, nrow=4, ncol=5)        # 4x5-ös integer mátrix
x
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    5    9   13   17
#> [2,]    2    6   10   14   18
#> [3,]    3    7   11   15   19
#> [4,]    4    8   12   16   20
x <- matrix(data=1:20, nrow=4, ncol=10)       # 4x10-es integer mátrix
x
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#> [1,]    1    5    9   13   17    1    5    9   13    17
#> [2,]    2    6   10   14   18    2    6   10   14    18
#> [3,]    3    7   11   15   19    3    7   11   15    19
#> [4,]    4    8   12   16   20    4    8   12   16    20
```

Az `ncol=5` szerepeltetése nem jelent változást az előző példához képest, az `x` mátrix 4 sort és 5 oszlopot fog tartalmazni, rövidebben 4 × 5-ös. A következő sorban az `ncol=10` argumentum már egy 40 elemű mátrix létrehozását kezdeményezi (4 × 10-es), így az `1:20` vektor ismétlésével állnak elő a szükséges elemek. (Figyelmeztetést kapunk, ha a szükséges mátrixelemszám eléréséhez nem egész számszor kell ismételni a kiinduló vektort, de a mátrix ebben az esetben is létre fog jönni.)

A fenti példában azt is megfigyelhetjük, hogy a 20 elemű vektorból oszlop-folytonosan jön létre a mátrix, vagyis először az első oszlop töltődik fel a vektorelemekkel, majd a második, és így tovább. Ha sor-folytonosan szeretnénk a bemenő vektor elemeiből mátrixot képezni, akkor a `byrow=` paramétert igazra kell állítanunk.

```
x <- matrix(1:12, nrow = 3, byrow = T)  # 3x4-es integer mátrix, sor-folytonosan
x
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    2    3    4
#> [2,]    5    6    7    8
#> [3,]    9   10   11   12
```

Mátrixot karakteres vagy logikai értékekből is építhetünk.

```
matrix(c("az","egy"), nrow=2, ncol=3, byrow=T) # 2x3-as karakteres mátrix
#>      [,1]  [,2]  [,3]
#> [1,] "az"  "egy" "az"
#> [2,] "egy" "az"  "egy"
matrix(c(T,F,T), nrow=2, ncol=6, byrow=T)      # 2x6-os logikai mátrix
#>      [,1]  [,2] [,3] [,4]  [,5] [,6]
#> [1,] TRUE FALSE TRUE TRUE FALSE TRUE
#> [2,] TRUE FALSE TRUE TRUE FALSE TRUE
```

Az előző fejezetben láttuk, hogy a vektorok elemeinek nevet is adhatunk, így olvashatóbbá tehetjük rögzített adatainkat. A `matrix()` függvény `dimnames=` argumentumában az egyes sorok és oszlopok elnevezéséről, valamint a két dimenzió nevéről is gondoskodhatunk.

```
x <- matrix(0, nrow = 2, ncol = 3,
      dimnames = list('1. dim. neve'=c("sor.1", "sor.2"),
                      '2. dim. neve'=c("oszl.1", "oszl.2", "oszl.3")))
x
#>             2. dim. neve
#> 1. dim. neve oszl.1 oszl.2 oszl.3
```

```
#>        sor.1      0      0      0
#>        sor.2      0      0      0
```

A `dimnames=` argumentum a dimenzió-, sor- és oszlopneveket listába rendezve várja. A listákról a 5.3.6. fejezetben olvashatunk. A sor- és oszlopnevek megadásánál tartsuk be az objektumok elnevezésével kapcsolatos szabályokat, azaz betűvel kezdjünk, kerüljük a szóközt és egyéb írásjeleket, tagolásra a pontot használjuk.

Létező mátrix esetén a `rownames()` és a `colnames()` függvényekkel tudjuk a sor- és oszlopneveket lekérdezni, illetve módosítani. Az egyes dimenziónevek módosítására a `names(dimnames(x))` konstrukciót használhatjuk.

```
rownames(x)                           # sornevek lekérdezése
#> [1] "sor.1" "sor.2"
colnames(x)                           # oszlopnevek lekérdezése
#> [1] "oszl.1" "oszl.2" "oszl.3"
rownames(x) <- c("eset.1", "eset.2")  # sornevek módosítása
colnames(x) <- c("o.1", "o.2", "o.3") # oszlopnevek módosítása
x
#>             2. dim. neve
#> 1. dim. neve o.1 o.2 o.3
#>       eset.1   0   0   0
#>       eset.2   0   0   0
names(dimnames(x)) <- c("esetek", "oszlopok") # dimenziónevek módosítása
x
#>         oszlopok
#> esetek   o.1 o.2 o.3
#>   eset.1   0   0   0
#>   eset.2   0   0   0
```

5.3.4.2. Mátrix indexelése

A mátrixok indexelése nagyon hasonló a vektorok indexeléséhez. Itt is az index operátort (`[]`) kell használnunk, de a két dimenzió miatt vesszővel választjuk el a sorra és az oszlopra vonatkozó indexeket. Mátrix indexelésének általános alakja:

```
# mátrix indexelése, az eredmény egy mátrix vagy egy vektor
mátrix[sor-indexvektor, oszlop-indexvektor]
```

A sor-indexvektorra és az oszlop-indexvektorra ugyanazok a szabályok érvényesek, mint vektor esetén az indexvektorra. Használhatunk numerikus, karakteres és logikai egy vagy több elemű vektort, numerikus indexeknél negatív értéket, és természetesen el is hagyhatjuk az egyes dimenziók indexvektorait. Nézzünk ezekre néhány példát.

```
x <- matrix(1:10, nrow=2, ncol=5, byrow=T)
x                       # 2x5-ös mátrix
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    2    3    4    5
#> [2,]    6    7    8    9   10
x[2, 3]                 # 1 elem elérése, vektor output
#> [1] 8
x[2, c(1,4)]            # 2 elem elérése, vektor output
#> [1] 6 9
x[, c(1,4)]             # 4 elem elérése, 2x2-es mátrix output
#>      [,1] [,2]
#> [1,]    1    4
#> [2,]    6    9
x[, -c(1,4)]            # 6 elem elérése, 2x3-as mátrix output
#>      [,1] [,2] [,3]
#> [1,]    2    3    5
#> [2,]    7    8   10
x[1, ]                  # 5 elem elérése, vektor output
#> [1] 1 2 3 4 5
x[c(2, 1), c(T, F, T)]  # 6 elem elérése, 2x3-as mátrix output
#>      [,1] [,2] [,3]
#> [1,]    6    8    9
#> [2,]    1    3    4
```

A mátrix indexelése során a kapott új adatszerkezetek elveszthetik a kétdimenziós jellegüket és így mátrix helyett vektor is lehet az indexelés eredménye. Ha ezt el akarjuk kerülni, használjuk a `drop=FALSE` paramétert az indexben, ekkor minden esetben mátrix lesz az eredmény.

```
x[2, 3, drop=F]             # 1 elem elérése, 1x1-es mátrix output
#>      [,1]
#> [1,]    8
x[2, c(1,4), drop=F]        # 2 elem elérése, 1x2-es mátrix output
#>      [,1] [,2]
#> [1,]    6    9
x[2, , drop=F]              # 5 elem elérése, 1x5-ös mátrix output
```

```
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    6    7    8    9   10
x[, 3, drop=F]              # 2 elem elérése, 2x1-es mátrix output
#>      [,1]
#> [1,]    3
#> [2,]    8
```

Amennyiben a mátrixunk sor- és oszlopnevekkel is rendelkezik, akkor ezeket is felhasználhatjuk az indexelés során.

```
x <- matrix(1:10, nrow=2, ncol=5, byrow=T)
rownames(x) <- c("eset1", "eset2")
colnames(x) <- paste("v", 1:5, sep=".")
x                                   # 2x5-ös mátrix sor- és oszlopnevekkel
#>       v.1 v.2 v.3 v.4 v.5
#> eset1   1   2   3   4   5
#> eset2   6   7   8   9  10
x["eset1", c("v.2", "v.1")]         # 2 elem elérése, vektor output
#> v.2 v.1
#>   2   1
x[1:2, c("v.2", "v.1")]             # 4 elem elérése, 2x2-es mátrix
#>       v.2 v.1
#> eset1   2   1
#> eset2   7   6
x["eset2", paste("v", 1:3, sep=".")]  # 3 elem elérése, vektor
#> v.1 v.2 v.3
#>   6   7   8
x["eset1", c(T,F), drop=F]          # 3 elem elérése, 1x3-as mátrix
#>       v.1 v.3 v.5
#> eset1   1   3   5
```

5.3.4.3. Számítások a mátrix soraiban és oszlopaiban

Az előző részben említettük, ha üresen hagyjuk a mátrix sor vagy oszlop pozícióját az indexelés során, akkor a mátrix teljes oszlopára vagy sorára tudunk hivatkozni, vagyis alapesetben vektort kapunk. Az így kapott vektorokkal tetszőleges műveleteket hajthatunk végre. Hozzunk létre egy 3×4-es mátrixot, amely 3 tanuló átlagát tartalmazza 4 tantárgyból.

```r
x <- matrix(c(3.7, 5.3, 5.1, 4.2, 4.4, 3.8, 2.9, 4.2, 5.1, 4, 3, 5),
            nrow=3, ncol=4, byrow=T,
            dimnames = list(c("Pál", "Ili", "Éva"),
                            c("matek", "magyar", "angol", "ének")))
x # mátrix kiírása, tanulók átlagai
#>     matek magyar angol ének
#> Pál   3.7    5.3   5.1  4.2
#> Ili   4.4    3.8   2.9  4.2
#> Éva   5.1    4.0   3.0  5.0
mean(x[1,])      # Pál féléves átlaga
#> [1] 4.575
sd(x[,4])        # énekből a csoport szórása
#> [1] 0.4618802
```

Négy speciális függvénnyel az oszlopok és sorok összegét és átlagát számíthatjuk ki.

```r
rowSums(x)      # sorösszegek, a tanulók jegyeinek összege
#>  Pál  Ili  Éva
#> 18.3 15.3 17.1
rowMeans(x)     # sorátlagok, a tanulók félév végi átlaga
#>   Pál   Ili   Éva
#> 4.575 3.825 4.275
colSums(x)      # oszlopösszegek, a tantárgyak jegyeinek összege
#>  matek magyar  angol   ének
#>   13.2   13.1   11.0   13.4
colMeans(x)     # oszlopátlagok, a tantárgyak átlaga
#>    matek   magyar    angol     ének
#> 4.400000 4.366667 3.666667 4.466667
```

Általánosabb megoldás, ha az `apply()` függvényt használjuk, amelyben a mátrix soraira vagy oszlopaira vonatkozó függvényt mi határozzuk meg, így az összegzésen és az átlagszámításon kívül más függvényeket is elérhetünk. Az `apply()` első paramétere maga a mátrix, a második helyen pedig 1 vagy 2 áll, attól függően, hogy a mátrix soraira (1) vagy oszlopaira (2) akarjuk a harmadik paraméterben szereplő függvényt alkalmazni.

```r
apply(x, 1, mean)    # sorátlagok, a tanulók félév végi átlaga
#>   Pál   Ili   Éva
#> 4.575 3.825 4.275
apply(x, 1, sd)      # soronkénti szórások
#>       Pál       Ili       Éva
```

```
#> 0.7544314 0.6652067 0.9844626
apply(x, 1, min)     # soronkénti minimumok
#> Pál Ili Éva
#> 3.7 2.9 3.0
apply(x, 2, mean)    # oszlopátlagok, a tantárgyak átlaga
#>    matek   magyar    angol     ének
#> 4.400000 4.366667 3.666667 4.466667
apply(x, 2, sd)      # oszloponkénti szórások
#>     matek    magyar     angol      ének
#> 0.7000000 0.8144528 1.2423097 0.4618802
apply(x, 2, min)     # oszloponkénti minimumok
#>  matek magyar  angol   ének
#>    3.7    3.8    2.9    4.2
```

Hiányzó értékek esetén a fenti függvények `NA` értéket adnak eredményül, így itt is szükséges az `na.rm=T` argumentum szerepeltetése.

```
x["Pál", "matek"] <- NA     # módosítjuk Pál matek jegyét hiányzóra
rowMeans(x)                 # Pálnál NA lesz
#>   Pál   Ili   Éva
#>    NA 3.825 4.275
apply(x, 1, mean)           # Pálnál NA lesz
#>   Pál   Ili   Éva
#>    NA 3.825 4.275
rowMeans(x, na.rm=T)        # így jó Pálnál is
#>      Pál      Ili      Éva
#> 4.866667 3.825000 4.275000
apply(x, 1, mean, na.rm=T)  # így jó Pálnál is
#>      Pál      Ili      Éva
#> 4.866667 3.825000 4.275000
```

A statisztikai függvények mindegyikében érdemes használni az `na.rm=T` argumentumot, így a hiányzó értékek figyelmen kívül hagyásával minden esetben megkapjuk eredményt. Ha nem tartalmaz hiányzó értékeket a mátrixunk, akkor az `na.rm=T` az eredményen természetesen nem változtat.

5.3.4.4. Sorok és oszlopok kezelése

Mátrixokat az `rbind()` és a `cbind()` függvényekkel is építhetünk.

```
cbind(1, 1:2, 1:4)    # mátrix létrehozása oszlopvektorokból
#>      [,1] [,2] [,3]
#> [1,]    1    1    1
#> [2,]    1    2    2
#> [3,]    1    1    3
#> [4,]    1    2    4
rbind(1, 1:2, 1:4)    # mátrix létrehozása sorvektorokból
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    1    1    1
#> [2,]    1    2    1    2
#> [3,]    1    2    3    4
```

Vektor paraméterek esetén, a felsorolt vektorok fogják alkotni az új mátrix oszlopait (`cbind()` esetén), illetve sorait (`rbind()` esetén), a rövidebb vektor, ha van ilyen, ismétlődni fog.

Új oszloppal vagy új sorral is kiegészíthetjük a már létező mátrixunkat.

```
x <- matrix(1:12, nrow=4, ncol=3); x
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
cbind(-3:0, x, 13:16)  # oszlopvektorok hozzáfűzése x elé és mögé
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]   -3    1    5    9   13
#> [2,]   -2    2    6   10   14
#> [3,]   -1    3    7   11   15
#> [4,]    0    4    8   12   16
rbind(-1, x, 1)        # sorvektorok hozzáfűzése x fölé és alá
#>      [,1] [,2] [,3]
#> [1,]   -1   -1   -1
#> [2,]    1    5    9
#> [3,]    2    6   10
#> [4,]    3    7   11
#> [5,]    4    8   12
#> [6,]    1    1    1
```

Tetszőleges pozícióba beszúrhatunk egy oszlopot vagy egy sort. Ehhez első lépésben a létező `x` mátrixhoz hozzáillesztjük az új oszlopot vagy sort, majd indexeléssel átrendezzük az oszlopokat vagy sorokat.

```
cbind(x, 13:16)[, c(1, 2, 4, 3)]  # oszlopvektor hozzáfűzése, majd oszlopok indexelése
#>      [,1] [,2] [,3] [,4]
#> [1,]    1    5   13    9
#> [2,]    2    6   14   10
#> [3,]    3    7   15   11
#> [4,]    4    8   16   12
rbind(x, -1)[c(1, 2, 3, 5, 4), ]  # sorvektor hozzáfűzése, majd sorok indexelése
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]   -1   -1   -1
#> [5,]    4    8   12
```

Hasznos lehetőség összesítő sorok vagy oszlopok mátrixhoz fűzése és elnevezése:

```
x <- matrix(1:12, nrow = 4, ncol = 3)
x
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
x <- rbind(x, apply(x, 2, mean))  # átlag sor hozzáfűzése
rownames(x) <- c(1:4, "átlag")  # az új sor nevének átírása
x
#>       [,1] [,2] [,3]
#> 1      1.0  5.0  9.0
#> 2      2.0  6.0 10.0
#> 3      3.0  7.0 11.0
#> 4      4.0  8.0 12.0
#> átlag  2.5  6.5 10.5
```

A sorok vagy oszlopok sorrendjét is megcserélhetjük a mátrixban, valamint ezek törlésére is van lehetőségünk:

```
x <- matrix(1:12, nrow=4, ncol=3); x
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
```

```
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
y <- x[, c(2, 3, 1)]          # oszlopcsere
y <- x[c(3, 2, 4, 1), ]       # sorcsere
y <- x[, c(1, 3)]             # a 2. oszlop törlése
y <- x[c(1, 3), ]             # az 2. és a 4. sor törlése
```

Látjuk, az oszlopok és sorok cseréjét, illetve törlését a mátrix indexelésével végezzük. Az oszlopok és sorok cseréje során a `c()` függvényben megadott sorrendben fogják felvenni az új pozíciójukat, míg törlés esetén csak azok az oszlopok és sorok maradnak meg, amelyeket a `c()` függvényben megadtunk.

🗂️ Összefoglalás

A mátrix homogén kétdimenziós adatszerkezet, és többnyire a `matrix()` függvénnyel hozzuk létre, de használhatjuk a `cbind()` és `rbind()` függvényeket is. Mátrix indexelése a `[,]` operátorral történik, ahol sor- és oszlopindex megadásra van lehetőségünk. A mátrix sorain vagy oszlopain külön-külön is tudunk műveleteket végezni az `apply()` függvénnyel, a sor- és oszlopneveket a `rownames()` és a `colnames()` függvénnyel kezelhetjük.

🎯 Feladatok

1. Hozzunk létre egy csupa 1-ből álló mátrixot, amelynek 3 sora és 2 oszlopa van!
2. Hozzunk létre egy 3 × 4-es karakteres mátrixot, amely 12 különböző keresztnevet tartalmaz!
3. Hozzunk létre egy 3 × 4-es logikai mátrixot, amelynek 1. és 3. sora `TURE` a 2. sora pedig `FALSE` értékeket tartalmaz!
4. Mátrixok indexelésére olyan speciális indexmátrix is használható, amelynek két oszlopa van, és az elérendő elemek sor- és oszlopkoordinátáit tartalmazza. Mutassunk példát erre a `mátrix[indexmátrix]` alakú mátrixindexelésre!
5. A fejezetben megismert `apply()` mellett nagyon népszerű az `sapply()` és a `vapply()` függvény is, amelyek a mátrixokkal is használhatóak. Ismerjük meg ezeket a függvényeket a súgójukból, és próbáljuk ki őket!
6. Mit jelent egy mátrix transzponálása? Hogyan tudjuk ezt megtenni R-ben?
7. Hozzunk létre egy 3 × 3-as egységmátrixot! Keressünk rá a `diag()` függvényre, és nézzük meg, hogy mit csinál! Keressünk rá a mátrixokkal kapcsolatos szokásos műveletekre is! Hozzunk létre egy 3 × 3-as véletlen mátrixot!

5.3.5. Faktor

A faktor adattípus nagyon hasonló a vektorhoz, ugyanis minden faktor egy speciális *integer* vektor, a faktor tehát homogén és egydimenziós adatszerkezet. Faktorokat elsősorban kategorikus változók értékeinek tárolására használjuk, ilyen például a személyek neme vagy iskolai végzettsége. A faktor egy lényeges ponton több mint egy egyszerű *integer* vektor. A faktor karbantart egy összerendelést az 1-gyel kezdődő numerikus egészek és a faktor lehetséges karakteres értékei, a címkék között (az 5.1. ábrán ezt egy piros kis téglalappal jelöltük). Egy faktorelem értéke csak ezekből a címkékből kerülhet ki, ami nagyfokú védelmet jelent számunkra az adatkezelés során. Ha például létrehozunk egy faktort az (1-`"férfi"`, 2-`"nő"`) összerendeléssel, akkor egy faktorelem csak a `"férfi"` vagy `"nő"` címkéket veheti fel, más értéket nem (az `NA` hiányzó érték természetesen lehet faktorelem értéke is). A munka során mindig a címkékkel találkozunk, a háttérben lévő numerikus egészek csak ritkán kapnak szerepet.

5.3.5.1. Faktor létrehozása

A faktorokat jellemzően karakteres vagy numerikus vektorokból hozzuk létre a `factor()` függvénnyel. A faktor létrehozásánál mindig gondoskodjunk a faktor lehetséges értékeinek, vagyis a faktor címkéiknek megadásáról. A címkéket néha (faktor)szinteknek (levels) is nevezzük. Mivel a kategorikus változóink lehetséges értékei többnyire ismertek az adatkezelés elején, a faktorszintek felsorolása nem okozhat nehézséget. Most hozzunk létre egy faktort, amely öt személy nemét tartalmazza.

```
x <- c("férfi", "férfi", "nő", "férfi", "nő")  # karakteres vektor létrehozása
x.f <- factor(x, levels=c("férfi", "nő"))      # faktor létrehozása
x.f                                            # faktor kiíratása
#> [1] férfi férfi nő    férfi nő
#> Levels: férfi nő
unclass(x.f)                                   # integer kódok a háttérben
#> [1] 1 1 2 1 2
#> attr(,"levels")
#> [1] "férfi" "nő"
```

Az `x.f` faktort az `x` karakteres vektorból hoztuk létre, így `x.f` ugyanúgy 5 hosszú, mint az `x`. Az `x.f` outputjában olvasható `Levels: férfi nő` rész azt közli velünk, hogy a háttérben az 1 numerikus értéknek a `"férfi"` címke, míg a 2-nek a `"nő"` címke felel meg. A belső integer kódok is feltárulnak az `unclass(x.f)` outputjában. A szám-címke összerendelést magunk szabályozhatjuk, ha a `factor()` függvény `levels=` argumentumában módosítunk a sorrenden.

```
x.f <- factor(x, levels=c("nő", "férfi"))      # faktor létrehozása
x.f
#> [1] férfi férfi nő    férfi nő
#> Levels: nő férfi
```

A fenti `x.f` faktor ugyanannak az 5 személynek a nemét tartalmazza, de az összerendelést a `levels=c("nő", "férfi")` paraméterrel (1-`"nő"`, 2-`"férfi"`)-re változtattuk. Láthatjuk, a címkék sorrendje a faktor értékeitől független, mégis fontos szerepet kap majd a táblázatok és ábrák megjelenítésénél, tehát érdemes rá odafigyelni.

A `levels=` argumentum szerepeltetése a `factor()` függvényben sok kellemetlenségtől kímélhet meg minket. Ha elhagyjuk, akkor a `factor()` függvény a karakteres vektorban aktuálisan rendelkezésre álló értékekből konstruálja meg a faktort. Nézzünk erre három esetet.

```
# helyesen írt, minden lehetséges érték szerepel: OK
(x.f.1 <- factor(c("férfi", "férfi", "nő", "férfi", "nő")))
#> [1] férfi férfi nő    férfi nő
#> Levels: férfi nő
# elgépelés miatt egy új szint is megjelenik: nem OK
(x.f.2 <- factor(c("férfi", "Férfi", "nő", "férfi", "nő")))
#> [1] férfi Férfi nő    férfi nő
#> Levels: férfi Férfi nő
# 5 azonos nemű személy: nem OK
(x.f.3 <- factor(c("nő", "nő", "nő", "nő", "nő")))
#> [1] nő nő nő nő nő
#> Levels: nő
```

Az első esetben a faktor létrehozásához használt karakteres vektor megegyezik a korábbival, azaz helyesen tartalmazza mind a `"férfi"`, mind a `"nő"` címkéket, így az `x.f.1` faktor a címkék lexikografikus rendezése alapján az (1-`"férfi"`, 2-`"nő"`) összerendeléssel jön létre. A második esetben karakteres vektorunk elgépelés miatt egy `"Férfi"` címkét is tartalmaz, ami az `x.f.2` faktor szintjei között is meg fog jelenni. A harmadik esetben az okozza a problémát, hogy 5 azonos nemű személy került a mintába, így a `"férfi"` címke egyáltalán nem jelenik meg az `x.f.3` faktor szintjei között. Az `x.f.2` és az `x.f.3` faktorok tehát más-más okok miatt, de hibásan tartalmazzák a faktorszinteket, és ez a későbbi működést alapvetően befolyásolja. Az `x.f.2` három különböző nemet ismer, az `x.f.3` pedig mindössze egyet. A fenti hibák a `levels=` szerepeltetésével könnyen kiküszöbölhetők.

```
# a "levels=" megadásával kiküszöbölhetők a problémák
(x.f.2 <- factor(c("férfi", "Férfi", "nő", "férfi", "nő"),
```

```
                 levels=c("férfi", "nő")))
#> [1] férfi <NA>  nő    férfi nő
#> Levels: férfi nő
(x.f.3 <- factor(c("nő", "nő", "nő", "nő", "nő"),
                 levels=c("férfi", "nő")))
#> [1] nő nő nő nő nő
#> Levels: férfi nő
```

A fenti példákban látható, hogy a `"Férfi"` címke helyére hiányzó érték került, az `x.f.3` faktor pedig már `"férfi"` értéket is fel tud venni a jövőben.

Numerikus vektorokból is készíthetünk faktorokat. Például a könnyebb rögzíthetőség miatt öt személy nemét most numerikus vektorban hoztuk létre azzal a szabállyal, hogy a 0 jelentése nő, az 1 jelentése férfi. A faktor létrehozása során ekkor a `levels=` szerepe a lehetséges numerikus értékek felsorolása lesz, és a plusz paraméterként szereplő `labels=` segít a faktorszintek beszédes elnevezésében. Az elnevezés a `levels=`-ben lévő numerikus értékek sorrendjében történik, ezért nagyon fontos, hogy a `labels=` címkéi kövessék ezt a sorrendet.

```
x <- c(1, 1, 0, 1, 0)     # numerikus vektor létrehozása, 0-nő, 1-férfi
# a "levels=" és a "labels=" legyen összhangban
(x.f.1 <- factor(x, levels=c(0, 1),
                    labels=c("nő", "férfi")))
#> [1] férfi férfi nő    férfi nő
#> Levels: nő férfi
(x.f.2 <- factor(x, levels=c(1, 0),
                    labels=c("férfi", "nő")))
#> [1] férfi férfi nő    férfi nő
#> Levels: férfi nő
```

A fenti példában látható, hogy a `levels=` értékeinek sorrendje vezérli az elnevezést, a 0 mindkét esetben `"nő"`, az 1 `"férfi"` címkéhez fog vezetni. Az `x.f.1` és `x.f.2` faktorok mindössze a háttérben lévő összerendelésben különböznek, első esetben az (1-`"nő"`, 2-`"férfi"`), míg második esetben az (1-`"férfi"`, 2-`"nő"`) lesz a faktorszintek sorrendje. Vegyük észre, hogy az eredeti 0 (`nő`) és 1 (`férfi`) értékek a faktorban már eltűnnek, szerepüket a címkék veszik át (`nő`, `férfi`) és az azok alapját jelentő 1-től sorszámozott integer vektor.

5.3.5.2. Rendezett faktor

A kategorikus változók két csoportját különböztetjük meg, a nominális változókat – ilyen volt az eddig látott *nem* változó –, és az ordinális változókat. Ez utóbbira példa az iskolai

végzettség, mert ennek lehetséges értékei (például “alap”, “közép” és “felső” értékekkel) sorba rendezhetők. Ha a változó szintjei közötti rendezettséget szeretnénk az R-ben is kifejezni, akkor rendezett faktort érdemes használni. Az eddigi `factor()` függvény is alkalmas az `ordered = T` argumentum használatával, de az `ordered()` függvényt is használhatjuk rendezett faktor létrehozására.

```
# rendezett faktor létrehozása
x <- c("felső", "közép", "alap", "közép", "felső")
x.f <- ordered(x = x, levels=c("alap", "közép", "felső"))
x.f
#> [1] felső közép alap  közép felső
#> Levels: alap < közép < felső
```

Az `ordered()` függvénnyel létrehozott rendezett faktor outputjában a szintek között a rendezettséget a kisebb (<) jelek teszik hangsúlyossá, de a függvény használata nem tér el a korábban látott `factor()` függvénytől.

5.3.5.3. Szabályos faktor létrehozása

Ismétlést tartalmazó faktorokat a `gl()` függvénnyel is létrehozhatunk. Tipikusan a szintek (`n=`) számát, az ismétlések számát (`k=`) és a címkéket (`labels=`) szoktuk megadni. Rendezett faktort az `ordered = T` argumentummal készíthetünk.

```
# szabályos faktorok létrehozása
(x.f <- gl(n = 3, k = 2))
#> [1] 1 1 2 2 3 3
#> Levels: 1 2 3
(x.f <- gl(n = 3, k = 2, labels=c("alap", "közép", "felső")))
#> [1] alap  alap  közép közép felső felső
#> Levels: alap közép felső
(x.f <- gl(n = 3, k = 2, labels=c("alap", "közép", "felső"), ordered=T))
#> [1] alap  alap  közép közép felső felső
#> Levels: alap < közép < felső
```

5.3.5.4. Értékek kizárása

A faktor létrehozásánál gondoskodhatunk bizonyos értékek kizárásáról, olyan értékekről, amelyeket nem szeretnénk a faktorban felsorolni.

```
# alapesetben az NA értékek ki lesznek zárva
factor(c(1:5, NA, 3:6))
#>  [1] 1    2    3    4    5    <NA> 3    4    5    6
#> Levels: 1 2 3 4 5 6
```

Alapértelmezés szerint az `NA` értéket zárjuk ki a faktor szintjeiből, de ezt megváltoztathatjuk az `exclude=` paraméter használatával:

```
# az NA értékek nem lesznek kizárva, ritkán van rá szükség
factor(c(1:5, NA, 3:6), exclude=NULL)
#>  [1] 1    2    3    4    5    <NA> 3    4    5    6
#> Levels: 1 2 3 4 5 6 <NA>
# az NA-n kívül más értékeket is ki tudunk zárni, most a 4-et
factor(c(1:5, NA, 3:6), exclude=c(4, NA))
#>  [1] 1    2    3    <NA> 5    <NA> 3    <NA> 5    6
#> Levels: 1 2 3 5 6
```

Ahogy látjuk a fenti példában, akár az `NA` értéket is bevonhatjuk a faktor szintjeibe, akár más értékeket is kizárhatunk az `NA`-n kívül.

5.3.5.5. Faktor indexelése és szűrése

Faktor indexelése a `[]` operátorral történik. Indexvektorként numerikus, karakteres és logikai vektorokat is használhatunk. Faktor indexelésének általános alakja:

```
faktor[indexvektor]     # az eredmény egy faktor
```

Hozzunk létre egy faktort, amely hat személy dohányzási szokását tartalmazza (“D”-dohányzik, “ND”-nem dohányzik).

```
(x.f <- factor(c("D", "D", "ND", "D", "ND", "ND"), levels = c("ND", "D")))
#> [1] D  D  ND D  ND ND
#> Levels: ND D
x.f[1]             # az x faktor 1. eleme (faktorszintek változatlanok)
#> [1] D
#> Levels: ND D
x.f[1, drop=T]     # az x faktor 1. eleme (faktorszintek változtak)
#> [1] D
```

```
#> Levels: D
x.f[1:3]           # az x faktor 1., 2. és 3. eleme
#> [1] D  D  ND
#> Levels: ND D
x.f[c(T, F)]       # az x faktor 1, 3. és 5. eleme
#> [1] D  ND ND
#> Levels: ND D
x.f[x.f == "D"]    # x szűrése (a dohányzók)
#> [1] D D D
#> Levels: ND D
x.f[x.f != "D"]    # x szűrése (a nem dohányzók)
#> [1] ND ND ND
#> Levels: ND D
```

Az indexelés eredménye minden esetben egy faktor lesz, amelynek szintjei alapesetben megegyeznek az eredeti faktor szintjeivel. A `drop=T` argumentum a nem használt címkéket eltávolítja a faktorszintek közül. Logika kifejezéseket is használhatunk az indexelés során, azaz szűrést is végezhetünk.

Indexelt faktor természetesen érékadás bal oldalán is szerepelhet. A faktor adatszerkezet megvéd minket az értékadások során, hiszen egy faktorelem csak a faktorszintekben szereplő értékek egyikét veheti fel.

```
x.f                        # az x.f faktor kiírása
#> [1] D  D  ND D  ND ND
#> Levels: ND D
x.f[1] <- "ND"             # az x.f faktor 1. eleme legális értéket kap
x.f[2] <- "nem dohányzik"  # az x.f faktor 2. eleme NA lesz
x.f
#> [1] ND   <NA> ND   D    ND   ND
#> Levels: ND D
```

Mivel a `"nem dohányzik"` címke nem szerepel a faktorszintek között, az `x.f` faktor 2. eleme `NA` lesz, egy figyelmeztető üzenet kíséretében.

5.3.5.6. Faktorok kezelése

A faktorok kényelmes használatát két további függvény segíti. Az `nlevels()` függvénnyel a faktorszintek számát ismerhetjük meg, a `levels()` függvénnyel pedig lekérdezhetők és módosíthatók a faktorszintek. Nézzünk egy példát az iskolai végzettséggel kapcsolatban.

Összesen 7 személyről tudjuk, hogy alap-, közép- vagy felsőfokú végzettségű, de az egyszerűbb rögzítés miatt indulásként ezt az információt számokkal kódoltuk (1-alap, 2-közép, 3-felső).

```
# numerikus vektor létrehozása
isk.vegz   <- c(1, 1, 2, 1, 3, 3, 2)
# faktor létrehozása
isk.vegz.f <- factor(isk.vegz, levels=c("1", "2", "3"))
isk.vegz.f                              # a faktor értéke
#> [1] 1 1 2 1 3 3 2
#> Levels: 1 2 3
nlevels(isk.vegz.f)                     # a faktor szintjeinek száma
#> [1] 3
levels(isk.vegz.f)                      # a faktor szintjei
#> [1] "1" "2" "3"
# a faktor szintjeinek módosítása
levels(isk.vegz.f) <- c("alap", "közép", "felső")
isk.vegz.f                              # a faktor értéke
#> [1] alap  alap  közép alap  felső felső közép
#> Levels: alap közép felső
```

Az `isk.vegz.f` faktort az `"1"`, `"2"` és `"3"` címkékkel hoztuk létre, de később a `levels()` függvénnyel beszédesebb faktorszinteket hoztunk létre.

Összefoglalás

A faktor olyan *integer* vektor, amely az 1-től sorszámozott értékeihez egy-egy karakteres címkét rendel. Ezek a címkék alkotják a faktorelemek lehetséges értékeit, amelyeket más néven faktorszinteknek is neveznek. A faktor létrehozásához a `factor()` függvényt használjuk és karakteres vektor konstansaiból vagy numerikus vektor címkeként kezelt számértékeiből jönnek létre a faktor lehetséges értékei. Rendezett faktorok szintjei között létezik egy természetes rendezettség, létrehozásukhoz az `ordered()` függvényt használjuk. Az `nlevels()` függvény a faktorszintek számát adja meg, míg a `levels()` a szintek nevének lekérdezését és módosítását szolgálja.

Feladatok

1. Hozzuk létre azt a karakteres vektort, amely a “férfi”, “nő” karakteres konstansokat, úgy helyezi el egymás mellett, hogy a 7 darab férfi érték után 13 db nő címke következik! Hány elemű az így létrejött vektor?
2. Egy vizsgálatban az első 10 személy neme “férfi”, a többi 8 neme “nő” volt. Hozzuk

létre azt a faktort, amely leírja a neme változót!
3. Egy vizsgálatban városi ("V") és falusi ("F") fiatalok vettek részt! A megkérdezettek településtípusa rendre a következő volt: F, F, V, F, V, V, V, F. Hozzuk létre azt a faktort, amely leírja a településtípus változót!
4. Egy vizsgálatban a dohányzási szokást egy kétértékű skálán mérték: 0-nem dohányzik; 1-dohányzik. A megkérdezettek dohányzási szokása a következő volt: 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0. Hozzuk létre azt a faktort, amely leírja a dohányzási szokás változót!

5.3.6. Lista

Az eddig megismert vektor, mátrix és faktor adatszerkezet mindegyike homogén volt, csak azonos típusú értékek tárolására használhatjuk őket. A lista típusú adatokban különböző adatszerkezetű elemeket is felsorolhatunk, de sem a típusra, sem a méretre nincs korlátozás. Egy listaelem lehet vektor, mátrix, faktor, adattábla vagy akár egy másik lista is (5.1. ábra). Látható, hogy a lista az R legszabadabb adatszerkezete, egydimenziós, és fő célja a logikailag összetartozó, de szerkezetileg különböző adatok tárolása.

5.3.6.1. Lista létrehozása

A `list()` függvénnyel hozhatunk létre legegyszerűbben listákat, itt vesszővel elválasztva kell megadnunk a lista elemeit.

```
x <- list(1:10, c("A", "B"), c = T)  # 3 elemű lista
x
#> [[1]]
#>  [1]  1  2  3  4  5  6  7  8  9 10
#>
#> [[2]]
#> [1] "A" "B"
#>
#> $c
#> [1] TRUE
```

A fenti példában `x` egy 3 elemű lista, az első eleme egy 10 elemű numerikus vektor, a második eleme egy 2 elemű karakteres vektor, a harmadik eleme pedig egy 1 elemű logikai vektor. A harmadik elemnek a `c` nevet adtuk, de bármelyik elemet elnevezhettük volna ezzel a módszerrel. Ha a lista értékét megjelenítjük a képernyőn, akkor a listaelemek egymás alatt

jelennek meg. Az első két esetben a kettős szögletes zárójelben (`[[]]`) lévő sorszám azonosítja a lista elemeit, a harmadik esetben pedig a listaelem általunk megadott neve a dollárjel (`$`) után.

A listaelemek nevét a `names()` függvénnyel kérdezhetjük le és állíthatjuk be.

```
names(x)                      # az x lista elemeinek neve
#> [1] ""  ""  "c"
names(x)[c(1,2)] <- c("a","b") # az x 1. és 2. elemének elnevezése
names(x)
#> [1] "a" "b" "c"
x
#> $a
#>  [1]  1  2  3  4  5  6  7  8  9 10
#>
#> $b
#> [1] "A" "B"
#>
#> $c
#> [1] TRUE
```

5.3.6.2. Lista indexelése

Egy lista indexelése a már megszokott `[]` indexoperátorral történik, amelyben továbbra is lehetőségünk van numerikus, karakteres és logikai indexvektor megadására is.

```
x[1]          # az x lista 1. elemét tartalmazó 1 elemű lista
#> $a
#>  [1]  1  2  3  4  5  6  7  8  9 10
x[c(2, 3)]    # az x lista 2. és 3. elemét tartalmazó 2 elemű lista
#> $b
#> [1] "A" "B"
#>
#> $c
#> [1] TRUE
x["a"]        # az x lista 1. elemét tartalmazó 1 elemű lista
#> $a
#>  [1]  1  2  3  4  5  6  7  8  9 10
x[c(T, F, T)] # az x lista 1. és 3. elemét tartalmazó 2 elemű lista
#> $a
```

```
#>  [1]  1  2  3  4  5  6  7  8  9 10
#>
#> $c
#> [1] TRUE
```

A `[]` operátorral kapott eredmény minden esetben lista, még akkor is, ha a lista egyetlen elemét érjük el. Nagyon fontos ettől megkülönböztetni a `[[]]` operátor eredményét, amely a lista valamelyik (egyetlen) elemével, annak az értékével tér vissza. Itt nincs mód több listaelem elérésére, és szokás szerint numerikus vagy karakteres értékkel indexelünk.

```
x[[1]]     # az x lista 1. eleme
#>  [1]  1  2  3  4  5  6  7  8  9 10
x[["b"]]   # az x lista 2. eleme
#> [1] "A" "B"
x[[3]]     # az x lista 3. eleme
#> [1] TRUE
```

A `[[]]` operátor alkalmazása helyett a rövidebb dollár (`$`) operátort is használhatjuk azoknak a listaelemeknek az elérésére, amelyeket korábban elneveztünk. A lista nevét és az elem nevét fűzzük össze a `$` operátorral.

```
x$a        # az x lista 1. eleme
#>  [1]  1  2  3  4  5  6  7  8  9 10
x$b        # az x lista 2. eleme
#> [1] "A" "B"
x$c        # az x lista 3. eleme
#> [1] TRUE
```

Ha a lista elemét valamelyik módszer segítségével elértük, akkor további indexelés segítségével az elem összetevőit is lekérdezhetjük vagy módosíthatjuk.

```
x[["a"]][3:4]       # az x lista 1. elemének 3. és 4. eleme
#> [1] 3 4
x$a[4:5] <- 0       # az x lista 1. elemének 4. és 5. eleme 0 lesz
x$c <- 1:2          # az x lista 3. elemének módosítása
x
#> $a
#>  [1]  1  2  3  0  0  6  7  8  9 10
#>
```

```
#> $b
#> [1] "A" "B"
#>
#> $c
#> [1] 1 2
```

A lista indexelésére tehát a következő lehetőségek állnak rendelkezésre:

```
lista[indexvektor]     # az eredmény egy lista
lista[[index]]         # az eredmény a lista egy eleme
lista$elemnév          # az eredmény a lista egy eleme
```

5.3.6.3. Művelet a listaelemekkel

Egy lista minden elemével az `lapply()` vagy az `sapply()` függvény segítségével hajthatunk végre műveletet.

```
lapply(X=x, FUN=length) # az x lista minden elemének a hossza egy listába
#> $a
#> [1] 10
#>
#> $b
#> [1] 2
#>
#> $c
#> [1] 2
sapply(X=x, FUN=length) # az x lista minden elemének a hossza egy vektorba
#>  a  b  c
#> 10  2  2
```

Az `lapply()` a bemenő lista elemszámával egyező méretű listával tér vissza, melynek értékei a második paraméterben szereplő függvény visszatérési értékei. Az `sapply()` hasonlóan jár el, de a visszatérési értéke egy vektor lesz.

> Összefoglalás
>
> A lista az R legszabadabb adatszerkezete, egydimenziós és inhomogén. Listát a `list()` függvénnyel hozhatunk létre, melynek argumentumában tetszőleges adatszerkezetű objektumokat felsorolhatunk, ezek alkotják a lista egyes elemeit. Lista indexelése a `[]`,

`[[]]` és `$` operátorokkal is lehetséges. A lista minden elemén végrehajthatjuk ugyanazt a műveletet az `lapply()` vagy az `sapply()` függvények segítségével. Az `lapply()` a bemenő lista elemszámával egyező méretű listával tér vissza, míg az `sapply()` hasonlóan jár el, de a visszatérési értéke egy vektor lesz.

🎯 Feladatok

1. Hozzunk létre egy háromelemű listát a TRUE, 12, és "Verseny" konstansokból!
2. Hozzunk létre egy háromelemű listát a TRUE, 12, és "Verseny" konstansokból, de gondoskodjunk az egyes elemek elnevezéséről, amelyek legyenek rendre: "befejezve", "indulok" és "leiras"!
3. Hozzunk létre egy háromelemű listát a TRUE, 12 és "Verseny" konstansokból, valamint az induló versenyzők végső pontszámaiból, amelyek rendre: 89, 78, 23, 67, 99, 69, 85, 77, 58, 72, 48, 81. Gondoskodjunk az egyes elemek elnevezéséről, amelyek legyenek rendre: "befejezve", "indulok", "leiras" és "pontszam"!

5.3.7. Adattábla

Az *adattábla* (*data frame*) az R legfontosabb adatszerkezete, központi szerepet játszik az adatfeldolgozásban, lényegében minden statisztikai munka kiindulópontja. Inhomogén, kétdimenziós szerkezet, sorok és oszlopok alkotják, alapvetően azonos hosszúságú vektorokból és faktorokból épül fel (5.1. ábra). Az adattábla egyesíti a mátrix és a lista adatszerkezet előnyeit. Az adattábla kétdimenziós, mint a mátrix, és inhomogén, mint a lista. Ha mátrixként tekintünk az adattáblára, akkor sorokból és oszlopokból áll, ha listaként, akkor azonos hosszúságú (oszlop)vektorok/faktorok egydimenziós sorozata.

5.3.7.1. Adattábla létrehozása

Adattáblát legegyszerűbben a `data.frame()` függvénnyel hozhatunk létre, amely azonos hosszú vektorokat vagy faktorokat vár az argumentumában. A `data.frame()` tehát listaszerűen konstruálja az adattáblát.

```
# egyszerű adattábla létrehozása
df <- data.frame(
  nev  = c("Péter", "Éva", "Lajos"),
  pont = c(34, 32, 29)
)
df      # adattábla kiírása
```

```
#>     nev pont
#> 1 Péter   34
#> 2   Éva   32
#> 3 Lajos   29
```

A fenti `df` adattáblát egy 3 elemű karakteres vektorból, és egy 3 elemű numerikus vektorból hoztuk létre. A `data.frame()` függvénynek ezt a két vektort adtuk meg, ennek megfelelően két oszlopa lesz az adattáblának. Mindkét vektor 3 elemű, így 3 sor lesz a `df`-ben. Adattáblánk így 3 × 2-es. Mindkét argumentumot elneveztük (`nev`, `pont`), ezekből oszlopnevek lesznek. Az oszlopok elnevezéséhez az objektumneveknél használt szabályokat vegyük figyelembe (5.1.2.3. fejezet), és ne használjunk ékezetes karaktereket és szóközt. A fenti outputból kiolvasható, hogy az adattábla sornevekkel is rendelkezik, ezek automatikusan jönnek létre 1-től kezdődő sorszámmal.

Ha a `data.frame()` függvényben a paraméterek hossza nem azonos, akkor a rövidebb vektorok és faktorok ismétléssel kiegészülnek a leghosszabb oszlop hosszára. Az ismétlés azonban csak egész számszor lehetséges, egyébként hibaüzenetet kapunk.

```
# adattábla létrehozása, ismétlődő "tipus" és "x" értékekkel
tipus <- factor(c('A','B')); x <- 6:8; y <- 1:6
df2 <- data.frame(
   tipus,
   pont.1=x,
   pont.2=y
)
df2
#>   tipus pont.1 pont.2
#> 1     A      6      1
#> 2     B      7      2
#> 3     A      8      3
#> 4     B      6      4
#> 5     A      7      5
#> 6     B      8      6
```

A példában egy 6 sorból és 3 oszlopból álló adattáblát készítettünk (`df2` 6 × 3-as). A `data.frame()` függvényben nem azonos a `tipus` faktor és a két numerikus vektor (`x`, `y`) hossza, így ismétléssel kapjuk meg a fenti eredményt. Továbbá, ha elhagyjuk az argumentum nevét, akkor az oszlopnév a megfelelő objektum neve alapján jön létre. Így kapta az első oszlop a `tipus` nevet.

5.3.7.2. Adattábla felépítése

Adattábláink ritkán olyan kicsik, mint a fenti `df` vagy `df2`. Sokszor több tucat sorból és oszlopból állnak, így az adattábla áttekintésére nem az adattáblát tároló objektum értékének képernyőre írása a legszerencsésebb. Kényelmesebb, ha az *RStudio* adatbázis ablakában jelenítjük meg az adattábla tartalmát, amit a *Környezet* panel megfelelő adatbázisnevén való kattintással és vagy a `View()` paranccsal kezdeményezhetünk. Próbáljuk ki a `View(df)` és `View(df2)` függvényhívásokat.

Hasznos információt szolgáltat az `str()` függvény is, amely az adattábla szerkezetéről ad felvilágosítást.

```
str(df)     # a df adattábla szerkezete
#> 'data.frame':    3 obs. of  2 variables:
#>  $ nev : chr  "Péter" "Éva" "Lajos"
#>  $ pont: num  34 32 29
```

Láthatjuk, hogy a `df` adattáblánk 3 sort (megfigyelést) és 2 oszlopot (változót) tartalmaz, valamint leolvashatjuk az egyes oszlopok típusát is. Megfigyelhetjük, hogy a `nev` oszlop karakteres, a `pont` pedig numerikus vektor.

Láttuk korábban, hogy az adattábla sorai és oszlopai névvel is rendelkeznek.

```
names(df); colnames(df)  # oszlopnevek
#> [1] "nev"  "pont"
#> [1] "nev"  "pont"
rownames(df)             # sornevek
#> [1] "1" "2" "3"
```

A `rownames()` a sorok nevét, a `colnames()` és a `names()` az oszlopok nevét írja ki, de segítségükkel ezeket módosíthatjuk is. A sorok és oszlopok nevének meghatározásánál ügyeljünk arra, hogy azok minden esetben legyenek egyediek. Két azonos sornév létrehozása hibaüzenethez vezet, de az azonos oszlopnevek használatát is kerüljük.

```
rownames(df) <- paste0(1:3, ".szemely")   # sornevek módosítása
names(df) <- c("X","Y")                   # oszlopnevek módosítása
df
#>              X  Y
#> 1.szemely Péter 34
#> 2.szemely   Éva 32
#> 3.szemely Lajos 29
```

A `length()` függvény az oszlopok számával tér vissza. Az `nrow()` és az `ncol()` a sor és oszlopok számával tér vissza.

```
length(df); ncol(df)   # oszlopok száma
#> [1] 2
#> [1] 2
nrow(df)               # sorok száma
#> [1] 3
```

5.3.7.3. Adattábla indexelése

Az adattáblák indexelése a mátrixok és a listáknál megtanult indexelési formákat jelentik. Az általános indexelési formák a következők:

```
adattábla[sorindexvektor, oszlopindexvektor] # adattábla, vektor vagy faktor
adattábla[oszlopindexvektor]                 # adattábla
adattábla$oszlopnév                          # vektor vagy faktor
```

A mátrixokhoz hasonlóan indexelhetjük a sorokat és az oszlopokat, hiszen az adattábla kétdimenziós. A `[]` operátorban szerepel egy vessző, amely a sor- és oszlopkoordinátákat választja el egymástól. Használhatjuk a következő hivatkozásokat:

```
df2               # a df2 adattábla kiírása
#>   tipus pont.1 pont.2
#> 1     A      6      1
#> 2     B      7      2
#> 3     A      8      3
#> 4     B      6      4
#> 5     A      7      5
#> 6     B      8      6
df2[2, 3]         # a df2 2. sorában a 3. oszlop adata, vektor eredmény
#> [1] 2
df2[c(2, 3), 3]   # a df2 2. és 3. sorában a 3. oszlop adata, vektor
#> [1] 2 3
df2[c(2, 3), 1:2] # a df2 2. és 3. sorában a 1. és 2. oszlop adata, adattábla
#>   tipus pont.1
#> 2     B      7
#> 3     A      8
df2[c(2, 3), ]    # a df2 2. és 3. sora, adattábla
```

```
#>   tipus pont.1 pont.2
#> 2     B      7      2
#> 3     A      8      3
df2[2, ]          # a df2 2. sora, adattábla
#>   tipus pont.1 pont.2
#> 2     B      7      2
df2[, 3]          # a df2 3. oszlopa, vektor
#> [1] 1 2 3 4 5 6
df2[, 3, drop=F]  # a df2 3. oszlopa, adattábla
#>   pont.2
#> 1      1
#> 2      2
#> 3      3
#> 4      4
#> 5      5
#> 6      6
df2[, 1:2]        # a df2 1. és 2. oszlopa, adattábla
#>   tipus pont.1
#> 1     A      6
#> 2     B      7
#> 3     A      8
#> 4     B      6
#> 5     A      7
#> 6     B      8
```

Numerikus indexvektorok mellett használhatunk karakteres és logikai vektorokat is indexelésre.

```
df2[, c("tipus", "pont.1")]         # minden sor, 1. és 2. oszlop
#>   tipus pont.1
#> 1     A      6
#> 2     B      7
#> 3     A      8
#> 4     B      6
#> 5     A      7
#> 6     B      8
df2[c(T, F), c("tipus", "pont.1")]  # páratlan sorok 1. és 2. oszlop
#>   tipus pont.1
#> 1     A      6
```

```
#> 3     A      8
#> 5     A      7
```

Karakteres vektorok tipikusan oszlopindexekben fordulnak elő, logikai vektorok pedig, később látjuk, az adattábla szűrésénél kapnak fontos szerepet.

Ha az adattáblára listaként tekintünk, akkor `[]` operátorban egyetlen indexvektort is szerepeltethetünk, amely az adattábla oszlopait indexeli, és minden esetben adattáblát szolgáltat, még akkor is, ha az adattábla egyetlen oszlopát érjük el.

```
df2[2]                    # a df2 2. oszlopa, adattábla
#>   pont.1
#> 1      6
#> 2      7
#> 3      8
#> 4      6
#> 5      7
#> 6      8
df2[1:2]                  # a df2 1. és 2. oszlopa, adattábla
#>   tipus pont.1
#> 1     A      6
#> 2     B      7
#> 3     A      8
#> 4     B      6
#> 5     A      7
#> 6     B      8
df2["tipus"]              # a df2 1. oszlopa, adattábla
#>   tipus
#> 1     A
#> 2     B
#> 3     A
#> 4     B
#> 5     A
#> 6     B
df2[c("tipus", "pont.2")] # a df2 1. és 3. oszlopa, adattábla
#>   tipus pont.2
#> 1     A      1
#> 2     B      2
#> 3     A      3
#> 4     B      4
```

```
#> 5     A      5
#> 6     B      6
```

Az adattábla egyes oszlopai a `$` operátorral is elérhetők, amely az adattábla nevét és az oszlop nevét választja el egymástól. Az eredmény minden esetben vektor vagy faktor lesz.

```
df2$tipus               # a df2 1. oszlopa, faktor
#> [1] A B A B A B
#> Levels: A B
df2$pont.1              # a df2 2. oszlopa, vektor
#> [1] 6 7 8 6 7 8
```

Az adattábla indexelése után kapott adatszerkezetek tovább indexelhetők. Attól függően, hogy a kiinduló adattábla indexelésével kapott adatszerkezet egy- vagy kétdimenziós használhatjuk a `[]` és `$` operátorokat is.

```
df2[4:1, 1:2][2]          # df2-ből adattábla, majd adattábla
#>   pont.1
#> 4      6
#> 3      8
#> 2      7
#> 1      6
df2[4:1, 1:2]$tipus       # df2-ből adattábla, majd faktor
#> [1] B A B A
#> Levels: A B
df2$pont.2[1:3]           # df2-ből vektor, majd vektor
#> [1] 1 2 3
```

Ne felejtsük el, hogy adattábla indexelése során a lekért elemek módosítására is lehetőségünk van, és a vektoraritmetika szabályai továbbra is teljesülnek.

```
df2[2, 3] <- 200              # egyetlen érték módosítása
df2$pont.2 <- df2$pont.2 + 1  # teljes oszlop módosítása
df2                           # df2 kiírása
#>   tipus pont.1 pont.2
#> 1     A      6      2
#> 2     B      7    201
#> 3     A      8      4
#> 4     B      6      5
#> 5     A      7      6
#> 6     B      8      7
```

5.3.7.4. Adattáblák szűrése

Az adattábla indexelésénél logikai vektorokat is használhatunk sorindexvektorban, melyek az adattábla tartalmára vonatkozó relációs kifejezések is lehetnek. Ezzel a módszerrel érhetjük el, hogy az adattábla sorait valamilyen szempont szerint leválogassuk, megszűrjük.

```
df2[df2$tipus == "A", ]                    # az A típusú sorok leválogatása
#>   tipus pont.1 pont.2
#> 1     A      6      2
#> 3     A      8      4
#> 5     A      7      6
df3 <- df2[df2$pont.1<8 & df2$pont.2>2, 2:3] # összetett logikai kifejezés
```

Az első szűrésünk az adattábla `"A"` címkékkel rendelkező sorait válogatta le, de csak képernyőn olvashatók ezek a sorok. A második szűrés eredményét azonban megőrizzük egy új `df3` objektumban, és látható, hogy a `pont.1` és a `pont.2` numerikus vektorokra vonatkozó összetett logikai kifejezéssel végezzük.

5.3.7.5. Adattáblák sorainak rendezése

Az adattábla sorainak rendezése a vektoroknál megismert `order()` függvény és a `[]` operátor kombinált alkalmazásával lehetséges. Rendezzük a `pont.1` változó alapján a `df2` sorait.

```
df2[order(df2$pont.1), ] # df2 sorainak rendezése pont.1 növekvő sorrendjében
#>   tipus pont.1 pont.2
#> 1     A      6      2
#> 4     B      6      5
#> 2     B      7    201
#> 5     A      7      6
#> 3     A      8      4
#> 6     B      8      7
```

Az `order()` függvény `decreasing=TRUE` argumentumával csökkenő sorrendet is elérhetünk. Az `order()` függvény több oszlopot is képes fogadni, így több oszlop alapján is tudunk sorokat rendezni.

```r
# df2 sorainak rendezése pont.1 és pont.2 csökkenő sorrendjében
df2[order(df2$pont.1, df2$pont.2, decreasing=T), ]
#>   tipus pont.1 pont.2
#> 6     B      8      7
#> 3     A      8      4
#> 2     B      7    201
#> 5     A      7      6
#> 4     B      6      5
#> 1     A      6      2
```

Összefoglalás

Az adattábla minden statisztikai munka kiindulópontja. Kétdimenziós, inhomogén szerkezet, de mivel azonos hosszú vektorok vagy faktorok listájának is tekinthető, oszlopaiban homogén adatszerkezet. Létrehozása a `data.frame()` függvénnyel lehetséges, ahol az argumentumban az oszlopokat alkotó vektorokat és faktorokat kell felsorolni. Az adattábla indexelése a mátrixoknál és a listáknál tanultak alapján lehetséges. Tipikus alakja az `adattábla[sorindexvektor, oszlopindexvektor]`, `adattábla[oszlopindexvektor]` és `adattábla$oszlopnév` formájú szerkezetek. Adattábla szűrése a sorkoordinátákban szereplő logikai kifejezések segítségével történik, a rendezés pedig a szintén sorkoordinátákba írt `order()` függvénnyel.

Feladatok

1. Hozzunk létre egy 30 × 3-as adattáblát, `csoport`, `matematika` és `magyar` oszlopnevekkel. A `csoport` változó legyen egy `5.a`, `5.b` és `5.c` címkéket tetszőleges sorrendben tartalmazó faktor, a `matematika` és a `magyar` pedig 1-5 osztályzatokat tartalmazó numerikus vektor.
2. Írassuk ki a `{MASS}` csomag `survey` adattáblájának 3. sorában az 5. oszlopban lévő értéket!
3. Írassuk ki a `{MASS}` csomag `survey` adattáblájának 3. és 6. sorában az 5. oszlopban lévő értékeket! Az adattábla típus maradjon meg!
4. Írassuk ki a `{MASS}` csomag `survey` adattáblájának 3. és 6. sorából az összes adatértéket!
5. Írassuk ki a `{MASS}` csomag `survey` adattábla `Pulse` oszlopát háromféle módszerrel!
6. Írassuk ki a `{MASS}` csomag `survey` adattábla `Pulse` változójának első 3 elemét háromféle módszerrel!
7. A `{HSAUR3}` csomag `Forbes2000` adattáblája 2000 vállalat adatát tartalmazza! Határozzuk meg a magyar cégek nevét és helyezését (`country` oszlop alapján)!

Írassuk ki a képernyőre a 10 legnagyobb piaci értékkel (`marketvalue` oszlop) rendelkező cég nevét és piaci értékét! Határozzuk meg a legkisebb profittal (`profits` oszlop) rendelkező 5 cég minden adatát! Határozzuk meg a legnagyobb profittal (`profits` oszlop) rendelkező 10 amerikai vagy japán cég nevét, országát és profitját!

5.4. További adatszerkezetek 😉

i Miről lesz szó? Ebben a fejezetben

- megismerjük a *tömb*, *táblázat*, *dátum*, *idő*, *időtartam* és *tibble* adatszerkezeteket,
- valamint a munkaterület és munkakönyvtár kezelésének függvényeit.

Az R legfontosabb adatszerkezeteit megismertük az előző fejezetben. Az adatelemzés kiindulópontja az *adattábla*, amely a *mátrix* és a *lista* adatszerkezet előnyeit egyesíti, lényegében *vektor*ok és *faktor*ok egymásutánja. A munka során azonban találkozhatunk három vagy több dimenzióba szervezett adatokkal (*tömb* és *táblázat*), valamint szükség lehet *dátum*, *idő* és *időtartam* kezelésére is. Sőt, a *Tidyverse R* megújította az adattáblát, és bevezette a saját *tibble* típusát az adatmátrixok tárolására. Definiáljuk pontosabban a fenti, új adatszerkezeteket:

- *tömb* - azonos alaptípusú értékekből 3 vagy több dimenzió mentén készítünk adatszerkezetet.
- *táblázat* - a gyakorisági táblázatok R megfelelője, amelyben tipikusan *integer* adatokat rögzítünk, egy, két vagy több dimenzió mentén.
- *dátum* - egyetlen *double* érték, amelynek jelentése az `1970-01-01` óta eltelt napok száma.
- *dátum-idő* - egyetlen *double* érték, amelynek jelentése az `1970-01-01` óta eltelt másodpercek száma.
- *időtartam* - egyetlen *double* érték, amely különböző mértékegységekben mutatja két időpont közötti különbséget.
- *tibble* - speciális adattábla, amely a *Tidyverse R* része, és megkönnyíti az adatok kezelését.

Az 5.8. táblázatban már korábban bemutattuk az R legfontosabb adatszerkezeteit, az 5.10. táblázat azokat az új adatszerkezeteket sorolja fel, amelyeket ebben a fejezetben mutatunk be. Most is közöljük, hogy a `typeof()` és a `class()` milyen outputot szolgáltat az egyes adatszerkezetek esetén.

5.10. táblázat: Adatszerkezetek (folytatás) *(saját szerkesztés)*

Adatszerkezet	Létrehozó parancs	typeof(x)	class(x)
integer tömb	`array(2L,dim=c(2,3,5))`	`integer`	`array`
double tömb	`array(2,dim=c(2,3,5))`	`double`	`array`
karakteres tömb	`array('a',dim=c(2,3,5))`	`character`	`array`
logikai tömb	`array(T,dim=c(2,3,5))`	`logical`	`array`
táblázat	`table(sample(1:10, 100, T))`	`integer`	`table`
dátum	`as.Date('1971-05-09')`	`double`	`Date`
dátum-idő	`as.POSIXct('2018-08-01 22:00','UTC')`	`double`	`POSIXct POSIXt`
időtartam	`as.difftime(7,units='days')`	`double`	`difftime`
tibble	`tibble(x=1:3,y=letters[1:3])`	`list`	`tbl_df tbl data.frame`

5.4.1. Tömbök és táblázatok

A *tömb* a mátrix általánosításával nyerhető adatszerkezet. Az azonos típusú adatokat a mátrix két dimenzió mentén rendezi össze. Azonban három vagy több dimenzió mentén is elvégezhető ez az összerendezés. Így nyerjük a három vagy több dimenziós tömböket. A mátrix két dimenziós tömbnek is tekinthető (vagy a vektor egy egy dimenziós tömbnek). A *táblázat* a tömbökhöz nagyon hasonló adatszerkezet, de tipikusan számlálással nyert *integer* értékeket rögzítünk bennük. A tömbökhöz hasonlóan lehetnek egy, két, vagy több dimenziósak.

5.4.1.1. Tömb létrehozása és indexelése

Az `array()` függvénnyel egyszerűen hozhatunk létre tömböt. A függvény a `data=` argumentumban megadott vektor elemeit a `dim=` argumentumban megadott dimenzió-méretek mentén rendezi össze.

```
x <- array(data=1:12, dim=c(2, 3, 2)) # az 1:12 vektorból 3 dimenziós tömb
x
#> , , 1
#>
#>      [,1] [,2] [,3]
#> [1,]    1    3    5
#> [2,]    2    4    6
#>
#> , , 2
#>
```

```
#>      [,1] [,2] [,3]
#> [1,]    7    9   11
#> [2,]    8   10   12
```

A háromdimenziós *integer* tömb 2 × 3 × 2-es, azaz 2 sorból, 3 oszlopból és 2 lapból áll. Természetesen *double*, *karakteres* és *logikai* tömbök is hasonló módszerrel hozhatók létre, csak a `data=` értéket kell megfelelően megválasztani.

A tömb kiíratása során az indexoperátorokban (`[]`) szereplő sorszámok segítségével igazodhatunk el az elemek között. A háromdimenziós `x` tömb dimenziói a sorok, oszlopok és a lapok. A 12 elemet két lapon a `, , 1` és a `, , 2` nevű lapokon, két-két sorba `[1, ]`, `[2, ]` és három-három oszlopba `[ ,1]`, `[ ,2]`, `[ ,3]` rendezve sorolja fel az R. A második lapon a 2. sor 1. eleméhez meg kell találnunk a `, , 2` lapot, a `[2, ]` sort és az `[ ,1]` oszlopot, ami esetünkben a 8.

A tömbök indexelése a mátrixokhoz hasonló, csak a dimenziószámnak megfelelő számú indexvektort kell használnunk. Ha `x` 3 dimenziós, akkor az `x[1,3,2]` egy lehetséges példa indexelésére, ahol az első sor harmadik oszlopában lévő elemre gondolunk, a második lapról. Emlékezhetünk, hogy kétdimenziós mátrixok esetén csak a sor és oszlop azonosító indexekre volt szükségünk (például `x[2,3]`), míg 4 vagy afeletti dimenziószámok esetén természetesen 4 vagy több, vesszővel elválasztott indexet kell megadnunk.

5.4.1.2. Táblázat létrehozása

Táblázatokat a `table()` függvénnyel hozhatunk létre, tipikusan kategorikus adatokból, vagyis faktor típusú objektumokból. A `{MASS}` csomag `survey` adattáblája több faktor oszlopot is tartalmaz, ezt használjuk a továbbiakban.

```
data("survey", package = "MASS")  # a survey betöltése
str(survey)
#> 'data.frame':    237 obs. of  12 variables:
#>  $ Sex   : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 1 2 1 2 2 ...
#>  $ Wr.Hnd: num  18.5 19.5 18 18.8 20 18 17.7 17 20 18.5 ...
#>  $ NW.Hnd: num  18 20.5 13.3 18.9 20 17.7 17.7 17.3 19.5 18.5 ...
#>  $ W.Hnd : Factor w/ 2 levels "Left","Right": 2 1 2 2 2 2 2 2 2 2 ...
#>  $ Fold  : Factor w/ 3 levels "L on R","Neither",..: 3 3 1 3 2 1 1 3 3 3 ...
#>  $ Pulse : int  92 104 87 NA 35 64 83 74 72 90 ...
#>  $ Clap  : Factor w/ 3 levels "Left","Neither",..: 1 1 2 2 3 3 3 3 3 3 ...
#>  $ Exer  : Factor w/ 3 levels "Freq","None",..: 3 2 2 2 3 3 1 1 3 3 ...
#>  $ Smoke : Factor w/ 4 levels "Heavy","Never",..: 2 4 3 2 2 2 2 2 2 2 ...
```

```
#>  $ Height: num  173 178 NA 160 165 ...
#>  $ M.I   : Factor w/ 2 levels "Imperial","Metric": 2 1 NA 2 2 1 1 2 2 2 ...
#>  $ Age   : num  18.2 17.6 16.9 20.3 23.7 ...
```

Egydimenziós gyakorisági táblázat létrehozásához egyetlen faktort használunk a `table()` argumentumában. Érdemes a `useNA="ifany"` argumentumot is használni, amely a faktorban lévő hiányzó értékek számát adja meg, amennyiben van hiányzó érték a változóban.

```
table(survey$Sex, useNA = "ifany")  # egydimenziós gyakorisági táblázat
#>
#> Female   Male   <NA>
#>    118    118      1
```

Az output első sorában az egydimenziós táblázat (*integer* vektor) elemeinek a nevét olvashatjuk, melyek a `Sex` faktor lehetséges értékeit és a hiányzó értékek címkéjét jelentik. A táblázat második sorában lévő számok az egyes címkék előfordulási gyakoriságát jelentik a faktorban. Ebben a kutatásban (`?survey`) 118 nőt és 118 férfit kérdeztek meg, egyetlen személynek nem ismerjük a nemét.

Kétdimenziós gyakorisági táblázat készítéséhez két faktorra van szükség. A nem (`Sex`) mellett a kezességet (`W.Hnd`) is bevontuk a vizsgálatba:

```
# kétdimenziós gyakorisági táblázat
table(survey$Sex, survey$W.Hnd, useNA = "ifany")
#>
#>          Left Right <NA>
#>   Female    7   110    1
#>   Male     10   108    0
#>   <NA>      1     0    0
```

A kétdimenziós gyakorisági táblázat (*integer* mátrix) sornevei és oszlopnevei segítenek értelmezni a gyakorisági értékeket. A 7 például a balkezes nők számát jelenti a mintában.

Három vagy magasabb dimenziószámú táblázatokat is hasonlóan készíthetünk: egyre több faktort vonunk be a `table()` függvénybe. Háromdimenziós gyakorisági táblázatra mutatunk példát az `Exer` faktor bevonásával.

```
# háromdimenziós gyakorisági táblázat
table(survey$Sex, survey$W.Hnd, survey$Exer, useNA = "ifany")
#> , ,  = Freq
#>
#>
#>          Left Right <NA>
#>   Female    3    45    1
#>   Male      3    62    0
#>   <NA>      1     0    0
#>
#> , ,  = None
#>
#>
#>          Left Right <NA>
#>   Female    1    10    0
#>   Male      2    11    0
#>   <NA>      0     0    0
#>
#> , ,  = Some
#>
#>
#>          Left Right <NA>
#>   Female    3    55    0
#>   Male      5    35    0
#>   <NA>      0     0    0
```

A háromdimenziós vagy afeletti táblázatok esetében az `ftable()` kétdimenziós ábrázolással segíti a gyakorisági adatok értelmezését.

```
tab3 <- table(survey$Sex, survey$W.Hnd, survey$Exer)
ftable(tab3)  # háromdimenziós táblázat két dimenzióban
#>               Freq None Some
#>
#> Female Left      3    1    3
#>        Right    45   10   55
#> Male   Left      3    2    5
#>        Right    62   11   35
tab4 <- table(survey$Sex, survey$W.Hnd, survey$Exer, survey$Smoke)
ftable(tab4)  # négydimenziós táblázat két dimenzióban
#>                   Heavy Never Occas Regul
```

```
#>
#> Female Left  Freq      0     2     1     0
#>               None      0     1     0     0
#>               Some      0     3     0     0
#>        Right Freq      3    36     4     2
#>               None      0     9     1     0
#>               Some      2    47     3     3
#> Male   Left  Freq      0     2     1     0
#>               None      0     1     0     1
#>               Some      1     3     1     0
#>        Right Freq      4    45     6     7
#>               None      1     7     2     0
#>               Some      0    31     0     4
```

A `table()` függvény helyett használhatjuk az `xtabs()` függvényt is, amely támogatja a kicsit kényelmesebb formula argumentumot. Az R formula olyan kifejezés, amely tartalmaz egy tilde (~) karaktert, és annak két oldalán rendszerint egy adattábla oszlopnevei jelennek meg. A `table()` és az `xtabs()` általános használata a következő:

```
table(df$változó_1, df$változó_2, ..., df$változó_n)
xtabs(~változó_1 + változó_2 + ... + változó_n, data=df)
```

Az `xtabs()` használatára mutatunk 3 példát. Figyeljük meg, hogy a hiányzó értékek megjelenítéséhez itt az `addNA=T` argumentumot kell használnunk. Az `xtabs()` függvény speciális formulájának bal oldala üres, jobb oldalán pedig a faktor változók `+` karakterrel vannak összekapcsolva.

```
xtabs(~Sex, data=survey, addNA = T)             # 1D gyakorisági táblázat
#> Sex
#> Female   Male   <NA>
#>    118    118      1
xtabs(~Sex+W.Hnd, data=survey, addNA = T)       # 2D gyakorisági táblázat
#>         W.Hnd
#> Sex      Left Right <NA>
#>   Female    7   110    1
#>   Male     10   108    0
#>   <NA>      1     0    0
xtabs(~Sex+W.Hnd+Exer, data=survey, addNA = T)  # 3D gyakorisági táblázat
#> , , Exer = Freq
#>
```

```
#>         W.Hnd
#> Sex      Left Right <NA>
#>   Female    3    45    1
#>   Male      3    62    0
#>   <NA>      1     0    0
#>
#> , , Exer = None
#>
#>         W.Hnd
#> Sex      Left Right <NA>
#>   Female    1    10    0
#>   Male      2    11    0
#>   <NA>      0     0    0
#>
#> , , Exer = Some
#>
#>         W.Hnd
#> Sex      Left Right <NA>
#>   Female    3    55    0
#>   Male      5    35    0
#>   <NA>      0     0    0
```

5.4.1.3. Táblázatok átalakítása

Korábban megismertük az `as.*()` kezdetű függvényeket, amelyek egyszerű típuskonverziót végeznek. A gyakorisági táblázatokat gyakran szeretnénk vektor, mátrix, tömb, vagy még gyakrabban adattábla típusban rögzíteni. Ezek az átalakítások az `as.vector()`, `as.matrix()`, `as.array()`, valamint az `as.data.frame()` függvénnyel könnyen elvégezhetők.

```
tab1 <- table(survey$Sex, useNA = "ifany")       # 1D gyakorisági táblázat
tab2 <- table(survey$Sex, survey$W.Hnd, useNA = "ifany")              # 2D
tab3 <- table(survey$Sex, survey$W.Hnd, survey$Exer, useNA = "ifany") # 3D
(vekt <- as.vector(tab1))     # 1D táblázatból vektor
#> [1] 118 118   1
(mat <- as.matrix(tab2))      # 2D táblázatból mátrix
#>
#>         Left Right <NA>
#>   Female    7   110    1
#>   Male     10   108    0
```

```
#>   <NA>      1     0    0
(tomb <- as.array(tab3))      # 3D táblázatból 3D tömb
#> , ,  = Freq
#>
#>
#>          Left Right <NA>
#>   Female    3    45    1
#>   Male      3    62    0
#>   <NA>      1     0    0
#>
#> , ,  = None
#>
#>
#>          Left Right <NA>
#>   Female    1    10    0
#>   Male      2    11    0
#>   <NA>      0     0    0
#>
#> , ,  = Some
#>
#>
#>          Left Right <NA>
#>   Female    3    55    0
#>   Male      5    35    0
#>   <NA>      0     0    0
(df1 <- as.data.frame(tab1))  # 1D táblázatból adattábla
#>     Var1 Freq
#> 1 Female  118
#> 2   Male  118
#> 3   <NA>    1
(df2 <- as.data.frame(tab2))  # 2D táblázatból adattábla
#>     Var1  Var2 Freq
#> 1 Female  Left    7
#> 2   Male  Left   10
#> 3   <NA>  Left    1
#> 4 Female Right  110
#> 5   Male Right  108
#> 6   <NA> Right    0
#> 7 Female  <NA>    1
#> 8   Male  <NA>    0
```

```
#> 9   <NA>  <NA>    0
(df3 <- as.data.frame(tab3))  # 3D táblázatból adattábla
#>      Var1  Var2 Var3 Freq
#> 1  Female  Left Freq    3
#> 2    Male  Left Freq    3
#> 3    <NA>  Left Freq    1
#> 4  Female Right Freq   45
#> 5    Male Right Freq   62
#> 6    <NA> Right Freq    0
#> 7  Female  <NA> Freq    1
#> 8    Male  <NA> Freq    0
#> 9    <NA>  <NA> Freq    0
#> 10 Female  Left None    1
#> 11   Male  Left None    2
#> 12   <NA>  Left None    0
#> 13 Female Right None   10
#> 14   Male Right None   11
#> 15   <NA> Right None    0
#> 16 Female  <NA> None    0
#> 17   Male  <NA> None    0
#> 18   <NA>  <NA> None    0
#> 19 Female  Left Some    3
#> 20   Male  Left Some    5
#> 21   <NA>  Left Some    0
#> 22 Female Right Some   55
#> 23   Male Right Some   35
#> 24   <NA> Right Some    0
#> 25 Female  <NA> Some    0
#> 26   Male  <NA> Some    0
#> 27   <NA>  <NA> Some    0
```

Az ellenkező irányú átalakítás is érdekes lehet, vagyis amikor egy-, két- vagy háromdimenziós tömbökből gyakorisági táblázatot képezünk (`as.table()` függvény), de főképp amikor az adattáblában létező gyakorisági adatokat táblázattá alakítjuk. Itt érdemes az `xtabs(Freq~Változó_1+Változó_2+...+Változó_n, data=df)` függvényhívást használni, ahol a tilde (~) előtti oszlop az adattábla gyakorisági adatait tartalmazza, a jobbra lévő változók pedig lényegében a faktor változókat nevezik meg.

```
as.table(vekt) # vektorból 1D táblázat
#>   A   B   C
```

```
#> 118 118   1
as.table(mat)  # mátrixból 2D táblázat
#>
#>          Left Right <NA>
#>   Female    7   110    1
#>   Male     10   108    0
#>   <NA>      1     0    0
as.table(tomb) # tömbből 3D táblázat
#> , ,  = Freq
#>
#>
#>          Left Right <NA>
#>   Female    3    45    1
#>   Male      3    62    0
#>   <NA>      1     0    0
#>
#> , ,  = None
#>
#>
#>          Left Right <NA>
#>   Female    1    10    0
#>   Male      2    11    0
#>   <NA>      0     0    0
#>
#> , ,  = Some
#>
#>
#>          Left Right <NA>
#>   Female    3    55    0
#>   Male      5    35    0
#>   <NA>      0     0    0
xtabs(Freq~Var1, data=df1)           # adattáblából 1D táblázat
#> Var1
#> Female   Male
#>    118    118
xtabs(Freq~Var1+Var2, data=df2)      # adattáblából 2D táblázat
#>         Var2
#> Var1     Left Right
#>   Female    7   110
#>   Male     10   108
```

```
xtabs(Freq~Var1+Var2+Var3, data=df3) # adattáblából 3D táblázat
#> , , Var3 = Freq
#>
#>         Var2
#> Var1     Left Right
#>   Female    3    45
#>   Male      3    62
#>
#> , , Var3 = None
#>
#>         Var2
#> Var1     Left Right
#>   Female    1    10
#>   Male      2    11
#>
#> , , Var3 = Some
#>
#>         Var2
#> Var1     Left Right
#>   Female    3    55
#>   Male      5    35
```

Érdekes lehet egy harmadik eset is, amikor a gyakorisági adatok állnak rendelkezésre (táblázatos vagy adattábla formátumban) és el szeretnénk készíteni ennek a nyers adatokat tartalmazó adattábla megfelelőjét. Vegyük a legbonyolultabb eddig tárgyalt esetet, és legyen a `tab3` a kiinduló pontunk, amely egy táblázat. A táblázatot a korábban tanult módszerrel gyakoriságokat tartalmazó adattáblává alakítjuk, majd eseteket (nyers adatokat) tartalmazó adattáblává.

```
# 3D gyakorisági táblázatból indulunk ki
tab3 <- table(survey$Sex, survey$W.Hnd, survey$Exer, useNA = "ifany")
ftable(tab3)  # 3D gyakorisági táblázat kiterítve
#>               Freq None Some
#>
#> Female Left      3    1    3
#>        Right    45   10   55
#>        NA        1    0    0
#> Male   Left      3    2    5
#>        Right    62   11   35
#>        NA        0    0    0
#> NA     Left      1    0    0
```

```
#>        Right     0    0    0
#>        NA        0    0    0
```

A `tab3` 3D táblázatot a `as.data.frame()` függvénnyel adattáblává alakítjuk, amelynek oszlopai a `Var1`, `Var2`, `Var3` és `Freq` nevet kapják.

```
# 3D táblázat adattáblává alakítása
df3 <- as.data.frame(df3)
head(df3)
#>     Var1  Var2 Var3 Freq
#> 1 Female  Left Freq    3
#> 2   Male  Left Freq    3
#> 3   <NA>  Left Freq    1
#> 4 Female Right Freq   45
#> 5   Male Right Freq   62
#> 6   <NA> Right Freq    0
```

A nyers adatok képzéséhez a `rep()` függvényt használjuk, amely a `Freq` oszlopban lévő számoknak megfelelően ismétli meg a `Var1`, `Var2` és `Var3` oszlopokban lévő értékeket. Az új adattábla sorainak sorszámait az `rownames()` függvénnyel állítjuk be.

```
# az átalakítás 2 sora:
df.long <- df3[rep(row.names(df3), df3$Freq), c("Var1", "Var2", "Var3")]
rownames(df.long) <- seq_along(rownames(df.long))
psych::headTail(df.long)  # a kész adattábla a nyers adatokkal
#>       Var1  Var2 Var3
#> 1   Female  Left Freq
#> 2   Female  Left Freq
#> 3   Female  Left Freq
#> 4     Male  Left Freq
#> ...   <NA>  <NA> <NA>
#> 234   Male Right Some
#> 235   Male Right Some
#> 236   Male Right Some
#> 237   Male Right Some
```

5.4.2. Dátum és idő

Az adatelemzés során a dátumok kezelésének két fő oka lehet, egyrészt szűrésekben használhatjuk őket, például adott dátum vagy időpont előtti, utáni vagy közötti sorok leválogatásában,

másrészt statisztikai elemzések is irányulhatnak két dátum vagy időpont között eltelt időtartamra.

5.4.2.1. Dátum kezelése

Amennyiben le akarjuk kérdezni az aktuális dátumot, akkor a `Sys.Date()` függvényt kell használnunk.

```
datum <- Sys.Date()     # aktuális dátum, dátum típusú objektum
datum                   # dátum kiírása
#> [1] "2025-07-21"
typeof(datum)           # dátum típusa
#> [1] "double"
class(datum)            # dátum osztálya
#> [1] "Date"
unclass(datum)          # dátum alapja
#> [1] 20290
```

Láthatjuk, hogy a `datum` objektum *dátum* (*Date*) típusú annak ellenére, hogy az objektum értéke a képernyőn kettős idézőjelek között jelenik meg. A *dátum* típus alapja egy *double* szám, amely az 1970-01-01 óta eltelt napok számát tartalmazza, ahogyan az `unclass(datum)` ezt számunkra meg is mutatja. Világos, hogy az a *double* érték lehet nulla vagy negatív is.

```
unclass(as.Date("1980-01-01")) # a double szám pozitív
#> [1] 3652
unclass(as.Date("1970-01-01")) # a double szám nulla
#> [1] 0
unclass(as.Date("1960-01-01")) # a double szám negatív
#> [1] -3653
```

5.4.2.2. Dátum létrehozása karakteres adatból

Dátumot legtöbb esetben karakteres konstansból hozunk létre az `as.Date()` függvény segítségével. A dátumok változatos formában jelenhetnek meg, a szabványos `"2019-02-12"` (ISO 8601) alak mellett sok olyan forma létezik, amelyben elválasztó karakterként a perjel vagy a pont szerepel, valamint az év-hó-nap hármas sorrendje is változhat. A konkrét dátum értelmezéséhez az `as.Date()` függvény `format=` argumentumát kell helyesen beállítani. A használható kódokat a 5.11. táblázat tartalmazza.

5.11. táblázat: Formátumkódok a dátumokban *(saját szerkesztés)*

Formátum kód	Jelentés
%Y	év 4 számjeggyel
%y	év 2 számjeggyel
%m	hónap
%b	hónap rövidített neve
%B	hónap teljes neve
%d	nap

```
as.Date("2020-04-12")   # szabványos, nem kell format= argumentum
#> [1] "2020-04-12"
as.Date("2020/04/12")   # szabványos, nem kell format= argumentum
#> [1] "2020-04-12"
as.Date("04/12/2020", format="%m/%d/%Y")          # amerikai stílus
#> [1] "2020-04-12"
as.Date("12.04.2020", format="%d.%m.%Y")          # brit stílus
#> [1] "2020-04-12"
as.Date("2020. 04. 12.", format="%Y. %m. %d.")    # magyar stílus
#> [1] "2020-04-12"
```

Látható, hogy a szabványos esetekben nem szükséges a `format=` argumentum használata, de a formátumkódokkal tetszőleges sztringet *dátum* típusúvá alakíthatunk. A hónapnevek megjelenése azonban nyelvfüggő, ezért itt a R verziónk helyi beállításaira is figyelni kell.

```
Sys.getlocale("LC_TIME")                          # a helyi beállítás magyar?
#> [1] "Hungarian_Hungary.utf8"
as.Date("2020. ápr. 12.", format="%Y. %b %d.")    # rövid magyar hónapnévvel
#> [1] "2020-04-12"
as.Date("2020. április 12.", format="%Y. %B %d.") # magyar hónapnévvel
#> [1] "2020-04-12"
```

Magyar számítógépes környezetben a helyi beállítás (`?locales`) alapértelmezés szerint magyar, ennek megfelelően a magyar hónapnevekkel dolgozik az `as.Date()` függvény, így a fenti konverziók a kívánt eredményt adják. A `Sys.getlocale("LC_TIME")` paranccsal vizsgálhatjuk meg, hogy milyen környezetben dolgozunk. A `Sys.setlocale("LC_TIME", "C")` utasítás észak-amerikai beállításra váltja az R-t, az angol hónapnevek felismerésére így nyílik lehetőség:

```
lct <- Sys.getlocale("LC_TIME")             # helyi beállítás mentése
Sys.setlocale("LC_TIME", "C")               # észak-amerikai beállítás
#> [1] "C"
as.Date("Apr 12, 2020", format="%b %d, %Y") # rövid angol hónapnévvel
#> [1] "2020-04-12"
as.Date("12 April 2020", format="%d %B %Y") # angol hónapnévvel
#> [1] "2020-04-12"
Sys.setlocale("LC_TIME", lct)               # magyar helyi beállítás visszatöltése
#> [1] "Hungarian_Hungary.utf8"
```

5.4.2.3. Dátum létrehozása numerikus adatokból

Dátumot a szeparáltan létező numerikus év, hónap, nap információkból is létrehozhatunk. Ehhez először az `ISOdate()` függvénnyel időpontot állítunk elő, majd az `as.Date()`-tel dátumot. Ezzel a módszerrel egyszerre több dátumot is előállíthatunk.

```
as.Date(ISOdate(year = 2020, month = 4, day = 12))   # dátum előállítása
#> [1] "2020-04-12"
as.Date(ISOdate(year = 2020, month = 1:4, day = 12)) # dátumok előállítása
#> [1] "2020-01-12" "2020-02-12" "2020-03-12" "2020-04-12"
```

5.4.2.4. Dátum konvertálása

Ha már van egy *dátum* típusú objektumunk, akkor azt változatos módon jeleníthetjük meg a `format()` függvény segítségével, amely egyszerű karakteres adattal tér vissza.

```
(datum <- as.Date("04/12/2020", format = "%m/%d/%Y"))
#> [1] "2020-04-12"
format(datum, format="%Y. %m. %d.") # magyar dátum
#> [1] "2020. 04. 12."
format(datum, format="%Y. %B %d.")  # magyar dátum
#> [1] "2020. április 12."
format(datum, format="%Y. %b %d.")  # magyar dátum
#> [1] "2020. ápr. 12."
format(datum, format="%Y-%m")       # csak az év és a hónap
#> [1] "2020-04"
format(datum, format="%Y")          # csak az év
#> [1] "2020"
```

5.4.2.5. Dátum-idő kezelése

A *dátum-idő* (*POSIXct* típus) olyan *double* érték, amelynek jelentése az `1970-01-01` óta eltelt másodpercek száma. Az aktuális dátum és idő lekérdezése `Sys.time()` függvénnyel lehetséges, és ez az általunk *dátum-idő* típusnak tekintett *POSIXct* objektummal tér vissza:

```
ido <- Sys.time()  # pontos dátum-idő, POSIXct típusú objektum
ido  # ido kírása
#> [1] "2025-07-21 15:09:41 CEST"
typeof(ido)  # ido típusa
#> [1] "double"
class(ido)  # ido típusa
#> [1] "POSIXct" "POSIXt"
unclass(ido)  # ido alapja
#> [1] 1753103382
```

A fentiek alapján úgy tűnhet, hogy a *POSIXct* objektum egész másodperceket tárol csupán, de ez nem így van. Az alapértelmezett megjelenítéseken módosítva láthatóvá válnak a tört másodpercek is:

```
op <- options(digits.secs = 6, digits = 16)
ido  # POSIXct kiírása változott digits.sec=6 miatt
#> [1] "2025-07-21 15:09:41.843744 CEST"
unclass(ido)  # double kiírása változott digits=6 miatt
#> [1] 1753103381.843745
options(op)  # alapértelmezések visszaállítása
```

A *POSIXct* objektumok másik érdekessége az időzóna tárolása, amely alapértelmezés szerint a magyar környezetben futó R helyi beállításainak megfelelően közép-európai (CEST) időt jelent. A saját rendszerünk időzónája a `Sys.timezone()` függvénnyel kérdezhető le, a lehetséges időzónákat az `OlsonNames()` függvény listázza. A legtöbb esetben ezzel nem kell foglalkoznunk, nemzetközi kutatások esetében azonban fontos lehet ismerni az időzóna váltás lehetőségét.

Egyik lehetőség, hogy eleve a kívánt időzónának megfelelő időpontokkal dolgozunk. Ekkor a `Sys.setenv()` függvénnyel beállíthatjuk a kívánt időzónát, amely a legtöbb esetben a a koordinált világidő (UTC) vagy másképp a greenwichi középidő (GMT). Tudjuk, hogy a magyarországi időzóna téli időszámításkor közép-európai idő (CET, UTC+1), nyáron közép-európai nyári idő (CEST, UTC+2).

```
tz <- Sys.timezone()  # helyi időzóna mentése
Sys.setenv(TZ = "UTC")  # UTC (GMT) időzóna beállítása
Sys.time()  # pontos dátum-idő lekérése UTC szerint
#> [1] "2025-07-21 13:09:41 UTC"
Sys.setenv(TZ = tz)  # alapértelmezett időzóna visszaállítása
```

A másik lehetőség, hogy már egy létező *POSIXct* objektumon végzünk időzóna konverziót, amely így az objektum óra (vagy egyéb) részét is érintheti.

```
ido <- Sys.time()  # pontos dátum-idő, helyi beállításnak megfelelően
ido.utc <- as.POSIXct(format(ido, tz = "UTC"), tz = "UTC")  # konverzió UTC-re
ido  # helyi időzónával
#> [1] "2025-07-21 15:09:41 CEST"
ido.utc  # UTC-vel
#> [1] "2025-07-21 13:09:41 UTC"
```

5.4.2.6. Dátum-idő létrehozása karakteres adatból

Amennyiben karakteres formában rendelkezésre áll egy időpont, akkor mindössze az egyes komponensek jelentését kell elmagyaráznunk az `as.POSIXct()` függvény `format=` argumentumában. Szabványos idő megadása esetén ((ISO 8601)) ezt el is hagyhatjuk.

```
as.POSIXct("2022-06-02 22:12:23", tz = "Europe/Budapest")  # szabványos idő
#> [1] "2022-06-02 22:12:23 CEST"
as.POSIXct("2019.09.06. 16 34 17", format = "%Y.%m.%d. %H %M %S", tz = "UTC")
#> [1] "2019-09-06 16:34:17 UTC"
```

A dátum értelmezéséhez használt kódok köre (5.11. táblázat) kibővül a 5.12. táblázatban szereplő időre vonatkozó kódokkal, így ezeket is használhatjuk a *POSIXct* objektum létrehozása során. Teljes listát az `?strptime` súgójában olvashatunk.

5.12. táblázat: Időre vonatkozó kódok *(saját szerkesztés)*

Formátum kód	Jelentés
%H	óra (00-23)
%I	óra (01-12)
%M	perc (00-59) neve
%p	AM/PM jelzése

Formátum kód	Jelentés
%S	másodperc (00-59)
%Z	időzóna (csak output)

5.4.2.7. Dátum-idő létrehozása numerikus adatokból

Dátum-időt szeparáltan létező információkból is létrehozhatunk. Ehhez az `ISOdatetime()` függvényt kell használni, ahol minden egyes komponens egyesével felsorolható:

```
# POSIXct objektum a dátum-idő tárolására
ISOdatetime(year=2022, month=7, day=3,
            hour=11, min=12, sec=3, tz = "Europe/Budapest")
#> [1] "2022-07-03 11:12:03 CEST"
```

5.4.2.8. Dátum-idő konvertálása

A *POSIXct* objektum *dátum* típusúvá konvertálható az `as.Date()` függvénnyel, illetve a `format()` függvény segítségével tetszőleges formájú karakteres dátumot/időt kinyerhetünk az objektumból.

```
ido <- Sys.time()  # pontos dátum-idő, helyi beállításnak megfelelően
ido                # dátum-idő objektum
#> [1] "2025-07-21 15:09:41 CEST"
as.Date(ido)       # dátum objektum
#> [1] "2025-07-21"
format(ido, format="%Y. %m. %d.")              # magyar dátum
#> [1] "2025. 07. 21."
format(ido, format="%Y. %B %d. %H.%M.%S")      # magyar dátum-idő
#> [1] "2025. július 21. 15.09.41"
format(ido, format="%Y. %b %d. %H:%M:%S")      # magyar dátum-idő
#> [1] "2025. júl. 21. 15:09:41"
format(ido, format="%Y. %m. %d. %H:%M:%S %Z")  # magyar dátum-idő
#> [1] "2025. 07. 21. 15:09:41 CEST"
```

5.4.2.9. Műveletek és az időtartam

A többnyire szöveges formában megjelenő dátumok és dátum-idők R objektummá alakításának a legnagyobb haszna, hogy a *Date* és *POSIXct* objektumokkal számos műveletet hajthatunk végre. Lehetőségünk van például különböző dátumok összehasonlítására, kivonására, léptetésére, vagy ábrákon a tengelyeket címkézhetjük dátum objektumokkal. Két dátum (vagy dátum-idő) különbsége az időtartam, amelyet a kivonás (–) operátorral, vagy a `difftime()` függvénnyel is előállíthatunk. Utóbbi nagyobb szabadságot ad, mert rendelkezik egy `unit=` argumentummal az időtartam mértékegységének megadására, így értéke lehet a `"secs"`, `"mins"`, `"hours"`, `"days"` vagy `"weeks"` is.

```
Sys.Date() - as.Date("2001-03-17")  # születésnap óta eltelt idő napokban
#> Time difference of 8892 days
difftime(Sys.Date(), as.Date("2001-03-17"))  # ua.
#> Time difference of 8892 days
difftime(Sys.Date(), as.Date("2001-03-17"), unit = "hours")  # órákban
#> Time difference of 213408 hours
as.numeric(difftime(Sys.Date(), as.Date("2001-03-17"), unit = "hours"))  # számként
#> [1] 213408
```

Ne feledjük, hogy az időtartam is egy típus az R-ben (*difftime* osztály), ahogyan a következő sorokban ez megfigyelhetjük:

```
difft <- difftime(Sys.Date(), as.Date("2001-03-17"), unit = "hours")
typeof(difft)  # difft típusa
#> [1] "double"
class(difft)  # difft típusa
#> [1] "difftime"
unclass(difft)  # difft alapja
#> [1] 213408
#> attr(,"units")
#> [1] "hours"
```

A `difftime()` működik dátum-idővel is, és természetesen két dátum vagy időpont között a szokásos műveletek is elvégezhetők. A Google Űrlap időbélyeg oszlopából rögzítettünk két adatot és elvégeztünk néhány műveletet köztük:

```
idobelyeg.1 <- as.POSIXct("2022.04.06. 11:11:33", format = "%Y.%m.%d. %H:%M:%S",
    tz = "UTC")
idobelyeg.2 <- as.POSIXct("2022.04.06. 12:06:35", format = "%Y.%m.%d. %H:%M:%S",
```

```
    tz = "UTC")
idobelyeg.1 == idobelyeg.2  # nem egyenlőek
#> [1] FALSE
idobelyeg.1 < idobelyeg.2  # az első időbélyeg a korábbi
#> [1] TRUE
# hány másodperc telt el a két válasz között
as.numeric(difftime(idobelyeg.2, idobelyeg.1, unit = "sec"))
#> [1] 3302
```

5.4.3. Tibble

A *Tidyverse R* használata során az adatainkat *tibble* típusú objektumban tároljuk. Használatához töltsük be a `{tidyverse}` csomagot.

```
library(tidyverse)

# tibble létrehozása
x <- rep(c("A", "B"), times = 4)
y <- rep(6:9, times = 2)
z <- 1:8
df <- tibble(nev = x, pont.1 = y, pont.2 = z)
df
#> # A tibble: 8 x 3
#>   nev   pont.1 pont.2
#>   <chr>  <int>  <int>
#> 1 A          6      1
#> 2 B          7      2
#> 3 A          8      3
#> 4 B          9      4
#> 5 A          6      5
#> 6 B          7      6
#> # i 2 more rows
```

A tibble objektumok alaptípusa lista, de az osztálytípusok között megjelennek a tibble-re specifikus osztályok is. A `tbl_df` osztály jelenléte hozza magával azokat az új tulajdonságokat és lehetőségeket, amik a *Tidyverse R* központi adatszerkezetévé teszik ezt az objektumtípust.

```
# a tibble típusú objektumok típusa és osztályai
attributes(df)
#> $class
#> [1] "tbl_df"     "tbl"        "data.frame"
#>
#> $row.names
#> [1] 1 2 3 4 5 6 7 8
#>
#> $names
#> [1] "nev"    "pont.1" "pont.2"
typeof(df)
#> [1] "list"
class(df)
#> [1] "tbl_df"     "tbl"        "data.frame"
```

A tibble és az adattábla (data frame) típusú objektumok között az átjárhatóságot az `as_tibble()` és az `as.data.frame()` függvény biztosítja.

```
# a tibble és a data frame közötti átkonvertálás
as_tibble(df)
#> # A tibble: 8 x 3
#>   nev   pont.1 pont.2
#>   <chr>  <int>  <int>
#> 1 A          6      1
#> 2 B          7      2
#> 3 A          8      3
#> 4 B          9      4
#> 5 A          6      5
#> 6 B          7      6
#> # i 2 more rows
as.data.frame(df)
#>   nev pont.1 pont.2
#> 1   A      6      1
#> 2   B      7      2
#> 3   A      8      3
#> 4   B      9      4
#> 5   A      6      5
#> 6   B      7      6
#> 7   A      8      7
#> 8   B      9      8
```

A tibble típus tesztelése az `is_tibble()` segítségével történik, de a tibble típusú objektumokra az `is.data.frame()` is igaz értékkel tér vissza:

```
# a tibble és a data frame típus tesztelése
is_tibble(df)
#> [1] TRUE
is.data.frame(df)
#> [1] TRUE
```

Melyek az adattábla és a tibble közötti különbségek? Már három eltérést akár észre is vehettünk. Az első a tibble létrehozásához kötődik. Egy tibble típusú objektum, csak azonos hosszúságú oszlopvektorokból hozható létre, így biztonságosabban konstruálható, mint az ismétlést is támogató adattábla típusú objektumok. Tibble esetében csak az 1 elemet tartalmazó vektorok ismétlése megengedett. Tehát ez a konstrukció működik:

```
# tibble létrehozása: csak az 1 elemű vektorok ismételhetők
tibble(a = c("a", "b", "c"), p = 1)
#> # A tibble: 3 x 2
#>   a         p
#>   <chr> <dbl>
#> 1 a         1
#> 2 b         1
#> 3 c         1
```

A második különbség, hogy a tibble a létrehozás során nem végez automatikus típuskonverziót. Tehát a karakteres vektorokat nem alakítja át faktorokká. Ez ma már a `data.frame()` függvény esetében is így van, de a régi hagyományos R-ben a `data.frame()` automatikusan faktorokká alakította a karakteres vektorokat. A tibble esetében ez nem történik meg, és a karakteres vektorok karakteresek maradnak.

```
# a karakteres vektorok nem konvertálódnak faktorokká
str(df)
#> tibble [8 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ nev   : chr [1:8] "A" "B" "A" "B" ...
#>  $ pont.1: int [1:8] 6 7 8 9 6 7 8 9
#>  $ pont.2: int [1:8] 1 2 3 4 5 6 7 8
```

A harmadik különbség az adatok megjelenítésében van. Tibble esetében csak az első 10 sor jelenik meg, és annyi oszlop, amennyi az aktuális képernyőre kifér. A többi oszlop neve alul jelenik meg. Az oszlopnevek alatt az oszlop típusa is megjelenik.

A negyedik eltérés a tibble indexeléséhez kötődik. Az `[` operátor használata során alapesetben tibble típusú objektumot kapunk, nem kaphatunk vektort, azaz nem történhet dimenzióvesztés.

```
# nincs dimenzióvesztés tibble esetén
df[, 2]
#> # A tibble: 8 x 1
#>   pont.1
#>    <int>
#> 1      6
#> 2      7
#> 3      8
#> 4      9
#> 5      6
#> 6      7
#> # i 2 more rows
df[1, ]
#> # A tibble: 1 x 3
#>   nev   pont.1 pont.2
#>   <chr>  <int>  <int>
#> 1 A          6      1
df[1, 2]
#> # A tibble: 1 x 1
#>   pont.1
#>    <int>
#> 1      6
df[1, 2, drop = T]  # ha előírjuk, akkor lehet dimenzióvesztés
#> [1] 6
```

A későbbi fejezetekben látni fogjuk, hogy a *Tidyveserse R* használata során többnyire *tibble* típusú objektumokkal dolgozunk. A `{tidyverse}` csomag betöltése után a `{tibble}` csomag automatikusan betöltődik. A `{tibble}` csomag függvényei közül a legfontosabbak a `glimpse()` és az `add_column()` függvények. A `glimpse()` függvény a tibble objektumok gyors áttekintésére szolgál.

```
# a tibble objektumok gyors áttekintése
glimpse(df)
#> Rows: 8
#> Columns: 3
#> $ nev    <chr> "A", "B", "A", "B", "A", "B", "A", "B"
```

```
#> $ pont.1 <int> 6, 7, 8, 9, 6, 7, 8, 9
#> $ pont.2 <int> 1, 2, 3, 4, 5, 6, 7, 8
```

Az `add_column()` függvény új oszlopok hozzáadására szolgál. Az `add_column()` függvény használata során megadhatjuk, hogy az új oszlopok hova kerüljenek a meglévő oszlopokhoz képest.

```
# új oszlopok hozzáadása a tibble objektumhoz
df <- add_column(df, uj1 = 1:8, uj2 = 1:8, .after = 1)
df
#> # A tibble: 8 x 5
#>   nev     uj1   uj2 pont.1 pont.2
#>   <chr> <int> <int>  <int>  <int>
#> 1 A         1     1      6      1
#> 2 B         2     2      7      2
#> 3 A         3     3      8      3
#> 4 B         4     4      9      4
#> 5 A         5     5      6      5
#> 6 B         6     6      7      6
#> # i 2 more rows
```

A fenti példában az `uj1` és `uj2` oszlopok a 1. oszlop után kerültek beillesztésre, azaz a 2. és 3. oszlopok lettek. Az `.after` mellett a `.before` argumentum is használható, amely a megadott oszlop előtt helyezi el az új oszlopokat.

A `{tibble}` csomag talán legnagyszerűbb függvénye a `tribble()`, amely a tibble objektumok gyors létrehozására szolgál. A `tribble()` függvény használata során a sorokat és oszlopokat is megadhatjuk, így a tibble objektumok gyorsan létrehozhatók. Hozzunk létre félév végi osztályzatokat tartalmazó egyszerű tibble objektumot:

```
# a tibble objektumok gyors létrehozása
df <- tribble(
  ~nev,   ~magyar, ~angol, ~matek,
  "Pál",        4,      5,      3,
 "Anna",        5,      4,      4,
"János",        3,      4,      5
)
df
#> # A tibble: 3 x 4
#>   nev   magyar angol matek
```

```
#>   <chr>  <dbl> <dbl> <dbl>
#> 1 Pál        4     5     3
#> 2 Anna       5     4     4
#> 3 János      3     4     5
glimpse(df)
#> Rows: 3
#> Columns: 4
#> $ nev    <chr> "Pál", "Anna", "János"
#> $ magyar <dbl> 4, 5, 3
#> $ angol  <dbl> 5, 4, 4
#> $ matek  <dbl> 3, 4, 5
```

A `tribble()` függvény használata során a `~` karakterrel jelöljük az oszlopneveket, majd vesszővel választjuk el az egyes értékeket. A sortörések csak a könnyebb olvashatóságot szolgálják, a `tribble()` függvény nem veszi figyelembe őket.

5.4.4. A *Tidyverse R* és a pipe operátor

A *Tidyverse R* kapcsán már említést tettünk a pipe operátorról (2.2. fejezet). Említettük, hogy a `{margittr}` csomagban található pipe operátor (`%>%`) a *Tidyverse R* része, amely a függvények egymásba ágyazásának egy új módját kínálja. A pipe operátor használatával a kódunk olvashatóbbá és karbantarthatóbbá válik. A pipe operátor az R 4.0 verziójától kezdve már az *Alap R* része, így a `{margittr}` csomag telepítése és betöltése nélkül is használható, de alakja megváltozott: `|>`. A `%>%` operátor továbbra is elérhető maradt a *Tidyverse R*-en belül, de ebben a könyvben a standard `|>` pipe operátort fogjuk használni.

Miben is rejlik a pipe operátor előnye? Hogyan teszi olvashatóbbá és könnyebben módosíthatóvá a parancsainkat? Ezt fogjuk ebben a fejezetben megvizsgálni.

Egy probléma megoldása általában több lépésben történik. Ha például az 1, 2 és 3 számok természetes alapú logaritmusának az összegét szeretnénk kiszámolni, akkor a következő sorokat írhatjuk:

```
x <- c(1, 2, 3)       # 1. lépés: vektor előállítása
out <- log(x)         # 2. lépés: természetes alapú logaritmus kiszámolása
out <- sum(out)       # 3. lépés: átlag kiszámolása
out                   # 4. lépés: eredmény kírása
#> [1] 1.791759
```

A részeredmények tárolására az `out` objektumot használtuk fel. A köztes eredmények tárolása azonos vagy eltérő nevek alatt is lehetséges, most végig az `out` változót használtuk.

A fenti, lépésenkénti megoldás helyett választhatjuk a függvények egymásba ágyazását is:

```
x <- c(1, 2, 3)       # vektor előállítása
sum(log(x))           # eredmény kiírása
#> [1] 1.791759
```

A fenti példa jóval tömörebb és nem szükséges köztes `out` objektumok létrehozása, a hagyományos R-ben nagyon gyakori összetett függvényhívásokkal (rész)problémák megoldása.

A pipe (`|>`) operátor kínál egy harmadik lehetőséget is:

```
x <- c(1, 2, 3)     # vektor előállítása
x |> log() |> sum() # eredmény kiírása
#> [1] 1.791759
```

A fenti sor olvasható úgy, hogy induljunk ki az `x` adatobjektumból, *azután* vegyük a 2-es alapú logaritmusát, *azután* vegyük az értékek összegét. A `|>` pipe operátor az *azután* lehetőséget kínálja, és egyben egy rendkívül jól olvasható és karbantartható kód létrehozását teszi lehetővé. A fentebb látott függvények egymásba ágyazásánál sokkal könnyebben javítható ez a kód, az azt megelőző, köztes objektumokkal dolgozó esetnél pedig jóval egyszerűbb és elegánsabb. A pipe operátort használó parancsok tipikusan adatobjektummal, vagy az azt létrehozó függvénnyel indítanak, majd azoknak a tevékenységeknek a felsorolása történik függvényhívások egymásutánjának megadásával, amelyeket a felsorolás sorrendjében az adatokkal tenni szeretnénk. Az adat rész ezekből a függvényhívásokból hiányzik, ugyanis a pipe operátor biztosítja, hogy mindegyik függvény első argumentumába megkapja azt.

```
10 |>
    log()  # ekvivalens alak: log(x=10)
#> [1] 2.302585
c(10, 100) |>
    log()  # ekvivalens alak: log(x=c(10, 100))
#> [1] 2.302585 4.605170
c(10, 100) |>
    log(base = 10)  # ekvivalens alak: log(x=c(10, 100), base=10)
#> [1] 1 2
```

A fenti példában, látható, hogyan gondoskodik a pipe operátor arról, hogy `log()` függvény első argumentuma (`x=`) megkapja a szükséges értékét. Néhány függvény esetében előfordul,

hogy a kiinduló adat apaértelmezés szerint nem az első paraméter. Ekkor az aláhúzás _ helyőrző segítségével expliciten jelezzük, hogy hová kerül a pipe operátor bal oldaláról érkező adat. Például a karakteres adat cseréjére használatos `sub()` függvény `x=` argumentuma alapértelmezés szerint a harmadik:

```
sub(pattern = "s", replacement = "z", x = "Sissi")  # egy s cseréje z-re
#> [1] "Sizsi"
```

A következő függvényhívás az argumentumok nevesítése miatt szintén működőképes, a pipe-ban `x="Sissi"` argumentumhívás valósul meg.

```
"Sissi" |> sub(pattern = "s", replacement = "z") # egy s cseréje z-re
#> [1] "Sizsi"
```

Azonban ha nem használunk argumentumneveket, akkor ez a hívás már nem kívánt eredményre vezet, mert alapértelmezés szerint az első argumentumba kerül az adat, amelyik a `sub()` esetében a `pattern=` argumentum.

```
"Sissi" |> sub("s", "z")                    # nem ezt akartuk
#> [1] "z"
sub(pattern="Sissi", replacement="s", x="z") # valójában ez hívódik
#> [1] "z"
```

Ha tehát nem az első argumentumban szeretnénk szerepeltetni a pipe bal oldaláról érkező adatot, akkor a _ helyőrzőt kell használnunk, de a helyőrző argumentumnevét ekkor is meg kell adnunk:

```
"Sissi" |> sub("s", "z", x=_)
#> [1] "Sizsi"
```

Most már megismertük a pipe operátor alapvető használatát, nézzük meg, hogyan egyszerűsíti a kódunkat a tipikus felhasználás során, vagyis amikor adattáblával dolgozunk. A *Tidyverse R* csomagokban a pipe operátor használata során a `{dplyr}` csomag függvényeit fogjuk használni, amelyek az adattáblák manipulálására szolgálnak. A `{dplyr}` csomag függvényei közül a legfontosabbak a `select()`, a `rename()`, a `mutate()`, a `filter()`, `slice()`, `arrange()` és az `summarise()` függvények. A függvények fő funkciói:

- `select()`: az adattábla oszlopainak szűrése, azaz az oszlopok kiválasztása,
- `rename()`: az adattábla oszlopainak átnevezése,
- `mutate()`: új oszlopok létrehozása, vagy meglévő oszlopok módosítása,

- `filter()`: az adattábla sorainak szűrése, azaz a sorok leválogatása tartalom alapján,
- `slice()`: az adattábla sorainak kiválasztása pozíció szerint, azaz a sorok indexelése,
- `arrange()`: az adattábla sorainak rendezése,
- `summarise()`: az adattábla sorainak összesítése.

A fenti függvényeket kiegészítik a `group_by()` és `ungroup()` függvények, amelyek a csoportosításra és a csoportosítás megszüntetésére szolgálnak.

A következő részben a célunk kettős:

- megismerjük a *Tidyverse R* használatát, és megmutatjuk a *Tidyverse R* és az *Alap R* közötti különbségeket,
- megismerjük a pipe operátor használatát, és megmutatjuk a pipe operátor és a hagyományos függvényhívások közötti különbségeket.

A fenti célok elérését egy példán keresztül fogjuk bemutatni.

Példa 5.1 (Egyetemisták felmérése)**.** A `{MASS}` csomag `survey` adattáblája 237 válaszadó adatait tartalmazza. A `survey` adattábla 7 változót tartalmaz. Ezek a következők: `Sex`: a válaszadó neme, `Wr.Hnd`: a domináns kéz mérete, `NW.Hnd`: a nem domináns kéz mérete, `W.Hnd`: a domináns kéz, `Fold`: a karok összekulcsolásának iránya, `Pulse`: a pulzus, `Clap`: a tenyér összecsapásának iránya, `Exer`: a testmozgás gyakorisága, `Smoke`: a dohányzás gyakorisága, `Height`: a magasság cm-ben és `Age`: az életkor. A válaszadók 18 és 30 év közötti egyetemisták.

Az adattábla elérésére a `data()` függvényt használjuk:

```
# a survey adattábla betöltése
data(survey, package = "MASS")
```

Adjunk gyors áttekintést az adattábláról a *Tidyverse R* `glimpse()` függvénye segítségével, használjuk a hagyományos és a pipe operátorral írt változatot is:

```
library(tidyverse)
glimpse(survey)     # Tidyverse R, hagyományos függvényhívás
survey |> glimpse() # Tidyverse R, pipe operátorral
```

Kérdezzük le az oszlopneveket, mindkét módszerrel:

```
names(survey)     # Alap R, hagyományos függvényhívás
#>  [1] "Sex"    "Wr.Hnd" "NW.Hnd" "W.Hnd"  "Fold"   "Pulse"  "Clap"
#>  [8] "Exer"   "Smoke"  "Height" "M.I"    "Age"
survey |> names() # Alap R, pipe operátorral
#>  [1] "Sex"    "Wr.Hnd" "NW.Hnd" "W.Hnd"  "Fold"   "Pulse"  "Clap"
#>  [8] "Exer"   "Smoke"  "Height" "M.I"    "Age"
```

Listázzuk ki az adattábla első 5 sorát az első 3 oszlopból, a hagyományos és a pipe operátorral írt változatban:

```
head(survey[, 1:3], 5)   # Alap R, hagyományos függvényhívás
#>      Sex Wr.Hnd NW.Hnd
#> 1 Female   18.5   18.0
#> 2   Male   19.5   20.5
#> 3   Male   18.0   13.3
#> 4   Male   18.8   18.9
#> 5   Male   20.0   20.0
survey |> _[, 1:3] |> head(5) # Alap R, pipe operátorral
#>      Sex Wr.Hnd NW.Hnd
#> 1 Female   18.5   18.0
#> 2   Male   19.5   20.5
#> 3   Male   18.0   13.3
#> 4   Male   18.8   18.9
#> 5   Male   20.0   20.0
```

A sorok elérésére a *Tidyverse R* `slice()` függvényét is használhatjuk, az oszlopok között a `select()` függvénnyel válogathatunk:

```
library(tidyverse)
select(slice(survey, 1:5), 1:3) # Tidyverse R, hagyományos függvényhívás
#>      Sex Wr.Hnd NW.Hnd
#> 1 Female   18.5   18.0
#> 2   Male   19.5   20.5
#> 3   Male   18.0   13.3
#> 4   Male   18.8   18.9
#> 5   Male   20.0   20.0
survey |> slice(1:5) |>
  select(1:3)                  # Tidyverse R, pipe operátorral
#>      Sex Wr.Hnd NW.Hnd
```

```
#> 1 Female   18.5   18.0
#> 2   Male   19.5   20.5
#> 3   Male   18.0   13.3
#> 4   Male   18.8   18.9
#> 5   Male   20.0   20.0
```

Listázzuk ki az első 5 legmagasabb fiú korát és testmagasságát:

```
# Alap R, hagyományos függvényhívás
survey[order(-survey$Height), c("Age", "Height")][1:5, ]
#>        Age Height
#> 51  18.500 200.00
#> 190 20.083 196.00
#> 151 25.500 195.00
#> 118 18.917 193.04
#> 163 17.500 191.80
# Tidyverse R, hagyományos függvényhívás
slice(dplyr::select(arrange(survey, desc(Height)), Age, Height), 1:5)
#>      Age Height
#> 1 18.500 200.00
#> 2 20.083 196.00
#> 3 25.500 195.00
#> 4 18.917 193.04
#> 5 17.500 191.80
# Tidyverse R, pipe operátorral
survey |>
  arrange(desc(Height))|>
  dplyr::select(Age, Height) |>
  slice(1:5)
#>      Age Height
#> 1 18.500 200.00
#> 2 20.083 196.00
#> 3 25.500 195.00
#> 4 18.917 193.04
#> 5 17.500 191.80
```

Adjunk hozzá egy új `Weight` oszlopot az adattáblához, amely a válaszadók testsúlyát tartalmazza. A `Weight` oszlopban a válaszadók súlya kg-ban van megadva. Most a `sample()` függvénnyel készítjük el ezt a fiktív testsúly oszlopot:

```r
# fiktív testsúly oszlop előkészítése
testsuly <- sample(x = 50:100, size = nrow(survey), replace = T)
```

A `testsuly` oszlop birtokában elvégezhetjük a kívánt műveletet.

```r
# Alap R, hagyományos függvényhívás
survey$Weight <- testsuly
# Tidyverse R, hagyományos függvényhívás
survey <- mutate(survey, Weight = testsuly)
# Tidyverse R, pipe operátorral
survey <- survey |>
  mutate(Weight = testsuly)
```

Számoljuk ki a BMI értéket egy új oszlopban, a rendelkezésre álló testsúly és testmagasság alapján.

```r
# Alap R, hagyományos függvényhívás
survey$BMI <- survey$Weight / (survey$Height/100)^2
# Tidyverse R, hagyományos függvényhívás
survey <- mutate(survey, BMI = Weight / (Height/100)^2)
# Tidyverse R, pipe operátorral
survey <- survey |>
  mutate(BMI = Weight / (Height/100)^2)
```

Nevezzük át a `BMI` oszlopot `Tomegindex`-re, a hagyományos és a pipe operátorral írt változatban:

```r
# Alap R, hagyományos függvényhívás
names(survey)[names(survey) == "BMI"] <- "Tomegindex"
# Tidyverse R, hagyományos függvényhívás
survey <- rename(survey, Tomegindex = BMI)
# Tidyverse R, pipe operátorral
survey <- survey |>
  rename(Tomegindex = BMI)
```

Most már tudjuk, hogyan kell új oszlopokat létrehozni és átnevezni őket. Most nézzük meg, hogyan lehet az adattáblát szűrni a `filter()` függvény segítségével. Szűrjük le azokat a válaszadókat, akiknek a BMI értéke 25 és 30 között van:

```r
# Alap R, hagyományos függvényhívás
survey[survey$Tomegindex >= 25 & survey$Tomegindex <= 30, ]
# Tidyverse R, hagyományos függvényhívás
filter(survey, Tomegindex >= 25 & Tomegindex <= 30)
# Tidyverse R, pipe operátorral
survey |>
  filter(Tomegindex >= 25 & Tomegindex <= 30)
```

Most nézzük meg, hogyan lehet az adattáblát rendezni. Rendezzük az adattáblát a `Tomegindex` oszlop szerint növekvő sorrendben:

```r
# Alap R, hagyományos függvényhívás
survey.r <- survey[order(survey$Tomegindex), ]
# Tidyverse R, hagyományos függvényhívás
survey.r <- arrange(survey, Tomegindex)
# Tidyverse R, pipe operátorral
survey.r <- survey |>
  arrange(Tomegindex)
```

Átlagoljuk a `Tomegindex` oszlopot és nevezzük át `Tomegindex.atlag`-ra:

```r
# Alap R, hagyományos függvényhívás
tomegindex.atlag <- mean(survey$Tomegindex, na.rm = T)

# Tidyverse R, hagyományos függvényhívás
tomegindex.atlag <- summarise(survey,
                              Tomegindex.atlag = mean(Tomegindex, na.rm = T))
# Tidyverse R, pipe operátorral
tomegindex.atlag <- survey |>
  summarise(Tomegindex.atlag = mean(Tomegindex, na.rm = T))
```

Átlagoljuk a `Tomegindex` oszlopot külön a fiúknál és a lányoknál, és nevezzük át `Tomegindex.atlag.2`-re:

```r
# Alap R, hagyományos függvényhívás
tomegindex.atlag.2 <- aggregate(survey$Tomegindex,
                                by = list(survey$Sex),
                                FUN = mean, na.rm = T)
# Tidyverse R, pipe operátorral
```

```
tomegindex.atlag.2 <- survey |>
  group_by(Sex) %>%
  summarise(tomegindex_atlag = mean(Tomegindex, na.rm = TRUE))
```

A fenti függvényhívások bemutatása a pipe operátor mélyebb megértését és a *Tidyverse R* függvények bevezetését célozta. Az 7.2. és 8.2. fejezetekben részletesebben is foglalkozunk a *Tidyverse R* csomagokkal és a pipe operátorral.

5.4.5. A munkaterület függvényei

Megbeszéltük, hogy a munka során az objektumaink a memória speciális területére, a munkaterületre (workspace) kerülnek. Ha még korábban nem is hoztunk létre objektumot, akkor a következő három parancs, három objektumot hoz létre a munkaterületen:

```
fib.0 <- 0
fib.1 <- 1
fib.2 <- fib.0 + fib.1
```

A munkaterületen létrehozott objektumok neveit az `ls()` függvény listázza ki:

```
ls()
#> [1] "fib.0" "fib.1" "fib.2"
```

A munkaterületről objektumot az `rm()` paranccsal távolíthatunk el, például a

```
rm(fib.0)         # fib.0 törlése
ls()
#> [1] "fib.1" "fib.2"
```

a `fib.0` objektumot távolította el, így az `ls()` eredményében ez nem is szerepel. Az összes munkaterület-objektum eltávolítása a

```
rm(list = ls())   # összes objektum törlése
ls()
#> character(0)
```

segítségével történik. A `character(0)` eredmény azt jelenti, hogy a munkaterületen egyetlen objektum sincs, azaz üres karakteres vektor az `ls()` visszatérési értéke.

5.4.6. A munkakönyvtár függvényei

Az R használata során mindig van egy kitüntetett, aktuális könyvtárunk, amelyet munkakönyvtárnak nevezünk. A munkakönyvtár célja, hogy az állományok nyitása és mentése során, ha nem használunk külön könyvtárhivatkozást, akkor ez lesz az alapértelmezett könyvtár.

A munkakönyvtár az R-ben lekérdezhető, illetve beállítható a `getwd()` és a `setwd()` parancsok kiadásával. Például

```
getwd()
setwd("C:/Data/peldak")
```

parancsokkal először megismerjük az aktuális könyvtárat, majd megváltoztatjuk a `C:/Data/peldak` könyvtárra. Figyeljük meg, hogy az elérési útban perjelet (`/`) használtunk.

Megjegyezzük, hogy az *RStudio* projekt üzemmódú használata során nincs szükség a munkakönyvtár beállítására a `setwd()` paranccsal, sőt, kerüljük a használatát. A munkakönyvtárunk a munka során végig maradjon meg az alapértelmezetten beállított könyvtár, azaz maga a projektkönyvtár.

A munkakönyvtár jelentőségét tovább növeli, hogy az R indításakor ebben a könyvtárban 2 állomány létezését figyeli:

- `.Rhistory` (a visszahívható parancsokat tartalmazó szöveges állomány)
- `.RData` (a tárolt objektumokat tartalmazó bináris állomány).

A fenti állományok ugyanis betöltésre kerülnek az R indításakor, ha azokat az R megtalálja a munkakönyvtárban. Így ezek után, az `.Rhistory` állományból jövő parancsok között válogathatunk a parancssor használata során, illetve az `.RData` állományban tárolt objektumok azonnal elérhetőek, vagyis lesz egy induló munkaterületünk.

5.4.7. Csomagkezelő függvények

Korábban megbeszéltük, hogy a csomagok adatobjektumokat és függvényeket tartalmaznak. Az ún. egyéb csomagok (számuk kb. 22 ezer) elsődleges célja az *Alap R* tudásának kiegészítése.

Az R indítása után néhány csomag automatikusan betöltésre kerül a standard csomagok közül. Ezeket a csomagokat és egyéb ún. környezeteket listázhatunk ki a `search()` függvénnyel.

```
# környezetek listázása
search()
# > [1] '.GlobalEnv' 'package:stats' 'package:graphics' > [4]
# 'package:grDevices' 'package:utils' 'package:datasets' > [7]
# 'package:methods' 'Autoloads' 'package:base'
```

A fenti eredményben a `package` karaktersorozattal kezdődő elemek mutatják, hogy melyek az éppen betöltött csomagok. A listában nem szereplő, de korábban telepített csomagok betöltéséhez használjuk a `library()` vagy `require()` függvényeket.

```
# csomag betöltése
library(MASS)
require(foreign)
search()  # a környezetek listázása
# > [1] '.GlobalEnv' 'package:foreign' 'package:MASS' > [4]
# 'package:stats' 'package:graphics' 'package:grDevices' > [7]
# 'package:utils' 'package:datasets' 'package:methods' > [10] 'Autoloads'
# 'package:base'
```

A fenti példában a `{MASS}` és a `{foreign}` csomag betöltését és annak hatását követhetjük nyomon a `search()` függvény outputjára. Egy csomag betöltése azt jelenti, hogy a csomagban lévő függvények és objektumok a memóriába kerültek, azokat a parancsainkban ezután szabadon felhasználhatjuk.

Egy adott csomagban (esetünkben a `{foreign}` csomagban) lévő függvények és objektumok a

```
library(help = foreign)
```

vagy a

```
help(package = foreign)
```

paranccsal kérdezhetők le. Betöltött csomagok esetében használhatjuk az

```
library(foreign)
ls(name = "package:foreign", all.names = T)
ls(name = "package:base", all.names = T)
```

parancsot is, amely a csomag adatobjektumainak és függvényeinek a nevét listázza.

Betöltött csomagot a `detach()` függvénnyel távolíthatunk el a memóriából:

```
# csomag eltávolítása a memóriából
detach(package:foreign)
detach(package:MASS)
```

Ha a használni kívánt csomag még nincs telepítve a számítógépünkre, akkor az 3.1.3. fejezetben ismertetett módok egyikét válasszuk, attól függően, hogy a csomag melyik tárhelyről érhető el.

A CRAN-ról elérhető csomagok közül telepítsük fel a `{DescTools}` és `{psych}` csomagokat:

```
# csomagok telepítése
install.packages("DescTools")
install.packages("psych")
```

A számítógépünkön telepített csomagokról az `installed.packages()` függvény ad tájékoztatást. Amennyiben a

```
# telepített csomagok listázása
csomagok <- installed.packages()
View(csomagok)  # RStudio-ban vagy RGui-ban
```

parancsot kiadjuk az *RStudio*-ban, akkor csomagjainkat kényelmesen áttekinthetjük.

Csomagok frissítésére használjuk a már korábban említett

```
# csomagok frissítése
update.packages()
```

parancsot.

> Összefoglalás
>
> A kétdimenziós mátrixok általánosítása a tömb, amelyek az `array()` függvénnyel hozhatók létre. A kategorikus adatok tárolására a táblázatok, mátrixok és adattáblák is alkalmasak, fontos ismerni az átjárást köztük. A dátumokat *Date* a dátum-időket *POSIXct* objektumban tároljuk az R-ben, melyekkel a szokásos dátumkezelő műveletek

már könnyen elvégezhetők. Időtartamot, vagyis két dátum vagy időpont közötti különbséget a `difftime()` függvénnyel határozhatunk meg. A *tibble* az adattáblák egy modern verziója, amely a *Tidyverse R* csomagokban található. Az R-ben a pipe operátor (`|>`) használatával egyszerűsíthetjük a kódunkat, és könnyebben olvashatóvá tehetjük a parancsainkat. A pipe operátorral írt kódokban az objektumok nevei nem szerepelnek, így a kódunk rövidebb és áttekinthetőbb lesz. A munkaterület objektumait az `ls()` függvénnyel listázhatjuk ki, míg az aktuális munkakönyvtárat a `getwd()` függvénnyel kérdezhetjük le. A csomagok telepítése az `install.packages()` függvénnyel történik, míg a betöltésük a `library()` vagy `require()` függvényekkel. A csomagok eltávolítására (a memóriából) a `detach()` függvényt használjuk. A telepített csomagok listázására az `installed.packages()` függvény szolgál.

🎯 Feladatok

1. Egy 30 fős osztályban a 13 fiúból 5 tanuló vett részt közlekedési versenyen, míg a lányok közül 8-an. Hozzuk létre e "nem" és "verseny" változók kapcsolatát leíró mátrixot, táblázatot és adattáblát is.
2. Konvertáljuk dátummá a következő két sztringet: `"6November2020"`, `"2013-02-29"`! Utóbbi esetben mi lehet a hiba oka?
3. A `seq()` függvény `from=` és `to=` argumentuma a dátum típusú objektumokkal is működik. A `by=` argumentum értéke ilyenkor lehet numerikus (ekkor napokat jelent), de lehet `x weeks`, `x months` vagy `x years`, ahol `x` nullánál nagyobb egész lehet. Hozzunk létre egy dátum-vektort 2020 összes hétfőjének dátumával!
4. A Halley-üstökös utoljára 1986-ban járt a Naprendszerünkben, így az előrejelzések szerint legközelebb 2061. július 26-ban tér vissza. Rögzítsük ezt dátumként, és számoljuk ki, hány napot kell még várni az üstökös érkezésére.
5. Az `iris` adatbázishoz hasonlóan a `penguins` nevű adattábla egy könnyen kezelhető, tanulásra szánt új adatbázis az *Alap R*-ben, amely különböző pingvinfajok morfológiai adatait tartalmazza. Használjuk a `penguins` adatbázist a *Tidyverse R* függvényeivel és a pipe (`|>`) operátorral! A példák legyenek egyszerű, jól értelmezhető műveletek. A cél a gyakorlás, nem a statisztikai mélység!
6. Írassuk ki a munkaterület objektumait!
7. Hozzunk létre egy `pulzus` nevű objektumot és újra írassuk ki a munkaterület objektumneveit!
8. Távolítsuk el a `pulzus` objektumot a munkaterületről!
9. Határozzuk meg az aktuális munkakönyvtárat!
10. Növeljük meg a betűk méretét az *RGui*, az *R Commander* és az *RStudio* alkalmazásokban is!
11. Vizsgáljuk meg, hogy a számítógépünkön van-e telepítve a `{DescTools}` csomag, ha

nincs telepítsük! Derítsük ki, hogy a `{DescTools}` csomagnak mi a célja? Soroljunk fel három függvényt és adattáblát ebből a csomagból! Távolítsuk el a memóriából a `{DescTools}` csomagot!

12. Telepítsük a számítógépünkre a következő csomagokat: `{HSAUR3}`, `{psych}`, `{prettyR}`, `{descr}` és `{pastecs}`!

5.5. Haladó nyelvi elemek 🤯

> **Miről lesz szó?** Ebben a fejezetben
>
> - megismerjük az objektumok attribútumait, és
> - az R objektum-orientált lehetőségeit.

5.5.1. Attribútumok

Az R-ben használható objektumok név-érték párok, vagyis minden objektumnak van neve és értéke. Objektumok alatt ebben a könyvben az adatobjektumokat értjük. Már említettünk, hogy valójában a függvények is objektumoknak tekinthetők az R-ben, hiszen a függvénynek is van neve és értéke. Egy függvény utasítások sorozata.

Az R-ben minden objektum – például az eddig vizsgált vektorok is – attribútumokkal is rendelkezhetnek. Az attribútumok név-értek párok, amelyek speciális tulajdonságokkal ruházzák fel az objektumunkat. Például a `names` nevű attribútummal a vektor egyes elemeit nevezhetjük el. Későbbiekben látjuk a `dim`, `dimnames`, `levels` és `class` attribútumok jelentőségét is.

Egy objektum összes attribútuma az `attributes()` függvénnyel kérdezhető le. Ha a `names` attribútumra vagyunk kíváncsiak a `names()` függvényt is használhatjuk.

```
x <- 1:5         # integer vektor
attributes(x)    # x attribútumainak kiírása
#> NULL
names(x)         # x name attribútumának kiírása
#> NULL
```

Az `x` numerikus vektornak nincsenek attribútumai. A `NULL` az általános, elem nélküli vektort jelenti. A fenti outputban szereplő két `NULL` esetünkben azt jelzi, hogy nem állítottunk be semmilyen attribútumot, így `names` attribútumot sem.

```
x <- c("a"=1, "b"=2, "c"=3, "d"=4, "e"=5) # integer vektor
attributes(x)                  # x attribútumainak kiírása
#> $names
#> [1] "a" "b" "c" "d" "e"
names(x)                       # x name attribútumának kiírása
#> [1] "a" "b" "c" "d" "e"
```

A fenti `x` objektumnak már van egy `names` attribútuma, amely a vektor elemeit nevezi meg. Az `x` objektum `names` attribútuma a `c()` függvényben megadott karakteres konstansokból áll.

A `names` attribútum beállítható a `names()` függvénnyel is.

```
names(x) <- c("elégtelen", "elégséges", "közepes", "jó", "jeles")
attributes(x)  # x attribútumainak kiírása
#> $names
#> [1] "elégtelen" "elégséges" "közepes"   "jó"        "jeles"
names(x)  # x name attribútumának kiírása
#> [1] "elégtelen" "elégséges" "közepes"   "jó"        "jeles"
```

A `names` attribútum értéke karakteres vektor lehet, amely az outputokban is megjelenik és a indexelésben is felhasználhatjuk.

```
# names attribútum szerepel a kimenetben
x
#> elégtelen elégséges   közepes        jó     jeles
#>         1         2         3         4         5
x[c("közepes", "jó")]
#> közepes      jó
#>       3       4
```

Rögzítsük a (0, 1, 2) értékek előfordulási gyakoriságait a (18, 12, 20) elemeket tartalmazó vektorban. Az elemek nevei most is karakteres konstansok lesznek, az automatikus konverzióról az R gondoskodik.

```
y <- c(18, 12, 20)
names(y) <- 0:2
y
#>  0  1  2
#> 18 12 20
names(y)
#> [1] "0" "1" "2"
```

Az `y` vektor indexelésénél fontos, hogy megkülönböztessük a numerikus és a karakteres indexeket, az utóbbiaknál mindig idézőjelet kell használnunk.

```
y[1]                      # numerikus indexelés
#>  0
#> 18
y["1"]                    # karakteres indexelés
#>  1
#> 12
y[c(1,3)]; y[c("0", "2")] # numerikus és karakteres indexelés
#>  0  2
#> 18 20
#>  0  2
#> 18 20
```

Egyetlen attribútum lekérdezésére és beállítására az `attr()` függvényt is használhatjuk. Az `attr()` függvényben meg kell adnunk az elérendő attribútum nevét is.

```
# names attribútum kezelése
attr(x, "names") <- c("A", "B", "C", "D", "E")
attr(x, "names")
#> [1] "A" "B" "C" "D" "E"
attributes(x)
#> $names
#> [1] "A" "B" "C" "D" "E"
```

Attribútumok törlésére a `NULL` értéket használjuk.

```
names(x) <- NULL             # names attribútum törlése
attr(x, "names") <- NULL     # names attribútum törlése
attributes(x) <- NULL        # az összes attribútum törlése
```

A `dim` attribútum *mátrix* és *tömb* adatszerkezet kísérő attribútuma. A `dim` attribútum megadja az objektum dimenzióit, vagyis a sorok és oszlopok számát. A `dim` attribútum értéke egy numerikus vektor, amelynek hossza megegyezik a dimenziók számával. A `dim` attribútum beállításához a `attr()` függvényt használhatjuk.

```
x <- 1:12                       # integer vektor
x
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12
attr(x, "dim") <- c(2,6)        # integer mátrix (2x6-os)
attributes(x)
#> $dim
#> [1] 2 6
x
#>      [,1] [,2] [,3] [,4] [,5] [,6]
#> [1,]    1    3    5    7    9   11
#> [2,]    2    4    6    8   10   12
attr(x, "dim") <- c(2, 3, 2)    # integer tömb (2x3x2-es)
attributes(x)
#> $dim
#> [1] 2 3 2
x
#> , , 1
#>
#>      [,1] [,2] [,3]
#> [1,]    1    3    5
#> [2,]    2    4    6
#>
#> , , 2
#>
#>      [,1] [,2] [,3]
#> [1,]    7    9   11
#> [2,]    8   10   12
```

A fenti példa megmutatja, hogy a `dim` attribútum hozzáadásával a vektorból mátrixot, illetve tömböt is készíthetünk.

Amennyiben a `dim` mellett szerepeltetjük a `dimnames` attribútumot is, akkor figyelnünk kell, hogy a `dimnames` attribútum egy listát tartalmazzon. A lista annyi elemet kell, hogy tartalmazzon, ahány dimenziós az objektum (például mátrix esetén kettőt: sor- és oszlopneveket). Az egyes elemek pedig karakteres vektorok, amelyek megadják a dimenzió mentén az elnevezéseket.

```
x <- 1:12                       # integer vektor
attr(x, "dim") <- c(2,6)        # integer mátrix (2x6-os)
attr(x, "dimnames") <- list(nem=c("férfi", "nő"), osztaly=LETTERS[1:6])
```

```
x
#>        osztaly
#> nem     A B C D  E  F
#>   férfi 1 3 5 7  9 11
#>   nő    2 4 6 8 10 12
attributes(x)
#> $dim
#> [1] 2 6
#>
#> $dimnames
#> $dimnames$nem
#> [1] "férfi" "nő"
#>
#> $dimnames$osztaly
#> [1] "A" "B" "C" "D" "E" "F"
```

Már meg sem lepődünk azon, hogy a faktorok esetében a szintek – a faktor kategóriái – az objektum egy speciális attribútumaként jelennek meg. A `levels` attribútum a faktor típusú objektumok egyik legfontosabb jellemzője.

```
# faktor készítése
x <- factor(c("piros", "zöld", "kék", "piros"))
x
#> [1] piros zöld  kék   piros
#> Levels: kék piros zöld
attributes(x) # a faktor attribútumai
#> $levels
#> [1] "kék"   "piros" "zöld"
#>
#> $class
#> [1] "factor"
```

Látjuk, hogy a faktor esetén egy másik, `class` nevű attribútum is megjelenik. Mielőtt megismerjük ezt a kitüntetett attribútumot, ejtsünk szót olyan attribútumról, amelyet az `attributes()` függvény nem listáz ki. Az eddig megismert attribútumok, a `names`, `dim`, `dimnames`, `levels` és következő alfejezetben bemutatott `class` attribútumok explicit attribútumoknak tekinthető.

Létezik két implicit (belső) attribútum (a hossz és a típus), amellyel minden objektum rendelkezik, ellentétben az fenti explicit attribútumokkal, amelyek szerepeltetése opcionális lehet az egyes objektumok esetén.

A *hossz* (*length*) attribútum azt mutatja, hogy hány elem található egy vektorban, listában vagy más objektumban. Lekérdezése és esetleges beállítása is a `length()` függvénnyel történik. A másik attribútum a *típus* (*type*), amely az objektum belső típusát adja meg. A típus lekérdezésére a `typeof()` függvényt használhatjuk.

```
x <- 1:5       # integer vektor
attributes(x)  # nincs explicit attribútuma
#> NULL
length(x)      # a hossz implicit attribútum
#> [1] 5
typeof(x)      # a típus implicit attribútum
#> [1] "integer"
```

A fenti példa megmutatja, hogy a `length()` és a `typeof()` függvényekkel lekérdezhetjük az implicit attribútumokat, miközben az objektumok explicit attribútummal nem rendelkezik.

5.5.2. Osztályok

A faktor objektum attribútumainak megjelenítésekor láttuk, hogy a kategóriákat tartalmazó `levels` attribútum mellett egy `class` attribútum is megjelenik.

```
# faktor készítése
x <- factor(c("piros", "zöld", "kék", "piros"))
x
#> [1] piros zöld  kék   piros
#> Levels: kék piros zöld
attributes(x) # a faktor attribútumai
#> $levels
#> [1] "kék"   "piros" "zöld"
#>
#> $class
#> [1] "factor"
```

A `class` attribútumra épül az R legelterjedtebb objektum-orientált rendszere az S3. Az R nyelv számos objektum-orientált megközelítést támogat, például az S4, R7, S7 rendszereket, de az S3 a legrégebbi és a legjobban támogatott. Az S3 rendszer a hagyományos objektum-orientált programozás (OOP) megközelítéseit követi, de nem olyan szigorúan, mint más nyelvekben. Az S3 rendszerben az objektumok osztályait a `class` attribútum határozza meg, amely egy karakteres vektor, amely az objektum osztályait tartalmazza. A fenti példában az `x` objektum

a `factor` osztály egy példányának tekinthető, mivel a `class` attribútuma a `factor` karakteres vektort tartalmazza.

Az objektumok osztályai a `class()` függvénnyel is lekérdezhetők, nem kell mindig a `attributes()` függvényt használni. A `class()` függvény egyszerűen az objektum `class` argumentumával tér vissza. Azoknál az objektumoknál, amelyek nem rendelkeznek `class` argumentummal, a `class()` visszatérési értéke a következőképpen alakul:

- `"numeric"` lesz a visszatérési érték, ha az objektum *integer* vagy *double* vektor,
- `"array"` és/vagy `"matrix"`, ha az objektum rendelkezik `dim` attribútummal,
- más esetben a `typeof()` visszatérési értékével tér vissza a `class()` függvény.

Nem hívtuk fel rá a figyelmet, de a korábban tárgyalt számos típus mindegyike valójában osztály, ilyen a `Date`, `difftime`, `POSIXct`, `POSIXlt` és a `table` is.

```
# dátum típus: Date osztályú objektum
x <- as.Date("2020-03-12")
attributes(x)
#> $class
#> [1] "Date"
class(x)
#> [1] "Date"
```

```
# difftime típus: difftime osztályú objektum
x <- Sys.Date() - as.Date("2020-03-12")
x
#> Time difference of 1957 days
attributes(x)
#> $class
#> [1] "difftime"
#>
#> $units
#> [1] "days"
class(x)
#> [1] "difftime"
```

```
# POSIXct típus: POSIXct osztályú objektum
x <- ISOdate(year = 2020, month = 12, day = 2)
x
#> [1] "2020-12-02 12:00:00 GMT"
attributes(x)
```

```
#> $class
#> [1] "POSIXct" "POSIXt"
#>
#> $tzone
#> [1] "GMT"
class(x)
#> [1] "POSIXct" "POSIXt"
```

```
# POSIXlt típus: POSIXlt osztályú objektum
x <- as.POSIXlt(x)
x
#> [1] "2020-12-02 12:00:00 GMT"
attributes(x)
#> $names
#>  [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"
#>  [8] "yday"   "isdst"  "zone"   "gmtoff"
#>
#> $class
#> [1] "POSIXlt" "POSIXt"
#>
#> $tzone
#> [1] "GMT"
#>
#> $balanced
#> [1] TRUE
class(x)
#> [1] "POSIXlt" "POSIXt"
```

```
# table típus: table osztályú objektum
x <- table(sample(LETTERS[1:3], 100, replace = T))
x
#>
#>  A  B  C
#> 32 41 27
attributes(x)
#> $dim
#> [1] 3
#>
#> $dimnames
#> $dimnames[[1]]
```

```
#> [1] "A" "B" "C"
#>
#>
#> $class
#> [1] "table"
class(x)
#> [1] "table"
```

Mi a jelentősége annak, hogy egy objektum valamilyen osztályba vagy osztályokba tartozik? Az R nyelvben az osztályoknak köszönhetően az objektumok különböző típusú műveletekhez és függvényekhez rendelhetők. Például a `plot()` függvény a `class` attribútum alapján dönti el, hogy milyen típusú grafikonokat készít az objektumból: faktor esetén oszlopdiagramot, numerikus vektor esetén pontdiagramot. Ugyanilyen a `summary()` függvény is, amely a `class` attribútum alapján dönti el, hogy milyen típusú összegzést készít az objektumból. Nézzünk utóbbira példát.

```
# faktor készítése
x <- factor(c("piros", "zöld", "kék", "piros"))
x
#> [1] piros zöld  kék   piros
#> Levels: kék piros zöld
summary(x) # faktor összegzése
#>   kék piros  zöld
#>     1     2     1
```

```
# numerikus vektor készítése
x <- c(1, 2, 3, 4, 5)
x
#> [1] 1 2 3 4 5
summary(x)  # numerikus vektor összegzése
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>       1       2       3       3       4       5
```

```
# karakteres vektor készítése
x <- c("a", "b", "c", "d", "e")
x
#> [1] "a" "b" "c" "d" "e"
summary(x)  # karakteres vektor összegzése
#>    Length     Class      Mode
#>         5 character character
```

```
# adattábla másolása
x <- survey
summary(x)  # adattábla összegzése
#>      Sex          Wr.Hnd          NW.Hnd        W.Hnd          Fold
#>  Female:118   Min.   :13.00   Min.   :12.50   Left : 18   L on R : 99
#>  Male  :118   1st Qu.:17.50   1st Qu.:17.50   Right:218   Neither: 18
#>  NA's  :  1   Median :18.50   Median :18.50   NA's :  1   R on L :120
#>               Mean   :18.67   Mean   :18.58
#>               3rd Qu.:19.80   3rd Qu.:19.73
#>               Max.   :23.20   Max.   :23.50
#>               NA's   :1       NA's   :1
#>      Pulse             Clap       Exer       Smoke         Height
#>  Min.   : 35.00   Left   : 39   Freq:115   Heavy: 11   Min.   :150.0
#>  1st Qu.: 66.00   Neither: 50   None: 24   Never:189   1st Qu.:165.0
#>  Median : 72.50   Right  :147   Some: 98   Occas: 19   Median :171.0
#>  Mean   : 74.15   NA's   :  1              Regul: 17   Mean   :172.4
#>  3rd Qu.: 80.00                            NA's :  1   3rd Qu.:180.0
#>  Max.   :104.00                                        Max.   :200.0
#>  NA's   :45                                            NA's   :28
#>        M.I           Age            Weight         Tomegindex
#>  Imperial: 68   Min.   :16.75   Min.   : 50.00   Min.   :14.44
#>  Metric  :141   1st Qu.:17.67   1st Qu.: 63.00   1st Qu.:21.00
#>  NA's    : 28   Median :18.58   Median : 75.00   Median :25.52
#>                 Mean   :20.37   Mean   : 75.53   Mean   :25.82
#>                 3rd Qu.:20.17   3rd Qu.: 88.00   3rd Qu.:30.24
#>                 Max.   :73.00   Max.   :100.00   Max.   :41.66
#>                                                  NA's   :28
```

A fenti példa alapján világos, hogy ugyanaz a függvény a `class` attribútum (vagy annak hiányában a `class()` által visszaadott érték) alapján különböző típusú összegzéseket készít az objektumokból. Faktor esetén egy gyakorisági táblázatot, numerikus vektor esetén egy statisztikai összegzést, minimummal, maximummal és átlaggal.

Az R nyelvben a `class` attribútum megváltoztatására is van lehetőség, sőt saját osztályokat is bevezethetünk és a generikus függvények (mint például a `print()`, `summary()` és `plot()` függvények, amelyeket a `class` attribútum alapján hív meg az R) számára speciális viselkedést írhatunk elő. Bevezethetjük például a `mi_a_hossz` nevű osztályt, amely az objektum értékének kiírása esetén először az objektum implicit hossz attribútumát jeleníti meg. Az új osztály létrehozásához a `class()` függvényt használhatjuk.

```r
# hagyományos működés
x <- 1:5
x  # valójában a print(x) hívódik
#> [1] 1 2 3 4 5
print(x)
#> [1] 1 2 3 4 5
```

Látható, hogy az objektumok értékének megjelenítése során a `print()` függvény hívódik meg, még akkor is, amikor nem szerepeltetjük explicit módon a parancssorban. Az R nyelvben a `print()` függvény a generikus függvények közé tartozik, amelynek működése az objektum `class` attribútumától függ. A `print()` függvény a `class` attribútum alapján dönti el, hogy milyen módon jelenjem meg az output. Ha a `mi_a_hossz` osztály számára egy speciális megjelenítést akarunk előírni, nevezetesen az objektum hosszának kiírását kezdeményezzük a szokásos objektum érték megjelenítése előtt, akkor annyi a dolgunk, hogy létrehozunk egy speciális nevű függvényt, amely név a `print` és az adott osztály (jelen esetben `mi_a_hossz`) nevéből áll. Azaz, ha a `mi_a_hossz` osztályú objektumok kiírásakor a `print()` függvény helyett a `print.mi_a_hossz()` függvényt szeretnénk használni, amely a `mi_a_hossz` osztályú objektumok megjelenítésére szolgál, akkor a következőképpen járunk el:

```r
# speciális viselkedés előírása
print.mi_a_hossz <- function(x, ...) {
  cat("Hossz:", length(x), "\n") # a hossz kiírása
  print(unclass(x))              # a print() függvény hívása az eredeti objektumra
}
```

Hogyan tudjuk felhasználni ezt a speciális viselkedésű `print()` függvényt? Mindössze egy objektumhoz hozzá kell rendelni a `mi_a_hossz` osztályt, amelyet a `class()` függvény segítségével tehetünk meg.

```r
# speciális print() működés
x <- 1:5
class(x) <- "mi_a_hossz" # osztály hozzárendelése
x  # valójában a print(x) hívódik
#> Hossz: 5
#> [1] 1 2 3 4 5
print(x)
#> Hossz: 5
#> [1] 1 2 3 4 5
```

A fenti példában a `print()` függvény a `mi_a_hossz` osztályú objektumok kiírásakor a `print.mi_a_hossz()` függvényt hívja meg, amely először kiírja az objektum hosszát, majd a szokásos módon megjeleníti az objektum értékét is. Az `unclass()` függvény eltávolítja az objektum `class` attribútumát, így a `print()` függvény a szokásos módon jeleníti meg az objektum értékét.

A fenti példa csupán szemléltette az S3 rendszer működését az R-ben, sokkal kifinomultabb eszközök és megoldások állnak rendelkezésünkre az objektum-orientált programozás során. Az S3 rendszer lehetővé teszi, hogy saját osztályokat és saját generikus függvényeket hozzunk létre, amelyek aztán osztályoktól függő, speciális viselkedéseket valósítanak meg. Ezeket a lehetőségeket számos szakkönyvben megtaláljuk. vegyük figyelembe, hogy az ismert adatszerkezetek esetén a `class` attribútumok módosítása nem ajánlott.

Összefoglalás

Az R-ben minden objektumnak -– legyen az például vektor, mátrix, lista vagy faktor -– lehetnek attribútumai, amelyek speciális információkat tartalmaznak az objektumhoz kapcsolódóan. Ilyen attribútum például a `names`, `dim`, `dimnames`, `levels` és a `class`. Ezeket az `attributes()` vagy az `attr()` függvénnyel kérdezhetjük le, illetve módosíthatjuk. A `names` attribútum lehetővé teszi a vektor elemeinek elnevezését, a `dim` és `dimnames` mátrixok és tömbök szerkezetét írja le, míg a `levels` a faktorok kategóriáit adja meg. A `class` attribútum pedig az objektum típusát határozza meg, és kulcsszerepet játszik az R S3 nevű objektum-orientált rendszerében. Lehetővé teszi, hogy ugyanaz a függvény (például `summary()` vagy `print()`) az objektum típusától függően eltérően viselkedjen. Ezt a típus szerinti viselkedést a generikus függvények biztosítják.

Feladatok

1. Milyen szakkönyvekből tudnánk még jobban megismerni az R nyelv objektum-orientált programozási lehetőségeit?
2. Mutassunk példát arra, hogy a `class` attribútum hibás módosításával, hogyan tudunk kárt okozni faktor és adattábla adatszerkezetek esetén!
3. Hozzunk létre egy `osszeg` nevű osztályt numerikus vektorokhoz, amely a `print()` függvény hívásakor először kiírja az elemek összegét, majd a szokásos módon megjeleníti az objektum értékét is. Teszteljük az új osztályt!

II. rész
Adatkezelés

6. Beolvasás


Ha ez lenne az utolsó nap az életedben, mit csinálnál?
Beülnék statra... az lenne életem leghosszabb napja!

6.1. Alapvető formátumok 🙂

> **Miről lesz szó?** Ebben a fejezetben
>
> - áttekintjük a táblázatkezelők állományainak beolvasását,
> - a tagolt szöveges állományok fogalmát,
> - és azok beolvasását az *Alap R* segítségével.

Az R-ben adatokkal dolgozunk, amelyek beolvasására és kiírására az R számos eljárást kínál. Adatokat beolvashatunk a billentyűzetről, a vágóasztalról és külső adatforrásból, állományból vagy adatbázisból is. Az R-ben feldolgozott adatokat a vágóasztalra, adatbázisba vagy állományba írhatjuk ki. Ebben a fejezetben csak a két legtermészetesebb beolvasási módszert ismertetjük, az adatok beolvasást táblázatkezelők (például Microsoft Excel, LibreOffice Calc) állományaiból és tagolt szöveges állományokból.

6.1.1. Táblázatkezelők

A táblázatkezelők saját állományai (például `.xlsx` és `.ods`) kezelhetők a legkényelmesebben az adatelemzés során. Az adatbevitel és a rögzített adatok karbantartása, későbbi módosítása ebben a formátumban a legegyszerűbb, ráadásul a számítógép alapú tesztek/kérdőívek sokszor ilyen típusú állományokba írják a válaszokat. A táblázatkezelők saját állományait olyan munkafüzetnek tekintjük, amely munkalapokból áll, így akár több adatbázist is tárolhatunk egyetlen állományban. Egy Excel vagy LibreOffice Calc adatmátrix esetén tudnunk kell, hogy az melyik munkalapon helyezkedik el, ritkábban pedig azt is, melyik tartományban foglal helyet a beolvasandó adatbázis.

Az Excel és LibreOffice Calc adattáblák beolvasást a `{rio}` csomag `import()` függvényével végezzük, melynek egyetlen kötelező argumentuma a fájl elérési útja (`file=`).

Előkészítettünk egy 4 munkalapos Excel és LibreOffice Calc adatbázist (`agatha_christie_m.xlsx` és `agatha_christie_m.ods`), amelynek mindegyik munkalapján ugyanazt az adatbázist találjuk meg, de egyre zajosabb környezetben.

Az 1. munkalapon nincsenek zavaró cellák, csupán a adatbázisunk értékes adatcellái az `A1` cellától kezdődően. Ebben az esetben nincs szükség más argumentumra, csak az állomány elérési útjára.

```
library(rio)
ac.1 <- import(file = "adat/agatha_christie_m.xlsx") # MS Excel
ac.1 <- import(file = "adat/agatha_christie_m.ods")  # LibreOffice Calc
```

A 2. munkalapon már nem az `A1` cellában kezdődik az adatbázis, de továbbra sincs zavaró egyéb cella. Az `import()` megtalálja az adatbázist a munkalapon az Excel adatbázis esetében, de a LibreOffice Calc adattábla beolvasásához pontosítani kell az adatbázis helyét. A tartomány közvetlen megadásával (`range="F7:I52"`) utasítjuk az `import()` függvényt, hogy honnan olvassa be az adatbázisunkat. Természetesen a megadott cellatartomány az oszlopneveket is tartalmazza annak első sorában.

Az `XLSX` és az `ODS` beolvasása közötti eltérés rávilágít az `import()` függvény működésére. Nem maga az `import()` végzi a közvetlen beolvasást, hanem okosan kiválasztja a beolvasandó állomány kiterjesztése alapján, hogy melyik csomag, melyik konkrét függvényét hívja. Az Excel állományokat a *Tidyverse R* `{readxl}` csomag `read_excel()` függvénye fogja beolvasni, a LibreOffice Calc állományokat a `{readODS}` csomag `read_ods()` függvénye, amelyek hívását már az `import()` végzi. Mivel két különböző függvény dolgozik a háttérben, így az `XLSX` és az `ODS` állományokat beolvasó parancs paraméterezése is eltérhet, és esetünkben el is tér. A `file=` argumentum azonban közös, és természetesen a munkalap sorszámát is meg kell adnunk, amely mindkét esetben a `sheet=2`-vel történik.

```
ac.2 <- import(file = "adat/agatha_christie_m.xlsx", sheet=2)
ac.2 <- import(file = "adat/agatha_christie_m.ods",  sheet=2, range="F7:I52")
```

A 3. munkalapon már az első 6 sor zavaró, nem az adatbázishoz tartozó adatokat tartalmaz, így azokat elegendő kihagyni (`skip=6`) a beolvasásból az Excel esetében, míg az `ODS`-hez továbbra is a tartomány pontos megadása szükséges a sikeres beolvasásához.

```
ac.3 <- import(file = "adat/agatha_christie_m.xlsx", sheet=3, skip=6)
ac.3 <- import(file = "adat/agatha_christie_m.ods",  sheet=3, range="F7:I52")
```

A 4. munkalapon már rendkívül terhelt az adatbázisunk a környező zavaró cellákról, így közvetlenül a tartomány megadásával (`range="F7:I52"`) utasítjuk az `import()` függvényt Excel esetében is, hogy honnan olvassa be az adatbázisunkat.

```
ac.4 <- import(file = "adat/agatha_christie_m.xlsx", sheet=4, range="F7:I52")
ac.4 <- import(file = "adat/agatha_christie_m.ods",  sheet=4, range="F7:I52")
```

Jegyezzük meg, ha csak tehetjük, adatainkat táblázatkezelő programmal hozzuk létre és annak saját formátumában (`XLSX` vagy `ODS`) tároljuk.

6.1.2. Tagolt szöveges állomány

A tagolt szöveges állományok kitüntetett szerepet játszanak a statisztikai adatfeldolgozásban, ugyanis minden statisztikai programcsomag és táblázatkezelő be tud olvasni ilyen formátumú állományokat, és ki tud exportálni ilyen formátumba. A tagolt szöveges állományok létrehozásához pedig egy jegyzettömbszerű szövegszerkesztő is elegendő, tehát ez a formátum elég nagy szabadságot ad az adataink kezeléshez.

6.1.2.1. Tagolt szöveges állomány létrehozása

A tagolt szöveges állomány egy egyszerű, formázást nem tartalmazó szöveges állomány, amelyet azonos szerkezetű sorok alkotnak. A sorokat az operációs rendszernek megfelelő sorvége karakterek zárják. Jegyzettömbszerű szövegszerkesztő használata során, az `Enter` leütésével ezek a sorvége karakterek kerülnek fizikailag a szöveges állományba. Linux és macOS operációs rendszer alatt az `LF` karakter, Windows platformon a `CR` és `LF` karakterek. Ezeket sorvége karaktereknek hívjuk, az `LF` a soremelés (`\n`), a `CR` a kocsi vissza (`\r`) karakter. Annak ellenére, hogy különböző platformokon más-más jelzi a sorvégét, a beolvasó függvények felismerik ezeket, és helyesen értelmezik. Ezzel nekünk nem kell külön foglalkoznunk.

Minden tagolt szöveges állományban van egy kitüntetett karakter, a tagoló karakter. Ez tipikusan a pontosvessző (`;`), a szóköz, a tabulátor (`\t`) vagy a vessző (`,`) karakter. A tagolt szöveges állomány összes sorában ezek egyikét használjuk az adatértékek elválasztására, ráadásul minden sorban azonos számú adatértéknek kell szerepelni, ennek megfelelően minden sorban azonos számú tagoló karakter van.

Nézzünk példát pontosvesszővel tagolt szöveges állományra:

```
nem;kor;pulzus
fiú;12;71
fiú;11;69
lány;14;70
```

Összesen 4 sora van, soronként 3 adatértékkel, és 2 pontosvesszővel. Látható, hogy az első sor kitüntetett, azaz igazából nem méréssel kapott adatokat tartalmaz, hanem megnevezi azokat. Gyakori, hogy a tagolt szöveges állományok első sora ilyen speciális fejlécsor, amely oszlopneveket tartalmaz. Ez nem kötelező, elképzelhető, hogy az első sorban már közvetlenül adatértékek helyezkednek el.

Nézzünk egy szóközzel tagolt szöveges állományt:

```
nem kor pulzus atlag
fiú 12 71 3,92
fiú 11 69 4,12
lány 14 70 5,00
```

Ez a tagolt szöveges állomány 4 sort, soronként 4 adatértéket és 3 elválasztó szóközt tartalmaz. Az első sora fejlécsor. Látható, hogy tizedes törtek is szerepelnek az állományban, a tizedesvesszőt valóban vesszővel jelöltük. Ez nem minden esetben van így, tizedes pont is elválaszthatná az egész és tört részt.

Tekintsünk egy elsőre kicsit rendezetlen szöveges állományt:

```
Általános iskolai felmérés
2019.03.02.

#2.b
nem,kor,pulzus,atlag
fiú,12,71,3.92
#fiú,11,69,4.12
lány,14,70,5.00 # ellenőrizni
```

Az állomány második felében fedezhetünk fel rendezettséget, a fejlécet 4 oszlopnévvel, és alatta a sorokat 4-4 adatértékkel. Ez a rész olyan, mint egy vesszővel elválasztott szöveges állomány, ahol a tizedes törtekben pontot használunk az egész és tört rész elválasztására.

Az állomány eleje azonban egyáltalán nem hasonlít a tagolt szöveges állományokra, és ráadásul kettős kereszttel (`#`), vagyis megjegyzésnek szánt szövegekkel tarkított soraink is vannak. Az ilyen szabadabb stílusú állományok is beolvashatók az R-be, csupán meg kell adnunk, hogy az elejéről hány sort hagyjon figyelmen kívül (3 sort), és mit tekintsen megjegyzésnek (a kettős keresztet) a beolvasását végző eljárás. Ebben az esetben a fejlécen kívül még 2 adatsor lesz a beolvasott adattáblában.

Melyek a tagolt szöveges állományok fontos jellemzői:

- a tagoló karakter,
- a decimális elválasztó,
- van-e fejlécsor,
- hány sort lépjünk át az elejéről,
- mi a megjegyzés karakter,
- milyen kódolású az állomány.

Ahogyan említettük tagolt szöveges állományt egy jegyzettömbszerű szövegszerkesztővel mi is létrehozhatunk, de ha módunk van rá, akkor a minél kényelmesebb adatbevitel miatt, használjunk táblázatkezelőt, és az abban elkészült táblázatos formában lévő adatokat exportáljuk ki tagolt szöveges állományba. Például magyar Excel esetén választhatjuk a `CSV (pontosvesszővel tagolt)`, vagy a `Szöveges (tabulátorral tagolt)` formátumot. Érdemes minden esetben megőrizni a táblázatkezelő saját formátumában az adatokat, mert a kényelmes szerkesztés miatt továbbra is abban érdemes az adatokat módosítani, de a változtatások után, utolsó lépésként, végezzük el az exportot tagolt szöveges állományba.

6.1.2.2. A `read.table()` család

Tagolt szöveges állományok beolvasásának hagyományos módja a `read.table()` függvénycsalád használata. Azért nevezzük függvénycsaládnak, mert valójában több függvényt használhatunk, amelyek csak a paraméterek alapértelmezett értékében térnek el egymástól:

- `read.table(  sep="",   dec=".")`
- `read.csv(    sep=",",  dec=".")`
- `read.csv2(   sep=";",  dec=",")`
- `read.delim(  sep="\t", dec=".")`
- `read.delim2( sep="\t", dec=",")`

Érdemes a fenti függvénynevek megtanulása helyett egyetlen függvényt, a `read.table()`-t használni, és inkább a lehetséges paraméterek jelentését jegyezzük meg:

- `file=` - A beolvasandó állomány elérési útja. A könyvtárakat a `/` karakterrel válasszuk el egymástól.
- `sep=` - A tagoló karakter a beolvasandó állományban. Tipikus értékei: `sep=";"`, `sep=" "`, `sep=","` és tabulátor elválasztó esetén `sep="\t"`.
- `dec=` - A decimális elválasztó, vagyis a tizedesvessző alakja az állományban. Tipikusan a `dec=","` beállítást kell használnunk, de előfordulhat, hogy pont a tizedes elválasztó, így a `dec="."` paraméterre van szükségünk.
- `header=` - Ha van fejléc a szöveges állományban, akkor a `header=TRUE`, egyébként a `header=FALSE` beállítást használjuk.
- `na.strings=` - A hiányzó érték jelölése a szöveges állományban. Az alapértelmezett beállítás a legtöbb esetben megfelelő, hiszen a `""` (semmi) és az `"NA"` jelölésből alapértelmezés szerint hiányzó érték lesz.
- `comment.char=` - A tipikus megjegyzés karakter a `#`, de meg tudjuk változtatni, ha szükséges.
- `skip=` - A szöveges állomány első néhány sorát figyelmen kívül hagyhatjuk.
- `stringsAsFactors=` - A `stringsAsFactors=TRUE` beállítással elérhetjük a karakteres oszlopok automatikus faktorrá konvertálását.

- `fileEncoding=` - A beolvasandó szöveges állományt alkotó karakterek kódolási szabványát adhatjuk meg. Tipikusan az UTF-8 kódolású szöveges állományok beolvasása során kell használnunk (`fileEncoding="UTF-8"`), de a magyar környezetben készült szöveges állományok esetében is rögzíthetjük a kódolási szabványt (`fileEncoding="latin2"`).
- `strip.white=` - A felesleges szóközök és tabulátorok eltávolítása az adatok elejéről és végéről.
- `quote=` - A szöveges állományban lévő adatvédő idézőjelek alakja. Alapértelmezés szerint a `'` szimpla és a `"` dupla idézőjeleket is figyeli a `read.table()`, amely sok esetben hibás beolvasásához vezethet. Ha nincs külön adatvédő idézőjel az állományban, akkor használjuk a `quote=""` beállítást, ha van, akkor pontosan adjuk meg (`quote="'"` vagy `quote='"'`).

Végezzük el az `egyetem.csv` tagolt szöveges állomány beolvasását, amelynek első néhány sora a következő:

```
hallgato;Height;neme;lefekves;felkeles;Drink
1;67;female;-2,5;5,5;víz
2;64;female;1,5;8;üdítő
3;61;female;-1,5;7,5;tej
```

A beolvasandó állományunk egy első sorában oszlopneveket tartalmazó szöveges állomány, amelynek a tartalmát a következő parancsok egy `df` adattáblában helyezik el. Az állományban az adatokat (és az oszlopneveket is) pontosvessző (`;`) választja el, a numerikus értékekben pedig tizedesvessző szerepel.

```
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/egyetem.csv"
df <- read.table(file = data.file, header=T,
                                   sep=";",
                                   dec=",",
                                   strip.white = T,
                                   stringsAsFactors = F,
                                   fileEncoding = "latin2")
str(df)
#> 'data.frame':    657 obs. of  6 variables:
#>  $ hallgato: int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Height  : num  67 64 61 61 70 63 61 64 66 65 ...
#>  $ neme    : chr  "female" "female" "female" "female" ...
#>  $ lefekves: num  -2.5 1.5 -1.5 2 0 1 1.5 0.5 -0.5 2.5 ...
#>  $ felkeles: num  5.5 8 7.5 8.5 9 8.5 7.5 7.5 7 8.5 ...
#>  $ Drink   : chr  "víz" "üdítő" "tej" "víz" ...
```

Összefoglalás

A statisztikai munka első lépése az adatmátrix beolvasása. Adatainkat legkényelmesebb Excel vagy LibreOffice Calc táblázatkezelők saját formátumú adatállományiban tárolni (`XLSX`, `ODS`), de néha nem kerülhetjük el a tagolt szöveges állományok használatát. A táblázatkezelők adatállományait a `{rio}` csomag `import()` függvényével olvashatjuk be, a tagolt szöveges állományokat az *Alap R* `read.table()` függvénnyel.

Feladatok

1. Olvassuk be a leniency.xls Excel állományt, állapítsuk meg hány sora és oszlopa van.
2. Olvassuk be a socsupport.csv tagolt szöveges állományt, állapítsuk meg hány sora és oszlopa van.
3. Olvassuk be a tv.txt tagolt szöveges állományt, állapítsuk meg hány sora és oszlopa van.

6.2. A *Tidyverse R* és az inline beolvasás

Miről lesz szó? Ebben a fejezetben

- áttekintjük a tagolt szöveges állományok *Tidyverse R* beolvasását,
- és az inline beolvasás eseteit.

6.2.1. A `read_delim()` függvénycsalád

A *Tidyverse R* is képes a tagolt szöveges állományok beolvasására, és rendszerint gyorsabb, jobban paraméterezhető lehetőséget nyújt. Lényeges különbség az előző részben látott `read.table()` családhoz képest, hogy alapértelmezés szerint *tibble* típusú adattábla a beolvasás eredménye.

A `read_delim()` család tagjai:

- `read_delim()`
- `read_csv()`
- `read_csv2()`
- `read_tsv()`

Minden esetben használjuk a `read_delim()` függvényt, amely nagyon hasonló paraméterekkel rendelkezik, mint a `read.table()`:

- `file=` - Ugyanaz, mint a `read.table()` függvénynél.
- `delim=` - Ugyanaz, mint a `read.table()` függvénynél a `sep=` argumentum.
- `locale=` - A decimális elválasztó és a kódolási szabvány beállításához a `locale()` függvényt használjuk. Ha a szokásos vessző a tizedes vessző alakja, akkor a `locale = locale(decimal_mark = ",")`, egyébként a `locale = locale(decimal_mark = ".")` beállítást használjuk. Ha a kódolási szabványt is be szeretnénk állítani a vessző decimális elválasztó mellett, akkor a `locale = locale(decimal_mark = ".", encoding = "UTF-8")` beállításra van szükségünk az UTF-8 beállításához.
- `col_names=` - Ugyanaz, mint a `read.table()` függvénynél a `header=` argumentum.
- `na=` - Ugyanaz, mint a `read.table()` függvénynél a `na.strings=` argumentum.
- `comment=` - Ugyanaz, mint a `read.table()` függvénynél a `comment.char=` argumentum.
- `skip=` - Ugyanaz, mint a `read.table()` függvénynél.
- `trim_ws=` - Ugyanaz, mint a `read.table()` függvénynél a `strip.white=` argumentum.
- `quote=` - Ugyanaz, mint a `read.table()` függvénynél.

Végezzük el a már korábban megismert `egyetem.csv` tagolt szöveges állomány beolvasását a *Tidyverse R* segítségével:

```
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/egyetem.csv"
library(tidyverse)
dt <- read_delim(file = data.file, col_names=T,  delim=";", trim_ws= T,
                  locale=locale(decimal_mark=",",
                                encoding = "latin2"))
str(dt)
#> spc_tbl_ [657 x 6] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
#>  $ hallgato: num [1:657] 1 2 3 4 5 6 7 8 9 10 ...
#>  $ Height  : num [1:657] 67 64 61 61 70 63 61 64 66 65 ...
#>  $ neme    : chr [1:657] "female" "female" "female" "female" ...
#>  $ lefekves: num [1:657] -2.5 1.5 -1.5 2 0 1 1.5 0.5 -0.5 2.5 ...
#>  $ felkeles: num [1:657] 5.5 8 7.5 8.5 9 8.5 7.5 7.5 7 8.5 ...
#>  $ Drink   : chr [1:657] "víz" "üdítő" "tej" "víz" ...
#>  - attr(*, "spec")=
#>   .. cols(
#>   ..   hallgato = col_double(),
#>   ..   Height = col_double(),
#>   ..   neme = col_character(),
#>   ..   lefekves = col_double(),
#>   ..   felkeles = col_double(),
```

```
#>   ..   Drink = col_character()
#>   .. )
#>  - attr(*, "problems")=<externalptr>
```

6.2.2. Inline beolvasás

Ebben a könyvben a külső adatállományokból való beolvasás mellett az inline adatbeolvasást is részletesen bemutatjuk. Kisebb adatbázisok, egyszerűbb adatfeldolgozás esetén az adatokat közvetlenül a parancsállományban is elhelyezhetjük, ezt nevezzük *sorok közötti*, vagy más néven *inline* adatbeolvasásának. A szokásos eset azonban a külső adatállományból való adatbeolvasás.

Az adatok inline beolvasása azt jelenti, hogy nem külső állományból, hanem az R parancsállományba gépelt adatokból indulunk ki. Felhasználjuk a `c()`, `factor()` és a `data.frame()` függvényeket az *Alap R*-ből, valamint a `tibble()` vagy `tribble()` függvényeket a *Tidyverse R*-ből. Esetleg használhatjuk az állományok beolvasását végző, most megismert `read.table()` (*Alap R*) és `read_delim()` (*Tidyverse R*) függvénycsaládokat is.

A legegyszerűbb adatfeldolgozási feladatok egyetlen változót érintenek, ezek pedig numerikus vektorban vagy faktorban tárolhatók az R-ben. Ez az inline beolvasás legegyszerűbb esete.

```
# 4 óvodás testmagassága cm-ben
magassag <- c(132, 143, 129, 145)
mean(magassag)  # testmagasság átlaga
#> [1] 137.25

# a 4 óvodás neme
nem <- factor(c("fiú", "lány", "lány", "fiú"), levels=c("fiú","lány"))
table(nem, useNA = "ifany")  # gyakorisági táblázat a nemre
#> nem
#>  fiú lány
#>    2    2
```

Több változó tárolása esetén *adattábla* (*data frame*) vagy *tibble* típusú objektumot hozunk létre a korábban már megismert `data.frame()` és `tibble()` függvények segítségével. Előkészítő lépésként természetesen az oszlopokat alkotó vektorokra is szükség van.

```
library(tidyverse)
magassag <- c(132, 143, 129, 145)
nem <- factor(c("fiú", "lány", "lány", "fiú"), levels=c("fiú","lány"))
df <- data.frame(magassag, nem) # data frame létrehozása
dt <- tibble(magassag, nem)     # tibble létrehozása
```

A létrehozott adattáblák is nagyon hasonlítanak egymásra, és felhasználásuk is azonos módon történik:

```
df  # data frame
#>   magassag  nem
#> 1      132  fiú
#> 2      143 lány
#> 3      129 lány
#> 4      145  fiú
dt  # tibble
#> # A tibble: 4 x 2
#>   magassag nem
#>      <dbl> <fct>
#> 1      132 fiú
#> 2      143 lány
#> 3      129 lány
#> 4      145 fiú
mean(df$magassag)               # testmagasság átlaga
#> [1] 137.25
mean(dt$magassag)               # testmagasság átlaga
#> [1] 137.25
table(df$nem, useNA = "ifany")  # gyakorisági táblázat a nemre
#>
#>  fiú lány
#>    2    2
table(dt$nem, useNA = "ifany")  # gyakorisági táblázat a nemre
#>
#>  fiú lány
#>    2    2
```

Nem kell feltétlenül az adattáblát alkotó oszlopokat külön numerikus vagy faktor oszlopokban előzőleg előkészíteni, ezeket a `data.frame()` vagy `tibble()` argumentumába közvetlenül is beírhatjuk:

```r
# adattábla létrehozása
df <- data.frame(
  magassag = c(132, 143, 129, 145),
  nem = factor(c("fiú", "lány", "lány", "fiú"), levels=c("fiú","lány"))
)
# tibble létrehozása
dt <- tibble(
  magassag = c(132, 143, 129, 145),
  nem = factor(c("fiú", "lány", "lány", "fiú"), levels=c("fiú","lány"))
)
```

Tibble létrehozásának másik módja a `tribble()` függvény, amelynek argumentumába táblázatos formában adhatjuk meg az adatokat. A változóneveket a ~ karakter vezeti be, minden adatértéket és oszlopnevet vessző választ el egymástól a `tribble()` argumentumában.

```r
dt <- tribble(
  ~magassag,  ~nem,
  132, "fiú",
  143, "lány",
  129, "lány",
  145, "fiú"
)
dt
#> # A tibble: 4 x 2
#>   magassag nem
#>      <dbl> <chr>
#> 1      132 fiú
#> 2      143 lány
#> 3      129 lány
#> 4      145 fiú
```

Vegyük észre, hogy a `nem` oszlop a fenti példában karakteres vektor, a faktorrá alakításáról a `factor()` függvénnyel gondoskodhatunk.

```r
dt$nem <- factor(dt$nem, levels = c("fiú", "lány"))
dt
#> # A tibble: 4 x 2
#>   magassag nem
#>      <dbl> <fct>
#> 1      132 fiú
```

```
#> 2      143 lány
#> 3      129 lány
#> 4      145 fiú
```

A `read.table()` és `read_delim()` függvénycsaládokat is használhatjuk inline beolvasásra. Mindkét esetben egy inline, szóközzel tagolt szöveges állományt illesztettünk a legkönnyebben a kódba, ennek megfelelően állítjuk be a tagoló karaktereket. A `read.table()` esetében körbevettük a `textConnection()` függvénnyel is a beillesztett adatokat, erre a *Tidyverse R* `read_delim()` függvényénél már nincs szükség. A `nem` faktorrá konvertálásáról se felejtkezzünk el.

```
# adattábla létrehozása
df <- read.table(file = textConnection("
magassag nem
132 fiú
143 lány
129 lány
145 fiú
"), header=T, sep=" ")
df$nem <- factor(df$nem, levels=c("fiú", "lány"))
df
#>   magassag  nem
#> 1      132  fiú
#> 2      143 lány
#> 3      129 lány
#> 4      145  fiú
# tibble létrehozása
dt <- read_delim("
magassag nem
132 fiú
143 lány
129 lány
145 fiú
", col_names=T, delim=" ")
dt$nem <- factor(dt$nem, levels=c("fiú", "lány"))
dt
#> # A tibble: 4 x 2
#>   magassag nem
#>      <dbl> <fct>
#> 1      132 fiú
```

```
#> 2      143 lány
#> 3      129 lány
#> 4      145 fiú
```

Összefoglalás

A `read_delim()` függvénycsalád a tagolt szöveges állományok beolvasásának *Tidyverse R* verziója. Rendszerint gyorsabb, jobban paraméterezhető lehetőséget nyújt az *Alap R*-hez képest. Előfordul, hogy adatainkat a parancsainkban rejtjük el, a sorok között. Tipikusan kisebb méretű adatmátrixok létrehozására használjuk a `data.frame()`, `tibble()` vagy `tribble()` függvényeket, de szintén az inline beolvasást támogatja a `read.table(file=textConnection())` és a `read_delim()`.

Feladatok

1. Olvassuk be a *Tidyverse R* segítségével a tv.txt tagolt szöveges állományt, állapítsuk meg hány sora és oszlopa van.
2. Olvassuk be *Tidyverse R* segítségével a socsupport.csv tagolt szöveges állományt, állapítsuk meg hány sora és oszlopa van.
3. Olvassuk be a *Tidyverse R* segítségével a leniency.xls Excel állományt, állapítsuk meg hány sora és oszlopa van.
4. Rögzítsük a következő adatbázist inline módon legalább 3 módszerrel! Az adatbázis a magyar utónevek gyakorisági statisztikáját tartalmazza!

ev	helyezes	nev	nem	elso	masodik
2021	1	Levente	fiú	1332	180
2021	2	Máté	fiú	1314	242
2021	3	Dominik	fiú	1289	259
2021	1	Hanna	lány	1355	336
2021	2	Zoé	lány	1138	420
2021	3	Anna	lány	1133	500

6.3. Kiírás és más lehetőségek 🤯

> ℹ Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük a tagolt szöveges állományok kiírását,
> - bináris állományok olvasását és írását,
> - más statisztikai programcsomagok adatállományainak olvasását és írását,
> - és a fix széles mezővel rendelkező állományok olvasását.

6.3.1. Tagolt szöveges állomány

Adattáblák és mátrixok kiírására a `write.table()` függvényt használhatjuk az *Alap R*-ből és a `write_delim()` függvényt a *Tidyverse R*-ből. Mindkét függvény egy-egy függvénycsalád reprezentánsa, de az *Alap R*-ből elegendő ismerni az említett tagot, a *Tidyverse R*-ből pedig az említetten kívül a `write_csv2()` függvényt. Néhány új argumentummal kell megismerkednünk, a korábban tanult argumentumok jelentését még egyszer nem soroljuk fel.

A `write.table()` mátrixok és adattáblák kiírására is alkalmas, míg a `write_delim()` és a `write_csv2()` csak az adattáblákat rögzíti. Az első paraméter (`x=`) a kiírandó objektum neve mindhárom függvény esetében.

Az első példában a `write.table()` függvénnyel egy mátrixot és a korábban létrehozott `df` adattáblát írjuk ki. A `row.names=` és a `col.names=` logikai paraméterek szabályozzák, hogy a sor és oszlopnevek szerepeljenek-e a kimeneti állományban. Ezek alapértelmezett értéke `TRUE`.

```
x.mat <- matrix(1:12, nrow=3) # mátrix létrehozása
# mátrix kiírása
write.table(x.mat, "output/adat/x_mat.txt", col.names=F, row.names=F)
# adattábla kiírása
write.table(x = df, file = "output/adat/df_out.txt", sep = "\t",
            quote = F, dec = ",", row.names = F, col.names = T,
            fileEncoding = "UTF-8")
```

A *Tidyverse R* kiíró függvényei UTF-8 kódolású állományt hoznak létre minden esetben, és a decimális elválasztó alakja `write_delim()` esetében pont, `write_csv2()` esetében pedig vessző. A sornevek soha nem íródnak ki, az oszlopnevek kiírását a `col_names=` argumentummal szabályozhatjuk.

```
library(tidyverse)
# tabulátorral tagolt szöveges állomány létrehozása
write_delim(x = dt, file = "output/adat/tbl_out.txt", delim = "\t", col_names = T)
# pontosvesszővel tagolt szöveges állomány létrehozása
write_csv2(x = dt, file = "output/adat/tbl_out.csv", col_names = T)
```

6.3.2. R objektumok

Az R-rel való munka során sok objektummal dolgozunk, többségük külső állományok beolvasásával jön létre, melyeket aztán a munka során változatos módon manipulálunk. Az *adattábla* és a *tibble* típusú objektumok képezik a statisztikai munka kiinduló pontját. Egyéb objektumok mentéséről eddig nem beszéltünk, pedig a munka során ezek mentése és beolvasása is érdekes lehet.

Egy objektum értékét eltárolhatjuk szöveges állományban a `dput()` függvénnyel, és visszaolvashatjuk a `dget()`-tel:

```
library(MASS)
# survey kiírása txt-be
dput(x = survey, file = "output/adat/dput_out.txt")
# survey beolvasása txt-ből
df <- dget(file = "output/adat/dput_out.txt")
```

Igazán gyors kiírást és visszaolvasást nem várhatunk a szöveges állományoktól, így nagyobb adatbázisok esetében (is) érdemes az objektumok bináris mentését és visszaállítását választani. A `saveRDS()` és a `readRDS()` *Alap R* függvényekkel tudjuk megoldani, hogy az R saját `RDS` formátumú bináris állományába mentsünk ki, majd töltsünk vissza egy-egy objektumot.

```
# survey kiírása bináris állományba
saveRDS(object = survey, file = "output/adat/survey.rds")
# survey beolvasása bináris állományból
df <- readRDS(file = "output/adat/survey.rds")
```

A *Tidyverse R* `write_rds()` és `read_rds()` függvényei ugyanezt a tevékenységet végzik, de alapértelmezés szerint nem tömörítenek, így némileg gyorsabb működést biztosítanak:

```
library(tidyverse)
# survey kiírása bináris állományba
write_rds(x = survey, file = "output/adat/survey_2.rds")
# survey beolvasása bináris állományból
df <- read_rds(file = "output/adat/survey_2.rds")
```

Egyszerre több objektum tárolását is elvégezhetjük az R másik saját bináris formátuma, az `RData` segítségével. A `save()` függvényben felsoroljuk a tárolni kívánt objektumok nevét, és megadunk egy `.RData` kiterjesztésű állományt. A visszaolvasás a `load()` segítségével történik. Figyeljük meg, hogy a `load()` használata során nincs szükség az értékadás (`<-`) operátorra, mert az `RData` állomány tartalmazza az objektumneveket is, így ezekkel a nevekkel jönnek létre a munkaterületen a bináris állományban eltárolt objektumok. Az azonos nevű, már létező objektumokat figyelmeztetés nélkül felülírja a `load()`, így legyünk óvatosak a függvény használatával.

```
# survey és Animals kiírása bináris állományba
save(survey, Animals, file = "output/adat/MASS_2.RData")
# survey és Animals beolvasása bináris állományból
load(file = "output/adat/MASS_2.RData")
```

Az összes objektum, amely pillanatnyilag a munkaterületen tartózkodik, elmenthető a `save.image()` segítségével. Visszatöltés szintén a `load()`-dal lehetséges.

```
# minden objektum mentése bináris állományba a munkaterületről
save.image(file = "output/adat/osszes_obj.RData")
# az objektumok beolvasása a munkaterületre
load(file = "output/adat/osszes_obj.RData")
```

6.3.3. Egyéb adatállományok

6.3.3.1. Az univerzális `rio::import()`

Az R számos más formátumú adatállomány beolvasását támogatja az eddig tanultakon kívül. Például az *Alap R* `{foreign}` csomagja DBF, Stata, Minitab, SPSS, SAS és Epi adatállományokat is be tud olvasni. A *Tidyverse R* `{haven}` csomagja SPSS, Stata, és SAS fájlokat, a `{readxl}` csomagja pedig Excel XLS és XLSX állományokat is. Json állományokat olvashatunk be a `{jsonlite}`, XML állományokat az `{xml2}` csomaggal.

A `{rio}` csomag különleges helyzetben van, ugyanis minden eddig felsorolt adatállomány beolvasását támogatja egyetlen parancs, az `import()` segítségével. A beolvasandó állomány kiterjesztéséből tudni fogja, hogy pontosan milyen módon (melyik csomag melyik függvénye segítségével) olvassa be az adatállományt. Az `import()` támogatja az *adattábla* és a *tibble* létrehozását is a `setclass=` argumentuma segítségével.

Példaképp pontosvesszővel tagolt szöveges, SPSS és XLSX állományokat olvasunk be:

```
library(rio)
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/egyetem.csv"
# CSV beolvasása
df <- import(file = data.file, sep=";", header=T, dec=",")
str(df)
#> 'data.frame':    657 obs. of  6 variables:
#>  $ hallgato: int  1 2 3 4 5 6 7 8 9 10 ...
#>  $ Height  : num  67 64 61 61 70 63 61 64 66 65 ...
#>  $ neme    : chr  "female" "female" "female" "female" ...
#>  $ lefekves: num  -2.5 1.5 -1.5 2 0 1 1.5 0.5 -0.5 2.5 ...
#>  $ felkeles: num  5.5 8 7.5 8.5 9 8.5 7.5 7.5 7 8.5 ...
#>  $ Drink   : chr  "v\xedz" "\xfcd\xedt\xf5" "tej" "v\xedz" ...
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/nepesseg.sav"
# SPSS beolvasása
df <- import(file = data.file)
str(df)
#> 'data.frame':    12 obs. of  4 variables:
#>  $ HONAP   : num  1 2 3 4 5 6 7 8 9 10 ...
#>   ..- attr(*, "label")= chr "Hónap 1994-ben"
#>   ..- attr(*, "format.spss")= chr "F5.0"
#>   ..- attr(*, "display_width")= int 12
#>   ..- attr(*, "labels")= Named num [1:12] 1 2 3 4 5 6 7 8 9 10 ...
#>   .. ..- attr(*, "names")= chr [1:12] "január" "február" "március" "április" ...
#>  $ NEPESSEG: num  10273 10270 10267 10265 10262 ...
#>   ..- attr(*, "label")= chr "Népesség száma hó végén"
#>   ..- attr(*, "format.spss")= chr "F5.0"
#>  $ ELVESZUL: num  10238 9285 10105 9617 9548 ...
#>   ..- attr(*, "label")= chr "Élveszületések száma"
#>   ..- attr(*, "format.spss")= chr "F5.0"
#>  $ HALAL   : num  13888 12825 12516 11753 12328 ...
#>   ..- attr(*, "label")= chr "Halálozások száma"
#>   ..- attr(*, "format.spss")= chr "F5.0"
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/pothoff2.xlsx"
```

```
# XLSX beolvasása
dt <- import(file = data.file, setclass = "tibble")
str(dt)
#> tibble [108 x 5] (S3: tbl_df/tbl/data.frame)
#>  $ person: num [1:108] 1 1 1 1 2 2 2 2 3 3 ...
#>  $ sex   : chr [1:108] "F" "F" "F" "F" ...
#>  $ age   : num [1:108] 8 10 12 14 8 10 12 14 8 10 ...
#>  $ y     : num [1:108] 21 20 21.5 23 21 21.5 24 25.5 20.5 24 ...
#>  $ agefac: num [1:108] 8 10 12 14 8 10 12 14 8 10 ...
```

A `{rio}` csomag univerzális állománykiíró függvénye az `export()`. Szintén a kiírandó állomány kiterjesztése dönti el, hogy pontosan melyik konkrét függvényt fogja működtetni az `export()`, ennek megfelelően a háttérben lévő függvény argumentumaival esetlegesen mi is bővíthetjük az `export()` argumentumlistáját. A következő példákban pontosvesszővel tagolt, SPSS, SAS, XLSX. ODS és RDS adatállományokat hozunk létre:

```
export(x=df, file="output/adat/rio_out.csv", dec=",", sep=";") # CSV
export(x=df, file="output/adat/rio_out.sav")                   # SPSS
export(x=df, file="output/adat/rio_out.sas7bdat")              # SAS
export(x=df, file="output/adat/rio_out.xlsx")                  # Excel
export(x=df, file="output/adat/rio_out.ods")                   # LibreOffice Calc
export(x=df, file="output/adat/rio_out.RDS")                   # RDS
```

6.3.4. Adatok csomagokban

Az adatelemzési munkánk R-ben az adattáblák létrehozásával kezdődik. Az adatokat külső adatállományból többféle módszerrel beolvashatjuk, illetve inline módon is létrehozhatjuk. Azonban számos csomag tartalmaz saját adattáblát, amelyeket például az R megismerése során is használhatunk. A csomagokban elérhető adattáblák nevét és rövid leírását a `data()` függvény segítségével ismerhetjük meg.

```
data(package="MASS")  # a MASS csomagban lévő adattáblák
data()                # betöltött csomagokban lévő adattáblák
# a telepített csomagokban lévő adattáblák
data(package = .packages(all.available = TRUE))
```

Amennyiben egy adattáblára szükségünk van egy csomagból, akkor a csomag betöltése nélkül is elérhetjük az adattáblát:

```
data(survey, package="MASS") # adattábla elérése csomagból
head(survey[1:6], n=3)       # az első 3 sor az első 6 oszlopból
#>      Sex Wr.Hnd NW.Hnd W.Hnd   Fold Pulse
#> 1 Female   18.5   18.0 Right R on L    92
#> 2   Male   19.5   20.5  Left R on L   104
#> 3   Male   18.0   13.3 Right L on R    87
```

A szokásos eljárás azonban a csomag betöltése, amely után az adattábla nevét szabadon használhatjuk:

```
library(tidyr)
smiths
#> # A tibble: 2 x 5
#>   subject     time   age weight height
#>   <chr>      <dbl> <dbl>  <dbl>  <dbl>
#> 1 John Smith     1    33     90   1.87
#> 2 Mary Smith     1    NA     NA   1.54
```

Ha az adattábla részletesebb leírására vagyunk kíváncsiak egy betöltött csomagból, akkor a `?` operátort vagy a `help()` függvényt is használhatjuk:

```
?survey                      # a survey leírása (MASS be van töltve)
help(topic = "smiths")       # a smiths leírása (tidyr be van töltve)
help(smiths, package="tidyr")# a smiths leírása (tidyr nincs betöltve)
```

6.3.5. Fix széles mezők

Ritkábban szükség lehet fix széles mezőket tartalmazó szöveges állományok beolvasására is. A `read.fwf()` (*Alap R*) és `read_fwf()` (*Tidyverse R*) függvények gondoskodnak arról, hogy az egyes mezőkben lévő adatokat úgy tudjuk azonosítani, hogy az előre rögzített, minden sorban azonos karakterekben mért szélességet adjuk meg a paraméterben.

Tekintsük a következő fix mezőszélességekkel rendelkező állományt:

```
nev;telefon;kor
    Ági+3630459785921,2
 Zoltán+3630459785942,4
    Bea+3630459785938,6
```

Az állomány tartalmaz egy fejlécsort, ami az oszlopok elnevezését segíti, és most pontosvesszővel tagolt. Ez a sor még nem tartozik a fix széles adatmezőkhöz. A következő három sorban azonban 7 pozíción a nevet, 12 pozíción a telefonszámot, 3 pozíción az egy tizedesre pontos életkort soroljuk fel.

Ideiglenesen hozzuk létre ezt az állományt magunk is a `cat()` függvénnyel. A `tempfile()` függvényt használjuk egy a rendszerünkben érvényes ideiglenes állomány nevének meghatározására. A `cat()` függvénnyel egy 4 soros szöveges állományt hozunk létre. Az első sor pontosvesszővel elválasztott oszlopneveket tartalmaz, a következő három sor pedig 3 fix széles adatmezőt.

```
file <- tempfile()  # ideiglenes állománynév
cat(file = file, "nev;telefon;kor",
                 "   Laci+3630459785921,2",
                 " Marika+3630459785942,4",
                 "Barbara+3630459785938,6", sep="\n")
```

A beolvasást először az *Alap R* `read.fwf()` függvényével végezzük el. A `width=` paraméterében kell megadnunk az egyes mezők hosszát. A függvény a megadott mezőhossz értékek alapján egy ideiglenes, tabulátorral elválasztott szöveges állományt hoz létre, amely a `read.table()` függvénnyel kerül ténylegesen feldolgozásra. A `header=TRUE` paraméterrel jelezzük, hogy az első sor oszlopneveket tartalmaz, a `sep=` paraméter pedig az első sorban használt elválasztó karaktert jelöli. A `sep=` paraméterre csak akkor van szükség, ha oszlopneveket tartalmazó sort is be akarunk olvasni. Láthatjuk, hogy a függvény által visszaadott adattábla 3 sort és 3 oszlopot tartalmaz.

```
# fix széles mezőkkel rendelkező szöveges állomány beolvasása
df <- read.fwf(file = file, header=T, sep=";", dec=",", strip.white=T,
               widths=c(7,12,4),
               colClasses=c("character", "character", "double"),
               fileEncoding = "UTF-8")
df
#>       nev      telefon  kor
#> 1    Laci +36304597859 21.2
#> 2  Marika +36304597859 42.4
#> 3 Barbara +36304597859 38.6
str(df)
#> 'data.frame':    3 obs. of  3 variables:
#>  $ nev    : chr  "Laci" "Marika" "Barbara"
#>  $ telefon: chr  "+36304597859" "+36304597859" "+36304597859"
#>  $ kor    : num  21.2 42.4 38.6
```

A *Tidyverse R* `read_fwf()` függvénye nagyon hasonlóan működik. Nem támogatja az oszlopnevek kiolvasását az állományból, így az első sort átlépjük (`skip=1`) és az oszlopneveket a `col_names=` argumentumban soroljuk fel, a szélességek megadására használt `fwf_width()` függvényben. Az oszlopok típusát itt is megadjuk a `col_types=` argumentumban.

```
library(tidyverse)
# fix széles mezőkkel rendelkező szöveges állomány beolvasása
dt <- read_fwf(file = file, skip=1,
               locale=locale(decimal_mark=",", encoding = "UTF-8"),
               col_positions =
                 fwf_widths(widths = c(7, 12, 4),
                            col_names = c("nev", "telefon", "kor")),
                col_types = "ccd")
dt
#> # A tibble: 3 x 3
#>   nev     telefon        kor
#>   <chr>   <chr>        <dbl>
#> 1 Laci    +36304597859  21.2
#> 2 Marika  +36304597859  42.4
#> 3 Barbara +36304597859  38.6
```

Összefoglalás

Adataink számos formátumban állhatnak rendelkezésre az adatelemzés elején, és szükség lehet adataink exportjára is a publikációhoz. Univerzális lehetőséget kínál a `{rio}` csomag, amely az `import()` és az `export()` függvényével a lehető legegyszerűbben oldja meg a fenti műveleteket. Ha R környezetek között szeretnénk az adatok gyors átvitelét biztosítani, akkor a bináris formátumot használó `.RDS` vagy `.Rdata` kiterjesztésű állományokat használjuk. Ezek írását és olvasását az *Alap R*, a *Tidyverse R* és a `{rio}` csomag is támogatja. Ritkábban szükség lehet csomagban tárolt adatok, vagy fix széles adatmezőket tartalmazó szöveges állományok használatára.

Feladatok

1. A `cat()` függvénnyel a `dput()`-hoz hasonlóan szöveges állományba írhatjuk egy karakteres, numerikus vagy logikai vektor értékét. Mindkét függvénnyel végezzük el a kiírást, és vessük össze a kapott szöveges állományok tartalmát!
2. A Kaggle egyik adatbázisában 4000 videójáték értékelése található. Töltsük le a CSV adatállományt, és nyissuk meg. Keressük meg az R-bloggers oldalon az adatállományhoz kapcsolódó cikket, és próbáljunk ki néhány elemző parancsot.

A blogger melyik csomag, melyik függvényével végezte a beolvasást?

3. Keressük fel és tanulmányozzuk a Great R packages for data import, wrangling and visualization oldalt! A bevezetésben lefektetett alapelvek közül melyiket erősíti meg ez az oldal?
4. Töltsünk le 10 érdekesnek tűnő adatállományt a TidyTuesday - A weekly social data project in R oldalról, és nyissuk meg őket!
5. Töltsük be az `{Ecdat}` csomagot, ha szükséges telepítsük! Vizsgáljuk meg az `{Ecdat}` csomag tartalmát, nevezzünk meg legalább 5 objektumot ebből a csomagból! Olvassuk be `d2` néven az `{Ecdat}` csomag `Diamond` adattábláját! Kérjünk leírást erről az adattábláról!

7. Adatmanipuláció

7.1. Adatkezelés az *Alap R*-ben 🙂

> **Miről lesz szó? Ebben a fejezetben**
>
> - megismerjük adatelőkészítés munkafázisait *Alap R* környezeteben, így
> - megtanuljuk az információkérés parancsait,
> - az oszlopok elnevezését, típusának és sorrendjének megváltoztatását,
> - a sorok manipulációját, rendezését és szűrését, valamint
> - a hiányzó adatok kezelését.

Ebben a fejezetben az adattáblák manipulációját tekintjük át, melyek az adatkezelés szempontjából a legfontosabb R műveletek. Mint korábban láttuk, az adattáblák a mátrixhoz hasonlóan sorokat és oszlopokat tartalmaz, illetve a listához hasonlóan elemekből, méghozzá azonos hosszúságú oszlopvektorokból épülnek fel (5.1. ábra). Az adattábla kettős eredete jelentősen megkönnyíti az adataink kezelését. Az adattáblákkal kapcsolatos alapvető műveleteket a 5.3.7. fejezetben már áttekintettük, érdemes ezeket feleleveníteni. Hogyan hozhatunk létre inline adattáblát, hogyan indexeljük, szűrjük és rendezzük az adattáblákat, sőt a 6. fejezetben azt is megtanultuk, hogyan tudunk Excel vagy CSV állományból adattáblát beolvasni.

Az adattábla sorai egyedekre (személyek, tárgyak, dolgok stb.) vonatkozó megfigyelések, az oszlopok pedig a megfigyelt tulajdonságok. A statisztikához közelebbi fogalmakkal, az adattáblában az adatmátrixunkat/többdimenziós mintánkat rögzíthetjük, a sorok a mintaelemek, az oszlopok a megfigyelt változók.

Az adattábla inhomogén adatszerkezet, oszlopai különböző típusú adatokat is tartalmazhatnak. Jellemzően kvalitatív (nominális és ordinális skálán mért) adatok tárolására a faktort használjuk, kvantitatív (intervallum és arányskálán mért) adatok tárolására a numerikus vektort. Természetesen adattáblában karakteres és logikai vektorok is szerepelhetnek, sőt dátumokat és időpontokat is kezelhetünk az adattáblában.

Az adatmanipuláció számos különböző tevékenységet felölel. A következő fejezetekben a legfontosabb adatkezelési feladatokat mutatjuk be, amelyek közé tartozik információkérés a beolvasott adatokról, a változók típusának ellenőrzése és módosítása, az oszlopok elnevezése és sorrendjének megváltoztatása, új oszlopok létrehozása és meglévő oszlopok törlése, valamint az adatok rendezése és szűrése, továbbá a hiányzó adatok kezelése.

7.1.1. Információ gyűjtése

Az adatbázis beolvasása (6. fejezet) után következik az információk begyűjtése a beolvasott adatokról. A legfontosabb információkérő függvényeket a 7.1. táblázat tartalmazza. Az

7.1. táblázat: A legfontosabb információkérő függvények *(saját szerkesztés)*

Függvény	Leírás	Példa
`str(object)`	szerkezet kiírása	`str(df)`
`dim(x)`	x dimenziói	`dim(df)`
`nrow(x)`	x sorainak száma	`nrow(df)`
`ncol(x)`	x oszlopainak száma	`ncol(df)`
`names(x)`	x elemeinek neve	`names(df)`
`colnames(x)`	x oszlopnevei	`colnames(df)`
`rownames(x)`	x sornevei	`rownames(df)`
`head(x,n=6)`	x első sorai	`head(df)`
`tail(x,n=6)`	x utolsó sorai	`tail(df)`
`View(x)`	x teljes tartalma	`View(df)`
`class(x)`	x típusa	`class(df);class(df$oszlop)`
`length(x)`	x hossza	`length(df);length(df$oszlop)`
`unique(x)`	x különböző értékei	`unique(df$oszlop)`
`table(...,useNA)`	gyakorisági tábla	`table(df$oszlop,useNA='ifany')`
`summary(object)`	leíró adatok	`summary(df);summary(df$oszlop)`

információ megszerzésének célja az egyszerű tájékozódáson kívül a beolvasás helyességének ellenőrzése. Kíváncsiak lehetünk, hogy rendelkezésre áll-e a kívánt sor- és oszlopszám, az oszlopnevek rendben vannak-e, a numerikusnak szánt változók valóban számokat tartalmaznak-e és a karakteres oszlopokban az esetleges magyar ékezetek rendben megjelennek-e?

Egy konkrét adatelemzési munka során az adatok beolvasása előtt már sok ismerettel rendelkezünk az adatbázisról, de most képzeljük el, hogy egy ismeretlen `flow.xlsx` adatbázist kell felfedeznünk.

Példa 7.1 (Flow adatbázis felfedezése)**.** Olvassuk be a `flow.xlsx` fájlt, és ismerkedjünk meg az adatbázissal! Az adatbázis egy kérdőíves felmérés eredményeit tartalmazza, amely a válaszadók életkorát, nemét, családi állapotát, iskolai végzettségét és egy *flow* kérdőív 20 kérdésére adott válaszait tartalmazza. A válaszok 1-5-ig terjedő skálán értékelhetők, ahol az 1-es a "legrosszabb" és az 5-ös a "legjobb" választ jelenti.

Végezzük el az adatok beolvasását a már ismert `rio::import()` függvény segítségével, és ismerkedjünk meg az adatbázissal. Ehhez az adatbázis szerkezetét és típusát feltáró `str()` és `class()` függvényeket használjuk, valamint a sor- és oszlopszámot megadó `dim()` függvényt.

```
flow <- rio::import(file = "adat/flow.xlsx") # beolvasás
str(flow)    # a teljes szerkezet
#> 'data.frame':    100 obs. of  25 variables:
#>  $ alkatoi.tev    : chr  "Igen" "Igen" "Nem" "Nem" ...
#>  $ kor            : num  26 20 22 21 21 25 53 21 22 21 ...
#>  $ nem            : chr  "Férfi" "Nő" "Nő" "Férfi" ...
#>  $ csaladi.allapot: chr  "Egyedülálló" "Egyedülálló" "Élettársi kapcso ...
#>  $ isk.vegz       : chr  "Egyetem" "Gimnázium" "Gimnázium" "Gimnázium" ...
#>  $ flow.1         : num  2 4 4 3 5 5 5 4 5 5 ...
#>  $ flow.2         : num  4 4 4 4 5 5 5 4 4 5 ...
#>  $ flow.3         : num  5 5 3 3 5 5 3 4 4 5 ...
#>  $ flow.4         : num  5 5 4 4 2 5 3 4 5 5 ...
#>  $ flow.5         : num  5 5 5 3 3 5 5 5 5 5 ...
#>  $ flow.6         : num  1 4 2 3 4 1 3 2 2 4 ...
#>  $ flow.7         : num  2 4 1 4 4 3 5 3 5 5 ...
#>  $ flow.8         : num  4 5 4 4 4 5 4 4 5 5 ...
#>  $ flow.9         : num  5 5 3 4 5 5 3 4 5 5 ...
#>  $ flow.10        : num  4 5 4 3 5 5 5 4 4 5 ...
#>  $ flow.11        : num  4 5 2 3 5 5 3 3 5 5 ...
#>  $ flow.12        : num  4 4 1 3 4 5 5 4 5 5 ...
#>  $ flow.13        : num  5 4 2 4 5 3 5 4 5 5 ...
#>  $ flow.14        : num  3 4 2 4 5 5 3 3 5 5 ...
#>  $ flow.15        : num  3 5 3 4 4 5 5 4 5 5 ...
#>  $ flow.16        : num  3 3 3 3 4 2 4 2 3 5 ...
#>  $ flow.17        : num  4 5 3 4 3 2 4 4 5 5 ...
#>  $ flow.18        : num  5 5 4 4 2 5 5 5 5 5 ...
#>  $ flow.19        : num  4 5 4 2 5 5 5 2 5 5 ...
#>  $ flow.20        : num  3 4 1 2 4 5 1 2 3 5 ...
class(flow)  # típus
#> [1] "data.frame"
dim(flow)    # sor- és oszlopszám
#> [1] 100  25
```

A fenti sorok után világossá válik, hogy egy 100 sort és 25 oszlopot tartalmazó *adattábla* (*data frame*) áll rendelkezésre. Az oszlopnevek a `names()` és a `colnames()` függvényekkel is megismerhetők.

```
names(flow)[1:4]  # az első 4 oszlop neve
#> [1] "alkatoi.tev"     "kor"             "nem"
#> [4] "csaladi.allapot"
```

```
colnames(flow)[5:8]  # a következő 4 oszlop neve
#> [1] "isk.vegz" "flow.1"   "flow.2"   "flow.3"
```

Az oszlopnevek viszonylag beszédesek, de jobban is megismerhetjük ezeket a változókat.

```
class(flow$alkatoi.tev)  # az 'alkotoi.tev' változó típusa
#> [1] "character"
unique(flow$alkatoi.tev)  # egyedi értékei
#> [1] "Igen" "Nem"
table(flow$alkatoi.tev, useNA = "ifany")  # gyakorisági táblázata
#>
#> Igen  Nem
#>   39   61
class(flow$kor)  # a kor változó típusa
#> [1] "numeric"
table(flow$kor, useNA = "ifany")  # gyakorisági táblázata
#>
#> 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 36 39 40 42 43 45 50 51
#>  1  2  2  4 22 18  4  4  3  5  4  1  2  1  1  1  1  1  2  1  1  3  3  4
#> 52 53 69
#>  5  3  1
```

A `class()` függvény megmutatja, hogy az `alkatoi.tev` oszlop típusa karakteres vektor (`character`), míg a `kor` oszlop típusa numerikus vektor (`numeric`). A `unique()` függvény a változó egyedi értékeit adja vissza, ami egyszerű ellenőrzést tesz lehetővé, hogy a változó tényleg az elvárt értékeket tartalmazza-e. A `table()` függvény a gyakorisági táblázatot készíti el, amely megmutatja, hogy az egyes értékekből hány darab található az oszlopban. A `useNA="ifany` argumentum segítségével megadhatjuk, hogy a hiányzó értékek is szerepeljenek-e a táblázatban, amennyiben előfordulnak az adott oszlopban.

Az adatbázis tetszőleges részét megjeleníthetjük a konzolban a szokásos indexelés segítségével:

```
d.df[sorindex, oszlopindex]
```

Például 4 sorból (az 1., 2., 50. és 51. sorból) és 5 oszlopból (az 1., 2., 3., 4. és 5. oszlopból) álló részhalmazt kérhetünk le:

```
# az adattábla egy résztáblája
flow[c(1:2, 50:51), 1:5]
#>    alkatoi.tev kor   nem csaladi.allapot  isk.vegz
#> 1         Igen  26 Férfi     Egyedülálló   Egyetem
#> 2         Igen  20    Nő     Egyedülálló Gimnázium
#> 50         Nem  22    Nő     Egyedülálló Gimnázium
#> 51        Igen  39    Nő           Házas   Egyetem
```

Ne feledjük, hogy az *RStudio*-ban az `Environment` fülön is megjelenik a `flow` adatbázis a sikeres beolvasás után. Kattintva a neven a teljes adatbázist áttekinthetjük a bal felső részben (ezt a `View(flow)` paranccsal is kezdeményezhetjük), de a `flow` előtti ikonon kattintva megjelenít-hetjük az adatbázis `str()`-ból ismert szerkezetét is.

További kényelmi lehetőség, ha a *tibble* típus barátságosabb megjelenítését is kihasználjuk. Ehhez konvertáljuk át az adattáblánkat *tibble* típusúvá, és egyszerűen jelenítsük meg az objektumot:

```
library(tidyverse)
flow.tbl <- as_tibble(flow)  # tibble típusú adatbázis létrehozása
flow.tbl  # kényelmes megjelenítés
#> # A tibble: 100 x 25
#>   alkatoi.tev   kor nem   csaladi.allapot  isk.vegz flow.1 flow.2 flow.3
#>   <chr>       <dbl> <chr> <chr>            <chr>     <dbl>  <dbl>  <dbl>
#> 1 Igen           26 Férfi Egyedülálló      Egyetem       2      4      5
#> 2 Igen           20 Nő    Egyedülálló      Gimnázi~      4      4      5
#> 3 Nem            22 Nő    Élettársi kapcs~ Gimnázi~      4      4      3
#> 4 Nem            21 Férfi Egyedülálló      Gimnázi~      3      4      3
#> 5 Igen           21 Nő    Egyedülálló      Gimnázi~      5      5      5
#> 6 Igen           25 Nő    Elvált           Egyetem       5      5      5
#> # i 94 more rows
#> # i 17 more variables: flow.4 <dbl>, flow.5 <dbl>, flow.6 <dbl>,
#> #   flow.7 <dbl>, flow.8 <dbl>, flow.9 <dbl>, flow.10 <dbl>,
#> #   flow.11 <dbl>, flow.12 <dbl>, flow.13 <dbl>, flow.14 <dbl>,
#> #   flow.15 <dbl>, flow.16 <dbl>, flow.17 <dbl>, flow.18 <dbl>,
#> #   flow.19 <dbl>, flow.20 <dbl>
```

7.1.2. Oszlopnevek módosítása

A sikeres beolvasás és a szükséges tájékozódás után az oszlopnevek áttekintése és esetleges módosítása a következő lépés. Ez kulcsfontosságú a további munka szempontjából, ugyanis a

jól megválasztott változónevek jelentősen meggyorsíthatják a további munkát, és fordítva, a kevésbe beszédes, következetlen, a túl rövid vagy túl hosszú változónevek akadályozhatják a sikeres adatelemzést. A változónevek mindig legyenek beszédesek, csak az angol ábécé kisbetűit és számjegyeket használjunk, több részből álló neveket aláhúzással (_), esetleg ponttal (.) tagoljuk (R kódolási stílus).

Egy adattábla oszlopait a `names()` vagy `colnames()`, a sorait a `rownames()` függvény használatával kérdezhetjük le és nevezhetjük át. A sornevek egymástól különböző, karakteres vagy numerikus egész értékek lehetnek, míg az oszlopnevek csak karakteres adatok.

Az oszlopnevek módosításának több oka lehet. Példánkban magyar változónevekre váltunk, de sokszor rövidítjük vagy beszédesebbé tesszük az oszlopaink nevét.

Példa 7.2 (Metál bandák adatbázisa)**.** Olvassuk be a `metal_bandak.xlsx` fájlt, és egységesítsük az oszlopok elnevezését! Az adatbázis a metál bandák alakulásának és felbomlásának időpontját, valamint a zenekarok származási országát tartalmazza.

A `bandak` adatbázis beolvasásához most is a `rio::import()` függvényt használjuk.

```
bandak <- rio::import(file = "adat/metal_bandak.xlsx")
bandak
#>       banda  fan formed split      origin
#> 1     Kiuas  106   2000  2013  Finnország
#> 2    Accept  681   1968  <NA> Németország
#> 3 Metallica 4122   1981  <NA>         USA
#> 4    Zonata   23   1998  2003  Svédország
#> 5   Therion 1266   1987  <NA>  Svédország
names(bandak)  # az összes oszlop neve
#> [1] "banda"  "fan"    "formed" "split"  "origin"
```

Látható, hogy a `bandak` adatbázisban az oszlopok nevei nem egységesek, és nem is beszédesek. Az oszlopnevek átírásához a `names()` vagy `colnames()` függvényeket használhatjuk. Az oszlopok átnevezésére több lehetőségünk van:

- átnevezhetünk egyetlen oszlopot úgy, hogy sorszámmal hivatkozunk rá,
- átnevezhetünk több oszlopot úgy, hogy sorszámmal hivatkozunk rájuk,
- átnevezhetünk egy oszlopot úgy, hogy a régi névre keresünk rá.

```
# a 2. oszlop nevének átírása
names(bandak)[2] <- "rajongo"
# a 3-5. oszlopok átnevezése
```

```
names(bandak)[3:5] <- c("alakulas", "felbomlas", "orszag")
# a 'banda' nevű oszlop átnevezése
names(bandak)[names(bandak) == "banda"] <- "nev"
bandak  # az átalakított adatbázis kiírása
#>         nev rajongo alakulas felbomlas      orszag
#> 1     Kiuas     106     2000      2013  Finnország
#> 2    Accept     681     1968      <NA> Németország
#> 3 Metallica    4122     1981      <NA>         USA
#> 4    Zonata      23     1998      2003  Svédország
#> 5   Therion    1266     1987      <NA>  Svédország
```

7.1.3. Oszlopok elérése

Az *Alap R*-ben az oszlopok elérése (indexelése) a `[` vagy `[[` operátor segítségével történhet. Mivel az adattáblák örökölték a kétdimenziós mátrix és az egydimenziós lista adatszerkezet indexelési lehetőségeit, így az oszlopokra négyféle módon hivatkozhatunk:

```
adattábla[,oszlopindex]  # hivatkozás egy vagy több oszlopra
adattábla[oszlopindex]   # hivatkozás egy vagy több oszlopra
adattábla[[oszlopnév]]   # hivatkozás egyetlen oszlopra
adattábla$oszlopnév      # hivatkozás egyetlen oszlopra
```

Az fenti példa első sorában mátrixszerűen, második sorában listaszerűen indexelünk. A továbbiakban a mátrixszerű, azaz vesszőt tartalmazó hivatkozást részesítjük előnyben. Az oszlopindex lehet numerikus vektor pozitív vagy negatív értékekkel, karakteres vektor, vagy akár logikai vektor is. Ha csak egyetlen oszlopra vagyunk kíváncsiak, akkor a `[[` vagy még gyakrabban a `$` operátort használjuk az oszlop nevének megadásával.

Az oszlopok elérése mindennapos a statisztikai munka során, így ezeket az indexelési formákat ismernünk kell. Például az átlagéletkor kiszámításához a teljes `kor` oszlopra szükség van, így a legegyszerűbb a `flow$kor` hivatkozás használata. A `summary()` függvény segítségével a `kor` és a `flow.1` oszlopok leíró statisztikai adatait kérhetjük le, így a `summary(flow[,c("kor", "flow.1")])` hivatkozás használata a legcélszerűbb. A `grepl()` függvény segítségével a `flow` adatbázisban a `flow` szót tartalmazó oszlopok számát is megkaphatjuk.

```
mean(flow$kor, na.rm = T)  # kor átlaga
#> [1] 29.26
summary(flow[, c("kor", "flow.1")])  # kor és flow.1 leíró statisztikai adatai
#>       kor            flow.1
```

```
#>  Min.   :17.00   Min.   :1.00
#>  1st Qu.:21.00   1st Qu.:3.00
#>  Median :23.00   Median :4.00
#>  Mean   :29.26   Mean   :4.01
#>  3rd Qu.:33.00   3rd Qu.:5.00
#>  Max.   :69.00   Max.   :5.00
# 'flow' szót tartalmazó oszlopok száma
ncol(flow[grepl(pattern = "flow", x = names(flow))])
#> [1] 20
```

Ne feledjük, hogy mátrixszerű indexelés során is kaphatunk egydimenziós eredményt, hiszen ha egyetlen oszlopra hivatkozunk, akkor a `[` operátor automatikusan az egydimenziós vektorra vált a kétdimenziós adattábla helyett. Ezt a `drop=F` használatával akadályozhatjuk meg.

```
flow[1:3, c("kor", "nem")]  # két oszlop, nincs dimenzióvesztés
#>   kor   nem
#> 1  26 Férfi
#> 2  20    Nő
#> 3  22    Nő
flow[1:3, "kor"]  # egy oszlop, dimenzióvesztés
#> [1] 26 20 22
flow[1:3, "kor", drop = F]  # egy oszlop, nincs dimenzióvesztés
#>   kor
#> 1  26
#> 2  20
#> 3  22
```

7.1.4. Oszlopok sorrendje

Ha már jól ismerjük az oszlopok indexelését, akkor számos további műveletre nyílik lehetőség. Ezek közül a legegyszerűbb az oszlopok sorrendjének megváltoztatása. Ha az oszlopindex hivatkozásai az eredeti oszlopsorrendtől eltérnek, akkor máris új oszlopsorrendet határoztunk meg.

```
# a korábbi 1. oszlop (alkotoi.tev) átkerül a 3. oszlopba
flow <- flow[, c(2, 3, 1, 4:25)]
flow[1:2, 1:3]  # az első 3 oszlop kiírása, az első két sorból
#>   kor   nem alkatoi.tev
```

```
#> 1  26 Férfi        Igen
#> 2  20    Nő        Igen
```

Természetesen oszlopsorszámok helyett változóneveket is használhatunk. A `bandak` adattábla esetén például a `rajongo` oszlopot áttehetjük az adattábla végére. Ehhez először is meg kell határoznunk az új oszlopsorrendet, amelyben a `rajongo` oszlop az utolsó helyen szerepel. Az új oszlopsorrend megadásához a `c()` függvényt használjuk.

```
bandak  # emlékeztetőül a bandák adattábla
#>         nev rajongo alakulas felbomlas      orszag
#> 1     Kiuas     106     2000      2013  Finnország
#> 2    Accept     681     1968      <NA> Németország
#> 3 Metallica    4122     1981      <NA>         USA
#> 4    Zonata      23     1998      2003  Svédország
#> 5   Therion    1266     1987      <NA>  Svédország
# a rajongo oszlop a végére kerül
bandak <- bandak[, c("nev", "alakulas", "felbomlas", "orszag", "rajongo")]
bandak
#>         nev alakulas felbomlas      orszag rajongo
#> 1     Kiuas     2000      2013  Finnország     106
#> 2    Accept     1968      <NA> Németország     681
#> 3 Metallica     1981      <NA>         USA    4122
#> 4    Zonata     1998      2003  Svédország      23
#> 5   Therion     1987      <NA>  Svédország    1266
```

7.1.5. Oszlopok létrehozása és törlése

Láttuk korábban a 5.3.4.4. fejezetben, hogy a `cbind()` segítségével oszlopokat adhatunk a meglévő adatbázisunkhoz. Például a meglévő `bandak` adatbázishoz adjunk hozzá egy kétoszlopos új adatbázist, amely a bandák Wikipédia oldalának címét, és egy szubjektív rangsort tartalmaz.

```
bandak.kieg <-
  data.frame(wikipedia=c("https://en.wikipedia.org/wiki/Kiuas",
                         "https://hu.wikipedia.org/wiki/Accept",
                         "https://hu.wikipedia.org/wiki/Metallica",
                         "https://en.wikipedia.org/wiki/Zonata",
                         "https://en.wikipedia.org/wiki/Therion_(band)"),
```

```
            rangsor=c(2,4,1,3,5))
bandak.2 <- cbind(bandak, bandak.kieg)
str(bandak.2)
#> 'data.frame':    5 obs. of  8 variables:
#>   $ nev      : chr  "Kiuas" "Accept" "Metallica" "Zonata" ...
#>   $ alakulas : chr  "2000" "1968" "1981" "1998" ...
#>   $ felbomlas: chr  "2013" NA NA "2003" ...
#>   $ orszag   : chr  "Finnország" "Németország" "USA" "Svédország" ...
#>   $ rajongo  : num  106 681 4122 23 1266
#>   $ rangsor  : num  2 4 1 3 5
#>   $ wikipedia: chr  "https://en.wikipedia.org/wiki/Kiuas" "https://hu. ...
#>   $ rangsor  : num  2 4 1 3 5
```

Egyetlen oszlop beszúrására is van lehetőségünk, és hasonlóan törölhetünk egyetlen oszlopot is:

```
adattábla$új.oszlopnév <- értékek # új oszlop beszúrása az adattábla végére
adattábla$oszlopnév    <- NULL    # oszlop törlése
```

Szúrjuk be a `rangsor` változót az eredeti `bandak` adatbázisba, majd távolítsuk el.

```
bandak$rangsor <- c(2,4,1,3,5)  # új oszlop beszúrása
bandak$rangsor <- NULL          # oszlop törlése
```

7.1.6. Típuskonverzió

Az oszlopok nevének és sorrendjének optimális beállítása után meg kell vizsgálnunk, hogy az oszlopaink típusa megfelel-e az általa reprezentált statisztikai változók mérési skálájának. Nem léphetünk tovább az elemzés felé, amíg ez az összefüggés nem teljesül.

7.2. táblázat: A statisztikai mérési skálák és az R típusok összefüggése *(saját szerkesztés)*

Változó mérési skálája	R típus
nominális	faktor
ordinális	faktor (rendezett)
intervallum	numerikus vektor
arány	numerikus vektor

7.3. táblázat: A típuskonverziók leggyakoribb esetei *(saját szerkesztés)*

Cél mérési skála	Típuskonverzió	R függvény
nominális	numerikusból faktor	`factor(x)`
	karakteresből faktor	
ordinális	numerikusból faktor (rendezett)	`ordered(x)`
	karakteresből faktor (rendezett)	
intervallum arány	karakteresből numerikus	`as.numeric(x)`
	faktorból numerikus	`as.numeric(as.character(x))`

A típuskonverzió leggyakoribb esete, amikor numerikus vagy karakteres vektorból hozunk létre faktort. Ezek bemutatásával kezdünk, de mindegyik, a 7.3. táblázatban szereplő esetre mutatunk egy példát.

```
# faktor változó létrehozása numerikus vektorból
factor(c(3, 3, 2, 3, 3, 3, 2))
#> [1] 3 3 2 3 3 3 2
#> Levels: 2 3
```

```
# faktor változó létrehozása karakteres vektorból
factor(c("Dohányzik", "Dohányzik", "Nem dohányzik"))
#> [1] Dohányzik     Dohányzik     Nem dohányzik
#> Levels: Dohányzik Nem dohányzik
```

```
# rendezett faktor változó létrehozása numerikus vektorból
ordered(c(3, 3, 2, 3, 3, 3, 2))
#> [1] 3 3 2 3 3 3 2
#> Levels: 2 < 3
```

```
# rendezett faktor változó létrehozása karakteres vektorból
ordered(c("alap", "alap", "közepes", "magas"))
#> [1] alap    alap    közepes magas
#> Levels: alap < közepes < magas
```

```
# intervallum/arány változó létrehozása karakteres vektorból
as.numeric(c("44", "39", "55", "38"))
#> [1] 44 39 55 38
```

```
# intervallum/arány változó létrehozása faktorból
as.numeric(as.character(ordered(c(3, 3, 2, 3, 3, 3, 2))))
#> [1] 3 3 2 3 3 3 2
```

A legtöbb időt talán a karakteres vektorból faktorrá alakítás veszi el az adatelőkészítés során. A következő lehetőségeket kell számításba venni a típuskonverzió során:

- mi legyen a faktor szintjeinek neve,
- mi legyen a faktor szintjeinek sorrendje,
- milyen értékeket zárjunk ki, vagyis tegyünk `NA`-vá az átalakítás során.

A `factor()` és az `ordered()` függvények `levels=` argumentumában megadhatjuk a létrejövő faktor szintjeinek sorrendjét, a `labels=` argumentumában pedig a szintek nevét véglegesíthetjük. Az `exclude=` argumentumban megadhatjuk, hogy mely értékeket zárjunk ki az átalakítás során.

Induljunk ki egy iskolai végzettség karakteres vektorból, amely 6 személyre vonatkozóan tartalmaz információt:

```
isk.vegz <- c("alap", "közép", "felső", "alap", "PhD", "felső")
isk.vegz
#> [1] "alap"  "közép" "felső" "alap"  "PhD"   "felső"
```

Készítsünk egy nominális változót, amely az iskolai végzettséget 3 kategóriával méri: alapfokú, középfokú és felsőfokú. Hozzuk létre az új faktort, nevezzük át az egyes szinteket és zárjuk ki a "PhD" értéket:

```
isk.vegz.fakt <- factor(isk.vegz,
                        levels = c("alap", "közép", "felső"),
                        labels = c("Alapfokú", "Középfokú", "Felsőfokú"),
                        exclude = c("PhD"))
isk.vegz.fakt
#> [1] Alapfokú  Középfokú Felsőfokú Alapfokú  <NA>      Felsőfokú
#> Levels: Alapfokú Középfokú Felsőfokú
```

Ugyanezt rendezett faktor esetén is megtehetjük, sőt ez a jobb megoldás, hiszen az iskolai végzettség ordinális változó.

```
isk.vegz.rend.fakt <- ordered(isk.vegz,
                          levels = c("alap", "közép", "felső"),
                          labels = c("Alapfokú", "Középfokú", "Felsőfokú"),
                          exclude = c("PhD"))
isk.vegz.rend.fakt
#> [1] Alapfokú  Középfokú Felsőfokú Alapfokú  <NA>      Felsőfokú
#> Levels: Alapfokú < Középfokú < Felsőfokú
```

Végezzük el a szükséges típuskonverziót a 9.1. példa `flow` adatbázisával is. Emlékeztetőül érdemes megvizsgálni a `flow` adatbázis kérdéses oszlopait. Ezek a 2-5. oszlopok, és a következő kód megmutatja, hogy karakteres vektorokként állnak rendelkezésre.

```
# a kérdéses oszlopok típusa
apply(flow[2:5], 2, class)
#>             nem     alkatoi.tev csaladi.allapot        isk.vegz
#>     "character"     "character"     "character"     "character"
```

A fenti 4 oszlopból a `nem`, az `alkatoi.tev` és a `csaladi.allapot` változókat faktorrá kell alakítanunk. Ezek nominális változók, azonban a szintek sorrendje a kívánt outputok miatt fontos lehet. A szintek sorrendje alapján történik a gyakorisági táblázatok és oszlopdiagramok megjelenítése.

```
# faktorrá alakítás átnevezéssel
flow$nem <- factor(flow$nem,
                   levels = c("Férfi", "Nő"),
                   labels = c("férfi", "nő"))
# faktorrá alakítás átnevezéssel
flow$alkatoi.tev <- factor(flow$alkatoi.tev,
                          levels = c("Igen", "Nem"),
                          labels = c("alkotó", "nem alkotó"))
# faktorrá alakítás átnevezés nélkül, de a sorrend meghatározásával
unique(flow$csaladi.allapot) # ellenőrizzük a lehetséges értékeket
#> [1] "Egyedülálló"              "Élettársi kapcsolatban él"
#> [3] "Elvált"                   "Özvegy"
#> [5] "Házas"
# a sorrend megállapítása, csak a rendezett outputhoz szükséges
flow$csaladi.allapot <-
  factor(flow$csaladi.allapot,
         levels = c("Házas", "Élettársi kapcsolatban él", "Elvált",
                    "Özvegy", "Egyedülálló"))
```

Az `isk.vegz` változót rendezett faktorrá kell alakítanunk, hiszen ez ordinális változó. A helyes szintsorrend megadása kritikus fontosságú.

```
unique(flow$isk.vegz) # ellenőrizzük a lehetséges értékeket
#> [1] "Egyetem"          "Gimnázium"        "Főiskola"
#> [4] "Szakközépiskola"  "Általános iskola"
# a sorrend megállapítása, átnevezés nélkül
flow$isk.vegz <-
  ordered(flow$isk.vegz,
          levels = c("Általános iskola", "Szakközépiskola",
                     "Főiskola", "Egyetem"))
```

7.1.7. Transzformáció

Számos esetben szükség lehet az adattábla oszlopaiban lévő értéke átalakítására (transzformálására). Az értékeket vagy helyben (ugyanabban az oszlopban) változtatjuk meg, vagy új oszlopként szúrjuk be az adattáblába. A transzformációnak 3 lényegesen eltérő típusát különböztethetjük meg:

- *numerikusból-numerikus* transzformáció: egy vagy több numerikus változóból egy új numerikus változót hozunk létre,
- *numerikusból-faktor* transzformáció: egy (vagy több) numerikus változóból egy új karakteres változót hozunk létre,
- *faktorból-faktor* transzformáció: egy (vagy több) faktorból egy új faktor változót hozunk létre.

7.1.7.1. Numerikusból-numerikus transzformáció

A numerikusból-numerikus transzformáció a leggyakoribb, és a legszélesebb körben használt transzformációs típus. Az ilyen típusú transzformációk során a numerikus változókat matematikai műveletekkel alakítjuk át. A leggyakoribb műveletek közé tartozik az összeadás, kivonás, szorzás és osztás, de a logaritmus, négyzetgyök és egyéb matematikai függvények is használhatók. A legtipikusabb példa azonban az egyszerű mértékegységváltás.

Példa 7.3 (Mértékegységváltás)**.** Tekintsük a beépített `{datasets}` csomag `women` adattábláját, amely a `weight` változóban font-ban mért értékeket tartalmaz. Ezt alakítsuk át kg-ban mért adatokká egy új, `suly` oszlopban. Végezzük el az átváltást a `height` változóból (inch) egy új `magassag` (cm) változóba is.

```
# adatok beolvasása, és megtekintése
data(women)
head(women)
#>   height weight
#> 1     58    115
#> 2     59    117
#> 3     60    120
#> 4     61    123
#> 5     62    126
#> 6     63    129
```

Ahogy említettük, a tipikus numerikusból-numerikus transzformáció matematikai műveletek elvégzését jelenti, vagyis az R matematikai operátorait és függvényeit használjuk fel (5.1. és 5.2. táblázat). Most a szorzás operátorra és a `round()` függvényre van szükség.

```
# mértékegységváltás: font-ból kg-ba
women$suly <- round(women$weight*0.45)
head(women) # az új oszlop megtekintése
#>   height weight suly
#> 1     58    115   52
#> 2     59    117   53
#> 3     60    120   54
#> 4     61    123   55
#> 5     62    126   57
#> 6     63    129   58
```

A fenti eredményt a `transform()` függvény segítségével is elérhetjük, amely az *Alap R* függvények közé tartozik. A `transform()` függvény a `data.frame` típusú objektumokkal dolgozik, és az új változókat a `data.frame`-ben lévő változók alapján hozza létre. Ezzel elkerülhetjük az adattábla nevének többszörös megadását, és a kódunk olvashatóbbá válik.

```
# mértékegységváltás: inch-ből cm-be
women <- transform(women, magassag=round(height*2.45))
head(women) # az új oszlop megtekintése
#>   height weight suly magassag
#> 1     58    115   52      142
#> 2     59    117   53      145
#> 3     60    120   54      147
#> 4     61    123   55      149
#> 5     62    126   57      152
#> 6     63    129   58      154
```

Amennyiben a fenti példákban nem új változók létrehozása a cél, hanem már létező változók átalakítása, akkor helyben is elvégezhetjük a transzformációt. Az alábbi példa a `women` adattábla `height` változójának értékeit csökkenti 10 egységgel:

```
# transzformáció helyben: height változó csökkentése
women <- transform(women, height=height-10)
```

A további numerikusból-numerikus transzformációk a kérdőívek tipikus kiértékeléséhez kapcsolódnak. Megismerjük a fordított itemek átalakítását, illetve a skálapontszámok kiszámításának tipikus eseteit. Kezdjük a példával.

Példa 7.4 (Kérdőív kiértékelése)**.** A `flow.xlsx` egy 20 itemes kérdőív adatait is tartalmazza a `flow.1-flow.20` oszlopokban. Végezzük el a kérdőív kiértékelését, ha tudjuk, hogy van két fordított item (5. és 18. item), és a kérdőív két faktort mér (`f1` és `f2`). Az első faktorhoz 11 item tartozik (1, 2, 6, 7, 8, 12, 13, 14, 15, 16, 17), míg a második faktorhoz 9 item (3, 4, 5, 9, 10, 11, 18, 19, 20). A két alskálát és a skálát átlagolással készítjük el.

A 7.4. példa megoldása több numerikusból-numerikus transzformációt tartalmaz. Először végezzük el a fordított itemek átalakítását. Fordítás elvégzéséhez csupán az itemek elvileg lehetséges legkisebb és legnagyobb értékét kell ismernünk. Ezt a két értéket (1 és 5) összeadjuk, és a fordított itemnek jelölt oszlopok értékeit kivonjuk belőle.

```
# a szükséges itemek fordítása: "új item" <- (1+5) - "régi item"
flow$flow.5  <- 6 - flow$flow.5
flow$flow.18 <- 6 - flow$flow.18
```

A `flow` kérdőív kiértékelése átlagolással történik, így a `rowMeans()` függvényt használjuk, amely az egyes sorok átlagát számolja ki. A `na.rm=T` argumentum megadásával figyelmen kívül hagyhatjuk a hiányzó értékeket. (Amennyiben összegzéssel szeretnénk kiértékelni a kérdőívet, akkor a `rowSums()` függvényt kellene használnunk.)

```
# az alskálákhoz tartozó itemek sorszáma
f1 <- c(1, 2, 6, 7, 8, 12, 13, 14, 15, 16, 17)
f2 <- c(3, 4, 5, 9, 10, 11, 18, 19, 20)

# megkeressük az első (flow.1) oszlop előtti oszlop sorszámát
d <- which(names(flow)=="flow.1") - 1

# több numerikus változóból egy új numerikus változó létrehozása (3x)
flow$f1   <- rowMeans(flow[,d+f1], na.rm=T)
```

```r
flow$f2   <- rowMeans(flow[,d+f2], na.rm=T)
flow$ossz <- rowMeans(flow[,d+c(f1,f2)], na.rm=T)
head(flow[,c("f1", "f2", "ossz")]) # az új oszlopok megtekintése
#>         f1       f2 ossz
#> 1 3.181818 3.555556 3.35
#> 2 4.181818 4.000000 4.10
#> 3 2.636364 2.666667 2.65
#> 4 3.636364 2.888889 3.30
#> 5 4.272727 4.222222 4.25
#> 6 3.727273 4.111111 3.90
```

A fenti kód több lépésben valósítja meg a kérdőív kiértékelését. Először rögzítjük az `f1` és `f2` alskálákhoz tartozó itemek sorszámát, amelyeket a `c()` függvény segítségével egy-egy vektorba gyűjtünk. Ezután megkeressük az a `d` eltolásai értéket, amely a `flow.1` oszlop előtti oszlop sorszámát jelenti. Ahogy látjuk, a `which()` függvény segítségével keressük meg a `flow.1` oszlop indexét, amelyet 1-gyel csökkentünk. Ezután a `rowMeans()` függvény segítségével kiszámítjuk az alskálák és a teljes skála átlagát. A `flow` adattábla új oszlopai a `f1`, `f2` és `ossz` változók, amelyek a kérdőív kiértékelését tartalmazzák. Az indexelés során a `d` eltolásai értéket hozzáadjuk az `f1` és `f2` vektorokhoz, hogy a megfelelő oszlopindexeket kapjuk. A `c(f1,f2)` vektorral pedig a teljes skála indexeit adjuk meg.

7.1.7.2. Numerikusból-faktor transzformáció

A változók átalakításának másik gyakori esete, amikor numerikus vektorból faktort hozunk létre, előre megadott osztáspontok segítségével. A 7.5. példa megoldásával átlépünk a numerikusból-faktor transzformációra.

Példa 7.5 (Korosztályok kialakítása)**.** A `flow.xlsx` összes 100 válaszadó életkorát tartalmazza. A 25 és 55 vágópontok segítségével alakítsunk ki 3 korosztályt (I., II. és III.)! Vizsgáljuk meg a létrejövő faktor szintjeinek gyakoriságát! Ha nem tartjuk megfelelőnek, akkor végezzünk összevonást!

A `cut()` függvény segítségével a numerikus vektorból faktort hozhatunk létre, amelynek szintjei a megadott osztáspontok alapján kerülnek meghatározásra. Az osztáspontokat a `breaks=` argumentumban adhatjuk meg, míg a szintek nevét a `labels=` argumentumban. A `right=` argumentum megadásával határozhatjuk meg, hogy a vágópontok zártak-e vagy nyitottak (a `right=TRUE` jelenti a jobbról zárt, balról nyitott intervallumot, míg a `right=FALSE` a balról zárt, jobbról nyitott intervallumot). A `cut()` függvény eredménye egy olyan faktor,

amelynek eggyel kevesebb szintje van, mint a `breaks=` argumentumban megadott vágópontok száma.

```
# numerikusból-faktor transzformáció
flow$korosztaly <- cut(flow$kor,
                       breaks=c(-Inf,25,55,Inf),
                       labels=c("I.", "II.", "III."),
                       right = T)
table(flow$korosztaly, useNA = "ifany") # a kategóriák gyakorisága
#>
#>   I.  II. III.
#>   60   39    1
```

Láthatjuk, ezek az osztáspontok rendkívül egyenetlen eloszlást jelentenek a most létrehozott `korosztaly` változó számára. Felmerül, hogy érintetlenül hagyva ezt a 3 szintű változót, hozzunk létre ebből kiindulva egy új, 2 szintű faktort, amely összevonja a "II." és "III." szinteket. Az új `korosztaly.2` faktor szintjei legyenek "fiatal" és "nem fiatal".

7.1.7.3. Faktorból-faktor transzformáció

Az előző fejezet végén vázolt feladat általánosítható: egy több szintű faktorból kiindulva, gyakran lehet szükségük egy kevesebb szintű faktorra. A `{car}` csomag `recode()` függvényét használhatjuk a feladat megoldására. Végezzük el tehát "II." és "III." faktorszintek összevonását úgy egy új `korosztaly.2` változóba, hogy a szintek legyenek a "fiatal" és a "nem fiatal" címkék.

```
# faktorból-faktor transzformáció
flow$korosztaly.2 <-
  car::recode(var = flow$korosztaly,
              recodes = '"I." = "fiatal"; c("II.", "III.")="nem fiatal"')
table(flow$korosztaly.2, useNA = "ifany") # a kategóriák gyakorisága
#>
#>     fiatal nem fiatal
#>         60         40
```

Látható, hogy a 2 kategóriás korosztály változó, már jóval kiegyensúlyozottabb eloszlást mutat. A célunkat a `recode()` függvény `recodes=` argumentumában megadott két átkódolási parancs segítségével értük el. A `recodes=` egy karakteres argumentum, amelyben az átkódoló parancsok pontosvesszővel elválasztva szerepelnek. Az első parancs a "fiatal" szint létrehozásához volt szükséges, míg a második parancs a "nem fiatal" szintet hozta létre, úgy, hogy a

`c()` segítségével olvasztottuk össze a két régi kategóriát (“II.” és “III.”) az új kategóriába (“nem fiatal”).

Ez a transzformáció már faktorból-faktor átalakítás, amelyre nézzünk egy másik példát is.

Példa 7.6 (Családi állapotok szintjeinek összevonása)**.** A `flow.xlsx` a válaszadók családi állapotát 5 szinttel méri. Ezeket a szinteket szeretnénk 2 szintre csökkenteni. Azt szeretnénk elérni, hogy a “Házas” és “Élettársi kapcsolatban él” szintek egy új szintet alkossanak, amelynek neve “Kapcsolatban él”. A másik új szint neve “Egyedülálló”, amely az “Elvált”, “Özvegy” és “Egyedülálló” régi szintekből áll össze. Az új faktort nevezzük `csaladi.allapot.2`-nek!

A 7.6. példa megoldásához először ellenőrizzük le a meglévő szintek neveit. Mivel faktorról van szó a `unique()` függvény helyett szerencsésebb a `levels()` függvényt használni, amely a faktor szintjeit adja vissza. A `levels()` függvény a faktortípusú változók szintjeinek kiírására szolgál, míg a `unique()` függvény bármilyen típusú vektor egyedi értékeit adja vissza.

```
# szintnevek kiírása
levels(flow$csaladi.allapot)
#> [1] "Házas"                    "Élettársi kapcsolatban él"
#> [3] "Elvált"                   "Özvegy"
#> [5] "Egyedülálló"
```

Az átalakítást most is a `recode()` függvény segítségével végezzük el.

```
# faktorból-faktor transzformáció
flow$csaladi.allapot.2 <-
  car::recode(var = flow$csaladi.allapot,
              recodes = 'c("Házas", "Kapcsolatban él") = "Kapcsolatban él";
                         else="Egyedül él"')
```

A `recode()` függvény `recodes=` argumentumában most is két átkódoló parancs található. Az első a szokásos `c(régi szintek)=új_szint` formátumú, a másik azonban az `else` kulcsszóval kezdődik, amely minden, korábban még nem használt szintről gondoskodik. Ezzel az elegáns megoldással elkerültük az “Egyedül él” új szinthez tartozó régi szintek megadását. Az `else` kulcsszó használata kódunk olvashatóságát jelentősen növeli.

7.1.8. Sorok elnevezése

Sokszor előfordul, hogy egy adattábla valamely változójának értékeivel szeretnénk a sorokat elnevezni, illetve fordítva, az adattábla sorneveit oszlopvektorban szeretnénk látni.

Példa 7.7 (Uniszex nevek). A `{fivethirtyeight}` csomag `unisex_names` adattáblája 919 sorban tartalmazza a leggyakoribb amerikai uniszex neveket. A `name` változóban található nevek alapján nevezzük el a sorokat! Az adattábla első 10 sorát jelenítsük meg! A további oszlopok adatai, `total`: a névvel élő amerikaiak száma, `male_share`: a névvel rendelkező férfiak aránya, `female_share`: a névvel rendelkező nők aránya, `gap`: a férfi és női arány közötti különbség.

Végezzük el az adatok beolvasását, és tekintsük meg az első 10 sort!

```
# adatok beolvasása
data(unisex_names, package = "fivethirtyeight")
unisex_names <- as.data.frame(unisex_names) # alakítsuk át adattáblává
head(unisex_names) # az első 6 sor megtekintése
#>     name    total male_share female_share        gap
#> 1  Casey 176544.3  0.5842866    0.4157134 0.16857313
#> 2  Riley 154860.7  0.5076391    0.4923609 0.01527814
#> 3 Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> 4 Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> 5  Avery 121797.4  0.3352131    0.6647869 0.32957385
#> 6  Jaime 109870.2  0.5617929    0.4382071 0.12358580
```

A `unisex_names` adattábla 4 változót tartalmaz, amelyből a `name` változóban található nevek alapján nevezzük el a sorokat. A `rownames()` függvény segítségével a `name` változó értékeit a sornevek helyére írhatjuk:

```
# sornév átírása
rownames(unisex_names) <- unisex_names$name
head(unisex_names)  # az első 6 sor megtekintése
#>          name    total male_share female_share        gap
#> Casey   Casey 176544.3  0.5842866    0.4157134 0.16857313
#> Riley   Riley 154860.7  0.5076391    0.4923609 0.01527814
#> Jessie Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> Jackie Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> Avery   Avery 121797.4  0.3352131    0.6647869 0.32957385
#> Jaime   Jaime 109870.2  0.5617929    0.4382071 0.12358580
```

Ilyen esetben a `name` változó feleslegessé válhat, így törölhetjük is.

```
unisex_names$name <- NULL  # a felesleges oszlop törlése
head(unisex_names)
#>           total male_share female_share        gap
#> Casey  176544.3  0.5842866    0.4157134 0.16857313
#> Riley  154860.7  0.5076391    0.4923609 0.01527814
#> Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> Avery  121797.4  0.3352131    0.6647869 0.32957385
#> Jaime  109870.2  0.5617929    0.4382071 0.12358580
```

Amennyiben a sorneveket vissza szeretnénk írni az adattáblába, új oszlopként hozzáadhatjuk a sorneveket. Érdemes az oszlopok sorrendjét az eredeti adattábla sorrendjéhez igazítani, továbbá az eredeti, standard sorneveket is újra megadhatjuk.

```
# sornevek visszaírása oszlopvektorként
unisex_names$name <- rownames(unisex_names)
# oszlopok sorrendjének megváltoztatása
unisex_names <- unisex_names[, c(5, 1:4)]
# átírjuk az oszlopneveket
rownames(unisex_names) <- seq_len(nrow(unisex_names))
head(unisex_names)  # az első 6 sor megtekintése
#>     name    total male_share female_share        gap
#> 1  Casey 176544.3  0.5842866    0.4157134 0.16857313
#> 2  Riley 154860.7  0.5076391    0.4923609 0.01527814
#> 3 Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> 4 Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> 5  Avery 121797.4  0.3352131    0.6647869 0.32957385
#> 6  Jaime 109870.2  0.5617929    0.4382071 0.12358580
```

7.1.9. Sorok rendezése

Az adattáblák rendezéséről már volt szó a 5.3.7.5. fejezetben, láttuk, hogy az `order()` függvény adattáblák és vektorok (5.3.3.18. fejezet) rendezésére is kiválóan alkalmas. Most további példákat mutatunk be a `flow` adattábla rendezésével kapcsolatban.

Példa 7.8 (A legidősebb 5 személy adata)**.** A `flow.xlsx` adattáblájában található 100 válaszadó életkorát a `kor` változó tartalmazza. Jelenítsük meg a legidősebb 5 válaszadó adatait! Azonos életkor esetén a skálapontszám szerinti növekvő rendezést vegyük figyelembe.

A `flow$kor` változóban található életkorokból indulunk ki, és az `order()` függvény eredményét a sorkoordináta helyére írjuk.

```
# a legidősebb 5 válaszadó adata: kor, nem és skálapontszám
flow[order(flow$kor, decreasing = T)[1:5],c("kor", "nem", "ossz")]
#>    kor   nem ossz
#> 52  69    nő 3.65
#> 7   53    nő 3.65
#> 42  53 férfi 3.65
#> 74  53 férfi 3.70
#> 43  52    nő 3.95
```

A fenti kód a `flow$kor` változóban található életkorok alapján rendezi az adattáblát, de az `order()` függvény `decreasing = T` argumentuma miatt csökkenő sorrendben. Az `[1:5]` indexek az első 5 `order()` által szolgáltatott értéket veszi ki, amely a csökkenő rendezés miatt a legidősebb 5 válaszadót jelenti.

Rendezésnél egynél több változót is figyelembe vehetünk, ekkor az `order()` függvényben több változónevet kell felsorolnunk vesszővel elválasztva, és azok kerülnek felhasználásra, ha a rendezés során az addigi változók értékei egyenlőek. Például az előző életkorra vonatkozó rendezést egészítsük ki a skálapontszám alapján történő rendezéssel, hiszen vannak azonos életkorú válaszadók, már az 5 legidősebb válaszadó esetén is (az 53 év háromszor szerepel).

```
# a legidősebb 5 válaszadó adata, plusz a skálapontszám figyelembe vétele
flow[order(flow$kor, flow$ossz, decreasing = T)[1:5],c("kor", "nem", "ossz")]
#>    kor   nem ossz
#> 52  69    nő 3.65
#> 74  53 férfi 3.70
#> 7   53    nő 3.65
#> 42  53 férfi 3.65
#> 76  52    nő 4.50
```

Több rendezési szempont esetén eltérhet az egyes szempontok rendezési iránya. Ilyenkor egy mínusz előjellel módosíthatjuk az `order()` függvény által meghatározott rendezési irányt. A példánkban a csökkenő rendezés a kiinduló pont a `decsreasing = T` miatt, de a `-flow$ossz` megadásával a skálapontszámok szerinti növekvő rendezést állíthatunk be.

```
# a legidősebb 5 válaszadó adata, plusz a skálapontszám növekvőbe
flow[order(flow$kor, -flow$ossz, decreasing = T)[1:5],c("kor", "nem", "ossz")]
#>    kor   nem ossz
```

```
#> 52  69    nő 3.65
#> 7   53    nő 3.65
#> 42  53 férfi 3.65
#> 74  53 férfi 3.70
#> 75  52 férfi 3.25
```

7.1.10. Adattábla szűrése

Sokszor előfordul, hogy az adattábla sorait egy vagy több változó (oszlop) értéke szerint szeretnénk leválogatni. Ilyenkor az adattábla indexelése során a sorkoordináta helyén logikai kifejezést szerepeltetünk. Az adattábla szűrését a 5.3.7.4. fejezetben már érintettük, most tovább mélyítjük ismereteinket.

Példa 7.9 (Férfiak az adott korosztályból)**.** A `flow.xlsx` adattábla tartalmát válogassuk le egy `flow.szukitett` nevű új adattáblába, amely csak azokat a férfi válaszadókat tartalmazza, akik 20 és 25 év közöttiek!

```
# adattábla szűrése: férfiak, 20 és 25 év között
flow.szukitett <- flow[flow$nem %in% "férfi" & flow$kor>=20 & flow$kor<=25,]
dim(flow.szukitett) # az új adattábla mérete
#> [1] 16 31
```

Látható, hogy sorkoordinátába összetett logikai kifejezést írtunk, amely a `flow$nem` változóban a "férfi" értéket keresi, és a `flow$kor` változóban a 20 és 25 közötti értékeket. A `&` operátorral összekapcsoltuk a két logikai kifejezést, amelynek eredménye egy logikai vektor. Azok a sorok kerülnek leválogatásra, amely pozícióban a `TRUE` érték szerepel a logikai vektorban.

A fenti szűrést a potenciálisan előforduló `NA` értékek picit megbonyolítják. Vegyük a következő egyszerű esetet:

```
# NA értéket is tartalmazó adattábla
adat <- "  modszer   pontszam
                A         42
                NA        67
                B         NA
                B         32  "
# adatmátrix beolvasása
df <- read.table(textConnection(adat), header=T, sep="")
```

Mindkét oszlop tartalmaz egy-egy hiányzó értéket. Így vizsgáljuk meg a fenti szűrés elemeit.

```
# egyenlőség vizsgálat, kerülendő
df[df$modszer == "A", ]
#>    modszer pontszam
#> 1        A       42
#> NA    <NA>       NA
```

A fenti lekérdezés egy csupa `NA` sort is eredményez, ezért az `==` operátorral történő fenti szűrés kerülendő.

```
# tartalmazás vizsgálat, ha nem kell az NA, akkor jó lehet
df[df$modszer %in% "A", ]
#>   modszer pontszam
#> 1       A       42
```

Látható, hogy az `%in%` operátor használata kezeli az `NA` értékeket, de a hasonlítás során `FALSE` lesz a hasonlítás eredménye, így az ilyen sorok kiesnek az outputból. Ha ez a célunk, azaz a `NA` értékek a `modszer` oszlopban nem szükségesek a lekérdezés eredményébe, akkor megtaláltuk a pontos lekérdezésünket.

```
# ha nem kell az NA, akkor jó lehet
base::subset(df, modszer == "A")
#>   modszer pontszam
#> 1       A       42
```

A beépített `subset()` függvény is jól kezeli a `NA` értékeket, pontosan úgy, mint az `%in%` operátor. Ennek megfelelően ugyanabban a helyzetben használhatjuk. A `subset()` függvény első paramétere a szűrendő adatmátrix, a második paraméter pedig a szűrési feltétel. A `transform()` függvényhez hasonlóan a `subset()` függvény is egy új adattáblát ad vissza, amely tartalmazza a szűrési feltételnek megfelelő sorokat. A `subset()` függvény használata során a `df$modszer` helyett elegendő a `modszer` változót megadni, hiszen a `subset()` függvény automatikusan figyelembe veszi az adattábla oszlopait, így jelentősen hozzájárul a kód olvashatóságához.

Abban az esetben, ha a `modszer` oszlopban az `NA` értéket tartalmazó sorokat is látni szeretnénk az outputban, akkor ki kell egészítenünk a lekérdezést egy `is.na()` kifejezéssel, amit az `|` (VAGY) operátorral kapcsolunk össze a `modszer` oszlop szűrésével.

```
# ha kell az NA, akkor jó lehet
df[df$modszer == "A" | is.na(df$modszer), ]
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
df[df$modszer %in% "A" | is.na(df$modszer), ]
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
base::subset(df, modszer == "A" | is.na(modszer))
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
```

A fenti outputokból kiolvasható, hogy mindhárom esetben megjelentek azok a sorok, amelyekben a `modszer` oszlopban `NA` érték szerepel.

Numerikus értékek esetében is hasonló összehasonlítást végezhetünk, szinte megegyező eredménnyel.

```
# nagyobb vizsgálat, kerülendő
df[df$pontszam > 50, ]
#>    modszer pontszam
#> 2     <NA>       67
#> NA    <NA>       NA
```

```
# ha nem kell az NA, akkor jó lehet
base::subset(df, pontszam > 50)
#>   modszer pontszam
#> 2    <NA>       67
```

```
# ha kell az NA, akkor jó lehet
df[df$pontszam > 50 | is.na(df$pontszam), ]
#>   modszer pontszam
#> 2    <NA>       67
#> 3       B       NA
base::subset(df, pontszam > 50 | is.na(pontszam))
#>   modszer pontszam
#> 2    <NA>       67
#> 3       B       NA
```

A fentiek alapján készíthetünk egy biztonságosabb szűrést, amely az `NA` értékeket nem zárja ki eleve a lekérdezésből és csupa `NA`-s sorokat sem szúr be. Ezt az `| is.na()` kifejezés hozzáadásával végezzük el. Az `NA` értékek figyelembe vételével a fenti példát így írhatjuk át:

```
# adattábla szűrése: férfiak, 20 és 25 év között
flow.szukitett <- flow[(flow$nem %in% "férfi" | is.na(flow$nem)) &
                       (flow$kor>=20 | is.na(flow$kor)) &
                       (flow$kor<=25 | is.na(flow$kor)),]
dim(flow.szukitett) # az új adattábla mérete
#> [1] 16 31
```

7.1.11. Hiányzó értékek kezelése

Láttuk, hogy a hiányzó értékek létezése a szűrés során mennyi problémát okozhat. A 5.3.3.10. fejezetben már érintettük a hiányzó értékek kezelését, most tekintsük át ezek részletesebben!

Példa 7.10 (Hiányzó értékek felderítése)**.** A `{MASS}` csomag `survey` adattáblája 237 válaszadó adatait tartalmazza. A `survey` adattábla 7 változót tartalmaz. Vizsgáljuk meg oszloponként a hiányzó értékek számát! Hány teljes (hiányzó értéket nem tartalmazó) sor van az adattáblában?

A hiányzó értékek felderítésének leggyakoribb módja a `is.na()` függvény használata, amely olyan logikai vektort ad vissza, amelyben a hiányzó értékek helyén `TRUE`, míg a nem hiányzó értékek helyén `FALSE` szerepel. Ha a `sum()` függvényt használjuk az `is.na()` függvény eredményére, akkor a hiányzó értékek számát kapjuk meg.

```
# a survey adattábla betöltése a MASS csomagból
data(survey, package = "MASS")
sum(is.na(survey$Sex))
#> [1] 1
```

Látható, hogy a nem (`Sex`) változóban mindössze 1 hiányzó érték van.

Az összes oszlopra is meghatározhatjuk a hiányzó értékek számát, ehhez a `sapply()` függvényt használhatjuk, amely az összes oszlopra alkalmazza a megadott függvényt. A `sapply()` függvény első paramétere az adattábla, a második paramétere pedig a függvény, amelyet alkalmazni szeretnénk. Az alábbi kódban rövidített névtelen függvényt (`\(x) sum(is.na(x))`)

használunk a hiányzó értékek számának meghatározására. A `sapply()` függvény eredménye egy vektor, amelyben az oszlopok nevei szerepelnek, és a hiányzó értékek száma van megadva.

```
# a hiányzó értékek számának meghatározása
sapply(survey, \(x) sum(is.na(x)))
#>    Sex Wr.Hnd NW.Hnd  W.Hnd   Fold  Pulse   Clap   Exer  Smoke Height
#>      1      1      1      1      0     45      1      0      1     28
#>    M.I    Age
#>     28      0
```

Meghatározhatjuk azoknak a soroknak a számát is, amelyek teljesek, azaz hiányzó értéket egyáltalán nem tartalmaznak. Ezt a `complete.cases()` függvény segítségével végezhetjük el, amely egy logikai vektort ad vissza, amelyben a teljes sorok helyén `TRUE`, míg a hiányzó értékeket tartalmazó sorok helyén `FALSE` szerepel.

```
# a teljes sorok számának meghatározása
sum(complete.cases(survey))
#> [1] 168
# legalább 1 hiányzó értéket tartalmazó sorok számának meghatározása
sum(!complete.cases(survey))
#> [1] 69
```

A teljes sorokat tartalmazó adattáblát a `na.omit()` függvény segítségével kaphatjuk meg, amely az összes hiányzó értéket tartalmazó sort eltávolítja az adattáblából. A `na.omit()` függvény eredménye egy új adattábla, amelyben csak a teljes sorok szerepelnek.

```
# a hiányzó értékeket tartalmazó sorok eltávolítása
survey.teljes <- na.omit(survey)
# a csak hiányzó értékeket tartalmazó sorok adatbázisa
survey.hianyzok <- survey[!complete.cases(survey),]
```

Gyakori adatmanipulációs lépés a hiányzó értékek helyettesítése egy érvényes értékkel. Helyettesítsük például a hiányzó testmagasság (`Height`) értékeket az átlagukkal. A `mean()` függvény `na.rm = T` argumentuma biztosítja, hogy a hiányzó értékek ne zavarják meg a számítást.

```
# hiányzó testmagasságok helyettesítése az átlaggal
survey$Height[is.na(survey$Height)] <- mean(survey$Height, na.rm = T)
sum(is.na(survey$Height)) # már nincsenek hiányzó értékek
#> [1] 0
```

Összefoglalás

Az *Alap R* rendszerben az adatkezelés alapját az adattábla jelenti, amely kettős természetű: egyszerre viselkedik mátrixként, azaz sor–oszlop struktúraként, és listaként, ahol minden oszlop egy-egy azonos hosszúságú vektor. Az adatelemzés első lépése az adatok szerkezetének áttekintése. Ehhez az `str()`, `dim()`, `nrow()`, `ncol()`, `names()` és `summary()` függvények nyújtanak segítséget. Az oszlopok elérésére több lehetőség is kínálkozik: a legelterjedtebb a `df$valtozo`, de használható a `df[, "valtozo"]` vagy a `df[["valtozo"]]` forma is. A változók típusának ellenőrzése és átalakítása kiemelten fontos feladat. A nominális és ordinális változókat faktor típusban célszerű tárolni. A `factor()` függvénnyel karakteres vagy numerikus vektorból faktort hozhatunk létre, míg az `ordered()` függvény rendezett faktort készít. A változók értékeinek módosítására, azaz transzformációra sokféle példa létezik. A legegyszerűbb eset, amikor numerikus értékekből új numerikus változót képzünk, például mértékegység-átváltással. A `transform()` függvény egyszerűsített módot kínál ilyen típusú módosításokra. A `cut()` függvény segítségével numerikus változót bontunk kategóriákra, vagyis faktort hozunk létre meghatározott vágópontok alapján. Az egyes kategóriák összevonása vagy átkódolása a `{car}` csomag `recode()` függvényével történik. Az adatelemzés során a sorok rendezése és szűrése is alapvető feladat. Rendezéshez az `order()` függvényt használjuk, akár több szempont figyelembevételével is. A szűrés logikai kifejezésekkel történik a sorkoordináta helyén. Például `df[df$kor > 25, ]` lekérdezi azokat, akiknek életkora 25 év felett van. A szűrés során azonban figyelni kell a hiányzó értékekre (NA), mivel ezek a logikai feltételekben `NA` értékként viselkednek, így a `==` helyett érdemes az `%in%` operátort használni. A beépített `subset()` függvény biztonságosan kezeli ezeket az eseteket. A hiányzó értékek felderítése az `is.na()` függvénnyel történik, amely logikai értékeket ad vissza.

Feladatok

1. A `survey` adattáblában a `Height` változó hiányzó értékeit helyettesítsük a változó mediánjával!
2. A `survey` adattáblában számoljuk a BMI értéket, majd alakítsuk át kategorikus változóvá!
3. A fejezetben névtelen függvény megadására használtuk a `\(x)` szintaxist. Járjunk utána a `\(x)` és a `function(x)` közötti különbségnek! Melyik esetben használható a `\(x)` szintaxis?
4. A `HSAUR3::Forbes2000` adattáblája 2000 vállalat adatát tartalmazza! Határozzuk meg a magyar cégek nevét és helyezését (`country` oszlop alapján)! Írassuk ki a képernyőre a 10 legnagyobb piaci értékkel (`marketvalue` oszlop) rendelkező cég nevét és piaci értékét! Határozzuk meg a legkisebb profittal (`profits` oszlop) rendelkező 5 cég minden adatát! Határozzuk meg a legnagyobb profittal (`profits`

oszlop) rendelkező 10 amerikai vagy japán cég nevét, országát és profitját!

7.2. Adatkezelés *Tidyverse R*-ben 🤔

Miről lesz szó? Ebben a fejezetben

- megismerjük az adatelőkészítő lépések végrehajtását *Tidyverse R*-ben,
- lényegében az *Alap R*-ben már megismert adatkezelési lépéseket hajtjuk végre újra, azaz
- megtanuljuk az információkérés parancsait,
- az oszlopok elnevezését, típusának és sorrendjének megváltoztatását,
- a sorok manipulációját, rendezését és szűrését, valamint
- a hiányzó adatok kezelését.

A *Tidyverse R* csomagcsalád a `{dplyr}` és `{tidyr}` csomagjában találhatóak azok a függvények, amelyek az adatok kezelésére szolgálnak. Az előző fejezetben az adatelőkészítés lépéseit az *Alap R* rendszerben hajtottuk végre, most pedig a már megismert lépéseket a *Tidyverse R* csomagcsalád segítségével valósítjuk meg. Jelen fejezet olvasása során minden bizonnyal észre fogjuk venni, hogy a *Tidyverse R* csomagcsalád használata során a parancsaink sokkal rövidebbek és könnyebben olvashatóak, az *Alap R*-beli változatokhoz képest. Az adatszemléletű, modern *Tidyverse R* megközelítést a pipe operátor (`|>`) – néha túlzott – használatával erősítjük.

Jelen fejezetben ugyanazokat a példákat fogjuk megoldani, mint az előző, *Alap R* rendszerre épülő 7.1. fejezetben, de a duplikációk elkerülése miatt a példákra csak hivatkozunk, és a kísérőszöveget is minimálisra csökkentjük. A *Tidyverse R*-hez kapcsolódó újdonságokat természetesen részletesen bemutatjuk, de megemlítjük, hogy támaszkodunk a 5.4.4. fejezetben megismert pipe operátorra (`|>`) és alapvető *Tidyverse R* függvényekre.

7.2.1. Információ gyűjtése

A 9.1. példában a `flow.xlsx` állomány megismerését tűztük ki célul. A fájl beolvasásához most a `{readxl}` csomag `read_xlsx()` függvényét használjuk. A fájl elérési útját a `path` argumentumban adjuk meg. Az adattábla megismerésére a `class()` és `dplyr::glimpse()` függvényeket használjuk.

```
library(tidyverse)
flow <- readxl::read_xlsx(path = "adat/flow.xlsx") # beolvasás
flow |> class() # az adattábla típusa
#> [1] "tbl_df"     "tbl"        "data.frame"
flow |> glimpse() # sorok és oszlopok száma és teljes szerkezet
#> Rows: 100
#> Columns: 25
#> $ alkatoi.tev     <chr> "Igen", "Igen", "Nem", "Nem", "Igen", "Igen", ~
#> $ kor             <dbl> 26, 20, 22, 21, 21, 25, 53, 21, 22, 21, 21, 50~
#> $ nem             <chr> "Férfi", "Nő", "Nő", "Férfi", "Nő", "Nő", "Nő"~
#> $ csaladi.allapot <chr> "Egyedülálló", "Egyedülálló", "Élettársi kapcs~
#> $ isk.vegz        <chr> "Egyetem", "Gimnázium", "Gimnázium", "Gimnáziu~
#> $ flow.1          <dbl> 2, 4, 4, 3, 5, 5, 5, 4, 5, 5, 4, 5, 5, 3, 5, 4~
#> $ flow.2          <dbl> 4, 4, 4, 4, 5, 5, 5, 4, 4, 5, 4, 5, 5, 3, 5, 3~
#> $ flow.3          <dbl> 5, 5, 3, 3, 5, 5, 3, 4, 4, 5, 5, 5, 5, 4, 5, 5~
#> $ flow.4          <dbl> 5, 5, 4, 4, 2, 5, 3, 4, 5, 5, 5, 5, 5, 4, 5, 5~
#> $ flow.5          <dbl> 5, 5, 5, 3, 3, 5, 5, 5, 5, 5, 5, 4, 1, 5, 5, 5~
#> $ flow.6          <dbl> 1, 4, 2, 3, 4, 1, 3, 2, 2, 4, 2, 3, 5, 4, 5, 3~
#> $ flow.7          <dbl> 2, 4, 1, 4, 4, 3, 5, 3, 5, 5, 3, 4, 5, 4, 4, 4~
#> $ flow.8          <dbl> 4, 5, 4, 4, 4, 5, 4, 4, 5, 5, 4, 3, 5, 4, 5, 3~
#> $ flow.9          <dbl> 5, 5, 3, 4, 5, 5, 3, 4, 5, 5, 5, 4, 5, 5, 5, 5~
#> $ flow.10         <dbl> 4, 5, 4, 3, 5, 5, 5, 4, 4, 5, 5, 3, 5, 5, 5, 4~
#> $ flow.11         <dbl> 4, 5, 2, 3, 5, 5, 3, 3, 5, 5, 5, 4, 5, 5, 5, 5~
#> $ flow.12         <dbl> 4, 4, 1, 3, 4, 5, 5, 4, 5, 5, 4, 5, 4, 4, 5, 3~
#> $ flow.13         <dbl> 5, 4, 2, 4, 5, 3, 5, 4, 5, 5, 4, 5, 4, 5, 5, 3~
#> $ flow.14         <dbl> 3, 4, 2, 4, 5, 5, 3, 3, 5, 5, 5, 5, 4, 5, 5, 3~
#> $ flow.15         <dbl> 3, 5, 3, 4, 4, 5, 5, 4, 5, 5, 5, 5, 5, 5, 4, 4~
#> $ flow.16         <dbl> 3, 3, 3, 3, 4, 2, 4, 2, 3, 5, 1, 5, 5, 5, 5, 2~
#> $ flow.17         <dbl> 4, 5, 3, 4, 3, 2, 4, 4, 5, 5, 4, 5, 5, 5, 5, 3~
#> $ flow.18         <dbl> 5, 5, 4, 4, 2, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5~
#> $ flow.19         <dbl> 4, 5, 4, 2, 5, 5, 5, 2, 5, 5, 3, 3, 5, 4, 5, 1~
#> $ flow.20         <dbl> 3, 4, 1, 2, 4, 5, 1, 2, 3, 5, 2, 1, 4, 4, 3, 1~
```

A fenti sorok után világossá válik, hogy egy 100 sort és 25 oszlopot tartalmazó *tibble* (*data frame* továbbfejlesztett változata) áll rendelkezésre a `flow` objektumban. Az oszlopnevek a `names()` és a `colnames()` függvényekkel most is megismerhetők.

```
flow |> names()    |> _[1:4] # az első 4 oszlop neve
#> [1] "alkatoi.tev"     "kor"             "nem"
#> [4] "csaladi.allapot"
```

```
flow |> colnames() |> _[5:8] # a következő 4 oszlop neve
#> [1] "isk.vegz" "flow.1"   "flow.2"   "flow.3"
```

Az egyes oszlopok jobb megismerése a klasszikus `class()` és `unique()` függvényekkel történik, de gyakorisági táblázat *Tidyverse R*-ben a `dplyr::count()` függvény segítségével készíthető el. A `count()` függvény első argumentuma az adattábla neve, a második argumentum pedig a változó neve, amelynek gyakorisági táblázatát szeretnénk elkészíteni. A `sort` argumentum megadásával a leggyakoribb értékek kerülnek az első helyekre. A pipe operátor használata miatt vegyük figyelembe, hogy a `count()` első argumentuma, az adattábla neve, valójában a `|>` operátor bal oldalán található, és nem a `count()` argumentumában.

```
flow$alkatoi.tev |>
    class()  # az 'alkotoi.tev' változó típusa
#> [1] "character"
flow$alkatoi.tev |>
    unique()  # egyedi értékei
#> [1] "Igen" "Nem"
flow |>
    count(alkatoi.tev)  # gyakorisági táblázata
#> # A tibble: 2 x 2
#>   alkatoi.tev     n
#>   <chr>       <int>
#> 1 Igen           39
#> 2 Nem            61
flow$kor |>
    class()  # a kor változó típusa
#> [1] "numeric"
flow |>
    count(kor, sort = T)  # gyakorisági táblázata
#> # A tibble: 27 x 2
#>     kor     n
#>   <dbl> <int>
#> 1    21    22
#> 2    22    18
#> 3    26     5
#> 4    52     5
#> 5    20     4
#> 6    23     4
#> # i 21 more rows
```

Az adatbázis tetszőleges résztábláját a `slice()` és `select()` függvények segítségével kérhetjük le. A `slice()` függvény a sorok kiválasztására szolgál pozíciójuk alapján, míg a `select()` függvény az oszlopok kiválasztására, pozíció vagy név alapján. A `|>` operátorral a két függvény eredményét összekapcsolhatjuk.

```
# az adattábla egy résztáblája
flow |>
    slice(c(1:2, 50:51)) |>
    select(1:5)
#> # A tibble: 4 x 5
#>   alkatoi.tev   kor nem   csaladi.allapot isk.vegz
#>   <chr>       <dbl> <chr> <chr>           <chr>
#> 1 Igen           26 Férfi Egyedülálló     Egyetem
#> 2 Igen           20 Nő    Egyedülálló     Gimnázium
#> 3 Nem            22 Nő    Egyedülálló     Gimnázium
#> 4 Igen           39 Nő    Házas           Egyetem
```

Ismert, hogy a *tibble* kényelmesebb megjelenést nyújt a konzolban, mint az adattábla típus, hiszen megjeleníti az oszlopok típusát is. Egyszerűen írassuk ki a `flow` adattáblát a konzolra, és figyeljük meg a megjelenését!

```
# tibble megjelenítése
flow
#> # A tibble: 100 x 25
#>   alkatoi.tev   kor nem   csaladi.allapot  isk.vegz flow.1 flow.2 flow.3
#>   <chr>       <dbl> <chr> <chr>            <chr>     <dbl>  <dbl>  <dbl>
#> 1 Igen           26 Férfi Egyedülálló      Egyetem       2      4      5
#> 2 Igen           20 Nő    Egyedülálló      Gimnázi~      4      4      5
#> 3 Nem            22 Nő    Élettársi kapcs~ Gimnázi~      4      4      3
#> 4 Nem            21 Férfi Egyedülálló      Gimnázi~      3      4      3
#> 5 Igen           21 Nő    Egyedülálló      Gimnázi~      5      5      5
#> 6 Igen           25 Nő    Elvált           Egyetem       5      5      5
#> # i 94 more rows
#> # i 17 more variables: flow.4 <dbl>, flow.5 <dbl>, flow.6 <dbl>,
#> #   flow.7 <dbl>, flow.8 <dbl>, flow.9 <dbl>, flow.10 <dbl>,
#> #   flow.11 <dbl>, flow.12 <dbl>, flow.13 <dbl>, flow.14 <dbl>,
#> #   flow.15 <dbl>, flow.16 <dbl>, flow.17 <dbl>, flow.18 <dbl>,
#> #   flow.19 <dbl>, flow.20 <dbl>
```

A megjelenő sorok és oszlopok száma is csökkent, a könnyebb áttekinthetőség kedvéért. Természetesen ezt mi is beállíthatjuk a `options()` függvény `tibble.print_max` és `tibble.print_min`

argumentumaival. A `tibble.print_max` argumentum megadja a maximális sorok számát, amelyet a konzolra írhatunk, míg a `tibble.print_min` argumentum megadja a minimális sorok számát. A kiírás során közvetlenül is megadhatjuk a `n` argumentumban a kívánt sorok számát, amelyet ki szeretnénk írni.

```
print(flow, n=3) # 3 sor kiírása
#> # A tibble: 100 x 25
#>   alkatoi.tev   kor nem   csaladi.allapot  isk.vegz flow.1 flow.2 flow.3
#>   <chr>       <dbl> <chr> <chr>            <chr>     <dbl>  <dbl>  <dbl>
#> 1 Igen           26 Férfi Egyedülálló      Egyetem       2      4      5
#> 2 Igen           20 Nő    Egyedülálló      Gimnázi~      4      4      5
#> 3 Nem            22 Nő    Élettársi kapcs~ Gimnázi~      4      4      3
#> # i 97 more rows
#> # i 17 more variables: flow.4 <dbl>, flow.5 <dbl>, flow.6 <dbl>,
#> #   flow.7 <dbl>, flow.8 <dbl>, flow.9 <dbl>, flow.10 <dbl>,
#> #   flow.11 <dbl>, flow.12 <dbl>, flow.13 <dbl>, flow.14 <dbl>,
#> #   flow.15 <dbl>, flow.16 <dbl>, flow.17 <dbl>, flow.18 <dbl>,
#> #   flow.19 <dbl>, flow.20 <dbl>
```

7.2.2. Oszlopnevek módosítása

Az oszlopnevek *Tidyverse R* átnevezését a 7.2. példa alapján tekintjük át. A `metal_bandak.xlsx` fájl beolvasásához a `readxl::read_xlsx()` függvényt használjuk, majd a `names()` segítségével meghatározzuk az oszlopneveket.

```
bandak <- readxl::read_xlsx(path = "adat/metal_bandak.xlsx")
bandak
#> # A tibble: 5 x 5
#>   banda       fan formed split origin
#>   <chr>     <dbl> <chr>  <chr> <chr>
#> 1 Kiuas       106 2000   2013  Finnország
#> 2 Accept      681 1968   <NA>  Németország
#> 3 Metallica  4122 1981   <NA>  USA
#> 4 Zonata       23 1998   2003  Svédország
#> 5 Therion    1266 1987   <NA>  Svédország
bandak |>
    names()  # az összes oszlop neve
#> [1] "banda"  "fan"    "formed" "split"  "origin"
```

A kívánt oszlopok átnevezésére a `dplyr::rename()` függvényt használjuk. Az első argumentum az adattábla neve (ami most is a pipe operátor előtt jelenik meg), a második argumentum pedig az új oszlopnevek megadása. Az új oszlopnevek megadásakor a régi oszlopneveket is meg kell adni, amelyeket az új nevekkel párosítunk. A megadás módja: `uj_oszlopnev = regi_oszlopnev`. Az új oszlopnevek megadásakor a régi oszlopneveket nem kell idézőjelbe tenni, sőt sorszámmal is hivatkozhatunk a régi oszlopnevekre. Az alábbi példában a `bandak` adattábla 2-5. oszlopait nevezzük át, és végül az 1. oszlopot is.

```
# oszlopok átnevezése: új oszlopnév = régi oszlopnév formában
bandak <- bandak |>
  rename(rajongo = 2,
         alakulas = 3,
         felbomlas = 4,
         orszag = 5,
         nev = 1)
bandak
#> # A tibble: 5 x 5
#>   nev       rajongo alakulas felbomlas orszag
#>   <chr>       <dbl> <chr>    <chr>     <chr>
#> 1 Kiuas         106 2000     2013      Finnország
#> 2 Accept        681 1968     <NA>      Németország
#> 3 Metallica    4122 1981     <NA>      USA
#> 4 Zonata         23 1998     2003      Svédország
#> 5 Therion      1266 1987     <NA>      Svédország
```

7.2.3. Oszlop elérése

Tidyverse R-ben az *Alap R*-ben megszokott módon indexelhetjük a *tibble* adatszerkezetünket: a `[` vagy `[[` operátort használjuk. Mivel a *tibble* egy továbbfejlesztett adattábla, így örökölték a kétdimenziós mátrix és az egydimenziós lista adatszerkezet indexelési lehetőségeit, azaz az oszlopokra négyféle módon hivatkozhatunk:

```
tibble[,oszlopindex]  # hivatkozás egy vagy több oszlopra
tibble[oszlopindex]   # hivatkozás egy vagy több oszlopra
tibble[[oszlopnév]]   # hivatkozás egyetlen oszlopra
tibble$oszlopnév      # hivatkozás egyetlen oszlopra
```

A lenti kód tehát egyetlen karakterben sem tér el a *Alap R*-beli indexeléstől. Ez adja a *tibble* erejét, nem kell lemondanunk a *Alap R*-ben megtanult lehetőségekről, a *Tidyverse R* csomagcsalád csak újabb lehetőségeket ad hozzá.

```
mean(flow$kor, na.rm = T)  # kor átlaga
#> [1] 29.26
summary(flow[c("kor", "flow.1")])  # kor és flow.1 leíró statisztikai adatai
#>       kor            flow.1
#>  Min.   :17.00   Min.   :1.00
#>  1st Qu.:21.00   1st Qu.:3.00
#>  Median :23.00   Median :4.00
#>  Mean   :29.26   Mean   :4.01
#>  3rd Qu.:33.00   3rd Qu.:5.00
#>  Max.   :69.00   Max.   :5.00
```

```
# 'flow' szót tartalmazó oszlopok száma
ncol(flow[grepl(pattern = "flow", names(flow))])
#> [1] 20
```

A *tibble* egyik határozott előnye, hogy az indexelés során nem történhet dimenzióvesztés. Ebben némileg eltér az *Alap R*-tól, ahol a `[` operátor használatakor a kiválasztott oszlopok számától függően dimenzióvesztés történhet. A *tibble* esetében ez nem fordulhat elő, így a kiválasztott oszlopok számától függetlenül mindig megmarad a kétdimenziós struktúra. Az alábbi példákban látható, hogy a `flow` adattábla első három sorának és a `kor` oszlopának kiválasztásakor a dimenzióvesztés nem történik meg.

```
flow[1:3, c("kor", "nem")]  # két oszlop, nincs dimenzióvesztés
#> # A tibble: 3 x 2
#>     kor nem
#>   <dbl> <chr>
#> 1    26 Férfi
#> 2    20 Nő
#> 3    22 Nő
```

```
flow[1:3, "kor"]  # egy oszlop, nincs dimenzióvesztés
#> # A tibble: 3 x 1
#>     kor
#>   <dbl>
#> 1    26
#> 2    20
#> 3    22
```

7.2.4. Oszlopok sorrendje

Oszlopok sorrendjét a `dplyr::select()` függvény segítségével határozzuk meg. Egyszerűen sorszámokkal hivatkozhatunk az oszlopokra. Az alábbi példában a `flow` adattábla 1., `alkotoi.tev` oszlopa átkerül a 3. oszlopba.

```
# a korábbi 1. oszlop (alkotoi.tev) átkerül a 3. oszlopba
flow <- flow |>
    select(2, 3, 1, 4:25)
flow[1:2, 1:3]
#> # A tibble: 2 x 3
#>     kor nem   alkatoi.tev
#>   <dbl> <chr> <chr>
#> 1    26 Férfi Igen
#> 2    20 Nő    Igen
```

Természetesen oszlopsorszámok helyett változóneveket is használhatunk. Láthatjuk a következő példában, hogy az oszlopneveket nem kell idézőjelben megadni, de ha szeretnénk, akkor megtehetjük. Továbbá a mínusz előjel (-) használatával az adott oszlopok kihagyható a hivatkozásból, az összes többi oszlop megmarad. A példában a `rajongo` oszlopot az adatmátrix végére helyezzük át.

```
bandak  # emlékeztetőül a bandák adattábla
#> # A tibble: 5 x 5
#>   nev       rajongo alakulas felbomlas orszag
#>   <chr>       <dbl> <chr>    <chr>     <chr>
#> 1 Kiuas         106 2000     2013      Finnország
#> 2 Accept        681 1968     <NA>      Németország
#> 3 Metallica    4122 1981     <NA>      USA
#> 4 Zonata         23 1998     2003      Svédország
#> 5 Therion      1266 1987     <NA>      Svédország
```

```
# a rajongo oszlop a végére kerül
bandak <- bandak |>
    select(-rajongo, rajongo)
bandak
#> # A tibble: 5 x 5
#>   nev       alakulas felbomlas orszag      rajongo
#>   <chr>     <chr>    <chr>     <chr>         <dbl>
#> 1 Kiuas     2000     2013      Finnország      106
```

```
#> 2 Accept    1968     <NA>      Németország     681
#> 3 Metallica 1981     <NA>      USA            4122
#> 4 Zonata    1998     2003      Svédország       23
#> 5 Therion   1987     <NA>      Svédország     1266
```

7.2.5. Oszlopok létrehozása és törlése

Az *Alap R* `cbind()` függvényének a *Tidyverse R*-ben a `dplyr::bind_cols()` függvény felel meg, amely új oszlopokat ad hozzá az adattáblához. Az új oszlopokat a jobb áttekintést nyújtó `tibble::tribble()` függvény segítségével hozzuk létre. A `bind_cols()` függvény első argumentuma az adattábla neve, a második argumentum pedig az új oszlopokat tartalmazó adattábla neve.

```
# új oszlopok létrehozása
bandak.kieg <- tibble::tribble(
  ~wikipedia,                                ~rangsor,
  "https://en.wikipedia.org/wiki/Kiuas",          2,
  "https://hu.wikipedia.org/wiki/Accept",         4,
  "https://hu.wikipedia.org/wiki/Metallica",      1,
  "https://en.wikipedia.org/wiki/Zonata",         3,
  "https://en.wikipedia.org/wiki/Therion_(band)", 5
)
```

```
# új oszlopok hozzáadása
bandak.2 <- bandak |>
    bind_cols(bandak.kieg)
bandak.2 |>
    glimpse()
#> Rows: 5
#> Columns: 7
#> $ nev       <chr> "Kiuas", "Accept", "Metallica", "Zonata", "Therion"
#> $ alakulas  <chr> "2000", "1968", "1981", "1998", "1987"
#> $ felbomlas <chr> "2013", NA, NA, "2003", NA
#> $ orszag    <chr> "Finnország", "Németország", "USA", "Svédország", "S~
#> $ rajongo   <dbl> 106, 681, 4122, 23, 1266
#> $ wikipedia <chr> "https://en.wikipedia.org/wiki/Kiuas", "https://hu.w~
#> $ rangsor   <dbl> 2, 4, 1, 3, 5
```

Szúrjuk be a `rangsor` változót az eredeti `bandak` adatbázisba, majd távolítsuk el. Ehhez a *Tidyverse R* `dplyr::mutate()` és `dplyr::select()` függvényeit használjuk. A `mutate()` függvény első argumentuma az adattábla neve, a második argumentum pedig az új oszlop neve és értéke. Az új oszlop neve az = operátorral van megadva, míg az értékek egy vektorban szerepelnek. Az új oszlop törlésére a `select()` függvényt használjuk, amelynek első argumentuma az adattábla neve, a második argumentum pedig a törlendő oszlop neve – előjellel.

```
# új oszlop beszúrása
bandak <- bandak |>
    mutate(rangsor = c(2, 4, 1, 3, 5))
# oszlop törlése
bandak <- bandak |>
    select(-rangsor)
```

7.2.6. Típuskonverzió

Az oszlopok típusának ellenőrzése 7.2. és 7.3. táblázatok alapján a *Tidyverse R*-ben sem hagyható ki. A `class()` függvény segítségével ellenőrizhetjük az oszlopok típusát, majd ha szükséges a `dplyr::mutate()` függvény segítségével végezzük el a típuskonverziót. A `mutate()` függvény a `{dplyr}` csomag része, és az adattábla oszlopainak módosítására szolgál. Az oszlopok típusának megváltoztatásához a `mutate()` függvényen belül az `as.factor()`, `as.character()`, `as.numeric()` vagy `as.integer()` függvényeket használhatjuk. A `mutate()` függvény segítségével új oszlopokat is létrehozhatunk, vagy meglévő oszlopokat módosíthatunk.

```
# a kérdéses oszlopok típusa
apply(flow |> _[2:5], 2, class)
#>             nem     alkatoi.tev csaladi.allapot        isk.vegz
#>     "character"     "character"     "character"     "character"
```

A fenti 4 oszlopból a `nem`, az `alkatoi.tev` és a `csaladi.allapot` változókat faktorrá kell alakítanunk.

```
# faktorrá alakítás átnevezéssel
flow <- flow |>
  mutate(nem = factor(nem,
                      levels = c("Férfi", "Nő"),
                      labels = c("férfi", "nő")),
         alkatoi.tev = factor(alkatoi.tev,
```

```
                              levels = c("Igen", "Nem"),
                              labels = c("alkotó", "nem alkotó")),
         csaladi.allapot = factor(csaladi.allapot,
                                  levels = c("Házas", "Élettársi kapcsolatban él",
                                             "Elvált", "Özvegy", "Egyedülálló")))
```

Az `isk.vegz` változót rendezett faktorrá kell alakítanunk, hiszen ez ordinális változó. A helyes szintsorrend megadása természetesen kritikus fontosságú.

```
flow$isk.vegz |> unique() # ellenőrizzük a lehetséges értékeket
#> [1] "Egyetem"          "Gimnázium"        "Főiskola"
#> [4] "Szakközépiskola"  "Általános iskola"
# a sorrend megállapítása, átnevezés nélkül
flow <- flow |>
  mutate(isk.vegz = ordered(flow$isk.vegz,
                            levels = c("Általános iskola", "Szakközépiskola",
                                       "Főiskola", "Egyetem")))
```

7.2.7. Transzformáció

7.2.7.1. Numerikusból-numerikus transzformáció

A 7.3. példában a mértékegységváltás problémáját kellett megoldanunk. A `women` adattábla a nők súlyát és magasságát tartalmazza, a `weight` oszlopban fontban, míg a `height` oszlopban inch-ben. A feladatunk az volt, hogy a `weight` oszlopot kilogrammban, míg a `height` oszlopot centiméterben tároljuk.

```
# adatok beolvasása, és megtekintése
data(women)
women |>
    head()  # az első 6 sor megtekintése
#>   height weight
#> 1     58    115
#> 2     59    117
#> 3     60    120
#> 4     61    123
#> 5     62    126
#> 6     63    129
```

Az új oszlopok beszúrásához most is a `dplyr::mutate()` függvényt használjuk, amelynek első argumentuma az adattábla neve, a második argumentum pedig az új oszlop neve és értéke.

```
# mértékegységváltás: font-ból kg és inch-ből cm
women <- women |>
    mutate(suly = round(weight * 0.45), magassag = round(height * 2.54))
women |>
    head()  # az első 6 sor megtekintése
#>   height weight suly magassag
#> 1     58    115   52      147
#> 2     59    117   53      150
#> 3     60    120   54      152
#> 4     61    123   55      155
#> 5     62    126   57      157
#> 6     63    129   58      160
```

A 7.4. példa megoldása több numerikusból-numerikus transzformációt tartalmaz. Először végezzük el a fordított itemek átalakítását a szokásos módon a `dplyr::mutate()` függvény segítségével.

```
# a szükséges itemek fordítása
flow <- flow |>
    mutate(flow.5 = 6 - flow.5, flow.18 = 6 - flow.18)
```

A `flow` kérdőív kiértékelése átlagolással történik, így a `rowMeans()` függvényt használjuk, amely az egyes sorok átlagát számolja ki. A `na.rm=T` argumentum megadásával figyelmen kívül hagyhatjuk a hiányzó értékeket.

```
# az alskálákhoz tartozó oszlopok nevei
f1_nevek <- paste0("flow.", c(1, 2, 6, 7, 8, 12, 13, 14, 15, 16, 17))
f2_nevek <- paste0("flow.", c(3, 4, 5, 9, 10, 11, 18, 19, 20))

# átlagos skálapontszámok kiszámítása
flow <- flow |>
  rowwise() |>
  mutate(
    f1 = mean(c_across(all_of(f1_nevek)), na.rm = TRUE),
    f2 = mean(c_across(all_of(f2_nevek)), na.rm = TRUE),
    ossz = mean(c_across(all_of(c(f1_nevek, f2_nevek))), na.rm = TRUE)
  ) |>
```

```
  ungroup()

# az új oszlopok megtekintése
flow |> select(f1, f2, ossz) %>% head()
#> # A tibble: 6 x 3
#>      f1    f2  ossz
#>   <dbl> <dbl> <dbl>
#> 1  3.18  3.56  3.35
#> 2  4.18  4     4.1
#> 3  2.64  2.67  2.65
#> 4  3.64  2.89  3.3
#> 5  4.27  4.22  4.25
#> 6  3.73  4.11  3.9
```

A fenti kód első lépésében megneveztük az egyes skálákhoz tartozó oszlopokat az `f1_nevek` és `f2_nevek` vektorokban. Az adatok átlagának kiszámításához a `rowwise()` függvényt használtuk, amely lehetővé teszi, hogy az egyes sorok átlagát számoljuk ki. A `c_across()` függvény segítségével kiválasztottuk az oszlopokat, amelyeken átlagot számoltunk. Az új oszlopok neveit a `mutate()` függvényen belül adtuk meg. Az `ungroup()` függvény visszaállítja az eredeti adattábla csoportosítását, így a további műveletek már nem soronként történnek.

7.2.7.2. Numerikusból-faktor transzformáció

A 7.5. példa átvezet a numerikus-faktor átalakításba, hiszen a `kor` változó egy numerikus változó, amelyet 3 szintű faktorrá kell alakítanunk. A művelethez a már megismert `cut()` függvényt használjuk, de egy kicsit olvashatóbb formában, a *Tidyverse R* csomagcsalád `dplyr::mutate()` függvényével.

```
# numerikusból-faktor transzformáció
flow <- flow |>
  mutate(
    korosztaly = cut(
      kor,
      breaks = c(-Inf, 25, 55, Inf),
      labels = c("I.", "II.", "III."),
      right = TRUE
    )
  )
```

```
# gyakorisági tábla NA értékekkel együtt
flow |>
  count(korosztaly) |>
  complete(korosztaly, fill = list(n = 0))
#> # A tibble: 3 x 2
#>   korosztaly     n
#>   <fct>      <int>
#> 1 I.            60
#> 2 II.           39
#> 3 III.           1
```

7.2.7.3. Faktorból-faktor transzformáció

Faktorszintek összevonása egyet jelent a faktorból-faktor átalakítással. A `flow` adatmátrix `korosztaly` változóját használjuk, amely 3 szinttel rendelkezik: "I.", "II." és "III.". Egy új változót hozunk létre, amely a `korosztaly.2` változót hozunk létre, amely csak két szintet tartalmaz: "fiatal" és "nem fiatal".

A `dplyr::case_when()` függvény segítségével a `korosztaly` változó szintjeit új szintekbe csoportosíthatjuk. A `case_when()` függvény argumentumonként egy-egy logikai kifejezést vár, amelynek igazságértéke alapján a megfelelő új faktor szintet határozza meg.

```
# faktorból-faktor transzformáció
flow <- flow |>
  mutate(
    korosztaly.2 = case_when(
      korosztaly == "I." ~ "fiatal",
      korosztaly %in% c("II.", "III.") ~ "nem fiatal",
      TRUE ~ NA_character_
    ),
    korosztaly.2 = factor(korosztaly.2, levels = c("fiatal", "nem fiatal"))
  )

# gyakorisági tábla
flow |>
  count(korosztaly.2)
#> # A tibble: 2 x 2
#>   korosztaly.2     n
#>   <fct>        <int>
```

```
#> 1 fiatal          60
#> 2 nem fiatal      40
```

A 7.6. példa megoldásához szintén faktorból-faktor átalakítást kell végeznünk. A `csaladi.allapot` változóban található szintek közül a "Házas" és "Élettársi kapcsolatban él" szinteket egy új szintbe, a "Kapcsolatban él" szintbe csoportosítjuk. Az "Elvált", "Özvegy" és "Egyedülálló" szinteket egy új szintbe, az "Egyedül él" szintbe csoportosítjuk. Az új változó neve `csaladi.allapot.2` lesz.

A meglévő szintek ellenőrzésével kezdünk.

```
# szintnevek kiírása
flow$csaladi.allapot |>
    levels()
#> [1] "Házas"                    "Élettársi kapcsolatban él"
#> [3] "Elvált"                   "Özvegy"
#> [5] "Egyedülálló"
```

Az átalakítást most is a `dplyr::case_when()` függvény segítségével végezzük el.

```
# faktorból-faktor transzformáció
flow <- flow |>
  mutate(
    csaladi.allapot.2 = case_when(
      csaladi.allapot %in% c("Házas", "Élettársi kapcsolatban él")
                            ~ "Kapcsolatban él",
      TRUE ~ "Egyedül él"
    ),
    csaladi.allapot.2 = factor(csaladi.allapot.2,
                               levels = c("Kapcsolatban él", "Egyedül él"))
  )
# gyakorisági tábla
flow |>
  count(csaladi.allapot.2)
#> # A tibble: 2 x 2
#>   csaladi.allapot.2     n
#>   <fct>             <int>
#> 1 Kapcsolatban él      42
#> 2 Egyedül él           58
```

A fenti kód bemutatja, hogyan tudjuk kihasználni a `case_when()` függvény előnyeit. Elegendő tisztázni a "Kapcsolatban él" új szinthez tartozó régi szintek nevét, a többi szintet a `TRUE` kifejezéshez rendelhetjük. Látható, hogy a `mutate()` második argumentumában szereplő `factor()` függvény beállítja az új szintek sorrendjét is, amely a további elemzések során fontos lehet.

7.2.8. Sorok elnevezése

A 7.7. példában a `unisex_names` adattábla sorneveit határoztuk meg. Végezzük el az adatok beolvasását, és tekintsük meg az első 5 sort!

```
# adatok beolvasása
data(unisex_names, package = "fivethirtyeight")
unisex_names |> head(5) # az első 5 sor megtekintése
#> # A tibble: 5 x 5
#>   name     total male_share female_share    gap
#>   <chr>    <dbl>      <dbl>        <dbl>  <dbl>
#> 1 Casey  176544.      0.584        0.416 0.169
#> 2 Riley  154861.      0.508        0.492 0.0153
#> 3 Jessie 136382.      0.478        0.522 0.0443
#> 4 Jackie 132929.      0.421        0.579 0.158
#> 5 Avery  121797.      0.335        0.665 0.330
```

A `unisex_names` adattábla 4 változót tartalmaz, amelyből a `name` változóban található nevek alapján nevezzük el a sorokat. A `tibble::column_to_rownames()` függvény segítségével a `name` változó értékeit a sornevek helyére helyezzük. A `column_to_rownames()` függvény első argumentuma az adattábla neve, a második argumentum pedig a változó neve, amelynek értékeit a sornevek helyére szeretnénk tenni.

```
# a name változó értéke kerül a sornevek helyére
unisex_names <- unisex_names |>
    tibble::column_to_rownames(var = "name")
unisex_names |>
    head(5)  # az első 5 sor megtekintése
#>           total male_share female_share        gap
#> Casey  176544.3  0.5842866    0.4157134 0.16857313
#> Riley  154860.7  0.5076391    0.4923609 0.01527814
#> Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> Avery  121797.4  0.3352131    0.6647869 0.32957385
```

Látjuk, hogy a `name` változó kikerült az adattáblából, de könnyen visszatehetjük a `tibble::rownames_to_column()` függvény segítségével. A `rownames_to_column()` függvény első argumentuma az adattábla neve, a második argumentum pedig a változó neve, amelybe a sorneveket szeretnénk tenni. Az új oszlop neve lehet azonos a régi oszlop nevével, de ez nem kötelező.

```
# sornevek visszaírása oszlopvektorként
unisex_names <- unisex_names |>
    rownames_to_column(var = "name")
unisex_names |>
    head(5)  # az első 5 sor megtekintése
#>     name    total male_share female_share        gap
#> 1  Casey 176544.3  0.5842866    0.4157134 0.16857313
#> 2  Riley 154860.7  0.5076391    0.4923609 0.01527814
#> 3 Jessie 136381.8  0.4778343    0.5221657 0.04433146
#> 4 Jackie 132928.8  0.4211326    0.5788674 0.15773480
#> 5  Avery 121797.4  0.3352131    0.6647869 0.32957385
```

7.2.9. Sorok rendezése

A 7.8. példa a `flow` adattábla sorinak rendezését kérte. Első rendezési szempont a életkor csökkenő sorrendje, második rendezési szempont a skálapontszám növekvő sorrendje. A `dplyr::arrange()` függvény segítségével végezzük el a rendezést. Az `arrange()` függvény első argumentuma az adattábla neve, a második argumentum pedig a rendezési szempontot tartalmazó változó neve. A `desc()` függvény segítségével csökkenő sorrendbe állíthatjuk a rendezést.

```
# a legidősebb 5 válaszadó adata, plusz a skálapontszám növekvőbe
flow |>
    arrange(desc(kor), ossz) |>
    slice_head(n = 5) |>
    select(kor, nem, ossz)
#> # A tibble: 5 x 3
#>     kor nem    ossz
#>   <dbl> <fct> <dbl>
#> 1    69 nő     3.65
#> 2    53 nő     3.65
#> 3    53 férfi  3.65
#> 4    53 férfi  3.7
#> 5    52 férfi  3.25
```

Láthatjuk, hogy a *Tidyverse R* megoldás mennyire elegáns és könnyen olvasható.

7.2.10. Adattábla szűrése

A 7.9. példa adattábla szűrését kérte. Szűrésére a `dplyr::filter()` függvényt használjuk. A `filter()` függvény első argumentuma az adattábla neve, a második argumentum pedig a szűrési feltétel. A szűrési feltétel logikai kifejezés, amelynek eredménye `TRUE` vagy `FALSE`. Azok a sorok kerülnek leválogatásra, amelyek pozíciójában a `TRUE` érték szerepel a logikai vektorban. Több szűrési feltételt is megadhatunk, amelyeket a `&` operátorral kapcsolhatunk össze, vagy egyszerűen vesszővel elválasztva soroljuk őket a `filter()` függvény második, harmadik stb. argumentumában. Az alábbi példában a `flow` adattábla szűrésére kerül sor, amelyben a `nem` változó "férfi" értéket vesz fel, és a `kor` változó 20 és 25 közötti értékeket tartalmaz.

```
# adattábla szűrése: férfiak, 20 és 25 év között
flow_szukitett <- flow |>
  filter(nem == "férfi", kor >= 20, kor <= 25)

flow.szukitett |> dim() # az új adattábla mérete
#> [1] 16 31
```

Az `NA` értékek szűrésére a `filter()` függvény használatakor külön figyelmet kell fordítanunk. Ugyan csupa `NA`-t tartalmazó sorokat nem fogunk kapni szűrési eredményként, mint az *Alap R*-ben, de ha az `NA` értékekre szükség van a leválogatás során, akkor a szokásos `| is.na()` konstrukciót most is használnunk kell. A probléma megértéséhez tekintsük át újra a példánkat.

Az alábbi kódban egy adatmátrixot olvasunk be, amely tartalmaz egy-egy hiányzó értéket.

```
adat <- tribble(
  ~modszer,     ~pontszam,
      "A",             42,
       NA,             67,
      "B",             NA,
      "B",             32
)
df
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
```

```
#> 3       B       NA
#> 4       B       32
```

Mindkét oszlop tartalmaz egy-egy hiányzó értéket. Így vizsgáljuk meg a lenti szűrések eseteit.

```
# egyenlőség vizsgálat, ha nem kell az NA, akkor jó lehet
df |>
    filter(modszer == "A")
#>   modszer pontszam
#> 1       A       42
df |>
    filter(modszer %in% "A")
#>   modszer pontszam
#> 1       A       42
```

Látható, hogy a `filter()` függvény megelőzi a csupa `NA`-s sorok visszaadását, de az `NA` értékeket tartalmazó sorokat nem adja vissza.

Abban az esetben, ha a `modszer` oszlopban az `NA` értéket tartalmazó sorokat is látni szeretnénk az outputban, akkor ki kell egészítenünk a lekérdezést egy `is.na()` kifejezéssel, amit az `|` operátorral kapcsolunk össze a `modszer` oszlop szűrésével.

```
# ha kell az NA, akkor jó lehet
df |>
    filter(modszer == "A" | is.na(modszer))
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
df |>
    filter(modszer %in% "A" | is.na(modszer))
#>   modszer pontszam
#> 1       A       42
#> 2    <NA>       67
```

A fenti outputokból kiolvasható, hogy mindkét esetben megjelentek azok a sorok, amelyekben a `modszer` oszlopban `NA` érték szerepel.

Numerikus értékek esetében is hasonló összehasonlítást végezhetünk, szinte megegyező eredménnyel.

```
# ha nem kell az NA, akkor jó lehet
df |>
    filter(pontszam > 50)
#>   modszer pontszam
#> 1    <NA>       67
```

```
# ha kell az NA, akkor jó lehet
df |>
    filter(pontszam > 50 | is.na(pontszam))
#>   modszer pontszam
#> 1    <NA>       67
#> 2       B       NA
```

A fentiek alapján készítsünk biztonságos szűrést, amely az `NA` értékeket nem zárja ki eleve a lekérdezésből és csupa `NA`-s sorokat sem szúr be. Ehhez az `| is.na()` kifejezést használjuk. Az `NA` értékek figyelembe vételével a fenti példát így írhatjuk át:

```
flow_szukitett <- flow |>
  filter(nem == "férfi" | is.na(nem),
         kor >= 20      | is.na(kor),
         kor <= 25      | is.na(kor)
         )
flow.szukitett |> dim() # az új adattábla mérete
#> [1] 16 31
```

7.2.11. Hiányzó értékek kezelése

A 7.10. példán keresztül áttekinthetjük a hiányzó értékek kezelését *Tidyverse R*-ben.

A hiányzó értékek felderítésének módja továbbra is az `is.na()` függvény, amely logikai vektort ad vissza, amelyben a hiányzó értékek helyén `TRUE`, míg a nem hiányzó értékek helyén `FALSE` szerepel. Ezt a függvényt a `summarise()` függvényen belül is használhatjuk, amely lehetővé teszi, hogy az adattábla egyes oszlopainak statisztikai mutatóit számoljuk ki. A `summarise()` függvény első argumentuma az adattábla neve, a második argumentum pedig a számolni kívánt statisztikai mutató neve és értéke. Ez esetünkben a `sum(is.na(Sex)` lesz, amely a `Sex` változóban található hiányzó értékek számát adja vissza. A `summarise()` függvény eredménye egy új adattábla, amelyben csak a számolt statisztikai mutatók szerepelnek.

```r
data(survey, package = "MASS")  # a survey adattábla betöltése
# hiányzó értékek számának meghatározása
survey |>
    summarise(hianyzo_ertekek = sum(is.na(Sex)))
#>   hianyzo_ertekek
#> 1               1
```

Látható, hogy a nem változóban mindössze 1 hiányzó érték van.

Az összes oszlopra is meghatározhatjuk a hiányzó értékeket, ehhez az `across()` függvényt használjuk, amely lehetővé teszi, hogy az összes oszlopra alkalmazzuk a `\(x ) sum(is.na(x))` névtelen függvényt. Az `across()` függvény első argumentuma az oszlopok neve, a második argumentum pedig a számolni kívánt statisztikai mutató neve és értéke. A `summarise()` függvény eredménye most egy új adattábla, amelyben csak a számolt statisztikai mutatók szerepel.

```r
# a hiányzó értékek számának meghatározása
survey |>
    summarise(across(everything(), \(x) sum(is.na(x))))
#>   Sex Wr.Hnd NW.Hnd W.Hnd Fold Pulse Clap Exer Smoke Height M.I Age
#> 1   1      1      1     1    0    45    1    0     1     28  28   0
```

Meghatározhatjuk azoknak a soroknak a számát is, amelyek teljesek, azaz hiányzó értéket egyáltalán nem tartalmaznak. Ezt a `complete.cases()` függvény segítségével végezhetjük el, amely egy logikai vektort ad vissza, amelyben a teljes sorok helyén `TRUE`, míg a hiányzó értékeket tartalmazó sorok helyén `FALSE` szerepel.

```r
# a teljes sorok számának meghatározása
survey |>
    complete.cases() |>
    sum()
#> [1] 168
# a nem teljes sorok számának meghatározása !!
survey |>
    {
        \(x) !complete.cases(x)
    }() |>
    sum()
#> [1] 69
```

A teljes sorokat tartalmazó adattáblát a `tidyr::drop_na()` függvény segítségével hozhatjuk létre, amely eltávolítja az összes olyan sort, amelyben hiányzó érték szerepel. A `drop_na()` függvény első argumentuma az adattábla neve, a második argumentum pedig a figyelembe vett oszlopok neve. Ha nem adunk meg oszlopneveket, akkor az összes oszlopra alkalmazza az `NA` értékek figyelését.

```
# a hiányzó értékeket tartalmazó sorok eltávolítása
survey.teljes <- survey |>
    tidyr::drop_na()
# a csak hiányzó értékeket tartalmazó sorok adatbázisa
survey.hianyzok <- survey |>
    filter(if_any(everything(), is.na))
```

Gyakori adatmanipulációs lépés a hiányzó értékek helyettesítése egy érvényes értékkel. A `tidyr::replace_na()` függvény segítségével helyettesíthetjük a hiányzó értékeket egy érvényes értékkel. A `replace_na()` függvény első argumentuma az oszlop neve, a második argumentum pedig a helyettesítendő érték. A `replace_na()` függvény eredménye egy új, már `NA`-t nem tartalmazó oszlop.

```
# hiányzó testmagasságok helyettesítése az átlaggal
survey <- survey |>
    mutate(Height = replace_na(Height, mean(Height, na.rm = TRUE)))
survey$Height |>
    is.na() |>
    sum()  # már nincsenek hiányzó értékek
#> [1] 0
```

Összefoglalás

A *Tidyverse R* adatkezelési megközelítése egy modern, adatszemléletű alternatívát kínál az *Alap R* hagyományos eszközeihez képest. A `{dplyr}` és `{tidyr}` csomagokra épülő rendszer célja, hogy az adatokkal végzett műveletek könnyen olvashatók, lépésről lépésre követhetők, és egymás után láncolhatók legyenek. Ennek legfontosabb eszköze a pipe operátor (`|>`), amely lehetővé teszi, hogy a műveleteket egymás után sorban, logikusan építsük fel. Az adatok megismerését a `glimpse()` függvény segíti, amely tömör és informatív áttekintést nyújt az adattábla szerkezetéről. A gyakorisági eloszlásokat a `count()` függvénnyel kaphatjuk meg, amely egyszerű és rendezhető táblázatot ad vissza. Az adattábla részeinek kiválasztására a `select()` és `slice()` függvények szolgálnak. Az oszlopok elnevezését és sorrendjének megváltoztatását a `rename()` és `select()` függvények teszik lehetővé. Új oszlopokat a `mutate()` segítségével hozhatunk létre,

amely emellett a meglévők módosítására is alkalmas. Típuskonverzió során például karakteres változóból faktort a `mutate()` és `factor()` kombinációjával készíthetünk, míg a rendezett faktorokhoz az `ordered()` függvény használható. A transzformációk három fő típusát különböztetjük meg: numerikusból-numerikus, numerikusból-faktor, és faktorból-faktor átalakítást. Az első esetben tipikus példa a mértékegység-váltás, ahol például font-ot kilogrammra számolunk át. A numerikusból-faktor konverzió esetén a `cut()` függvény segít az adatok kategóriákba sorolásában, például életkor alapján. A faktorszintek összevonását `case_when()` segítségével végezhetjük, ahol több korábbi szintet egy új szintbe gyűjthetünk. A sorok rendezésére az `arrange()` szolgál, ahol a `desc()` függvény segítségével csökkenő sorrendet is kérhetünk. A `filter()` segítségével pedig feltétel szerinti szűréseket hajthatunk végre. Ebben a környezetben is fontos figyelni az `NA` értékekre, és ha szeretnénk, hogy ezek is megjelenjenek a lekérdezés eredményében, akkor az `is.na()` függvényt is be kell vonnunk a logikai kifejezésbe. A hiányzó értékek kezelése *Tidyverse R* környezetben különösen kényelmes. Az `is.na()` függvénnyel kiszűrhetjük a hiányzó értékeket, a `summarise()` és `across()` kombinációjával pedig egyszerre kérdezhetjük le több oszlop hiányzó értékeinek számát. A `drop_na()` segítségével egyszerűen eltávolíthatók azok a sorok, amelyek bármelyik változóban hiányzó értéket tartalmaznak. Ezzel szemben, ha pótolni szeretnénk az `NA`-kat, például az oszlopok átlagával, akkor a `replace_na()` függvény nyújt hatékony megoldást. A `column_to_rownames()` és `rownames_to_column()` függvények lehetővé teszik, hogy egy oszlopból sorneveket készítsünk, vagy éppen a sorneveket új oszlopként visszaírjuk az adattáblába. Ezek a funkciók akkor hasznosak, ha például azonosító jellegű információt tartalmaz egy oszlop, amelyet inkább a sornevek között szeretnénk látni.

🎯 Feladatok

1. A `survey` adattáblában a `Height` változó hiányzó értékeit helyettesítsük a változó mediánjával! Használjuk a *Tidyverse R* eszközeit! Vessük össze az *Alap R*-beli megoldással! Mi a véleményünk a két megoldásról, az olvashatóság és karbantarthatóság szempontjából?
2. A `survey` adattáblában számoljuk a BMI értéket, majd alakítsuk át kategorikus változóvá! Használjuk a *Tidyverse R* eszközeit!
3. A `HSAUR3::Forbes2000` adattáblája 2000 vállalat adatát tartalmazza! Határozzuk meg a magyar cégek nevét és helyezését (`country` oszlop alapján)! Írassuk ki a képernyőre a 10 legnagyobb piaci értékkel (`marketvalue` oszlop) rendelkező cég nevét és piaci értékét! Határozzuk meg a legkisebb profittal (`profits` oszlop) rendelkező 5 cég minden adatát! Határozzuk meg a legnagyobb profittal (`profits` oszlop) rendelkező 10 amerikai vagy japán cég nevét, országát és profitját! Használjuk a *Tidyverse R* eszközeit!

7.3. Haladó adatkezelés 🤯

> **Miről lesz szó? Ebben a fejezetben**
>
> - áttekintjük az adattáblák összekapcsolását,
> - a széles-hosszú és hosszú-széles formátum közötti átalakítást, és
> - az oszlopok szétválasztását és egyesítését.

Megismertük az adatelőkészítés alapjait, azokat a leggyakoribb műveleteket, amelyeket közvetlenül az adatok beolvasása után, és a konkrét adatelemzési lépések előtt el szoktunk végezni egy tipikus adatelemzés során. Az 7. fejezetben az *Alap R* , a 7.2. fejezetben pedig a *Tidyverse R* eszközeivel mutattuk be az adatok előkészítését. A fentiekben bemutatott műveletek közé tartozott többek között az információgyűjtés, az oszlopok és sorok kiválasztása, új oszlopok létrehozása, oszlopok típusának megváltoztatása, a sorok rendezése és szűrése.

Az adatelőkészítés azonban sokkal szélesebb spektrumot ölel fel, mint amit a fenti fejezetekben bemutattunk, szinte lehetetlen minden egyes előkészítő lépést számba venni. Ebben a fejezetben mégis néhány további, nagyobb figyelmet igénylő és egyben hasznos műveletet mutatunk be. A haladó adatkezeléshez a következő tevékenységet soroljuk ebben a fejezetben:

- adattáblák összekapcsolása,
- széles-hosszú és hosszú-széles formátum közötti átalakítás,
- oszlopok szétválasztása és egyesítése.

A fejezetben a *Tidyverse R* eszközeivel igyekszünk a bemutatott műveleteket végrehajtani, de természetesen az *Alap R* eszközeivel is megoldhatóak lennének ezek a feladatok.

7.3.1. Adattáblák összekapcsolása

Tegyük fel, hogy elemzésünk során a vizsgálati személyek demográfia adatait (nem, életkor, iskolai végzettség stb.) és a vizsgálat során keletkező adatokat (például reakcióidő) külön tároljuk. Az elemzés egy adott fázisában szükség lehet a két adattábla egyesítésére. Ez a művelet az adatbázis-kezelés szokásos *összekapcsolás* vagy *join* művelete, amire az R-ben is lehetőség/szükség van.

Olvassuk be a demográfiai adatokat:

```
d.szemely <- tribble(
  ~id, ~nem, ~kor,
  "KS", "f", 12,
  "TR", "f", 11,
  "SE", "f", 12
)
d.szemely
#> # A tibble: 3 x 3
#>   id    nem     kor
#>   <chr> <chr> <dbl>
#> 1 KS    f        12
#> 2 TR    f        11
#> 3 SE    f        12
```

Láthatjuk, hogy 3 személyről van szó, akik a `KS`, `TR` és `SE` azonosítókkal rendelkeznek. Olvassuk be a vizsgálat során keletkező adatokat, személyenként 2-2 reakcióidő méréssel:

```
d.vizsgalat <- tribble(
   ~id, ~kerdes,   ~reakcioido,
  "KS",    "k.1",          1305,
  "KS",    "k.2",          1238,
  "TR",    "k.1",          1427,
  "TR",    "k.2",          1391,
  "SE",    "k.1",          1265,
  "SE",    "k.2",          1405)
d.vizsgalat
#> # A tibble: 6 x 3
#>   id    kerdes reakcioido
#>   <chr> <chr>       <dbl>
#> 1 KS    k.1          1305
#> 2 KS    k.2          1238
#> 3 TR    k.1          1427
#> 4 TR    k.2          1391
#> 5 SE    k.1          1265
#> 6 SE    k.2          1405
```

A fenti adattáblát úgy értelmezhetjük, hogy a `KS` azonosítójú személy 2 kérdésre adott válaszának reakcióidői 1305 és 1238 ms, míg a `TR` azonosítójú személy 2 kérdésre adott válaszának reakcióidői 1427 és 1391 ms. A `SE` azonosítójú személy 2 kérdésre adott válaszának reakcióidői pedig 1265 és 1405 ms.

A fenti két adattábla egyesítése során a közös oszlop a `id`, amely azonosítja a vizsgálati személyeket. Egy új adattáblát szeretnénk kapni, amely tartalmazza a demográfiai adatokat és a vizsgálati adatokat is.

A `dplyr::left_join()` függvény segítségével a két adattáblát összekapcsolhatjuk. A `left_join()` függvény első argumentuma az adattábla neve, amelynek minden sorát meg szeretnénk tartani (jelen esetben a `d.vizsgalat`), a második argumentum pedig a kapcsolódó adattábla neve (jelen esetben a `d.szemely`). Az összekapcsolás során a közös oszlop nevét nem lenne fontos megadni, hiszen a `id` oszlop mindkét adattáblában szerepel.

```
# adattáblák összekapcsolása: left join
df <- d.vizsgalat |>
    left_join(d.szemely, by = "id")
df
#> # A tibble: 6 x 5
#>   id    kerdes reakcioido nem     kor
#>   <chr> <chr>       <dbl> <chr> <dbl>
#> 1 KS    k.1          1305 f        12
#> 2 KS    k.2          1238 f        12
#> 3 TR    k.1          1427 f        11
#> 4 TR    k.2          1391 f        11
#> 5 SE    k.1          1265 f        12
#> 6 SE    k.2          1405 f        12
```

Az adattáblák sorrendjét a `left_join()` függvényben meg is fordíthattuk volna. Ha a `d.szemely` adattáblát helyezzük bal oldalra, akkor a `d.vizsgalat` adattábla lesz a kapcsolódó adattábla. Ilyenkor a `d.szemely` adattábla minden sora megmarad, és az `id` oszlop azonos értékei alapján a `d.vizsgalat` adattábla adatai kerülnek hozzá.

```
# adattáblák összekapcsolása: left join
df <- d.szemely |>
    left_join(d.vizsgalat, by = "id")
df
#> # A tibble: 6 x 5
#>   id    nem     kor kerdes reakcioido
#>   <chr> <chr> <dbl> <chr>       <dbl>
#> 1 KS    f        12 k.1          1305
#> 2 KS    f        12 k.2          1238
#> 3 TR    f        11 k.1          1427
#> 4 TR    f        11 k.2          1391
```

```
#> 5 SE    f        12 k.1          1265
#> 6 SE    f        12 k.2          1405
```

Látjuk, hogy a két összekapcsolási sorrend között annyi az eltérés, hogy az oszlopok neve az adattáblák sorrendjétől függően változik. Az első esetben a `d.vizsgalat` adattábla oszlopai kerülnek előre, míg a második esetben a `d.szemely` adattábla oszlopai kerülnek előre. Annak, hogy a két összekapcsolás azonos eredményre vezet, az az oka, hogy a `d.szemely` adattábla minden sorára van adat a `d.vizsgalat` adattáblában, és fordítva, a `d.vizsgalat` adattábla minden személye szerepel a `d.szemely` adattáblában. Az összekapcsolás során ugyanis a `left_join()` függvény a bal oldali adattábla minden sorát megtartja, és csak azokat a sorokat kapcsolja össze a jobb oldali adattáblával, amelyekben a közös oszlopban szereplő értékek megegyeznek.

Azonban ez az ideális eset nem mindig teljesül. Tegyük fel, hogy még egy személyről van demográfiai adatunk, de ő még nem végezte a kísérletet.

```
# új személy adataival bővítjük a d.szemely adattáblát
d.szemely.uj <- d.szemely |>
    add_row(id = "XY", nem = "f", kor = 12)
d.szemely.uj
#> # A tibble: 4 x 3
#>   id    nem     kor
#>   <chr> <chr> <dbl>
#> 1 KS    f        12
#> 2 TR    f        11
#> 3 SE    f        12
#> 4 XY    f        12
```

Látható, hogy a `d.szemely.uj` adattáblában szerepel egy új személy, akinek az azonosítója `XY`, neme `f`, kora pedig 12 év. A `d.vizsgalat` adattáblában azonban nem szerepel ez a személy.

Ekkor az összekapcsolásnál figyelnünk kell az adattáblák helyes sorrendjére a `left_join()` függvényben. A bal oldalra a `d.szemely.uj` adattáblát kell írnunk, jobb oldalra megy a `d.vizsgalat` adattábla.

```
# adattáblák összekapcsolása
df <- d.szemely.uj |>
    left_join(d.vizsgalat, by = "id")
print(df, n = 10)
#> # A tibble: 7 x 5
#>   id    nem     kor kerdes reakcioido
```

```
#>   <chr> <chr> <dbl> <chr>       <dbl>
#> 1 KS    f        12 k.1          1305
#> 2 KS    f        12 k.2          1238
#> 3 TR    f        11 k.1          1427
#> 4 TR    f        11 k.2          1391
#> 5 SE    f        12 k.1          1265
#> 6 SE    f        12 k.2          1405
#> 7 XY    f        12 <NA>           NA
```

Ellenkező sorrend esetén adatvesztés történhet. Ellenőrizzük le!

```
# adattáblák összekapcsolása: adatvesztés
df <- d.vizsgalat |>
    left_join(d.szemely.uj, by = "id")
print(df, n = 10)
#> # A tibble: 6 x 5
#>   id    kerdes reakcioido nem     kor
#>   <chr> <chr>       <dbl> <chr> <dbl>
#> 1 KS    k.1          1305 f        12
#> 2 KS    k.2          1238 f        12
#> 3 TR    k.1          1427 f        11
#> 4 TR    k.2          1391 f        11
#> 5 SE    k.1          1265 f        12
#> 6 SE    k.2          1405 f        12
```

A fenti összekapcsolás esetén a XY azonosítójú személy adatai (neme és kora) eltűntek a további elemzés számára. Tehát az előbbi sorrendet (`d.szemely.uj` bal oldalon) kell alkalmaznunk.

Természetesen az is előfordulhat, hogy a `d.vizsgalat` adattáblában szerepelnek olyan személyek, akiknek a demográfiai adatai nem állnak rendelkezésre. Hozzunk létre egy ilyen `d.vizsgalat.uj` adattáblát:

```
# új személy adataival bővítjük a d.vizsgalat adattáblát
d.vizsgalat.uj <- d.vizsgalat |>
    add_row(id = "QW", kerdes = "k.3", reakcioido = 1234)
print(d.vizsgalat.uj, n = 10)
#> # A tibble: 7 x 3
#>   id    kerdes reakcioido
#>   <chr> <chr>       <dbl>
#> 1 KS    k.1          1305
```

```
#> 2 KS    k.2          1238
#> 3 TR    k.1          1427
#> 4 TR    k.2          1391
#> 5 SE    k.1          1265
#> 6 SE    k.2          1405
#> 7 QW    k.3          1234
```

Láthatjuk, hogy megjelent a `QW` azonosítójú személy, akinek a `d.vizsgalat` adattáblában szerepel a reakcióideje, de a demográfiai adatai nem állnak rendelkezésre. Ha most a `d.vizsgalat.uj` adattáblát az eredeti a `d.szemely` adattáblával kapcsoljuk össze, akkor a `QW` azonosítójú személy adatai megjelennek a `df` adattáblában, de az `NA` értékekkel. Ehhez arra van szükség, hogy a `d.vizsgalat.uj` adattáblát bal oldalon helyezzük el a `left_join()` függvényben, ellenkező esetben szintén adatvesztés történik.

```
# adattáblák összekapcsolása: left join
df <- d.vizsgalat.uj |>
    left_join(d.szemely, by = "id")
print(df, n = 10)
#> # A tibble: 7 x 5
#>   id    kerdes reakcioido nem     kor
#>   <chr> <chr>       <dbl> <chr> <dbl>
#> 1 KS    k.1          1305 f        12
#> 2 KS    k.2          1238 f        12
#> 3 TR    k.1          1427 f        11
#> 4 TR    k.2          1391 f        11
#> 5 SE    k.1          1265 f        12
#> 6 SE    k.2          1405 f        12
#> 7 QW    k.3          1234 <NA>     NA
```

Utolsó esetként nézzük meg azt a szintén gyakori esetet, amikor a `d.vizsgalat.uj` adattáblát a `d.szemely.uj` adattáblával szeretnénk összekapcsolni. Ez annak felel meg, hogy demográfiai adat rendelkezésre áll még nem vizsgált személyről, illetve ezzel egyidőben, a vizsgálati adatot tárolunk demográfiai adattal nem rendelkező személyről.

Ekkor egy új függvényre van szükség, hiszen mindkét adattábla tartalmaz olyan `id` értéket, amely a másikban nem szerepel, és a `left_join()` függvény ügyes használata már nem tudja megoldani a feladatot. A `dplyr::full_join()` függvény segítségével azonban a két adattáblát összekapcsolhatjuk, amely megőrzi az összes sort mindkét adattáblából, és a hiányzó értékeket `NA`-val tölti ki.

```
# adattáblák összekapcsolása: full join
df <- d.vizsgalat.uj |>
    full_join(d.szemely.uj, by = "id")
print(df, n = 10)
#> # A tibble: 8 x 5
#>   id    kerdes reakcioido nem     kor
#>   <chr> <chr>       <dbl> <chr> <dbl>
#> 1 KS    k.1          1305 f        12
#> 2 KS    k.2          1238 f        12
#> 3 TR    k.1          1427 f        11
#> 4 TR    k.2          1391 f        11
#> 5 SE    k.1          1265 f        12
#> 6 SE    k.2          1405 f        12
#> 7 QW    k.3          1234 <NA>     NA
#> 8 XY    <NA>           NA f        12
```

Láthatjuk, hogy a `full_join()` függvénnyel el tudtuk kerülni az adatvesztést, és az `NA` értékek jelennek meg a hiányzó adatok helyén.

7.3.2. Széles-hosszú átalakítás

Adataink változatos formában állhatnak rendelkezésre. Ez különösen igaz, ha ismételt méréses adatgyűjtésről van szó, amikor ugyanazt a mérést többször végezzük el ugyanazon vizsgálati személyen. Tegyük fel, hogy tesztet három különböző időpontban, reggel, délben és este is ki kell tölteni a vizsgálati személyeknek. Minden személyhez tehát három tesztpontszám tartozik. Ezeket az adatokat két alapvetően különböző formátumban tárolhatjuk. Az egyik a széles formátum, a másik a hosszú formátum.

Nézzük meg konkrét adatbázison is a példát. Kezdjük a széles formátummal, amelyben minden egyes tesztpontszám külön oszlopban szerepel.

```
# széles adatbázis létrehozása: id, kor, nem, reggel, delben, este
df_szeles <- tribble(
  ~id,   ~kor, ~nem, ~reggel, ~delben, ~este,
  "KS",   12,   "f",      10,      20,    30,
  "TR",   11,   "f",      15,      25,    35,
  "SE",   12,   "l",      20,      30,    40
)
print(df_szeles, n=10)
```

```
#> # A tibble: 3 x 6
#>   id      kor nem   reggel delben  este
#>   <chr> <dbl> <chr>  <dbl>  <dbl> <dbl>
#> 1 KS       12 f         10     20    30
#> 2 TR       11 f         15     25    35
#> 3 SE       12 l         20     30    40
```

Látható, hogy a fenti adatbázist azért nevezhetjük széles formátumúnak, mert minden egyes tesztpontszám külön oszlopban szerepel. Ha lenne még egy mérés (például éjjel), akkor az adatbázis szélessége nőne, vagyis új oszlopként kerülnének be az éjjeli mérési eredmények.

A statisztikai programok egy része az ismételt méréseket tartalmazó adatbázisokat, ebben a széles formátumban várják. Ilyen például a jamovi vagy az SPSS is. Az R-ben többnyire az ismételt méréseket hosszú formátumban tároljuk, ami azt jelenti, hogy a tesztpontszámokat egyetlen oszlopban rögzítjük, az ismételt mérések egymás alá kerülnek.

Nézzük meg a hosszú formátumú adatbázist is!

```
# hosszú adatbázis létrehozása: id, kor, nem, mikor, pontszam
df_hosszu <- tribble(
   ~id, ~kor,  ~nem,     ~mikor,   ~pontszam,
  "KS",   12,   "f",   "reggel",          10,
  "KS",   12,   "f",   "delben",          20,
  "KS",   12,   "f",     "este",          30,
  "TR",   11,   "f",   "reggel",          15,
  "TR",   11,   "f",   "delben",          25,
  "TR",   11,   "f",     "este",          35,
  "SE",   12,   "l",   "reggel",          20,
  "SE",   12,   "l",   "delben",          30,
  "SE",   12,   "l",     "este",          40
)
print(df_hosszu, n=10)
#> # A tibble: 9 x 5
#>   id      kor nem   mikor  pontszam
#>   <chr> <dbl> <chr> <chr>     <dbl>
#> 1 KS       12 f     reggel       10
#> 2 KS       12 f     delben       20
#> 3 KS       12 f     este         30
#> 4 TR       11 f     reggel       15
#> 5 TR       11 f     delben       25
#> 6 TR       11 f     este         35
```

```
#> 7 SE       12 l     reggel       20
#> 8 SE       12 l     delben       30
#> 9 SE       12 l     este         40
```

Látható, hogy a hosszú formátumú adatbázisban a tesztpontszámok egyetlen oszlopban szerepelnek, és a `mikor` oszlopban található az ismételt mérések időpontja. Így tudjuk beazonosítani az egyes méréseket, az `id` megmondja, hogy melyik személyhez tartozik, a `mikor` oszlop pedig megmondja, hogy mikor történt a mérés, reggel, délben vagy este. A `kor` és `nem` oszlopok pedig a vizsgálati személyek demográfiai adatait tartalmazzák, és látjuk, hogy redundánsan, többször ismétlődnek a hosszú formátumú adatbázisban.

Világos a hosszú formátum neve is, hiszen azzal, hogy a tesztpontszámokat egyetlen oszlopban tároljuk, egy új mérés (például éjjel) az adatbázis hosszúsága nő, új sorok hozzáadásával.

Ennek a fejezetnek az a célja, hogy bemutassa, hogyan tudunk széles formátumú adatbázist hosszú formátumúvá alakítani, és fordítva, hogyan tudunk hosszú formátumú adatbázist széles formátumúvá alakítani. A `{tidyr}` csomag `pivot_longer()` és `pivot_wider()` függvényei segítségével könnyedén elvégezhetjük a kívánt átalakítást.

Amennyiben széles formátumú `df_szeles` adattáblából indulunk ki, akkor a `tidyr::pivot_longer()` függvény segítségével alakíthatjuk át hosszú formátumúvá. A `pivot_longer()` függvény első argumentuma az adattábla neve, a második argumentum pedig azoknak az oszlopoknak a neve, amelyeket hosszú formátumúvá szeretnénk alakítani. Az új oszlopok nevét a `names_to` argumentumban adhatjuk meg, míg az új oszlop értékeit a `values_to` argumentumban.

```
# széles-hosszú átalakítás
df_hosszu_uj <- df_szeles |>
  pivot_longer(cols = c(reggel, delben, este),
               names_to = "mikor",
               values_to = "pontszam")
print(df_hosszu_uj, n=10)
#> # A tibble: 9 x 5
#>   id      kor nem   mikor  pontszam
#>   <chr> <dbl> <chr> <chr>     <dbl>
#> 1 KS       12 f     reggel       10
#> 2 KS       12 f     delben       20
#> 3 KS       12 f     este         30
#> 4 TR       11 f     reggel       15
#> 5 TR       11 f     delben       25
#> 6 TR       11 f     este         35
#> 7 SE       12 l     reggel       20
```

```
#> 8 SE       12 l     delben       30
#> 9 SE       12 l     este         40
```

Láthatjuk, hogy a `col=` argumentumban elegendő volt megadni a széles formátumú adattáblában szereplő, ismételt méréseket tartalmazó oszlopok nevét. Ezek el fognak tűnni a hosszú formátumú adattáblából, és helyettük 2 új oszlop fog megjelenni. Az egyik oszlop a `names_to` argumentumban megadott oszlopnév fog szerepelni, amely a mérések időpontját tartalmazza, és lényegében a `cols=`-ban megadott oszlopok neveit tartalmazza fogja tartalmazni. A másik a `values_to` argumentumban megadott oszlopnév (`pontszam`) lesz, amely azokat a tesztpontszámokat tartalmazza, amelyek eddig `col=`-ban megadott oszlopokban szerepeltek.

Ha összehasonlítjuk a most kapott `df_hosszu_uj` adattáblát a korábban létrehozott `df_hosszu` adattáblával, akkor láthatjuk, hogy a két adattábla azonos tartalommal bír, csak a sorok sorrendjében különbözhetnek.

Ha a másik irányt tekintjük, vagyis most a hosszú formátumú `d_hosszu` adattáblát szeretnénk széles formátumúvá alakítani, akkor a `tidyr::pivot_wider()` függvényt kell használnunk. A `pivot_wider()` függvény első argumentuma az adattábla neve, a második argumentum (`names_from`) pedig az az oszlop neve, amelyből az új oszlopok neveit szeretnénk létrehozni. A harmadik argumentum (`values_from`) pedig az az oszlop neve, amelynek értékeit szeretnénk széles formátumúvá alakítani.

```
# hosszú-széles átalakítás
df_szeles_uj <- df_hosszu |>
  pivot_wider(names_from = mikor,
              values_from = pontszam)
print(df_szeles_uj, n=10)
#> # A tibble: 3 x 6
#>   id      kor nem   reggel delben  este
#>   <chr> <dbl> <chr>  <dbl>  <dbl> <dbl>
#> 1 KS       12 f         10     20    30
#> 2 TR       11 f         15     25    35
#> 3 SE       12 l         20     30    40
```

Láthatjuk, hogy a `names_from` argumentumban megadott oszlop (`mikor`) értékei alapján új oszlopok jönnek létre, amelyek értékei a `values_from` argumentumban megadott oszlop (`pontszam`) értékeit tartalmazzák. Ha összehasonlítjuk a most kapott `df_szeles_uj` adattáblát a korábban létrehozott `df_szeles` adattáblával, akkor láthatjuk, hogy a két adattábla azonos tartalommal bír, csak a sorok sorrendjében különbözhetnek.

7.3.3. Oszlopok szétválasztása és egyesítése

Előfordulhat, hogy adattáblánk egyik oszlopa több olyan információt sűrít magába, amelyet külön-külön oszlopokba szeretnénk elrendezni. A `tidyr::separate()` függvény segítségével egy oszlopot több oszlopra tudunk bontani. Ha tekintünk egy standard dátum mezőt, amelyben a dátum `év-hónap-nap` formában szerepel, akkor három új oszlopot is létrehozhatunk ezen dátum oszlop alapján. Tekintsük a következő adatbázist:

```
# adatok beolvasása
library(tidyverse)
df <- tribble(
    ~nev,           ~datum,
  "Anna",     "2023-01-01",
  "Béla",     "2023-02-02",
 "Csaba",     "2023-03-03",
  "Dóra",     "2023-04-04"
)
df
#> # A tibble: 4 x 2
#>   nev   datum
#>   <chr> <chr>
#> 1 Anna  2023-01-01
#> 2 Béla  2023-02-02
#> 3 Csaba 2023-03-03
#> 4 Dóra  2023-04-04
```

A `datum` oszlopban található dátumokat szeretnénk három külön oszlopra bontani, amelyeket `ev`, `honap` és `nap` néven szeretnénk elnevezni. A `separate()` függvény első argumentuma az adattábla neve, a második argumentum a szétválasztani kívánt oszlop neve, a harmadik argumentum pedig az új oszlopok neveit tartalmazó karakterlánc. Az új oszlopok nevét egy karakterláncban kell megadni, amelyet a `sep` argumentumban megadott karakterrel választunk el egymástól.

```
# a dátum oszlop szétválasztása
df <- df |>
    separate(datum, into = c("ev", "honap", "nap"), sep = "-")
df
#> # A tibble: 4 x 4
#>   nev   ev    honap nap
#>   <chr> <chr> <chr> <chr>
```

```
#> 1 Anna  2023  01    01
#> 2 Béla  2023  02    02
#> 3 Csaba 2023  03    03
#> 4 Dóra  2023  04    04
```

Az ellenkező irányú műveletet a `tidyr::unite()` függvény segítségével végezhetjük el, amely egyesít több oszlopot egy új oszlopba. A `unite()` függvény első argumentuma az új oszlop neve, a második argumentum pedig azoknak az oszlopoknak a neve, amelyeket egyesíteni szeretnénk. Az oszlopok nevét egy karakterláncban kell megadni, amelyet a `sep` argumentumban megadott karakterrel választunk el egymástól.

```
# a datum oszlop egyesítése
df <- df |>
    unite(datum, ev, honap, nap, sep = "-")
df
#> # A tibble: 4 x 2
#>   nev   datum
#>   <chr> <chr>
#> 1 Anna  2023-01-01
#> 2 Béla  2023-02-02
#> 3 Csaba 2023-03-03
#> 4 Dóra  2023-04-04
```

Izgalmasabb példa kiindulópontja lehet a Google Űrlap típusú kérdőíves adatgyűjtő rendszerek kimeneti Excel/CSV állománya. A kérdőíves adatgyűjtés során gyakran találkozunk többszörös választást tartalmazó kérdésekkel. Tegyük fel, hogy egy kérdőívben a válaszadók több hobbit is megadhatnak, amelyeket egy oszlopban tárolunk, vesszővel elválasztva. A `hobbi` oszlopban található hobbi értékeket szeretnénk külön oszlopokba szétbontani, ahol az új oszlopok nevei a hobbi egyes esetei lesznek. Hozzuk létre a következő adattáblát, amelyben a válaszadók neve és hobbi értéke található:

```
# Fentit megadhatjuk a `tribble()` függvény segítségével is
df <- tribble(
  ~nev,            ~hobbi,
  "Anna", "olvasás, zene",
  "Béla",   "sport, zene",
  "Csaba",      "olvasás",
  "Dóra",              NA
)
df
```

```
#> # A tibble: 4 x 2
#>   nev   hobbi
#>   <chr> <chr>
#> 1 Anna  olvasás, zene
#> 2 Béla  sport, zene
#> 3 Csaba olvasás
#> 4 Dóra  <NA>
```

A fenti adattáblát úgy értelmezzük, hogy az `Anna` nevű válaszadó hobbiként jelölte az olvasást és a zenét, míg a `Béla` nevű válaszadó a sportot és a zenét. A `hobbi` oszlopban található hobbi értékeket szeretnénk külön oszlopokba szétbontani, ahol az új oszlopok nevei az egyes hobbiknak felelnek meg. A `tidyr::separate_rows()` függvény segítségével a `hobbi` oszlopot több sorra tudjuk bontani, amelyben az egyes hobbi értékek szerepelnek. A `separate_rows()` függvény első argumentuma az adattábla neve, a második argumentum a szétválasztani kívánt oszlop neve, a harmadik argumentum pedig a hobbi értékeket elválasztó karakter.

```
# oszlop szétválasztása több sorba
df_hosszu <- df |>
  separate_rows(hobbi, sep = ",\\s*") |>    # szétbontjuk sorokra
  mutate(van = TRUE)                        # új oszlop, jelzi a választ
print(df_hosszu, n=10)
#> # A tibble: 6 x 3
#>   nev   hobbi   van
#>   <chr> <chr>   <lgl>
#> 1 Anna  olvasás TRUE
#> 2 Anna  zene    TRUE
#> 3 Béla  sport   TRUE
#> 4 Béla  zene    TRUE
#> 5 Csaba olvasás TRUE
#> 6 Dóra  <NA>    TRUE
```

Most már nincs más teendőnk, mint a `pivot_wider()` függvény segítségével az új `df_hosszu` adattáblát széles formátumra alakítani. A `values_fill` argumentumban megadjuk, hogy a hiányzó értékek helyére `FALSE` értéket szeretnénk írni. Ezzel érjük el, hogy a válaszadók által megadott hobbi értékek helyén `TRUE` érték szerepeljen, míg a nem jelölt hobbiknál `FALSE` érték fog szerepelni.

```
# hosszú-széles átalakítás
df_szeles <- df_hosszu |>
  pivot_wider(names_from = hobbi,
```

```
              values_from = van,
              values_fill = FALSE) # hiányzó = FALSE

df_szeles
#> # A tibble: 4 x 5
#>   nev   olvasás zene  sport `NA`
#>   <chr> <lgl>   <lgl> <lgl> <lgl>
#> 1 Anna  TRUE    TRUE  FALSE FALSE
#> 2 Béla  FALSE   TRUE  TRUE  FALSE
#> 3 Csaba TRUE    FALSE FALSE FALSE
#> 4 Dóra  FALSE   FALSE FALSE TRUE
```

Összefoglalás

A haladó adatkezelési műveletek a beolvasást követő alapvető lépések után további, összetettebb feladatokat foglalnak magukban. Az adattáblák összekapcsolására akkor van szükség, amikor külön táblákban tároljuk a mért adatainkat. Ezeket a táblákat a közös azonosító alapján – rendszerint egy "id" oszlop segítségével – kapcsoljuk össze. A `left_join()` függvénnyel a bal oldali adattábla minden sora megmarad, míg a `full_join()` akkor hasznos, ha mindkét tábla teljes tartalmát szeretnénk megőrizni. A széles és hosszú formátum közötti átalakítás az ismételt méréseken alapuló adatoknál kiemelten fontos. A széles formátumban minden mérési időpont külön oszlopot kap, míg a hosszú formátumban ezek az értékek egymás alá kerülnek. A `pivot_longer()` segítségével alakíthatjuk át a széles formátumú adatokat hosszúvá, míg a `pivot_wider()` lehetővé teszi a visszaalakítást. Az oszlopok szétválasztása és egyesítése szintén gyakran előforduló feladat. A `separate()` függvénnyel egy összetett oszlop új oszlopokra bontható, míg az `unite()` segítségével ezeket újra egyesíthetjük. Különösen hasznos ez a művelet olyan adatforrások esetén, amelyek többszörös választást tartalmazó kérdéseit egyetlen cellában, vesszővel elválasztva tárolják. Ebben az esetben a `separate_rows()` függvénnyel sorokra bontjuk az értékeket, majd a `pivot_wider()` segítségével egy bináris, igaz/hamis értékeket tartalmazó, széles formátumú táblává alakítjuk, amely már jól használható további elemzésekhez.

Feladatok

1. Az adattáblák összekapcsolásánál a `left_join()` függvény helyett használhatjuk a `right_join()` függvényt is. Mi a különbség a két függvény között? Milyen esetben használjuk az `inner_join()` függvényt? Ha eltér a két összekapcsolandó adattábla azonosítójának neve, akkor hogyan tudjuk összekapcsolni őket?

2. A `{tidyr}` csomag `gapminder` adattáblája hosszú formátumban tárolja az egyes kontinensekre vonatkozó népességadatokat, különböző évekre vonatkozóan. Készítsünk egy széles formátumú adattáblát, amelyben az egyes évek – egy "Y" előtaggal – az oszlopok nevei lesznek! Végezzük el ennek a hosszú formátumra alakítását is!
3. Három személy 3 napos tréningen vesz részt, amelynek része egy speciális IQ teszt és kreativitás teszt kitöltése a tréning minden napján. Az adatokat a lenti táblázatnak megfelelően rögzítsük rövid formátumban. Alakítsuk át az adattáblát hosszú formátumúvá!

Id	1.nap, IQ	1.nap, KREAT	2.nap, IQ	2.nap, KREAT	3.nap, IQ	3.nap, KREAT
1	8	12	10	8	13	6
2	12	10	12	9	7	12
3	9	7	12	10	12	8

4. Három személy 3 napos tréningen vesz részt, amelynek része egy speciális IQ teszt és kreativitás teszt kitöltése a tréning minden napján. Az adatokat a lenti táblázatnak megfelelően rögzítsük hosszú formátumban. Alakítsuk át az adattáblát széles formátumúvá!

Id	Tréning	Teszt	Eredmény
1	1.nap	IQ	8
1	1.nap	KREAT	9
2	1.nap	IQ	12
2	1.nap	KREAT	5
3	1.nap	IQ	9
3	1.nap	KREAT	11
1	2.nap	IQ	11
1	2.nap	KREAT	12
2	2.nap	IQ	11
2	2.nap	KREAT	9
3	2.nap	IQ	11
3	2.nap	KREAT	10
1	3.nap	IQ	10
1	3.nap	KREAT	11
2	3.nap	IQ	9
2	3.nap	KREAT	11
3	3.nap	IQ	7
3	3.nap	KREAT	12

III. rész
Leíró statisztika

8. Mutatók és táblázatok

8.1. Az *Alap R* lehetőségei 🙂

> Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük az *Alap R* mutatószámoló és
> - táblázat készítő lehetőségeit.

A leíró statisztika célja az adatstruktúrába való elsődleges betekintés, az adatok eloszlásának felderítése. A leíró statisztika eszközei a statisztikai mérőszámok (mutatók), a táblázatok és a grafikonok. Ebben a fejezetben a mutatókkal és a táblázatokkal foglalkozunk, az ábrák készítése a következő fejezet témája lesz.

8.1.1. Mutatók

8.1.1.1. Egy változó, egy mutató

A statisztikai mérőszámok vagy mutatók a mintából számolt statisztikai függvények. A két legfontosabb a mintaátlag és a minta szórása. A `{MASS}` csomag `survey` adattáblája 237 egyetemista testmagasságát tartalmazza a `Height` oszlopában. Számoljuk ki ezt a két fontos mutatót:

```
data(survey, package = "MASS") # survey betöltése
mean(survey$Height, na.rm = T) # testmagasság átlaga
#> [1] 172.3809
sd(survey$Height, na.rm = T)   # testmagasság szórása
#> [1] 9.847528
```

Az `na.rm=T` argumentum megadására szükség van, mivel a `Height` változó tartalmaz hiányzó értékeket, csak így kapjuk meg a mintaátlagot és a minta szórását.

Természetesen további mutatók is számolhatók:

```
median(survey$Height, na.rm = T)            # medián
#> [1] 171
var(survey$Height, na.rm = T)               # variancia
#> [1] 96.9738
min(survey$Height, na.rm = T)               # minimum
#> [1] 150
```

```r
max(survey$Height, na.rm = T)                # maximum
#> [1] 200
range(survey$Height, na.rm = T)              # minimum és maximum
#> [1] 150 200
diff(range(survey$Height, na.rm = T))        # terjedelem
#> [1] 50
IQR(survey$Height, na.rm = T)                # interkvartilis-eltérés
#> [1] 15
quantile(survey$Height, na.rm = T)           # kvantilisek
#>   0%  25%  50%  75% 100%
#>  150  165  171  180  200
moments::skewness(survey$Height, na.rm = T) # ferdeség
#> [1] 0.2173972
moments::kurtosis(survey$Height, na.rm = T) # csúcsosság
#> [1] 2.587255
```

8.1.1.2. Több változó, egy mutató

Sokszor, egyszerre több numerikus változó statisztikai mutatóit szeretnénk meghatározni, ehhez az `apply(MARGIN=2)` vagy az `sapply()` függvényt használhatjuk. Az `apply()` általánosabb, így a `MARGIN=2` segítségével tudjuk az oszloponkénti függvényvégrehajtására utasítani, míg az `sapply()` esetében épp ez a működés lényege. Most három numerikus változó (kézméret, testmagasság és életkor) legfontosabb statisztikai mutatóit íratjuk ki:

```r
# mutatók 3 változóra: apply()
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = mean, na.rm = T)
#>    Wr.Hnd    Height       Age
#>  18.66907 172.38086  20.37451
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = sd, na.rm = T)
#>   Wr.Hnd   Height      Age
#> 1.878981 9.847528 6.474335
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = range, na.rm = T)
#>      Wr.Hnd Height   Age
#> [1,]   13.0    150 16.75
#> [2,]   23.2    200 73.00
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = quantile, na.rm = T)
#>      Wr.Hnd Height    Age
#> 0%     13.0    150 16.750
#> 25%    17.5    165 17.667
```

```
#> 50%    18.5    171 18.583
#> 75%    19.8    180 20.167
#> 100%   23.2    200 73.000
```

```
# mutatók 3 változóra: sapply()
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = mean, na.rm = T)
#>    Wr.Hnd    Height       Age
#>  18.66907 172.38086  20.37451
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = sd, na.rm = T)
#>   Wr.Hnd   Height      Age
#> 1.878981 9.847528 6.474335
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = range, na.rm = T)
#>      Wr.Hnd Height   Age
#> [1,]   13.0    150 16.75
#> [2,]   23.2    200 73.00
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = quantile, na.rm = T)
#>      Wr.Hnd Height    Age
#> 0%     13.0    150 16.750
#> 25%    17.5    165 17.667
#> 50%    18.5    171 18.583
#> 75%    19.8    180 20.167
#> 100%   23.2    200 73.000
```

8.1.1.3. Egy változó, több mutató

Amennyiben egyszerre több statisztikai mutatóra van szükségünk, akkor az *Alap R* lehetőségei közül a `summary()` függvény az első lehetőség.

```
summary(survey$Wr.Hnd)  # kézméret mutatói
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#>   13.00   17.50   18.50   18.67   19.80   23.20       1
summary(survey$Height)  # testmagasság mutatói
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
#>   150.0   165.0   171.0   172.4   180.0   200.0      28
summary(survey$Age)  # életkor mutatói
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   16.75   17.67   18.58   20.37   20.17   73.00
```

8.1.1.4. Mutatók csoportokra

Nagyon fontos lehetőség a mutatók meghatározása különböző csoportokban. Csoportok alatt a faktor által meghatározott, adott faktorszinthez tartozó mintaelemeket értjük. Például a `survey` adattábla nem (`Sex`) faktora két csoportra osztja a 237 elemű mintát.

```
table(survey$Sex, useNA = "ifany")  # nem faktor eloszlása
#>
#> Female   Male   <NA>
#>    118    118      1
```

A nem szerint meghatározott két csoport azonos elemszámú, 118 személy tartozik a nők csoportjába, és 118 a férfiak csoportjába is. Egy másik faktor, a dohányzási szokás (`Smoke`) már nem mutat egyenletes eloszlást:

```
table(survey$Smoke, useNA = "ifany")  # dohányzási szokás faktor eloszlása
#>
#> Heavy Never Occas Regul  <NA>
#>    11   189    19    17     1
```

Az erős dohányosok mindössze 11-en vannak, míg a legtöbben a nem dohányzók táborába tartoznak (189 fő).

A faktor által meghatározott csoportok nagyon fontosak lehetnek az adatelemzés során. Például kíváncsiak lehetünk a nem faktor két csoportjában (nők és férfiak) a testmagasság átlagára. Az *Alap R* az ilyen jellegű kérdések megválaszolására 3 függvényt is nyújt számunkra: `tapply()`, `aggregate()` és `by()`.

A `tapply()` egyetlen numerikus változó egy vagy több faktor csoportjaiban képes statisztikai mutató meghatározására. Fontos, hogy a statisztikai mutató egyetlen értékkel térjen vissza (például `mean()` vagy `sd()`), mert a táblázatos elrendezés az outputban csak így lehetséges, tehát a `range()` és a `quantile()` nem használható.

```
# mintaátlag csoportokra
tapply(survey$Wr.Hnd, INDEX = survey$Sex, FUN = mean, na.rm = T)
#>   Female     Male
#> 17.59576 19.74188
tapply(survey$Wr.Hnd, INDEX = survey[, c("Sex", "Smoke")], FUN = mean, na.rm = T)
#>         Smoke
#> Sex         Heavy    Never    Occas  Regul
#>   Female 17.90000 17.65657 16.82222 17.480
```

```
#>   Male   20.31667 19.60568 19.83000 20.275
tapply(survey$Wr.Hnd, INDEX = survey[, c("Sex", "Smoke", "Exer")], FUN = mean,
    na.rm = T)
#> , , Exer = Freq
#>
#>         Smoke
#> Sex         Heavy    Never    Occas    Regul
#>   Female 17.73333 17.53333 15.92000 19.55000
#>   Male   18.92500 19.75319 20.27143 20.85714
#>
#> , , Exer = None
#>
#>         Smoke
#> Sex      Heavy   Never Occas Regul
#>   Female    NA 17.5900 18.50    NA
#>   Male    23.2 19.2875 17.95  19.5
#>
#> , , Exer = Some
#>
#>         Smoke
#> Sex      Heavy    Never    Occas Regul
#>   Female 18.15 17.76600 17.76667 16.10
#>   Male   23.00 19.47273 20.50000 19.45
```

A `tapply()` függvényt tipikusan egy vagy két faktor esetén használjuk, a táblázatos output kényelmes áttekintést ad az egyes csoportok statisztikai mutatóiról. A legnagyobb szabadságot az `aggregate()` és a `by()` nyújtja számunkra, ezek használatát érdemes elsajátítani. Paraméterezésük megegyezik (az argumentumok neve nem), csak az outputban térnek el. Az `aggregate()` visszatérési értéke egy *adattábla* típusú objektum, amelyet később kényelmesen felhasználhatunk, a `by()` egy egyszerűbb szöveges listával tér vissza.

```
# mintaátlag csoportokra
aggregate(survey[, "Wr.Hnd"], by = survey[, "Sex", drop = F], FUN = mean, na.rm = T)
#>      Sex        x
#> 1 Female 17.59576
#> 2   Male 19.74188
by(survey[, "Wr.Hnd"], INDICES = survey[, "Sex", drop = F], FUN = mean, na.rm = T)
#> Sex: Female
#> [1] 17.59576
#> ------------------------------------------------------------
```

```
#> Sex: Male
#> [1] 19.74188
aggregate(survey[, "Wr.Hnd"], by = survey[, c("Sex", "Smoke")], FUN = mean,
    na.rm = T)
#>      Sex Smoke        x
#> 1 Female Heavy 17.90000
#> 2   Male Heavy 20.31667
#> 3 Female Never 17.65657
#> 4   Male Never 19.60568
#> 5 Female Occas 16.82222
#> 6   Male Occas 19.83000
#> 7 Female Regul 17.48000
#> 8   Male Regul 20.27500
by(survey[, "Wr.Hnd"], INDICES = survey[, c("Sex", "Smoke")], FUN = mean, na.rm = T)
#> Sex: Female
#> Smoke: Heavy
#> [1] 17.9
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Heavy
#> [1] 20.31667
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Never
#> [1] 17.65657
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Never
#> [1] 19.60568
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Occas
#> [1] 16.82222
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Occas
#> [1] 19.83
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Regul
```

```
#> [1] 17.48
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Regul
#> [1] 20.275
aggregate(survey[, "Wr.Hnd"], by = survey[, c("Sex", "Smoke", "Exer")], FUN = mean,
    na.rm = T)
#>       Sex Smoke Exer        x
#> 1  Female Heavy Freq 17.73333
#> 2    Male Heavy Freq 18.92500
#> 3  Female Never Freq 17.53333
#> 4    Male Never Freq 19.75319
#> 5  Female Occas Freq 15.92000
#> 6    Male Occas Freq 20.27143
#> 7  Female Regul Freq 19.55000
#> 8    Male Regul Freq 20.85714
#> 9    Male Heavy None 23.20000
#> 10 Female Never None 17.59000
#> 11   Male Never None 19.28750
#> 12 Female Occas None 18.50000
#> 13   Male Occas None 17.95000
#> 14   Male Regul None 19.50000
#> 15 Female Heavy Some 18.15000
#> 16   Male Heavy Some 23.00000
#> 17 Female Never Some 17.76600
#> 18   Male Never Some 19.47273
#> 19 Female Occas Some 17.76667
#> 20   Male Occas Some 20.50000
#> 21 Female Regul Some 16.10000
#> 22   Male Regul Some 19.45000
by(survey[, "Wr.Hnd"], INDICES = survey[, c("Sex", "Smoke", "Exer")], FUN = mean,
    na.rm = T)
#> Sex: Female
#> Smoke: Heavy
#> Exer: Freq
#> [1] 17.73333
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Heavy
#> Exer: Freq
```

```
#> [1] 18.925
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Never
#> Exer: Freq
#> [1] 17.53333
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Never
#> Exer: Freq
#> [1] 19.75319
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Occas
#> Exer: Freq
#> [1] 15.92
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Occas
#> Exer: Freq
#> [1] 20.27143
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Regul
#> Exer: Freq
#> [1] 19.55
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Regul
#> Exer: Freq
#> [1] 20.85714
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Heavy
#> Exer: None
#> [1] NA
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Heavy
#> Exer: None
```

```
#> [1] 23.2
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Never
#> Exer: None
#> [1] 17.59
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Never
#> Exer: None
#> [1] 19.2875
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Occas
#> Exer: None
#> [1] 18.5
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Occas
#> Exer: None
#> [1] 17.95
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Regul
#> Exer: None
#> [1] NA
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Regul
#> Exer: None
#> [1] 19.5
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Heavy
#> Exer: Some
#> [1] 18.15
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Heavy
#> Exer: Some
```

```
#> [1] 23
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Never
#> Exer: Some
#> [1] 17.766
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Never
#> Exer: Some
#> [1] 19.47273
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Occas
#> Exer: Some
#> [1] 17.76667
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Occas
#> Exer: Some
#> [1] 20.5
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Regul
#> Exer: Some
#> [1] 16.1
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Regul
#> Exer: Some
#> [1] 19.45
```

Az `aggregate()` legnagyobb előnye, hogy több numerikus változót megadhatunk az első paraméterükben, a `by()` esetében pedig olyan függvényeket is alkalmazhatunk az egyes csoportokra, amelyek nem egyetlen értékkel térnek vissza.

```
# mintaátlag csoportokra
aggregate(survey[, c("Wr.Hnd", "Height", "Age")], by = survey[, "Sex", drop = F],
    FUN = mean, na.rm = T)
#>      Sex   Wr.Hnd   Height      Age
```

```
#> 1 Female 17.59576 165.6867 20.40753
#> 2   Male 19.74188 178.8260 20.33196
aggregate(survey[, c("Wr.Hnd", "Height", "Age")], by = survey[, c("Sex", "Smoke")],
    FUN = mean, na.rm = T)
#>      Sex Smoke   Wr.Hnd   Height      Age
#> 1 Female Heavy 17.90000 166.9360 22.73360
#> 2   Male Heavy 20.31667 180.6080 20.27767
#> 3 Female Never 17.65657 165.6584 20.02952
#> 4   Male Never 19.60568 178.2800 20.50565
#> 5 Female Occas 16.82222 167.4150 21.90733
#> 6   Male Occas 19.83000 178.6425 18.93350
#> 7 Female Regul 17.48000 159.8067 22.86660
#> 8   Male Regul 20.27500 182.2200 20.40283
aggregate(survey[, c("Wr.Hnd", "Height", "Age")], by = survey[, c("Sex", "Smoke",
    "Exer")], FUN = mean, na.rm = T)
#>       Sex Smoke Exer   Wr.Hnd   Height      Age
#> 1  Female Heavy Freq 17.73333 167.1667 25.86133
#> 2    Male Heavy Freq 18.92500 179.6667 20.45800
#> 3  Female Never Freq 17.53333 167.1683 19.73297
#> 4    Male Never Freq 19.75319 180.3112 20.46987
#> 5  Female Occas Freq 15.92000 166.8640 19.74980
#> 6    Male Occas Freq 20.27143 178.3767 19.26200
#> 7  Female Regul Freq 19.55000 167.0000 19.79150
#> 8    Male Regul Freq 20.85714 182.5971 21.98814
#> 9    Male Heavy None 23.20000 171.0000 20.91700
#> 10 Female Never None 17.59000 162.9956 20.40010
#> 11   Male Never None 19.28750 173.6000 22.14588
#> 12 Female Occas None 18.50000      NaN 41.58300
#> 13   Male Occas None 17.95000 176.0000 17.91700
#> 14   Male Regul None 19.50000 177.8000 17.58300
#> 15 Female Heavy Some 18.15000 166.5900 18.04200
#> 16   Male Heavy Some 23.00000 193.0400 18.91700
#> 17 Female Never Some 17.76600 164.9171 20.18670
#> 18   Male Never Some 19.47273 176.6703 20.16918
#> 19 Female Occas Some 17.76667 168.3333 18.94467
#> 20   Male Occas Some 20.50000 182.8800 18.66700
#> 21 Female Regul Some 16.10000 156.2100 24.91667
#> 22   Male Regul Some 19.45000 182.8133 18.33350
# normalitás vizsgálat
by(survey[, "Wr.Hnd"], INDICES = survey[, c("Sex", "Smoke")], FUN = shapiro.test)
```

```
#> Sex: Female
#> Smoke: Heavy
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.86155, p-value = 0.2339
#>
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Heavy
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.83787, p-value = 0.1252
#>
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Never
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.9461, p-value = 0.0004983
#>
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Never
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.98671, p-value = 0.5119
#>
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Occas
#>
#>  Shapiro-Wilk normality test
```

```
#>
#> data:  dd[x, ]
#> W = 0.97224, p-value = 0.9131
#>
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Occas
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.91854, p-value = 0.3449
#>
#> ------------------------------------------------------
#> Sex: Female
#> Smoke: Regul
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.9035, p-value = 0.4295
#>
#> ------------------------------------------------------
#> Sex: Male
#> Smoke: Regul
#>
#>  Shapiro-Wilk normality test
#>
#> data:  dd[x, ]
#> W = 0.92621, p-value = 0.3417
# összegzés
by(survey[, c("Wr.Hnd", "Height", "Age")], INDICES = survey[, "Sex", drop = F],
    FUN = summary, na.rm = T)
#> Sex: Female
#>      Wr.Hnd         Height           Age
#>  Min.   :13.0   Min.   :150.0   Min.   :16.92
#>  1st Qu.:17.0   1st Qu.:162.6   1st Qu.:17.50
#>  Median :17.5   Median :166.8   Median :18.42
#>  Mean   :17.6   Mean   :165.7   Mean   :20.41
#>  3rd Qu.:18.5   3rd Qu.:170.0   3rd Qu.:19.98
```

```
#>  Max.   :20.8   Max.   :180.3   Max.   :73.00
#>                 NA's   :16
#> ------------------------------------------------------------
#> Sex: Male
#>      Wr.Hnd          Height          Age
#>  Min.   :14.00   Min.   :154.9   Min.   :16.75
#>  1st Qu.:18.50   1st Qu.:172.8   1st Qu.:17.92
#>  Median :19.50   Median :180.0   Median :18.88
#>  Mean   :19.74   Mean   :178.8   Mean   :20.33
#>  3rd Qu.:21.00   3rd Qu.:185.0   3rd Qu.:20.29
#>  Max.   :23.20   Max.   :200.0   Max.   :70.42
#>  NA's   :1       NA's   :12
```

8.1.2. Táblázatok

8.1.2.1. Abszolút gyakoriság

Az eddig látott mutatók a numerikus változók leíró statisztikai elemzésére szolgáltak. A gyakorisági táblázatokat többnyire a faktor típusú változók jellemzésére használjuk. Az *Alap R* korábban megismert függvényei, a `table()`, `xtabs()` és `ftable()` már biztosítják számunkra a nyers, ún. abszolút gyakorisági táblázatok kiírását.

Egydimenziós táblázatot, az ún. gyakorisági táblázatot a `summary()` függvénnyel is létrehozhatunk. Ha a hívásban faktor argumentumot adunk meg, akkor gyakorisági táblázatot kapunk, amely az egyes faktorszintek mintabeli előfordulási számát adja:

```
summary(survey$W.Hnd)
#>  Left Right  NA's
#>    18   218     1
```

Láthatjuk, hogy a 237 megkérdezett hallgatóból 18 balkezes, 218 jobbkezes, egy diáknak pedig nem jegyezték fel a kezességét. Hasonló eredményt kapunk, ha a táblázatok létrehozására szánt `table()` függvényt használjuk, de itt a hiányzó értékek megjelenítéséről már külön gondoskodnunk kell:

```
table(survey$W.Hnd, useNA = "ifany")  # kezesség gyakorisági táblázata (1D)
#>
#>  Left Right  <NA>
#>    18   218     1
```

```
xtabs(~W.Hnd, data = survey, addNA = T)
#> W.Hnd
#>  Left Right  <NA>
#>    18   218     1
```

A már korábban megismert `apply()` vagy `sapply()` segítségével több faktor gyakorisági táblázatát is kiírhatjuk:

```
sapply(survey[, c("Sex", "W.Hnd", "Exer", "Smoke")], FUN = table, useNA = "ifany")
#> $Sex
#>
#> Female   Male   <NA>
#>    118    118      1
#>
#> $W.Hnd
#>
#>  Left Right  <NA>
#>    18   218     1
#>
#> $Exer
#>
#> Freq None Some
#>  115   24   98
#>
#> $Smoke
#>
#> Heavy Never Occas Regul  <NA>
#>    11   189    19    17     1
```

A fenti outputból kiemeljük a `Smoke` változót, amely a hallgatók dohányzási szokását tartalmazza. A változó 4 szintű faktor, melynek értékei: `Never`-nem dohányzik, `Occas`-ritkán dohányzik, `Regul`-rendszeresen dohányzik, `Heavy`-sokat dohányzik. Láthatjuk, hogy a 237 megkérdezett hallgatóból 189 diák nem dohányzik és csak 11 erős dohányos.

A `table()` függvényt numerikus argumentum esetén is használhatjuk, ekkor a különböző számok előfordulási gyakoriságát kapjuk. A folytonos kvantitatív változóink jellemzően minden mérésnél más és más értéket adnak, így a legtöbbször a `table()` hívásnak nincs értelme folytonos numerikus vektor esetén. Azonban diszkrét numerikus változók esetén hasznos lehet a gyakorisági táblázat megjelenítése, mert ezek értékei sokszor ismétlődnek, de ez természetesen a változó jellegétől is nagy mértékben függ. Most a diákok pulzusára (`Pulse`) hívjuk meg a `table()` függvényt:

```
# gyakorisági táblázat diszkrét numerikus változóra
table(survey$Pulse, useNA = "ifany")
#>
#>   35   40   48   50   54   55   56   59   60   61   62   63   64   65
#>    1    1    2    2    1    1    1    1   12    1    4    1    9    6
#>   66   67   68   69   70   71   72   73   74   75   76   78   79   80
#>    6    1   16    1   13    2   14    1    5    5   13    4    3   18
#>   81   83   84   85   86   87   88   89   90   92   96   97   98  100
#>    1    4    5    4    3    2    4    1    8    6    3    1    1    2
#>  104 <NA>
#>    2   45
xtabs(~Pulse, data = survey, addNA = T)
#> Pulse
#>   35   40   48   50   54   55   56   59   60   61   62   63   64   65
#>    1    1    2    2    1    1    1    1   12    1    4    1    9    6
#>   66   67   68   69   70   71   72   73   74   75   76   78   79   80
#>    6    1   16    1   13    2   14    1    5    5   13    4    3   18
#>   81   83   84   85   86   87   88   89   90   92   96   97   98  100
#>    1    4    5    4    3    2    4    1    8    6    3    1    1    2
#>  104 <NA>
#>    2   45
```

Leolvashatjuk, hogy leggyakoribb pulzus a 80, hiszen az 18-szor fordul elő, valamint 45 személynek nem ismerjük a pulzusát. A fenti táblázatot áttekinthetőbbé tehetjük, ha az előfordulási értékek szerint rendezzük a cellákat. A rendezésre használjuk a `sort(decreasing=T)` függvényt:

```
# rendezett gyakorisági táblázat
sort(table(survey$Pulse, useNA = "ifany"), decreasing = T)
#>
#> <NA>   80   68   72   70   76   60   64   90   65   66   92   74   75
#>   45   18   16   14   13   13   12    9    8    6    6    6    5    5
#>   84   62   78   83   85   88   79   86   96   48   50   71   87  100
#>    5    4    4    4    4    4    3    3    3    2    2    2    2    2
#>  104   35   40   54   55   56   59   61   63   67   69   73   81   89
#>    2    1    1    1    1    1    1    1    1    1    1    1    1    1
#>   97   98
#>    1    1
```

Láthatjuk, hogy a fenti outputból már könnyen kiolvashatók a legnagyobb és legkisebb gyakoriságú értékek.

Többdimenziós táblázatokat a szokásos módon, több faktor felsorolásával adhatunk meg:

```
# kétdimenziós táblázat
# kezesség gyakorisági táblázata (2D)
table(survey$Sex, survey$W.Hnd, useNA = "ifany")
#>
#>          Left Right <NA>
#>   Female    7   110    1
#>   Male     10   108    0
#>   <NA>      1     0    0
```

```
# háromdimenziós táblázat
# kezesség gyakorisági táblázata (3D)
ftable(table(survey$Sex, survey$W.Hnd, survey$Exer, useNA = "ifany"))
#>               Freq None Some
#>
#> Female Left      3    1    3
#>        Right    45   10   55
#>        NA        1    0    0
#> Male   Left      3    2    5
#>        Right    62   11   35
#>        NA        0    0    0
#> NA     Left      1    0    0
#>        Right     0    0    0
#>        NA        0    0    0
```

8.1.2.2. Táblázat összesített adatokból

Egydimenziós táblázatokat összesített gyakorisági adatok alapján is létrehozhatunk a `c()` függvény és az `as.table()` segítségével. Tegyük fel, hogy rendelkezésre áll az az információ is a fenti 237 hallgatóról, hogy közülük 137 fővárosi és 100 vidéki.

```
# gyakorisági táblázat összesített adatok alapján
lakhely <- as.table(c(főváros = 137, vidék = 100))
lakhely
#> főváros   vidék
#>     137     100
as.data.frame(lakhely)
#>      Var1 Freq
```

```
#> 1 főváros  137
#> 2   vidék  100
```

A következő táblázat 3888 szülés előtt lévő hölgy koffein fogyasztásáról és családi állapotáról tartalmaz gyakorisági adatokat. Készítsünk egy mátrixot, majd táblázatot a következő adatokból.

Családi állapot	0	1-150	151-300	>300
Házas	652	1537	598	242
Elvált, különváltan él, özvegy	36	46	38	21
Egyedül él	218	327	106	67

A `matrix()` függvényt használjuk, az adatokat pedig sor folytonosan adjuk meg az első argumentumban. A könnyebb áttekinthetőség kedvéért, adjunk nevet a soroknak és az oszlopoknak.

```
# mátrix készítése gyakorisági adatokból
m <- matrix(c(652, 1537, 598, 242, 36, 46, 38, 21, 218, 327, 106, 67), nrow = 3,
    byrow = T)
rownames(m) <- c("Házas", "Házas.volt", "Egyedül.él")
colnames(m) <- c("0", "1-150", "151-300", ">300")
m
#>              0 1-150 151-300 >300
#> Házas      652  1537     598  242
#> Házas.volt  36    46      38   21
#> Egyedül.él 218   327     106   67
```

A példában szereplő gyakorisági adatokat sikeresen rögzítettük egy numerikus mátrixba. Azonban akkor lesz belőle R-beli táblázat, ha az `as.table()` függvénnyel átalakítjuk a mátrixunkat.

```
# mátrix átalakítása táblázattá
koff.csalad <- as.table(m)
koff.csalad
#>               0 1-150 151-300 >300
#> Házas       652  1537     598  242
#> Házas.volt   36    46      38   21
#> Egyedül.él  218   327     106   67
```

```
# adattábla készítése táblázatból
as.data.frame((koff.csalad))
#>          Var1    Var2 Freq
#> 1       Házas       0  652
#> 2  Házas.volt       0   36
#> 3  Egyedül.él       0  218
#> 4       Házas   1-150 1537
#> 5  Házas.volt   1-150   46
#> 6  Egyedül.él   1-150  327
#> 7       Házas 151-300  598
#> 8  Házas.volt 151-300   38
#> 9  Egyedül.él 151-300  106
#> 10      Házas    >300  242
#> 11 Házas.volt    >300   21
#> 12 Egyedül.él    >300   67
```

8.1.2.3. Relatív gyakoriság

Az (abszolút) gyakorisági táblázatok mellett relatív gyakorisági, illetve százalékos relatív gyakorisági táblázatra is szükségünk lehet. Ezek létrehozásához a `prob.table()` függvényt használhatjuk.

```
# százalékos relatív gyakorisági táblázat
kezesseg <- table(survey$W.Hnd)
prop.table(kezesseg)
#>
#>       Left      Right
#> 0.07627119 0.92372881
100 * prop.table(kezesseg)
#>
#>      Left     Right
#>  7.627119 92.372881
```

A könnyebb értelmezhetőség kedvéért használjuk a `round(digits=1)` függvényt:

```
# százalékos relatív gyakorisági táblázat, kerekítve
round(100 * prop.table(kezesseg), digits = 1)
#>
#>  Left Right
#>   7.6  92.4
```

Kereszttáblák (2D táblázatok) esetén a teljes, a soronkénti és az oszloponkénti eloszlás számítására is lehetőséget ad a `prob.table()` függvény.

A soronkénti relatív gyakorisághoz a `margin=1`, az oszloponkéntihez a `margin=2` argumentumot használjuk. A példában százalékos relatív gyakorisági táblázatot használunk.

```
# kétdimenziós táblázat
tab <- table(survey$Sex, survey$W.Hnd, useNA = "ifany")
round(100 * prop.table(tab), digits = 1)  # teljes
#>
#>          Left Right <NA>
#>   Female  3.0  46.4  0.4
#>   Male    4.2  45.6  0.0
#>   <NA>    0.4   0.0  0.0
round(100 * prop.table(tab, margin = 1), digits = 1)  # soronkénti
#>
#>           Left Right  <NA>
#>   Female   5.9  93.2   0.8
#>   Male     8.5  91.5   0.0
#>   <NA>   100.0   0.0   0.0
round(100 * prop.table(tab, margin = 2), digits = 1)  # oszloponkénti
#>
#>           Left Right  <NA>
#>   Female  38.9  50.5 100.0
#>   Male    55.6  49.5   0.0
#>   <NA>     5.6   0.0   0.0
```

Kétdimenziós táblázatok esetén, a marginális eloszlások számítására is van lehetőségünk. Használhatjuk az `apply(FUN=sum)` függvényt, de a direkt erre a célra létrehozott `margin.table()` függvényt is.

```
# soronkénti összesítés
apply(koff.csalad, MARGIN = 1, FUN = sum)
#>      Házas Házas.volt Egyedül.él
#>       3029        141        718
# oszloponkénti összesítés
apply(koff.csalad, MARGIN = 2, FUN = sum)
#>       0   1-150 151-300    >300
#>     906    1910     742     330
# soronkénti összesítés
margin.table(koff.csalad, margin = 1)
```

```
#>      Házas Házas.volt Egyedül.él
#>       3029        141        718
# oszloponkénti összesítés
margin.table(koff.csalad, margin = 2)
#>       0   1-150 151-300    >300
#>     906    1910     742     330
```

8.1.2.4. Kumulált gyakorisági táblázatok

A relatív gyakorisági táblázatok mellett a kumulált relatív gyakorisági táblázatokat megjelenítését is kérhetjük egydimenziós esetben. Természetesen, ekkor a változónk legalább ordinális skálán mért. A kumulált értékek meghatározására a `cumsum()` függvényt használjuk:

```
# rendezett faktor készítése
survey$Smoke <- ordered(survey$Smoke,
                        levels=c("Never", "Occas", "Regul", "Heavy"))
# gyakorisági táblázat
dohanyzas <- table(survey$Smoke)
# kumulált gyakorisági táblázat
cumsum(round(100*prop.table(dohanyzas), digits=0))
#> Never Occas Regul Heavy
#>    80    88    95   100
```

Összefoglalás

Az *Alap R* mutatószámoló függvényeiből (például `mean()`, `sd()`, `median()`, `min()`, `max()`) megismertük, hogyan számíthatók ki egy változóra az olyan alapvető statisztikai mérőszámok, mint az átlag, szórás, medián, minimum, maximum. Ezeket a mutatókat akár több változóra is egyszerre meghatározhatjuk az `apply()` vagy `sapply()` függvények segítségével. Ha pedig egyetlen változóra több mutatót szeretnénk látni, akkor a `summary()` függvény a legegyszerűbb választás. Mutatókat csoportonként is számíthatunk, ekkor a `tapply()`, `aggregate()` és `by()` függvények kínálnak kényelmes megoldásokat, az utóbbi két függvény akár több változóra és több csoportra is alkalmazható. Fontos, hogy az `aggregate()` adattáblát, míg a `by()` egy listát ad vissza. Táblázatok készítésére az `Alap R` több függvényt is kínál, például a `summary()` és `table()`, vagy az `xtabs()` és `ftable()`. Ezek a táblák lehetnek abszolút vagy relatív gyakorisági táblák, utóbbiakhoz a `prop.table()` függvény szükséges. A táblákban a rendezés (`sort()`) és összesítés (`margin.table()`, `apply()`) is lehetséges. A kumulált relatív gyakoriságok rendezett faktorváltozók esetén hasznosak. Ezek a `cumsum()` függvénnyel készíthetők el.

🎯 Feladatok

1. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` változójára!
2. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` és `Sex` változójára!
3. Határozza meg a pulzus átlagát a `survey` adattábla `Sex` és `Smoke` változói szerint!

8.2. A *Tidyverse R* lehetőségei 😉

ℹ Miről lesz szó? Ebben a fejezetben

- áttekintjük a *Tidyverse R* mutatószámoló és
- táblázat készítő lehetőségeit.

Az előző fejezetben bemutattuk az *Alap R* leíró statisztikai eszközeit, a mutatók és táblázatok esetében. Ezek a műveletek a *Tidyverse R* csomagjaival is kényelmesen elvégezhetők, és mint tudjuk, ez a kód olvashatóságát is jelentősen megkönnyíti.

8.2.1. Mutatók

8.2.1.1. Egy változó, egy mutató

A statisztikai mutatók számításához a `summarise()` és `summarise(across())` függvényeket használjuk. A `{MASS}` csomag `survey` adattáblájában például a `Height` oszlop tartalmazza a testmagasságokat. Ezt használjuk az alábbi példában a `summarise()` bemutatására.

```
library(tidyverse)
data(survey, package = "MASS")

# mintaátlag és szórás
survey |> summarise(atlag = mean(Height, na.rm = TRUE),
                    szoras = sd(Height, na.rm = TRUE))
#>      atlag   szoras
#> 1 172.3809 9.847528
```

További mutatók is kiszámíthatók:

```
# a szokásos leíró statisztikai mutatók
survey |> summarise(
  median = median(Height, na.rm = TRUE),
  variancia = var(Height, na.rm = TRUE),
  min = min(Height, na.rm = TRUE),
  max = max(Height, na.rm = TRUE),
  terjedelem = diff(range(Height, na.rm = TRUE)),
  IQR = IQR(Height, na.rm = TRUE),
  Q = list(quantile(Height, na.rm = TRUE)),
  ferdeseg = moments::skewness(Height, na.rm = TRUE),
  csucsossag = moments::kurtosis(Height, na.rm = TRUE)
)
#>   median variancia min max terjedelem IQR                       Q
#> 1    171   96.9738 150 200         50  15 150, 165, 171, 180, 200
#>    ferdeseg csucsossag
#> 1 0.2173972   2.587255
```

8.2.1.2. Több változó, egy mutató

A `summarise(across(...))` függvénnyel egyszerre több oszlopra is alkalmazhatjuk ugyanazt a mutatót:

```
# átlag és szórás több változóra
survey |>
    summarise(across(c(Wr.Hnd, Height, Age), \(x) mean(x, na.rm = TRUE)))
#>     Wr.Hnd   Height      Age
#> 1 18.66907 172.3809 20.37451
survey |>
    summarise(across(c(Wr.Hnd, Height, Age), \(x) sd(x, na.rm = TRUE)))
#>     Wr.Hnd   Height      Age
#> 1 1.878981 9.847528 6.474335
```

8.2.1.3. Egy változó, több mutató

Az *Alap R* `summary()` függvénye helyett építsünk saját összesítő függvényt, ez a *Tidyverse R* filozófiához jobban illik:

```
# saját összesítő függvény
survey |>
    summarise(across(Wr.Hnd, list(mean = ~mean(.x, na.rm = TRUE), sd = ~sd(.x,
        na.rm = TRUE), median = ~median(.x, na.rm = TRUE), Na = ~sum(is.na(.x)))))
#>   Wr.Hnd_mean Wr.Hnd_sd Wr.Hnd_median Wr.Hnd_Na
#> 1    18.66907  1.878981          18.5         1
```

8.2.1.4. Mutatók csoportokra

A `group_by()` segítségével csoportok szerinti mutatókat számolhatunk:

```
# mintaátlag a nem csoportjaira
survey |> group_by(Sex) |>
  summarise(atlag = mean(Height, na.rm = TRUE))
#> # A tibble: 3 x 2
#>   Sex    atlag
#>   <fct>  <dbl>
#> 1 Female  166.
#> 2 Male    179.
#> 3 <NA>    172
# mintaátlag a nem és a dohányzási szokás csoportjaira
survey |> group_by(Sex, Smoke) |>
  summarise(atlag = mean(Wr.Hnd, na.rm = TRUE), .groups = "drop")
#> # A tibble: 10 x 3
#>   Sex    Smoke atlag
#>   <fct>  <fct> <dbl>
#> 1 Female Heavy  17.9
#> 2 Female Never  17.7
#> 3 Female Occas  16.8
#> 4 Female Regul  17.5
#> 5 Male   Heavy  20.3
#> 6 Male   Never  19.6
#> # i 4 more rows
```

Több változóra egyszerre:

```
# több mutató csoportonként
survey |>
  group_by(Sex, Smoke) |>
```

```
  summarise(across(c(Wr.Hnd, Height, Age), \(x) mean(x, na.rm = TRUE)),
            .groups = "drop")
#> # A tibble: 10 x 5
#>   Sex    Smoke Wr.Hnd Height   Age
#>   <fct>  <fct>  <dbl>  <dbl> <dbl>
#> 1 Female Heavy   17.9   167.  22.7
#> 2 Female Never   17.7   166.  20.0
#> 3 Female Occas   16.8   167.  21.9
#> 4 Female Regul   17.5   160.  22.9
#> 5 Male   Heavy   20.3   181.  20.3
#> 6 Male   Never   19.6   178.  20.5
#> # i 4 more rows
```

A fenti kódban látható `.groups = "drop"` argumentum megadása opcionális, de ajánlott, mert így a `group_by()` által létrehozott csoportok törlődnek a `summarise()` után. Ha ezt nem tesszük meg, akkor a következő műveletek során a csoportok megmaradnak, ami nem mindig kívánatos.

8.2.2. Táblázatok

8.2.2.1. Abszolút gyakoriság

Faktor változók esetén a `count()` függvényt használjuk a *Tidyverse R*-ben.

```
# gyakorisági táblázat
survey |>
    count(W.Hnd)
#>   W.Hnd   n
#> 1  Left  18
#> 2 Right 218
#> 3  <NA>   1
survey |>
    count(W.Hnd, sort = TRUE)
#>   W.Hnd   n
#> 1 Right 218
#> 2  Left  18
#> 3  <NA>   1
```

Több változót is szerepeltethetünk a `count()` hívásban, ekkor a táblázat sorainak száma a megadott változók faktorszintjeinek kombinációjával nő. A `count()` függvény a `{dplyr}` csomag része, így a `{tidyverse}` csomag betöltése után automatikusan elérhetővé válik.

```
# kétdimenziós táblázat, adattáblában
survey |>
    count(Sex, W.Hnd)
#>      Sex W.Hnd   n
#> 1 Female  Left   7
#> 2 Female Right 110
#> 3 Female  <NA>   1
#> 4   Male  Left  10
#> 5   Male Right 108
#> 6   <NA>  Left   1
# háromdimenziós táblázat, adattáblában
survey |>
    count(Sex, W.Hnd, Exer)
#>       Sex W.Hnd Exer  n
#> 1  Female  Left Freq  3
#> 2  Female  Left None  1
#> 3  Female  Left Some  3
#> 4  Female Right Freq 45
#> 5  Female Right None 10
#> 6  Female Right Some 55
#> 7  Female  <NA> Freq  1
#> 8    Male  Left Freq  3
#> 9    Male  Left None  2
#> 10   Male  Left Some  5
#> 11   Male Right Freq 62
#> 12   Male Right None 11
#> 13   Male Right Some 35
#> 14   <NA>  Left Freq  1
```

Látható, hogy a `count()` függvény a `table()` függvénytől eltérően adattáblát hoz létre, így az eredmény könnyen felhasználható további műveletekhez. A `count()` függvény a `sort = TRUE` argumentummal rendezett táblázatot ad vissza, amely a legnagyobb gyakoriságú faktorszinttel kezdődik.

8.2.2.2. Relatív gyakoriság

A `count()` után a `mutate()` segítségével számítható a relatív és százalékos gyakoriság.

```
# relatív gyakoriság és százalékos gyakoriság
survey |>
    count(W.Hnd) |>
    mutate(rel = round(n/sum(n), 2), szazalek = round(100 * rel, 1))
#>   W.Hnd   n  rel szazalek
#> 1  Left  18 0.08        8
#> 2 Right 218 0.92       92
#> 3  <NA>   1 0.00        0
```

8.2.2.3. Kumulált gyakoriság

Ha a változó rendezett faktor, akkor a kumulált gyakoriságí táblázat kiíratása *Tidyverse R*-ben is szóba jöhet. A kumulált relatív gyakoriságok kiszámításához a `cumsum()` függvényt használjuk, amely a kumulált összegzésre szolgál.

```
# rendezett faktor készítése
survey <- survey |>
    mutate(Smoke = ordered(Smoke, levels = c("Never", "Occas", "Regul", "Heavy")))

# kumulált relatív gyakoriság, százalékos kumulált relatív gyakoriság
survey |>
    count(Smoke) |>
    mutate(kumulalt = cumsum(n), rel_kumulalt = 100 * cumsum(n/sum(n)))
#>   Smoke   n kumulalt rel_kumulalt
#> 1 Never 189      189     79.74684
#> 2 Occas  19      208     87.76371
#> 3 Regul  17      225     94.93671
#> 4 Heavy  11      236     99.57806
#> 5  <NA>   1      237    100.00000
```

Összefoglalás

A *Tidyverse R* segítségével a leíró statisztika számos lépése elegánsan és olvasható módon valósítható meg. A statisztikai mutatók számításához a `summarise()` és az `across()` függvények a leggyakrabban használt eszközök. Egy adott változóra könnyedén ki-

számíthatjuk az átlagot, szórást, mediánt, varianciát, minimumot, maximumot. Ezek a mutatók nemcsak egy-egy változóra alkalmazhatók, hanem egyszerre több oszlopra is, a `summarise(across(...))` konstrukció segítségével. Amennyiben egy változóra több mutatót szeretnénk egyszerre megjeleníteni, érdemes saját összesítő függvényt építeni, amely jól illeszkedik a *Tidyverse* szemléletéhez. A `group_by()` függvénnyel lehetőségünk van csoportok szerinti mutatók számítására. Így például meghatározhatjuk a testmagasság átlagát nemenként, vagy akár több faktor – például nem és dohányzási szokások – szerinti bontásban is. A gyakorisági táblák létrehozása is hatékony a *Tidyverse R*-ben. Az *Alap R* `table()` függvényének helyettesítője a `count()`, amely faktorváltozók gyakoriságát adja meg táblázatos formában. A `count()` több változóra is alkalmazható, így kétdimenziós vagy háromdimenziós kereszttáblákat is létrehozhatunk. A táblázatokat tovább bővíthetjük relatív gyakoriság és százalékos arány kiszámításával a `mutate()` segítségével. A rendezett faktorváltozók esetén a `cumsum()` függvény használatával kumulált gyakoriságokat is számolhatunk, így könnyen követhetjük az értékek fokozatos felhalmozódását.

🎯 Feladatok

1. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` változójára! Használja a *Tidyverse R* csomagjait!
2. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` és `Sex` változójára! Használja a *Tidyverse R* csomagjait!
3. Határozza meg a pulzus átlagát a `survey` adattábla `Sex` és `Smoke` változói szerint! Használja a *Tidyverse R* csomagjait!

8.3. Haladó lehetőségek 🤯

ℹ Miről lesz szó? Ebben a fejezetben

- megismerjük a mutatók és táblázatok kiírásának kényelmes lehetőségeit,
- a `psych`, `DescTools` és `summarytools` csomagok segítségével.

A leíró statisztika minden adatelemezés kihagyhatatlan része, nem csodálkozhatunk tehát azon, ha az R-hez készült mintegy 22 ezer külső csomagból számos kínál megoldást a mutatók és táblázatok kényelmes kiírására. Ahogy láttuk az előző fejezetekben az *Alap R* csomagok mellett a *Tidyverse R* csomagjait is egyszerűen használhatjuk, de talán a legjobban áttekinthető, és a legkisebb erőfeszítéssel elkészíthető mutatók és táblázatok az *egyéb csomagok* között vannak elrejtve. Ebben a fejezetben a `psych`, `DescTools` és a `summarytools` csomagok

lehetőségeit tekintjük át. Mindhárom csomag rendkívül kényelmessé teszi a mutatók és a gyakorisági táblázatok kiírását. Mindhárom csomag képes több változót kezelni, több faktorra csoportosítva több statisztikai mutatót megjeleníteni, tehát a legnagyobb szabadságot adják a kutató kezébe.

8.3.1. A `{psych}` csomag

A `{psych}` csomag az R egyik legnépszerűbb pszichológiai statisztikai elemzéseket támogató csomagja, amely kifejezetten az ilyen jellegű kutatások igényeihez lett kialakítva. Alkotója William Revelle.

A csomagban található `describe()` és `describeBy()` függvények kiválóan alkalmas numerikus változók mutatóinak kiszámítására, azonban a csomag a gyakorisági táblázatok készítését nem támogatja.

Nézzük sorba a lehetőségeket! A `describe()` függvény alkalmas a teljes adattábla áttekintésére. Paraméterben mindössze az elemzendő adattáblát kell megadnunk.

```
# a teljes adattábla leírása
library(psych)
describe(x = survey)
#>        vars   n   mean    sd median trimmed   mad    min   max range
#> Sex*      1 236   1.50  0.50   1.50    1.50  0.74   1.00   2.0  1.00
#> Wr.Hnd    2 236  18.67  1.88  18.50   18.61  1.48  13.00  23.2 10.20
#> NW.Hnd    3 236  18.58  1.97  18.50   18.55  1.63  12.50  23.5 11.00
#> W.Hnd*    4 236   1.92  0.27   2.00    2.00  0.00   1.00   2.0  1.00
#> Fold*     5 237   2.09  0.96   3.00    2.11  0.00   1.00   3.0  2.00
#> Pulse     6 192  74.15 11.69  72.50   74.02 11.12  35.00 104.0 69.00
#> Clap*     7 236   2.46  0.76   3.00    2.57  0.00   1.00   3.0  2.00
#> Exer*     8 237   1.93  0.95   2.00    1.91  1.48   1.00   3.0  2.00
#> Smoke*    9 236   1.36  0.81   1.00    1.15  0.00   1.00   4.0  3.00
#> Height   10 209 172.38  9.85 171.00  172.19 10.08 150.00 200.0 50.00
#> M.I*     11 209   1.67  0.47   2.00    1.72  0.00   1.00   2.0  1.00
#> Age      12 237  20.37  6.47  18.58   18.99  1.61  16.75  73.0 56.25
#>          skew kurtosis   se
#> Sex*    0.00    -2.01 0.03
#> Wr.Hnd  0.18     0.30 0.12
#> NW.Hnd  0.02     0.44 0.13
#> W.Hnd* -3.17     8.10 0.02
#> Fold*  -0.18    -1.89 0.06
```

```
#> Pulse  -0.02     0.33 0.84
#> Clap*  -0.98    -0.60 0.05
#> Exer*   0.14    -1.88 0.06
#> Smoke*  2.15     3.45 0.05
#> Height  0.22    -0.44 0.68
#> M.I*   -0.74    -1.46 0.03
#> Age     5.16    33.47 0.42
```

Az output soraiban minden változó megjelenik, a `vars` oszlopban a változó sorszáma, a `n` oszlopban a nem hiányzó (érvényes) értékek száma, a `mean` oszlopban az átlag, a `sd` oszlopban a szórás, a `median` oszlopban a medián, a `trimmed` oszlopban egy robusztusabb átlag (a legkisebb és legnagyobb 10%-ot figyelmen kívül hagyó átlag), a `mad` oszlopban a medián abszolút eltérés, a `min` és `max` oszlopokban az alsó és felső kvartilisek, a `range` oszlopban az értékek terjedelme, a `skew` oszlopban az eloszlás ferdesége, a `kurtosis` oszlopban pedig az eloszlás csúcsossága található. Az utolsó oszlopban pedig az átlag szórása szerepel. A faktor változók esetén a `*` jelzi, hogy a sorban megjelenő mutatók nem biztos hogy értelmezhetőek, így ezeket a sorokat érdemes figyelmen kívül hagyni.

Létezik két másik adatbázis leíró függvény is, a `describeData()` és a `describeFast()`.

```
# adattábla leírása
describeData(survey, head = 3, tail = 3)
#> n.obs =  237 of which  168   are complete cases.
#>        variable # n.obs type     H1       H2      H3     T1     T2
#> Sex*            1   236    2 Female     Male    Male Female   Male
#> Wr.Hnd          2   236    1   18.5     19.5    18.0   17.5   21.0
#> NW.Hnd          3   236    1   18.0     20.5    13.3   16.5   21.5
#> W.Hnd*          4   236    2  Right     Left   Right  Right  Right
#> Fold*           5   237    2 R on L   R on L  L on R R on L R on L
#> Pulse           6   192    1     92      104      87   <NA>     90
#> Clap*           7   236    2   Left     Left Neither  Right  Right
#> Exer*           8   237    2   Some     None    None   Some   Some
#> Smoke*          9   236    2  Never    Regul   Occas  Never  Never
#> Height         10   209    1  173.0    177.8    <NA>  170.0  183.0
#> M.I*           11   209    2 Metric Imperial    <NA> Metric Metric
#> Age            12   237    1 18.250   17.583  16.917 18.583 17.167
#>            T3
#> Sex*   Female
#> Wr.Hnd   17.6
#> NW.Hnd   17.3
```

```
#> W.Hnd*  Right
#> Fold*  R on L
#> Pulse      85
#> Clap*   Right
#> Exer*    Freq
#> Smoke*  Never
#> Height  168.5
#> M.I*   Metric
#> Age    17.750
```

A `describeData()` függvény outputjából megtudjuk a mintaelemszámot (237) és a teljes sorok (168) számát, valamint transzponálva áttekinthetjük a változók nevét, érvényes adatainak számát, illetve a `head=` és `tail=` argumentumban szabályozottan az adatbázis első és utolsó néhány sorát.

Megjegyezzük, hogy a `headTail()` függvény a `psych` csomagban tökéletesen integrálja a `head()` és `tail()` függvényeket.

```
# az adattábla első és utolsó néhány sorának megjelenítése
headTail(survey)
#>        Sex Wr.Hnd NW.Hnd W.Hnd   Fold Pulse    Clap Exer Smoke Height
#> 1   Female   18.5     18 Right R on L    92    Left Some Never    173
#> 2     Male   19.5   20.5  Left R on L   104    Left None Regul  177.8
#> 3     Male     18   13.3 Right L on R    87 Neither None Occas   <NA>
#> 4     Male   18.8   18.9 Right R on L  <NA> Neither None Never    160
#> ...   <NA>    ...    ...  <NA>   <NA>   ...    <NA> <NA>  <NA>    ...
#> 234 Female   18.5     18 Right L on R    88   Right Some Never    160
#> 235 Female   17.5   16.5 Right R on L  <NA>   Right Some Never    170
#> 236   Male     21   21.5 Right R on L    90   Right Some Never    183
#> 237 Female   17.6   17.3 Right R on L    85   Right Freq Never  168.5
#>          M.I   Age
#> 1     Metric 18.25
#> 2   Imperial 17.58
#> 3       <NA> 16.92
#> 4     Metric 20.33
#> ...     <NA>   ...
#> 234   Metric 16.92
#> 235   Metric 18.58
#> 236   Metric 17.17
#> 237   Metric 17.75
```

A másik adatbázis leíró függvény a `describeFast()`, amely minden változó esetén tisztázza annak típusát.

```
# a teljes adattábla leírása
print(describeFast(survey), short=F)
#>  Number of observations =  237 of which  168   are complete cases.
#> numeric and  7  are factors
#>        var n.obs numeric factor logical character type
#> Sex      1   236       0      1       0         0   NA
#> Wr.Hnd   2   236       1      0       0         0   NA
#> NW.Hnd   3   236       1      0       0         0   NA
#> W.Hnd    4   236       0      1       0         0   NA
#> Fold     5   237       0      1       0         0   NA
#> Pulse    6   192       1      0       0         0   NA
#> Clap     7   236       0      1       0         0   NA
#> Exer     8   237       0      1       0         0   NA
#> Smoke    9   236       0      1       0         0   NA
#> Height  10   209       1      0       0         0   NA
#> M.I     11   209       0      1       0         0   NA
#> Age     12   237       1      0       0         0   NA
```

A `describe()` függvény azonban nemcsak a teljes adattábla leírására alkalmas, hanem egyes változók leírására is. A `describe()` függvény első argumentuma lehet a vizsgálandó változó, a második argumentum pedig a `fast=`, amelynek `TRUE` értéke esetén gyorsabban, de kevesebb információt ad vissza a függvény.

```
# testmagasság leírása
describe(survey$Height)
#>    vars   n   mean   sd median trimmed   mad min max range skew
#> X1    1 209 172.38 9.85    171  172.19 10.08 150 200    50 0.22
#>    kurtosis   se
#> X1    -0.44 0.68
```

```
# testmagasság leírása, kevesebb információ
describe(survey$Height, fast = T)
#>    vars   n   mean   sd median min max range skew kurtosis   se
#> X1    1 209 172.38 9.85    171 150 200    50 0.22    -0.44 0.68
```

A `{psych}` csomag talán legerősebb leíró statisztikai függvénye a `describeBy()`, amely csoportokra bontva adja vissza a mutatókat. A `describeBy()` függvény első argumentuma a

vizsgálandó változó, a második argumentum pedig a csoportosító változó. A `mat=` argumentum `TRUE` értéke esetén mátrix formátumban kapjuk meg az eredményt.

```
# testmagasság leírása nemre és dohányzási szokásra bontva
describeBy(x = survey$Height, group = survey[, c("Sex", "Smoke")], mat = T,
    fast = T)
#>     item group1 group2 vars  n     mean       sd median    min    max
#> X11    1 Female  Never    1 86 165.6584 6.370229 165.80 150.00 180.34
#> X12    2   Male  Never    1 82 178.2800 8.825695 180.00 154.94 200.00
#> X13    3 Female  Occas    1  8 167.4150 3.621219 168.00 160.02 172.00
#> X14    4   Male  Occas    1  8 178.6425 6.665686 176.50 171.00 190.00
#> X15    5 Female  Regul    1  3 159.8067 7.302338 160.02 152.40 167.00
#> X16    6   Male  Regul    1 11 182.2200 5.515099 180.34 172.72 190.00
#> X17    7 Female  Heavy    1  5 166.9360 3.124977 166.40 163.00 170.18
#> X18    8   Male  Heavy    1  5 180.6080 8.399305 179.00 171.00 193.04
#>     range        skew   kurtosis        se
#> X11 30.34 -0.24281202 -0.4967863 0.6869194
#> X12 45.06 -0.16370591 -0.2480796 0.9746350
#> X13 11.98 -0.78216144 -0.4439181 1.2802943
#> X14 19.00  0.42080839 -1.4714651 2.3566757
#> X15 14.60 -0.02918946 -2.3333333 4.2160065
#> X16 17.28  0.02045201 -1.4047690 1.6628650
#> X17  7.18 -0.02894376 -2.0555580 1.3975321
#> X18 22.04  0.32790041 -1.6739624 3.7562833
```

8.3.2. A `{DescTools}` csomag

A `{DescTools}` csomag egy átfogó eszköz leíró statisztikákhoz és egyszerű elemzésekhez, amelyet Andri Signorell fejlesztett. Ebben a csomagban a `Desc()` függvény a legfontosabb, a bemenő paramétertől függően az épp kívánt mutatókat számítja ki. Fontos megjegyezni, hogy a `Desc()` függvény nemcsak numerikus, hanem faktor változók esetén is használható, továbbá alapértelmezés szerint ábrát is rajzol. Ez utóbbi kényelmi funkció a `plotit=F` argumentummal letiltható.

Kezdjük a `Desc()` függvény használatát a `survey` adattáblán.

```
library(DescTools)
# a teljes adattábla leírása
Desc(x = survey, plotit = F)
#> ──────────────────────────────────────────────────────────────────────────────
```

```
#> Describe survey (data.frame):
#>
#> data frame:  237 obs. of  12 variables
#>      168 complete cases (70.9%)
#>
#>   Nr  Class  ColName  NAs         Levels
#>   1   fac    Sex       1 (0.4%)   (2): 1-Female, 2-Male
#>   2   num    Wr.Hnd    1 (0.4%)
#>   3   num    NW.Hnd    1 (0.4%)
#>   4   fac    W.Hnd     1 (0.4%)   (2): 1-Left, 2-Right
#>   5   fac    Fold      .          (3): 1-L on R, 2-Neither, 3-R on L
#>   6   int    Pulse    45 (19.0%)
#>   7   fac    Clap      1 (0.4%)   (3): 1-Left, 2-Neither, 3-Right
#>   8   fac    Exer      .          (3): 1-Freq, 2-None, 3-Some
#>   9   ord    Smoke     1 (0.4%)   (4): 1-Never, 2-Occas, 3-Regul,
#>                                    4-Heavy
#>   10  num    Height   28 (11.8%)
#>   11  fac    M.I      28 (11.8%)  (2): 1-Imperial, 2-Metric
#>   12  num    Age       .
#>
#>
#> ---------------------------------------------------------------------
#> 1 - Sex (factor - dichotomous)
#>
#>   length      n    NAs unique
#>      237    236      1      2
#>           99.6%   0.4%
#>
#>         freq   perc  lci.95  uci.95'
#> Female   118  50.0%   43.7%   56.3%
#> Male     118  50.0%   43.7%   56.3%
#>
#> ' 95%-CI (Wilson)
#>
#> ---------------------------------------------------------------------
#> 2 - Wr.Hnd (numeric)
#>
#>   length      n    NAs  unique     0s   mean  meanCI'
#>      237    236      1      60      0  18.67   18.43
#>           99.6%   0.4%           0.0%          18.91
```

```
#>
#>      .05    .10    .25  median    .75    .90     .95
#>    16.00  16.50  17.50   18.50  19.80  21.15   22.05
#>
#>    range     sd  vcoef     mad    IQR   skew    kurt
#>    10.20   1.88   0.10    1.48   2.30   0.18    0.30
#>
#> lowest : 13.0 (2), 14.0 (2), 15.0, 15.4, 15.5 (2)
#> highest: 22.5 (4), 22.8, 23.0 (2), 23.1, 23.2 (3)
#>
#> heap(?): remarkable frequency (9.7%) for the mode(s) (= 17.5)
#>
#> ' 95%-CI (classic)
#> ...
```

A `Desc()` fenti outputja kimerítő leíró elemzést nyújt a kutatónak. Az output első részében a teljes adattáblát tekinthetjük át, ezt követően változónként kapunk elemzést. Az egyes változókra vonatkozó elemzéseket egyesével is kikérhetjük, ehhez csak a bemenő paramétert kell a kívánt változóra állítani. A bemenő paramétertől, vagyis az oszlop típusától függően kapunk mutatókat (numerikus vektor) vagy gyakorisági táblázatot (faktor esetén) eredményül.

```
# testmagasság leírása: példa numerikus változóra
Desc(x = survey$Height, plotit = F)
#> ──────────────────────────────────────────────────────────────────────────────
#> survey$Height (numeric)
#>
#>   length       n     NAs  unique      0s    mean  meanCI'
#>      237     209      28      67       0  172.38  171.04
#>            88.2%   11.8%           0.0%          173.72
#>
#>      .05     .10     .25  median     .75     .90     .95
#>   157.00  160.00  165.00  171.00  180.00  185.42  189.60
#>
#>    range      sd   vcoef     mad     IQR    skew    kurt
#>    50.00    9.85    0.06   10.08   15.00    0.22   -0.44
#>
#> lowest : 150.0, 152.0, 152.4, 153.5, 154.94 (2)
#> highest: 191.8, 193.04, 195.0, 196.0, 200.0
#>
#> ' 95%-CI (classic)
```

```
# testmozgás leírása: példa faktor változóra, 1D gyakorisági táblázatra
Desc(x = survey$Exer, plotit = F)
#> ──────────────────────────────────────────────────────────────────────
#> survey$Exer (factor)
#>
#>   length      n    NAs unique levels  dupes
#>      237    237      0      3      3      y
#>          100.0%   0.0%
#>
#>    level  freq   perc  cumfreq  cumperc
#> 1   Freq   115  48.5%      115    48.5%
#> 2   Some    98  41.4%      213    89.9%
#> 3   None    24  10.1%      237   100.0%
```

Egyszerre több változó hatását is figyelembe vehetjük, akkor nem az eddig használt `x=` argumentumot használjuk, hanem a `formula=` argumentumot.

```
# két faktor: példa 2D gyakorisági táblázatra
Desc(formula = Exer ~ Sex, data = survey, plotit = F)
#> ──────────────────────────────────────────────────────────────────────
#> Exer ~ Sex (survey)
#>
#> Summary:
#> n: 236, rows: 2, columns: 3
#>
#> Pearson's Chi-squared test:
#>   X-squared = 5.7184, df = 2, p-value = 0.05731
#> Log likelihood ratio (G-test) test of independence:
#>   G = 5.7449, X-squared df = 2, p-value = 0.05656
#> Mantel-Haenszel Chi-squared:
#>   X-squared = 5.4577, df = 1, p-value = 0.01948
#>
#> Contingency Coeff.     0.154
#> Cramer's V             0.156
#> Kendall Tau-b          -0.146
#>
#>
#>          Exer    Freq   None   Some    Sum
#> Sex
#>
```

```
#> Female   freq      49     11     58    118
#>          perc   20.8%   4.7%  24.6%  50.0%
#>          p.row  41.5%   9.3%  49.2%      .
#>          p.col  43.0%  45.8%  59.2%      .
#>
#> Male     freq      65     13     40    118
#>          perc   27.5%   5.5%  16.9%  50.0%
#>          p.row  55.1%  11.0%  33.9%      .
#>          p.col  57.0%  54.2%  40.8%      .
#>
#> Sum      freq     114     24     98    236
#>          perc   48.3%  10.2%  41.5% 100.0%
#>          p.row      .      .      .      .
#>          p.col      .      .      .      .
#>
```

```
# 1 numerikus vektor és 1 faktor
Desc(formula = Height ~ Exer, data = survey, plotit = F)
#> ──────────────────────────────────────────────────────────────────────────────
#> Height ~ Exer (survey)
#>
#> Summary:
#> n pairs: 237, valid: 209 (88.2%), missings: 28 (11.8%), groups: 3
#>
#>
#>            Freq     None     Some
#> mean    174.607  169.028  170.397
#> median  173.000  167.000  168.950
#> sd        9.779    9.471    9.472
#> IQR      12.340    8.500   12.500
#> n           105       20       84
#> np      50.239%   9.569%  40.191%
#> NAs          10        4       14
#> 0s            0        0        0
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 13.093, df = 2, p-value = 0.001435
```

```
# 1 numerikus vektor és 2 faktor
Desc(formula = Height ~ Exer * Sex, data = survey, plotit = F)
#> ──────────────────────────────────────────────────────────────────────────────
#> Height ~ Exer (survey)
#>
#> Summary:
#> n pairs: 237, valid: 209 (88.2%), missings: 28 (11.8%), groups: 3
#>
#>
#>            Freq     None     Some
#> mean    174.607  169.028  170.397
#> median  173.000  167.000  168.950
#> sd        9.779    9.471    9.472
#> IQR      12.340    8.500   12.500
#> n           105       20       84
#> np      50.239%   9.569%  40.191%
#> NAs          10        4       14
#> 0s            0        0        0
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 13.093, df = 2, p-value = 0.001435
#>
#>
#>
#> ──────────────────────────────────────────────────────────────────────────────
#> Height ~ Sex (survey)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 2
#>
#>
#>          Female     Male
#> mean    165.687  178.826
#> median  166.750  180.000
#> sd        6.152    8.380
#> IQR       7.440   12.210
#> n           102      106
#> np      49.038%  50.962%
#> NAs          16       12
#> 0s            0        0
```

```
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 96.677, df = 1, p-value < 2.2e-16
#>
#>
#> Warning:
#>   Grouping variable contains 1 NAs (0.422%).
#>
#> ──────────────────────────────────────────────────────────────────────
#> Height ~ Exer:Sex (survey)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 6
#>
#>
#>         Freq:Female  None:Female  Some:Female    Freq:Male    None:Male
#> mean        167.131      162.996      164.838      180.353      173.964
#> median      167.640      165.000      165.050      180.340      171.000
#> sd            6.421        4.795        5.890        7.944        9.616
#> IQR           8.000        9.000        7.120        9.950       11.570
#> n                45            9           48           59           11
#> np          21.635%       4.327%      23.077%      28.365%       5.288%
#> NAs               4            2           10            6            2
#> 0s                0            0            0            0            0
#>
#>           Some:Male
#> mean        177.809
#> median      179.550
#> sd            8.201
#> IQR          11.455
#> n                36
#> np          17.308%
#> NAs               4
#> 0s                0
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 104.51, df = 5, p-value < 2.2e-16
#>
#>
#> Warning:
```

```
#>   Grouping variable contains 1 NAs (0.422%).
```

```
# 2 numerikus vektor
Desc(formula = Height ~ Pulse, data = survey, plotit = F)
#> ──────────────────────────────────────────────────────────────────────────────
#> Height ~ Pulse (survey)
#>
#> Summary:
#> n pairs: 237, valid: 171 (72.2%), missings: 66 (27.8%)
#>
#>
#> Pearson corr. : -0.084
#> Spearman corr.: -0.110
#> Kendall corr. : -0.076
```

A fenti példák alapján elmondhatjuk, hogy a `Desc()` függvény a `formula=` argumentumot használva képes a numerikus változók és faktorok közötti összefüggések vizsgálatára, továbbá a mutatók és táblázatok megjelenítését kapcsolódó hipotézisvizsgálatokkal egészíti ki.

8.3.3. A `{summarytools}` csomag

A `{summarytools}` csomag egy kiváló eszköz az adatok gyors és áttekinthető összefoglalására R-ben. Elsősorban adatelemzési beszámolók, interaktív riportok, valamint oktatási célok esetén hasznos, mert látványos, informatív outputot biztosít, jól paraméterezhető, egyszerű függvényhívásokkal operál.

A csomag legfontosabb 4 függvényét a 8.2. táblázat foglalja össze.

8.2. táblázat: A `{summarytools}` csomag leíró statisztikai eszközei *(saját szerkesztés)*

Függvény	Célja
`dfSummary()`	Teljes adattábla összegzése
`freq()`	Egy változó gyakorisági táblája
`descr()`	Numerikus leíró statisztikák
`ctable()`	Kereszttábla készítése

A `{summarytools}` kiváló leírása olvasható a csomag hivatalos oldalán. A egyik legnagyobb előnye a gazdag paraméterezhetőség. Mielőtt bármelyik függvényt használnánk, érdemes a csomag ezen beállításait megadni.

A 8.2. táblázat függvényei sok olyan kísérő címkét jelenítenek meg, amelyek a leíró statisztikai eredmények értelmezését segítik, például a `descr()` a numerikus változó mutatóinak közlése során megjeleníti a `Mean` és `Std.Dev` címkéket, de gyakorisági táblázatok megjelenítése során (`freq()`) a `% Valid` és `% Total` címkék is feltűnnek.

```
# a testmagasság leírása
library(summarytools)
summarytools::descr(survey[, c("Height", "Pulse")], stats = c("mean", "sd",
    "min", "max", "n.valid"), transpose = T)
#> Descriptive Statistics
#> survey
#> N: 237
#>
#>                  Mean   Std.Dev      Min      Max   N.Valid
#> ------------ -------- --------- -------- -------- ---------
#>       Height   172.38      9.85   150.00   200.00    209.00
#>        Pulse    74.15     11.69    35.00   104.00    192.00
```

```
# a testmozgás leírása
summarytools::freq(survey$Exer, order = "name", cumul = F)
#> Frequencies
#> survey$Exer
#> Type: Factor
#>
#>               Freq   % Valid   % Total
#> ----------- ------ --------- ---------
#>        Freq    115     48.52     48.52
#>        None     24     10.13     10.13
#>        Some     98     41.35     41.35
#>        <NA>      0                0.00
#>       Total    237    100.00    100.00
```

A fenti outputokban látható címkék angol nyelven jelennek meg, amelyeket a csomag automatikusan generál. Azonban ezeket a címkéket átírhatjuk akár magyar nyelvre is, ha a csomag hivatalos oldaláról letöltjük azt a CSV sablont, amellyel testre szabhatjuk a megjelenő eredményeket.

A nyelvi állomány betöltése a `use_custom_lang()` függvény segítségével történik, miután átírtuk a kívánt címkéket és az *RStudio* projektünk könyvtárában eltároltuk a CSV állományt.

```
# a nyelvi állomány betöltése
summarytools::use_custom_lang(file = "summarytools_hu.csv")
```

A korábbi elemzéseket újra végrehajtva már magyar nyelvű címkéket kapunk.

```
# a nyelvi állomány betöltése után már magyar nyelvű címkék jelennek meg
summarytools::descr(survey[, c("Height", "Pulse")], stats = c("mean", "sd",
    "min", "max", "n.valid"), transpose = T)
#> Descriptive Statistics
#> survey
#> N: 237
#>
#>                 Átlag   Szórás     Min.     Max.   Érvényes
#> ------------ -------- -------- -------- -------- ----------
#>       Height   172.38     9.85   150.00   200.00     209.00
#>        Pulse    74.15    11.69    35.00   104.00     192.00
```

```
summarytools::freq(survey$Exer, order = "name", cumul = F)
#> Frequencies
#> survey$Exer
#> Type: Factor
#>
#>                  Gyakoriság   % Érvényes   % Összesen
#> -------------- ------------ ------------ ------------
#>           Freq          115        48.52        48.52
#>           None           24        10.13        10.13
#>           Some           98        41.35        41.35
#>           <NA>            0                      0.00
#>       Összesen          237       100.00       100.00
```

A `{summarytools}` csomag rendelkezik egy `st_options()` nevű globális beállító függvénnyel is, amelynek segítségével a csomag alapértelmezett beállításait adhatjuk meg. A `st_options()` függvény egyik argumentuma a `lang=`, amelynek azonban egyelőre nem lehet az értéke `"hu"`, épp ezért volt szükség a `use_custom_lang()` függvényre és a nyelvi sablon kitöltésére.

Az `st_options()` függvény azonban számunkra is hasznos beállításokat tartalmaz, például a `headings = F` argumentum megadásával elrejthetjük a címkéket, a `freq.cumul = F` argumentummal pedig a kumulált gyakoriságokat kapcsolhatjuk ki. A `style = "rmarkdown"` és `plain.ascii = FALSE` beállítása esetén alkalmassá válnak a `{summarytools}` csomag által generált táblázatok a `Quarto` dokumentumokban való megjelenítésre. Ehhez a csonkban a `#|`

`results:"asis"` beállítást is alkalmazni kell. A további beállítások is az egyszerűbb megjelenítést szolgálják, például a `dfSummary.silent = TRUE` elrejti a redundáns üzeneteket, a `footnote = NA` minimalizálja az aláírásokat, a `freq.report.nas = F` pedig elrejti az `NA` értékek figyelembe vételét.

```
# a summarytools csomag beállításai
summarytools::st_options(
  headings = F,    # elrejti a címkéket
  freq.cumul = F,  # elrejti a kumulált gyakoriságokat
  style = "rmarkdown", # Quarto output számára ez a megfelelő
  plain.ascii = FALSE, # Quarto output számára ez a megfelelő
  dfSummary.silent = TRUE, # elrejti a redundáns üzeneteket
  footnote = NA,      # minimalizálja az
  freq.report.nas = F # elrejti a NA értékek figyelembe vételét
)
```

A kívánt magyar nyelvű outputok beállításához érdemes a tizedesvesszőt is beállítani, amelyet a `options(OutDec = ",")` parancs segítségével tehetünk meg.

```
options(OutDec = ",")  # tizedesvessző beállítása
```

A fenti beállítások után már magyar nyelvű címkéket és kevesebb outputot várunk, valamint a táblázataink *Quarto* dokumentumokba is jobban illeszthetők, ráadásul a tizedesvesszőt használják az egész- és törtrész elválasztására. Futtassuk le újra a korábbi példákat, hogy lássuk a különbséget!

```
# magyar címkék, tizedesvessző, nincs fejléc, Quarto kompatibilis
summarytools::descr(survey[, c("Height", "Pulse")], stats = c("mean", "sd",
    "min", "max", "n.valid"), transpose = T)
```

	Átlag	Szórás	Min.	Max.	Érvényes
Height	172,38	9,85	150,00	200,00	209,00
Pulse	74,15	11,69	35,00	104,00	192,00

```
# magyar címkék, tizedesvessző, nincs fejléc, Quarto kompatibilis
summarytools::freq(survey$Exer, order = "name", cumul = F)
```

	Gyakoriság	%
Freq	115	48,52
None	24	10,13
Some	98	41,35
Összesen	237	100,00

Mostanra világossá válhatott számunkra, hogy a `{summarytools}` csomag milyen nagyszerű lehetőségeket kínál a leíró statisztikai mutatók és táblázatok megjelenítésére. Használhatjuk a HTML, MS Word vagy PDF dokumentumokban, de természetesen az egyszerű konzolos megjelenítést is kényelmesebbé teszik.

Fókuszáljunk most már azokra a leíró statisztikai elemzésekre, amit a `{summarytools}` csomag nyújt. Ahogyan a 8.2. táblázatban összefoglaltuk a `dfSummary()` a teljes adattábla összegzésére használható. Ennek az outputját terjedelmi okok miatt nem közöljük.

```
# a teljes adattábla leírása
summarytools::dfSummary(survey)
```

Az egydimenziós gyakorisági táblázatok készítésére a `freq()` függvény használható. A `{summarytools}` függvényei gazdagon paraméterezhetők, így például a `totals=F` argumentum megadásával elrejthetjük az összesítő sorokat a gyakorisági táblázatokban.

```
# 1D gyakorisági táblázat
summarytools::freq(survey$Exer, order = "name", totals = F)
```

	Gyakoriság	%
Freq	115	48,52
None	24	10,13
Some	98	41,35

Kétdimenziós gyakorisági táblázat készítésére a `ctable()` függvény használható. Továbbra is rejthetjük az összesítő sorokat a `totals=F` argumentummal, a `prop = "r"` argumentummal pedig a soronként vett relatív gyakoriságok megjelenését írhatjuk elő. A `chisq = T` argumentum megadásával a függvény elvégzi a khí-négyzet próbát is, amelynek eredménye szintén megjelenik az outputban. A `useNA = "no"` argumentum megadásával pedig elrejthetjük az `NA` értékeket.

```
# 2D gyakorisági táblázat
summarytools::ctable(x = survey$Exer, y = survey$Sex, prop = "r", totals = F,
    chisq = T, useNA = "no")
```

Sex	Female	Male
Exer		
Freq	49 (43,0%)	65 (57,0%)
None	11 (45,8%)	13 (54,2%)
Some	58 (59,2%)	40 (40,8%)

Chi.squared	df	p.value
5,7184	2	0,0573

Numerikus változókat a `descr()` függvény segítségével elemezhetünk. A `stats=` argumentumban megadhatjuk a kívánt mutatókat, például `mean`, `sd`, `min`, `max`, `median`, `mad`, `iqr`, `cv`, `skewness`, `se.skewness`, `kurtosis`, `n.valid`, `n` és `pct.valid`. Most a `stats = "common"` argumentumot használjuk, amely a leggyakoribb mutatókat adja vissza.

```
# több numerikus változó leírása
summarytools::descr(survey[, c("Height", "Wr.Hnd")], stats = "common")
```

	Height	Wr.Hnd
Átlag	172,38	18,67
Szórás	9,85	1,88
Min.	150,00	13,00
Medián	171,00	18,50
Max.	200,00	23,20
Érvényes	209,00	236,00
N	237,00	237,00
% Érvényes	88,19	99,58

Csoportokra is alkalmazhatjuk a `descr()` függvényt, amelyhez a `{summarytools}` csomag `stby()` függvényét használhatjuk. A `stby()` függvény első argumentuma a vizsgálandó változó, a második argumentuma a csoportosító változó, a harmadik argumentuma pedig a kívánt mutatókat tartalmazza. A `transpose = T` argumentum megadásával a kimenetet transzponálhatjuk, amely így jobban áttekinthetővé válik.

```r
# több numerikus változó csoportokra
with(survey,
    stby(data    = Height,
         INDICES = Exer,
         FUN     = descr,
         stats   = c("mean", "sd", "min", "med", "max"),
         useNA   = TRUE,
         transpose = T))
```

	Átlag	Szórás	Min.	Medián	Max.
Freq	174,61	9,78	150,00	173,00	200,00
None	169,03	9,47	157,48	167,00	190,50
Some	170,40	9,47	152,00	168,95	193,04

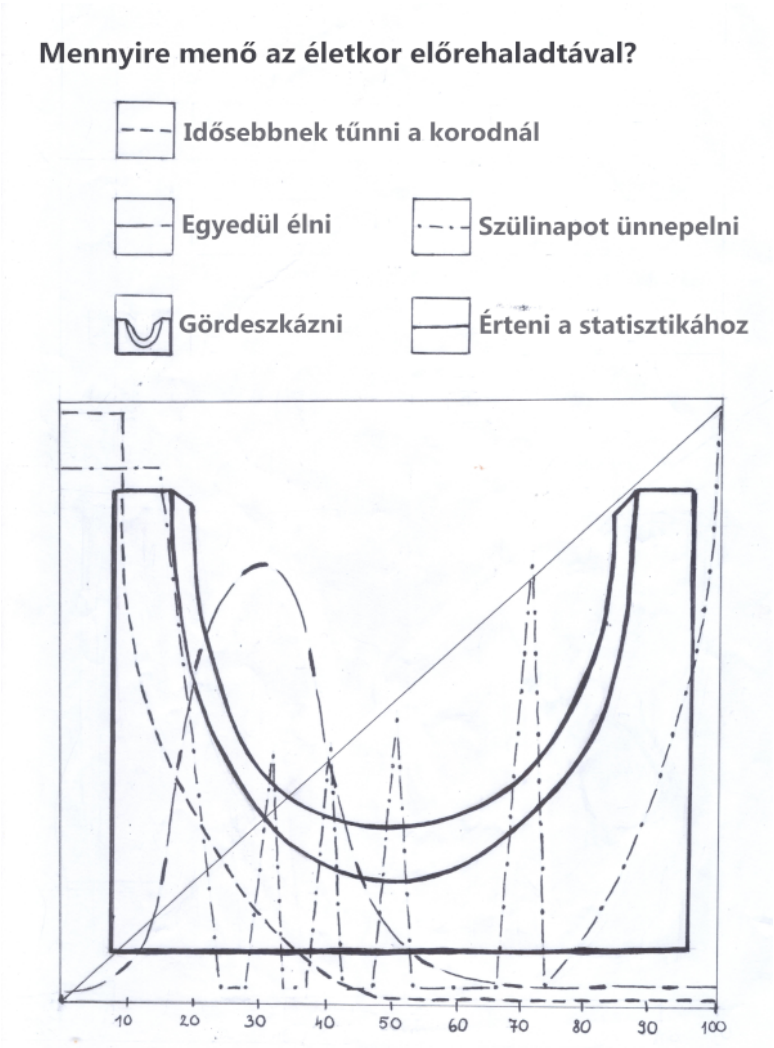
Összefoglalás

A `{psych}`, `{DescTools}` és `{summarytools}` csomagok a mutatók és gyakorisági táblák kényelmes, gyors és informatív előállítását teszik lehetővé – sokszor egyszerűbb és részletesebb formában, mint az *Alap R* vagy *Tidyverse R* eszközei. A `{psych}` csomag a pszichológiai kutatások igényeihez igazodik, főként numerikus változók mutatóinak kiszámítására szolgál. A `describe()` függvénnyel egy teljes adattábla vagy egyes változók mutatói nyerhetők ki – például átlag, szórás, ferdeség, csúcsosság. A `describeBy()` függvény segítségével ugyanezeket a statisztikákat csoportokra bontva is megkaphatjuk. A `{DescTools}` csomag átfogó eszköztárat nyújt. Kiemelt függvénye, a `Desc()`, egyszerre tud mutatókat számítani numerikus változókra és gyakorisági táblákat készíteni faktorokra. A függvény különlegessége, hogy `formula=` szintaxissal kombinált változóelemzésre is alkalmas, továbbá az eredményekhez ábrákat is társít (ez kikapcsolható). A `Desc()` az egyik legsokoldalúbb eszköz a teljes adattábla vagy akár két faktor (kereszttábla) elemzésére is. A `{summarytools}` csomag látványos és strukturált outputot kínál, különösen jól használható oktatási anyagokban, interaktív riportokban vagy *Quarto* dokumentumokban. Négy fő függvénye: `dfSummary()` (teljes adattábla), `freq()` (gyakorisági táblák), `descr()` (numerikus mutatók), valamint `ctable()` (kereszttáblák). A `st_options()` függvénnyel testre szabhatók a megjelenítési beállítások, a `use_custom_lang()` segítségével pedig a kimenet akár magyar nyelvre is átállítható.

🎯 Feladatok

1. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` változójára! Használja a `{DescTools}`, majd a `{summarytools}` csomagot!
2. Készítsen gyakorisági táblázatot a `survey` adattábla `Smoke` és `Sex` változójára! Használja a `{DescTools}`, majd a `{summarytools}` csomagot!
3. Határozza meg a pulzus átlagát a `survey` adattábla `Sex` és `Smoke` változói szerint! Használja a `{psych}`, `{DescTools}`, majd a `{summarytools}` csomagot!

9. Modern grafika



Az R-ben több grafikus rendszer közül választhatunk, amikor ábrák rajzolásához kezdünk. A hagyományos grafikus rendszer mellett elérhető az ún. grid grafikus rendszer, a trellis/lattice rendszer és a ggplot2 rendszer is. Az egyes rendszerek eltérő megközelítést használnak az ábrák létrehozásához, és természetesen különböző csomagok különböző függvényeit használják.

- A *hagyományos grafikus* rendszer az S nyelv grafikus rendszerének implementációja. A magasszintű grafikus függvények a komplett ábrák létrehozásért felelősek, az alacsony szintű függvényekkel pedig kisebb-nagyobb grafikus elemeket helyezhetünk a meglévő ábrára. Mindig "rárajzolunk" a meglévő elemekre, a későbbi módosításra vagy törlésre nincs lehetőségünk.
- A `{grid}` csomagból elérhető grafikus rendszer grafikus primitívek rendkívül gazdag tárháza. Segítségükkel grafikus objektumokat építhetünk, amelyek az ábrától független reprezentációval rendelkeznek, így azok később módosíthatók. A saját koordináta-rendszerrel rendelkező viewport-ok rendszere tetszőlegesen bonyolult ábrák létrehozását segíti. A grid rendszer maga nem tartalmaz statisztikai rajzfüggvényeket, de számos, a grid csomagra épülő rendszer igen, például a lattice és a ggplot2.
- A `{lattice}` csomag grafikus rendszere az ún. trellis grafikus rendszer megvalósítása R-ben. A hagyományos grafikus rendszerhez képest rendkívül sok fejlesztést tartalmaz. A grid grafikus rendszerre épül, így hordozza annak rugalmasságát.
- A `{ggplot2}` csomag grafikus rendszere kísérletet tesz arra, hogy a hagyományos és a lattice grafikus rendszer előnyös tulajdonságait ötvözze. Szintén a grid rendszerre épül. A `{ggplot2}` csomag a *Tidyverse R* része, így a modern R grafikus megjelenítőjének is tekinthetjük. A többi felsorolt grafikus rendszer az *Alap R* része.

A következő fejezetekben a ggplot2 grafikus rendszert ismertetjük.

A továbblépés előtt gondoskodjunk a következő csomagok betöltéséről, ha szükséges az `install.packages("csomagnév")` parancs segítségével telepítsük őket.

```
library(ggplot2)      # install.packages("ggplot2")
library(gridExtra)    # install.packages("gridExtra")
library(tidyr)        # install.packages("tidyr")
library(tibble)       # install.packages("tibble")
library(scales)       # install.packages("scales")
library(ggbeeswarm)   # install.packages("ggbeeswarm")
library(ggthemes)     # install.packages("ggthemes")
library(RColorBrewer) # install.packages("RColorBrewer")
library(ggrepel)      # install.packages("ggrepel")
library(showtext)     # install.packages("showtext")
```

```
library(ggh4x)         # install.packages("ggh4x")
library(LaplacesDemon)# install.packages("LaplacesDemon")
library(rio)           # install.packages("rio")
library(psych)         # install.packages("psych")
```

9.1. A ggplot2 alapjai 🙂

> Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük a ggplot2 rendszer alapelveit,
> - és a legalapvetőbb ábratípusok létrehozását.

9.1.1. Alapelvek

A ggplot2 grafika teljesen más megközelítést használ az ábrák létrehozásához, mint az *Alap R* grafikus rendszerei. Nevét Leland Wilkinson *Grammar of Graphics* (Wilkinson, 2005) könyve után kapta, amely a grafika általános formális és strukturált leírására tett kísérletet. A ggplot2 grafika a Hadley Wickham által készített `{ggplot2}` csomagban érhető el. A ggplot2 ábrák létrehozása némi tanulás után átlátható és következetes lesz, maguk az elkészült grafikák gyönyörűek és alkalmasak azonnali publikálásra.

Egy ggplot2 grafika három összetevőn alapul:

- *adat rész (data)*: Minden ábra egy adattáblában alapul (*data frame* vagy *tibble*). Az adatoszlopok ennek megfelelően többnyire faktorok vagy numerikus vektorok.
- *geometria objektumok (geometric objects, geoms)*: Az ábrán megjelenő geometriai elemek, például a pont, az oszlop (téglalap) vagy a vonal. Ezek határozzák meg végső soron az ábra típusát, így létrehozhatunk például pontdiagramot, oszlopdiagramot, hisztogramot vagy vonal diagramot, de ezekből akár többet is felhasználhatunk, így lehetnek különböző rétegei ugyanannak az ábrának.
- *esztétikai leképezések (aesthetic mapping)*: A leképezés lényegében összerendelés az adatbázis egyes változói és az ábrán megjelenő geometriai elemek paraméterei között. Ilyen lehetséges geom paraméter például pont geom esetén az x és y koordináta, a szín, a méret vagy az alak, oszlop geom esetén például a magasság és a kitöltő szín.

Az ábrák létrehozását a `ggplot()` függvénnyel kezdjük, amely rendszerint tartalmazza az első két összetevőt (adat és leképezés), és `+` operátorral adjuk ehhez a geom elemeket. Az általános forma a következő:

ggplot2 használata (egyszerű ábrák)

```
ggplot(data = <ADAT>, mapping = aes(<LEKÉPEZÉS>)) + <GEOM_FÜGGVÉNY>()
```

Természetesen az ábra számos összetevőjét (például feliratok, tengelyek, rácsos elrendezés ténye, jelmagyarázat stb.) módosíthatjuk, és ezt ugyancsak a + operátor után megjelenő függvényekkel tehetjük meg. Rendszerint az ábránkat objektumban tároljuk, és vagy a képernyőn jelenítjük meg, vagy a háttértárra írjuk ki a `ggsave()` függvény segítségével. A szokásos munka a ggplot2 rendszerben tehát a következő:

ggplot2 használata (összetett ábrák)

```
# 1. lépés: ábra létrehozása objektumban
p1 <- ggplot(data = <ADAT>, mapping = aes(<LEKÉPEZÉS>)) +
  <GEOM_FÜGGVÉNY_1>() + ... + <GEOM_FÜGGVÉNY_N>() +
  <MÓDOSÍTÓ_1>() + ...  + <MÓDOSÍTÓ_N>()
# 2. lépés: megjelenítés a képernyőn
p1
# 3. lépés: png fájl írása háttértárra
ggsave(filename = "<KEP_NEVE>.png", plot = p1)
```

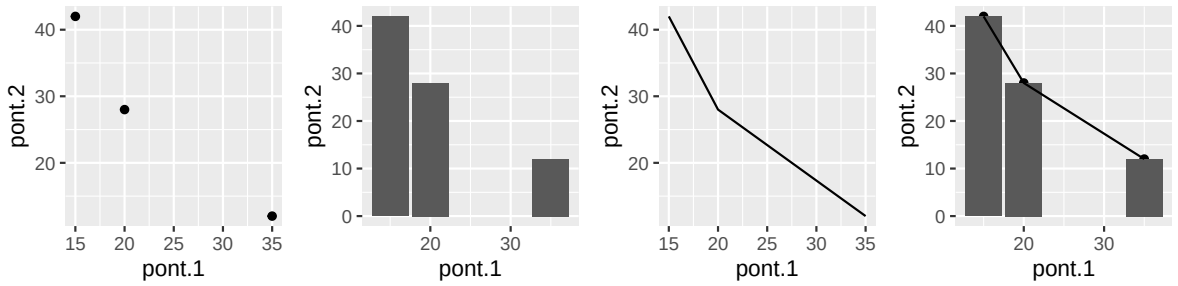
A ggplot2 megismeréséhez nézzünk egy egyszerű példát! Hozzunk létre egy adatbázist és négy ábrát, majd jelenítsük meg őket. Az adatbázist a könnyebb áttekinthetőség kedvéért inline módon hozzuk létre (6.2.2. fejezet).

```
# data frame létrehozása, 3 sor, 4 oszlop
df <- read.table(header=T, stringsAsFactors=T, text="
  csoport pont.1 pont.2 pont.3
       AA     15     42     12
       BB     20     28     18
       CC     35     12     21
")
```

Az adatbázist létrehozó fenti parancs egy 3 sorból és 4 oszlopból álló adattáblát (`df`) hoz létre, amely egy faktorrá konvertált karakteres (`csoport`) és három numerikus oszlopot (`pont.1`, `pont.2`, `pont.3`) tartalmaz. Ez adja a rajzparancsaink *adat* részét.

Az áttekinthetőbb megjelenítés kedvéért a továbbiakban mindig a `{gridExtra}` csomag `grid.arrange()` függvényét használjuk fel, amely képes egyszerre több ábrát egymás mellett (vagy más tetszőleges elrendezésben) megjeleníteni.

```
# p1 - pontdiagram
p1 <- ggplot(data=df, mapping=aes(x=pont.1, y=pont.2)) + geom_point()
# p2 - oszlopdiagram
p2 <- ggplot(data=df, mapping=aes(x=pont.1, y=pont.2)) + geom_col()
# p3 - vonaldiagram
p3 <- ggplot(data=df, mapping=aes(x=pont.1, y=pont.2)) + geom_line()
# p4 - pontdiagram, oszlopdiagram és vonaldiagram egy ábrán
p4 <- ggplot(data=df, mapping=aes(x=pont.1, y=pont.2)) +
  geom_point() + geom_col() + geom_line()
grid.arrange(p1, p2, p3, p4, ncol=4) # ábrák megjelenítése
```



Mint említettük a legtöbb ggplot2 ábra létrehozása a `ggplot()` függvénnyel indul, ahol a `data=` argumentum az adat részt tartalmazza, a `mapping=` argumentum pedig a leképezés során használt geom paramétereket és az adat részben definiált adatbázis változóit rendeli össze. Utóbbihoz az `aes()` függvényt használjuk, amely az adatoszlopok és a megjelenést meghatározó geom paraméterek közötti kapcsolatot adja meg. Eddig a pontig az ábráinkat létrehozó fenti 4 rajzparancs megegyezik. A geom elemeket a `+` operátorral adjuk hozzá a `ggplot()` függvényhez, ezek rendre a pontelemeket létrehozó `geom_point()`, az oszlopelemeket definiáló `geom_col()`, és a vonalelemekért felelős `geom_line()`. Az utolsó rajzparancsban mindhárom geom elem szerepel, ugyanis az egyes `geom_*()` függvénnyel létrehozott geom elemek külön rétegre kerülnek, így egyszerre is megjeleníthetők.

Az első ábra létrehozásához a `ggplot()` függvényt és a pontelemek hozzáadásához a `geom_point()` függvényt használtuk, az utóbbiban nincs is szükség argumentumok megadására. A `ggplot()` első argumentuma a `data=df`, amely az adatösszetevőt tisztázta, míg a `mapping=` argumentumban az `aes()` függvényben az `x` és `y` geom paraméterekről gondoskodtunk az `x=pont.1` és `y=pont.2` adatbázisváltozók megadásával. Az `x` és `y` geom paraméterek természetesen a hozzáadott pontok x és y koordinátáira vonatkoznak.

A második ábrán a pontelemek helyett oszlopelemeket használtunk a `geom_col()` geom függvény segítségével. Ekkor az `x` és `y` geom paraméterek az oszlop x koordinátáját és az oszlop magasságát jelentik. A harmadik ábrán a `geom_line()` geom elemet használtuk, és az `x` és `y`

paraméterek a vonalak létrehozásához használt pontok x és y koordinátái. A negyedik ábrán mindhárom ábraelem (pont, oszlop, vonal) felvonul.

Az ábra létrehozásakor használt `geom_*()` geom függvény részben meghatározza azokat a paramétereket, amelyeket az `aes()` függvénnyel definiált leképezésben használni fogunk, használnunk kell. Vannak ugyanis kötelező és opcionális geom paraméterek (9.1. táblázat). A kötelező geom paraméterek megadása nélkül hibaüzenetet kapunk a rajzparancs kiadása után, ugyanis ezek feltétlenül szükségesek az adott geom elem megjelenítéséhez.

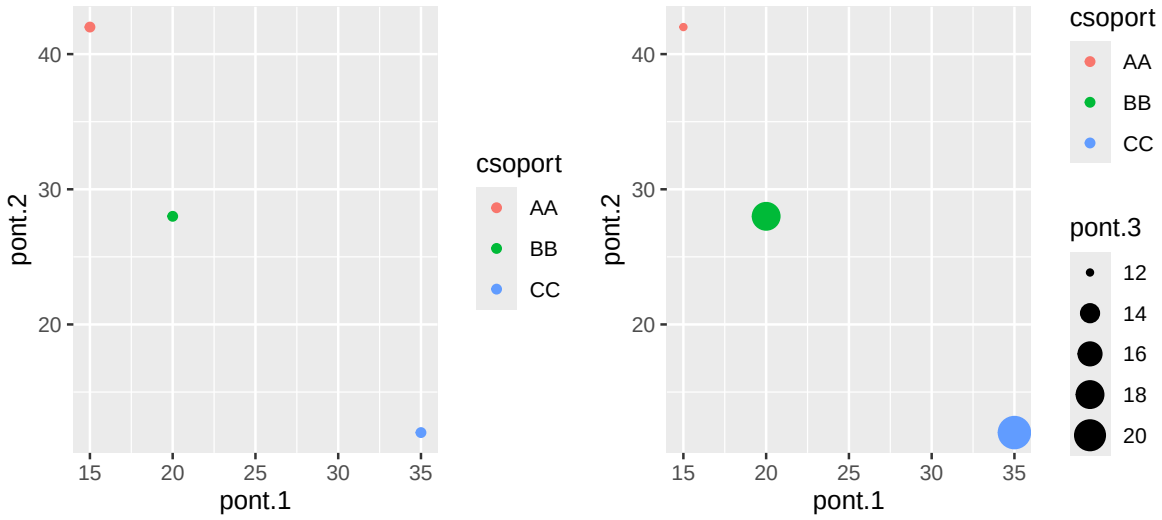
9.1. táblázat: A `geom` függvények a kötelező és opcionális paraméterekkel *(saját szerkesztés)*

Ábra elemek	Kötelező geom paraméter	Opcionális geom paraméter
`geom_abline()`	`slope`, `intercept`	`alpha`, `color`, `linetype`, `linewidth`
`geom_hline()`	`yintercept`	`alpha`, `color`, `linetype`, `linewidth`
`geom_vline()`	`xintercept`	`alpha`, `color`, `linetype`, `linewidth`
`geom_area()`	`x`, `ymin`, `ymax`	`alpha`, `colour`, `fill`, `group`, `linetype`, `linewidth`
`geom_col()`	`x`, `y`	`alpha`, `colour`, `fill`, `group`, `linetype`, `linewidth`
`geom_bar()` `geom_bar()`	`x`, `y` `x`, `y`	`alpha`, `colour`, `fill`, `group`, `linetype`, `linewidth`
`geom_boxplot()`	`x`, `lower`, `middle`, `upper`, `ymax`, `ymin`	`alpha`, `color`, `fill`, `group`, `linetype`, `shape`, `linewidth`, `weight`
`geom_density()`	`x`, `y`	`alpha`, `color`, `fill`, `group`, `linetype`, `linewidth`, `weight`
`geom_dotplot()`	`x`, `y`	`alpha`, `color`, `fill`, `group`, `linetype`, `stroke`
`geom_histogram()`	`x`	`alpha`, `color`, `fill`, `linetype`, `linewidth`, `weight`
`geom_jitter()`	`x`, `y`	`alpha`, `color`, `fill`, `shape`, `size`
`geom_line()`	`x`, `y`	`alpha`, `color`, `linetype`, `linewidth`
`geom_point()`	`x`, `y`	`alpha`, `color`, `fill`, `shape`, `size`
`geom_ribbon()`	`x`, `ymax`, `ymin`	`alpha`, `color`, `fill`, `linetype`, `linewidth`
`geom_smooth()`	`x`, `y`	`alpha`, `color`, `fill`, `linetype`, `linewidth`, `weight`
`geom_text()`	`label`, `x`, `y`	`alpha`, `angle`, `color`, `family`, `fontface`, `hjust`, `lineheight`, `size`, `vjust`

A 9.1. táblázatból kiolvasható, hogy az általunk használt `geom_point()`, `geom_col()` és `geom_line()` geom függvényekben az `x` és `y` geom paraméterek megadása kötelező, ezért nem

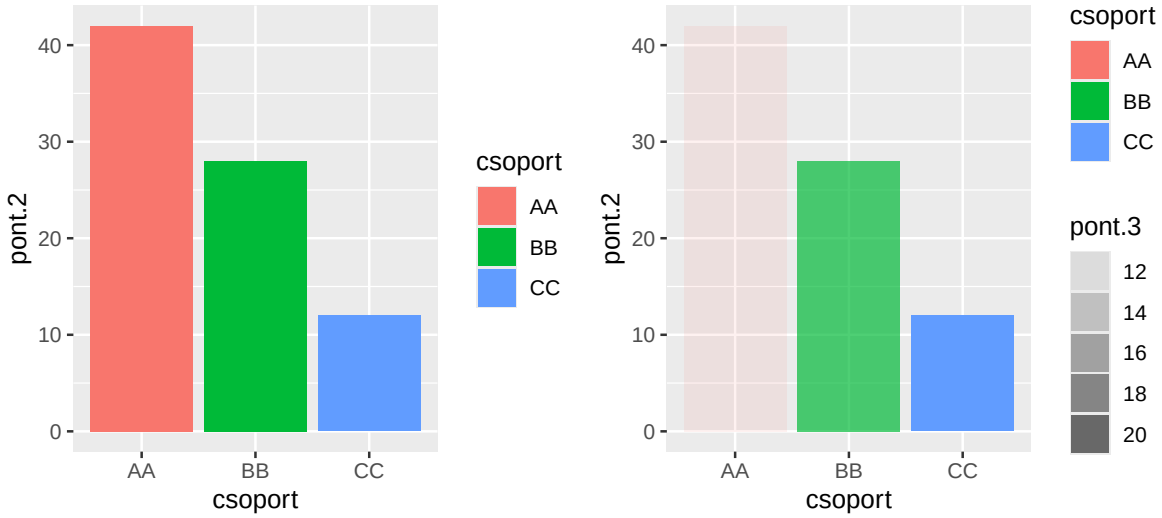
is felejtkeztünk el róluk a rajzparancsainkban. Azonban más, opcionális geom paramétereket is használhatunk. Ha a például a szín és a pontméret jellemzőket is be akarjuk állítani, akkor az `aes()` függvényben adjunk meg a `colour=` és a `size=` argumentumoknak egy-egy adattábla-változót. Ilyen esetekben jelmagyarázat is megjelenik, amelynek az alapértelmezett pozíciója az ábra jobb oldala (ezért nem kell megadnunk a következő módosító parancsot: `theme(legend.position = "right")`).

```
# p1 - pontdiagram, színezett pontokkal
p1 <- ggplot(df, aes(x = pont.1, y = pont.2, colour = csoport)) + geom_point()
# p2 - pontdiagram, színezett és átméretezett pontokkal
p2 <- ggplot(df, aes(x = pont.1, y = pont.2, colour = csoport, size = pont.3)) +
    geom_point()
grid.arrange(p1, p2, ncol = 2)  # ábrák megjelenítése
```



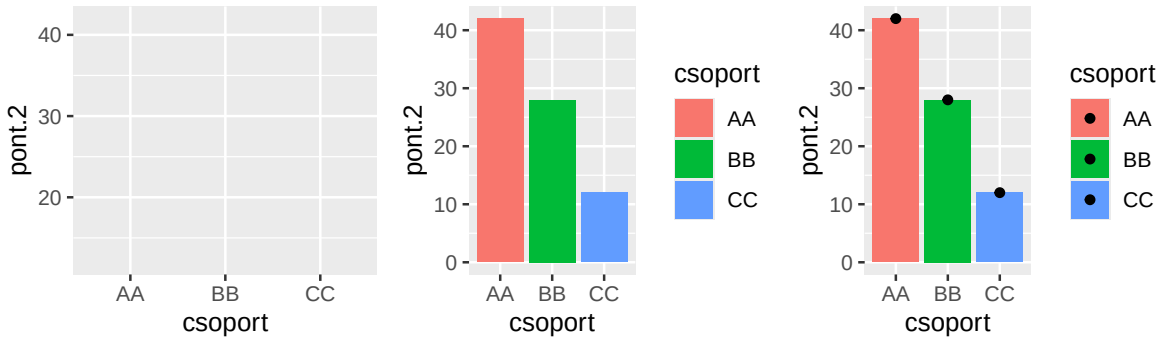
Oszlopdiagram esetén használhatjuk például a `fill=` vagy az `alpha=` argumentumokat, amelyekkel az egyes oszlopok kitöltőszínét és átlátszóságát tudjuk beállítani.

```
# p1 - oszlopdiagram kitöltőszín megadásával
p1 <- ggplot(df, aes(x = csoport, y = pont.2, fill = csoport)) + geom_col()
# p2 - oszlopdiagram kitöltőszín és átlátszóság megadásával
p2 <- ggplot(df, aes(x = csoport, y = pont.2, fill = csoport, alpha = pont.3)) +
    geom_col()
grid.arrange(p1, p2, ncol = 2)  # ábrák megjelenítése
```



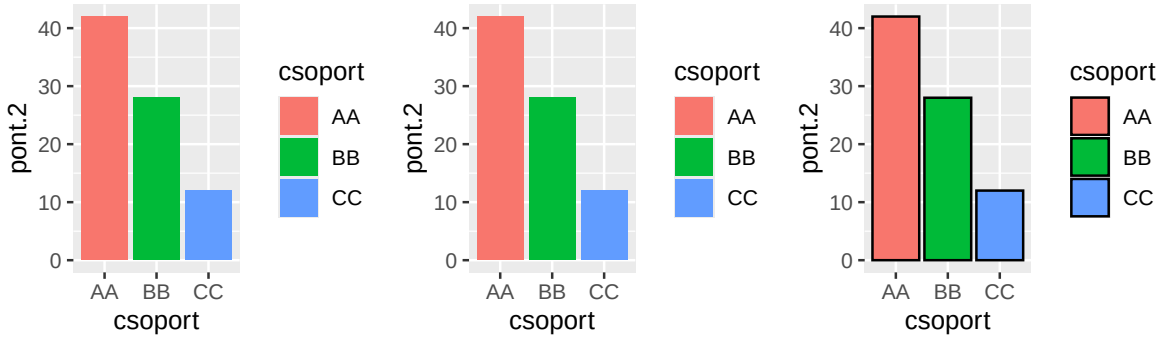
Korábban említettük, hogy a ggplot2 ábrák több geom réteget is tartalmazhatnak. Valójában az 1. rétegen mindig az adatbázis és a leképezés szerepel, amely már önmagában meghatározza például a feliratokat és a tengelyeket. A 2. és a további rétegeken az egyes geom elemek szerepelnek. A következő példában három olyan ábrát jelenítünk meg, melyeken rendre egy, kettő és három réteg szerepel.

```
# p1 - csak az 1. réteg: a tengelyek és a feliratok
p1 <- ggplot(df, aes(x=csoport, y=pont.2, fill=csoport))
# p2 - a 2. réteg is megjelenik, amely egy oszlopdiagramot tartalmaz
p2 <- ggplot(df, aes(x=csoport, y=pont.2, fill=csoport)) +
  geom_col()
# p3 - a 2. és 3. réteg is megjelenik, amely egy oszlopdiagramot és
# egy pontdiagramot is tartalmaz
p3 <- ggplot(df, aes(x=csoport, y=pont.2, fill=csoport)) +
  geom_col() + geom_point()
grid.arrange(p1, p2, p3, ncol=3) # ábrák megjelenítése
```



Utolsó általános megjegyzés, hogy az egyes rétegek valójában saját adatrésszel és leképezés résszel is rendelkezhetnek, azaz például a `geom_col()` függvényben saját `data=` és `mapping=` argumentumot is megadhatunk. A fenti példákban nem szerepeltek ezek az argumentumok, így a `ggplot()` függvényben definiáltakat vette át az adott réteg, vagyis a réteget definiáló `geom_col()` függvény.

```
# p1 - adat és leképezés a ggplot()-ban
p1 <- ggplot(data = df, mapping = aes(x = csoport, y = pont.2, fill = csoport)) +
    geom_col()
# p2 - adat és leképezés a geom_col()-ban
p2 <- ggplot() + geom_col(data = df, mapping = aes(x = csoport, y = pont.2,
    fill = csoport))
# p3 - adat a ggplot()-ban, leképezés a geom_col()-ban, plusz egy konstans
# keretszín megadás
p3 <- ggplot(df) + geom_col(mapping = aes(x = csoport, y = pont.2, fill = csoport),
    colour = "black")
grid.arrange(p1, p2, p3, ncol = 3)  # ábrák megjelenítése
```



A fenti 1. és 2. ábra minden pontban megegyezik, látható, nincs jelentősége, hogy a `ggplot()` függvényben vagy a réteget meghatározó `geom_col()` függvényben adjuk meg az adatrészt és a leképezést. A 3. ábrát létrehozó függvényben figyeljük meg, hogy a geom paraméterek konstans értékre is állíthatók, ekkor azonban nem szabad az leképezés definiáláshoz az `aes()` függvényt használnunk. A 3. ábrán az oszlopok keretszínét állítottuk be (`colour="black"`).

Az alapok ismertetése után ízelítőül adunk néhány példát az ábratípusokra és a létrehozásukhoz szükséges geom elemekre (9.1 ábra).

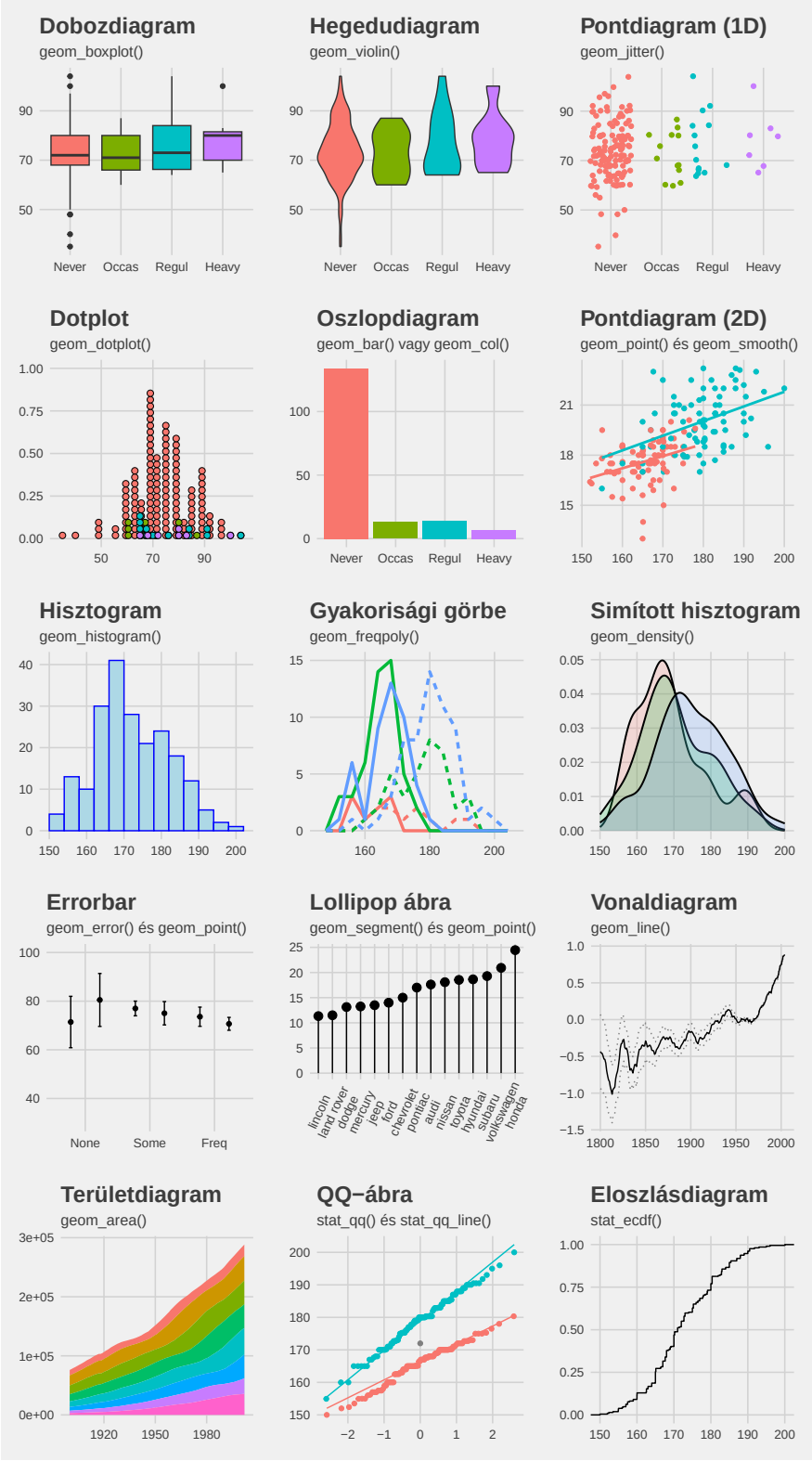
Látjuk, a ggplot2 rendszer számos geom elemet kínál, számos ábratípus hozható létre ezek segítségével. A ggplot2 ábrák rajzolásának és beállításainak részletes ismertetése egy külön könyvet megtöltene. A következő táblázatban újra kiemelünk néhány ábratípust, bemutatjuk a kötelező paraméterek nevét és azok típusát is. A típusra vonatkozó információ az adatelemzés során jut kulcsszerephez. A 9.2. táblázatban láthatjuk, melyek a numerikus és melyek a faktor adattípust megjelenítő geom függvények.

9.2. táblázat: Ábratípusok és `geom` elemek a kötelező paraméterekkel *(saját szerkesztés)*

Ábra típusa	Geom elem	Szokásos kötelező változók
Dobozdiagram	`geom_boxplot(y)`	1 numerikus
Hisztogram	`geom_histogram(x)`	1 numerikus
Simított hisztogram	`geom_density(x)`	1 numerikus
Gyakorisági poligon	`geom_freqpoly(x)`	1 numerikus
1D pontdiagram	`geom_dotplot(x)`	1 numerikus
QQ-ábra	`geom_qq(sample)`	1 numerikus
QQ vonal	`geom_qq_line(sample)`	1 numerikus
Eloszlás-görbe	`stat_ecdf(x)`	1 numerikus
2D pontdiagram	`geom_point(x, y)`	2 numerikus
Vonaldiagram	`geom_line(x, y)`	2 numerikus
Oszlopdiagram	`geom_bar(x)`	1 faktor
Oszlopdiagram	`geom_col(x, y)`	1 faktor (`x`), 1 numerikus (`y`)

9.1.2. 1D pontdiagram

Egydimenziós pontdiagramokat legtöbb esetben kvantitatív változók eloszlásának vizsgálatára használjuk. A nyers pontértékek megjelenítésével képet kaphatunk a minta terjedelméről és a fontosabb sűrűsödési helyeiről. Folytonos, azaz sok különböző értékkel rendelkező numerikus változók esetén is kiváló adatbetekintő eszköz (9.1.2.1. fejezet), de diszkrét, tipikusan kevesebb egymástól eltérő értékekkel rendelkező változók esetén is értékes eszköz (9.1.2.2. fejezet).



9.1. ábra: Ábrátípusok és geom elemek

9.1.2.1. Folytonos kvantitatív változók

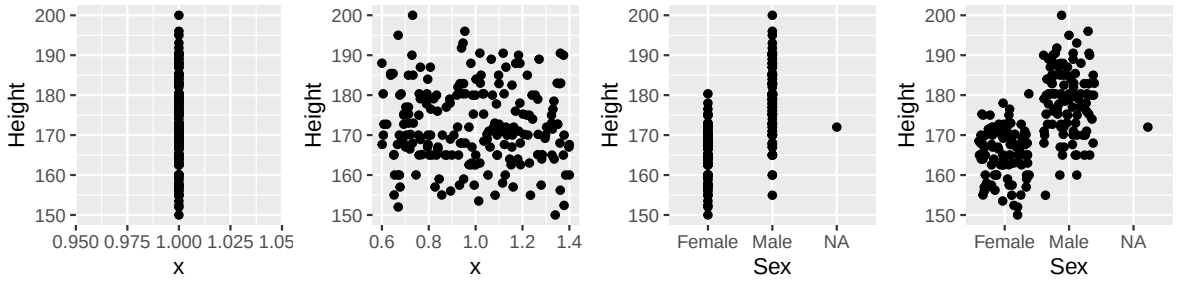
Példa 9.1 (Ausztrál egyetemisták)**.** Egy felmérésben 237 ausztrál egyetemista adatát gyűjtötték össze. Többek között nemre (`Sex`), kezességre (`W.Hnd`), testmozgásra (`Exer`), dohányzási szokásra (`Smoke`), testmagasságra (`Height`), kézméretre (`Wr.Hnd`) és a pulzusra (`Pulse`) vonatkozó információt gyűjtöttek. Az adatokat a `kerdoiv.xlsx` adatbázis tartalmazza.
Forrás: Venables és Ripley (2002)

Végezzük el a beolvasást és a nem változó (`Sex`) faktorrá alakítását. A továbbiakban a testmagasság változót (`Height`) vizsgáljuk, amely folytonos kvantitatív változó.

```
# adatok beolvasása
kerdoiv <- rio::import(file = "adat/kerdoiv.xlsx")
kerdoiv$Sex <- factor(kerdoiv$Sex)  # konvertálás faktorrá
psych::headTail(kerdoiv[, c(1:6)])
#>         Sex Wr.Hnd NW.Hnd W.Hnd   Fold Pulse
#> 1    Female   18.5     18 Right R on L    92
#> 2      Male   19.5   20.5  Left R on L   104
#> 3      Male     18   13.3 Right L on R    87
#> 4      Male   18.8   18.9 Right R on L  <NA>
#> ...    <NA>    ...    ...  <NA>   <NA>   ...
#> 234 Female   18.5     18 Right L on R    88
#> 235 Female   17.5   16.5 Right R on L  <NA>
#> 236    Male     21   21.5 Right R on L    90
#> 237 Female   17.6   17.3 Right R on L    85
```

A testmagasság nyers adatpontjait a `geom_point()` vagy a `geom_jitter()` segítségével rajzolhatjuk ábrára. Ahogyan a 9.1 táblázatból kiolvashatjuk, mindkét függvénynek kötelező paramétere az `x=` és az `y=`, amelyek a pontok helyét határozzák meg a koordináta rendszerben. Egydimenziós pontdiagram esetén tipikusan az `x=1` vagy az `x=<faktor>` megoldást választjuk az első argumentumnak, mert a vizsgált változót az `y=<numerikus vektor>` paraméterrel rögzítjük. Esetünkben a `y=Height` paramétert használjuk. Vizsgáljuk meg ezzel a módszerrel az ausztrál egyetemisták (9.1. példa) testmagasságát.

```
p1 <- ggplot(data = kerdoiv, mapping = aes(x = 1, y = Height)) + geom_point()
p2 <- ggplot(data = kerdoiv, mapping = aes(x = 1, y = Height)) + geom_jitter()
p3 <- ggplot(data = kerdoiv, mapping = aes(x = Sex, y = Height)) + geom_point()
p4 <- ggplot(data = kerdoiv, mapping = aes(x = Sex, y = Height)) + geom_jitter()
grid.arrange(p1, p2, p3, p4, nrow = 1)
```



A fenti ábrákon látható, hogy a `geom_jitter()` a `geom_point()` elemhez képest mindössze véletlen *x* és *y* irányú elmozdulást ad az eredeti koordinátákhoz, ezzel csökkenti az adatpontok átfedését.

A fenti ábrán látható, hogy a nem (`Sex`) faktor tartalmaz hiányzó értéket, így ez a kategória is megjelenik az *x* tengelyen. A hiányzó érték mint kategória megjelenítését kétféle módszerrel kerülhetjük el: az adatbázis manipulációjával, vagy ggplot2 módosítókkal. A következő példa 1. ábrájában a `kerdoiv |> tidyr::drop_na(Sex) |> ggplot()` sor már a `ggplot()` függvénybe olyan adatbázist küld, amelyben nincsenek a nem oszlopban hiányzó értékek (7.2.11. fejezet). A másik módszer a `scale_x_discrete(na.translate = F)` módosító használata. A következő példa 2-4. ábráiban ezt a módszert használjuk a hiányzó nem érték eltávolítására.

```
p1 <- kerdoiv |> tidyr::drop_na(Sex) |>
  ggplot(mapping = aes(x = Sex, y = Height)) +
  geom_point(position = position_jitter(width = 0.1, height = 0))
p2 <- ggplot(data=kerdoiv,
             mapping = aes(x = Sex, y = Height), alpha = 0.5) +
  geom_jitter(width = 0.1, height = 0) +
  scale_x_discrete(na.translate = F)
p3 <- ggplot(data=kerdoiv,
             mapping = aes(x = Sex, y = Height, colour = Sex)) +
  geom_jitter(width = 0.1, height = 0, alpha=0.5) +
  scale_x_discrete(na.translate = F) + theme(legend.position = "none")
p4 <- ggplot(data=kerdoiv,
             mapping = aes(x = Sex, y = Height, colour = Sex)) +
  geom_quasirandom(show.legend=F) + scale_x_discrete(na.translate = F) +
  theme_bw()
grid.arrange(p1, p2, p3, p4, nrow=1)
```

A fenti példa 1. ábrájából az is kiolvasható, hogy `geom_point()` függvény is paraméterezhető úgy, hogy visszakapjuk belőle a `geom_jitter()` hatását (`position = position_jitter(width = 0.1, height = 0)`). A `width=` és a `height=` megadásával a véletlen x és y irányú elmozdulás mértékét szabályozhatjuk. Vegyük észre, hogy az y elmozdulást most letiltottuk (`height = 0`).

A vizsgált numerikus változónk (`Height`) sűrűsödési pontjainak még jobb kitapintása érdekében a pontok átlátszóságát is beállíthatjuk (2-4. ábrák). Ennek mértékét most nem adatbázisváltozótól tesszük függővé, hanem fixre állítjuk, ezért nem a leképezés részben, az `aes()` függvényben szerepeltetjük, hanem azon kívül, például a `ggplot()` vagy a `geom_jitter()` függvényben.

A fenti 3-4. ábra színezését a leképezésben használt `colour=Sex` beállítással értük el, az ilyenkor automatikusan megjelenő jelmagyarázatot a `theme(legend.position = "none")` módosítóval vagy a `show.legend=F` paraméterrel nyomhatjuk el.

A 4. ábrán a `{ggbeeswarm}` csomag `geom_quasirandom()` geom elemét használtuk, amely a `geom_jitter()` függvényhez hasonlóan, de annál nagyobb mértékben csökkenti a pontok egymásra rajzolását. Továbbá a `theme_bw()` módosító függvénnyel az ábra kinézetén is változtattunk.

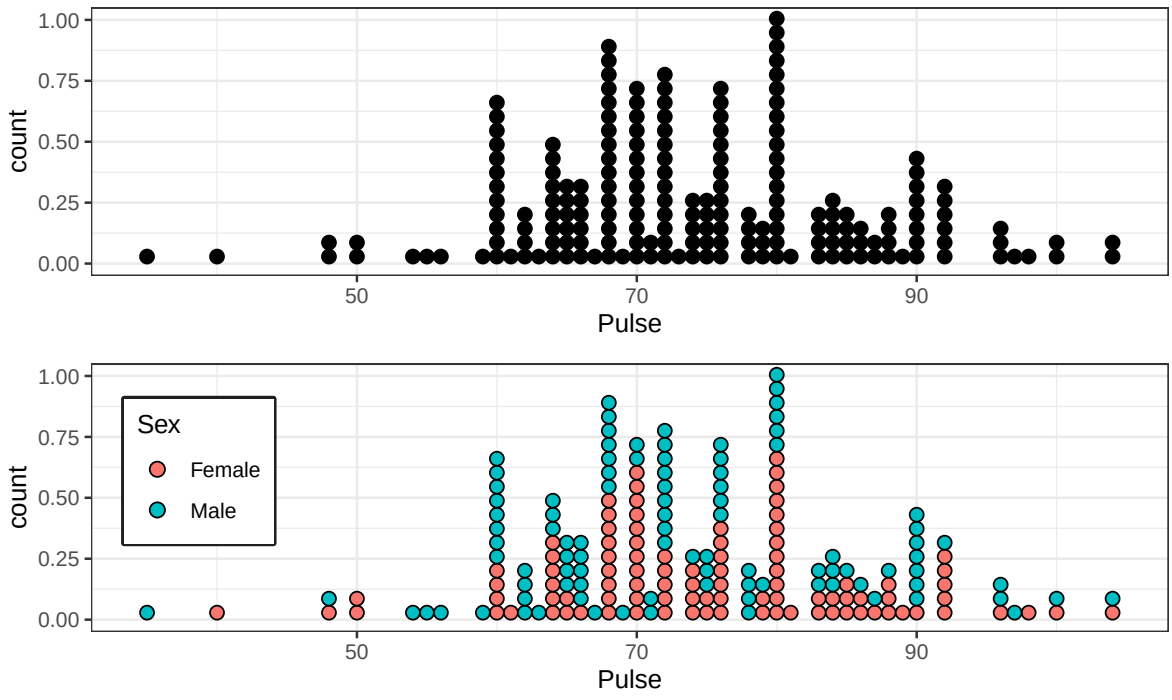
Megjegyezzük, hogy a `theme_bw()` módosításait a további ábrákon alapértelmezettként állítjuk be. Ezt a `theme_set()` függvény segítségével tehetjük meg. Így elkerüljük, hogy a `theme_bw()` módosítót minden esetben szerepeltessük a rajzparancsainkban.

```
# a theme_bw() módosításainak globális érvényesítése
old.theme <- theme_set(theme_bw())
# a korábbi állapot visszaállítása, ha erre szükség lenne
# theme_set(old.theme)
```

9.1.2.2. Diszkrét kvantitatív változók

Vizsgáljuk meg a 237 ausztrál egyetemista pulzusát (9.1. példa). A pulzus diszkrét kvantitatív változó, így várhatóan a mintában előfordulnak ismétlődések. Ebben az esetben a változó eloszlását a `geom_dotplot()` geom elemmel vizsgálhatjuk, de természetesen az előző, 9.1.2.1 fejezet `geom_point()`, `geom_jitter()` vagy `ggbeeswarm::geom_quasirandom()` geom elemei ugyanúgy rendelkezésre állnak.

```
p1 <- ggplot(data = kerdoiv, mapping = aes(x = Pulse)) +
  geom_dotplot(binwidth = 1)
p2 <- ggplot(data = kerdoiv, mapping = aes(x = Pulse, fill=Sex)) +
  geom_dotplot(binwidth = 1, stackgroups = T) +
  scale_fill_discrete(na.translate = F) +
  theme(legend.position = c(0.1, 0.6),
        legend.box.background = element_rect(colour = "grey12",
                                             linewidth = 1.2))
grid.arrange(p1, p2, ncol=1)
```

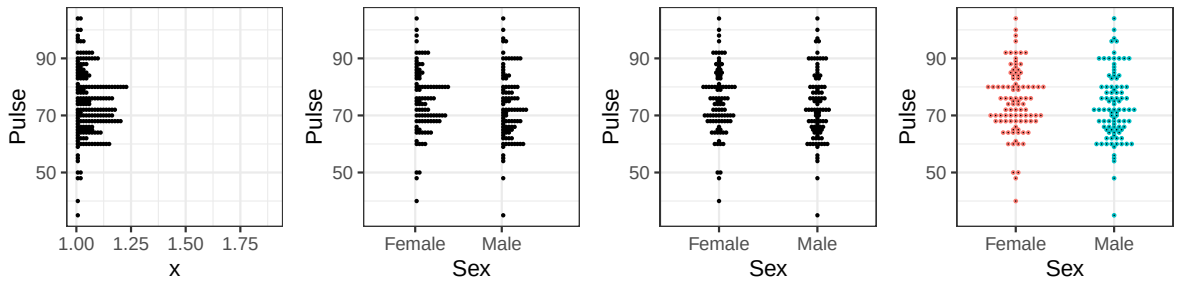


Diszkrét kvantitatív változók esetében elképzelhető az adatok egymásra halmozása, ezt mutatja be a fenti két ábra. A `binwidth=1` paraméterrel biztosítjuk, hogy minden adatpont

különálló legyen, lényegében 1 hosszú osztályintervallumokkal számolunk. Az alsó ábrán a nem (`Sex`) változó megjelenítéséről is gondoskodunk. Csoportok helyes megjelenítéséhez a `stackgroups=T` paramétert is meg kell adnunk. A nem változóból a hiányzó értéket a szokásos módon távolítjuk el (`scale_fill_discrete(na.translate=F)`), a jelmagyarázatot pedig a rajzterületen jelenítjük meg.

Az egymásra halmozott pontok az y tengely mentén is megjeleníthetők. Ehhez a leképezésben az `y=<numerikus változó>` és az `x=1` vagy a `x=<faktor>` összerendeléseket kell használnunk. Pluszban a `binaxis = "y"` paramétert is meg kell adnunk a `geom_dotplot()` függvényben, így közöljük, hogy az y tengely mentén szeretnénk az értékek egymásra halmozását kezdeményezni.

```
p1 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=1, y = Pulse)) +
  geom_dotplot(binwidth = 1, binaxis = "y")
p2 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=Sex, y = Pulse)) +
  geom_dotplot(binwidth = 1, binaxis = "y")
p3 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=Sex, y = Pulse)) +
  geom_dotplot(binwidth = 1, binaxis = "y", stackdir = "center")
p4 <- kerdoiv |> drop_na(Sex) |>
  ggplot(aes(x=Sex, y = Pulse, colour=Sex)) +
  geom_dotplot(binwidth = 1, binaxis = "y", stackdir = "center",
               show.legend = F, stackratio=1.6, dotsize=1.1)
grid.arrange(p1, p2, p3, p4, nrow=1)
```



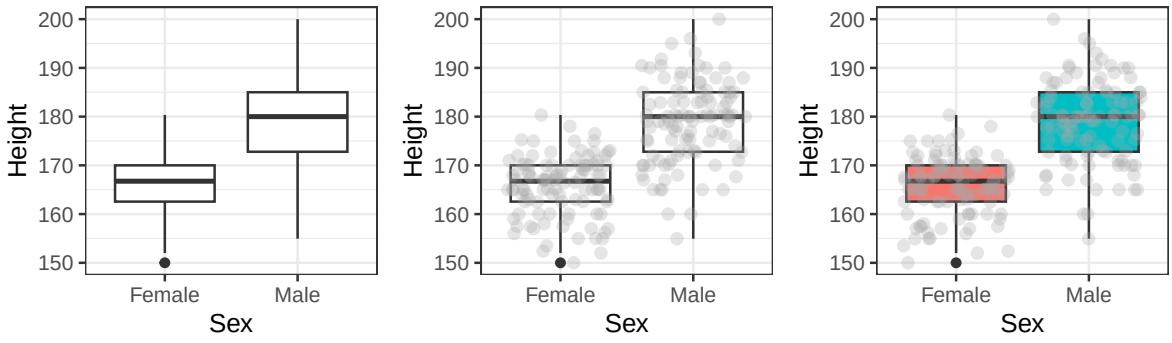
A fenti négy ábrán az y tengely értékeit halmoztuk egymásra. A 3-4. ábrán a `stackdir="center"` beállítással középre igazítottuk a pontokat, így az eloszlás könnyebben áttekinthető.

A 4. ábrán a `dotsize=` és a `stackratio=` paraméterekkel finomítottunk a pontok megjelenésén. A `dotsize=` paraméter a megjelenített pontok átmérőjét módosítja az osztályintervallum szélességéhez (`binwidth=`) viszonyítva. A `stackratio=` azt szabályozza, hogy milyen közel kell egymásra rajzolni a pontokat. Az alapértelmezés 1, ahol a pontok éppen érintkeznek. Kisebb értékek esetén átfedő pontokat kapunk.

9.1.3. Dobozdiagram

Az előző fejezetben bemutatott egydimenziós pontdiagram kiváló eszköz nyers adatok megjelenítésére, azonban sokszor van szükség származtatott adatok ábrázolására is, ilyen például a mintaátlag, medián, szórás, interkvartilis eltérés, minimum vagy maximum. Ez utóbbi ábrák sorába tartozik a dobozdiagram is, amellyel most a testmagasságot hasonlítjuk össze a nem két csoportjában (9.1. példa). Az 1. ábrán két réteg szerepel, az elsőn az adatok és a leképezésre vonatkozó információk, miszerint az x tengelyre a kétértékű `Sex` faktor kerül, az y tengelyre pedig a folytonos `Height` változó. A második rétegen már a dobozdiagramok is szerepelnek a `geom_boxplot()` függvény miatt. A 2. ábrára már három réteg került, a nyers pontokat is megjelenítjük a `geom_jitter()` segítségével, egyben ezek pontok méretét, színét és átláthatóságát is beállítjuk. A 3. ábrán pusztán a dobozoknak adtunk kitöltőszínt (`fill=Sex`).

```
# p1 - dobozdiagram
p1 <- ggplot(kerdoiv, aes(x = Sex, y = Height)) + geom_boxplot() +
  scale_x_discrete(na.translate = F)
# p2 - dobozdiagram nyers pontokkal
p2 <- ggplot(kerdoiv, aes(x = Sex, y = Height)) + geom_boxplot() +
  geom_jitter(size = 2, colour = 'darkgrey', alpha = 0.3) +
  scale_x_discrete(na.translate = F)
# p3 - dobozdiagram kitöltőszínnel és nyers pontokkal
p3 <- ggplot(kerdoiv, aes(x = Sex, y = Height, fill=Sex)) +
  geom_boxplot() + theme(legend.position = "none") +
  geom_jitter(size = 2, colour = 'darkgrey', alpha = 0.3) +
  scale_x_discrete(na.translate = F)
grid.arrange(p1, p2, p3, ncol=3)
```



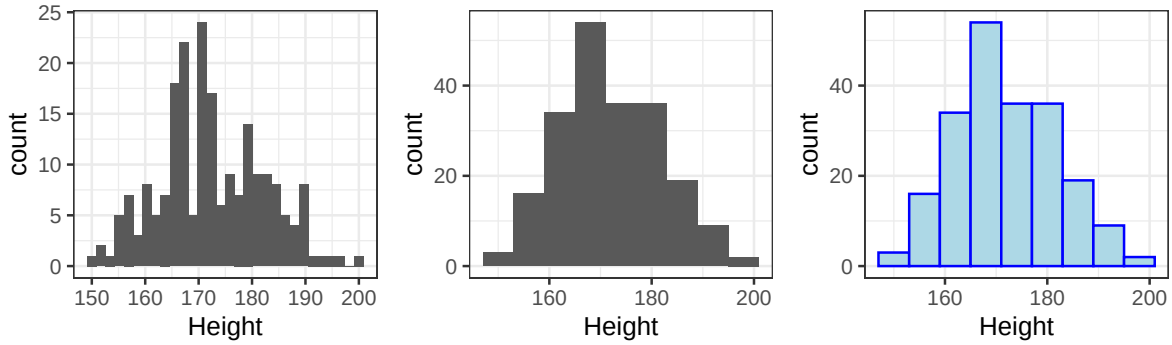
A dobozdiagram a három kvartilis, illetve a minimum és maximum megjelenítésével világossá teszi, hogy a mintában a férfiak magasabbak a nőknél.

A dobozdiagramok megjelenítése során kérdésként merülhet fel, hogy a származtatott adatok mikor kerülnek be a megjelenítendő adatok közé, hiszen az eredeti `kerdoiv` adatbázis nem tartalmazza a mediánt és a szélső értékeket sem. A megoldást a 9.2.1. fejezetben találjuk.

9.1.4. Hisztogram

Egy változó eloszlásának megismerése rendkívüli jelentőséggel bír. Numerikus változó esetében hisztogram segítségével megtudhatjuk, hogy az adatok hol, milyen sűrűn fordulnak elő, egyszerre kaphatunk információt a középértékről, a szóródásról és a kiugró értékekről. A hisztogram a dobozdiagramhoz hasonlóan származtatott információt jelenít meg.

```
# p1 - alapértelmezett hisztogram
p1 <- ggplot(kerdoiv, aes(x = Height)) + geom_histogram()
# p2 - hisztogram 15 (kg) széles csoportokkal
p2 <- ggplot(kerdoiv, aes(x=Height)) + geom_histogram(binwidth=6)
# p3 - hisztogram 15 (kg) széles csoportokkal és színezéssel
p3 <- ggplot(kerdoiv, aes(x=Height)) +
  geom_histogram(binwidth=6, colour="blue", fill="lightblue")
grid.arrange(p1, p2, p3, ncol=3)
```



9.2. ábra: Bevezető példák hisztogram megjelenítésére

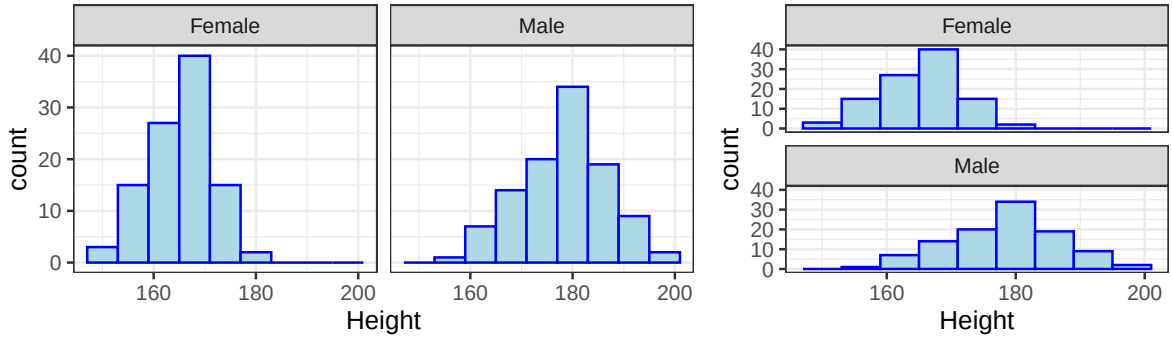
Az 1. ábrán már a `geom_histogram()` függvénynek köszönhetően megjelenik egy alapértelmezett kinézetű hisztogram, amelyet a 2. ábrán kicsit értelmezhetőbbé teszünk a `binwidth=15` argumentummal, amely a megjelenített intervallumok szélességét határozza meg. A 3. ábrán mindössze a színeket állítjuk be.

A ggplot2 rendszer nem ajánl optimális értéket az osztályintervallumok számára, így azt nekünk kell minden esetben a helyesnek vélt értékre beállítani. A következő számításokkal ajánlásokat kapunk az osztályintervallumok szélességére (Venables és Ripley (2002)). Láthatjuk, hogy esetünkben az 5 cm-es vagy a 6 cm-es osztályintervallum-hossz lehet a megfelelő.

```
# Hisztogram - osztályintervallumok hossza
x <- na.omit(kerdoiv$Height)
2 * IQR(x)/(length(x)^(1/3))  # Freedman-Diaconis formula
#> [1] 5.05521
3.49 * sd(x)/(length(x)^(1/3))  # Scott formula
#> [1] 5.791226
```

A fenti hisztogramok világossá teszik számunkra, hogy a testmagasság adatok hol, milyen sűrűn fordulnak elő, de igazán az lenne érdekes, ha ezt a két nem esetében külön-külön is láthatnánk. Rácsos megjelenítéssel részábrákra bonthatjuk a meglévő ábránkat, ehhez csak a `facet_wrap()` függvényben kell egy vagy több kategorikus változót megadni. A legegyszerűbb esetben, egyetlen faktor esetén, használjuk a `facet_wrap(~<faktor>)` alakú módosítót.

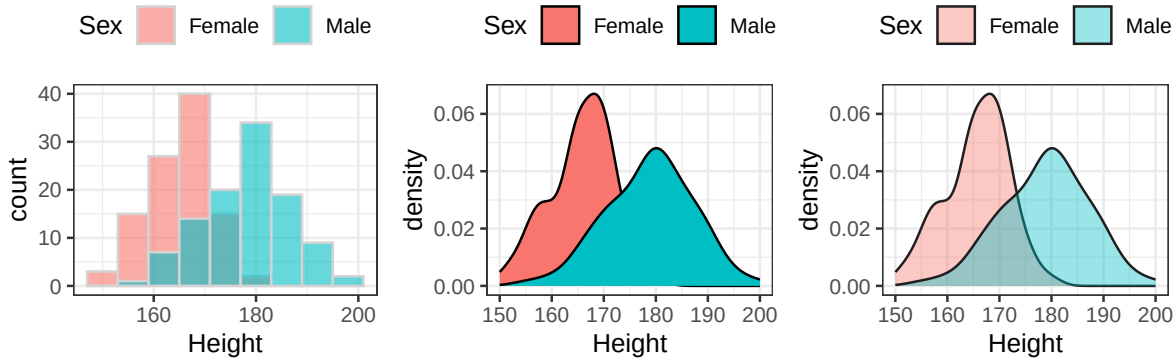
```
# p1 - alapértelmezett felbontás a Grazing faktor alapján
p1 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height)) +
  geom_histogram(binwidth = 6, colour="blue", fill="lightblue") +
  facet_wrap(~ Sex)
# p2 - a hisztrogram egyoszlopos felbontása a Grazing faktor mentén
p2 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height)) +
  geom_histogram(binwidth = 6, colour="blue", fill="lightblue") +
  facet_wrap(~ Sex, ncol=1)
grid.arrange(p1, p2, ncol=2, layout_matrix=cbind(1, 1, 1, 2, 2))
```



Mivel a `Sex` faktor mindössze két különböző értéket tartalmaz, így két részábra jön létre, melyek elrendezését az `nrow=` vagy `ncol=` argumentumok megadásával lehet szabályozni.

Hisztogram esetében használhatunk egy másik megoldást is a `Sex` faktor bevonására. Ha az adatok erre alkalmassá teszik, akkor a két csoport hisztogramját egyetlen ábrán is megjeleníthetjük. Egészítsük ki `position="identity"` argumentummal a hisztogramért felelős függvényt. További beállítási lehetőségek a `boundary=` (az egyik osztályintervallum végpontjának rögzítése) és a `closed=` (melyik oldal legyen zárt) argumentumokkal lehetségesek. Hasonló céllal hozunk létre simított hisztogramot, erre a `geom_density()` függvényt használhatjuk.

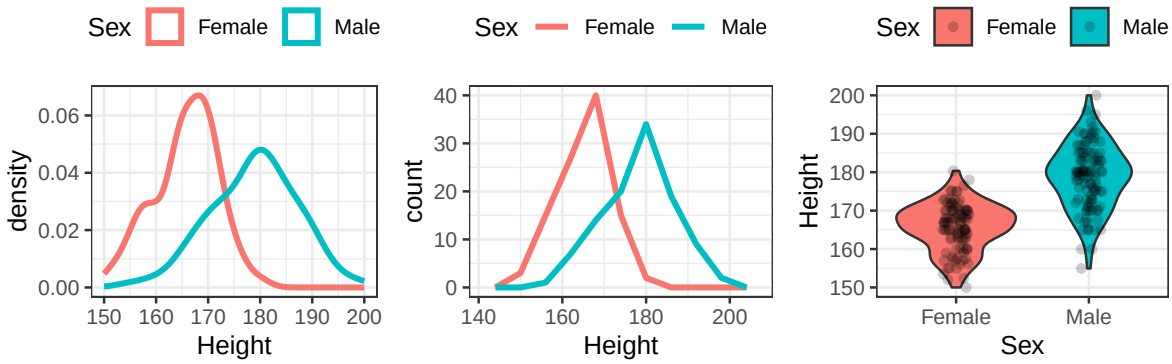
```
# p1 - két hisztogram egy ábrán
p1 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height, fill=Sex)) +
  geom_histogram(binwidth = 6, colour="lightgrey",
                 position="identity", alpha=0.6) +
  theme(legend.position = "top")
# p2 - két simított hisztogram egy ábrán
p2 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height, fill=Sex)) +
  geom_density() + theme(legend.position="top")
# p3 - két simított hisztogram egy ábrán
p3 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height, fill=Sex)) +
  geom_density(alpha=0.4, colour="gray12") +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, ncol=3)
```



A hisztogramról és a simított hisztogramról leolvasható információkhoz hasonlót szolgáltatnak a gyakorisági poligonok és a hegedűdiagramok is. Elsőként egy simított hisztogramot rajzolunk, amelynek most nem a kitöltését (`fill=`), hanem a vonalszínét határozzuk meg a `Sex` két szintje mentén (`colour=Sex`). Látható, hogy a következő két ábrán a gyakorisági

poligont a `geom_freqpoly()` a területdiagramot pedig a `geom_violin()` függvénnyel hoztuk létre. A hegedűdiagramra a nyers adatpontokat is rárajzoltuk a `geom_jitter()` segítségével.

```
# p1 - két simított hisztogram
p1 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height, colour=Sex)) +
  geom_density(linewidth=1.2) + theme(legend.position = "top")
# p2 - két gyakorisági poligon
p2 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x = Height, colour=Sex)) +
  geom_freqpoly(binwidth=6, linewidth=1.2) + theme(legend.position = "top")
# p3 - két területábra
p3 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=Sex, y = Height, fill=Sex)) +
  geom_violin() + geom_jitter(height = 0, width = 0.1, alpha=0.2) +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, ncol=3)
```

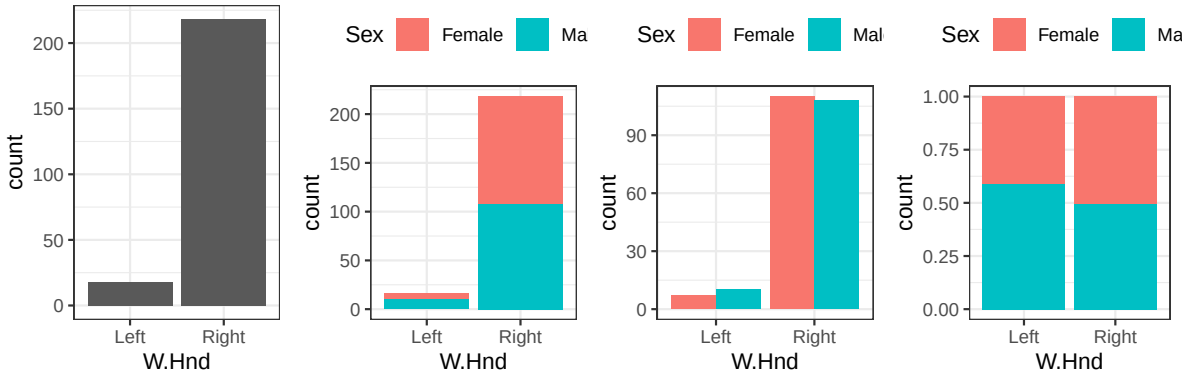


9.1.5. Oszlopdiagram

Az oszlopdiagram alapvetően kategorikus változó eloszlását jeleníti meg, de összesített numerikus értékek, például átlagok megjelenítésére is használhatjuk. Ez utóbbi lehetőségekről a 9.2.2. fejezetben lesz szó.

A kategorikus változók eloszlásának vizsgálatát a 9.1. példával kezdjük, és a nem (`Sex`), valamint a kezesség (`W.Hnd`) változókat elemezzük. A `kerdoiv` adatbázisból a `geom_bar()` geom függvény segítségével készíthetünk egy- vagy kétdimenziós oszlopdiagramokat.

```r
# oszlopdiagramok nyers adatokból
p1 <- kerdoiv |> drop_na(W.Hnd) |>
  ggplot(aes(x=W.Hnd)) + geom_bar()
p2 <- kerdoiv |> drop_na(W.Hnd, Sex) |>
  ggplot(aes(x=W.Hnd, fill=Sex)) + geom_bar() +
  theme(legend.position = "top")
p3 <- kerdoiv |> drop_na(W.Hnd, Sex) |>
  ggplot(aes(x=W.Hnd, fill=Sex)) + geom_bar(position="dodge") +
  theme(legend.position = "top")
p4 <- kerdoiv |> drop_na(W.Hnd, Sex) |>
  ggplot(aes(x=W.Hnd, fill=Sex)) + geom_bar(position="fill") +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



Az első ábra a kezesség (`W.Hnd`) egydimenziós oszlopdiagramja. Láthatjuk, a jobbkezesség az ausztrál egyetemisták között is gyakoribb. A 2-4. ábrák kétdimenziósak, vagyis 2 faktoron alapulnak (kezesség és nem). Oszlopdiagramok esetében az első faktort tipikusan a leképezés (`aes()`) `x=<faktor1>` paraméterében, a második faktort a `fill=<faktor2>` argumentumában adjuk meg. A 2-4. ábrák a `position=` paraméterben térnek el. A 2. ábra az alapértelmezett `position="stack"` szerint jelenik meg, azaz a gyakorisági értékeket a 2. faktor mentén egymásra halmozzuk. A 3. ábra `position="dodge"` beállítása miatt egymás melletti oszlopokban jeleníti meg a nem szerinti gyakoriságokat. A 4. ábra azonos magasságú oszlopokat jelenít meg (`position="fill"` beállítás), így relatív gyakoriságok összevetésére van lehetőségünk.

Fontos kiemelnünk, hogy a fenti oszlopdiagramok nyers adatok alapján jöttek létre. A másik lehetőség, hogy összesített, eleve gyakoriságokat tartalmazó adatmátrixot használunk. A 9.3. táblázat összefoglalja az oszlopdiagram rajzolásának leggyakoribb eseteit. Láthatjuk, hogy megkülönböztetünk egy és kétdimenziós oszlopdiagramokat, és az adatbázis is tartalmazhat nyers vagy összesített gyakorisági adatokat.

9.3. táblázat: Az oszlopdiagram `geom` paraméterezése *(saját szerkesztés)*

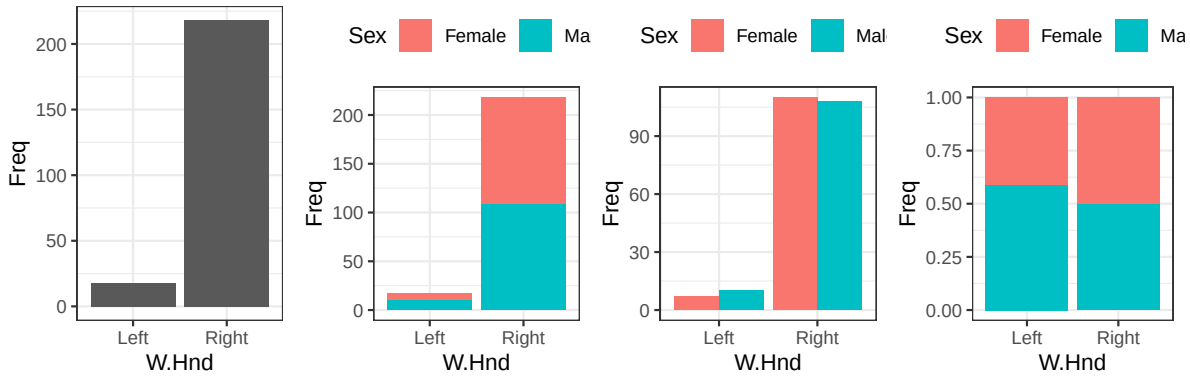
Dimenzió	Adatbázis formája	Geom függvény	Szokásos paraméterek
1D	nyers adatok	`geom_bar()`	`x=`
1D	összesített adatok	`geom_col()`	`x=, y=`
1D	összesített adatok	`geom_bar()`	`x=, weight=`
2D	nyers adatok	`geom_bar()`	`x=, fill=`
2D	összesített adatok	`geom_col()`	`x=, y=, fill=`
2D	összesített adatok	`geom_bar()`	`x=, weight=, fill=`

Összesített gyakorisági adatokat tartalmazó adatmátrixot magunk is létrehozhatunk (5.4.1.3. fejezet).

```
(tablazat.1D <- xtabs(~W.Hnd, data = kerdoiv))
#> W.Hnd
#>  Left Right
#>    18   218
(df.1D <- as.data.frame(tablazat.1D))
#>   W.Hnd Freq
#> 1  Left   18
#> 2 Right  218
(tablazat.2D <- xtabs(~W.Hnd + Sex, data = kerdoiv))
#>        Sex
#> W.Hnd   Female Male
#>   Left       7   10
#>   Right    110  108
(df.2D <- as.data.frame(tablazat.2D))
#>   W.Hnd    Sex Freq
#> 1  Left Female    7
#> 2 Right Female  110
#> 3  Left   Male   10
#> 4 Right   Male  108
```

```
# oszlopdiagramok összesített gyakorisági adatokból
p1 <- ggplot(df.1D, aes(x=W.Hnd, y=Freq)) + geom_col()
p2 <- ggplot(df.2D, aes(x=W.Hnd, y=Freq, fill=Sex)) + geom_col() +
  theme(legend.position = "top")
p3 <- ggplot(df.2D, aes(x=W.Hnd, y=Freq, fill=Sex)) +
  geom_col(position="dodge") +
```

```
  theme(legend.position = "top")
p4 <- ggplot(df.2D, aes(x=W.Hnd, y=Freq, fill=Sex)) +
  geom_col(position="fill") +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti 4 ábrát összesített adatok alapján a `geom_col()` függvénnyel rajzoltuk, és a kapott oszlopdiagramok szó szerint megegyeznek a nyers adatok alapján a `geom_bar()` függvénnyel rajzolt diagramokkal. Kétdimenziós esetben a `geom_col()` szokásos előkészítése a leképezés részben: `x=<faktor1>`, `<fill=faktor2>` és `y=<gyakorisági adatok>`.

A gyakorlatban előfordulhat, hogy az összesített adatokat tartalmazó adatmátrix elemszámokat tartalmazó oszlopa közvetlenül nem alkalmas a megjelenítésre. Tekintsük a következő példát.

Példa 9.2 (Kétpettyes katicabogarak)**.** A kétpettyes katicabogarak (Adalia bipunctata) ipari és vidéki területen való előfordulásáról `ladybirds_morph_colour.csv` adatállomány tartalmaz információt. A `Habitat` változó megmutatja, hogy melyik területen történt a gyűjtés (`Industrial`, `Rural`), a `Site` változó a mérési hely azonosítója (10 különböző hely, 5 ipari és 5 vidéki területen), a `morph_color` változó a katicabogár két változatát (`red` és `black`) tartalmazza. Az iránt érdeklődünk, hogy a szennyezettebb élőhely (`Industrial`) valóban kedvezőbb-e azon rovarok számára, amelyek sötétebb morfológiával rendelkeznek, mert így kevésbe vannak kitéve a ragadozó állatoknak.
Forrás: Beckerman és mtsai. (2017)

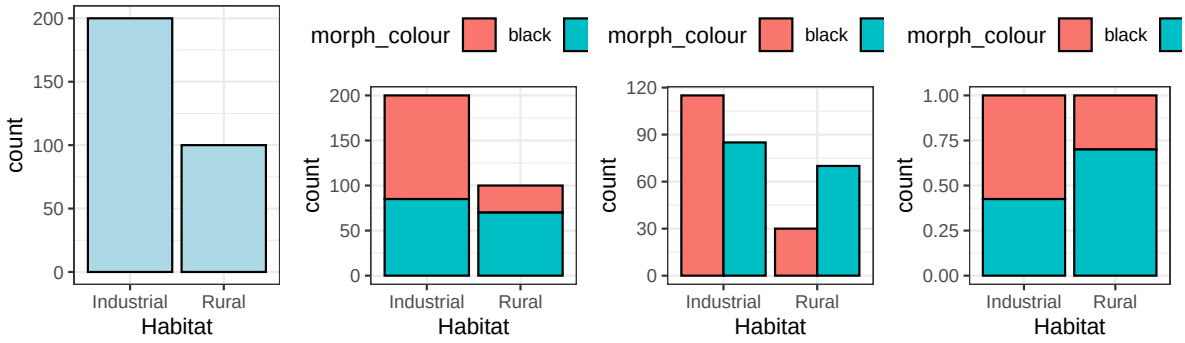
Olvassuk be a `ladybirds_morph_colour.csv` adatállományt.

```
# Adatbázis beolvasása
katica <- read.table(file = "adat/ladybirds_morph_colour.csv", header = T, sep = ",",
    dec = ".", stringsAsFactors = T)
psych::headTail(katica)
#>        Habitat Site morph_colour number
#> 1        Rural   R1        black     10
#> 2        Rural   R2        black      3
#> 3        Rural   R3        black      4
#> 4        Rural   R4        black      7
#> ...       <NA> <NA>         <NA>    ...
#> 17  Industrial   U2          red     23
#> 18  Industrial   U3          red     21
#> 19  Industrial   U4          red      9
#> 20  Industrial   U5          red     15
```

A beolvasott `katica` adatbázis összesített gyakorisági adatokat tartalmaz (`number` változó), de három faktor alapján. Például az első sorban lévő `number` érték 10, azaz vidéki területen, az R1-es mérőhelyen fekete katicából 10 darabot figyeltek meg. Az eddigi megoldásaink sem egy-, sem kétdimenziós gyakorisági táblázatok megjelenítését nem támogatják. Ahogyan a következő példában láthatjuk, a `geom_bar()` geom függvényt kell használnunk a szokásos módon, de a leképezésben a `weight=<gyakorisági adatok>` paraméter is használnunk kell.

A fenti adatbázisból a következő rajzparancsokkal készíthetünk 1D és 2D oszlopdiagramokat.

```
# oszlopdiagramok (részben) összesítetlen gyakorisági adatokból
p1 <- ggplot(katica, aes(x=Habitat, weight=number)) +
  geom_bar(fill="#ADD8E6", colour="black")
p2 <- ggplot(katica, aes(x=Habitat, fill=morph_colour,
                         weight=number)) +
  geom_bar(colour="black") + theme(legend.position = "top")
p3 <- ggplot(katica, aes(x=Habitat, fill=morph_colour,
                         weight=number)) +
  geom_bar(colour="black", position = "dodge") +
  theme(legend.position = "top")
p4 <- ggplot(katica, aes(x=Habitat, fill=morph_colour,
                         weight=number)) +
  geom_bar(colour="black", position = "fill") +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



9.1.6. 2D pontdiagram

Kétdimenziós pontdiagramok segítségével két numerikus változó kapcsolatát jeleníthetjük meg. Tekintsük a következő példát.

Példa 9.3 (Almafák terméshozama). A `compensation.csv` egy gyümölcsöskert almafáinak terméshozamát tartalmazza (`Fruit` változó kg-ban). Rögzítettük azt is, hogy az almafát milyen alanyra oltották (az alany szélessége mm-ben: `Root` változó). Továbbá egyes fák a gyümölcsös azon részeiben vannak, ahol szarvasmarhákat legeltetnek (`Grazing` változó `Grazed` értékkel), más fák pedig legeltetéstől mentes részen vannak (`Grazing` változó `Ungrazed` értékkel). A legeltetés csökkentheti a fű mennyiségét, így ezen a területen kevésbé kell osztozni az almafáknak a fűvel közös erőforrásokon.
Forrás: Beckerman és mtsai. (2017)

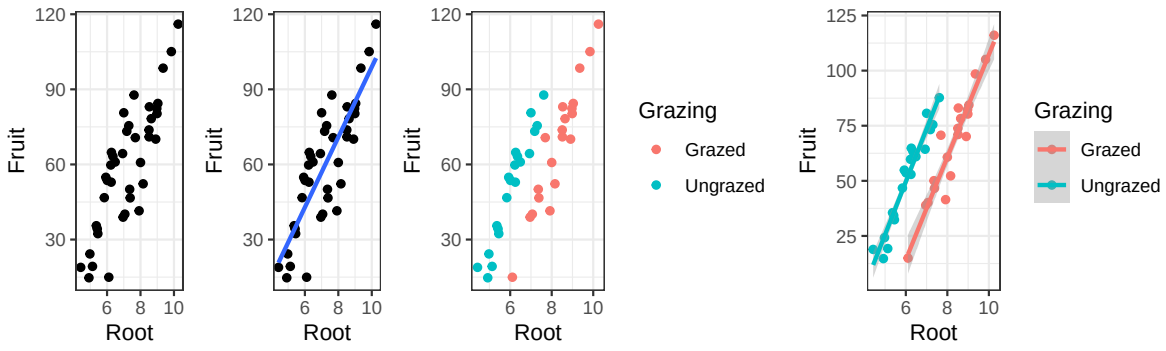
Végezzük el a `compensation.csv` adatállomány beolvasását.

```
# adatbázis beolvasása
alma <- read.table(file = "adat/compensation.csv", header = T, sep = ",", dec = ".",
    stringsAsFactors = T)
str(alma)
#> 'data.frame':    40 obs. of  3 variables:
#>  $ Root   : num  6.22 6.49 4.92 5.13 5.42 ...
#>  $ Fruit  : num  59.8 61 14.7 19.3 34.2 ...
#>  $ Grazing: Factor w/ 2 levels "Grazed","Ungrazed": 2 2 2 2 2 2 2 2 2 2 ...
psych::headTail(alma)
#>     Root Fruit  Grazing
#> 1   6.22 59.77 Ungrazed
#> 2   6.49 60.98 Ungrazed
#> 3   4.92 14.73 Ungrazed
```

```
#> 4   5.13 19.28 Ungrazed
#> ...  ...   ...     <NA>
#> 37  8.16 52.26   Grazed
#> 38  7.38 46.64   Grazed
#> 39  8.52 71.01   Grazed
#> 40  8.53 83.03   Grazed
```

Vizsgáljuk meg két numerikus változó, az alany mérete és a terméshozam kapcsolatát. Tudjuk, hogy a `ggplot()` függvény `data=` argumentumában meg kell adnunk az `alma` adatbázis nevét, és a `mapping=` argumentumban az `aes()` függvényhívásban az adatbázis `Root` és `Fruit` numerikus változóit az `x` és `y` geom paraméterekhez kötjük. Magáról a pontok megjelenítéséről `geom_point()` gondoskodik.

```
p1 <- ggplot(data = alma, mapping = aes(x = Root, y = Fruit)) + geom_point()
p2 <- ggplot(data = alma, mapping = aes(x = Root, y = Fruit)) + geom_point() +
    geom_smooth(method = "lm", se = F)
p3 <- ggplot(alma, aes(x = Root, y = Fruit, colour = Grazing)) + geom_point()
p4 <- ggplot(alma, aes(x = Root, y = Fruit, colour = Grazing)) + geom_point() +
    geom_smooth(method = "lm")
grid.arrange(p1, p2, p3, p4, layout_matrix = rbind(c(1, 2, 3, 3, 4, 4)))
```



A fenti ábrákon vegyük észre a pozitív összefüggést az *x* tengelyen ábrázolt alany-szélesség és az *y* tengelyen megjelenő almahozam között. Már az 1. ábráról jól kivehető két pontcsoport, amely a legeltetéssel kapcsolatos. Ezt ellenőrizzük a 3. ábrán, amely megjeleníti a `Grazing` faktor hatását, ezért a leképezés részben szerepeltettük az `x=` és `y=` paraméterek mellett a `colour=Grazing` paramétert is, amely a megjelenő pontok színét határozza meg. A `Grazing` két értékkel rendelkezik (`Grazed` és `Ungrazed`), ezekhez a ggplot2 rendszer két alapértelmezett színt rendel (`"#F8766D"` és `"#00BFC4"`; a `{scales}` csomag `hue_pal()` függvényét használja erre). A pontok színezésével párhuzamosan a jelmagyarázat is megjelenik az ábránkon. A 2. és

4. ábrákon lineáris regressziós egyeneseket is megjelenítettünk a `geom_smooth(method="lm")` módosító segítségével. A regressziós egyeneshez tartozó konfidencia intervallum alapértelmezetten megjelenik, de az `se = FALSE` paraméterrel letiltható.

9.1.7. Ábrák mentése

Az ábrák mentése a `ggsave()` függvénnyel történik. A kép méreteit alapértelmezetten hüvelykben (inch-ben, 1 inch 2,54 cm) adhatjuk meg a `width=` és a `height=` paraméterekben. A felbontást a `dpi=` paraméterben változtathatjuk meg, nyomtatásban a 300 dpi a legtöbb esetben elegendő. A `scale=` alapértelmezett 1 értékének változtatásával kicsinyíthetjük (1-nél kisebb értékek) vagy nagyíthatjuk (1-nél nagyobb érték) az elkészült ábrát. Az elkészült ábra formátumát a képállmony kiterjesztése határozza meg. A `.png` kiterjesztés vagyis a PNG formátum a legtöbb esetben megfelelő, de beállíthatunk `.eps`, `.ps`, `.tex`, `.pdf`, `.jpeg`, `.tiff`, `.bmp` vagy `.svg` kiterjesztéseket is.

```
# p1 ábra létrehozása
p1 <- ggplot(data = alma, aes(x = Grazing, y = Fruit)) + geom_boxplot()
# PNG képállomány létrehozása p1-ből
ggsave(filename = "output/kep/gs_kep1.png", plot = p1, width = 4, height = 3,
    dpi = 300, scale = 0.9)
```

A háttértárra írt képállomány több ábrát is tartalmazhat. Első lépésben hozzuk létre az ábrákat (például `p1` és `p2`), majd a `{gridExtra}` csomag `grid.arrange()` függvényével illesszük őket össze, és az így létrehozott ábrát mentsük el a `ggsave()` segítségével.

```
# p2 ábra létrehozása, p1 már létezik (lásd fent)
p2 <- ggplot(alma, aes(x = Root, y = Fruit)) + geom_point()
# p1 és p2 egymás mellett
p.1.2 <- grid.arrange(p1, p2, ncol = 2)
# PNG képállomány létrehozása p1-ből és p2-ből
ggsave(filename = "output/kep/gs_kep1_2.png", plot = p.1.2, width = 7, height = 3,
    dpi = 300, scale = 0.9)
```

Összefoglalás

A ggplot2 ábra rajzolása egy adatmátrix létezését feltételezi. El kell döntenünk, hogy a számos faktor és/vagy numerikus változó közül melyeket szeretnénk az ábra létrehozásába bevonni. Faktor változó vizsgálata esetén tipikusan oszlopdiagramot használunk, numerikus változónál pontdiagramot, dobozdiagramot vagy hisztogramot. Az ábra típusa

meghatározza, hogy milyen `geom_*()` függvényt fogunk használni a rajzparancsainkban. Például oszlopdiagram esetén a `geom_col()`, dobozdiagram esetén a `geom_boxplot()` geom függvényt használjuk. A rajzparancsunk harmadik összetevője az a leképezés, amelyet a `ggplot(mapping=aes(<LEKÉPEZÉS>))` kifejezésben használunk, amely az adatmátrix változóit és a használt geom elemek paramétereit köti össze. Az `aes()` függvényben számos paraméter beállítható, például `x=`, `y=`, `fill=` és `colour=`, amelyek jelentősen hozzájárulnak az ábra összetettségéhez és értelmezéséhez, melyet jelmagyarázat is segít. Amennyiben geom paramétert az `aes()` függvényen kívül adunk meg, az csak az ábra szépségéhez járul hozzá.

🎯 Feladatok

1. A 9.1 példában 237 Ausztrál egyetemista adata szerepel. A `kerdoiv.xlsx`-ből elérhető adatokban megtalálható a `Fold` faktor és az `Age` numerikus változó. Megkérték az egyetemistákat, hogy kulcsolják össze a kezüket, majd lejegyezték a felül lévő kart. Így a `Fold` lehetséges értékei `"R on L"`, `"L on R"`, `"Neither"`. Az `Age` változó az egyetemisták életkorát tartalmazza évben kifejezve. Vizsgáljuk meg mindkét változó eloszlását a szokásos módon. A faktor változót oszlopdiagrammal, a numerikus változót egydimenziós pontdiagrammal, dobozdiagrammal és hisztogrammal is.
2. Mentsük el háttértárra PNG állományok formájában az előző feladat négy ábráját!
3. Az 1. feladatban szereplő két változó (`Fold` és `Age`) eloszlását vizsgáljuk meg nemenként is! Továbbra is használjunk oszlopdiagramot, illetve egydimenziós pontdiagramot, dobozdiagramot és hisztogramot.
4. Vizsgáljuk meg több módszerrel is, hogy az `Age` változó hisztogramjában milyen széles osztályintervallumokkal lenne érdemes dolgozni. Hívjuk segítségül az *Alap R* `hist()` függvényét.
5. Egyetlen ábrán jelenítsünk meg 3 db normális eloszláshoz tartozó sűrűségfüggvényt, melyek paraméterei: $\mu_1 = 1, \sigma_1 = 2$; $\mu_2 = 1, \sigma_2 = 3$; $\mu_3 = 2, \sigma_3 = 1$.
6. Mutassuk be a standard normális eloszlás és a t eloszlás kapcsolatát! Egy ábrán jelenítsük meg a standard normális eloszlás sűrűségfüggvényét és az 5, 10, 15, 20 és 25 szabadsági fokú t eloszlás sűrűségfüggvényét!
7. Jelenítsük meg az $n = 50, p = 1/3$ paraméterű binomiális eloszlás eloszlását bemutató vonalas ábrát!

9.2. Számítások az ábrán 🤔

> **i** Miről lesz szó? Ebben a fejezetben
>
> - megismerjük az összesített adatokkal dolgozó geom elemek (például oszlopdiagram, dobozdiagram és a hisztogram) rajzolási elveit,
> - megtanuljuk, hogyan helyezhetjük el a pont- vagy intervallumbecslés eredményét az ábrán,
> - és áttekintjük az átlagábrák rajzolási lehetőségeit.

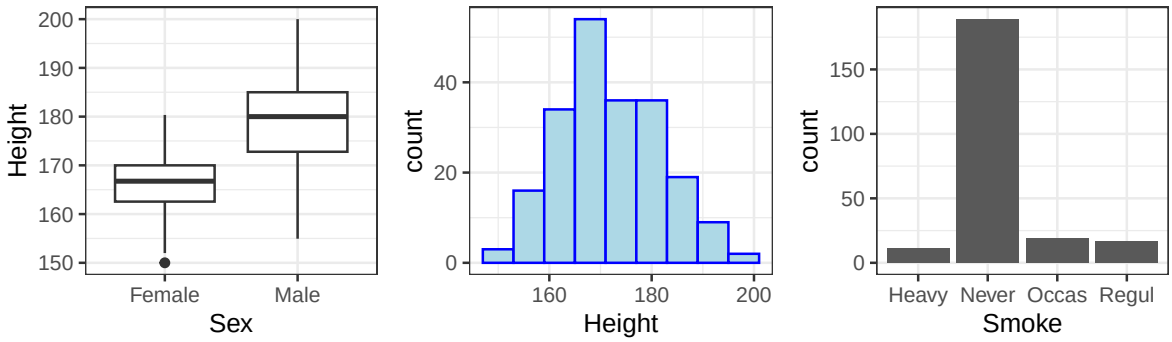
Minden ggplot2 ábra valamilyen adatbázison alapul, és az abban lévő változók értékei a leképezés segítségével válnak a megjelenített geom elemek tulajdonságaivá. Sokszor nem közvetlenül az adatbázis változók értékeit szeretnénk az ábrán megjeleníteni, hanem az azokból származó mutatókat vagy transzformált adatokat. Ezek a megjelenítésre szánt származtatott adatok kétféle módszerrel jöhetnek létre:

- vagy automatikusan a megjelenítés során (9.2.1. fejezet),
- vagy direkt módon, az általunk előírt recept szerint (9.2.2. fejezet).

9.2.1. Automatikus számítások

Emlékezzünk vissza a dobozdiagram ábrára (9.1.3. fejezet). A `ggplot()` függvényben nyers adatokat adtunk meg (237 egyetemista testmagassága), azonban az ábráról származtatott adatokat olvashattunk le, például a testmagasság mediánját, minimumát és maximumát. Hogyan jöttek ezek létre? Hasonló kérdést tehetünk fel a hisztogram és az oszlopdiagram egyes oszlopainak magasságával kapcsolatban is, ahogyan a lenti ábrákon ezt láthatjuk.

```
# p1 - dobozdiagram
p1 <- ggplot(kerdoiv, aes(x = Sex, y = Height)) + geom_boxplot() +
  scale_x_discrete(na.translate = F)
# p2 - dobozdiagram nyers pontokkal
p2 <- ggplot(kerdoiv, aes(x = Height)) +
  geom_histogram(binwidth=6, colour="blue", fill="lightblue")
# p3 - dobozdiagram kitöltőszínnel és nyers pontokkal
p3 <- ggplot(kerdoiv, aes(x = Smoke)) + geom_bar() +
  scale_x_discrete(na.translate = F)
grid.arrange(p1, p2, p3, ncol=3)
```



9.2.1.1. Származtatott változók

Először is a `ggplot_build()` függvényt hívjuk segítségül, hogy pontosan megkaphassuk azt az adatbázist, ami alapján az ábra valóban létrejön. Természetesen ez az ideiglenes ábra-adatbázis a nyers `kerdoiv` adatbázison alapul, de az ábra megalkotásához szükséges specifikus információkat is szerepelnek benne, ezek az ún. származtatott változók. Az igen összetett adatszerkezet `data` részének megjelenítésével pontosan tájékozódhatunk a háttérben keletkező új adatokról. Vizsgáljuk most csak a `p1` objektumban tárolt, fenti 1. ábra hátterét.

```
ggplot_build(p1)$data[[1]][, 1:5]  # csak az első 5 oszlop
#>     ymin  lower middle upper   ymax
#> 1 152.00 162.56 166.75   170 180.34
#> 2 154.94 172.79 180.00   185 200.00
names(ggplot_build(p1)$data[[1]])  # az összes oszlop neve
#>  [1] "ymin"        "lower"       "middle"      "upper"
#>  [5] "ymax"        "outliers"    "notchupper"  "notchlower"
#>  [9] "x"           "flipped_aes" "PANEL"       "group"
#> [13] "ymin_final"  "ymax_final"  "xmin"        "xmax"
#> [17] "xid"         "newx"        "new_width"   "weight"
#> [21] "colour"      "fill"        "alpha"       "shape"
#> [25] "linetype"    "linewidth"
```

A `p1` ábra ábra-adatbázisa mindössze két sort tartalmaz a két dobozdiagram számára. Származtatott változóként többek között tartalmazza a három kvartilist (`lower`, `middle`, `upper`) és a szélsőértékeket (`ymin`, `ymax`) is. A fenti outputokból kiderül, hogy a `geom_boxplot()` nem egyszerűen a dobozdiagram geom elem hozzáadásáról gondoskodik, hanem a nyers adatbázis egyfajta transzformációját is elvégzi.

9.4. táblázat: Transzformációt használó `geom` elemek *(saját szerkesztés)*

Geom elem	Transzformáció	Származtatott változók
`geom_boxplot()`	`stat_boxplot()`	`width`, `ymin`, `lower`, `notchlower`, `middle`, `notchupper`, `upper`, `ymax`
`geom_bar()`	`stat_count()`	`count`, `prop`
`geom_density()`	`stat_density()`	`density`, `count`, `scaled`, `ndensity`
`geom_histogram()`	`stat_bin()`	`count`, `density`, `ncount`, `ndensity`
`geom_freqpoly()`		
`geom_violin()`	`stat_ydensity()`	`density`, `scaled`, `count`, `violinwidth`, `n`, `width`
`geom_smooth()`	`stat_smooth()`	`y`, `ymin`, `ymax`, `se`
`stat_ecdf()`	`stat_ecdf()`	`x`, `y`
`geom_qq()`	`stat_qq()`	`sample`, `theoretical`
`geom_qq_line()`	`stat_qq_line()`	`x`, `y`

A 9.4. táblázat összefoglalja az automatikus transzformációt végző geom elemeket. Dobozdiagram esetében a geom megjelenési paramétereket a `geom_boxlot()` által meghívott `stat_boxplot()` függvény biztosítja, így nekünk ezekről nem kell gondoskodni, elegendő egyetlen numerikus vektort (`y=`) megadnunk. Hisztogram esetében a `stat_bin()`, oszlopdiagram esetében a `stat_count()` végzi a származtatott változók létrehozását.

További ábrákhoz tekintsük a következő példát.

Példa 9.4 (Soay juhok)**.** A Soay skót félvad juhféle Hirta egy kis szigetén Skócia nyugati partjainál él. Igazoltnak látszik, hogy a nagyobb testméret evolúciós előnyökkel jár. Az anyajuh fittségét mutatja az utódok száma (reprodukciós siker). A `SoaySheepFitness.csv` adatbázis `body.size` változója az anyajuh standardizált, átlagos tömegét (kg-ban) méri, a `fitness` pedig az utódok számát a teljes élettartam alatt.
Forrás: Beckerman és mtsai. (2017)

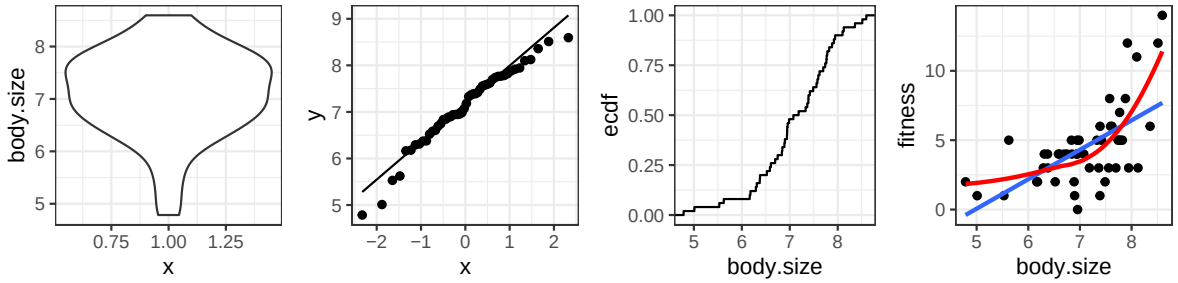
Olvassuk be a `SoaySheepFitness.csv` állományt.

```
# adatbázis megnyitása
soay <- read.table(file = "adat/SoaySheepFitness.csv", header = T, sep = ",",
    dec = ".", stringsAsFactors = T)
psych::headTail(soay)
#>     fitness body.size
#> 1         4      6.37
#> 2         3      7.18
```

```
#> 3         2      6.16
#> 4        14       8.6
#> ...     ...       ...
#> 47        3      7.36
#> 48        7      7.77
#> 49        2      6.89
#> 50        8      7.88
```

A következő ábrákon a dobozdiagramhoz hasonló elven működő, automatikus számításokat végző geom elemekre mutatunk példát. A hegedűdiagram, a QQ-ábra, az eloszlásfüggvény és a görbeillesztés egyes esetei mind-mind ebbe a csoportba tartoznak. A hegedűdiagram, a QQ-ábra és a tapasztalati eloszlásfüggvény létrehozásához elegendő egyetlen numerikus vektort biztosítani, a megjelenítéshez szükséges számításokról maga a geom függvény gondoskodik. Könnyen illeszthetünk egyenest vagy a görbét a kétdimenziós pontdiagramra, ehhez mindössze a `geom_smooth()` függvényt kell használnunk. Valójában az illesztett egyenes és görbe ábrázolásához szükséges adatok előállításáról maga a `geom_smooth()` gondoskodott, nekünk ezzel nincs teendőnk.

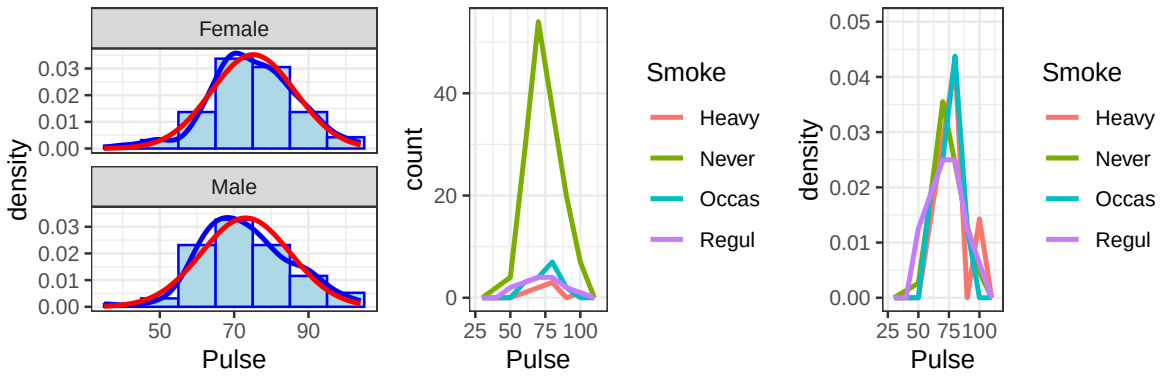
```
# p1 - hegedűdiagram a testsúlyra
p1 <- ggplot(soay, aes(x=1, y=body.size)) + geom_violin()
# p2 - QQ-ábra a testsúlyra
p2 <- ggplot(soay, aes(sample=body.size)) + geom_qq() +
  geom_qq_line()
# p3 - tapasztalati eloszlásfüggvény a testsúlyra
p3 <- ggplot(soay, aes(x=body.size)) + stat_ecdf()
# p4 - kétdimenziós pontdiagram, regressziós egyenessel és
# görbeillesztéssel
p4 <- ggplot(soay, aes(x = body.size, y = fitness)) + geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  geom_smooth(method = "loess", span = 1, colour = "red", se = FALSE)
grid.arrange(p1, p2, p3, p4, ncol=4)
```



9.2.1.2. A származtatott változók elérése

Az ábra-adatbázis új oszlopait az `after_stat()` függvénnyel tudjuk elérni a rajzparancsainkban. Ez biztosítja, hogy az eredeti adatbázis változói és a származtatott új változók ne keveredjenek össze. Az `after_stat()` argumentumában csak származtatott változók szerepelhetnek. A függvény használatára gyakrabban van szükségünk mint gondolnánk. Tudjuk, ha hisztogramot rajzolunk, akkor használhatjuk a `stat_bin()` függvény `density` változóját, vagy ha oszlopdiagramot hozunk létre szükség lehet a `stat_count()` által létrehozott `count` változóra. Nézzünk ezekre példát!

```
p0 <- kerdoiv |> drop_na(Sex, Smoke) |> ggplot(aes(x=Pulse)) +
  geom_histogram(aes(y=after_stat(density)),
                 binwidth=10, colour="blue", fill="lightblue")
p1 <- p0 + geom_density(linewidth=1, colour="blue") +
  stat_theodensity(colour = "red", linewidth=1) +
  facet_wrap(~Sex, ncol=1)
p2 <- ggplot(kerdoiv, aes(x=Pulse, colour=Smoke)) +
  geom_freqpoly(binwidth=10, linewidth=1) +
  scale_color_discrete(na.translate=F)
p3 <- ggplot(kerdoiv, aes(x=Pulse, colour=Smoke)) +
  geom_freqpoly(aes(y=after_stat(density)), binwidth=10, linewidth=1)+
  scale_color_discrete(na.translate=F) +
  coord_cartesian(ylim=c(0,0.05))
grid.arrange(p1, p2, p3, ncol=3)
```



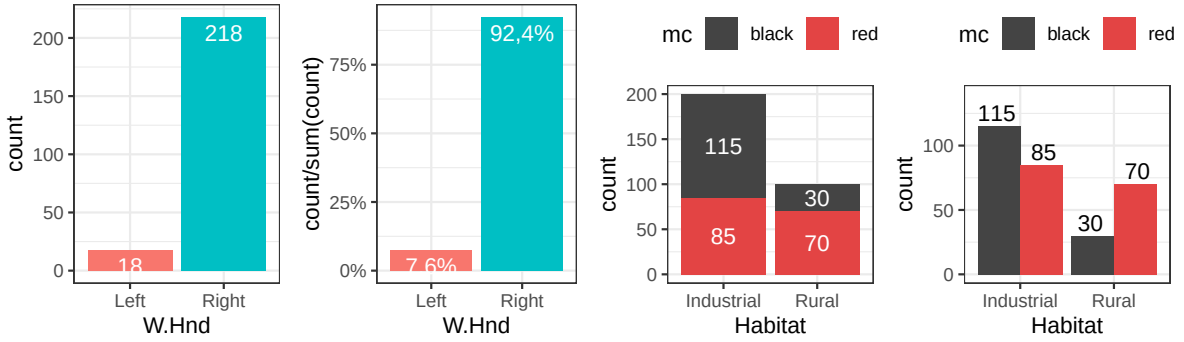
Az 1. ábrán a hisztogram rajzolásához nem az alapértelmezett gyakoriságot használjuk (`aes(y=after_stat(count))`), hanem a gyakorisági sűrűséget (`aes(y=after_stat(density))`),

amely lehetőséget ad arra, hogy egy ábrán szerepeljen a simított hisztogrammal `geom_density()` és a normális eloszlás sűrűségfüggvényével (`stat_theodensity()`). Ez utóbbi lehetőség a
`{ggh4x}` csomag betöltése után áll rendelkezésre. A megjelenítéshez szükséges paraméterek (átlag és szórás) becslése automatikusan megtörténik.

A 2. és 3. ábrán a gyakorisági poligonok ábrázolását vethetjük össze. A 2. ábrán gyakorisági értékek, míg a 3. ábrán relatív gyakorisági értékek alapján történik az ábrázolás, így az eltérő elemszámokhoz tartozó eloszlások összehasonlítása lényegesen leegyszerűsödik.

```
# oszlopdiagramok nyers adatokból
p1 <- kerdoiv |> drop_na(W.Hnd) |>
  ggplot(aes(x=W.Hnd, fill=W.Hnd)) + geom_bar() +
  geom_text(aes(label=after_stat(count)),
            stat="count", color="white", vjust=1.5) +
  theme(legend.position = "none")
p2 <- kerdoiv |> drop_na(W.Hnd) |>
  ggplot(aes(x=W.Hnd, fill=W.Hnd, y=after_stat(count/sum(count)))) +
  geom_bar() +
  geom_text(aes(label=scales::percent(after_stat(count/sum(count)),
                                      decimal.mark = ",",
                                      accuracy = 0.1)),
            color="white", vjust=1.5, stat="count") +
  theme(legend.position="none") +
  scale_y_continuous(labels = scales::percent)
p3 <- ggplot(data = katica, aes(x = Habitat, weight=number,
                                fill = morph_colour)) +
  geom_bar() +
  geom_text(aes(label=after_stat(count)),
            position= position_stack(vjust=0.5),
            colour="white", stat="count") +
  scale_fill_manual(values = c("black"="#444444",
                                      "red"="#e34444"), name = "mc") +
  theme(legend.position = "top")
p4 <- ggplot(data = katica, aes(x = Habitat,
                                weight=number, fill = morph_colour)) +
  geom_bar(position = "dodge") +
  geom_text(aes(label=after_stat(count)),
            position=position_dodge(width=1),
            color="black",vjust=-.3, stat = "count") +
  scale_fill_manual(values = c("black"="#444444",
```

```
                                    "red"="#e34444"), name = "mc") +
  coord_cartesian(ylim=c(0, 140)) +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti ábrákon a `geom_bar()` által meghívott `stat_count()` függvény `count` származtatott változóját használjuk az `after_stat(count)` kifejezéssel. Mindegyik ábrán felhasználjuk az egyes gyakoriságok megjelenítéséhez a `geom_text()` függvényben, de a 2. ábrán az `after_stat(count/sum(count))` kifejezéssel relatív gyakoriságokat használunk.

Összefoglalva, léteznek olyan grafikus elemek, amelyek változtatás nélkül képesek a bemenő adatok alapján a geom paramétereket beállítani (például `geom_point()`, `geom_col()`), és léteznek olyanok is, amelyek a bemenő adatokat áttranszformálják egy köztes adattáblába, és a geom paraméterek konkrét értékét innen veszik. Természetesen ezeknek a geom elemeknek pontosan a származtatott adatokat megjelenítése a célja. A 9.4. táblázat ezeket a statisztikai transzformációkat foglalja össze.

9.2.2. Direkt számítások

Statisztikai számításokat mi is előírhatunk egy-egy ggplot2 ábra létrehozása során. Ehhez a `stat_summary()` függvényt használjuk, amelynek fontos paramétere a konkrét statisztikai számítást előíró `fun=`, `fun.min=`, `fun.max=` és `fun.data=`.

Ebben a részben a következő példát is felhasználjuk.

Példa 9.5 (A vonzerő és a nem hatása az ítéletekre)**.** Egy kísérletben a résztvevőket arra kérték, hogy jelöljék meg, hány év börtönbüntetést szabnának ki vádiratok alapján. A 60 résztvevőből 20 egyszerűen megkapta az írásbeli szöveget, nem volt csatolva fotó a vádlottról, 20 résztvevő azonban a szöveg mellé egy vonzó vádlottól származó fotót

> kapott, az utolsó 20 fős csoport pedig egy nem vonzó vádlottól származót. A fotón férfi vagy nő vádlottak jelentek meg, és a résztvevők közé is egyaránt válogattak férfiakat és nőket is. Az adatok a `vallomasok.sav` állományban találhatók. A `sexdiff` változó a vizsgálati személy és az vádlott nemének egyezését tartalmazza (`"ugyanaz"`, `"eltérő"`), az `attract` a vádlott vonzóságára vonatkozó információ (`"vonzó"`, `"nem vonzó"`, `"nincs kép"`), míg a `sentence` az ítélet években kifejezve.
> *Forrás: Brace és mtsai. (2016, o. 213)*

Olvassuk be az adatokat!

```
# adatok beolvasása és előkészítése
vallomas <- rio::import(file = "adat/vallomasok.sav")
vallomas$sexdiff <- factor(vallomas$sexdiff)
levels(vallomas$sexdiff) <- c("ugyanaz", "eltérő")
vallomas$attract <- factor(vallomas$attract)
levels(vallomas$attract) <- c("vonzó", "nem vonzó", "nincs kép")
tibble::glimpse(vallomas)
#> Rows: 60
#> Columns: 3
#> $ sexdiff  <fct> ugyanaz, ugyanaz, ugyanaz, ugyanaz, ugyanaz, ugyanaz,~
#> $ attract  <fct> vonzó, vonzó, vonzó, vonzó, vonzó, vonzó, vonzó, vonz~
#> $ sentence <dbl> 6, 8, 7, 9, 5, 7, 10, 9, 9, 5, 5, 11, 7, 5, 8, 5, 9, ~
psych::headTail(vallomas)
#>     sexdiff   attract sentence
#> 1   ugyanaz     vonzó        6
#> 2   ugyanaz     vonzó        8
#> 3   ugyanaz     vonzó        7
#> 4   ugyanaz     vonzó        9
#> ...    <NA>      <NA>      ...
#> 57   eltérő nincs kép       12
#> 58   eltérő nincs kép       12
#> 59   eltérő nincs kép       13
#> 60   eltérő nincs kép       16
```

9.2.2.1. Pontbecslések ábrázolása

Sokszor felmerül, hogy az elkészült ábránkon még egy középértéket (például mintaátlagot) pluszban jelenítsünk meg. Az átlag az ábrán leolvasható lehet egy pont y koordinátájaként, egy oszlop magasságának formájában, vagy egy vízszintes/függőleges egyenesként, ahol az

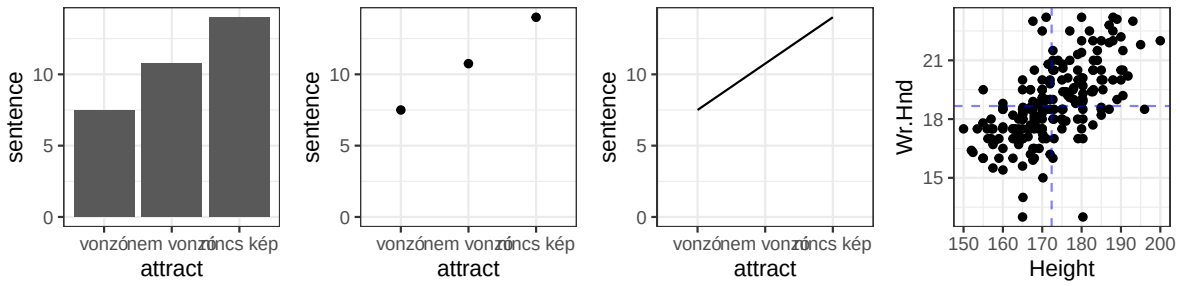
átlagot az y/x tengellyel való metszéspont jeleníti meg. Több átlagot, ha azokat pontokkal jelenítjük meg, össze is köthetünk vonallal. A 9.5. táblázatban összefoglaltuk ezeket a lehetőségeket.

9.5. táblázat: Pontszerű számítások a `stat_summary()` segítségével. A `fun=mean` helyett, bármely más vektorból skalárt előállító függvényt használhatjuk, tipikus még, a `fun=median` *(saját szerkesztés)*

Geom elem	`stat_summary()` paramétere	Beállított változó
oszlop	`fun=mean, geom="col"`	`y` - az oszlopok magassága
pont	`fun=mean, geom="point"`	`y` - a pontok y koordinátája
vonal(diagram)	`fun=mean, geom="point"`	`y` - a pontok y koordinátája
vízszintes vonal	`fun=mean, geom="hline"`	`yintercept=after_stat(y)` - metszéspont az y tengellyel
függőleges vonal	`fun=mean, geom="vline"`	`xintercept=after_stat(y)` - metszéspont az x tengellyel

A következő ábrán bemutatjuk a 9.5 táblázatban szereplő lehetőségeket.

```
p1 <- ggplot(vallomas, aes(x = attract, y = sentence)) +
  stat_summary(fun = mean, geom = "col")
p2 <- ggplot(vallomas, aes(x = attract, y = sentence)) +
  stat_summary(fun = mean, geom = "point") +
  coord_cartesian(ylim=c(0,14))
p3 <- ggplot(vallomas, aes(x = attract, y = sentence)) +
  stat_summary(fun = mean, geom = "line", group=1) +
  coord_cartesian(ylim=c(0,14))
p4 <- ggplot(kerdoiv, aes(x = Height, y = Wr.Hnd)) +
  geom_point() +
  stat_summary(fun = mean, geom = "hline",
               colour="blue", alpha=.5, linetype=2,
               aes(x=0, yintercept=after_stat(y))) +
  stat_summary(fun = mean, geom = "vline",
               colour="blue", alpha=.5, linetype=2,
               aes(y=0, xintercept=after_stat(x)), orientation = "y") +
  coord_cartesian(xlim=range(kerdoiv$Height, na.rm = T),
                  ylim=range(kerdoiv$Wr.Hnd, na.rm = T))
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti 4 példában végig az átlagszámító `fun=mean` argumentumot használtuk a `stat_summary()` függvényben. Az átlag számítása az 1-3. ábrán az x tengely kategóriái szerint történik (az alapértelmezett `orientation="x"` szerinti beállításnak megfelelően). Mivel három csoport van az x tengelyen, ezért három átlagolt érték keletkezik, amely az `y` származtatott értékként használható fel később (`after_stat(y)`-et kell használni ha közvetlenül szeretnénk rá hivatkozni).

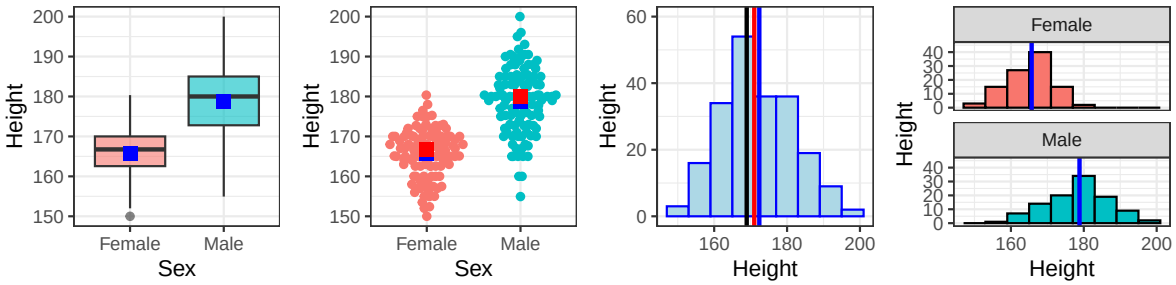
Az 1. ábrán a `geom="col"` hatására oszlopdiagram jön létre, és az új `y` származtatott változó az oszlopok magasságát írja le. A 2. ábrán a `geom="point"` miatt pontdiagram jön lére, és az `y` jelentése a pontok y koordinátája. A 3. ábrán megjelenő vonaldiagram lényegében ezeket a pontokat köti össze.

A 4. ábrán egy vízszintes vonal (`geom="hline"`) és egy függőleges vonal (`geom="vline"`) is megjelenik. A `geom_hline()` kötelező paramétere a `yintercept=`, így ennek az `y` átlagot az `after_stat(y)` függvényhívással tudjuk közvetlenül átadni. Az `x=0` biztosítja, hogy egyetlen csoportunk legyen az x tengely mentén, vagyis minden mintabeli személy kézméretét vegye figyelembe az átlagolás során. A függőleges vonal a testmagasságok átlagát reprezentálja, így egyrészt fordítani kell az összegzés irányán (`orientation="y"`), vagyis az x tengelyen lévő értékeket fogjuk összegezni. Másrészt az `y=0` segítéségével jelezzük, hogy egyetlen átlagot szeretnénk létrehozni, és végül a `geom_vline()` függvény `xintercept=` paraméterét állítjuk be a kiszámolt átlagra (`after_stat(x)`).

Sok esetben meglévő ábra kiegészítésére használjuk az összesített értékeket.

```
p1 <- ggplot(kerdoiv, aes(x = Sex, y = Height, fill=Sex)) +
  geom_boxplot(alpha=0.6) + theme(legend.position = "none") +
  stat_summary(fun=mean, geom = "point", colour="blue", size=3,
               shape=15) +
  scale_x_discrete(na.translate = F)
p2 <- ggplot(kerdoiv, aes(x = Sex, y = Height, colour=Sex)) +
  geom_quasirandom() +
  stat_summary(fun=mean, geom = "point", colour="blue", size=3,
               shape=15) +
```

```
  stat_summary(fun=median, geom = "point", colour="red", size=3,
               shape=15) +
  scale_x_discrete(na.translate = F) + theme(legend.position = "none")
p3 <- ggplot(kerdoiv, aes(x=Height)) +
  geom_histogram(binwidth = 6, colour = "blue", fill="lightblue") +
  stat_summary(aes(x = 165, y = Height, xintercept = after_stat(y)),
               fun = mean, geom = "vline", colour="blue", size=1) +
  stat_summary(aes(x = 165, y = Height, xintercept = after_stat(y)),
               fun = median, geom = "vline", colour="red", size=1) +
  stat_summary(aes(x = 165, y = Height, xintercept = after_stat(y)),
               fun = LaplacesDemon::Mode, geom = "vline",
               colour="black", size=1) +
  coord_cartesian(ylim = c(0,60))
p4 <- kerdoiv |> drop_na(Sex) |>
  ggplot(aes(x=Height, fill = Sex)) +
  geom_histogram(binwidth = 6, colour = "black") +
  stat_summary(aes(x = 165, y = Height, xintercept = after_stat(y),
                   group = Sex),
               fun = mean, geom = "vline", colour="blue", size=1) +
  facet_wrap(~ Sex, ncol=1) +
  theme(legend.position = "none") +
  coord_cartesian(ylim = c(0,45))
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti példa 1. ábráján dobozdiagramon jelenítjük meg az átlagot (kék négyzetek). A 2. ábrán egydimenziós pontdiagramon az átlag mellett a medián (piros négyzet) is szerepel. A 3. és 4. ábra pedig arra mutat példát, hogy hisztogramon hogyan tudunk átlagot, mediánt és móduszt reprezentáló függőleges egyeneseket megjeleníteni.

9.2.2.2. Intervallumbecslések ábrázolása

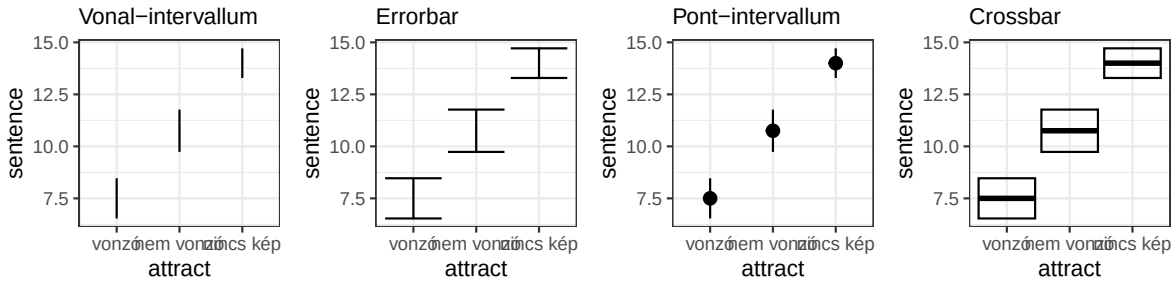
A pontszerű értékek becslésén túl gyakran van szükség olyan geom elemek megjelenítésére, amelyek intervallumokat reprezentálnak. A hiba jelzése vagy a különböző megbízhatóságú konfidencia intervallumok megjelenítése pontosan ebbe a kategóriába tartozik. Elsőként nézzük meg, milyen geom elemek alkalmasak egyáltalán intervallum megjelenítésére. A 9.6. táblázat felsorolja a négy leggyakrabban használt intervallumszerű geom elemet.

9.6. táblázat: Intervallum megjelenítésére használt geom elemek *(saját szerkesztés)*

Geom elem	Geom függvény	`stat_summary()` paramétere
Vonal-intervallum	`geom_linerange()`	`fun.data=mean_cl_normal, geom="linerange"`
Errorbar	`geom_errorbar()`	`fun.data=mean_cl_normal, geom="errorbar"`
Pont-intervallum	`geom_pointrange()`	`fun.data=mean_cl_normal, geom="pointrange"`
Crossbar	`geom_crossbar()`	`fun.data=mean_cl_normal, geom="crossbar"`

A következő parancsokból kiderül, hogy intervallumokat nem csak a 9.6. táblázat 2. oszlopában lévő geom függvényekkel hozzuk létre (például errorbart a `geom_errorbar()` függvénnyel), hanem a 3. oszlopban látható `stat_summary()` függvénnyel is. Ehhez a táblázat 3. oszlopát kell alapul vennünk.

```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun.data = mean_cl_normal, geom="linerange") +
  labs(subtitle = "Vonal-intervallum")
p2 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun.data = mean_cl_normal, geom="errorbar") +
  labs(subtitle = "Errorbar")
p3 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun.data = mean_cl_normal, geom="pointrange") +
  labs(subtitle = "Pont-intervallum")
p4 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun.data = mean_cl_normal, geom="crossbar") +
  labs(subtitle = "Crossbar")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



Látható, hogy a `stat_summary()` paraméterében a `fun.data=` argumentumot használtuk, amely most a `mean_cl_normal()` függvényre hivatkozik. Ez a függvény előállítja a mintaátlagot (`y=`), valamint az intervallum megjelenítéséhez szükséges `ymin` és `ymax` végpontokat, amelyek ebben az esetben a várható értékre vonatkozó 95%-os megbízhatóságú konfidencia intervallum határait jelentik. Vonal-intervallum és errorbar esetében elegendőek a végpontok ahhoz, hogy a vonzóság három kategóriájában a kiszabott büntetések várható értékére vonatkozó 95%-os megbízhatóságú konfidencia intervallumot megjelenítsük. Pont-intervallum és a crossbar már igényli az `y=` paramétert, vagyis a pont helyét is. A 9.7 táblázat összefoglalja, hogy milyen függvényeket használhatunk az `y=`, `ymin=` és `ymax=` értékek előállítására, így ezek is szerepelhetnek a `fun.data=` paraméterében.

9.7. táblázat: Összegző függvények a `stat_summary(fun.data=)`-ban *(saját szerkesztés)*

Összegző függvény	Paraméter	Leírás
`mean_cl_normal()`	`conf.int=0.95`	mintaátlag és 95%-os konfidenciaintervallum
`mean_se()`		mintaátlag és standard hiba felmérve mindkét irányban
`mean_sdl()`	`mult=2`	mintaátlag és szórás kétszerese felmérve mindkét irányban
`median_hilow`	`conf.int=0.95`	medián, valamint a 2,5% és 97,5%-os kvantilis

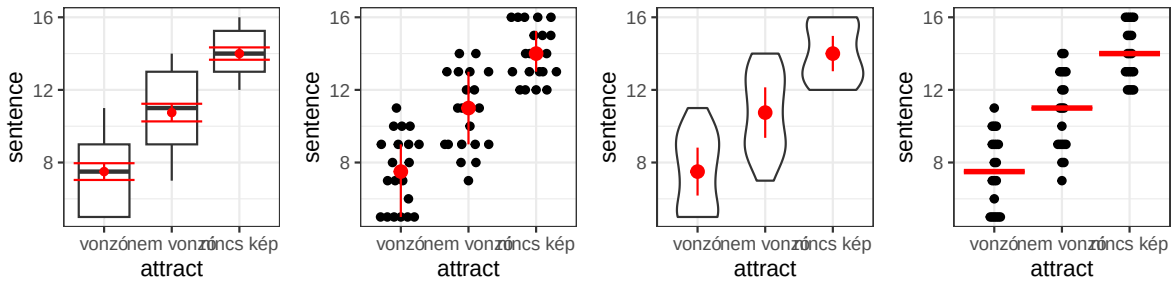
A következő parancsok a 9.7. táblázat függvényeire adnak példát. Figyeljük meg, hogy az outputban az `y=`, `ymin=` és `ymax=` értékek is megjelennek, függvényenként eltérő tartalommal.

```
# átlag és 95%-os megbízhatóságú konfidencia intervallum (t-eloszlás)
ggplot2::mean_cl_normal(kerdoiv$Height, conf.int = 0.95)
#>          y    ymin     ymax
#> 1 172.3809 171.038 173.7237
```

```
# átlag és átlag ± SE
ggplot2::mean_se(kerdoiv$Height)
#>          y     ymin    ymax
#> 1 172.3809 171.6997 173.062
# átlag és átlag ± 2xSD
ggplot2::mean_sdl(kerdoiv$Height, mult = 2)
#>          y     ymin     ymax
#> 1 172.3809 152.6858 192.0759
# median, lower, upper, ahol [lower, upper] 95%-ot fed le
ggplot2::median_hilow(kerdoiv$Height, conf.int = 0.95)
#>     y    ymin  ymax
#> 1 171 154.952 190.5
```

A következő parancsok a 9.7 táblázat összegző függvényeire mutatnak példát. Az egyes függvények paraméterezésén is változtathatunk, ehhez a `stat_summary()` függvény `fun.args=` argumentumát kell megadnunk, amely egy listát vár.

```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_boxplot() +
  stat_summary(fun.data = mean_se, geom="errorbar", col="red") +
  stat_summary(fun = mean, geom="point", col="red")
p2 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_quasirandom() +
  stat_summary(fun.data = median_hilow, fun.args = list(conf.int=0.5),
               geom="pointrange", col="red")
p3 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_violin() +
  stat_summary(fun.data = mean_cl_normal, col="red",
               fun.args = list(conf.int=0.99), geom="pointrange")
p4 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_beeswarm() +
  stat_summary(fun = median, fun.min=median, fun.max=median,
               geom="crossbar", col="red")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



9.3. ábra: Példák a `fun.data=` argumentum használatára

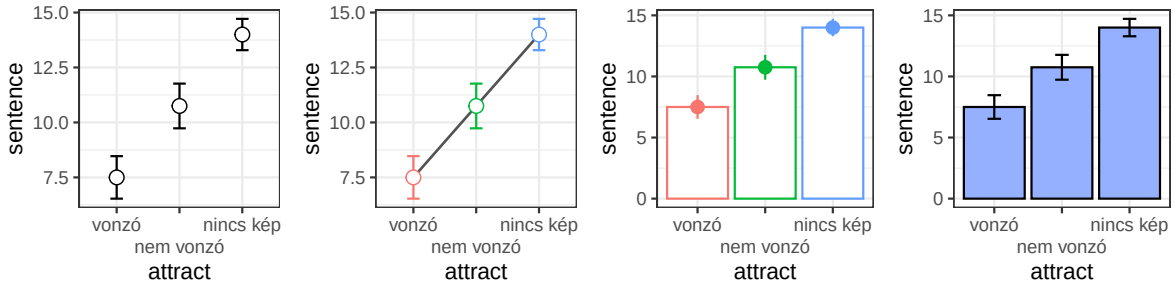
Az 1. ábra a vonzóság három kategóriájában az átlagot és a standard hibát jeleníti meg egy dobozdiagramon. A 2. ábrán egydimenziós pontdiagramra a mediánt, valamint a 25%-os és 75%-os kvantiliseket rajzolja. A 3. ábra egy hegedűábrán jelenít meg 99%-os konfidencia intervallumot a három várható értékre. A 4. ábra arra mutat példát, hogy a `fun.data=` argumentum helyett a `fun=`, `fun.min=` és `fun.max` is elegendő intervallum megjelenítésére. A crossbar szükséges paramétereit itt pontszerű függvények, mindhárom esetben a `median()` függvény szolgáltatja.

9.2.2.3. Átlagábrák

Az eddig tanultakat felhasználhatjuk egy- és kétdimenziós átlagábrák készítéséhez. A következő négy diagram az egyszempontos varianciaelemzés során használható átlagábrákra mutat példát.

```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  stat_summary(fun=mean, geom="point", size=3, shape=21, fill="white")+
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
p2 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=attract)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  stat_summary(fun=mean, geom="line", size=0.6, aes(group=1),
               colour="grey32") +
  stat_summary(fun=mean, geom="point", size=3, shape=21,
               fill="white") +
  theme(legend.position = "none") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
p3 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=attract)) +
  stat_summary(fun=mean, geom="col", fill="white") +
  stat_summary(fun.data=mean_cl_normal, geom="pointrange") +
```

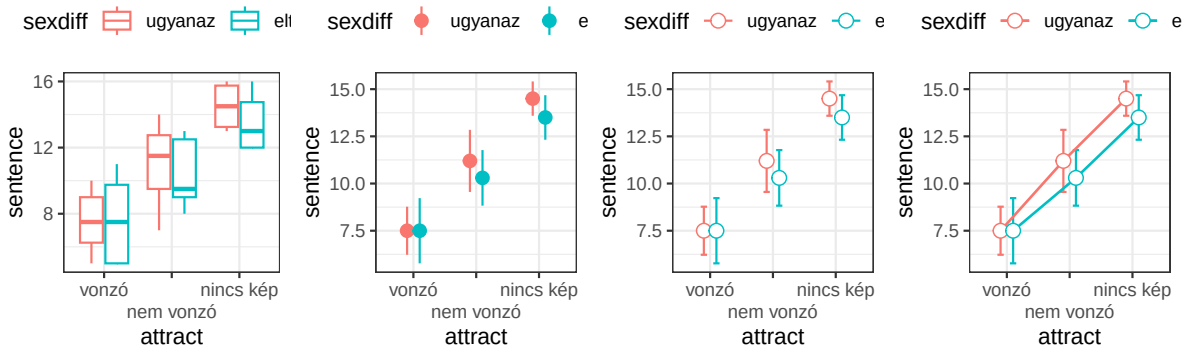
```
  theme(legend.position = "none") +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
p4 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  stat_summary(fun=mean, geom="col", fill="#95b0ff", colour="black") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti ábrákon a pont- és intervallumbecslések eredményét együtt mutatjuk be. Kétszempontos elemzésekhez is könnyen készíthetünk ábrát.

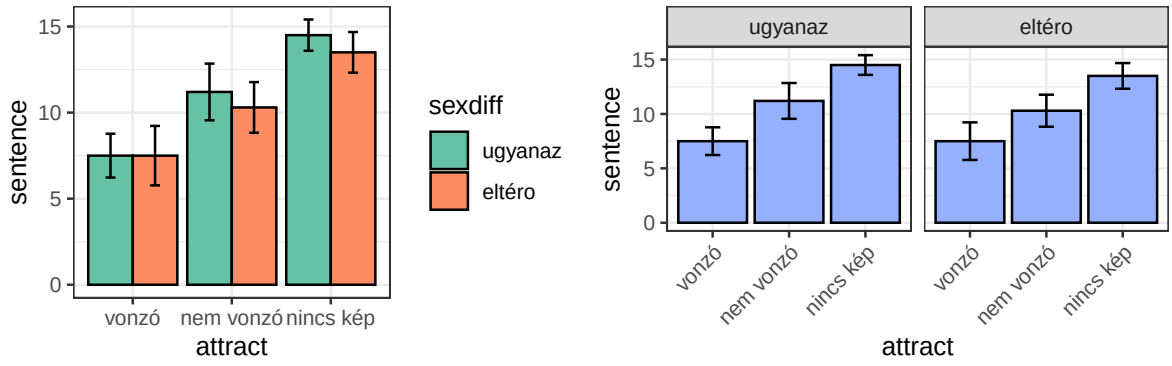
```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=sexdiff)) +
  geom_boxplot() +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position = "top")
p2 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=sexdiff)) +
  stat_summary(fun.data=mean_cl_normal, geom="pointrange",
               position = position_dodge(width=0.4)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position = "top")
p3 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=sexdiff)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2,
               position = position_dodge(width=0.4)) +
  stat_summary(fun=mean, geom="point", size=3, shape=21, fill="white",
               position =position_dodge(width=0.4)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position = "top")
p4 <- ggplot(vallomas, aes(x=attract, y=sentence, colour=sexdiff)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2,
               position = position_dodge(width=0.4)) +
  stat_summary(fun=mean, geom="line", size=0.6, shape=21, fill="white",
```

```
                aes(group=sexdiff),
                position = position_dodge(width=0.4)) +
  stat_summary(fun=mean, geom="point", size=3, shape=21, fill="white",
                position = position_dodge(width=0.4)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2)) +
  theme(legend.position = "top")
grid.arrange(p1, p2, p3, p4, ncol=4)
```



A fenti 1. ábra hagyományosnak tekinthető, mert két szempont elrendezett dobozdiagramokat mutat be. A 2-4. ábrák hasonló elrendezésben már pont- és intervallumbecsléseket tartalmaznak. A mintaátlagokat gyakran oszlopdiagramok segítségével ábrázoljuk, akár két faktor mentén elrendezve.

```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence, fill=sexdiff)) +
  stat_summary(fun=mean, geom="col", colour="black",
                position=position_dodge()) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2,
                position=position_dodge(width=0.9)) +
  scale_fill_brewer(palette="Set2")
p2 <- ggplot(vallomas, aes(x=attract, y=sentence, fill=sexdiff)) +
  stat_summary(fun=mean, geom="col", colour="black", fill="#95b0ff") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  facet_wrap(~sexdiff) +
  scale_x_discrete(guide = guide_axis(angle = 45))
grid.arrange(p1, p2, ncol=2)
```



A fenti ábratípusok mindegyikére igaz, hogy a kötelező változókon kívül a geom más paraméterei faktor vagy numerikus változó segítségével tovább finomíthatók. Amennyiben egy új faktort szeretnénk beépíteni a meglévő ábrába, akkor az megjelenhet a meglévő geom elem új paramétereként, például szín vagy kitöltőszín képében, de a `facet_wrap()` függvénnyel mátrixosan is felbonthatjuk az ábránkat az új faktor mentén. A fenti 2. ábrán erre mutattunk példát.

Összefoglalás

Nemcsak nyers adatokat tudunk az ábráinkon megjeleníteni. A dobozdiagram, hisztogram és oszlopdiagram például olyan származtatott adatokat (mediánt, gyakoriságot) jelenít meg, amelyek a nyers adatokból automatikusan számolódnak. A ggplot2 rendszer a háttérben végzi ezeket a számításokat, nem nekünk kell ezt kezdeményezni. Ha szeretnénk elérni ezeket a származtatott változókat, akkor az `after_stat()` függvényt kell használni. Azonban mi is végezhetünk számításokat a nyers adatokkal, tipikusan pont- és intervallumbecsléseket, de ezek eredményének megjelenítéséről már magunknak kell gondoskodni. A leggyakrabban a `stat_summary(fun=mean)` és a `stat_summary(fun.data=mean_cl_normal)` módosítókkal gondoskodunk mintaátlag és várható értékre vonatkozó 95%-os konfidenciaintervallum megjelenítéséről. A fenti becslések ábrára helyezése csak geom elemmel lehetséges, a `geom=` argumentum értéke tipikusan pontbecslés esetén `"col"` vagy `"point"`, intervallumbecslés esetén `"linerange"`, `"errorbar"`, `"pointrange"` vagy `"crossbar"`. A fenti lehetőségeket legtöbbször átlagábrák rajzolásához használjuk.

Feladatok

1. A 9.1 példában 237 Ausztrál egyetemista adata szerepel. Olvassuk be a `kerdoiv.xlsx` állományt, majd hasonlítsuk össze az életkort tartalmazó `Age` változó eloszlását a normális eloszlással. Használjunk hisztogramot és simított hisztogra-

mot is. Próbáljuk egyetlen ábrán elvégezni az összehasonlítást.

2. A 9.1 példában a nem (`Sex`) és a kezesség (`W.Hnd`) faktorok kapcsolata 4 fajta kereszttáblával is vizsgálható: (1) gyakorisági, (2) teljes kereszttáblára vonatkozó százalékos relatív gyakorisági, (3) soronként vett százalékos relatív gyakorisági és (4) oszloponként vett százalékos relatív gyakorisági táblázattal. Mind a négy kétdimenziós gyakorisági táblázatnak rajzoljuk meg az oszlopdiagram megfelelőjét. Az egyes oszlopok feliratként a gyakorisági és a százalékos relatív gyakorisági értékeket is tartalmazzák. A soronként vagy oszloponként vett relatív gyakorisági táblázatokból készült ábrák létrehozásához használjuk a `{GGally}` csomag `stat_prop()` függvényét.

9.3. Ábrák testreszabása 🤯

Miről lesz szó? Ebben a fejezetben

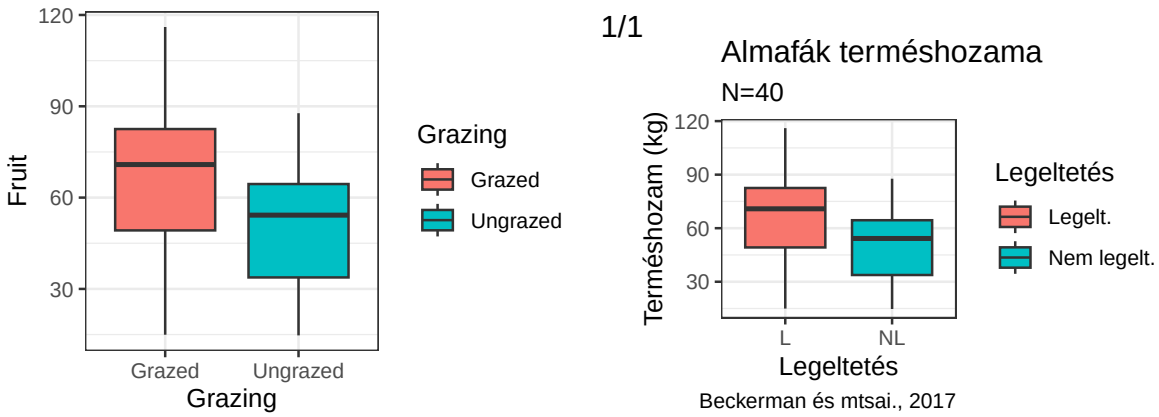
- áttekintjük, hogy az ábráink milyen módszerekkel lehetnek még tetszetősebbek és még jobban értelmezhetőek,
- így megtanuljuk a feliratok, jelmagyarázatok és a tengelyek beállításait,
- a színek és a témák kezelését,
- rácsozott (faceting) ábrák készítését,
- valamint az elkészült ábra annotálási lehetőségeit.

Megismerkedtünk a ggplot2 működésével és a legfontosabb ábratípusokkal. Ugyan ábráink szinte publikációkész állapotban vannak, kisebb nagyobb módosításokra még így is szükség lehet. Ezekből a finomítási lehetőségekből tekintünk át néhányat. Már eddig is számos lehetőséggel megismerkedtünk, de ezeket most szisztematikusan áttekintjük.

9.3.1. Feliratok

Egy ábra számos feliratot tartalmazhat és ezek nagy részét a `labs()` függvénnyel is beállíthatjuk. Megadhatjuk a főcímet (`title=`), az alcímet (`subtitle=`), képaláírást (`caption=`), címkét (`tag=`), az x tengely feliratát (`x=`) és az y tengely feliratát (`y=`). Amennyiben az ábrán jelmagyarázat is található, akkor a jelmagyarázat feliratát is megváltoztathatjuk úgy, hogy a jelmagyarázat alapját képező geom paraméter nevét meghatározzuk. Ilyen lehet például a `colour=`, `fill=` vagy `linewidth=` paraméterek beállítása. A jelmagyarázatokon megjelenő szövegek is beállíthatók a `scale_color_discrete()`, `scale_fill_discrete()` vagy `scale_linewidth_discrete()` függvények `labels=` argumentumával.

```
p1 <- ggplot(alma, aes(x = Grazing, y = Fruit, fill=Grazing)) +
  geom_boxplot() + scale_x_discrete(na.translate = F)
p2 <- p1 + labs(title = "Almafák terméshozama",
                subtitle = "N=40",
                caption = "Beckerman és mtsai., 2017",
                tag = "1/1",
                x = "Legeltetés",
                y = "Terméshozam (kg)",
                fill = "Legeltetés",
                colour = "Legeltetés") +
        scale_x_discrete(labels = c("Grazed" = "L",
                                    "Ungrazed" = "NL")) +
        scale_fill_discrete(labels = c("Grazed" = "Legelt.",
                                       "Ungrazed" = "Nem legelt.")) +
        scale_color_discrete(labels = c("Grazed" = "Legelt.",
                                        "Ungrazed" = "Nem legelt."))
grid.arrange(p1, p2, ncol=2)
```



A 9.8. táblázatban az ábrán megjelenő feliratok beállításait foglaltuk össze. Minden felirat szöveges tartalma beállítható a `labs()` függvénnyel, de bizonyos esetekben használhatjuk a `gtitle()`, `xlab()` és `ylab()` függvényeket is. A feliratok formátumát a `theme()` függvény megfelelő argumentumával állíthatjuk be.

9.8. táblázat: Az ábra feliratainak beállításai *(saját szerkesztés)*

Felirat	Szöveg tartalma	Szöveg formázása
cím	`labs(title=)`	`theme(plot.title=)`
	`ggtitle(label=)`	
alcím	`labs(subtitle=)`	`theme(plot.sub=)`
	`ggtitle(subtitle=)`	
képaláírás	`labs(caption=)`	`theme(plot.caption=)`
címke	`labs(tag=)`	`theme(plot.tag=)`
x tengely felirat	`labs(x=)`	`theme(axis.title.x=)`
	`xlab(label=)`	
y tengely felirat	`labs(y=)`	`theme(axis.title.y=)`
	`ylab(label=)`	

A formátum meghatározásához az `element_text()` függvényt használhatjuk, amely számos beállítási lehetőséggel rendelkezik (9.9. táblázat). Ha törölni szeretnénk az adott feliratot az ábráról, akkor az `element_blank()` függvényt használhatjuk.

9.9. táblázat: A szöveg formátumának beállítása az `element_text()` függvényben *(saját szerkesztés)*

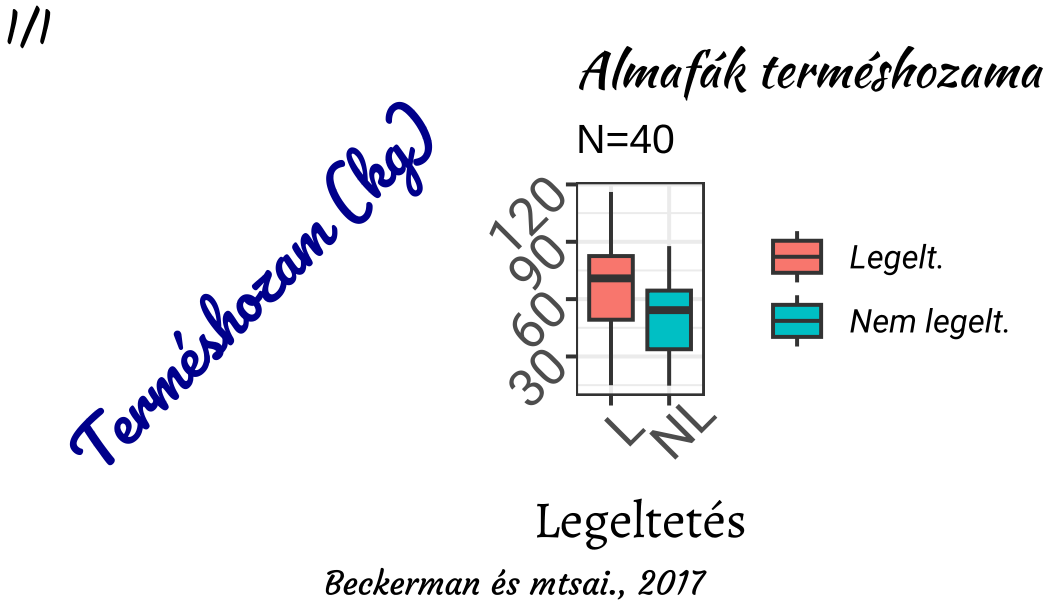
Paraméter	Jelentés
`family=`	betűtípus (pl. Times New Roman)
`face=`	betűstílus (pl. plain, italic, bold, bold.italic
`colour=`	szöveg színe
`size=`	szöveg mérete pt-ban
`hjust=`	horizontális igazítás(lehetséges értékek: [0, 1]
`vjust=`	vertikális igazítás (lehetséges értékek [0, 1])
`lineheight=`	sormagasság, többsoros szöveg esetén a sorközhöz
`angle=`	a szöveg forgatása (lehetséges értékek [0, 360])

A következő ábrát a már korábban elkészült `p2` ábra módosításával hozzuk létre. Az `element_text()` függvény lehetőségeire hívjuk fel a figyelmet az új ábrával. Első lépésként a `{showtext}` csomag segítségével betöltjük a Times New Roman és a Roboto betűtípusokat. A `{showtext}` csomag használatával a Google fontokat is betölthetjük, amelyeket a Google Fonts weboldalról tölthetünk le. A következő példában a Times New Roman és a Roboto betűtípusokat használjuk.

```
library(showtext)
# lokális font betöltése
font_add("Times New Roman", regular = "times.ttf")
## Google font betöltése (https://fonts.google.com/)
font_add_google("Roboto", "roboto")
font_add_google("Pacifico", "pacifico")
font_add_google("Alegreya", "alegreya")
font_add_google("Kalam", "kalam")
font_add_google("Kaushan Script", "kaushan")
font_add_google("Courgette", "courgette")
## Betöltött fontok használata
showtext_auto()
```

A fontok birtokában elkészítjük a `p2` ábrát, amelyen a cím, az alcím és a képaláírás betűtípusát is megváltoztatjuk. A `theme()` függvény segítségével a tengelyek feliratait és a tengelyek szövegét is átírjuk. A `theme()` függvény argumentumainak beállításait a 9.9. táblázatban található paraméterekkel végezzük el.

```
p2 + theme(plot.title=element_text(family = "kaushan"),
           plot.sub=element_text(family = "Times New Roman"),
           plot.caption=element_text(family = "courgette"),
           plot.tag=element_text(family = "kalam"),
           axis.title.y=element_text(family = "pacifico",
                                     face = "bold",
                                     colour = "darkblue",
                                     size=16,
                                     angle = 45,
                                     vjust=0.5),
           axis.title.x=element_text(family = "alegreya", size="14"),
           axis.text.x = element_text(family="roboto", angle=45, size=14),
           axis.text.y = element_text(family="roboto", angle=45, size=14),
           legend.title = element_blank(),
           legend.text = element_text(family="roboto", face="italic"))
showtext_auto(FALSE) # a fontok használatának kikapcsolása
```



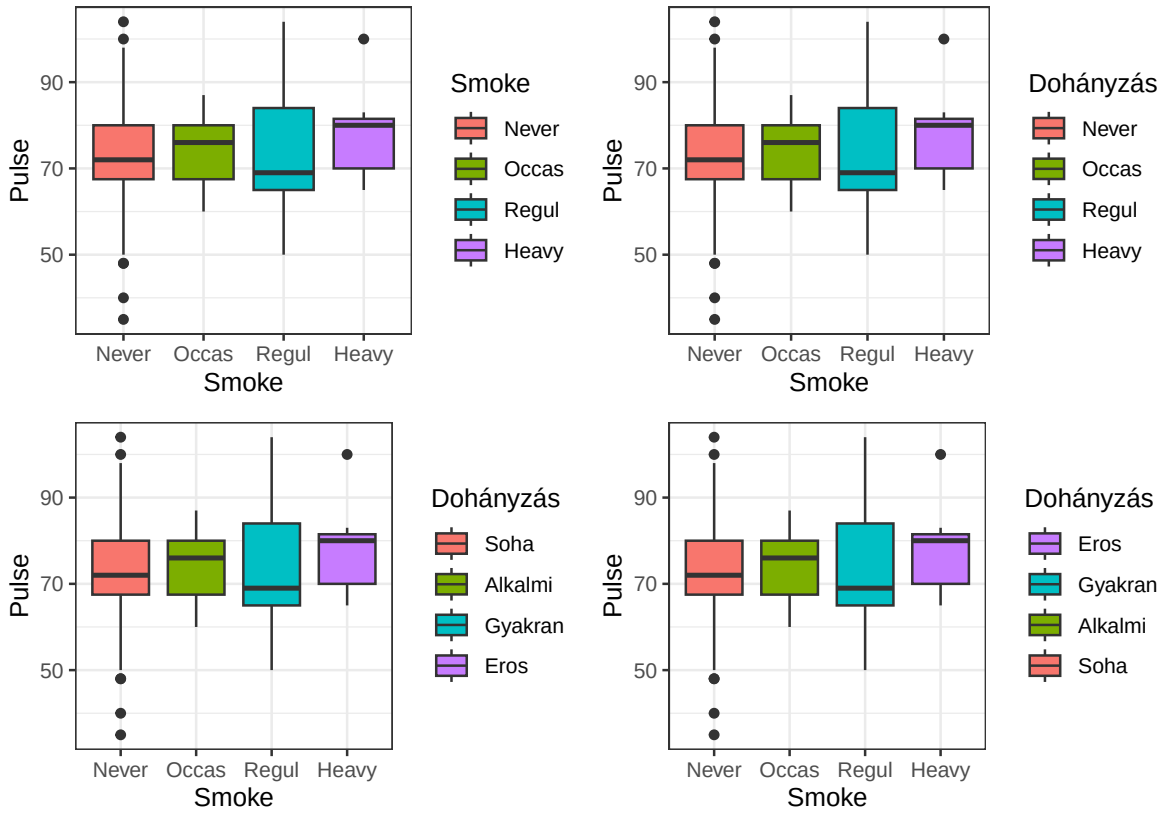
Beckerman és mtsai., 2017

9.3.2. Jelmagyarázat

Ábráinkat a jelmagyarázatok teszik beszédessé. A ggplot2 rendszer szükség esetén automatikusan megjeleníti a jelmagyarázatot. Már most megemlítjük, hogy a `theme(legend.position="none")` segítségével tilthatjuk meg a jelmagyarázat megjelenítését. A következő példában a leképezés részben használtuk a kitöltő színt (`fill=Smoke`), így a jelmagyarázat automatikusan megjelenik. A jelmagyarázatban lévő szövegek átírását, ahogyan korábban is láttuk már, a `labs(fill=)` és a `scale_fill_discrete(name=,labels=)` segítségével szabályozhatjuk. A jelmagyarázatban megjelenő szövegek sorrendjét a `guides(fill=guide_legend(reverse=T))` segítségével fordíthatjuk meg.

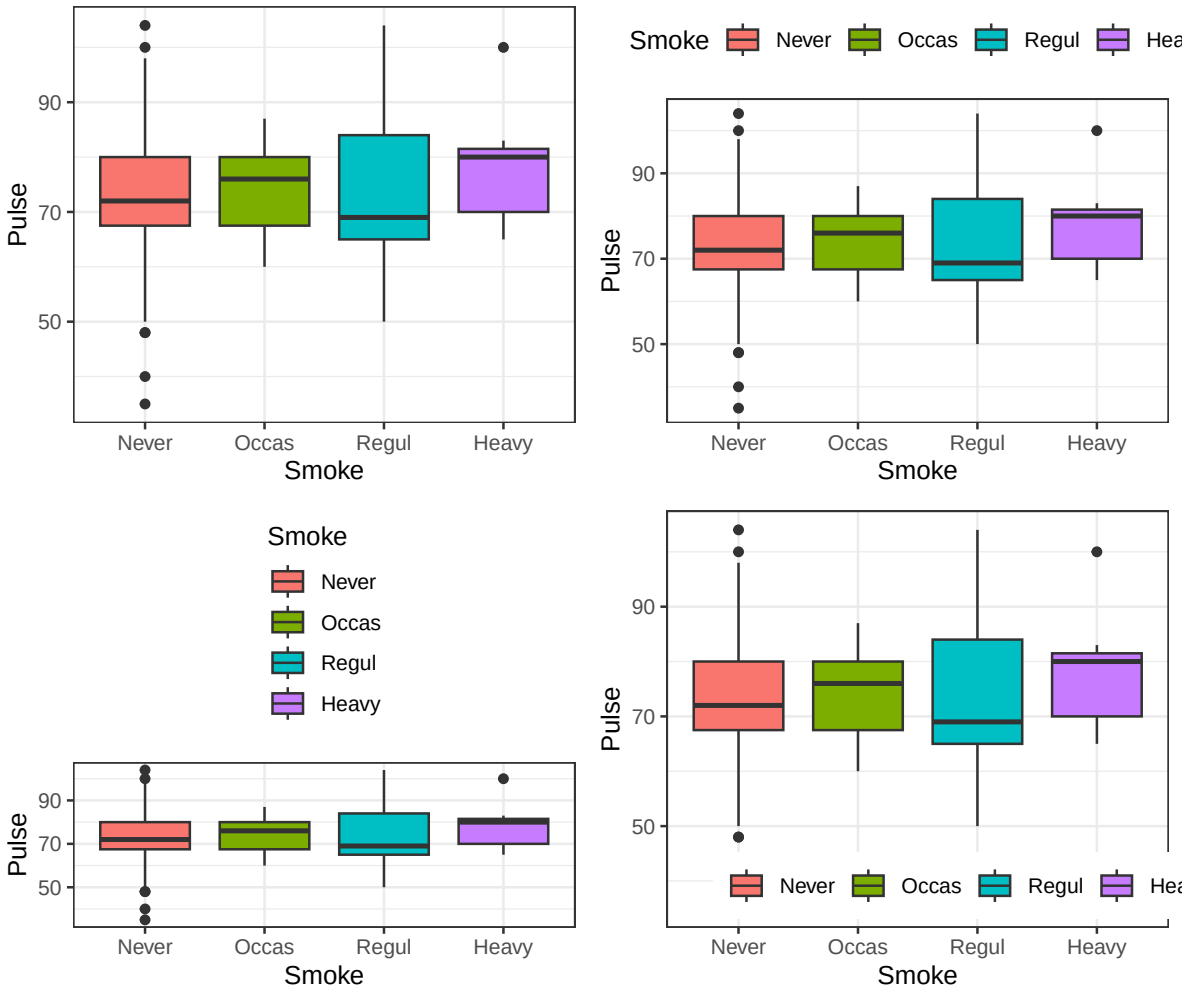
```
# a dohányzási szokás ordinális változó, a címkék sorrendjét átírjuk
kerdoiv$Smoke <- factor(kerdoiv$Smoke,
                        levels=c("Never", "Occas", "Regul", "Heavy"))
p1 <- kerdoiv |>  drop_na("Smoke") |>
  ggplot(aes(x=Smoke, y=Pulse, fill=Smoke)) + geom_boxplot()
p2 <- p1 + labs(fill="Dohányzás")
p3 <- p1 + scale_fill_discrete(name="Dohányzás",
                          labels=c("Never" = "Soha",
                                   "Occas" = "Alkalmi",
                                   "Regul" = "Gyakran",
                                   "Heavy" = "Erős"))
```

```
p4 <- p3 + guides(fill = guide_legend(reverse=T))
grid.arrange(p1, p2, p3, p4, ncol=2)
```

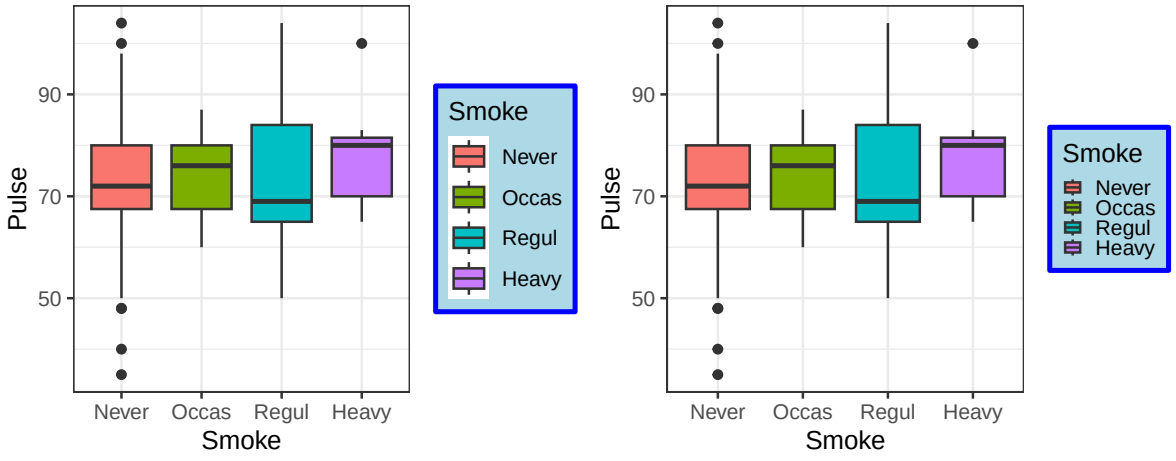


A jelmagyarázat helyét és irányát a kívánt módon állíthatjuk be.

```
p0 <- kerdoiv |>  drop_na("Smoke") |>
  ggplot(aes(x=Smoke, y=Pulse, fill=Smoke)) + geom_boxplot()
p1 <- p0 + theme(legend.position = "none")
p2 <- p0 + theme(legend.position = "top")
p3 <- p0 + theme(legend.position = "top",
                 legend.direction = "vertical")
p4 <- p0 + theme(legend.position = c(0.6, 0.1),
                 legend.direction = "horizontal",
                 legend.title = element_blank())
grid.arrange(p1, p2, p3, p4, ncol=2)
```




```
p0 <- kerdoiv |>  drop_na("Smoke") |>
  ggplot(aes(x=Smoke, y=Pulse, fill=Smoke)) + geom_boxplot()
p1 <- p0 + theme(legend.background = element_rect(fill="lightblue",
                                                  colour="blue",
                                                  linewidth = 1,
                                                  linetype="solid"))
p2 <- p1 + theme(legend.key = element_rect(fill="lightblue"),
                 legend.key.size = unit(0.3, "cm"))
grid.arrange(p1, p2, ncol=2)
```



9.10. táblázat: A jelmagyarázat megjelenésének beállításai *(saját szerkesztés)*

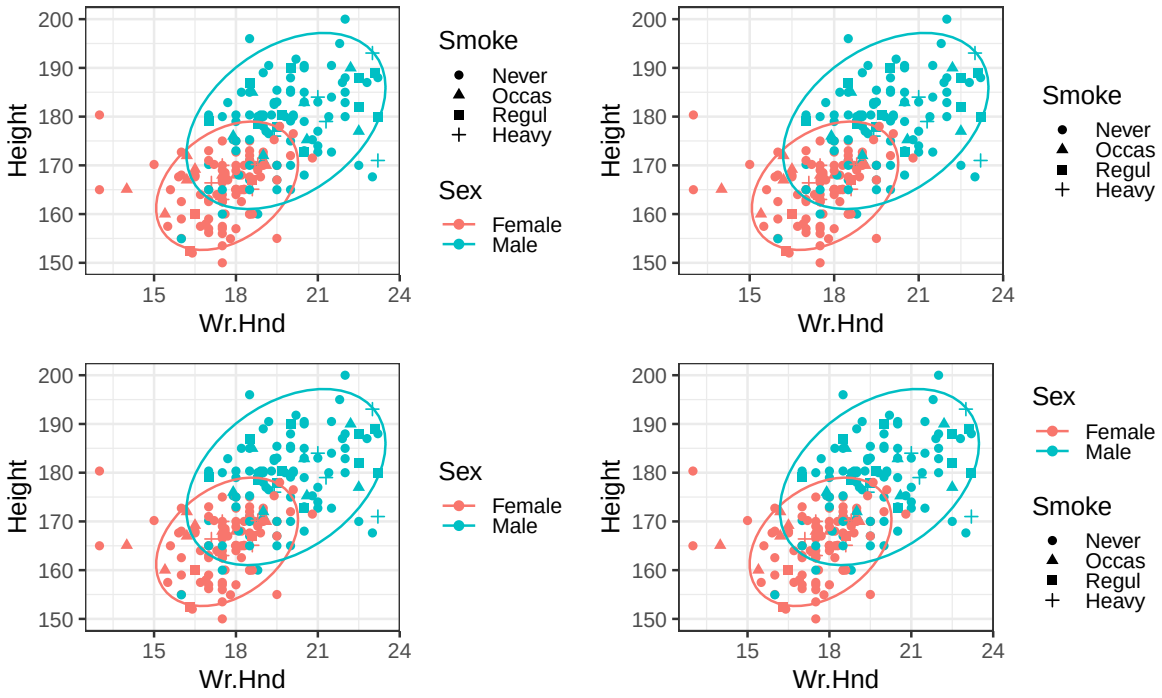
Jellemző	Formázó paraméter a `theme()` függvényben	Lehetséges érték(ek)
pozíció	`legend.position=`	`none, right, left, top, bottom` vagy `c(x, y)`, ahol `x` és `y` [0,1] közötti
irány	`legend.direction=`	`horizontal, vertical`
cím	`legend.title=`	`element_text()` lásd 9.9. táblázat
szöveg	`legend.text=`	`element_text()` lásd 9.9. táblázat
háttér	`legend.background=`	`element_rect(fill, colour, linewidth, linetype)`
kulcs	`legend.key=`	`element_rect(fill, colour)`
kulcs mérete	`legend.key.size=`	`unit()`

A `legend.position=` értéke lehet két szám, 0 és 1 között, ahol a c(0,0) megfelel a bal alsó saroknak, a c(1,1) a jobb felső saroknak.

A jelmagyarázatban megjelenő kulcsok sorrendje és a hozzá tartozó címkék is megváltoztathatók (9.10. táblázat).

```
p1 <- kerdoiv |>  drop_na("Sex", "Smoke") |>
  ggplot(aes(x=Wr.Hnd, y=Height, colour=Sex, shape=Smoke)) +
  geom_point() +
  stat_ellipse(aes(x=Wr.Hnd, y=Height, colour=Sex), inherit.aes = F) +
  theme(legend.spacing = unit(0.5, "cm"),
```

```
        legend.spacing.y = unit(0.1, "cm"),
        legend.spacing.x = unit(0.01, "cm"),
        legend.key.height = unit(0.2, "cm"))
p2 <- p1 + guides(colour=F)
p3 <- p1 + guides(shape=F)
p4 <- p1 + guides(colour=guide_legend(order=1),
                  fill=guide_legend(order=2))
grid.arrange(p1, p2, p3, p4, ncol=2)
```



A `guide_legends()` beállításai és a korábban használt beállítási lehetőségek között nagy átfedés van. Például az adott geom paraméterre vonatkozó címet a `title=` argumentummal is át tudjuk írni.

9.3.3. Tengelyek beállításai

Az ábra tengelyeinek beállítása legtöbb esetben a következő összetevőket jelenti:

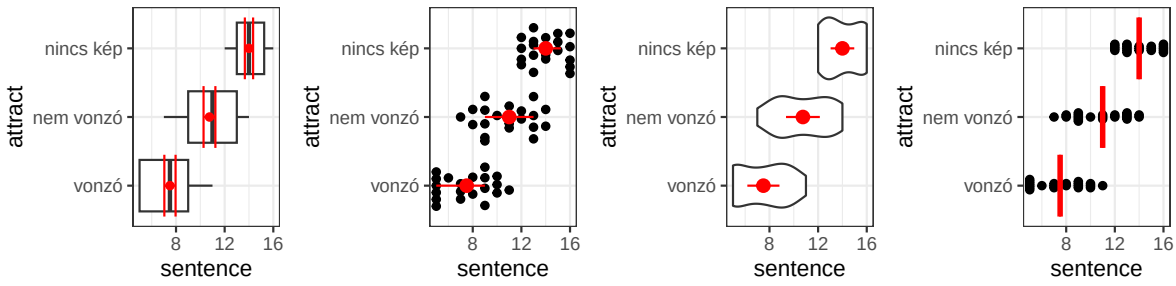
- tengelyek megfordítása (`coord_flip()`),

- láthatósági tartomány beállítása (`coord_cartesian(xlim=,ylim=)`),
- tengelyek átskálázása (például `scale_y_log10()`, `scale_y_sqrt()`, `scale_y_reverse()`),
- tengely osztásközök beállításai (például `scale_x_discrete` és `scale_y_continuous()`).

9.3.3.1. Tengelyek felcserélése

A lenti ábrákat már korábban megrajzoltuk. Most a `coord_flip()` függvénnyel egyszerűen felcseréljük az ábra x és y tengelyét.

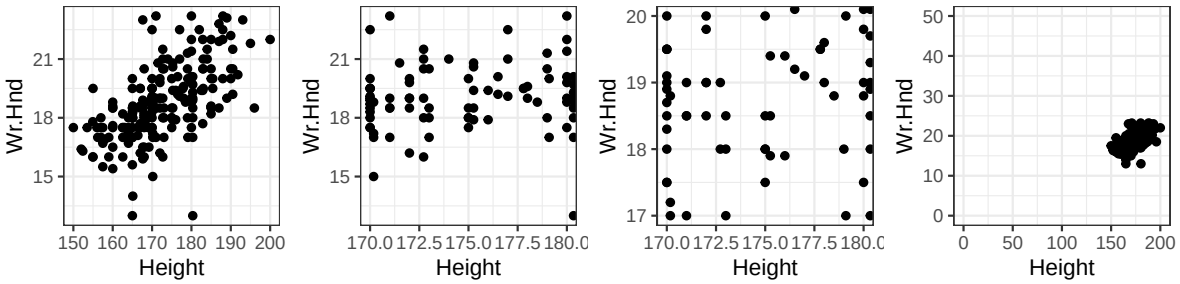
```
p1 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_boxplot() +
  stat_summary(fun.data = mean_se, geom="errorbar", col="red") +
  stat_summary(fun = mean, geom="point", col="red")
p1 <- p1 + coord_flip()
p2 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_quasirandom() +
  stat_summary(fun.data = median_hilow, fun.args = list(conf.int=0.5),
               geom="pointrange", col="red")
p2 <- p2 + coord_flip()
p3 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_violin() +
  stat_summary(fun.data = mean_cl_normal, col="red",
               fun.args = list(conf.int=0.99), geom="pointrange")
p3 <- p3 + coord_flip()
p4 <- ggplot(vallomas, aes(x=attract, y=sentence)) +
  geom_beeswarm() +
  stat_summary(fun = median, fun.min=median, fun.max=median,
               geom="crossbar", col="red")
p4 <- p4 + coord_flip()
grid.arrange(p1, p2, p3, p4, ncol=4)
```



9.3.3.2. Tengelyek láthatósági tartománya

A ggplot2 rendszer az x és y tengely ábrán látható tartományát automatikusan határozza meg a leképezésben megadott változók mintabeli értéktartománya alapján. Előfordul, hogy ezen szeretnénk változtatni. Használjuk a `coord_cartesian(xlim=,ylim=)` függvényt, ahol az x és y tengely határait a két-két végpont beállításával adhatjuk meg.

```
p1 <- ggplot(kerdoiv, aes(x=Height, y=Wr.Hnd)) + geom_point()
p2 <- p1 + coord_cartesian(xlim = c(170, 180))
p3 <- p1 + coord_cartesian(xlim = c(170, 180), ylim=c(17, 20))
p4 <- p1 + coord_cartesian(xlim = c(0, 200), ylim=c(0, 50))
grid.arrange(p1, p2, p3, p4, ncol=4)
```

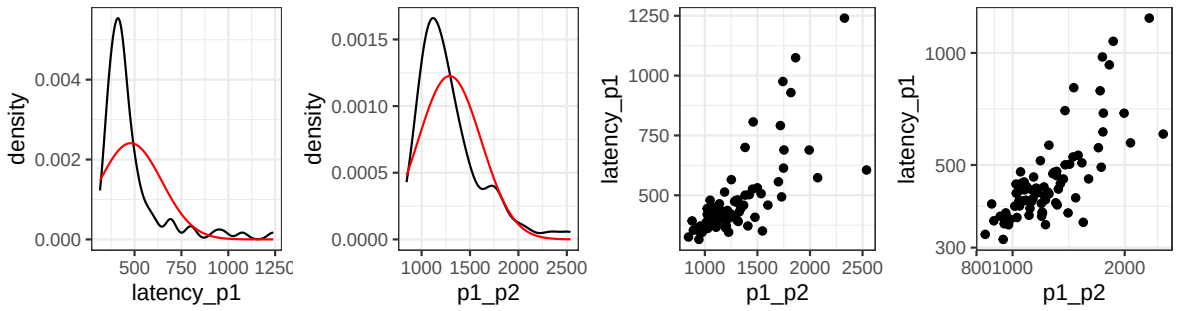


A fenti 1. ábrán az alapértelmezett tengelytartományokat láthatjuk, a 2-4. ábrákon pedig változtattunk ezeken. Vegyük észre, hogy milyen nagy mértékben járul hozzá az ábrák értelmezéséhez a tengelyek láthatósági tartománya.

9.3.3.3. Tengelyek skálázása

A tengelyek skálázásával jelentősen tudunk változtatni az ábránk megjelenésén. A `{fosdata}` csomag `brake` adatbázisa reakcióidőket tartalmaz (`latency_p1` és `p1_p2`) ms-ban mérve. A reakcióidők gyakran jobbra ferde eloszlásúak. A lenti 1-2. ábráról leolvasható, hogy mindkét vizsgált változónk valóban jobbra ferde, a könnyebb értelmezés miatt az adatokra illeszthető normális eloszlás sűrűségfüggvényét is berajzoltuk (piros görbe). A 3. ábrán a két jobbra ferde reakcióidő kétdimenziós pontdiagramját láthatjuk, míg a 4. ábrán ugyanezt, a két tengely 10-es alapú logaritmikus transzformációja után (`scale_x_log10()` és `scale_y_log10()`).

```
fek <- fosdata::brake
p1 <- ggplot(fek, aes(x=latency_p1))  + geom_density() +
  stat_theodensity(colour="red")
p2 <- ggplot(fek, aes(x=p1_p2)) + geom_density() +
  stat_theodensity(colour="red")
p3 <- ggplot(fek, aes(x=p1_p2, y=latency_p1)) + geom_point()
p4 <- ggplot(fek, aes(x=p1_p2, y=latency_p1)) + geom_point() +
  scale_x_log10() + scale_y_log10()
grid.arrange(p1, p2, p3, p4, ncol=4)
```



9.3.3.4. Osztásközök beállításai

A tengelyekkel kapcsolatos utolsó beállítási lehetőség, hogy az x és y tengelyen hol és milyen címkékkel rajzoljunk osztásközöket. A megjelenő címkék hozzájárulnak az ábra könnyebb értelmezéséhez, így jelentőségük nagyobb mint gondolnánk.

A példákban a következő egyszerű adatbázist fogjuk használni, egy karakteres oszloppal és két numerikussal.

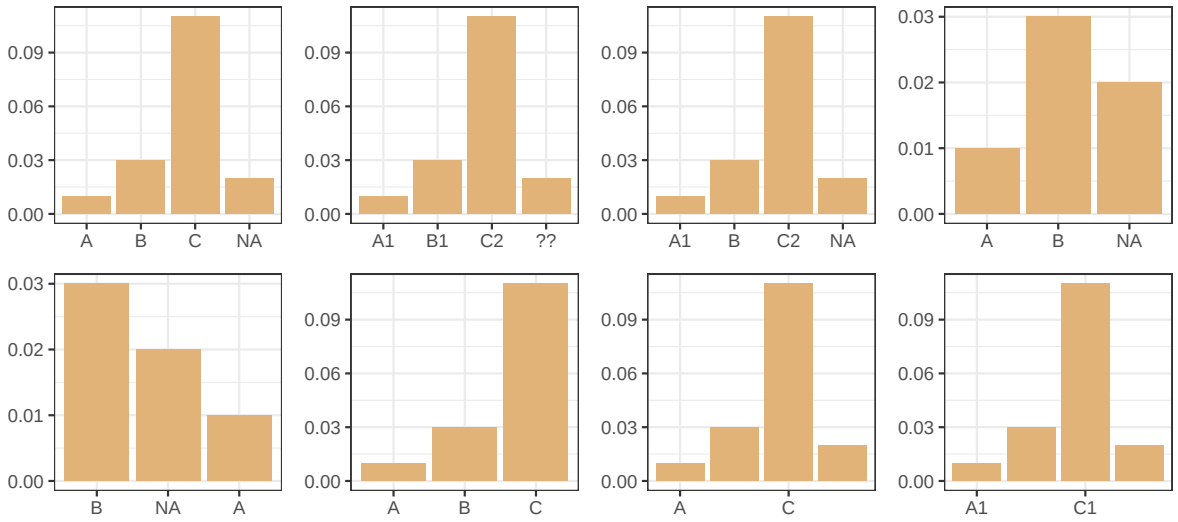
```
d.tbl <- tribble(
  ~csop, ~ertek, ~x,
  "A", 0.01, 2,
  "B", 0.03, 4,
  "C", 0.11, 5,
   NA, 0.02, 4.5
)
```

Amennyiben valamely tengelyhez kategorikus változó tartozik, akkor a `scale_x_discrete()` vagy `scale_y_discrete()` függvényt használhatjuk a tengelyek beállításaira. Összefoglaljuk a leggyakrabban használt paramétereket:

- `name=` - A tengely felirata.
- `breaks=` - Az osztásközök megadása, ami egyben a megjelenő léptékeket a rácsvonalakat is beállítja. Lehetséges értékei például:
  - `NULL` - Minden osztásköz eltávolítása.
  - Karakteres vagy numerikus vektor, amely a megjelenő osztásközöket tartalmazza.
- `labels=` - Az egyes osztásközök felirata. Lehetséges értékei például:
  - `NULL` - Minden osztásköz feliratának eltávolítása.
  - Karakteres vagy numerikus vektor, amely a megjelenő osztásközök feliratát tartalmazza, így átnevezésre is van lehetőségünk.
- `limits=` - Karakteres vektor, amely a megjelenő adatkategóriákat tartalmazza a megfelelő sorrendben.

A fenti paraméterek használatára mutat példát a következő nyolc ábra.

```
# alapértelmezett megjelenítés
p1 <- ggplot(d.tbl, aes(x=csop, y=ertek)) +
  geom_col(fill="#E1B378") + labs(x=NULL, y=NULL)
 # átnevezzük az x tengely értékeit, mindet
p2 <- p1 + scale_x_discrete(labels=c("A1", "B1", "C2", "??"))
# átnevezzük az x tengely értékeit, néhányat
p3 <- p1 + scale_x_discrete(labels=c("A"="A1", "C"="C2"))
# nem kell minden az x tengelyről
p4 <- p1 + scale_x_discrete(limits=c("A","B", NA))
# sorrend az x tengelyen
p5 <- p1 + scale_x_discrete(limits=c("B", NA, "A"))
# NA eltávolítása
p6 <- p1 + scale_x_discrete(na.translate = F)
# Csak bizonyos jelölések jelenjenek meg
p7 <- p1 + scale_x_discrete(breaks=c("A", "C"))
# Csak bizonyos jelölések jelenjenek meg, átnevezve
p8 <- p1 + scale_x_discrete(breaks=c("A", "C"), labels=c("A1", "C1"))
grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, ncol=4)
```



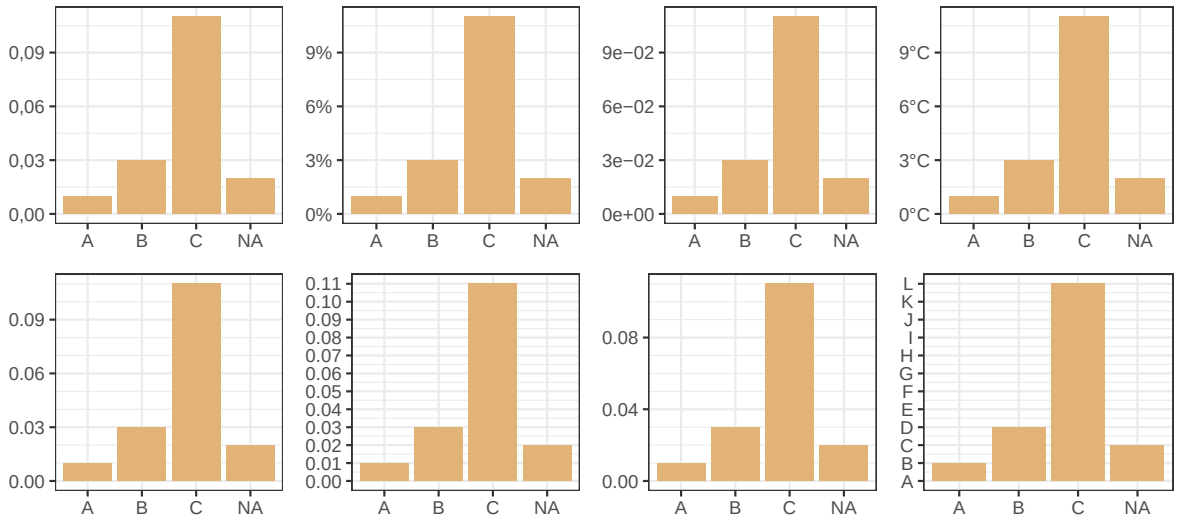
Numerikus tengelyek esetében a `scale_x_continuous()` vagy `scale_y_continuous()` függvényeket használhatjuk az osztásközök beállítására. A szokásos argumentumok a következők:

- `name=` - A tengely felirata.
- `breaks=` - Az osztásközök megadása, ami egyben a megjelenő fő léptékeket a fő rácsvonalakat is beállítja. Lehetséges értékei például:
  - `NULL` - Minden osztásköz eltávolítása.
  - Numerikus vektor, amely a megjelenő fő osztásközöket tartalmazza.
- `minor_breaks=` - A kis osztásközök megadása, ami egyben a megjelenő kis léptékeket a kis rácsvonalakat is beállítja. Lehetséges értékei például:
  - `NULL` - Minden kis osztásköz eltávolítása.
  - Numerikus vektor, amely a megjelenő kis osztásközöket tartalmazza.
- `labels=` - A fő osztásközök felirata. Lehetséges értékei például:
  - `NULL` - Minden fő osztásköz feliratának eltávolítása.
  - Karakteres vektor, amely a megjelenő fő osztásközök feliratát tartalmazza.
  - Egy függvény, amely előállítja a feliratokat. Gyakran használjuk a következő függvényeket a `{scales}` csomagból:
    * `label_number(accuracy,scale,prefix,suffix,big.mark,decimal.mark)` - hagyományos, tizedes törtes megjelenítés
    * `label_number(accuracy,scale,prefix,suffix,big.mark,decimal.mark)` - százalékos formára konvertált megjelenítés
    * `label_scientific(digits,scale,prefix,suffix,decimal.mark)` - exponenciális alakú számok megjelenítése

- `limits=` A megjelenő értékek kezdő és végpontja, `limits=c(<min>, <max>)` formában.

Látható, hogy numerikus tengely esetén egy kicsivel több lehetőségünk van a beállítások során, mint kategorikus tengely esetében. A következő nyolc ábra a fenti paraméterek beállításaira mutat példát.

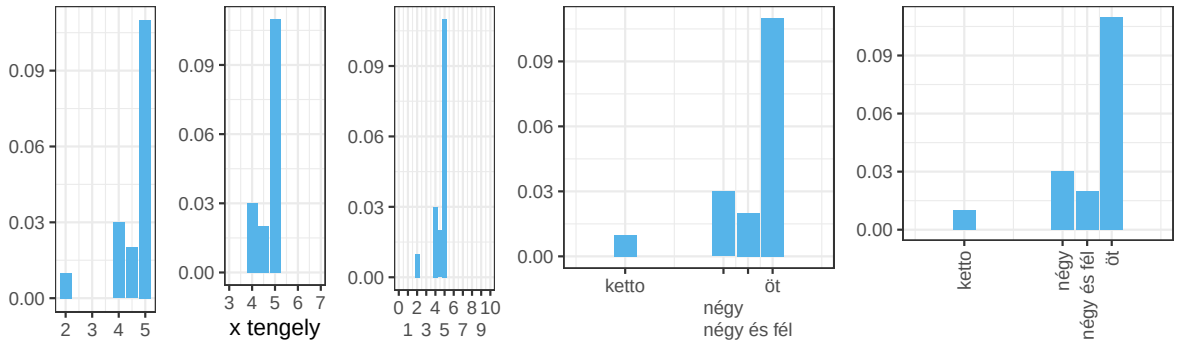
```
# ebből az ábrából indulunk ki
p0 <- ggplot(d.tbl, aes(x=csop, y=ertek)) +
  geom_col(fill="#E1B378") + labs(x=NULL, y=NULL)
# a tizedes törtes megjelenítés előírása, tizedesvessző beállítása
p1 <- p0 +
  scale_y_continuous(labels=label_number(decimal.mark = ",",
                                         accuracy=0.01))
# százalékos forma előírása, tizedesvessző beállítása
p2 <- p0 +
  scale_y_continuous(labels = label_percent(decimal.mark = ",",
                                            accuracy = 1))
# exponenciális alak előírása, tizedesvessző beállítása
p3 <- p0 + scale_y_continuous(labels=label_scientific())
# skálázás (szorzás 100-zal) és suffix beállítása (Celsius)
p4 <- p0 + scale_y_continuous(labels=label_number(decimal.mark = ",",
                                                  scale = 100,
                                                  accuracy = 1,
                                                  suffix="\u00b0C"))
# kis léptékek beállítása
p5 <- p0 + scale_y_continuous(minor_breaks = seq(0, 0.12, 0.01))
# nagy léptékek beállítása
p6 <- p0 + scale_y_continuous(breaks = seq(0, 0.12, 0.01))
# nagy és kis léptékek
p7 <- p0 +
  scale_y_continuous(breaks = seq(0, 0.12, 0.04),
                     minor_breaks = seq(0, 0.12, 0.01))
# nagy léptékek beállítása és a feliratok megváltoztatása
p8 <- p0 +
  scale_y_continuous(breaks = seq(0, 0.12, 0.01),
                     labels = LETTERS[1+100*seq(0, 0.12, 0.01)])
grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, ncol=4)
```



A további példákban egy új ábrából indulunk ki, ahol az x tengely a numerikus. Fontos megjegyezni, hogy a `guides(x = guide_axis(n.dodge = 2))` módosítóval az x tengelyen 2 soros osztásköz feliratot tudunk beállítani. A `guides()` függvény számos korábban megismert beállítási lehetőség alternatíváját kínálja.

```
# új ábra
p1 <- ggplot(d.tbl, aes(x=x, y=ertek)) + geom_col(fill="#56B4E9") +
  labs(x=NULL, y=NULL)
# nem minden értéket szeretnénk látni, plusz tengelyfelirat
p2 <- p1 + scale_x_continuous(limits = c(3, 7),
                              name="x tengely")
# hol legyen fő lépték
p3 <- p1 + scale_x_continuous(breaks = seq(0, 10, 1),
                              limits=c(0, 10)) +
  guides(x = guide_axis(n.dodge = 2))
# szöveges felirat
p4 <- p1 +
  scale_x_continuous(breaks = c(2, 4, 4.5, 5),
                     limits = c(1, 6),
                     labels=c("kettő","négy", "négy és fél", "öt")) +
  guides(x = guide_axis(n.dodge = 3))
# szöveges felirat, plusz eltolás az osztások feliratában
p5 <- p1 +
  scale_x_continuous(breaks = c(2, 4, 4.5, 5),
                     limits = c(1, 6),
```

```
                    labels=c("kettő","négy", "négy és fél", "öt")) +
  guides(x = guide_axis(angle=90))
grid.arrange(p1, p2, p3, p4, p5, ncol=5,
             layout_matrix=rbind(c(1,2,3,4,4,5,5)))
```



9.3.4. Színek

A színek használatával ábráink szebbek és jobban értelmezhetőek lehetnek. Korábban számos alkalommal adtunk meg közvetlenül színt. Például a hisztogram téglalapjainak kitöltőszínét (`fill=`) vagy keretszínét (`colour=`) is meghatároztuk (9.2 ábra). Amikor R-ben egy konkrét színre hivatkozunk 3 módszert használhatunk:

- szín megnevezése (például `"blue"`, `"lightblue"`; az összes használható színnév a `colors()` függvénnyel kiíratható)
- hexadecimális kód (például `"#000000"`, `"#FFFFFF"`, ahol a piros (R), zöld (Z) és kék (K) mennyiségét adjuk meg a színben #RRGGBB formában)
- az `rgb(maxColorValue = 255)` függvény, ahol 0 és 255 közötti számmal adjuk meg a piros, zöld és kék komponensek értékét (például `rgb(red=255, green=165, blue=0, maxColorValue = 255)`).

A fentiek alapján, amikor a korábban említett 9.2 ábrán a `fill="lightblue"` paramétert használtuk, akkor a `fill="#ADD8E6"` és a `fill=rgb(red = 173, green = 216, blue = 230, maxColorValue = 255)` kifejezéseket is megadhattuk volna. A `col2rgb()` függvénnyel magunk is meggyőződhetünk a három színmegadási mód azonosságáról.

```
# ugyanaz a szín háromféle megadási móddal
col2rgb("lightblue")
#>       [,1]
#> red    173
```

```
#> green  216
#> blue   230
col2rgb("#ADD8E6")
#>       [,1]
#> red    173
#> green  216
#> blue   230
col2rgb(rgb(red = 173, green = 216, blue = 230, maxColorValue = 255))
#>       [,1]
#> red    173
#> green  216
#> blue   230
```

Amikor a `fill=` vagy `colour=` paraméterekkel színek megjelenésére adunk utasítást, akkor figyelnünk kell, hogy azt a leképezésen (`aes()` függvényen) belül tesszük meg, vagy valahol azon kívül. Megkülönböztetünk tehát két esetet, ahol a színeket specifikálhatjuk:

- leképezésen belül, azaz az `aes()` függvényben, ekkor a szín megadásához az adatmátrix változóit használjuk, faktorokat vagy numerikus vektorokat:
  - `fill=<változónév>`
  - `colour=<változónév>`

- leképezésen kívül, azaz közvetlenül a `ggplot()` függvényben, vagy valamelyik `geom_*()` függvényben, ekkor konkrét színt adunk meg:
  - `fill="white"`
  - `colour="#c1c1c1"`.

Leképezésen belül, változók megadásával akkor írjuk elő színek megjelenését, ha az ábra tartalmát, értelmezését, felépítését szeretnénk megváltoztatni. A fejezet további részében ezeket az eseteket vesszük sorra. Leképezésen kívül, konkrét színkódok megadásával akkor adunk meg színt, ha szépíteni szeretnénk ábránk megjelenésén. Ebbe a kategóriába tartozott a már korábban említett 9.2. hisztogram kék színűre színezése is.

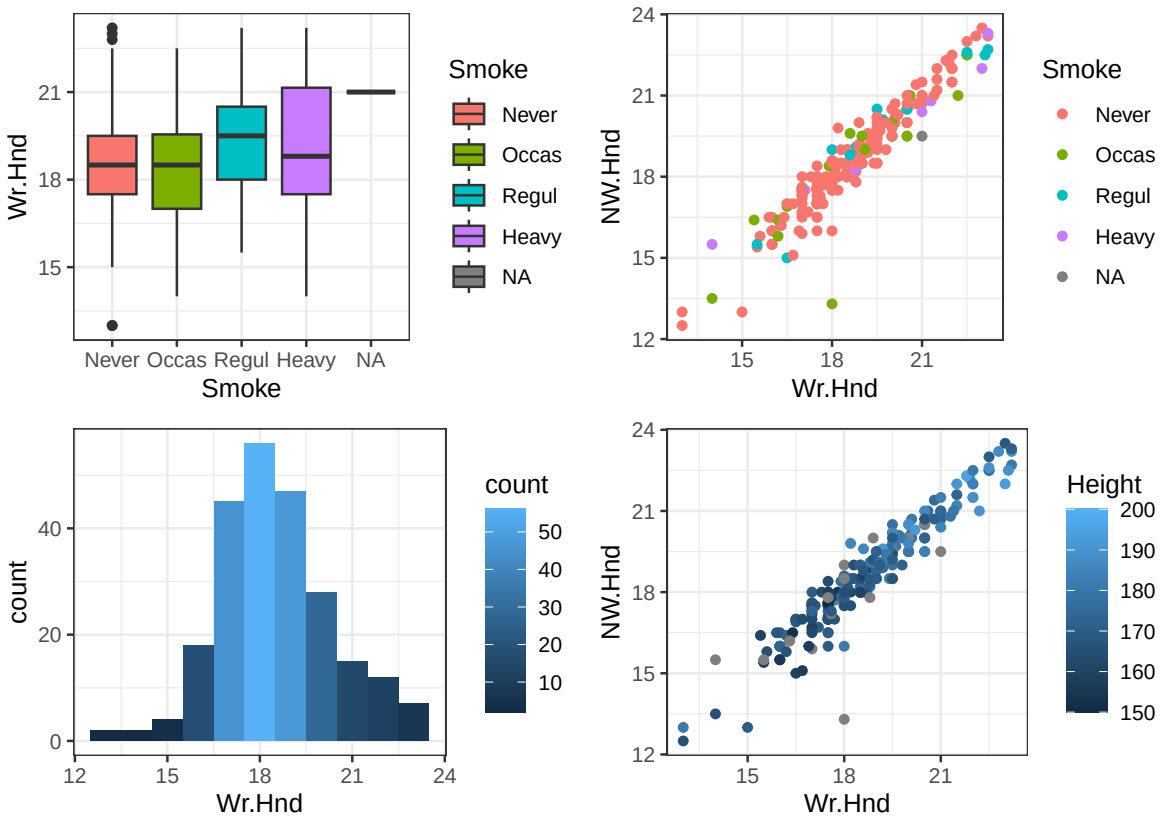
A színezésre megfogalmazott fenti szabály a geom elemek más paramétereire is igazak. Tudnunk kell, hogy a `linetype=`, `linewidth=`, `shape=` `size=`, `alpha=` és egyéb geom paramétereket a leképezésen belül vagy azon kívül kívánjuk használni. Az első esetben ezeknek a geom paramétereknek konkrét jelentése van, míg a második esetben csak az ábra külalakján finomítanak.

A továbbiakban a jelentéssel bíró, az ábra értelmezését meghatározó színezéseket fogjuk áttekinteni. Ilyenkor a leképezésen belül írjuk elő színek használatát, nem konkrét színkódot adunk meg, hanem változót. A ggplot2 rendszer automatikusan gondoskodik arról, hogy a változó értékeihez milyen konkrét színkódok fognak tartozni. Ezt az automatikus folyamatot, vagyis a változóérték-színkód összerendelést közvetett és közvetlen módon is felülbírálhatjuk. Ezt a két esetet tekintjük át részletesebben.

9.3.4.1. Színek közvetett beállítása

A ggplot2 rendszerben a színek alkalmazása automatikusan történik, amikor a leképezésben (`aes()` függvényben) a szokásos `x=` és/vagy `y=` paraméterek mellett a `colour=` vagy `fill=` argumentumokat is használjuk. Ekkor receptszerűen előírjuk, hogy ábránkon a megadott kategorikus vagy numerikus változó értékének megfelelően több szín is jelenjen meg. De melyek lesznek a ténylegesen használt színkódok?

```
p1 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot()
p2 <- ggplot(kerdoiv, aes(x=Wr.Hnd, y=NW.Hnd, colour=Smoke)) +
  geom_point()
p3 <- ggplot(kerdoiv, aes(x=Wr.Hnd, fill=after_stat(count))) +
  geom_histogram(binwidth=1)
p4 <- ggplot(kerdoiv, aes(x=Wr.Hnd, y=NW.Hnd, colour=Height)) +
  geom_point()
grid.arrange(p1, p2, p3, p4, ncol=2)
```



A fenti 1-2. ábra kategorikus változót (`Smoke`) rendel a `fill=` (1. ábra) és a `colour=` (2. ábra) argumentumhoz. A jelmagyarázatban a `Smoke` értékei és az automatikusan hozzárendelt színeket látjuk. Az automatikus hozzárendeléshez a ggplot2 a `scale_fill_hue()` és `scale_colour_hue()` függvényt használja, amelyek valójában a `scales::hue_pal()(4)` függvényt hívják, mivel a `Smoke` faktornak négy szintje van. A használt színkódok a következők:

```
# alapértelmezett színkódok
(scales::hue_pal())(4)
#> [1] "#F8766D" "#7CAE00" "#00BFC4" "#C77CFF"
```

A fenti 3-4. ábra numerikus változót rendel a `fill=` (3. ábra) és a `colour=` (4. ábra) argumentumhoz. A jelmagyarázatban egy színskála szerepel, amely a `count`, illetve a `Height` változó értékeinek teljes tartományát reprezentálja. A ggplot2 a `scale_fill_gradient()` és a `scale_colour_gradient()` függvényeket használja a színskála összeállításához.

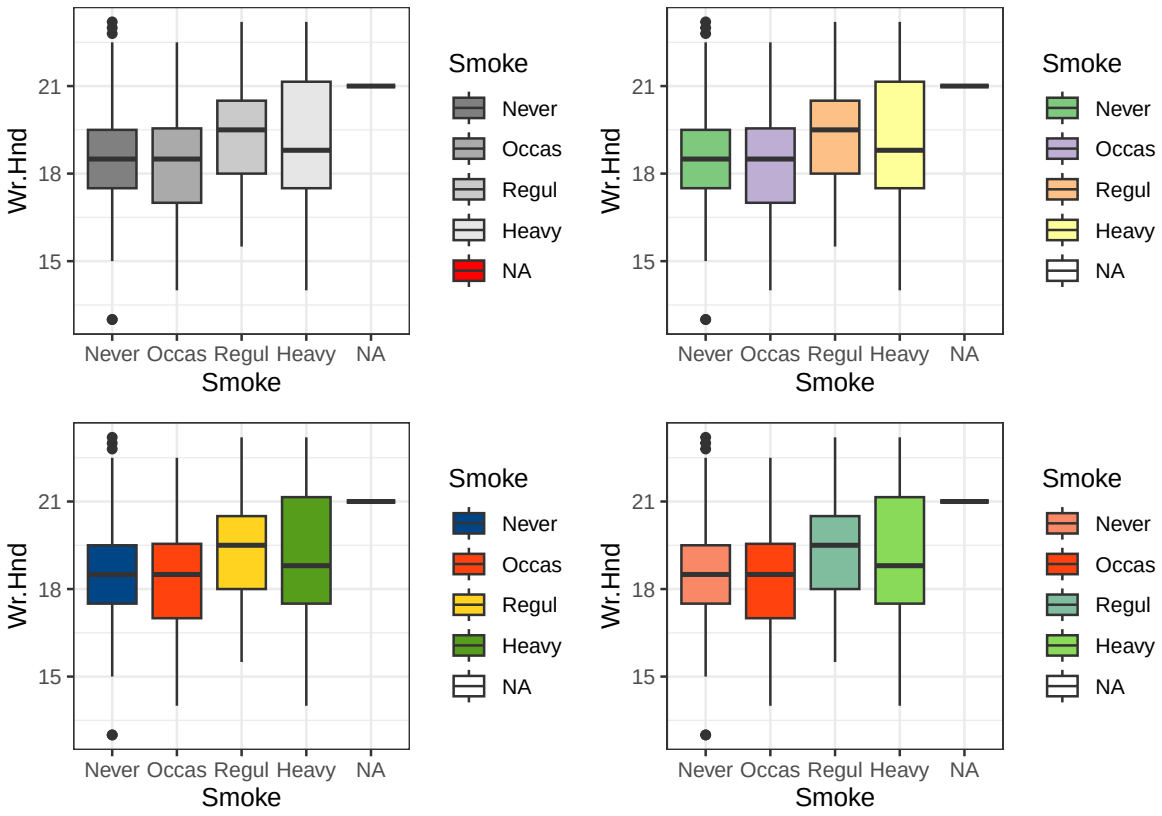
Látjuk, hogy a ggplot2 a fenti ábrák létrehozásához nem kért tőlünk konkrét színkódokat, mégis a faktor vagy numerikus változó egyes értékeinek reprezentáláshoz automatikusan megjelentek bizonyos színek. Ezeken természetesen magunk is változtathatunk.

Felsorolunk néhány lehetőséget kitöltőszín (`fill=`) automatikus beállítására (a lenti függvényekben a `fill` helyett a `colour` szót használjuk, ha a `colour=` szerepel a leképezésben):

- Kategorikus változó esetén:
  - `scale_fill_hue()` - ez az alapértelmezés
  - `scale_fill_grey()` - szürke színárnyalatok magadása
  - `scale_fill_viridis_d()` - színek a `{viridisLite}` csomagból
  - `scale_fill_brewer()` - színek a `{RColorBrewer}` csomagból
  - `scale_fill_calc()` - színek a LibreOffice Calc-ból
  - `ggthemes::scale_fill_canva()` canva.com színpalettái
  - `ggthemes::scale_fill_colorblind()` színpaletta színtévesztők számára
  - `ggthemes::scale_fill_pander()` színpaletta a `{pander}` csomagból
  - `ggthemes::scale_fill_wsj()` Wall Street Journal színpaletta
  - `ggthemes::scale_fill_tableau()` Tableau színpalettája
  - `ggthemes::scale_fill_stata()` Stata színpalettája
  - `ggthemes::scale_fill_hc()` Highcharts színpaletta
- Numerikus változók esetén:
  - `scale_fill_gradient()`
  - `scale_fill_gradient2()`
  - `scale_fill_gradientn()`
  - `ggthemes::scale_fill_gradient_tableau()`

A fenti függvények az alapértelmezett színpalettát cserélik le, így az ábrán már más színek fognak megjelenni. A fenti függvények többsége számos paraméterrel tovább specifikálható. Például a `scale_fill_grey(start = 0.5, end = 0.9)` az alapértelmezett ([0, 1]) feketétől fehérig terjedő színárnyalatokból való választást eltolja a világosak irányába ([0,5;0,9]). A `scale_fill_brewer(palette = "Accent")` a számos előre definiált paletta közül az `Accent` nevűt választja a változóérték-színkód összerendelés során. A következő négy ábra az alapértelmezett színpaletta megváltoztatására mutat példát kategorikus változó esetében.

```
p1 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot() + scale_fill_grey(start = 0.5, end = 0.9)
p2 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot() + scale_fill_brewer(palette = "Accent")
p3 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot() + scale_fill_calc()
p4 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot() + scale_fill_canva()
grid.arrange(p1, p2, p3, p4, ncol=2)
```

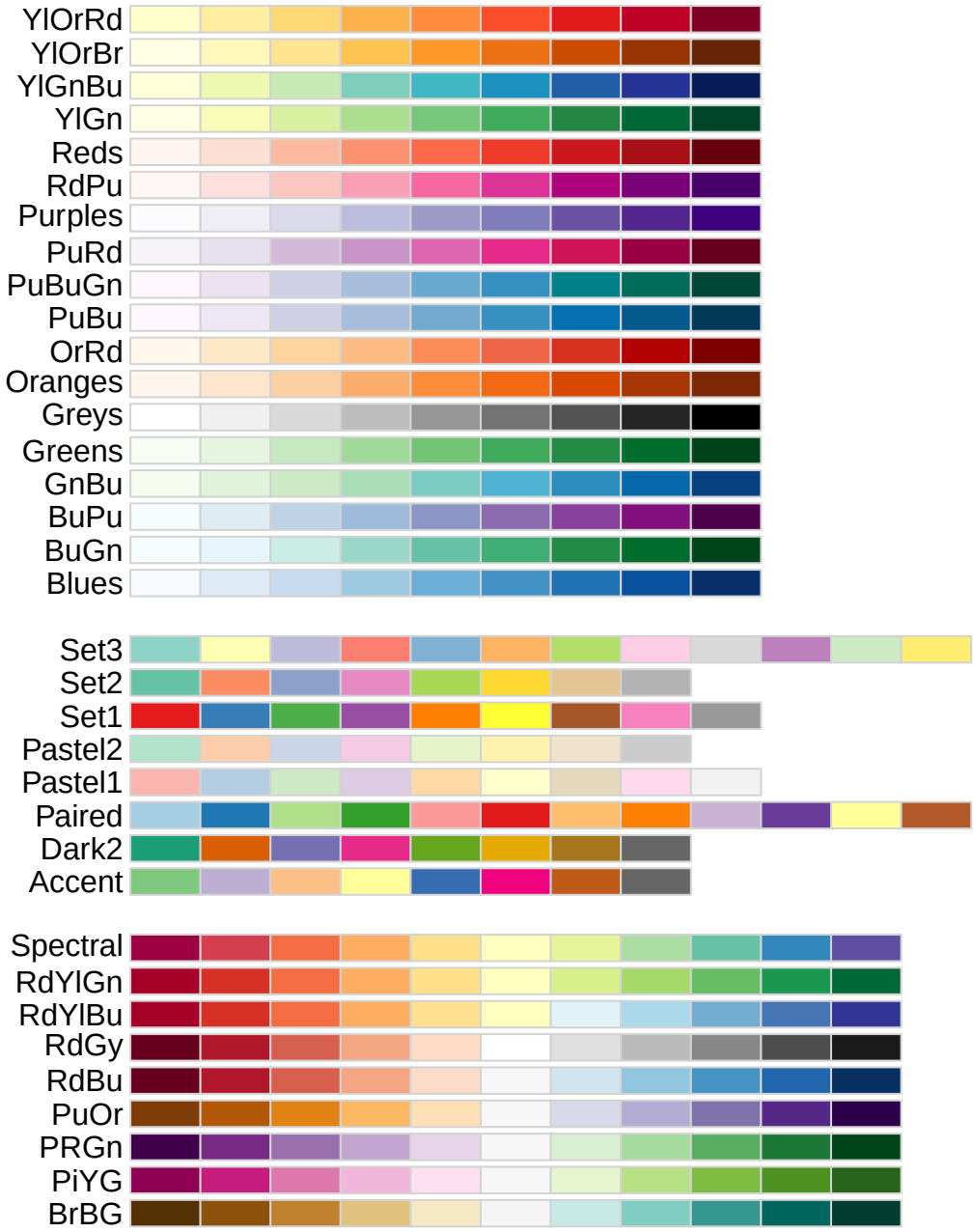


9.4. ábra: Alapértelmezett színpaletta megváltoztatása kategorikus változó esetén

Az `{RColorBrewer}` csomag színpalettái különösen hasznosak lehetnek ábráink színezésére. Kategorikus változó esetén használható összes színpalettát a 9.5. ábra tartalmazza (`RColorBrewer::display.brewer.all()`).

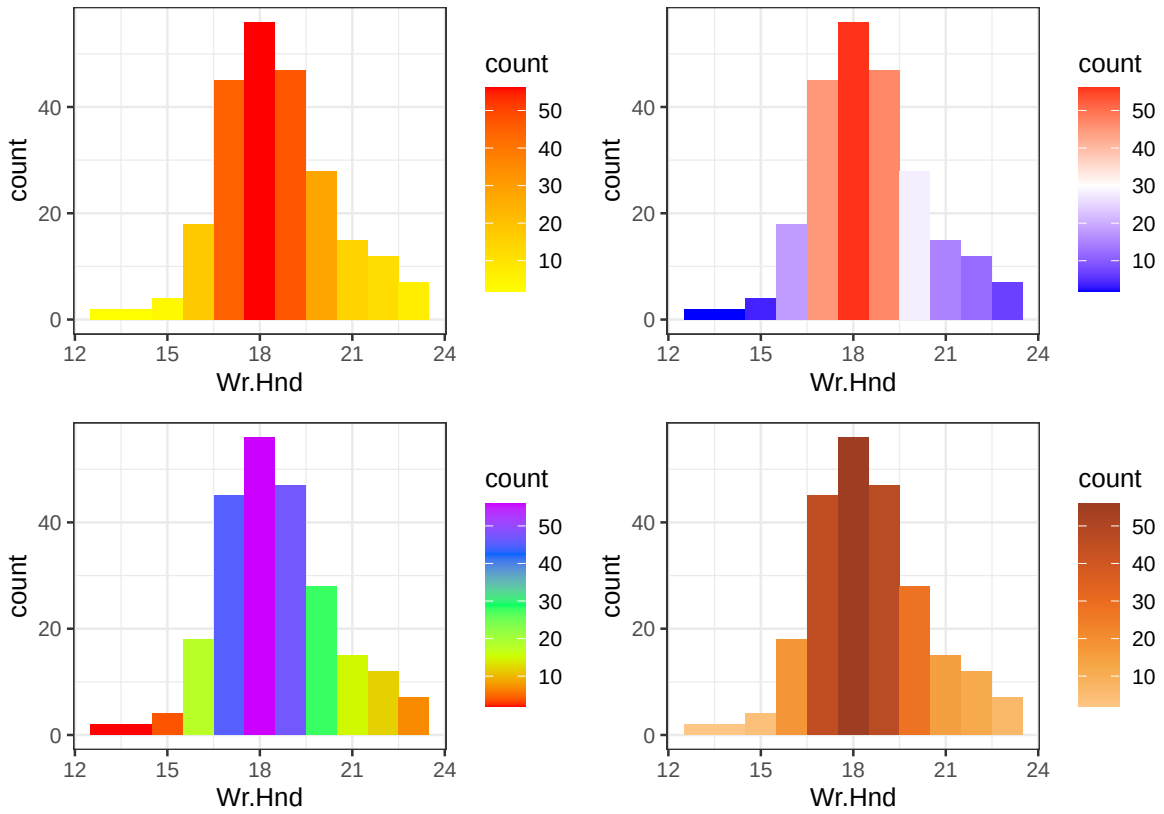
Numerikus változók esetén is van módunk állítani az előre definiált változóérték-színkód összerendelésen. Az alapértelmezett `scale_fill_gradient(low = "#132B43", high = "#56B1F7")` összerendelés a sötétkéktől a világoskékig tartó intervallumból választ színeket. Ezt például a `low = "yellow", high = "red"` paraméterekkel felülírhatjuk. Ha a `scale_fill_gradient2()` függvényt használjuk, akkor egy közbülső pont színének megadására is van módunk, míg a `scale_fill_gradientn()` tetszőlegesen sok szín megadását teszi lehetővé. A folytonos színpaletták használatára mutat példát a lenti négy ábra.

```
p1 <- ggplot(kerdoiv, aes(x=Wr.Hnd, fill=after_stat(count))) +
  geom_histogram(binwidth=1) +
  scale_fill_gradient(low = "yellow", high = "red")
p2 <- ggplot(kerdoiv, aes(x=Wr.Hnd, fill=after_stat(count))) +
```



9.5. ábra: Az RColorBrewer színpalettái

```
  geom_histogram(binwidth=1) +
  scale_fill_gradient2(low = "blue",mid = "white", high = "red",
                       midpoint=30)
p3 <- ggplot(kerdoiv, aes(x=Wr.Hnd, fill=after_stat(count))) +
  geom_histogram(binwidth=1) +
  scale_fill_gradientn(colours = rainbow(5))
p4 <- ggplot(kerdoiv, aes(x=Wr.Hnd, fill=after_stat(count))) +
  geom_histogram(binwidth=1) +
  scale_fill_gradient_tableau(palette = "Orange")
grid.arrange(p1, p2, p3, p4, ncol=2)
```



9.3.4.2. Színek közvetlen megadása

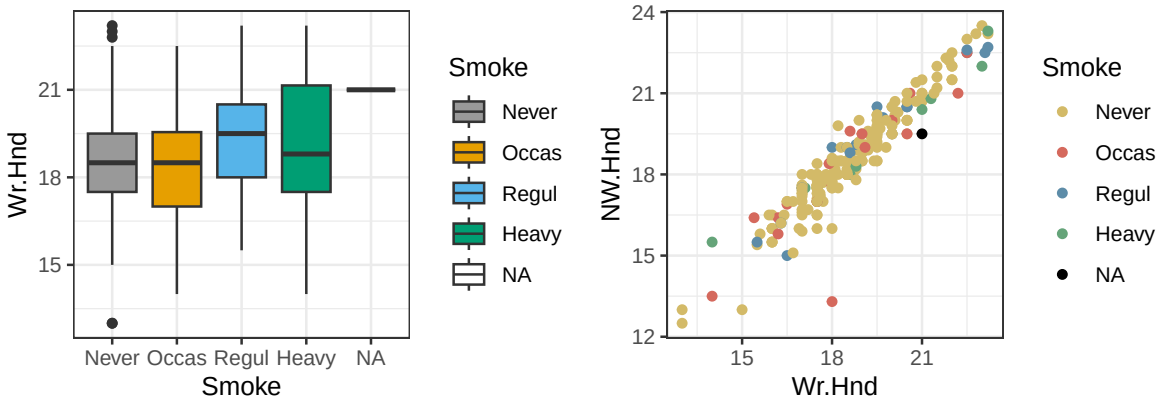
A kategorikus változók színeit közvetlenül is meghatározhatjuk, nem kell feltétlenül egy előre definiált színpaletta színeit használni úgy, ahogy azt az előző fejezetben láthattuk.

Szinek közvetlen beállítására a leképezésben használt paramétertől függően használhatjuk a következő függvényeket:

- `scale_fill_manual()` - dobozdiagram, oszlopdiagram stb. kitöltőszínének beállítására, amikor a `fill=` szerepel a leképezésben
- `scale_color_manual()` - vonalak vagy pontok színének beállítására, amikor a `colour=` szerepel a leképezésben.

Színek közvetlen beállítására mutat példát a következő két ábra. Láthatjuk, hogy a hiányzó érték jelzésére használt színt az `na.value=` argumentummal szabályozhatjuk. Ezt az argumentumot az előző fejezet összes `scale_fill_*()` és `scale_colour_*()` függvényében is használhattuk volna.

```
p1 <- ggplot(kerdoiv, aes(x=Smoke, y=Wr.Hnd, fill=Smoke)) +
  geom_boxplot() +
  scale_fill_manual(values=c("#999999","#E69F00",
                             "#56B4E9","#009E73"), na.value = "white")
p2 <- ggplot(kerdoiv, aes(x=Wr.Hnd, y=NW.Hnd, colour=Smoke)) +
  geom_point() +
  scale_color_manual(values=c("#D3BA68", "#D5695D",
                    "#5D8CA8", "#65A479"),na.value = "black")
grid.arrange(p1, p2, ncol=2)
```

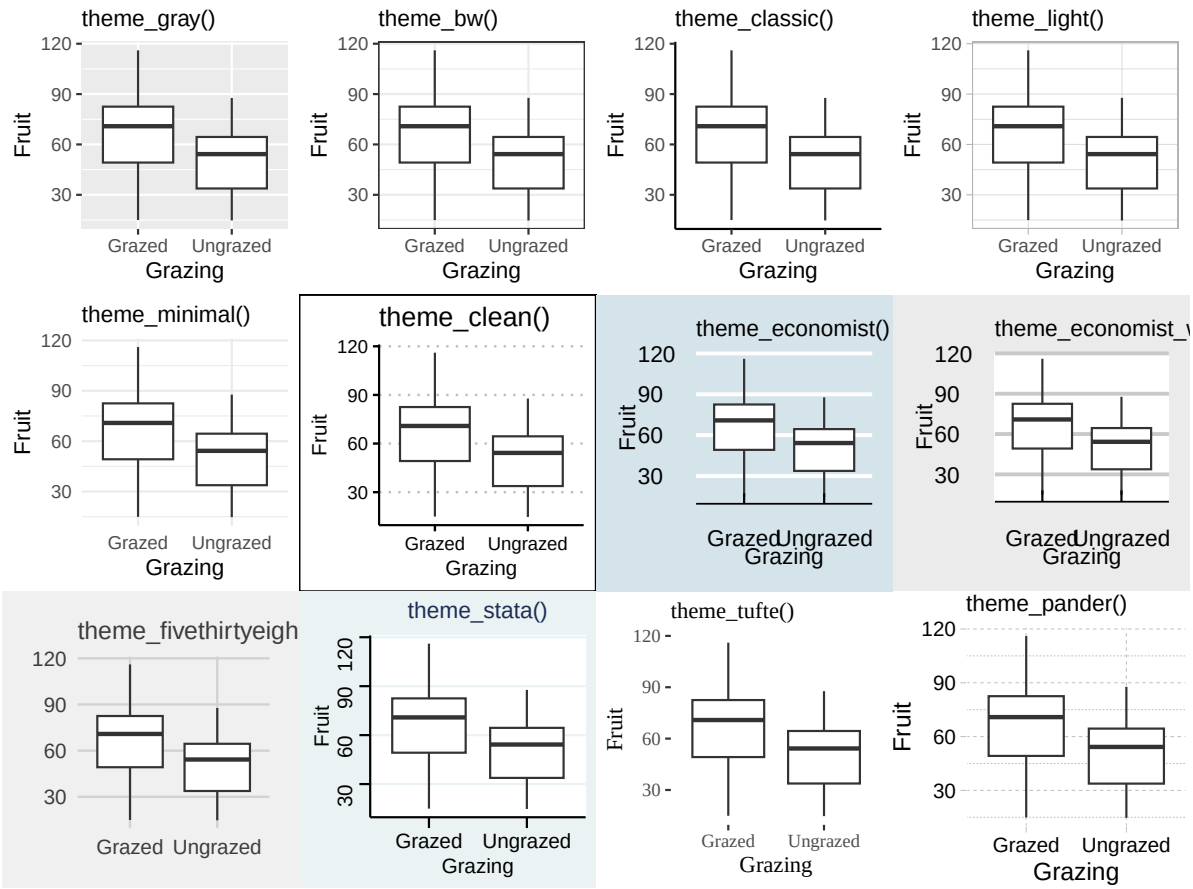


9.3.5. Témák

A ggplot2 rendszerhez számos ábrastílus készült, amelyeket a `theme_*()` függvényekkel adhatunk az ábránkhoz. Az egyes témák egyéni beállítások sorát tartalmazzák például az

ábra színezésére, betűtípusára és betűnagyságára vonatkozóan. A `{ggplot2}` csomag maga is több témát tartalmaz, de ezeket például a `{ggthemes}` csomag újabb ábrastílusokkal egészíti ki. Nézzünk példát néhány ábrastílusra. Megjegyezzük, hogy a 9.1.2.1. fejezet utolsó ábrájától kezdve az összes ábránk stílusa `theme_bw()` volt, és erről a beállításáról az `old.theme <- theme_set(theme_bw())` globálisan érvényes paranccsal gondoskodtunk.

```
p0  <- ggplot(data=alma, aes(x=Grazing, y=Fruit)) + geom_boxplot()
p1  <- p0 + theme_gray() + labs(subtitle = "theme_gray()")
p2  <- p0 + theme_bw() + labs(subtitle = "theme_bw()")
p3  <- p0 + theme_classic() + labs(subtitle = "theme_classic()")
p4  <- p0 + theme_light() + labs(subtitle = "theme_light()")
p5  <- p0 + theme_minimal() + labs(subtitle = "theme_minimal()")
p6  <- p0 + theme_clean() + labs(subtitle = "theme_clean()")
p7  <- p0 + theme_economist() + labs(subtitle = "theme_economist()")
p8  <- p0 + theme_economist_white() +
  labs(subtitle = "theme_economist_white()")
p9  <- p0 + theme_fivethirtyeight() +
  labs(subtitle = "theme_fivethirtyeight()")
p10 <- p0 + theme_stata() + labs(subtitle = "theme_stata()")
p11 <- p0 + theme_tufte() + labs(subtitle = "theme_tufte()")
p12 <- p0 + theme_pander() + labs(subtitle = "theme_pander()")
grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8,
             p9, p10, p11, p12, nrow=3, ncol=4)
```



9.3.6. Ábra annotálása

Az elkészült ábrát további információkkal is elláthatjuk a ggplot2 rendszer annotálási lehetőségeinek köszönhetően. Elsősorban szöveges információval szokás kiegészíteni az ábrát, de valójában bármilyen geom elemet ráhelyezhetünk a már elkészült képre.

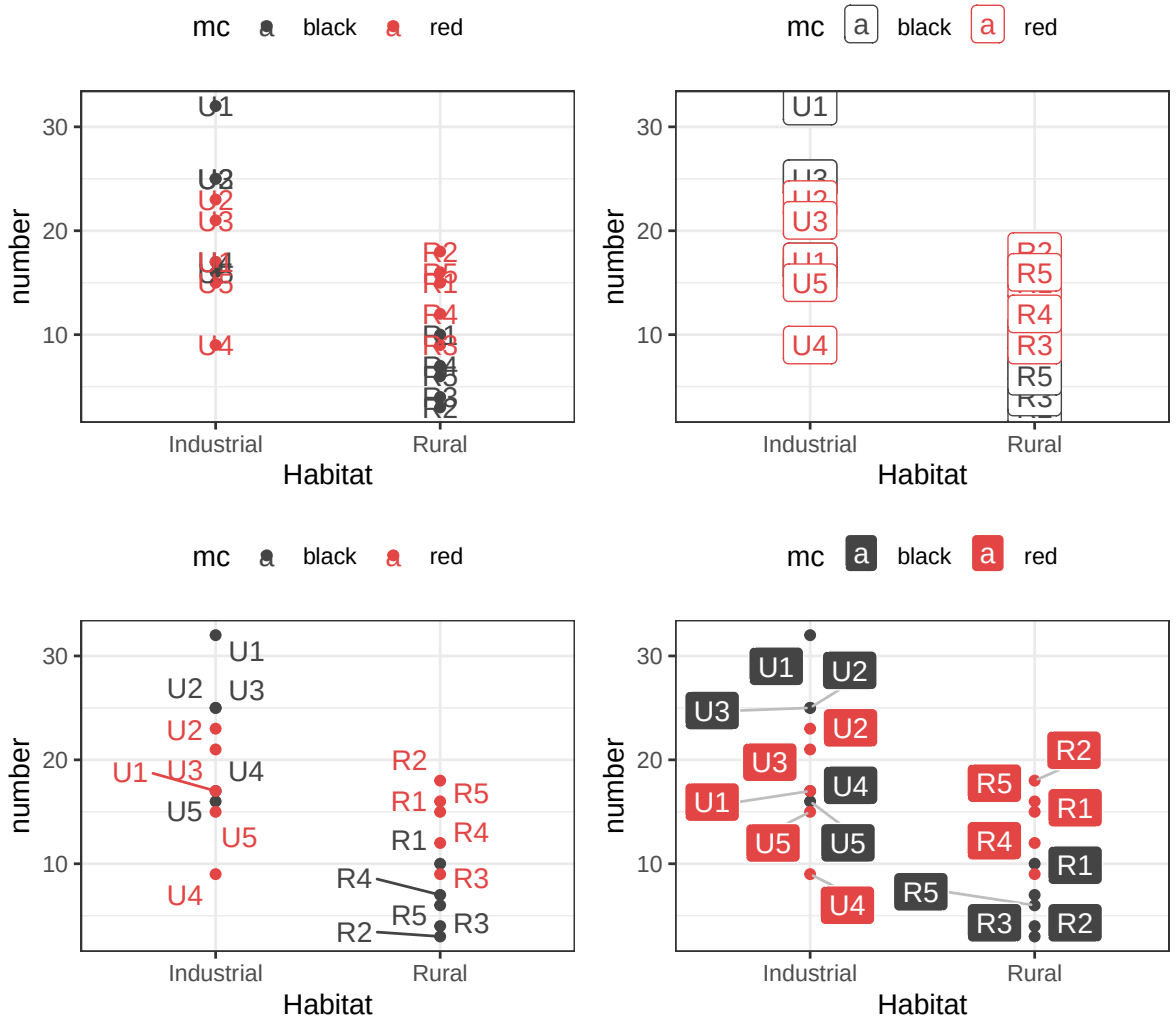
A részletes, mintaelemszintű szöveges annotálást a 9.11. táblázatban szereplő függvények támogatják. Ezek tipikusan valamely pontdiagramhoz köthetők, és a mintaelemeket reprezentáló pontok mellett tudnak szöveget megjeleníteni.

9.11. táblázat: A mintaelemek szöveges reprezentálása *(saját szerkesztés)*

Geom függvény	Leírás
`geom_text()`	Szöveg rajzolása
`geom_label()`	Szöveg rajzolása keretben
`geom_text_repel()`	Szöveg rajzolása (`{ggrepel}`)

Geom függvény	Leírás
`geom_label_repel()`	Szöveg rajzolása keretben (`{ggrepel}`)

```
# p0 - alapábra, önmagában nem jelenik meg
p0 <- ggplot(katica, aes(x = Habitat, y=number, colour=morph_colour))+
  geom_point() + theme(legend.position = "top") +
  scale_color_manual(values = c("black"="#444444", "red"="#e34444"),
                     name = "mc") +
  scale_fill_manual(values = c("black"="#444444", "red"="#e34444"),
                    name = "mc")
# p1 - feliratokkal
p1 <- p0 + geom_text(aes(label=Site))
# p2 - feliratok kerettel
p2 <- p0 + geom_label(aes(label=Site))
# p3 - feliratok, nem fedik egymást
p3 <- p0 + geom_text_repel(aes(label=Site))
# p4 - feliratok kerettel, nem fedik egymást
p4 <- p0 + geom_label_repel(aes(label=Site, fill=morph_colour),
                            colour="white", segment.colour = "grey")
grid.arrange(p1, p2, p3, p4, ncol=2)
```

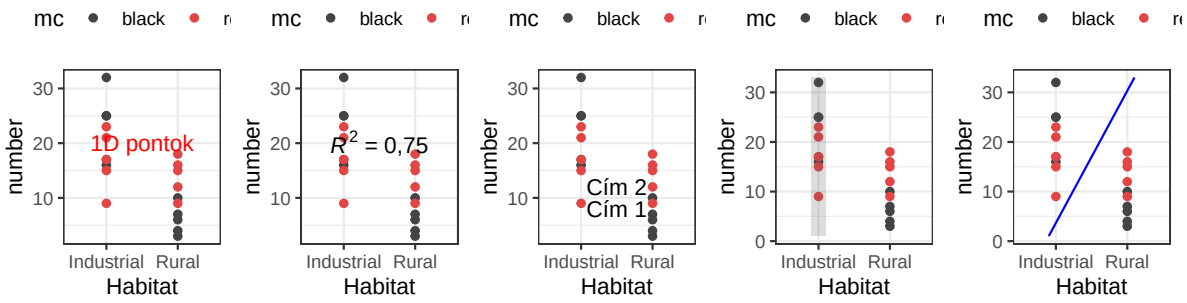


A fenti 4 kép a 9.11. táblázatban szereplő négy szövegrajzoló függvényre mutat példát. Látjuk, hogy a legjobb eredményt a `{ggrepel}` csomag függvényeivel érhetjük el.

Amennyiben nem mintaelem szinten szeretnénk szöveget megjeleníteni, akkor az `annotate(geom="text")` függvényt is használhatjuk. A `geom=` argumentumot azonban beállíthatjuk úgy, hogy téglalapot (`rect`) vagy egy egyenes szakaszt hozzunk létre (`segment`). Erre mutat példát a következő öt ábra.

```
# p1 - egyetlen szöveg az ábrára
p1 <- p0 + annotate(geom="text", x=1.5, y=20, label="1D pontok",
                    color="red")
# p2 - egyetlen szöveg az ábrára, képlettel
```

```
p2 <- p0 + annotate(geom="text", x=1.5, y=20,
                    label="paste(italic(R) ^ 2, \" = 0,75\")",
                    parse=TRUE)
# p3 - több szöveg az ábrára
p3 <- p0 + annotate("text", x=c(1.5, 1.5), y=c(8, 12),
                    label=c("Cím 1", "Cím 2"))
# p4 - téglalap az ábrára
p4 <- p0 + annotate("rect", xmin=0.9, xmax=1.1,
                    ymin=1, ymax=33, alpha=.2)
# p2 - vonal az ábrára
p5 <- p0 + annotate("segment", x=0.9, xend=2.1, y=1, yend=33,
                    colour = "blue")
grid.arrange(p1, p2, p3, p4, p5, ncol=5)
```

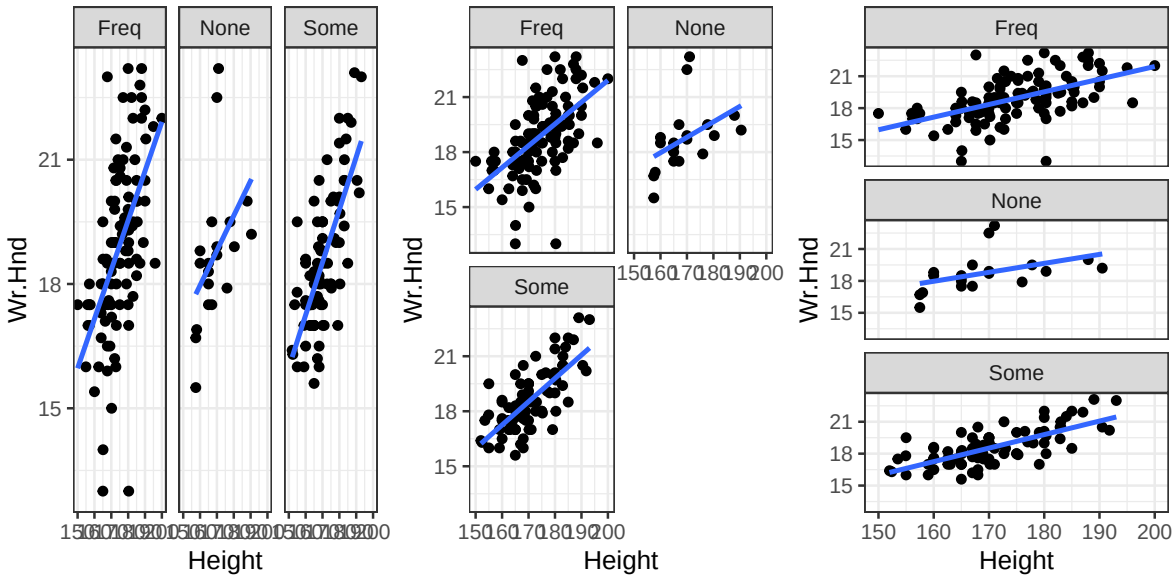


9.3.7. Rácsozás

A rácsozás (faceting) egy igen hatékony ábrázolási technika ggplot2 rendszerben, amely egy vagy több kategorikus változó segítségével több ábra egyidejű létrehozását teszi lehetővé a faktor változó(k) egyes csoportjai mentén.

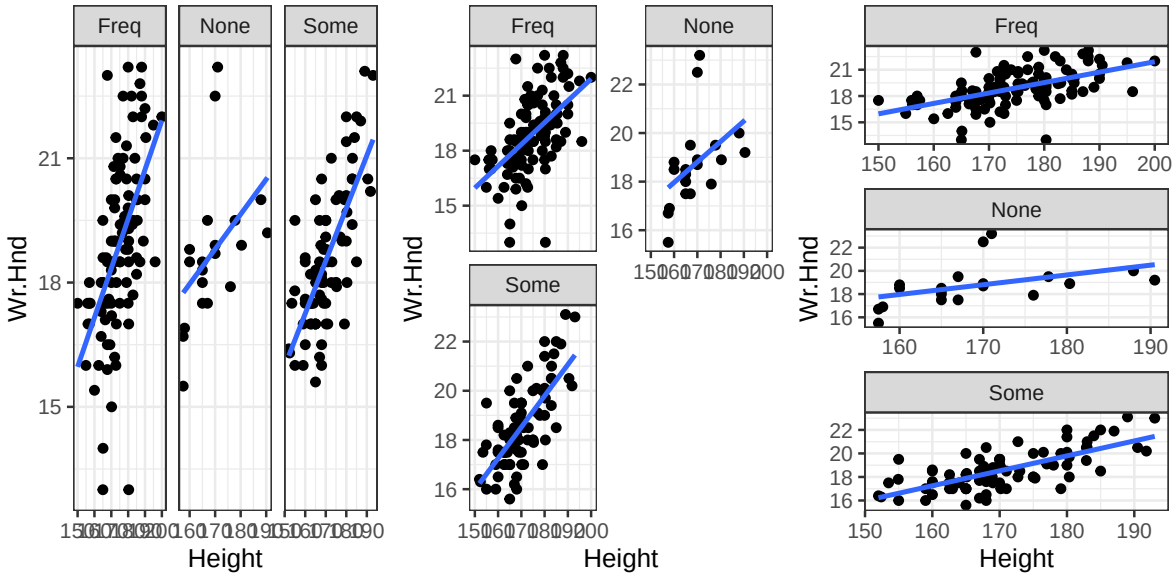
Korábban már vizsgáltuk a testmagasság és a kézméret kapcsolatát (9.1. példa), most a testmozgás (`Exer`) három szintű faktor segítségével három ábrára bontjuk az alap kétdimenziós pontdiagramot. Minden szinthez tartozni fog egy-egy kétdimenziós pontdiagram, mely csak az adott testmozgású személyek testmagasságát és kézméretét tartalmazza. A `facet_wrap(.~Exer)` függvény ad erre lehetőséget, sőt az `nrow=` és `ncol=` argumentumok beállításával szabályozhatjuk, hogy az ábrák a rácsszerű elrendezésben hány sorba, és hány oszlopba rendezetten jelenjenek meg. Lássunk példát három-, két- és egyoszlopos megjelenésre.

```
p0 <- ggplot(kerdoiv, aes(x=Height, y=Wr.Hnd)) + geom_point() +
  geom_smooth(method = "lm", se=F)
p1 <- p0 + facet_wrap(. ~ Exer, ncol = 3) # 1 sor, 3 oszlop
p2 <- p0 + facet_wrap(. ~ Exer, ncol = 2) # 2 sor, 2 oszlop
p3 <- p0 + facet_wrap(. ~ Exer, ncol = 1) # 3 sor, 1 oszlop
grid.arrange(p1, p2, p3, nrow=1)
```



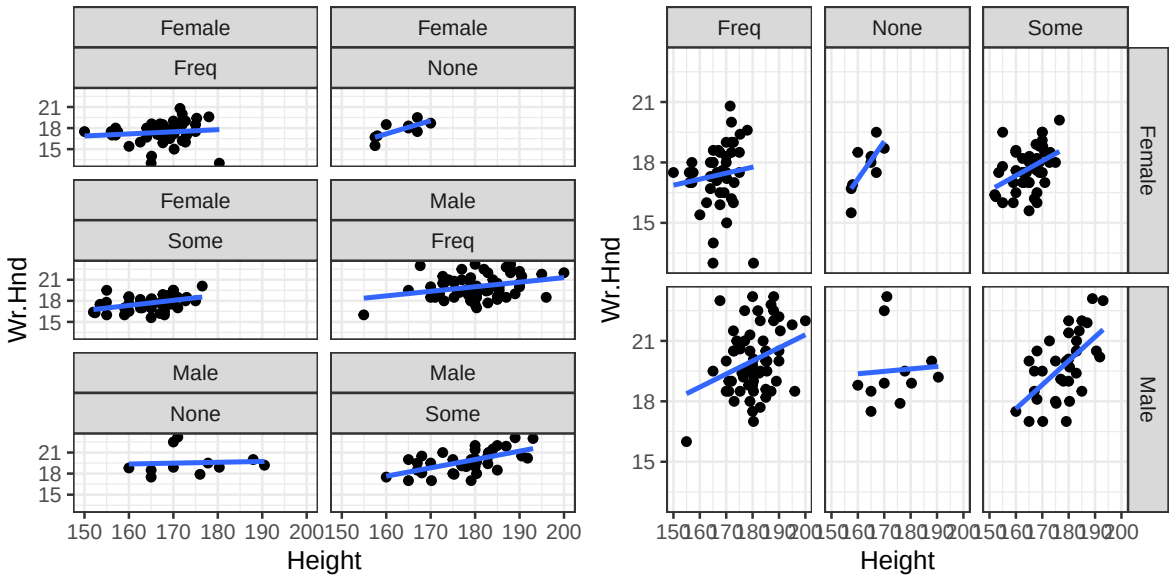
A rácsozás egyik előnye, hogy alapértelmezés szerint az egyes csoportokban (a kis képeken) a tengelyek értéktartományai azonosak, így az eredmények összehasonlítása nagyon egyszerű. Nagyon ritkán írjük felül ezt a működést, ha mégis szükség van rá, akkor átírhatjuk az x tengelyt, az y tengelyt vagy mindkettőt "szabadra" úgy, hogy a `facet_wrap()` függvény `scales=` argumentumában a `"free_x"`, `"free_y"` vagy `"free"` értéket adjuk meg. Erre látunk példát a lenti ábrán.

```
p0 <- ggplot(kerdoiv, aes(x=Height, y=Wr.Hnd)) + geom_point() +
  geom_smooth(method = "lm", se=F)
p1 <- p0 + facet_wrap(. ~ Exer, ncol = 3, scale="free_x") # x szabad
p2 <- p0 + facet_wrap(. ~ Exer, ncol = 2, scale="free_y") # y szabad
p3 <- p0 + facet_wrap(. ~ Exer, ncol = 1, scale="free") # x, y szabad
grid.arrange(p1, p2, p3, nrow=1)
```



Két vagy akár több faktor segítségével tovább finomíthatjuk a rácsozott megjelenítést, de a két kategorikus változó esetén a `facet_grid()` függvénnyel létrehozott rácsozott ábra adja a legjobban értelmezhető megjelenést. Most mindkét lehetőséget kipróbáljuk a nem (`Sex`) és testmozgás (`Exer`) faktorok bevonásával. Mivel az egyik változó két, a másik három szintű, így a rácsozott ábrán hat kis kép megjelenését várjuk.

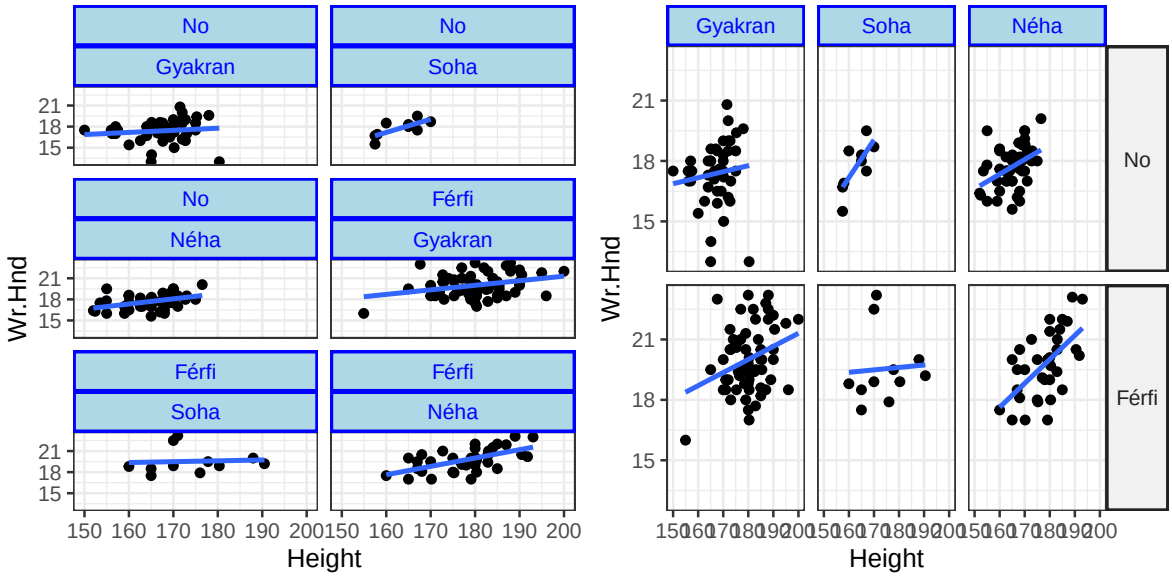
```
p0 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=Height, y=Wr.Hnd)) +
  geom_point() + geom_smooth(method = "lm", se=F)
p1 <- p0 + facet_wrap(Sex ~ Exer, ncol = 2)
p2 <- p0 + facet_grid(Sex ~ Exer)
grid.arrange(p1, p2, nrow=1)
```



A rácsozott ábrák finomabb beállítására is lehetőségünk van. A paneleken megjelenő szöveg a `labeller=` paraméterrel írható felül, mind a `facet_wrap()`, mind a `facet_grid()` függvényben. A paraméter értéke a `labeller()` függvénnyel összeállított objektum, melynek paraméterezésében a régi szintneveket újra cserélhetjük a lenti példában látható módon.

```
p0 <- kerdoiv |> drop_na(Sex) |> ggplot(aes(x=Height, y=Wr.Hnd)) +
  geom_point() + geom_smooth(method = "lm", se=F)
p1 <- p0 + facet_wrap(Sex ~ Exer, ncol = 2,
                      labeller = labeller(Sex=c("Female"="Nő",
                                                "Male"="Férfi"),
                                          Exer=c("Freq"="Gyakran",
                                                 "None"="Soha",
                                                 "Some"="Néha"))) +
  theme(
    strip.background = element_rect(fill = "lightblue",
                                    color = "blue", size = 1),
    strip.text = element_text(colour = "blue")
)
p2 <- p0 + facet_grid(Sex ~ Exer,
                      labeller = labeller(Sex=c("Female"="Nő",
                                                "Male"="Férfi"),
                                          Exer=c("Freq"="Gyakran",
                                                 "None"="Soha",
                                                 "Some"="Néha"))) +
```

```
  theme(
    strip.background.x = element_rect(fill = "lightblue",
                                      color = "blue", size = 1),
    strip.text.x = element_text(colour = "blue"),
    strip.background.y = element_rect(fill = "#f1f1f1",
                                      color = "#212121", size = 1),
    strip.text.y = element_text(colour = "#212121", angle = 0)
)
grid.arrange(p1, p2, nrow=1)
```



A panelek formázására is van lehetőségünk, a `facet_wrap()` esetében a `strip.background=` a téglalap formátumát, a `strip.text=` a szöveg stílusát határozza meg, míg a `facet_grid()` esetén külön az x és y tengelyen megjelenő panelek állítására is van módunk. Ezek a paramétereket a `theme()` függvényben kell elhelyeznünk.

Összefoglalás

A ggplot2 ábráink szinte azonnal publikációkészek, de kisebb nagyobb módosításokra még így is szükség lehet. A `labs(title= , x= , y=, fill =, colour = )` módosító ábracímet, tengelyfeliratokat és jelmagyarázatbeli címkéket is be tud állítani, míg a `theme(legend.position=)` magának a jelmagyarázatnak a helyét határozza meg. A tengelyek láthatósági tartományát a `coord_cartesian(xlim=,`

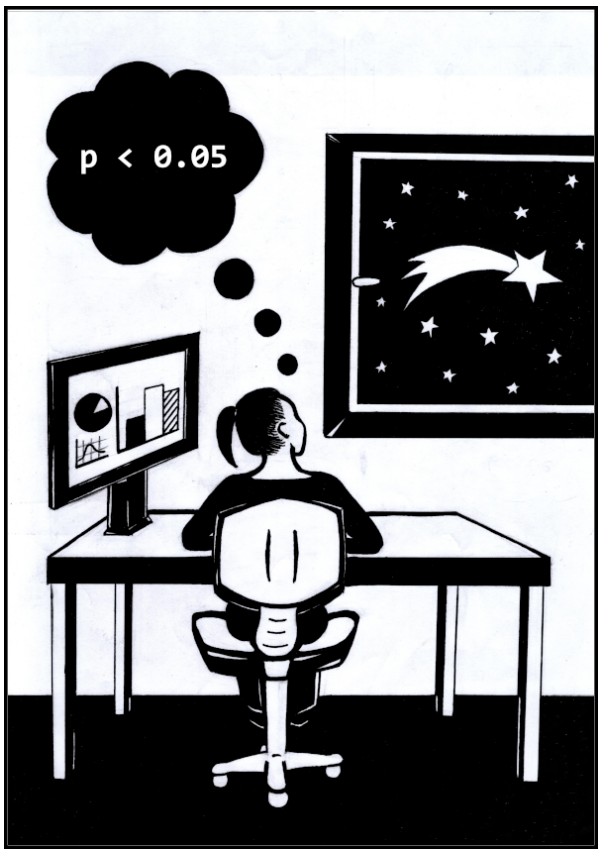
`ylim=)`, osztásközeit a `scale_[x|y]_[discrete|continuous](name=, breaks=, limits=, labels=)` függvénnyel szabályozhatjuk. A faktor változók értékei és az egyes színkódok közötti alapértelmezett összerendelést legtöbbször a `scale_[colour|fill]_grey()` és `scale_[colour|fill]_brewer()`, míg numerikus változó esetén a `scale_[colour|fill]_gradient(low=, high=)` függvénnyel bíráljuk felül. A számos előre definiált ábrastílus közül a `theme_*()` módosítóval választhatunk. Rácsozott ábrát tipikusan a `facet_wrap(. ~ <faktor>)` vagy `facet_grid(<faktor1> ~ <faktor2>)` módosítókkal hozunk létre.

🎯 Feladatok

1. A 9.4. ábrán már használtuk az `{RColorBrewer}` csomag színpalettáját kategorikus változó esetén. Próbáljuk ki magunk is a 9.5. ábrán felsorolt palettaneveket “Set3”-tól “Accent”-ig!
2. A standard normális eloszlás és az 5, 10, 15 és 20 szabadsági fokú t-eloszás sűrűségfüggvényét jelenítsük meg egy-egy ábrán! A görbék megjelenítéséhez használjuk a `stat_function()` függvényt!
3. Mutassuk be az R hagyományos grafikus rendszerét!

IV. rész
Következtetések

10. Hipotézisvizsgálatok



A statisztikai elemzések során a kutató a legtöbb figyelmet a különböző hipotézisek tesztelésére és a p értékeken alapuló döntések meghozatalára fordítja. Ez a teljes kutatási folyamat talán legizgalmasabb része, hiszen a kiinduló hipotéziseinkről hozzunk döntést: elfogadjuk vagy elvetjük őket.

A statisztikailag szignifikáns hatások feltárásának a hipotézisvizsgálat fontos eszköze, azonban ismert, hogy a p értéket befolyásolja a minta mérete. Az alacsony p érték nem feltétlenül utal nagy hatásra vagy a gyakorlatban jelentős hatásra. Ne feledjük, hogy a szignifikáns eredmények nem feltétlenül relevánsak számunkra, tehát a statisztikai szignifikancia és a szakmai szignifikancia elválik egymástól. Az utóbbi időben a statisztikai elemzésekben megnőtt a leíró statisztika, a mutatók, táblázatok és ábrák, valamint a konfidencia intervallumok és a hatásmértékek meghatározásának szerepe. Ezért a legalapvetőbb egy- és kétdimenziós hipotézisvizsgálatok bemutatása és a p értékek mellett, a hatásmértékre és a konfidencia intervallumokra is hangsúlyt helyezünk.

A hipotézisvizsgálatok eredményének értelmezéséhez statisztikai alapismeretek szükségesek, terjedelmi okok miatt ezekre csak utalunk. Az elméleti háttér iránt érdeklődő olvasó számos nagyszerű szakkönyvből tájékozódhat, ilyen például Brace és mtsai. (2016), Vargha (2000), Reiczigel és mtsai. (2007), Mangiafico (2016), Dalgaard (2008) és Rasch és mtsai. (2011).

Ebben a fejezetben a statisztika azon klasszikus próbáit foglaltuk össze, amelyek jellemzően egy- vagy kétmintás hipotézisvizsgálatokat jelentenek. Az öt alfejezet a nullhipotézisben szereplő állításoknak és paramétereknek megfelelően a statisztikai próbák különböző csoportjait fedi le:

- paraméteres próbák (10.2. fejezet),
- nemparaméteres próbák (10.3. fejezet),
- normalitás vizsgálatára vonatkozó próbák (10.4. fejezet),
- varianciára vonatkozó próbák (10.5. fejezet),
- valószínűségre vonatkozó próbák (10.6. fejezet).

A fejezet további témakörei:

- hatásméret számítása az egyes próbák esetén(10.7. fejezet),
- statisztikai erő és mintanagyság számítása (10.8. fejezet),
- a próbák alternatívái, a `{jmv}` csomag lehetőségei (10.9. fejezet).

Megjegyezzük, hogy az itt ismertetett statisztikai következtetéseket a Neyman–Pearson-féle megközelítésre alapozzuk, a másik két alternatívát, a bayesiánus és a likelihood megközelítést nem vagy alig érintjük (Dienes, 2016). A próbák bemutatása során arra fókuszálunk, hogyan hajtjuk végre az adott eljárást a R-ben, ezért fiktív miniadatbázisokkal dolgozunk és az elméleti hátteret teljesen nélkülözzük.

10.1. Az adatelemző munka 🙂

Mielőtt belevágunk az egyes statisztikai próbák leltárszerű felsorolásába oldjunk meg egy konkrét példát, amely megmutatja számunkra, hogy az eddig tanult beolvasó, adatelőkészítő, leíró statisztikai és grafikai funkciók hogyan simulnak bele ebbe a nagy fejezetbe, ahol hipotézisvizsgálat végrehajtásához, hatásvizsgálathoz és statisztikai erő számításához adunk muníciót.

Egy adott statisztikai próba elvégzése előtt tisztában kell lennünk a próbával kapcsolatos alapvető ismeretekkel, mikor, milyen feltételek mellett, és persze hogyan végezhető el. Jelen fejezet csupán az utolsó pontban nyújt támogatást, a próbák elméleti hátterét és alkalmazási feltételeit nem tárgyaljuk részletesen. Nézzük meg azonban, hogy a klasszikus kétmintás t-próba esetén milyen előzetes tudásra kell építenünk.

Hipotézisek kétmintás t-próba esetén:

- *Nullhipotézis*: A populációbeli változó várható értékei, amelyekből az adatokat mintavételezték, mindkét csoportban egyenlők ($H_0 \ : \ \mu_1 = \mu_2$).
- *Alternatív hipotézis*: A populációbeli változó várható értékei, amelyekből az adatokat mintavételezték, a két csoportban nem egyenlők ($H_1 \ : \ \mu_1 \neq \mu_2$).

Alkalmazási feltételek kétmintás t-próba esetén:

- Kétmintás adatok. Azaz egy mérési változó két csoportban, két populációban.
- A függő változó intervallum/arány, és folytonos.
- A független változó kétszintű faktor, vagyis két csoportunk van.
- Az egyes populációk adatai normál eloszlásúak.
- Student-féle t-próbához a két mintának azonos szórással kell rendelkeznie. A Welch-féle t-próba, amelyet alapértelmezés szerint használ az R, nem feltételez egyenlő szórást.
- A csoportok közötti megfigyelések függetlenek. Vagyis nem párosított vagy ismételt mérési adatok.

A fenti ismeretek birtokában már belevághatunk egy konkrét statisztika adatelemzésbe, ami mint látni fogjuk épp kétmintás t-próba végrehajtásához vezet.

Példa 10.1 (Tanulás zajban)**.** Huszonnégy ember részt vett egy kísérletben, amelyben annak megállapítását tűzték ki célul, hogy a háttérzaj (zene, ajtócsapódás, kávékészítés zaja stb.) hogyan befolyásolja a rövid távú memóriát (szavak visszahívását). A résztvevők fele (NOISE csoport) megpróbált memorizálni 2 perc alatt egy 20 szavas listát, miközben fülhallgatón keresztül az előre felvett zaj szólt. A többi résztvevő is viselt fülhallgatót (NO NOISE csoport), de ők nem hallottak zajt a szavak memorizálása közben. Közvetlenül ezután megállapították, hogy hány szóra emlékeztek vissza. Vizsgáljuk meg, hogy van-e

eltérés a visszahívott szavak számában a két kondícióban, vagyis a zajnak van-e hatása a rövidtávú memóriára!
Forrás: Dancey és Reidy (2011) alapján

Tudjuk, hogy az adatelemzés 4 alapvető lépésre bontható:

1. Adatok beolvasása, amelyről a 6. fejezetben olvashatunk részletesen.
2. Adatok előkészítése elemzésre, amelyről 7. fejezet számol be részletesen, de természetesen épít az R nyelvet bemutató 5. nagy fejezetre is.
3. Adatok elemzése, amely magába foglalja a leíró statisztikai elemzéseket (a mutatókról és a táblázatokról a 8. fejezetben, az ábrák készítéséről a 9. fejezetben olvashatunk) és a jelen fejezetben bemutatott statisztikai próbákat.
4. Eredmények publikációja, amelyet a következő, 11. fejezetben részletezünk, és nagyon röviden az 1-3. pontokban létrehozott R parancsok QMD állományba illesztését jelenti.

Tekintsük át, hogy a fenti lépések, hogyan teszik lehetővé a fenti, 10.1. példa megoldását.

Végezzük el az adatok beolvasását, ellenőrzését és átalakítását! Az adatbázist a jobb áttekinthetőség kedvéért inline módon olvassuk be (6.2. fejezet). Az adatbázis tartalmát a `head()` függvénnyel ellenőrizzük, amely az első néhány sort jeleníti meg. Ez a lehetőség már az adatelőkészítő lépésen belül az információ gyűjtésére szolgál és az 7.1.1. és 7.2.1. fejezetekben tértünk ki rá.

```
# adatok beolvasása
d <- read.table(file = textConnection("
    NOISE   NO.NOISE
     5.00      15.00
    10.00       9.00
     6.00      16.00
     6.00      15.00
     7.00      16.00
     3.00      18.00
     6.00      17.00
     9.00      13.00
     5.00      11.00
    10.00      12.00
    11.00      13.00
    9.00       11.00
"), header=T, sep="")
# az adatbázis tartalma
head(d) # az első 6 sor
```

```
#>   NOISE NO.NOISE
#> 1     5       15
#> 2    10        9
#> 3     6       16
#> 4     6       15
#> 5     7       16
#> 6     3       18
```

Adatbázisunk kényelmi okok miatt széles formátumban van, amelyet a `{tidyr}` csomag `pivot_longer()` függvényével át kell alakítanunk hosszú formátumra (lásd 7.3.2. fejezet).

```
# széles-hosszú átalakítás
library(tidyverse)
d <- d |>
    pivot_longer(cols = everything(), names_to = "csoport", values_to = "szavak")
head(d)  # az első 6 sor
#> # A tibble: 6 x 2
#>   csoport  szavak
#>   <chr>     <dbl>
#> 1 NOISE         5
#> 2 NO.NOISE     15
#> 3 NOISE        10
#> 4 NO.NOISE      9
#> 5 NOISE         6
#> 6 NO.NOISE     16
```

Utolsó adatelőkészítő parancsunk a karakteres `csoport` oszlop faktorrá konvertálása lesz, és ezt követően az `str()` függvénnyel ellenőrizzük az adatbázis végső szerkezetét (7. fejezet).

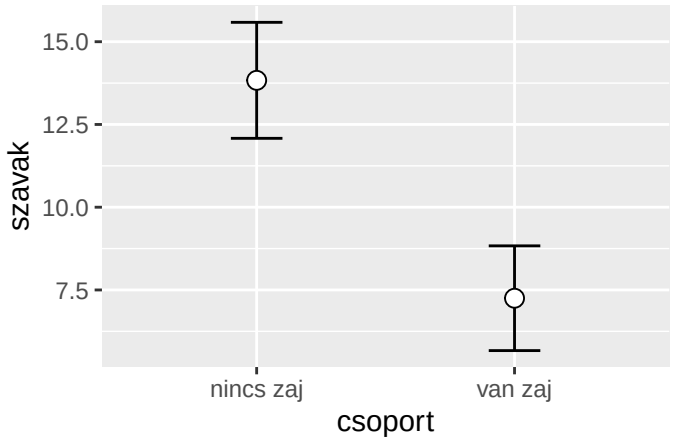
```
# faktorrá alakítás, és a szintek átnevezése
d <- d |>
  mutate(csoport = factor(csoport,
                          levels = c("NO.NOISE", "NOISE"),
                          labels = c("nincs zaj", "van zaj")))


str(d) # az adatbázis szerkezete
#> tibble [24 x 2] (S3: tbl_df/tbl/data.frame)
#>  $ csoport: Factor w/ 2 levels "nincs zaj","van zaj": 2 1 2 1 2 1 2 1 2 1 ...
#>  $ szavak : num [1:24] 5 15 10 9 6 16 6 15 7 16 ...
```

Miután mindent rendben találtunk az adatmátrixszal kapcsolatban tovább léphetünk a leíró statisztikai elemzésre, amelyet most a `psych` csomag `describeBy()` függvényével hajtunk végre (8.3.1. fejezet).

```
# leíró statisztikai mutatók
psych::describeBy(d$szavak, d$csoport, mat = TRUE, digits = 2, fast = T)
#>     item    group1 vars  n  mean   sd median min max range  skew
#> X11    1 nincs zaj    1 12 13.83 2.76   14.0   9  18     9 -0.15
#> X12    2   van zaj    1 12  7.25 2.49    6.5   3  11     8  0.00
#>     kurtosis   se
#> X11    -1.36 0.80
#> X12    -1.45 0.72
```

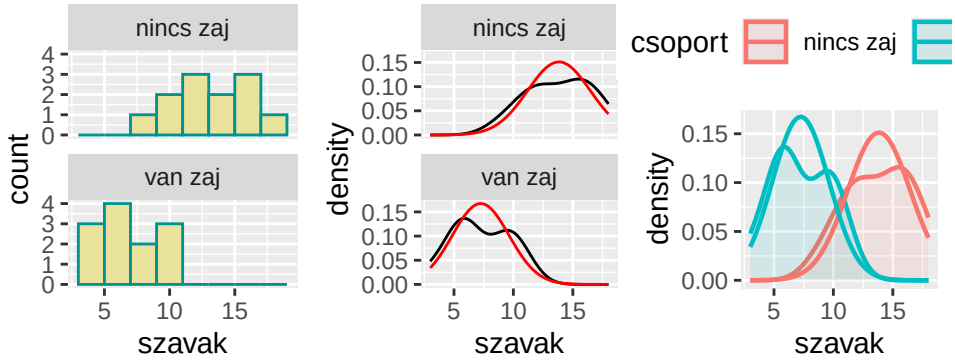
Miután betekintést nyertünk az adatainkba, azaz megtudtuk például, hogy mindkét csoportban 12-en vannak, és a zaj nélküli tanulás után majdnem dupla annyi szót tudnak átlagosan visszaidézni, érdemes ábrával is ellenőrizni az adataink eloszlását. Az átlagokat és a 95%-os konfidencia intervallumokat jelenítjük meg (9.2. fejezet).

```
library(ggplot2)
ggplot(d, aes(x=csoport, y=szavak)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  stat_summary(fun=mean, geom="point", size=3, shape=21, fill="white")
```



A leíró ábráink nem nélkülözhetik a hisztogramok megjelenítését, de most emellett simított hisztogramot és a normális eloszlás görbéjét is berajzoljuk, a két csoport eloszlásának vizsgálatához (9.1.4. fejezet).

```
library(ggplot2)
library(ggh4x)
p1 <- ggplot(d, aes(x=szavak)) +
  geom_histogram(binwidth = 2,
                 colour="#009392FF", fill="#E9E29CFF") +
  facet_wrap(~csoport, ncol=1)
p2 <- ggplot(d, aes(x=szavak))  + geom_density() +
  stat_theodensity(colour="red") + facet_wrap(~csoport, ncol=1)
p3 <- ggplot(d, aes(x=szavak, fill=csoport, colour=csoport)) +
  geom_density(alpha=0.1, linewidth=0.8) +
  stat_theodensity(linewidth=0.8) + theme(legend.position = "top")
gridExtra::grid.arrange(p1, p2, p3, ncol=3)
```



A statisztikai hipotézisvizsgálat felé lépve megállapíthatjuk, hogy a 10.1. példa megoldásához kétmintás t-próba lenne a legmegfelelőbb eljárás. Ellenőrizzük a normalitásra és szóráshomogenitásra vonatkozó feltételeket (ezeket az eljárásokat itt ebben a fejezetben ismerjük meg, konkrétan a 10.4. és 10.5. alfejezetekben). A normalitást a Shapiro–Wilk-próba segítségével ellenőrizzük.

```
# Shapiro-Wilk próba két csoportra
library(onewaytests)
nor.test(formula = szavak ~ csoport, data = d, method = "SW",
         plot = NULL, alpha = 0.05)
#>
#>   Shapiro-Wilk Normality Test (alpha = 0.05)
#> -------------------------------------------------
#>   data : szavak and csoport
#>
```

```
#>       Level Statistic   p.value   Normality
#> 1 nincs zaj 0.9629863 0.8254603  Not reject
#> 2   van zaj 0.9375536 0.4670452  Not reject
#> ------------------------------------------------
```

A homogenitásvizsgálatot Levene-próbával ellenőrizzük.

```
# Levene-próba
DescTools::LeveneTest(formula = szavak ~ csoport, data = d, center = mean)
#> Levene's Test for Homogeneity of Variance (center = mean)
#>       Df F value Pr(>F)
#> group  1  0.1768 0.6782
#>       22
```

Mivel mindkét alkalmazási feltétel teljesült, bátran elvégezhetjük a kétmintás t-próbát (10.2.2. alfejezet)

```
# kétmintás t-próba
t.test(formula = szavak ~ csoport, data = d, var.equal = T,
       conf.level = 0.95)
#>
#>  Two Sample t-test
#>
#> data:  szavak by csoport
#> t = 6, df = 22, p-value = 4e-06
#> alternative hypothesis: true difference in means between group
#>   nincs zaj and group van zaj is not equal to 0
#> 95 percent confidence interval:
#>  4.4 8.8
#> sample estimates:
#> mean in group nincs zaj   mean in group van zaj
#>                    13.8                    7.2
```

A próba szignifikáns ($p < 0{,}001$), vagyis valós különbség van a két csoport között, más szóval a zajos tanulásnak szignifikáns, negatív hatása van a rövidtávú memóriára. Érdemes kiszámítani ennek a hatásnak a nagyságát is (10.7. alfejezet).

```
# hatásméret számítása és értelmezése
effectsize::cohens_d(x = szavak ~ csoport, data = d, pooled_sd = T, ci = 0.95)
#> Cohen's d |       95% CI
#> -------------------------
#> 2.51      | [1.40, 3.58]
#>
#> - Estimated using pooled SD.
effectsize::interpret_cohens_d(d = 2.51, rules = "cohen1988")
#> [1] "large"
#> (Rules: cohen1988)
```

Látható, hogy ez a hatás ($d = 2{,}51$) nagynak mondható. A statisztikai erővel és mintanagysággal kapcsolatos számításokat az adatgyűjtés előtt szokták elvégezni, de a teljesség kedvéért itt is bemutatjuk (10.8. alfejezet).

```
# statisztikai erő számítása
library(pwr)
pwr.t.test(d = 2.51, sig.level = 0.05, n = 12,
           type = "two.sample", alternative = "two.sided")
#>
#>      Two-sample t test power calculation
#>
#>               n = 12
#>               d = 2.51
#>       sig.level = 0.05
#>           power = 0.9999526
#>     alternative = two.sided
#>
#> NOTE: n is number in *each* group
```

Látható, hogy ekkora hatásmérték és mintaelemszám mellett a statisztikai erő 0,99, ami azt jelenti, hogy a próba 99%-os valószínűséggel képes észlelni a hatást. Arra a kérdésre is választ kaphatunk, hogy mondjuk 95%-os erő mellett, mekkora mintanagyságra lenne szükségünk egy létező, 2,51-es hatás kimutatásához.

```
# mintanagyság számítása
pwr.t.test(d = 2.51, sig.level = 0.05, power = 0.95,
           type = "two.sample", alternative = "two.sided")
#>
```

```
#>      Two-sample t test power calculation
#>
#>               n = 5.311994
#>               d = 2.51
#>       sig.level = 0.05
#>           power = 0.95
#>     alternative = two.sided
#>
#> NOTE: n is number in *each* group
```

Látható, hogy egy 12 elemű (6-6 fős) minta már elegendő lenne.

A fenti elemzéseket szövegesen úgy foglalhatnánk össze, hogy a kétmintás t-próba eredménye azt mutatja, a két csoport között szignifikáns különbség van a visszahívott szavak számában ($t(22) = 6{,}14; p < 0{,}001; d = 2{,}51$). A zajos környezetben tanuló diákok átlagosan 7,25 szót tudtak visszaidézni, míg a zajmentes környezetben tanuló diákok átlagosan 13,83 szót ($M_{vanzaj} = 7{,}25; SD_{vanzaj} = 2{,}49; M_{nincszaj} = 13{,}83; SD_{nincszaj} = 2{,}76$).

A fejezet további részére úgy tekinthetünk, hogy a fenti adatelemzési munkát egyfajta sablonnak tekintve, hogyan tudunk a kétmintás t-próba helyett egy másik, a konkrét adatelemzési tevékenységünkhöz igazodó próbát végrehajtani, a kapcsolódó hatásméretre és statisztikai erőre vonatkozó számításokkal együtt. Ha mindezeket a parancsokat, a beolvasó, adatelőkészítő és leíró statisztikai R sorokkal együtt bemásoljuk a 11. fejezetben részletesen bemutatott *Quarto* dokumentumba, akkor máris teljesítettük e könyv fő célkitűzését, publikációkész adatelemzést hajtottunk végre.

10.2. Paraméteres próbák 🙂

> **i** Miről lesz szó? Ebben a fejezetben
>
> - áttekintjük a t-próba különböző formáit,
> - az egyszempontos független és összetartozó mintás varianciaelemzést,
> - és a korreláció- és regressziószámítás egyszerű eseteit.

A paraméteres próbák a leggyakoribb és legismertebb statisztikai próbák. Ezek közé tartoznak például a t-próba egyes változatai, a varianciaelemzés különböző esetei és a korreláció- és regressziószámítás egyszerű formája.

10.2.1. Egymintás t-próba

Egymintás teszteket nem használunk túl gyakran, de hasznosak, ha egy adatsort egy adott értékkel szeretnénk összehasonlítani. Például megkérdezhetjük, hogy a tanulói pontszámok várható értéke jelentősen eltér-e az „alapértelmezett” vagy „semleges” 10-es pontszámtól.

Tekintsük a következő, pontszámokat tartalmazó adatmátrixot.

```
adat <- "  pontszam
                12
                 8
                 7
                11
                 8
                12  "

# adatmátrix beolvasása
df01 <- read.table(textConnection(adat), header=T, sep="")
```

```
# adatmátrix szerkezete
str(df01)
#> 'data.frame':    6 obs. of  1 variable:
#>  $ pontszam: int  12 8 7 11 8 12
```

Egymintás t-próbát a `t.test()` függvénnyel végezhetünk el, amely a konfidencia intervallumot is meghatározza.

```
# --= egymintás t-próba =--
t.test(x = df01$pontszam, mu = 10, conf.level = 0.95)
#>
#>  One Sample t-test
#>
#> data:  df01$pontszam
#> t = -0.36274, df = 5, p-value = 0.7316
#> alternative hypothesis: true mean is not equal to 10
#> 95 percent confidence interval:
#>   7.304465 12.028868
#> sample estimates:
#> mean of x
#>  9.666667
```

A fenti output tartalmazza a próbastatisztika értékét (`t=`), a szabadsági fokok számát (`df=`) és a p értéket (`p-value=`). Az `alternative hypothesis:` sor megfogalmazza az ellenhipotézist: a várható érték nem egyenlő 10-zel. A következő két sor a várható értékre vonatkozó 95%-os megbízhatóságú konfidencia intervallum határa ($95\%CI$: [7,30; 12,03]), az utolsó sorban pedig a mintaátlag szerepel (M = 9,67). A nullhipotézist megtartjuk ($t(5) = -0{,}363$; p = 0,732), azaz a minta várható értéke nem tér el a 10-es alapértéktől.

10.2.2. Kétmintás t-próba

A kétmintás (független vagy párosítatlan) t-próba egy általánosan használt teszt, amely a két minta mögötti változók várható értékét hasonlítja össze. Most két tanítási módszer (A és B) hatékonyságát vizsgáljuk. Mindkét módszerrel 4-4 diák tanult, és a hatékonyságot egy dolgozat pontszámával mérjük.

```
adat <- "  modszer   pontszam
                A         42
                A         38
                A         67
                A         11
                B         28
                B         33
                B         58
                B         32  "

# adatmátrix beolvasása
df02 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df02$modszer <- factor(df02$modszer)
str(df02)  # adatmátrix szerkezete
#> 'data.frame':    8 obs. of  2 variables:
#>  $ modszer : Factor w/ 2 levels "A","B": 1 1 1 1 2 2 2 2
#>  $ pontszam: int  42 38 67 11 28 33 58 32
```

Kétmintás t-próba végrehajtásához továbbra is a `t.test()` függvényt használjuk, de `formula=<numerikus vektor> ~ <faktor>` és `var.equal=T` argumentummal.

```
# --= kétmintás t-próba =--
t.test(formula=pontszam ~ modszer, data = df02, var.equal = T,
       conf.level = 0.95)
#>
#>  Two Sample t-test
#>
#> data:  pontszam by modszer
#> t = 0.13111, df = 6, p-value = 0.9
#> alternative hypothesis: true difference in means between group A and group B
#>   is not equal to 0
#> 95 percent confidence interval:
#>  -30.90925  34.40925
#> sample estimates:
#> mean in group A mean in group B
#>           39.50           37.75
```

A fenti outputban megjelenik a próbastatisztika értéke (`t=`), a szabadsági fokok száma (`df=`) és a p érték (`p-value=`). Az `alternative hypothesis:` sor itt is megfogalmazza az ellenhipotézist: a két csoport (A és B) várható értékének különbsége nem nulla, azaz a két csoportban nem azonosak a várható értékek. A következő két sorban a várható értékek különbségére vonatkozó 95%-os megbízhatóságú konfidencia intervallum határa jelenik meg, majd az utolsó sorban a mintaátlag, mindkét csoportban. A nullhipotézist itt is megtartjuk ($t(6) = 0{,}131$, $p = 0{,}900$), azaz a két módszer hatékonysága nem tér el egymástól.

A `var.equal=F` argumentummal a fenti függvényhívás a Welch-féle d-próbát hajtja végre.

```
# --= Welch-féle d-próba =--
t.test(formula=pontszam ~ modszer, data = df02, var.equal = F,
       conf.level = 0.95)
#>
#>  Welch Two Sample t-test
#>
#> data:  pontszam by modszer
#> t = 0.13111, df = 4.894, p-value = 0.9009
#> alternative hypothesis: true difference in means between group A and group B
#>   is not equal to 0
#> 95 percent confidence interval:
#>  -32.78459  36.28459
#> sample estimates:
#> mean in group A mean in group B
#>           39.50           37.75
```

A fenti outputot hasonlóan értelmezzük, mint a kétmintás t-próba esetében. A próba továbbra sem szignifikáns ($t(4{,}894) = 0{,}131; p = 0{,}901$), a két módszer hatékonysága nem tér el egymástól. A Welch-féle d-próba esetén a szabadsági fok tizedes tört is lehet.

10.2.3. Páros t-próba

A páros t-próba páros megfigyeléseket tartalmazó adatmátrixon alapul, két populáció várható értékét teszteli, miszerint a párok közötti különbség statisztikailag különbözik-e nullától. Tegyük fel, hogy súlycsökkentő kúrán vesz részt 4 személy. Mindenki testsúlyát megmérték a kúra előtt, és a kúra után is.

Páros minta adatbázisa két különböző formában is előfordulhat, széles vagy hosszú formában. Tekintsük a súlycsökkentő kúra eredményeit először a széles formában.

```
adat <- "  szemely   elotte   utana
                a       72      64
                b       88      81
                c       77      76
                d       91      86  "

# adatmátrix beolvasása
df03_szeles <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df03_szeles$szemely <- factor(df03_szeles$szemely)
str(df03_szeles) # adatmátrix szerkezete
#> 'data.frame':    4 obs. of  3 variables:
#>  $ szemely: Factor w/ 4 levels "a","b","c","d": 1 2 3 4
#>  $ elotte : int  72 88 77 91
#>  $ utana  : int  64 81 76 86
```

A fenti adatbázisban a személyek azonosítója is szerepel, de ez elhagyható, valójában nem szükséges a páros t-próbához.

Olvassuk be a fenti adattáblát hosszú formából is.

```
adat <- "  szemely   idopont   testsuly
                a    elotte         72
                b    elotte         88
                c    elotte         77
```

```
                d    elotte         91
                a     utana         64
                b     utana         81
                c     utana         76
                d     utana         86  "

# adatmátrix beolvasása
df03_hosszu <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df03_hosszu$szemely <- factor(df03_hosszu$szemely)
df03_hosszu$idopont <- factor(df03_hosszu$idopont)
str(df03_hosszu) # adatmátrix szerkezete
#> 'data.frame':    8 obs. of  3 variables:
#>  $ szemely : Factor w/ 4 levels "a","b","c","d": 1 2 3 4 1 2 3 4
#>  $ idopont : Factor w/ 2 levels "elotte","utana": 1 1 1 1 2 2 2 2
#>  $ testsuly: int  72 88 77 91 64 81 76 86
```

A fenti hosszú formátumra is igaz, hogy a személy faktor nem feltétlenül szükséges a próba végrehajtásához.

Páros t-próbához továbbra is `t.test()` függvényt használjuk a `paired=T` argumentummal, és széles formátumú adatbázist vár az argumentumába.

```
# --= páros t-próba széles adatbázisból =--
t.test(x = df03_szeles$elotte, y = df03_szeles$utana, paired=T,
       conf.level = 0.95)
#>
#>  Paired t-test
#>
#> data:  df03_szeles$elotte and df03_szeles$utana
#> t = 3.3918, df = 3, p-value = 0.04272
#> alternative hypothesis: true mean difference is not equal to 0
#> 95 percent confidence interval:
#>   0.324057 10.175943
#> sample estimates:
#> mean difference
#>            5.25
```

Az outputban szokásos módon megjelenik a próbastatisztika értéke (`t=`), a szabadsági fokok száma (`df=`) és a p érték (`p-value=`). Valamint az `alternative hypothesis:` sor végén az

ellenhipotézist: a két módszerhez tartozó változók várható értékének különbsége nem egyelő nullával, vagyis a két várható érték nem azonos. Láthatjuk a várható értékek különbségére vonatkozó 95%-os megbízhatóságú konfidencia intervallum határait és mintaátlagot a várható értékek különbségére. A nullhipotézist most elvetjük, a próba 5%-os szinten szignifikáns ($t(3) = 3{,}392$; $p = 0{,}043$), azaz a két módszer hatékonysága eltér egymástól.

A páros t-próba elvégzése hosszú formátumú adatbázis esetén is lehetséges, de először a `pivot_wider()` függvénnyel át kell alakítunk széles formátumra az adatbázist. A páros t-próba eredménye természetesen a fentivel megegyező lesz.

```
# --= páros t-próba hosszú adatbázisból =--
# elvégezzük a hosszú-széles átalakítást
df03_szeles2 <- tidyr::pivot_wider(data = df03_hosszu,
                                   names_from = idopont,
                                   values_from = testsuly)
t.test(x = df03_szeles2$elotte, y = df03_szeles2$utana, paired=T,
       conf.level = 0.95)
#>
#>  Paired t-test
#>
#> data:  df03_szeles2$elotte and df03_szeles2$utana
#> t = 3.3918, df = 3, p-value = 0.04272
#> alternative hypothesis: true mean difference is not equal to 0
#> 95 percent confidence interval:
#>   0.324057 10.175943
#> sample estimates:
#> mean difference
#>            5.25
```

10.2.4. Egyszempontos varianciaelemzés

Az egyszempontos varianciaanalízis (ANOVA) hasonló a független kétmintás t-próbához, azzal a különbséggel, hogy több mint két csoport összehasonlítására is képes. Most legyen három tanítási módszerünk (A, B és C), módszerenként 4, 4 és 3 diákkal. A módszerek hatékonyságot továbbra is egy dolgozat pontszámával mérjük.

```
adat <- "  modszer pontszam
                A       42
                A       38
                A       67
```

```
                A       11
                B       28
                B       33
                B       58
                B       32
                C       76
                C       92
                C       87  "

# adatmátrix beolvasása
df04 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df04$modszer <- factor(df04$modszer)
str(df04) # adatmátrix szerkezete
#> 'data.frame':    11 obs. of  2 variables:
#>  $ modszer : Factor w/ 3 levels "A","B","C": 1 1 1 1 2 2 2 2 3 3 ...
#>  $ pontszam: int  42 38 67 11 28 33 58 32 76 92 ...
```

Egyszempontos varianciaelemzéshez a `summary(aov())` függvényeket használjuk. Tipikusan két lépéseben végezzük a próbát, először új objektumot (például `aov_1`) hozunk létre az `aov()` függvénnyel, majd az objektumra alkalmazzuk a `summary()` függvényt, amely megjeleníti a szokásos anova táblázatot.

```
# --= egyszempontos varianciaelemzés =--
aov_1 <- aov(formula = pontszam ~ modszer, data=df04)
summary(aov_1)
#>             Df Sum Sq Mean Sq F value Pr(>F)
#> modszer      2   4698    2349   8.273 0.0113 *
#> Residuals    8   2272     284
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A fenti output tartalmazza a csoportok közötti és a csoporton belüli szabadsági fokok számát (`Df`), a próbastatisztika értékét (`F value`) és a p értéket (`Pr(>F)`). A próba szignifikáns ($F(2,8) = 8{,}273$; $p = 0{,}011$), azaz a három módszer hatékonysága nem azonos, így szükség van utóelemzésre is. Számos módszer közül választhatunk.

Hagyományosan a `TukeyHSD()` függvényt használhatjuk Tukey-próba végrehajtására. Páronkénti összehasonlításokat a `pairwise.t.test()` függvénnyel végezhetünk, a p értékek

módosítási lehetőségeit `p.adjust.methods=` argumentumával szabályozzuk, választhatunk például `"bonferroni"`, `"holm"`, `"hochberg"`, `"hommel"`, `"BH"`, `"BY"` és `"none"` értékekből. További információt a `?p.adjust` paranccsal kérhetünk az elérhető lehetőségekről. A legtöbb esetben a közös szóráson alapuló számítást elvetjük a `pool.sd=F` argumentummal.

```
# Egyszempontos varianciaelemzés – utóelemzés
# – Tukey-próba
TukeyHSD(x = aov_1, conf.level = 0.95)
#>   Tukey multiple comparisons of means
#>     95% family-wise confidence level
#>
#> Fit: aov(formula = pontszam ~ modszer, data = df04)
#>
#> $modszer
#>      diff        lwr      upr     p adj
#> B-A -1.75 -35.798505 32.29851 0.9881968
#> C-A 45.50   8.723411 82.27659 0.0187231
#> C-B 47.25  10.473411 84.02659 0.0154501
```

A Tukey-próba fenti outputjában az egyes csoportpárok várható értékeinek különbségét (`diff`), a rájuk vonatkozó konfidencia intervallum határait (`lwr` és `upr`), és annak a próbának a p értékét (`p adj`) láthatjuk, amelynek ellenhipotézise a két csoport várható értékének eltérését állítja. Az eredmények alapján a B és A módszerek között nem találunk szignifikáns eltérést ($p = 0{,}988$), míg a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}0187$ és $p = 0{,}0154$).

```
# Egyszempontos varianciaelemzés – utóelemzés
# – Bonferroni-féle páronkénti összehasonlítás, Holm módszer
pairwise.t.test(x = df04$pontszam, g = df04$modszer,
                p.adjust.method = "holm", pool.sd = F)
#>
#>  Pairwise comparisons using t tests with non-pooled SD
#>
#> data:  df04$pontszam and df04$modszer
#>
#>   A      B
#> B 0.9009 -
#> C 0.0440 0.0076
#>
#> P value adjustment method: holm
```

A `pairwise.t.test()` fenti outputjában csak p értékeket láthatunk táblázatos elrendezésben. A statisztikai próbák ellenhipotézise a sor és oszlop mentén megnevezett két csoport várható értékének eltérését állítja. Jelen esetben az A és B módszerek között nem találunk szignifikáns eltérést ($p = 0{,}901$), míg a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}044$ és $p = 0{,}008$).

Mivel a páronkénti összehasonlítások ebben az esetben nem vezetnek konfidencia intervallumok meghatározásához, érdemes a `{DescTools}` csomag `PostHocTest()` függvényét is megismernünk. A `method=` argumentum definiálja a páronkénti összehasonlítás módját, amely lehet `"hsd"`, `"bonf"`, `"lsd"`, `"scheffe"`, `"newmankeuls"` (lásd `?PostHocTest`). A `PostHocTest()` függvény hívása nagyon kényelmes, hiszen az `x=` argumentum a korábban létrehozott `aov_1` modellobjektum.

```
# Egyszempontos varianciaelemzés - utóelemzés
# - Tukey-próba
DescTools::PostHocTest(x = aov_1, method="hsd",  conf.level = 0.95)
#>
#>   Posthoc multiple comparisons of means : Tukey HSD
#>     95% family-wise confidence level
#>
#> $modszer
#>      diff     lwr.ci   upr.ci   pval
#> B-A -1.75 -35.798505 32.29851 0.9882
#> C-A 45.50   8.723411 82.27659 0.0187 *
#> C-B 47.25  10.473411 84.02659 0.0155 *
#>
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Az Tukey-próbán alapuló páros összehasonlítások alapján a B és A módszerek között nem találunk szignifikáns eltérést ($p = 0{,}988$), míg a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}019$ és $p = 0{,}016$). A `lwr.ci` és `upr.ci` oszlopokban a várható értékek különbségére vonatkozó 95%-os megbízhatóságú konfidencia intervallum határai láthatók.

```
# Egyszempontos varianciaelemzés - utóelemzés
# - Bonferroni-féle páronkénti összehasonlítás, Bonferroni módszer
DescTools::PostHocTest(x = aov_1, method="bonferroni",
                       conf.level = 0.95)
#>
#>   Posthoc multiple comparisons of means : Bonferroni
```

```
#>     95% family-wise confidence level
#>
#> $modszer
#>      diff     lwr.ci   upr.ci   pval
#> B-A -1.75 -37.684973 34.18497 1.0000
#> C-A 45.50   6.685793 84.31421 0.0230 *
#> C-B 47.25   8.435793 86.06421 0.0189 *
#>
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A Bonferroni-féle páros összehasonlítások esetén a B és A módszerek között nem találunk szignifikáns eltérést ($p = 1{,}000$), míg a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}023$ és $p = 0{,}019$). A `lwr.ci` és `upr.ci` oszlopokban a várható értékek különbségére vonatkozó 95%-os megbízhatóságú konfidencia intervallum határai láthatók.

Amennyiben egy kitüntetett csoport várható értékét szeretnénk összehasonlítani a többi csoportéval, azaz a Dunnet-próbát szeretnénk végrehajtani, akkor a `{DescTools}` csomag `DunnettTest()` függvényét használjuk. A kitüntetett (kontroll) csoport nevét a `control=` argumentumban nevezzük meg.

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Dunnett-próba, kontrollcsoport a "C" módszer
DescTools::DunnettTest(x = df04$pontszam, g = df04$modszer, control="C",
                       conf.level=0.95)
#>
#>   Dunnett's test for comparing several treatments with a control :
#>     95% family-wise confidence level
#>
#> $C
#>       diff    lwr.ci    upr.ci   pval
#> A-C -45.50 -79.71296 -11.28704 0.0136 *
#> B-C -47.25 -81.46296 -13.03704 0.0112 *
#>
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A fenti eredmények alapján a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}014$ és $p = 0{,}011$).

Amennyiben az egyszempontos varianciaelemzés szóráshomogenitási feltétele nem teljesül a `oneway.test()` függvényt használjuk.

```
# --= Welch-féle egyszempontos varianciaelemzés =--
oneway.test(formula = pontszam ~ modszer, data=df04)
#>
#>  One-way analysis of means (not assuming equal variances)
#>
#> data:  pontszam and modszer
#> F = 17.087, num df = 2.0000, denom df = 5.1938, p-value =
#> 0.005196
```

A fenti outputból kiolvasható a próbastatisztika értéke (`F=`), a két szabadsági fok (`num df=` és `denom df=`) és a p érték (`p-value=`). A próba ebben az esetben is szignifikáns ($F(2; 5{,}1938) = 17{,}087$; $p = 0{,}005$), azaz a három módszer hatékonysága nem azonos.

A szóráshomogenitás sérülése esetén a szokásos utóelemzési módszer a Games–Howell próba, amelyhez most az `{rstatix}` csomag `games_howell_test()` függvényét használjuk.

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Games--Howell próba, szóráshomogenitás sérülése esetén
library(rstatix)
gh.1 <- games_howell_test(formula = pontszam ~ modszer, data=df04,
                  conf.level = 0.95, detailed = F)
print(as.data.frame(gh.1)[c(-1,-8)], digits=2) # rövidebb kiírás
#>   group1 group2 estimate conf.low conf.high p.adj
#> 1      A      B     -1.8   -45.52        42 0.991
#> 2      A      C     45.5     0.97        90 0.047
#> 3      B      C     47.2    19.97        75 0.006
```

A fenti output alapján a C módszer várható értéke szignifikánsan eltér az A és B módszerekétől ($p = 0{,}047$ és $p = 0{,}006$). A `estimate` oszlopban a csoportok közötti különbség látható, a `conf.low` és `conf.high` oszlopokban pedig a várható értékek különbségére vonatkozó 95%-os megbízhatóságú konfidencia intervallum határai.

10.2.5. Összetartozó mintás egyszempontos varianciaelemzés

Az összetartozó mintás egyszempontos varianciaelemzés a páros t-próba általánosításának tekinthető: ugyanazokon a kísérleti egységeken kettőnél több időpontban vagy helyzetben történik mérés. Példánkban négy tanulónak felmérik három sportágban (úszás, futás, labdarúgás) az ügyességét egy 1-10-es skálán.

Az adatok széles vagy hosszú formátumban is előfordulhatnak. Mindkét eset beolvasását megmutatjuk, de megjegyezzük, hogy az R számára csak a hosszú formátum lesz megfelelő. Kezdjük a széles adatmátrixszal és alakítsuk át azonnal hosszú formátumúra.

```
adat <- "  szemely    uszas   futas labdarugas
                a        2       9          7
                b        1       8          4
                c        5       6          2
                d        1       7          3  "

# adatmátrix beolvasása
df05_szeles <- read.table(textConnection(adat), header=T, sep="")
```

```
# széles-hosszú átalakítás
df05_hosszu <- tidyr::pivot_longer(data = df05_szeles,
                                   cols = 2:4,
                                   names_to = "sport",
                                   values_to = "pontszam")
# faktorrá alakítások
df05_hosszu$szemely <- factor(df05_hosszu$szemely)
df05_hosszu$sport   <- factor(df05_hosszu$sport)
str(df05_hosszu) # adatmátrix szerkezete
#> tibble [12 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ szemely : Factor w/ 4 levels "a","b","c","d": 1 1 1 2 2 2 3 3 3 4 ...
#>  $ sport   : Factor w/ 3 levels "futas","labdarugas",..: 3 1 2 3 1 2 3 1 2 3 ...
#>  $ pontszam: int [1:12] 2 9 7 1 8 4 5 6 2 1 ...
```

Könnyebb dolgunk van, ha az adatok eleve hosszú formátumban állnak rendelkezésre. Ekkor csak a faktorrá alakításokat kell elvégeznünk.

```
adat <- "  szemely      sport  pontszam
                a      uszas         2
                b      uszas         1
                c      uszas         5
                d      uszas         1
                a      futas         9
                b      futas         8
                c      futas         6
                d      futas         7
                a labdarugas         7
```

```
                b labdarugas         4
                c labdarugas         2
                d labdarugas         3  "

# adatmátrix beolvasása
df05_hosszu <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df05_hosszu$szemely <- factor(df05_hosszu$szemely)
df05_hosszu$sport   <- factor(df05_hosszu$sport)
str(df05_hosszu) # adatmátrix szerkezete
#> 'data.frame':    12 obs. of  3 variables:
#>  $ szemely : Factor w/ 4 levels "a","b","c","d": 1 2 3 4 1 2 3 4 1 2 ...
#>  $ sport   : Factor w/ 3 levels "futas","labdarugas",..: 3 3 3 3 1 1 1 1 2 2 ...
#>  $ pontszam: int  2 1 5 1 9 8 6 7 7 4 ...
```

Egyszempontos ismételt méréses varianciaelemzést az `{afex}` csomag `aov_ez()` függvényével tudunk kényelmesen végrehajtani. Az `id=` argumentumba a személyeket azonosító faktort írjuk (`szemely`), a `within=` a csoporton belüli független faktort tartalmazza (`sport`), a függő változó (`pontszam`) a `dv=` paraméterbe kerül. Az `aov_ez()` által visszaadott `aov_ez_1` objektumot eltároljuk, a próba eredményét pedig a `summary(aov_ez_1)` szolgáltatja.

```
# --= összetartozó mintás egyszempontos varianciaelemzés =--
library(afex)
aov_ez_1 <- aov_ez(id = "szemely", within = "sport", dv="pontszam",
                   data = df05_hosszu)
summary(aov_ez_1)
#>
#> Univariate Type III Repeated-Measures ANOVA Assuming Sphericity
#>
#>              Sum Sq num Df Error SS den Df F value   Pr(>F)
#> (Intercept) 252.083      1   8.9167      3  84.813 0.002708 **
#> sport        57.167      2  20.8333      6   8.232 0.019054 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#>
#> Mauchly Tests for Sphericity
#>
#>       Test statistic p-value
```

```
#> sport        0.19046 0.19046
#>
#>
#> Greenhouse-Geisser and Huynh-Feldt Corrections
#>  for Departure from Sphericity
#>
#>        GG eps Pr(>F[GG])
#> sport 0.55263    0.05616 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#>          HF eps Pr(>F[HF])
#> sport 0.6388785  0.0453546
```

A megjelenő output három nagyobb részből áll. A középső rész (`Mauchly Tests for Sphericity`) a Mauchly-próba eredményét tartalmazza, amely az összetartozó mintás varianciaanalízis egyik előfeltételét, a szfericitási feltétel meglétét teszteli. Leolvashatjuk a próbastatisztika értékét (`Test statistic`) és a p értéket (`p-value`). Mivel esetünkben a $p = 0{,}190$, így azt mondjuk, hogy a szfericitási feltétel teljesül, és a fenti output felső részét tekintjük. Itt láthatjuk az egyszempontos ismételt méréses varianciaelemzés eredményét (`Univariate Type III Repeated-Measures ANOVA Assuming Sphericity`). Elegendő a táblázat alsó sorát tekinteni (`sport`), amely tartalmazza a szabadsági fokokat (`num Df` és `den Df`), a próbastatisztika értékét (`F value`) és a p értéket (`Pr(>F)`). A próba szignifikáns ($p = 0{,}019$), így szükség van utóelemzésre.
Amennyiben a Mauchly-próba eredménye szignifikáns lenne, akkor az output 3. részét kellene tekinteni, a `Greenhouse-Geisser and Huynh-Feldt Corrections` utáni részt. Itt két további hipotézisvizsgálat eredményét láthatjuk, amely a szfericitás sérülése esetén az összetartozó egyszempontos varianciaelemzés p értékeit szolgáltatja. A `Pr(>F[GG])` a Greenhouse-Geisser korrekció, a `Pr(>F[HF])` Huynh-Feldt korrekció p értékeit mutatja.

A fenti output alapján a három sportág hatékonysága eltér egymástól ($F(2{,}6) = 8{,}232$; $p = 0{,}019$), a Mauchly-próba eredménye alapján a szfericitási feltétel fennáll ($p = 0{,}190$). Szükség van utóelemzésre.

Az összetartozó mintás varianciaelemzés utóelemzésének klasszikus módja R-ben a már korábban megismert `pairwise.t.test()` a `paired=T` argumentummal. A `p.adjust.method=` értékei a korábban megismert értékek lehetnek.

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Bonferroni-féle páronkénti összehasonlítás, Holm módszer
pairwise.t.test(x = df05_hosszu$pontszam, g = df05_hosszu$sport,
```

```
                paired = T,
                p.adjust.method = "holm")
#>
#>  Pairwise comparisons using paired t tests
#>
#> data:  df05_hosszu$pontszam and df05_hosszu$sport
#>
#>            futas labdarugas
#> labdarugas 0.018 -
#> uszas      0.071 0.379
#>
#> P value adjustment method: holm
```

Jelen esetben a futás és labdarúgás között szignifikáns eltérést találunk ($p = 0{,}018$), míg a labdarúgás és úszás között nem találunk szignifikáns eltérést ($p = 0{,}379$) és a futás és úszás között sem ($p = 0{,}071$).

Az utóelemzések elvégzésére több lehetőséget nyújt számunkra az `{emmeans}` csomag, amely más módszerekkel is képes módosítani a megjelenő p értékeket a páros összehasonlítások során, illetve kontrollcsoporttal való összehasonlítást is lehetővé tesz. A lenti parancsokban vegyük észre, hogy a korábban eltárolt `aov_ez_1` objektumot használtuk a `emmeans()` függvényben. A `specs=` argumentumban formulával meghatározzuk az összehasonlítás módját. Páronkénti összehasonlítások esetén a `pairwise`, kontroll csoporttal való összehasonlítás esetén a `trt.vs.ctrl` beépített függvénynevet használjuk a formula bal oldalán. A jobb oldalt a `sport` független faktort adjuk meg (`?"contrast-methods"`). Az `adjust=` argumentummal beállíthatjuk a p érték módosítását, a szokásos értékekhez képest használhatjuk még a `"tukey"`,`"scheffe"`,`"sidak"`,`"dunnettx"` és `"mvt"` értékeket is (további információ: `?summary.emmGrid`).

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Tukey-próba
library(emmeans)
emmeans(object = aov_ez_1, specs = pairwise ~ sport,
        adjust="tukey", level=0.95)
#> $emmeans
#>  sport      emmean    SE df lower.CL upper.CL
#>  futas        7.50 0.645  3    5.446     9.55
#>  labdarugas   4.00 1.080  3    0.563     7.44
#>  uszas        2.25 0.946  3   -0.762     5.26
#>
```

```
#> Confidence level used: 0.95
#>
#> $contrasts
#>  contrast           estimate   SE df t.ratio p.value
#>  futas - labdarugas     3.50 0.50  3   7.000  0.0122
#>  futas - uszas          5.25 1.44  3   3.656  0.0702
#>  labdarugas - uszas     1.75 1.70  3   1.028  0.6117
#>
#> P value adjustment: tukey method for comparing a family of 3 estimates
```

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Bonferroni-féle páronkénti összehasonlítás, Holm módszer
library(emmeans)
emmeans(object = aov_ez_1, specs = pairwise ~ sport,
        adjust="holm", level=0.95)
#> $emmeans
#>  sport      emmean    SE df lower.CL upper.CL
#>  futas        7.50 0.645  3    5.446     9.55
#>  labdarugas   4.00 1.080  3    0.563     7.44
#>  uszas        2.25 0.946  3   -0.762     5.26
#>
#> Confidence level used: 0.95
#>
#> $contrasts
#>  contrast           estimate   SE df t.ratio p.value
#>  futas - labdarugas     3.50 0.50  3   7.000  0.0180
#>  futas - uszas          5.25 1.44  3   3.656  0.0707
#>  labdarugas - uszas     1.75 1.70  3   1.028  0.3794
#>
#> P value adjustment: holm method for 3 tests
```

```
# Összetartozó mintás egyszempontos varianciaelemzés - utóelemzés
# - Dunnett-próba, kontrollcsoport a labdarúgás
library(emmeans)
emmeans(object = aov_ez_1, specs = trt.vs.ctrl ~ sport, ref=2,
        level=0.95)
#> $emmeans
#>  sport      emmean    SE df lower.CL upper.CL
#>  futas        7.50 0.645  3    5.446     9.55
#>  labdarugas   4.00 1.080  3    0.563     7.44
```

```
#>  uszas        2.25 0.946  3   -0.762     5.26
#>
#> Confidence level used: 0.95
#>
#> $contrasts
#>  contrast           estimate  SE df t.ratio p.value
#>  futas - labdarugas     3.50 0.5  3   7.000  0.0107
#>  uszas - labdarugas    -1.75 1.7  3  -1.028  0.5699
#>
#> P value adjustment: dunnettx method for 2 tests
```

Az `emmeans()` függvény outputja mindhárom fenti esetben két részt tartalmaz. Az első rész megmutatja a három kondíció becsült átlagát a standard hibákkal és a konfidenciaintervallumokkal együtt. A másik rész az elvégzett páronkénti próbák eredményét tartalmazza, a táblázat végén a p értékkel. A fenti Tukey- és Holm-féle páronkénti vizsgálat összesen 3 összehasonlítás eredményét tartalmazza, de a kitüntetett csoportot használó Dunnett-próba csak két összehasonlítást végez. Utóbbi esetben a kontrollcsoportot a `ref=` argumentummal jelöljük meg.

10.2.6. Korreláció és lineáris regresszió

A korreláció és a lineáris regresszió egyaránt képes két kvantitatív változó (lineáris) kapcsolatát feltárni.

A korreláció mutatja meg, hogy az egyik változó szisztematikusan változik-e egy másik változó változásával. A változók egyenrangúak, és tipikusan a kapcsolat iránya és erőssége határozható meg. Bemutatjuk a korreláció három formáját, a Pearson-, Kendall- és Spearman féle korrelációt, bár utóbbi kettőt nemparaméteres eljárásnak tekintjük.

Az egyszerű lineáris regresszió (könyvünkben csak ezt az esetet mutatjuk be) végrehajtása során az egyik változót függetlennek, míg a másikat függőnek tekintjük. A cél a két változó közötti lineáris kapcsolat függvényszerű meghatározása. A lineáris regresszióhoz kapcsolódó tesztek parametrikusak, és feltételezik a normalitást, a homoszkedaszticitást és a maradékok függetlenségét, valamint a két változó közötti lineáris kapcsolatot.

Vizsgáljuk meg öt nyári nap megfigyelésével, a napi középhőmérséklet és az eladott jégkrémek száma közötti lineáris kapcsolatot. Kezdjük az adatok beolvasásával.

```
adat <- "  homerseklet  jegkrem
                    32      277
                    30      287
                    29      149
                    36      337
                    25      131
                    27      142
                    32      201
                    31      231  "

# adatmátrix beolvasása
df06 <- read.table(textConnection(adat), header=T, sep="")
```

```
# adatmátrix szerkezete
str(df06)
#> 'data.frame':    8 obs. of  2 variables:
#>  $ homerseklet: int  32 30 29 36 25 27 32 31
#>  $ jegkrem    : int  277 287 149 337 131 142 201 231
```

10.2.6.1. Korrelációszámítás

Korrelációszámítást a `cor.test()` függvénnyel végezhetünk, a `method=` argumentum `"pearson"`, `"kendall"` vagy `"spearman"` értékével döntünk a korreláció típusáról. A `cor.test()` minden esetben szolgáltat p értéket, így dönthetünk a populációbeli korrelációs együtthatók nullától való eltéréséről, konfidencia intervallumot azonban csak a Pearson-féle korrelációs együttható esetében kapunk. Ezért a `{DescTools}` csomag `KendallTauB()` és `SpearmanRho()` függvényét használjuk a megbízhatósági intervallumok meghatározására.

```
# --= Pearson-féle korreláció =--
cor.test(formula = ~ homerseklet + jegkrem, data=df06,
         method = "pearson", conf.level = 0.95)
#>
#>  Pearson's product-moment correlation
#>
#> data:  homerseklet and jegkrem
#> t = 4.0143, df = 6, p-value = 0.007004
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.3738293 0.9730093
```

```
#> sample estimates:
#>       cor
#> 0.8536321
```

```
# --= Kendall-féle korreláció =--
cor.test(formula = ~ homerseklet + jegkrem, data=df06,
         method = "kendall")
#>
#>  Kendall's rank correlation tau
#>
#> data:  homerseklet and jegkrem
#> z = 2.3688, p-value = 0.01784
#> alternative hypothesis: true tau is not equal to 0
#> sample estimates:
#>       tau
#> 0.6910233
# 95%-os konfidencia intervallum a tau-b-re
DescTools::KendallTauB(x = df06$homerseklet, y=df06$jegkrem,
                       conf.level = 0.95)
#>     tau_b    lwr.ci    upr.ci
#> 0.6910233 0.2223790 1.0000000
```

```
# --= Spearman-féle rangkorreláció =--
cor.test(formula = ~ homerseklet + jegkrem, data=df06,
         method = "spearman")
#>
#>  Spearman's rank correlation rho
#>
#> data:  homerseklet and jegkrem
#> S = 15.592, p-value = 0.01384
#> alternative hypothesis: true rho is not equal to 0
#> sample estimates:
#>       rho
#> 0.8143859
# 95%-os konfidencia intervallum a rho-ra
DescTools::SpearmanRho(x = df06$homerseklet, y=df06$jegkrem,
                       conf.level = 0.95)
#>       rho    lwr.ci    upr.ci
#> 0.8143859 0.2574669 0.9651708
```

A fenti outputokból kiolvasható, hogy mindhárom korrelációs együtthatóra vonatkozó hipo-

tézisvizsgálat szignifikáns a megjelenő p értékek alapján, továbbá láthatjuk a mintabeli r, τ_b és r_S értékeket és a 95%-os megbízhatóságú konfidencia intervallum határait. A Spearman-féle rangkorrelációra vonatkozó vizsgálat alapján például azt mondhatjuk, hogy a hőmérséklet és a jégkrémek közötti rangkorrelációs együttható szignifikánsan eltér a nullától ($r_S = 0{,}814; p = 0{,}014$), azaz a hőmérséklet és a jégkrémek közötti kapcsolat erős pozitív együtt járást mutat. A Spearman-féle rangkorrelációs együtthatóra vonatkozó 95%-os megbízhatóságú konfidencia intervalluma $95\%CI : [0{,}257; 0{,}965]$.

10.2.6.2. Regressziószámítás

Egyszerű lineáris regressziót a `summary(lm())` függvényekkel hozhatunk létre. Első lépésben itt is érdemes eltárolni az `lm()` által szolgáltatott modellobjektumot (`lm_1`), majd a `summary()` már a hipotézisvizsgálat eredményét mutatja meg.

Az `lm()` függvényben a `formula=` argumentumot `<függő változó> ~ <független változó>` formában kell megadnunk, amely esetünkben `jegkrem ~ homerseklet` alakú.

```
# --= egyszerű lineáris regresszió =--
lm_1 <- lm(formula = jegkrem ~ homerseklet, data=df06)
summary(lm_1)
#>
#> Call:
#> lm(formula = jegkrem ~ homerseklet, data = df06)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -52.280 -22.346   1.659  15.935  72.469
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -366.695    146.786  -2.498   0.0466 *
#> homerseklet   19.374      4.826   4.014   0.0070 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 43.03 on 6 degrees of freedom
#> Multiple R-squared:  0.7287, Adjusted R-squared:  0.6835
#> F-statistic: 16.11 on 1 and 6 DF,  p-value: 0.007004
```

A fenti viszonylag terjedelmes output első sorai (`Call:`) megismétlik az `lm()` függvényhívás alakját, hogy egy későbbi felhasználás során tudjuk mire vonatkozik az output.

A következő részben (`Residuals:`) a reziduumok eloszlására vonatkozó statisztikai mutatókat olvashatjuk le, amelyekkel gyors ellenőrzést végezhetünk a módszer előfeltételein: a mediánnak nulla körüli értéknek kell lennie, a legkisebb és legnagyobb érték abszolút értékben körülbelül meg kell egyezzen.

A legizgalmasabb `Coefficients:` részben, az `Estimate` oszlopban láthatjuk a regressziós egyenes együtthatóinak mintából becsült értékeit. Továbbá az `Std. Error` oszlopban a becslések standard hibáit is láthatjuk. A `t value` oszlopban az együtthatók t próbastatisztikáinak értékét olvashatjuk le, a `Pr(>|t|)` oszlopban az elvégzett hipotézisvizsgálat `p értékei` olvashatók. Mindkét együtthatóra leolvasható a p érték, de a tengelymetszet `(Intercept)` sorában csak akkor kell figyelni ezt az értéket, ha azt vizsgáljuk, hogy az origón átmegy-e a regressziós egyenes. A `homerseklet` sorában lévő p érték adja meg a választ, hogy a jégkrém fogyasztás függ-e a hőmérséklettől (függ, $p = 0{,}007$).

Az output utolsó három sora a teljes modellre vonatkozik. A `Residual standard error:` a regressziós becslés standard hibáját és a szabadsági fokok számát tartalmazza. A következő sorban (`Multiple R-squared:`) a determinációs együttható és a korrigált determinációs együttható értékét olvashatjuk le, ebben az egyszerű kétváltozós esetben a determinációs együttható megegyezik a korrelációs együttható négyzetével, a korrigált mutatót pedig csak többszörös regresszió esetén érdemes figyelni. Az utolsó sorban a modell egészére vonatkozó F-próbastatisztika értéke, szabadsági fokainak száma és a p érték olvasható. Kétváltozós esetben a meredekségre vonatkozó t-próba eredményével azonos értéket kapunk, többszörös regresszió esetén viszont eltér a két érték.

Példánkban a hőmérséklet és a jégkrémek közötti lineáris kapcsolat szignifikáns ($F(1; 6) = 16{,}11; p = 0{,}007$), a hőmérséklet növekedésével a jégkrémek fogyasztása is növekszik. A regressziós egyenes meredeksége 19,37, azaz 1 fokos hőmérséklet-emelkedés esetén 19,37 jégkrém fogyasztásának emelkedésével számolhatunk.

A standardizált együtthatók megjelenítésének többszörös lineáris regresszió során van jelentősége, de az együtthatók intervallumbecslése egyszerű lineáris regresszió esetén is érdekes lehet.

```
# standardizált regressziós paraméterek,
#  konfidencia intervallumok a regressziós együtthatókra
lsr::standardCoefs(lm_1)
#>                    b      beta
#> homerseklet 19.37421 0.8536321
confint(object = lm_1, level = 0.95)
#>                   2.5 %    97.5 %
#> (Intercept) -725.866193 -7.523744
#> homerseklet    7.564737 31.183691
```

A lineáris regresszió igazi ereje az előrejelző képességében van, azaz tetszőleges független változóbeli értékhez kaphatunk egy modell által becsült függő változóbeli értéket. A `predict()` függvény ennek értelmében a `homerseklet` változó értékeit tartalmazó adattáblát vár a második argumentumában, az első argumentum pedig a korábban már eltárolt `lm_1` modellobjektum lesz.

```
# Predikció lineáris regresszióval
# adatok előkészítése a predikcióhoz
uj_adat <- data.frame(homerseklet=c(25, 30, 35))
predict(object = lm_1, newdata = uj_adat) # predikció végrehajtása
#>        1        2        3
#> 117.6604 214.5314 311.4025
```

A fenti outputban a 25, 30 és 35 fokos középhőmérséklettel rendelkező napok becsült jégkrém fogyásáról kapunk információt: 117,66, 214,53 és 311,40 jégkrém.

A mintabeli magyarázó változó összes értékére is megkaphatjuk a becsült értékeket a `fitted()` függvény segítségével. Ha megjelenítenénk a magyarázó változó és a függő változó becsült értékeit, akkor a regressziós egyenes pontjait kapnánk vissza.

```
# a jósolt Y pontok, az egyenes Y pontjai
fitted(lm_1)
#>        1        2        3        4        5        6        7        8
#> 253.2799 214.5314 195.1572 330.7767 117.6604 156.4088 253.2799 233.9057
```

A lineáris regressziós modell alkalmazási feltételeit is érdemes megvizsgálni. A regressziós diagnosztikának nevezett eljárás több pontot is tartalmaz, amely elvégzéséhez segítő függvényeket ajánl az R. A `residuals()` függvény magukat a reziduumokat szolgáltatja, így a normalitási feltétel segítségével akár tesztelhető is.

```
# reziduumok kiírása
residuals(lm_1)
#>         1         2         3         4         5         6         7
#>  23.72013  72.46855 -46.15723   6.22327  13.33962 -14.40881 -52.27987
#>         8
#>  -2.90566
# reziduumok normalitásának tesztelése
shapiro.test(residuals(lm_1))
#>
#>  Shapiro-Wilk normality test
```

```
#>
#> data:  residuals(lm_1)
#> W = 0.95405, p-value = 0.7519
```

Összefoglalás

A t-próbák különböző változatai a `t.test()` függvénnyel végezhetők el. A `paired=` argumentumot használva páros mintás t-próbát végezhetünk, a `var.equal=` argumentummal pedig a szóráshomogenitás feltételét is ellenőrizhetjük. A kétmintás t-próba általánosítása az egyszempontos varianciaelemzés, amelyet az `aov()` függvénnyel hajthatunk végre. Az egyszempontos varianciaelemzés utóelemzései közül a Tukey- és Dunnett-próbát a `{DescTools}` csomag `PostHocTest()` és `DunnettTest()` függvényeivel végezhetjük el. A szóráshomogenitás sérülése esetén a Welch-féle varianciaelemzést alkalmazzuk, amelyhez a `oneway.test()` függvényt használjuk. Az összetartozó mintás egyszempontos varianciaelemzéshez az `{afex}` csomag `aov_ez()` függvényét használjuk, amelynek utóelemzéseihez az `{emmeans}` csomagot használjuk. A korrelációszámításhoz a `cor.test()` függvényt használjuk, amelynek `method=` argumentumában a Pearson-, Kendall- és Spearman-féle korrelációt is megadhatjuk. A lineáris regresszióhoz az `lm()` függvényt használjuk, amelynek eredményeit a `summary()` függvénnyel elemezhetjük. A regressziós diagnosztikához a `residuals()` és `fitted()` függvényeket használhatjuk, becslésekhez a `predict()` függvényt.

Feladatok

1. Az eddig ismertetett próbák közül az u-próba (z-próba) különböző változatai kimaradtak. A `{DescTools}` csomag `ZTest()` függvénye az egymintás, a kétmintás és a páros eseteket is képes kezelni. Adjunk példát ezekre a próbákra a `?ZTest` tanulmányozása után!
2. A kétmintás t-próba és az egyszempontos varianciaelemzés több alternatívája is elérhető a `{onewaytests}` csomagban. Ezen próbák null- és ellenhipotézise ugyanúgy a várható értékekre vonatkozik. Mutassunk példát ezekre az eljárásokra Dag és mtsai. (2018) alapján!
3. A páros vizsgálatok elemzésére a `{PairedData}` csomagot használhatjuk. Mutassunk példát azokra a kényelmi lehetőségekre, amelyek az ábrák rajzolását támogatják!
4. Egyszempontos varianciaelemzéshez több módszerrel is készíthetünk magyarázó átlagábrát. Foglaljuk össze az ismert eseteket és egészítsük ki a `{PASWR2}` csomag `oneway.plots()`, a `{gplots}` csomag `plotmeans()` és a `{yarrr}` csomag `pirateplot()` függvényével.

5. Az egyszerű lineáris regresszió alkalmazási feltételei nem merülnek ki a reziduumok normalitásában. Ismertessük a lehetséges további feltételeket, és azok R-beli vizsgálati lehetőségeit! Használjuk a TidyTuesday csokoládék kakaótartalmát tartalmazó adatbázisát. Mi a kakaótartalom és a csokoládé kedvelésének kapcsolata? Használjuk a beolvasásához a következő kódokat:

```
# install.packages("tidytuesdayR") # tidytuesdayR csomag telepítése
# A TidyTuesday adatok letöltése a Github-ról
tuesdata <- tidytuesdayR::tt_load('2022-01-18', )
# a két vizsgált oszlop leválogatása; tibble-ből data frame konverzió
chocolate <- as.data.frame(
  tuesdata$chocolate[c("cocoa_percent", "rating")])
# a százalékjelet is tartalmazó kakaótartalom numerikussá alakítása
chocolate$cocoa_percent <- as.numeric(gsub("%", "",
                                           chocolate$cocoa_percent))
```

10.3. Nemparaméteres próbák 🙂

Miről lesz szó? Ebben a fejezetben

- áttekintjük a klasszikus nemparaméteres próbákat,
- az egyváltozós, független kétmintás és páros mintás próbákat,
- valamint a 3 vagy több csoport/helyzet mediánjait összevető próbákat és utóvizsgálataikat.

A nemparaméteres (eloszlásfüggetlen) próbák kisebb statisztikai erővel rendelkeznek a paraméteres próbákhoz képest, így ha lehet, inkább az utóbbi csoportból válasszunk próbát. Sok esetben azonban az alkalmazási feltétel sérülése miatt – leggyakrabban a normalitási feltétel hiányában, vagy ordinális függő változó esetében –, nem alkalmazhatunk paraméteres próbát. Ekkor a nemparaméteres próbák sietnek a segítségünkre.

A hagyományos nemparaméteres próbák rang alapú tesztek. A függő változó numerikus értékei helyett a relatív rangokon alapul a vizsgálat.

Például képzeljük el, hogy rendelkezésre áll 6 diák matematika dolgozat pontszáma. A `rank()` függvénnyel állíthatjuk elő a pontszámokhoz tartozó rangokat.

```
# rangok számítása
pontszam <- c(11, 7, 3, 14, 14, 16)
names(pontszam) <- letters[1:6]
pontszam  # nyers pontszámok
#>  a  b  c  d  e  f
#> 11  7  3 14 14 16
rank(pontszam)  # pontszámok rangsora
#>   a   b   c   d   e   f
#> 3.0 2.0 1.0 4.5 4.5 6.0
```

A legkisebb pontszámú a `c` tanul, így ő az 1. helyen van. A `b`-nek a következő legkisebb a pontszáma, így az ő rangja 2-es, és így tovább. Figyeljük meg, hogy a `d` és az `e` tanuló pontszámában holtverseny van (azonosak), így mivel ők a 4. és 5. helyen állnak a 4,5-ös közös rangot kapják meg. Ez utóbbi esetet nevezik kapcsolt rangnak. Látni fogjuk, hogy próbáink végrehajtását meghatározza az a tény, hogy tartalmaz a vizsgált változó kapcsolt rangot vagy sem.

A fenti példában azt is vegyük észre, hogy `c` tanuló pontszámát csökkentve (például 3-ról 2-re), ugyanúgy az 1 rangot rendelnénk `c`-hez, vagyis az abszolút pontszámokra vonatkozó információ elvész, és csak a relatív rangsor marad meg a próbák során.

10.3.1. Egymintás Wilcoxon-próba

A legegyszerűbb nemparaméteres próba egyetlen – populációban szimmetrikus – minta esetén a változó mediánját (nagyságszintjét) hasonlítja össze egy fix értékkel. Ez a fix érték az egymintás t-próbához hasonlóan lehet egy alapértelmezett, semleges, referenciaszerű vagy korábban publikált érték. A példánkban 7 hallgató egy oktató felkészültségét likert skálán (1-5) értékelte.

```
adat <- "  felkeszultseg
                       3
                       5
                       3
                       5
                       4
                       5
                       4  "

# adatmátrix beolvasása
```

```
df07 <- read.table(textConnection(adat), header=T, sep="")
str(df07) # adatmátrix szerkezete
#> 'data.frame':    7 obs. of  1 variable:
#>  $ felkeszultseg: int  3 5 3 5 4 5 4
```

Egymintás Wilcoxon-próbát a `wilcox.test()` függvénnyel hajthatjuk végre. Ha a mintaelemszám 50-nél kisebb, és nincsenek kapcsolt rangok, akkor egzakt próbát kapunk, egyébként normális eloszlás közelítésével számolja a p értéket, folytonossági korrekció nélkül. Az egzakt próba végrehajtását magunk is szabályozhatjuk az `exact=T` argumentummal, a folytonossági korrekciót pedig a `correct=T` argumentummal állíthatjuk. Kisebb mintaelemszám esetén a próba egzakt változatát érdemes használni (`exact=T`), de ez csak kapcsolt rangok hiányában áll rendelkezésre.

```
# --= egymintás Wilcoxon-próba, normális közelítés, folyt. korrekció nélkül =--
wilcox.test(x = df07$felkeszultseg, mu = 3,
            conf.int = T, conf.level=0.80, correct=F, exact=F)
#>
#>  Wilcoxon signed rank test
#>
#> data:  df07$felkeszultseg
#> V = 15, p-value = 0.03843
#> alternative hypothesis: true location is not equal to 3
#> 80 percent confidence interval:
#>  4.000064 5.000000
#> sample estimates:
#> (pseudo)median
#>       4.500042
```

A fenti próbában (az alacsony mintaelemszám ellenére) normális közelítéssel számolt egymintás Wilcoxon-próbát hajtottunk végre, folytonossági korrekció nélkül. A próba szignifikáns ($V = 15$; $p = 0{,}038$), azaz a medián nem egyenlő 3-mal. A szokásostól eltérő, 80%-os megbízhatóságú konfidencia intervallum határai: $80\%CI : [4{,}00; 5{,}00]$.

10.3.2. Egymintás előjel-próba

Egyetlen minta esetén az előjel-próba is használható a medián vizsgálatára. A már ismert `df07` adatbázisban 7 hallgató egy oktató felkészültségét likert skálán (1-5) értékelte.

```r
# --= egymintás előjel-próba =--
library(DescTools)
SignTest(x = df07$felkeszultseg, mu = 3, conf.level = 0.95)
#>
#>  One-sample Sign-Test
#>
#> data:  df07$felkeszultseg
#> S = 5, number of differences = 5, p-value = 0.0625
#> alternative hypothesis: true median is not equal to 3
#> 98.4 percent confidence interval:
#>  3 5
#> sample estimates:
#> median of the differences
#>                         4
```

A fenti output alapján nem tudjuk elvetni a nullhipotézist, miszerint a medián egyenlő 3-mal ($S = 5; p = 0{,}063$). A 95%-os megbízhatóságú konfidencia intervallum határai: $95\%CI$: [3,00; 5,00].

10.3.3. Mann–Whitney-próba

A Mann–Whitney-próba a kétmintás t-próba nemparaméteres megfelelője. A példánkban 4-4 hallgató most két oktató (A és B) felkészültségét ítéli meg egy likert skálán (1-5).

```r
adat <- "  oktato felkeszultseg
                A             1
                A             1
                A             3
                A             2
                B             4
                B             5
                B             5
                B             4  "

# adatmátrix beolvasása
df08 <- read.table(textConnection(adat), header=T, sep="")
# faktorrá alakítás
df08$oktato <- factor(df08$oktato)
str(df08) # adatmátrix szerkezete
```

```
#> 'data.frame':    8 obs. of  2 variables:
#>  $ oktato       : Factor w/ 2 levels "A","B": 1 1 1 1 2 2 2 2
#>  $ felkeszultseg: int  1 1 3 2 4 5 5 4
```

A Mann–Whitney-próba végrehajtásához a `wilcox.test()` függvényt használhatjuk, amelynek `formula=` argumentumban a függő változót és a csoportosító változót kell megadnunk. A `conf.int=T` és `conf.level=0.95` argumentumokkal 95%-os megbízhatóságú konfidencia intervallumot kapunk a medián különbségre. A `correct=F` argumentummal kikapcsolhatjuk a folytonossági korrekciót, az `exact=F` argumentummal pedig normális közelítést kérünk a p értékhez.

```
# --= Mann-Whitney-próba =--s
wilcox.test(formula = felkeszultseg ~ oktato, data=df08,
            conf.int = T, conf.level=0.95, correct=F, exact=F)
#>
#>  Wilcoxon rank sum test
#>
#> data:  felkeszultseg by oktato
#> W = 0, p-value = 0.01868
#> alternative hypothesis: true location shift is not equal to 0
#> 95 percent confidence interval:
#>  -4 -1
#> sample estimates:
#> difference in location
#>              -2.999974
```

A fenti eredmény azt mutatja, hogy szignifikáns különbség van a két oktató megítélése között ($W = 0; p = 0{,}019$).

10.3.4. Kétmintás Mood medián-próba

Két független csoport mediánjának összehasonlítására a Mood medián-próba is használható. A Mood medián-próba végrehajtásához a `{RVAideMemoire}` csomag `mood.medtest()` függvényét használhatjuk, amelynek `formula=` argumentumában a függő változót és a csoportosító változót kell megadnunk. Az `exact=F` argumentummal a normális közelítést kérhetjük a p értékhez. Az `{rcompanion}` csomag `groupwiseMedian()` függvényével pedig megkaphatjuk a mediánok konfidencia intervallumait is. A `bca=FALSE` argumentummal a bootstrap konfidencia intervallumot tilthatjuk, a `conf=0.95` argumentummal pedig 95%-os megbízhatóságú konfidencia intervallumot kérhetünk.

A `df08` adatbázist használjuk, ahol 4-4 hallgató két oktató (A és B) felkészültségét ítéli meg egy likert skálán (1-5).

```
# --= Mood medián-próba =--
library(RVAideMemoire)
mood.medtest(formula = felkeszultseg ~ oktato, data=df08, exact=F)
#>
#>  Mood's median test
#>
#> data:  felkeszultseg by oktato
#> X-squared = 4.5, df = 1, p-value = 0.03389
library(rcompanion)
groupwiseMedian(formula = felkeszultseg ~ oktato, data=df08, bca=FALSE,
                perc=TRUE, conf = 0.95)
#>   oktato n Median Conf.level Percentile.lower Percentile.upper
#> 1      A 4    1.5       0.95               1               3
#> 2      B 4    4.5       0.95               4               5
```

A fenti output alapján a két oktató mediánja között szignifikáns különbség van ($\chi^2(1) = 4{,}5$; $p = 0{,}034$). A mediánok 95%-os megbízhatóságú konfidencia intervallumai 1-3 (A oktató) és 4-5 (B oktató) között helyezkednek el, az A oktató mediánja 1,5, míg a B oktató mediánja 4,5.

10.3.5. Páros Wilcoxon-próba

A páros Wilcoxon-próba a páros t-próba nemparaméteres megfelelője. A páros t-próba alkalmazásának feltétele, hogy a különbségek normális eloszlásúak legyenek. A páros Wilcoxon-próba esetén ez a feltétel nem szükséges, így a próbát akkor is alkalmazhatjuk, ha a különbségek eloszlása nem normális.

A példánkban 6 hallgató egy oktató felkészültségét egy hónap eltéréssel kétszer ítéli meg. Az adataink szokásos módon széles formátumban fordulnak elő.

```
adat <- "  hallgato  oktober   november
                 a        2          5
                 b        2          4
                 c        1          3
                 d        3          4
                 e        3          5
                 f        4          5  "
```

```
# adatmátrix beolvasása
df09_szeles <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df09_szeles$hallgato <- factor(df09_szeles$hallgato)
str(df09_szeles)  # adatmátrix szerkezete
#> 'data.frame':    6 obs. of  3 variables:
#>  $ hallgato: Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6
#>  $ oktober : int  2 2 1 3 3 4
#>  $ november: int  5 4 3 4 5 5
```

Elképzelhető azonban, hogy hosszú formátumban állnak rendelkezésünkre az adatok.

```
adat <- "  hallgato    idopont  felkeszultseg
                 a     oktober              2
                 b     oktober              2
                 c     oktober              1
                 d     oktober              3
                 e     oktober              3
                 f     oktober              4
                 a    november              5
                 b    november              4
                 c    november              3
                 d    november              4
                 e    november              5
                 f    november              5  "

# adatmátrix beolvasása
df09_hosszu <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df09_hosszu$hallgato <- factor(df09_hosszu$hallgato)
df09_hosszu$idopont <- factor(df09_hosszu$idopont)
str(df09_hosszu)  # adatmátrix szerkezete
#> 'data.frame':    12 obs. of  3 variables:
#>  $ hallgato     : Factor w/ 6 levels "a","b","c","d",..: 1 2 3 4 5 6 1 2 3 4 ...
#>  $ idopont      : Factor w/ 2 levels "november","oktober": 2 2 2 2 2 2 1 1 1 1 ...
#>  $ felkeszultseg: int  2 2 1 3 3 4 5 4 3 4 ...
```

Páros minta esetében a `wilcox.test()` függvény `paired=T` argumentumát kell használnunk a két különböző időpontban mért értékek összehasonlításához. Széles formátumú adatmátrix esetén a `x=` argumentumban az egyik, a `y=` argumentumban pedig a másik időpontban mért értékeket kell megadnunk.

```
# --= páros Wilcoxon-próba, széles adatmátrix =--
wilcox.test(x = df09_szeles$november, y = df09_szeles$oktober, paired=T,
            conf.int = T, conf.level=0.90, correct=F, exact=F)
#>
#>  Wilcoxon signed rank test
#>
#> data:  df09_szeles$november and df09_szeles$oktober
#> V = 21, p-value = 0.0256
#> alternative hypothesis: true location shift is not equal to 0
#> 90 percent confidence interval:
#>  1.000050 2.499967
#> sample estimates:
#> (pseudo)median
#>       1.999928
```

Hosszú formátumú adatmátrix esetén első lépésben elvégezzük a szélessé alakítást a `tidyr::pivot_wider()` függvénnyel, amelynek `names_from=` argumentumában a csoportosító változót, a `values_from=` argumentumban pedig a függő változót kell megadnunk.

```
# --= páros Wilcoxon-próba, hosszú adatmátrix =--
# hosszú-széles átalakítás
df09_szeles2 <- tidyr::pivot_wider(data = df09_hosszu,
                                   names_from = idopont,
                                   values_from = felkeszultseg)
wilcox.test(x = df09_szeles2$november, y = df09_szeles2$oktober, paired=T,
            conf.int = T, conf.level=0.90, correct=F, exact=F)
#>
#>  Wilcoxon signed rank test
#>
#> data:  df09_szeles2$november and df09_szeles2$oktober
#> V = 21, p-value = 0.0256
#> alternative hypothesis: true location shift is not equal to 0
#> 90 percent confidence interval:
#>  1.000050 2.499967
#> sample estimates:
```

```
#> (pseudo)median
#>       1.999928
```

Látható, hogy a páros Wilcoxon-próba szignifikáns ($V = 21; p = 0{,}026$), vagyis a két időpontban mért mediánok között szignifikáns különbség van. A mediánok 90%-os megbízhatóságú konfidencia intervalluma $90\%CI : [1{,}00; 2{,}50]$, azaz a populációbeli mediánok közötti különbség nagyjából 1 és 2,5 között helyezkedik el.

10.3.6. Páros előjel-próba

A páros előjel-próba szintén a páros t-próba nemparaméteres megfelelője. A `{DescTools}` csomag `SignTest()` függvényével hajtható végre, amelynek `x=` argumentumában az egyik, `y=` argumentumban pedig a másik időpontban mért értékeket kell megadnunk. A `conf.level=` argumentummal 95%-os megbízhatóságú konfidencia intervallumot kérhetünk.

A példánkban `df09_szeles` adatbázist használjuk, amely 6 hallgató egy oktató felkészültségét egy hónap eltéréssel, összesen kétszer ítéli meg.

```
# --= páros előjel-próba, széles adatmátrix =--
library(DescTools)
SignTest(x = df09_szeles$november, y = df09_szeles$oktober, conf.level = 0.95)
#>
#>  Dependent-samples Sign-Test
#>
#> data:  df09_szeles$november and df09_szeles$oktober
#> S = 6, number of differences = 6, p-value = 0.03125
#> alternative hypothesis: true median difference is not equal to 0
#> 96.9 percent confidence interval:
#>  1 3
#> sample estimates:
#> median of the differences
#>                         2
```

A fenti eredmények alapján a páros előjel-próba szignifikáns ($S = 6; p = 0{,}031$), vagyis a két időpontban mért mediánok között szignifikáns különbség van. A mediánok 97%-os megbízhatóságú konfidencia intervalluma $97\%CI : [1; 3]$, azaz a populációbeli mediánok közötti különbség nagyjából 1 és 3 között helyezkedik el.

10.3.7. Kruskal–Wallis-próba

A Kruskal–Wallis-próba a (független) egyszempontos varianciaelemzés nemparaméteres megfelelője. A próbát akkor alkalmazzuk, ha a függő változó eloszlása nem normális, vagy csak ordinális skálán mért. Kezelhetjük úgy a próbát, hogy a különböző csoportok mediánjait hasonlítja össze.

A példánkban 11 hallgató most három oktató (A, B és C) felkészültségét ítéli meg egy likert skálán (1-5).

```
adat <- "  oktato felkeszultseg
                A             1
                A             1
                A             3
                A             2
                B             4
                B             5
                B             5
                B             4
                C             1
                C             2
                C             3  "

# adatmátrix beolvasása
```

```
df10 <- read.table(textConnection(adat), header = T, sep = "")
# faktorrá alakítás
df10$oktato <- factor(df10$oktato)
str(df10)  # adatmátrix szerkezete
#> 'data.frame':    11 obs. of  2 variables:
#>  $ oktato       : Factor w/ 3 levels "A","B","C": 1 1 1 1 2 2 2 2 3 3 ...
#>  $ felkeszultseg: int  1 1 3 2 4 5 5 4 1 2 ...
```

Kruskal–Wallis-próbát a `kruskal.test()` függvénnyel hajthatunk végre, amelynek `formula=` argumentumában a függő változót és a csoportosító változót kell megadnunk. A `data=` argumentumban az adatmátrixot szerepeltetjük.

```
# --= Kruskal-Wallis próba =--
kruskal.test(formula = felkeszultseg ~ oktato, data = df10)
#>
```

```
#>  Kruskal-Wallis rank sum test
#>
#> data:  felkeszultseg by oktato
#> Kruskal-Wallis chi-squared = 7.3192, df = 2, p-value = 0.02574
```

```
# Kruskal-Wallis próba, utóvizsgálatok - Dunn-próba
library(DescTools)
DunnTest(formula = felkeszultseg ~ oktato, data = df10, method = "holm")
#>
#>  Dunn's test of multiple comparisons using rank sums : holm
#>
#>     mean.rank.diff   pval
#> B-A      5.7500000 0.0375 *
#> C-A      0.5833333 0.8145
#> C-B     -5.1666667 0.0755 .
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Látható, hogy a Kruskal–Wallis-próba szignifikáns ($\chi^2(2) = 7{,}32$; $p = 0{,}026$), vagyis a három oktatóra vonatkozó vélemények mediánja között szignifikáns különbség van. Az elvégezett utóvizsgálat (Dunn-próba) alapján a B oktató mediánja szignifikánsan eltér az A oktató mediánjától ($p = 0{,}0375$).

10.3.8. Többmintás Mood medián-próba

A többmintás Mood medián-próba a Kruskal–Wallis-próba alternatívája, több csoport mediánját hasonlítja össze.

A `df10` adatmátrixot használjuk, amelyben 11 hallgató három oktató (A, B és C) felkészültségét ítéli meg egy likert skálán (1-5). A próba elvégzéséhez a `mood.medtest()` függvényt használjuk, amelyet már a kétmintás esetben megismertünk.

```
# --= Mood medián-próba (többmintás eset) =--
library(RVAideMemoire)
mood.medtest(formula = felkeszultseg ~ oktato, data=df10,
             exact = FALSE)
#>
#>  Mood's median test
#>
```

```
#> data:  felkeszultseg by oktato
#> X-squared = 11, df = 2, p-value = 0.004087
library(rcompanion)
groupwiseMedian(formula = felkeszultseg ~ oktato, data=df10, bca=FALSE,
                perc=TRUE, conf = 0.95)
#>   oktato n Median Conf.level Percentile.lower Percentile.upper
#> 1      A 4    1.5       0.95               1               3
#> 2      B 4    4.5       0.95               4               5
#> 3      C 3    2.0       0.95               1               3
```

A fenti output megmutatja, hogy a három oktatóra vonatkozó megítélés nem azonos ($\chi^2(2) = 11$, $p = 0{,}004$). A mediánokra vonatkozó 95%-os megbízhatóságú konfidencia intervallumok: 1-3 (A oktató), 4-5 (B oktató) és 1-3 (C oktató).

Az utóvizsgálat elvégzéséhez a `{rcompanion}` csomag `pairwiseMedianTest()` függvényét használhatjuk, amely a páronkénti medián-próbát hajtja végre.

```
# Utóvizsgálat, Mood medián-próba (többmintás eset)
# - Páronkénti medián-próba
library(rcompanion)
pairwiseMedianTest(formula = felkeszultseg ~ oktato, data=df10,
                   exact  = NULL, method = "fdr")
#>   Comparison  p.value p.adjust
#> 1  A - B = 0 0.008151  0.02445
#> 2  A - C = 0   0.8231  0.82310
#> 3  B - C = 0   0.1797  0.26960
```

A páronkénti medián-próba eredményei alapján a B oktató mediánja szignifikánsan eltér az A oktató mediánjától ($p = 0{,}024$), míg a C oktató mediánja nem tér el szignifikánsan sem az A, sem a B oktató mediánjától ($p = 0{,}823$ és $p = 0{,}270$).

10.3.9. Friedman-próba

A Friedman-próba a páros Wilcoxon-próba általánosítása, az összetartozó egyszempontos varianciaelemzés nemparaméteres megfelelője. A próbát akkor alkalmazzuk, ha a függő változó eloszlása nem normális, vagy a változó ordinális skálán mért. Kezelhetjük úgy a próbát, hogy a különböző helyzetek/időpontok mediánjait hasonlítja össze.

Példánkban 6 hallgatónak három különböző időpontban kellett értékelni egy oktatót. Tegyük fel, hogy az adataink széles formátumban érhetők el. A próba elvégzéséhez hosszú formátumba kell alakítanunk az adatokat.

```
adat <- "  hallgato  oktober  november  december
                  a        2         5         3
                  b        2         4         4
                  c        1         3         2
                  d        3         4         1
                  e        3         5         1
                  f        4         5         2  "

# adatmátrix beolvasása
df11_szeles <- read.table(textConnection(adat), header=T, sep="")
# széles-hosszú átalakítás
df11_hosszu <- tidyr::pivot_longer(data = df11_szeles,
                                   cols = 2:4,
                                   names_to = "idopont",
                                   values_to = "felkeszultseg")
```

```
# faktorrá alakítások
df11_hosszu$hallgato <- factor(df11_hosszu$hallgato)
df11_hosszu$idopont <- factor(df11_hosszu$idopont)
dplyr::glimpse(df11_hosszu)  # adatmátrix szerkezete
#> Rows: 18
#> Columns: 3
#> $ hallgato      <fct> a, a, a, b, b, b, c, c, c, d, d, d, e, e, e, f, ~
#> $ idopont       <fct> oktober, november, december, oktober, november, ~
#> $ felkeszultseg <int> 2, 5, 3, 2, 4, 4, 1, 3, 2, 3, 4, 1, 3, 5, 1, 4, ~
```

Szerencsés esetben az adatok eleve hosszú formátumban állnak rendelkezésünkre. Ebben az esetben a fenti átalakításokra nincs szükség. A `df11_hosszu` adatmátrixot használjuk a továbbiakban a próba elvégzéséhez.

```
adat <- "  hallgato    idopont  felkeszultseg
                 a     oktober              2
                 b     oktober              2
                 c     oktober              1
                 d     oktober              3
                 e     oktober              3
                 f     oktober              4
                 a    november              5
                 b    november              4
                 c    november              3
```

```
                d    november             4
                e    november             5
                f    november             5
                a    december             3
                b    december             4
                c    december             2
                d    december             1
                e    december             1
                f    december             2  "

# adatmátrix beolvasása
df11_hosszu <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df11_hosszu$hallgato <- factor(df11_hosszu$hallgato)
df11_hosszu$idopont <- factor(df11_hosszu$idopont)
dplyr::glimpse(df11_hosszu)  # adatmátrix szerkezete
#> Rows: 18
#> Columns: 3
#> $ hallgato      <fct> a, b, c, d, e, f, a, b, c, d, e, f, a, b, c, d, ~
#> $ idopont       <fct> oktober, oktober, oktober, oktober, oktober, okt~
#> $ felkeszultseg <int> 2, 2, 1, 3, 3, 4, 5, 4, 3, 4, 5, 5, 3, 4, 2, 1, ~
```

Friedman-próbát a `friedman.test()` függvénnyel hajthatunk végre, amelynek `formula=` argumentumában a függő változót és a csoportosító változót kell megadnunk, illetve a személyek azonosító változóját is.

```
# --= Friedman-próba =--
friedman.test(formula = felkeszultseg ~ idopont | hallgato,
              data = df11_hosszu)
#>
#>  Friedman rank sum test
#>
#> data:  felkeszultseg and idopont and hallgato
#> Friedman chi-squared = 7.913, df = 2, p-value = 0.01913
```

A fenti eredmények azt mutatják, hogy a három időpontban mért felkészültség nem azonos ($\chi^2(2) = 7{,}91$; $p = 0{,}019$).

A Friedman-próba utóvizsgálatát a `{PMCMRplus}` csomag `frdAllPairsNemenyiTest()` függvényével hajthatjuk végre, amelynek `formula=` argumentumában a függő változót és a csoportosító változót kell megadnunk, illetve a személyek azonosító változóját is. A `data=` argumentumban az adatmátrixot szerepeltetjük.

```
# Friedman-próba, utóvizsgálat
# - Nemenyi-próba
library(PMCMRplus)
frdAllPairsNemenyiTest(formula = felkeszultseg ~ idopont | hallgato,
                       data = df11_hosszu)
#>          december november
#> november 0.055    -
#> oktober  0.989    0.038
```

Az utóvizsgálat megmutatta, hogy az októberi és a novemberi időpontban mért mediánok között szignifikáns különbség van ($p = 0{,}038$).

Összefoglalás

A nemparaméteres próbák olyan statisztikai eljárások, amelyeket akkor alkalmazunk, ha a paraméteres próbák feltételei – például a normalitás vagy a metrikus skálaszint – nem teljesülnek. Tipikus példák az ordinális skálán mért változók, vagy a kis elemszámú minták. Ezek a próbák a nyers értékek helyett azok rangsoraival dolgoznak, így robusztusabbak a szélsőértékekkel és az eloszlási torzulásokkal szemben. Egyetlen minta mediánjának vizsgálatára használható az `wilcox.test()` (egymintás Wilcoxon-próba) vagy a `SignTest()` (egymintás előjel-próba). Két független minta összehasonlítására szintén a `wilcox.test()` (Mann–Whitney-próba) szolgál, míg páros minta esetén ugyanennek a függvénynek a `paired=TRUE` argumentummal ellátott változata (páros Wilcoxon-próba), illetve a `SignTest()` függvény (páros előjel-próba) alkalmazható. Több független csoport mediánjait a `kruskal.test()` (Kruskal–Wallis-próba) vagy a `mood.medtest()` (Mood medián-próba) hasonlítja össze. Az összetartozó mérések esetén a `friedman.test()` függvény (Friedman-próba) használható. Bár ezek a próbák kisebb statisztikai erővel rendelkeznek, megbízható alternatívát nyújtanak nem ideális adatviszonyok mellett is.

Feladatok

1. Több nemparaméteres próbát is használja a `correct=` és az `exact=` argumentumokkal. Tekintsük át ezek lehetséges értékeit az egyes függvények esetén. Nézzünk utána, hogy ezeket milyen körülmények között érdemes használni!
2. Mood medián-próba esetén a `{RVAideMemoire}` csomag `mood.medtest()` függvényét

használtuk, de számos alternatívája létezik az R-ben. Melyek ezek?
3. A Kruskal–Wallis-próba utóvizsgálatára a Dunn-próbát használtuk, de számos alternatívája létezik az R-ben. Melyek ezek?
4. A Friedman-próba utóvizsgálatára a Nemenyi-próbát használtuk, de számos alternatívája létezik az R-ben. Melyek ezek?

10.4. Normalitás vizsgálata 🙂

Miről lesz szó? Ebben a fejezetben

- áttekintjük a Shapiro–Wilk és a Kolmogorov–Smirnov próbát,
- a ferdeségi és csúcsossági együtthatóra épülő D'Agostino-féle próbát,
- mindezt egy és több csoport esetén is.

Egy változó normalitásának vizsgálatára számos teszt érhető el, a leggyakoribbak a Shapiro-Wilk és a Kolmogorov–Smirnov próba. Ezek a tesztek csoportosítás nélküli és csoportosított változók esetén is könnyen használhatók. Minden esetben a nullhipotézis az, hogy az adatok eloszlása nem tér el a normálistól. A szignifikáns p érték ($p < 0{,}05$) arra utal, hogy az adatok nem normális eloszlásúak.

Normalitás nem csak a két fenti klasszikus próbával ellenőrizhető, a grafikus vizsgálatokkal például hisztogram (9.1.4. fejezet) vagy QQ-ábra (9.4. példa) megrajzolásával sokszor megbízhatóbb eredményt kapunk (Mangiafico, 2016).

A normalitás ellenőrzése alapozható a két alakmutatóra is, a ferdeségi és a csúcsossági együtthatóra, amelyek értéke normális eloszlás esetén nullával egyenlő.

10.4.1. Shapiro–Wilk próba

A normalitás ellenőrzésének formális eszközét adja a Shapiro–Wilk próba. Elképzelt példánkban 7 film IMDB értékelését gyűjtöttük össze. A cél a normalitás ellenőrzése.

```
adat <- "  ertekeles
                3,5
                5,7
                8,3
                5,2
                4,5
```

```
                5,4
                4,2  "

# adatmátrix beolvasása
df12 <- read.table(textConnection(adat), header=T, sep="", dec=",")
str(df12) # adatmátrix szerkezete
#> 'data.frame':    7 obs. of  1 variable:
#>  $ ertekeles: num  3.5 5.7 8.3 5.2 4.5 5.4 4.2
```

A Shapiro–Wilk próba végrehajtásához mindössze a `shapiro.test()` függvényben a vizsgált mintát kell megadnunk (`x=`). A megengedett mintaelemszám a 3 és 5000 közötti tartományból való.

```
# --= Shapiro-Wilk próba egy mintára =--
shapiro.test(x = df12$ertekeles)
#>
#>  Shapiro-Wilk normality test
#>
#> data:  df12$ertekeles
#> W = 0.89516, p-value = 0.3027
```

A Shapiro–Wilk próba outputja a próbastatisztika értékét (`W=`) és a p értéket (`p-value=`) tartalmazza. Az eredmények alapján a nullhipotézist nem utasítjuk el, azaz a minta normális eloszlásúnak tekinthető ($W = 0{,}90$; $p = 0{,}303$).

A normalitás ellenőrzésére sokszor több csoportban is szükség van. Példánkban 3 műfaj (A, B és C) filmjeinek IMDB értékelése szerepel.

```
adat <- "  mufaj  ertekeles
               A        6,2
               A        2,8
               A        2,1
               A        2,1
               A        2,8
               B        3,3
               B        5,8
               B        5,8
               B        6,8
               B        5,1
               C        5,6
```

```
               C        4,8
               C        5,3
               C        4,8
               C        3,2  "

# adatmátrix beolvasása
df13 <- read.table(textConnection(adat), header=T, sep="", dec=",")
```

```
# faktorrá alakítás
df13$mufaj <- factor(df13$mufaj)
str(df13)  # adatmátrix szerkezete
#> 'data.frame':    15 obs. of  2 variables:
#>  $ mufaj    : Factor w/ 3 levels "A","B","C": 1 1 1 1 1 2 2 2 2 2 ...
#>  $ ertekeles: num  6.2 2.8 2.1 2.1 2.8 3.3 5.8 5.8 6.8 5.1 ...
```

Több minta esetén kényelmes a `{onewaytests}` csomag `nor.test()` függvényét használni. A formula argumentumban megadjuk a numerikus vektort (`ertekeles`) és a csoportosító faktort (`mufaj`). A `method = "SW"` argumentum rögzíti, hogy Shapiro-Wilk próbát hajtunk végre. Lehetséges értékek még: `"SF"` - Shapiro-Francia próba, `"LT"`- Kolmogorov-Smirnov próba Lilliefors változata, `"AD"` - Anderson-Darling próba, `"CVM"` - Cramer-von Mises próba és `"PT"`: Pearson khí-négyzet próba.

A `plot=NULL` argumentummal az ábra megjelenítését most letiltjuk, de a megfelelő karakteres paraméter megadásával (lehetséges értékek: `“qqplot-histogram”`, `“qqplot”` és `“histogram”`) nagyon hasznos vizuális segítséget kapunk a normalitás ellenőrzéséhez.

```
# --= Shapiro-Wilk próba több csoportra =--
library(onewaytests)
nor.test(formula = ertekeles ~ mufaj, data = df13, method = "SW", plot = NULL,
    alpha = 0.05)
#>
#>   Shapiro-Wilk Normality Test (alpha = 0.05)
#> -------------------------------------------------
#>   data : ertekeles and mufaj
#>
#>   Level Statistic    p.value   Normality
#> 1     A 0.7133799 0.01314958      Reject
#> 2     B 0.9193695 0.52588040  Not reject
#> 3     C 0.8578142 0.22050645  Not reject
#> -------------------------------------------------
```

A fenti outputból szövegesen is kiolvasható a mindegyik csoporton végrehajtott Shapiro–Wilk próba eredménye (`Normality`), ami visszautasított (`Reject`) vagy megtartott (`Not reject`) lehet. A döntést a paraméterében beállított szignifikanciaszint alapján (`alpha=0.05`) hozza meg a függvény. A szokásos próbastatisztika értéket (`Statistic`) és p értéket is (`p.value`) is láthatjuk.

A fenti eredmények alapján a B és C csoportok normális eloszlásúnak tekinthetők, míg az A csoport nem tekinthető normális eloszlású sokaságnak a populációban ($p = 0{,}013$).

10.4.2. Kolmogorov-Smirnov próba

A normalitás ellenőrzésének másik klasszikus módja a Kolmogorov–Smirnov próba, amelynek Lilliefors változatát mutatjuk be. A Shapiro–Wilk próba tárgyalása során (10.4.1) bemutatott egymintás és hárommintás adatbázisokat használjuk fel. A cél továbbra is a normalitás ellenőrzése. A minimális mintaelemszám 4.

Egymintás esetben a `{DescTools}` csomag `LillieTest()` függvényét használhatjuk.

```
# --= Kolmogorov-Smirnov próba Lilliefors változat, egymintás eset =--
DescTools::LillieTest(df12$ertekeles)
#>
#>  Lilliefors (Kolmogorov-Smirnov) normality test
#>
#> data:  df12$ertekeles
#> D = 0.24409, p-value = 0.2351
```

A fenti output a KS-próba Lilliefors változatának próbastatisztika értékét (`D=`) és p értékét (`p-value`) tartalmazza. Az output értelmezése megegyezik a Shapiro–Wilk próbánál megbeszéltekkel, ez az eredmény megerősíti a nullhipotézist, azaz a minta normális eloszlásúnak tekinthető ($D = 0{,}24$; $p = 0{,}235$).

Több minta esetén most is a `{onewaytests}` csomag `nor.test()` függvényét használjuk, annyi eltéréssel a Shapiro–Wilk próbához képest, hogy a `method = "LT"` argumentumot adjuk meg.

```
# --= Kolmogorov-Smirnov próba Lilliefors változat, több csoportra =--
library(onewaytests)
nor.test(formula = ertekeles ~ mufaj, data = df13, method = "LT", plot = NULL,
    alpha = 0.05)
#>
#>   Lilliefors (Kolmogorov-Smirnov) Normality Test (alpha = 0.05)
#> ------------------------------------------------
```

```
#>   data : ertekeles and mufaj
#>
#>   Level Statistic    p.value   Normality
#> 1     A 0.3923070 0.01145988      Reject
#> 2     B 0.2323793 0.50180860  Not reject
#> 3     C 0.3258234 0.08895664  Not reject
#> ------------------------------------------------
```

A fenti output értelmezése megegyezik a Shapiro–Wilk próbánál megbeszéltekkel (10.4.1. fejezet).

10.4.3. D'Agostino-próba

A D'Agostino–próba a ferdeségi és a csúcsossági együtthatók alapján szintén a változó normalitására vonatkozó nullhipotézist teszteli. A próba végrehajtását az `{fBasics}` csomag `dagoTest()` függvényével kezdeményezhetjük. Egyetlen paramétere a numerikus adatminta.

A próbát elvégezzük az egymintás és a három csoportos adatmátrix esetén is. Mivel a minimális mintaelemszáma a próbának 20, így átváltunk a `{fosdata}` csomag `movies` adatbázisára, amely szintén tartalmaz értékelésre (`rating`) és műfajra (`genres`) vonatkozó adatot minden film esetén.

```
# adatmátrix előkészítése
df14 <- fosdata::movies[c("rating", "genres")]
df14 <- df14[df14$genres %in% c("Thriller", "Western", "Sci-Fi"), ]
df14$genres <- factor(df14$genres)
summary(df14)
#>      rating           genres
#>  Min.   :0.500   Sci-Fi  :142
#>  1st Qu.:3.000   Thriller:628
#>  Median :3.500   Western :151
#>  Mean   :3.447
#>  3rd Qu.:4.000
#>  Max.   :5.000
```

A próba egymintás változatához adjuk meg a `x=df14$rating` argumentumot.

```
# --= D'Agostino-próba, egymintás eset =--
library(fBasics)
dagoTest(x = df14$rating)
#>
#> Title:
#>  D'Agostino Normality Test
#>
#> Test Results:
#>   STATISTIC:
#>     Chi2 | Omnibus: 60.5135
#>     Z3  | Skewness: -7.6873
#>     Z4  | Kurtosis: 1.1912
#>   P VALUE:
#>     Omnibus  Test: 7.239e-14
#>     Skewness Test: 1.51e-14
#>     Kurtosis Test: 0.2336
```

A fenti output három statisztikai próba eredményét tartalmazza. Az első a normalitás vizsgálatához a ferdeségi és csúcsossági együtthatókat is használja. Ebben az esetben a próbastatisztika értékét és a p értékét a `Chi2 | Omnibus:` és az `Omnibus  Test:` sorokból olvashatjuk ki. Ehhez az eljáráshoz khí-négyzet eloszlást használ a próba.

Az output további részében külön a ferdeségre és külön a csúcsosságra vonatkozó próbák eredményét olvashatjuk ki. Mindkét próba nullhipotézise az adott együttható zérus értékét állítja, ennek teljesülése támogatja a változó normalitását. A `Z3 | Skewness` után a ferdeségi, míg a `Z4 | Kurtosis:` után a csúcsossági próbák próbastatisztika értéke olvasható. A p értékek rendre a `Skewness Test:` és `Kurtosis Test:` értékeket követik. A ferdeségre és a csúcsosságra vonatkozó próbák standard normális eloszlást használnak.

Az outputból kiolvasható, hogy a normalitásra és a ferdeségre vonatkozó próba szignifikáns ($p < 0{,}05$), míg a csúcsosságra vonatkozó próba nem szignifikáns ($p = 0{,}234$). A normalitás sérülését tehát a ferdeség okozza.

D'Agostino-próba több csoport esetén is végrehajtható, ehhez a standard `by()` függvényt használjuk.

```
# --= D'Agostino-próba, több csoportra =--
library(fBasics)
by(data = df14$rating, INDICES = df14$genres, FUN = dagoTest,
   description = "", simplify = F)
#> df14$genres: Sci-Fi
```

```
#>
#> Title:
#>  D'Agostino Normality Test
#>
#> Test Results:
#>   STATISTIC:
#>     Chi2 | Omnibus: 11.5287
#>     Z3  | Skewness: -3.3943
#>     Z4  | Kurtosis: -0.0873
#>   P VALUE:
#>     Omnibus  Test: 0.003137
#>     Skewness Test: 0.0006881
#>     Kurtosis Test: 0.9305
#>
#> ------------------------------------------------------
#> df14$genres: Thriller
#>
#> Title:
#>  D'Agostino Normality Test
#>
#> Test Results:
#>   STATISTIC:
#>     Chi2 | Omnibus: 34.4907
#>     Z3  | Skewness: -5.8171
#>     Z4  | Kurtosis: 0.8074
#>   P VALUE:
#>     Omnibus  Test: 3.239e-08
#>     Skewness Test: 5.987e-09
#>     Kurtosis Test: 0.4194
#>
#> ------------------------------------------------------
#> df14$genres: Western
#>
#> Title:
#>  D'Agostino Normality Test
#>
#> Test Results:
#>   STATISTIC:
#>     Chi2 | Omnibus: 12.5011
#>     Z3  | Skewness: -3.3836
```

```
#>     Z4  | Kurtosis: 1.0257
#>   P VALUE:
#>     Omnibus  Test: 0.001929
#>     Skewness Test: 0.0007153
#>     Kurtosis Test: 0.305
```

A fenti outputból mindhárom műfaj esetén el tudjuk dönteni, hogy az értékelés változó eloszlása eltér-e a normális eloszlástól. A normalitásvizsgálat további lehetőségeiért futtassuk a `?dagoTest` parancsot.

Összefoglalás

A normalitás vizsgálatára szintén többféle módszer áll rendelkezésre R-ben. A legismertebb a Shapiro-–Wilk-próba (`shapiro.test()`), amely kis és közepes mintákra ajánlott, illetve a Kolmogorov—Smirnov-próba Lilliefors-módosítása (`LillieTest()` a `{DescTools}` csomagból). A `{onewaytests}` csomag `nor.test()` függvénye lehetővé teszi a normalitás tesztelését több csoportra is, akár többféle eljárással. Ezen kívül létezik a D'Agostino-próba is (`dagoTest()` az `{fBasics}` csomagból), amely a ferdeségi és csúcsossági mutatók alapján teszteli a normalitást, és omnibusz próbát is tartalmaz. Mindezek mellett a grafikus diagnosztikai eszközök (hisztogram, QQ-ábra) sokszor intuitív és megbízható kiegészítői a normalitásvizsgálatnak.

Feladatok

1. Több kutató a normalitás vizsgálatának grafikus módszerét részesíti előnyben (QQ-ábra, hisztogram) a hipotézisvizsgálatokkal szemben. Keressen bizonyítékot ennek alátámasztására!
2. Generáljon normális adatokat az `rnorm()` és torzított eloszlásúakat az `rexp()` és `rchisq()` függvényekkel. Vizsgálja meg, mennyire érzékeny a Shapiro—Wilk-próba a minta méretére!

10.5. Varianciára vonatkozó próbák 🙂

> **Miről lesz szó?** Ebben a fejezetben
>
> - áttekintjük az egy minta varianciájára vonatkozó khí-négyzet próbát,
> - a két független minta varianciáját vizsgáló F-próbát, és ennek páros mintás változatát,
> - és a több minta esetén használható, a szóráshomogenitás meghatározásáért felelős Bartlett-, Levene- és Fligner–Killeen próbát.

A szórások vagy varianciák vizsgálata legtöbb esetben más próbák előfeltétel vizsgálata kapcsán kerül előtérbe. Ez az ún. szóráshomogenitás vagy homoszkedaszticitás ellenőrzése. Vannak olyan esetek is, amikor kimondottan a szórás vizsgálata áll a kutatás középpontjában.

A szóráshomogenitás vizsgálata során minden esetben a nullhipotézis az, hogy a két vagy több csoportban a vizsgált változó szórása nem tér el egymástól. A szignifikáns p érték ($p < 0{,}05$) arra utal, hogy a szóráshomogenitási feltétel nem áll fenn.

10.5.1. Egymintás khí-négyzet próba a varianciára

Egyetlen kvantitatív minta esetén a variancia (vagy szórás) összehasonlítható egy hipotetikus értékkel. A példánkban 6 tanuló dolgozatának pontszáma alapján a hipotetikus 10 szórástól való esetleges eltérést vizsgáljuk.

Olvassuk be az adatbázisunkat!

```
adat <- "  pontszam
                43
                25
                53
                25
                34
                45  "

# adatmátrix beolvasása
df15 <- read.table(textConnection(adat), header=T, sep="")
str(df15) # adatmátrix szerkezete
#> 'data.frame':    6 obs. of  1 variable:
#>  $ pontszam: int  43 25 53 25 34 45
```

A `{TeachingDemos}` csomag `sigma.test()` függvényében a hipotetikus varianciát (mint esetünkben: `sigmasq=100`), vagy a hipotetikus szórást kell megadnunk (esetünkben ez `sigma=10` lenne).

```
# --= egymintás khí-négyzet próba a varianciára =--
library(TeachingDemos)
sigma.test(x = df15$pontszam, sigmasq = 100, conf.level = 0.95)
#>
#>  One sample Chi-squared test for variance
#>
#> data:  df15$pontszam
#> X-squared = 6.515, df = 5, p-value = 0.5186
#> alternative hypothesis: true variance is not equal to 100
#> 95 percent confidence interval:
#>   50.76952 783.79559
#> sample estimates:
#> var of df15$pontszam
#>                130.3
```

A fenti outputot a 10.2.1. fejezetben megismertek alapján tudjuk értelmezni, azzal a különbséggel, hogy ez a próbastatisztika a khí-négyzet eloszláson alapul, és várható érték helyett a varianciaára vonatkozik az intervallum- és a pontbecsléseket látjuk. A próba nem szignifikáns ($\chi^2(5) = 6{,}52$; $p = 0{,}519$), azaz a szórás eltérése a hipotetikus 10-es szórástól nem bizonyítható.

10.5.2. F-próba

Két független minta esetén a populációbeli varianciák (vagy szórások) egyezését vizsgálhatjuk F-próbával. A példánkban matematika és angol dolgozatok eredményének szórását hasonlítjuk össze. Mindkét dolgozat maximum 20 pontos lehet, és összesen 8 tanulót vontunk be a vizsgálatba.

Olvassuk be az adatmátrixot!

```
adat <- "  tantargy   pontszam
              matek         14
              matek         19
              matek         13
              matek         20
              angol         11
```

```
              angol          9
              angol         18
              angol         20  "

# adatmátrix beolvasása
df16 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df16$tantargy <- factor(df16$tantargy)
str(df16)  # adatmátrix szerkezete
#> 'data.frame':    8 obs. of  2 variables:
#>  $ tantargy: Factor w/ 2 levels "angol","matek": 2 2 2 2 1 1 1 1
#>  $ pontszam: int  14 19 13 20 11 9 18 20
```

F-próba a `var.test()` függvénnyel hajtható végre, a `formula=` tartalmazza a numerikus vektort (`pontszam`) és a két szintű faktort (`tantargy`).

```
# --= F-próba két független mintára =--
var.test(formula = pontszam ~ tantargy, data = df16, conf.level = 0.95)
#>
#>  F test to compare two variances
#>
#> data:  pontszam by tantargy
#> F = 2.2973, num df = 3, denom df = 3, p-value = 0.5123
#> alternative hypothesis: true ratio of variances is not equal to 1
#> 95 percent confidence interval:
#>   0.1487966 35.4683920
#> sample estimates:
#> ratio of variances
#>           2.297297
```

Az F-próba F eloszlást használ a p érték meghatározásához. A próbastatisztika értéke (`F=`), a két szabadsági fok (`num df=` és `denom df=`) és a p érték (`p-value=`) kiolvasható az outputból. A próba nem szignifikáns ($p = 0{,}512$). Az `alternative hypothesis:` sorban olvashatjuk a próba pontos ellenhipotézisét, amely a két változó varianciájának az arányáról azt állítja, hogy nem egyelő 1-eggyel, vagyis nem egyeznek meg. Az intervallum- és a pontbecslés is a varianciák hányadosára vonatkozik.

A példa adatbázisa szerint a matematika és angol dolgozatok varianciája nem különbözik egymástól szignifikánsan ($F(3{,}3) = 2{,}297$; $p = 0{,}512$).

10.5.3. Pitman–Morgan próba

Páros minta esetében is vizsgálhatjuk a két kondícióban mért változó varianciájának az egyezőségét. Példánkban azt a hipotézist vizsgáljuk, hogy egy adott tréning előtt és a tréning után az intelligencia varianciája megegyezik egymással! Összesen 12 személyt vontunk be a vizsgálatba.

Széles adatbázissal fogunk dolgozni. Olvassuk be az adatokat!

```
adat <- "  szemely   IQ1   IQ2
                a   127   137
                b    98   108
                c   105   115
                d    83    93
                e   113   143
                f   133   123
                g   127   143
                h   133   113
                i   144   153
                j    90   100
                k   107   117
                l    98   108  "

# adatmátrix beolvasása
df17 <- read.table(textConnection(adat), header = T, sep = "")
```

```
# faktorrá alakítás
df17$szemely <- factor(df17$szemely)
str(df17)  # adatmátrix szerkezete
#> 'data.frame':    12 obs. of  3 variables:
#>  $ szemely: Factor w/ 12 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
#>  $ IQ1    : int  127 98 105 83 113 133 127 133 144 90 ...
#>  $ IQ2    : int  137 108 115 93 143 123 143 113 153 100 ...
```

A `{PairedData}` csomag `Var.test()` függvénye hajtja végre a párosított mintán alapuló Pitman–Morgan próbát. Az `x=` és `y=` argumentumában megadjuk a két mintát, és gondoskodunk a párosított vizsgálatról a `paired=T` megadásával.

```
# --= Pitman-Morgan próba páros mintára =--
library(PairedData)
Var.test(x = df17$IQ1, y = df17$IQ2, paired = T, conf.level = 0.95)
#>
#>  Paired Pitman-Morgan test
#>
#> data:  x and y
#> t = 0.12881, df = 10, p-value = 0.9001
#> alternative hypothesis: true ratio of variances is not equal to 1
#> 95 percent confidence interval:
#>  0.4551047 2.4274573
#> sample estimates:
#> variance of x variance of y
#>      375.6061      357.3561
```

A fenti output értelmezése megegyezik az F-próba esetén megbeszéltekkel (10.5.2. fejezet), azzal a kivétellel, hogy a pontbecslés a két minta egyenkénti varianciáját tartalmazza. Az eredmények szerint a két minta varianciája nem különbözik egymástól szignifikánsan ($t(10) = 0{,}129$; $p = 0{,}900$).

10.5.4. Bartlett-próba

A Bartlett-próba az F-próba általánosításának tekinthető abban az értelemben, hogy kettőnél több független mintát is vizsgálhatunk segítségével. Továbbra is a varianciák (szórások) azonosságát vizsgáljuk az egyes csoportok között.

A példánkban matematika, angol és fizika dolgozatok eredményének szórását hasonlítjuk össze. Mindhárom dolgozat maximum 20 pontos. Már 12 tanulót vontunk be a vizsgálatba.

Olvassuk be az adatmátrixot!

```
adat <- "  tantargy   pontszam
              matek         14
              matek         19
              matek         13
              matek         20
              angol         11
              angol          9
              angol         18
              angol         20
```

```
             fizika         12
             fizika         13
             fizika         11
             fizika         12  "

# adatmátrix beolvasása
df18 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df18$tantargy <- factor(df18$tantargy)
str(df18)  # adatmátrix szerkezete
#> 'data.frame':    12 obs. of  2 variables:
#>  $ tantargy: Factor w/ 3 levels "angol","fizika",..: 3 3 3 3 1 1 1 1 2 2 ...
#>  $ pontszam: int  14 19 13 20 11 9 18 20 12 13 ...
```

Bartlett-próba a `bartlett.test()` függvénnyel hajtható végre, a `formula=` tartalmazza a numerikus vektort (`pontszam`) és a több szintű faktort (`tantargy`). A faktor lehet két szintű is, de tipikusan három vagy annál több különböző értéket tartalmaz.

```
# --= Bartlett-próba =--
bartlett.test(formula = pontszam ~ tantargy, data = df18)
#>
#>  Bartlett test of homogeneity of variances
#>
#> data:  pontszam by tantargy
#> Bartlett's K-squared = 6.3188, df = 2, p-value = 0.04245
```

A Bartlett-próba khí-négyzet eloszlást használ, az outputban közli a próbastatisztika értékét (`K-squared=`), a szabadsági fokok számát (`df=`) és a p értéket (`p-value=`).

A próba szignifikáns ($\chi^2(2) = 6{,}319$; $p = 0{,}042$), azaz a tantárgyak varianciája nem egyezik meg egymással. A próba érzékeny a normális eloszlástól való eltérésre, így ha a normalitás feltétele nem teljesül, akkor a Levene- vagy Fligner–Killeen-próbát érdemes használni.

10.5.5. Levene-próba

A Levene-próba és annak robusztusabb Brown–Forsythe változata is a Bartlett-próbához hasonlóan több csoportban a varianciák (szórások) egyezését vizsgálja. A próba bemutatásához így az előző, 10.5.4. fejezet adatbázisát is használhatjuk.

A Levene-próba végrehajtásához a `{DescTools}` csomag `LeveneTest()` függvényét használjuk. A formula argumentumot a szokásos numerikus vektor (`pontszam`), többszintű faktor (`tantargy`) sorrendben töltjük. A próba eredeti változatához a `center=mean`, a robusztusabb Brown–Forsythe változathoz a `center=median` argumentumot használjuk.

```
# --= Levene-próba (eredeti változat: center=mean) =--
library(DescTools)
LeveneTest(formula = pontszam ~ tantargy, data = df18, center = mean)
#> Levene's Test for Homogeneity of Variance (center = mean)
#>       Df F value    Pr(>F)
#> group  2    24.5 0.0002284 ***
#>        9
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# --= robusztus Levene-próba (Brown-Forsythe változat: center=median) =--
library(DescTools)
LeveneTest(formula = pontszam ~ tantargy, data = df18, center = median)
#> Levene's Test for Homogeneity of Variance (center = median)
#>       Df F value    Pr(>F)
#> group  2    24.5 0.0002284 ***
#>        9
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A Levene-próba outputjából (változattól függetlenül) a szabadsági fokokat (`Df`), a próbastatisztika értékét (`F value`), és a p értéket (`Pr(>F)`) olvashatjuk ki. Mindkét változatban szignifikáns a próba. Láthatjuk, hogy a Levene-próba F eloszlást használ. Mindkét változat alapján a tantárgyak varianciája nem egyezik meg egymással ($F(2{,}9) = 24{,}5$; $p < 0{,}001$).

10.5.6. Fligner–Killeen próba

A Fligner–Killeen próba egy másik olyan teszt, amely a változók szóráshomogenitásának ellenőrzésére szolgál, és a Levene-próbához hasonlóan ellenálló az adatok normalitástól való eltérésével szemben.

A próba végrehajtásához használt `fligner.test()` paraméterezése teljesen megegyezik a Bartlett-próba és a Levene-próba példáival. Az adatbázis szerkezete és előkészítése is azonos (10.5.4. fejezet).

```r
# --= Fligner-Killeen próba =--
fligner.test(formula = pontszam ~ tantargy, data = df18)
#>
#>  Fligner-Killeen test of homogeneity of variances
#>
#> data:  pontszam by tantargy
#> Fligner-Killeen:med chi-squared = 8.5557, df = 2, p-value =
#> 0.01387
```

A próba khí-négyzet eloszlást használ (`?fligner.test`). Az outputból kiolvasható a próbastatisztika értéke (`chi-squared=`), a szabadsági fok (`df=`) és a p érték (`p-value=`). Ez a próba is szignifikáns ($\chi^2(2) = 8{,}556$; $p = 0{,}014$), azaz a tantárgyak pontszámainak varianciája nem egyezik meg egymással.

Összefoglalás

A varianciára vonatkozó próbák a t-próbák és az ANOVA előfeltételeinek ellenőrzésére szolgálnak, de önálló kutatási kérdések esetén is fontosak lehetnek. Egymintás esetben a variancia vagy szórás összehasonlítható egy előre megadott hipotetikus értékkel, amelyhez a `sigma.test()` függvényt használhatjuk a `{TeachingDemos}` csomagból. Ha két független minta szórásának egyezőségét szeretnénk vizsgálni, az F-próba a megfelelő választás, amely a `var.test()` függvénnyel hajtható végre. Páros minták esetében a Pitman–Morgan-próba használható, amely a `{PairedData}` csomag `Var.test()` függvényében a `paired = TRUE` argumentummal aktiválható. Ha három vagy több csoport varianciáját kívánjuk összehasonlítani, többféle próbát is választhatunk. A klasszikus Bartlett-próba (`bartlett.test()`) a varianciák azonosságát teszteli, azonban érzékeny a normalitás sérülésére. Ezért robusztusabb alternatívát kínál a Levene-próba, amely a `LeveneTest()` függvénnyel (a `{DescTools}` csomagból) végezhető el. Az eredeti Levene-próba a csoportok átlagaihoz, míg Brown-–Forsythe változata a mediánokhoz viszonyítva vizsgálja a szórásokat (`center = "mean"` vagy `center = "median"`). A normalitás megsértésével szemben talán a legellenállóbb a Fligner–Killeen-próba (`fligner.test()`), amely szintén a varianciák homogenitását teszteli, de nem igényli a normális eloszlás feltételezését.

Feladatok

1. A szóráshomogenitás (homoszkedaszticitás) vizsgálatára nem csak statisztikai próbák, hanem diagnosztikus ábrák is léteznek, amelyek vizuálisan segítik a szórásazonosság feltételezésének ellenőrzését. Melyek ezek?

10.6. Valószínűség próbái 🙂

> **Miről lesz szó?** Ebben a fejezetben
>
> - áttekintjük a populációbeli arányokra vonatkozó klasszikus próbákat,
> - az illeszkedésvizsgálat és a kapcsolatvizsgálat gyakori eseteit.

10.6.1. Illeszkedésvizsgálat egy valószínűségre

Amennyiben egy nominális változó egyik kategóriájának populációbeli arányára kérdezünk rá, akkor használhatjuk a binomiális próbát. Példánkban a hallgatók nemét vizsgáljuk a pszichológia szakon. Feltételezésünk, hogy nők aránya 0,8. Az adatok több formában is rendelkezésre állhatnak.

A próba végrehajtásához 3 szám szükséges, amely megfeleltethető a binomiális próbáért felelős `binom.test()` három paraméterének. Ezeket három objektumban is rögzíthetjük, nincs szükségünk adatmátrixra. Nagyobb mintaelemszám esetén a `prop.test()` függvény használata ajánlott, amely a normális eloszlásra épít.

```
x =  84    # nők száma a mintában
n = 124    # összes mintaelemszám (férfiak+nők)
p = 0.8    # a nők feltételezett aránya (hipotetikus valószínűség)

# -- = egzakt binomiális próba =--
binom.test(x = x, n = n, p = p, conf.level = 0.95)
#>
#>  Exact binomial test
#>
#> data:  x and n
#> number of successes = 84, number of trials = 124, p-value =
#> 0.001491
#> alternative hypothesis: true probability of success is not equal to 0.8
#> 95 percent confidence interval:
#>  0.5875885 0.7585492
#> sample estimates:
#> probability of success
#>              0.6774194
```

```
# --= egymintás arányteszt, Yates-féle korrekció nélkül =--
prop.test(x = x, n = n, p = p, correct = F, conf.level = 0.95)
#>
#>  1-sample proportions test without continuity correction
#>
#> data:  x out of n, null probability p
#> X-squared = 11.645, df = 1, p-value = 0.0006437
#> alternative hypothesis: true p is not equal to 0.8
#> 95 percent confidence interval:
#>  0.5908803 0.7532960
#> sample estimates:
#>         p
#> 0.6774194
```

A `binom.test()` outputja megismétli a bemenő paraméterek értékét, a kedvező esetek számát, ami esetünkben a nők száma a mintában: 84 (`number of successes`). Megadja az összes eset számát, vagyis mintaelemszámot, ami a esetünkben a nők és a férfiak száma: 124 (`number of trials`). Ez alapján egy megfigyelt, mintabeli arány is számításra kerül: 0,68 (`probability of success`). Az `alternative hypothesis:` sorban olvasható a bemenő adatként szintén megadott, feltételezett nő-arány: 0,8. Az elemzés eredménye a p érték (`p-value`) esetünkben $p = 0{,}001$, ami a nullhipotézis elvetését jelenti, azaz a nők aránya eltér a feltételezett 0,8-tól. A 95%-os konfidenciaintervallum (`95 percent confidence interval:`) is kiolvasható az outputból: $95\%CI\ :\ [0{,}59; 0{,}76]$.

A normális eloszlásra építő `prop.test()` függvény outputja is hasonlóan néz ki, és a következtetésünk is hasonló. Azonban a két próba között van egy lényeges különbség. A `binom.test()` függvény pontos binomiális próbát végez, míg a `prop.test()` függvény egy közelítő normális eloszlásra épít. Ezért a két próba eredményei eltérhetnek egymástól, és ez a különbség nagyobb minták esetén jelentkezik. A `prop.test()` függvény outputja tartalmazza a próbastatisztika értékét (`X-squared=`), a szabadsági fokok számát (`df=`) és a p értéket (`p-value=`). A konfidencia intervallum (`95 percent confidence interval:`) és a mintabeli arány (`sample estimates:`) is kiolvasható az outputból.A nők aránya eltér a feltételezett 0,8-tól ($\chi^2(1) = 11{,}65; p < 0{,}001$),A konfidenciaintervallum $95\%CI\ :\ [0{,}59; 0{,}76]$ és a mintabeli arány 0,68.

Az adataink természetes formája az adatmátrix. Vizsgáljuk továbbra is a nemek arányát egy elképzelt kisebb adatbázison.

```
adat <- "  nem
            nő
            nő
```

```
            nő
            nő
            nő
            nő
            nő
         férfi
         férfi
         férfi  "

# adatmátrix beolvasása
df19 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df19$nem <- factor(df19$nem)
str(df19)  # adatmátrix szerkezete
#> 'data.frame':    10 obs. of  1 variable:
#>  $ nem: Factor w/ 2 levels "férfi","nő": 2 2 2 2 2 2 2 1 1 1
```

Ebben az esetben a három bemenő paramétert az adatmátrixból olvassuk ki

```
x = sum(df19$nem %in% "nő")  # nők száma a mintában
n = sum(!is.na(df19$nem))    # összes mintaelemszám (férfiak+nők)
p = 0.8    # a nők feltételezett aránya (hipotetikus valószínűség)

# --= egzakt binomiális próba =--
binom.test(x = x, n = n, p = p, conf.level = 0.95)
#>
#>  Exact binomial test
#>
#> data:  x and n
#> number of successes = 7, number of trials = 10, p-value =
#> 0.4296
#> alternative hypothesis: true probability of success is not equal to 0.8
#> 95 percent confidence interval:
#>  0.3475471 0.9332605
#> sample estimates:
#> probability of success
#>                    0.7
```

```r
# --= egymintás arányteszt, Yates-féle korrekció nélkül =--
prop.test(x = x, n = n, p = p, correct = F, conf.level = 0.95)
#>
#>  1-sample proportions test without continuity correction
#>
#> data:  x out of n, null probability p
#> X-squared = 0.625, df = 1, p-value = 0.4292
#> alternative hypothesis: true p is not equal to 0.8
#> 95 percent confidence interval:
#>  0.3967781 0.8922087
#> sample estimates:
#>   p
#> 0.7
```

Az eredmények értelmezése hasonlóan történik. A binomiális próbát végeztünk annak meghatározására, hogy a minta sikerességi aránya szignifikánsan eltér-e a feltételezett 0,8-es aránytól. A minta 10 kísérletből állt, melyek közül 7 volt sikeres, és 3 sikertelen. Az eredmények nem jeleztek szignifikáns eltérést a feltételezett aránytól $p = 0{,}430$. A megfigyelt sikerességi arány 0,7 volt (7 a 10-ből), ami nem különbözik szignifikánsan a feltételezett 0,8-as aránytól. A sikertelenségek aránya 0,3 (3 a 10-ből). Ennek alapján a nullhipotézist – miszerint a siker valószínűsége 0,8 – nem utasíthatjuk el.

A `prop.test()` függvény outputja is hasonlóan néz ki, és a következtetésünk is hasonló ($\chi^2(1) = 0{,}63; p = 0{,}429$). A konfidenciaintervallum $95\%CI : [0{,}40; 0{,}89]$ és a mintabeli arány 0,70.

10.6.2. Illeszkedésvizsgálat több valószínűségre

Abban az esetben, ha a nominális változónk kettő vagy több szintjére van elképzelésünk a populációbeli valószínűségekre vonatkozóan, akkor az egzakt multinominális-próbát használhatjuk.

Például, ha azt gondoljuk, hogy a három szinttel mért dohányzási szokás változó ma a fiatalok körében így alakul: naponta dohányzik: 0,35, alkalmi dohányos: 0,05, nem dohányzik: 0,6.

A bemenő adatok itt is alapvetően numerikus skalárok. Az `{RVAideMemoire}` csomag `multinomial.test()` függvényét használjuk. A populációbeli valószínűségekre konfidenciaintervallumot is számíthatunk a `{DescTools}` csomag `MultinomCI()` függvényével. Nagyobb mintaelemszám esetén itt is használhatjuk a normális eloszlásra építő `prop.test()` függvényt.

```r
naponta_dohanyzik <- 92  # naponta dohányzók száma a mintában
alkalmi_dohanyos  <- 32  # alkalmai dohányosok száma a mintában
nem_dohanyzik     <- 149 # nem dohányzók száma a mintában

# a három csoport létszám numerikus vektora
x <- c(naponta_dohanyzik, alkalmi_dohanyos, nem_dohanyzik)
p <- c(0.35, 0.05, 0.6) # hipotetikus eloszlás (3 valószínűség)

# --= egzakt multinominális-próba =--
library(RVAideMemoire)
RVAideMemoire::multinomial.test(x = x, p = p)
#>
#>  Exact multinomial test
#>
#> data:  x
#> p-value = 4e-05
library(DescTools)
MultinomCI(x, conf.level=0.95, method="sisonglaz")
#>            est     lwr.ci    upr.ci
#> [1,] 0.3369963 0.27838828 0.4006901
#> [2,] 0.1172161 0.05860806 0.1809099
#> [3,] 0.5457875 0.48717949 0.6094813
```

```r
# --= többmintás arányteszt, Yates-féle korrekció nélkül =--
prop.test(x = x, n = rep(sum(x), 3), p = p)
#>
#>  3-sample test for given proportions without continuity
#>  correction
#>
#> data:  x out of rep(sum(x), 3), null probabilities p
#> X-squared = 29.513, df = 3, p-value = 1.747e-06
#> alternative hypothesis: two.sided
#> null values:
#> prop 1 prop 2 prop 3
#>   0.35   0.05   0.60
#> sample estimates:
#>    prop 1    prop 2    prop 3
#> 0.3369963 0.1172161 0.5457875
```

A `multinomial.test()` outputja alapján azt mondhatjuk, hogy a három csoport arányai nem

egyeznek meg a hipotetikus eloszlással. A p érték ($p < 0{,}001$). A konfidencia intervallumokat is tartalmazó output alapján a naponta dohányzók aránya a mintában 0,34, a 95%-os megbízhatósági intervallum alsó határa 0,28, míg a felső határ 0,40 ($95\%CI : [0{,}28; 0{,}40]$). Az alkalmi dohányzók aránya a mintában 0,11, a 95%-os megbízhatósági intervallum: $95\%CI : [0{,}06; 0{,}18]$. A nem dohányzók aránya a mintában 0,55, a 95%-os megbízhatósági intervallum: $95\%CI : [0{,}49; 0{,}61]$.

A `prop.test()` outputja hasonló eredményre vezet, azaz a három csoport arányai nem egyeznek meg a hipotetikus eloszlással ($\chi^2(2) = 29{,}51; p < 0{,}001$).

A szignifikáns p érték miatt érdemes megvizsgálni, melyik csoport esetén van eltérés a hipotetikus aránytól. A `multinomial.theo.multcomp()` függvény egzakt binomiális próbát végez az egyes csoportokra, és a p értékeket korrigálja a többszörös összehasonlítások miatt.

```
# Egzakt multinominális-próba - utóvizsgálat
# - Páronkénti egzakt binomiális próba
multinomial.theo.multcomp(x = x, p = p, p.method = "fdr")
#>
#>         Pairwise comparisons using exact binomial tests
#>
#> data:  x and p
#>
#>  observed expected   P-value
#>        92    95.55 7.035e-01
#>        32    13.65 2.742e-05 ***
#>       149   163.80 1.097e-01
#>
#> P value adjustment method: fdr
```

A fenti eredmények azt mutatják, hogy a naponta dohányzók aránya nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}704$), míg az alkalmi dohányosok aránya szignifikánsan eltér a hipotetikus eloszlástól ($p < 0{,}001$). A nem dohányzók aránya szintén nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}120$).

Több valószínűségre vonatkozó próbát adatmátrixból is építhetünk.

```
adat <- "  dohanyzasi_szokas
             'nem dohányzik'
             'nem dohányzik'
             'nem dohányzik'
             'nem dohányzik'
             'nem dohányzik'
```

```
         'naponta dohányzik'
         'naponta dohányzik'
         'naponta dohányzik'
          'alkalmi dohányos'  "

# adatmátrix beolvasása
df20 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítás
df20$dohanyzasi_szokas <- factor(df20$dohanyzasi_szokas)
str(df20)  # adatmátrix szerkezete
#> 'data.frame':    9 obs. of  1 variable:
#>  $ dohanyzasi_szokas: Factor w/ 3 levels "alkalmi dohányos",..: 3 3 3 3 3 2 2 2 1
```

Ebben az esetben a `multinomial.test()` és a `prop.test()` függvény bemeneti adatait az adatmátrixból olvassuk ki.

```
# naponta dohányzók száma a mintában
naponta_dohanyzik <- sum(df20$dohanyzasi_szokas %in% "naponta dohányzik")
# alkalmai dohányosok száma a mintában
alkalmi_dohanyos  <- sum(df20$dohanyzasi_szokas %in% "alkalmi dohányos")
# nem dohányzók száma a mintában
nem_dohanyzik     <- sum(df20$dohanyzasi_szokas %in% "nem dohányzik")

# a három csoport létszám numerikus vektora
x <- c(naponta_dohanyzik, alkalmi_dohanyos, nem_dohanyzik)
p <- c(0.35, 0.05, 0.6) # hipotetikus eloszlás (3 valószínűség)

# --= egzakt multinominális-próba =--
library(RVAideMemoire)
RVAideMemoire::multinomial.test(x = x, p = p)
#>
#>  Exact multinomial test
#>
#> data:  x
#> p-value = 0.4757
library(DescTools)
MultinomCI(x, conf.level=0.95, method="sisonglaz")
#>           est    lwr.ci    upr.ci
#> [1,] 0.3333333 0.1111111 0.7111421
```

```
#> [2,] 0.1111111 0.0000000 0.4889199
#> [3,] 0.5555556 0.3333333 0.9333643
```

```
# --= többmintás arányteszt, Yates-féle korrekció nélkül =--
prop.test(x = x, n = rep(sum(x), 3), p = p)
#>
#>  3-sample test for given proportions without continuity
#>  correction
#>
#> data:  x out of rep(sum(x), 3), null probabilities p
#> X-squared = 0.79267, df = 3, p-value = 0.8512
#> alternative hypothesis: two.sided
#> null values:
#> prop 1 prop 2 prop 3
#>   0.35   0.05   0.60
#> sample estimates:
#>    prop 1    prop 2    prop 3
#> 0.3333333 0.1111111 0.5555556
```

Látjuk a fenti esetben nincs szükség utóvizsgálatra, a p érték $p = 0{,}476$ és $p = 0{,}851$, azaz a három csoport arányai megegyeznek a hipotetikus eloszlással. Az outputból a konfidencia intervallumok is kiolvashatók.

Az általános illeszkedésvizsgálatra eddig két megközelítést láttunk, de még legalább 2 lehetőség rendelkezésre áll. A következő listában összefoglaljuk a legfontosabbakat:

- Egzakt multinomiális próba: `RVAideMemoire::multinomial.test()`
- Arányteszt: `prop.test()`
- Khí-négyzet próba illeszkedésvizsgálatra: `chisq.test()`
- G-próba illeszkedésvizsgálatra: `G.test()`

Tekintsük át röviden a khí-négyzet próbát és a G-próbát. A bemeneteket most konstans értékekből olvassuk ki, de természetesen az adatmátrixból is kiolvashatók a számok, amint korábban már két esetben is láttuk.

```
naponta_dohanyzik <- 92  # naponta dohányzók száma a mintában
alkalmi_dohanyos  <- 32  # alkalmai dohányosok száma a mintában
nem_dohanyzik     <- 149 # nem dohányzók száma a mintában

# a három csoport létszám numerikus vektora
x <- c(naponta_dohanyzik, alkalmi_dohanyos, nem_dohanyzik)
```

```r
p <- c(0.35, 0.05, 0.6) # hipotetikus eloszlás (3 valószínűség)

# --= khí-négyzet próba illeszkedésvizsgálatra =--
chisq.test(x = x, p = p)
#>
#>  Chi-squared test for given probabilities
#>
#> data:  x
#> X-squared = 26.137, df = 2, p-value = 2.11e-06
```

```r
# --= G-próba illeszkedésvizsgálatra =--
library(RVAideMemoire)
G.test(x = x, p = p)
#>
#>  G-test for given probabilities
#>
#> data:  x
#> G = 19.341, df = 2, p-value = 6.313e-05
```

A khí-négyzet próba outputja alapján azt mondhatjuk, hogy a három csoport arányai nem egyeznek meg a hipotetikus eloszlással ($\chi^2(2) = 26{,}14$; $p < 0{,}001$). A G-próba eredménye alapján szintén azt mondhatjuk, hogy a három csoport arányai nem egyeznek meg a hipotetikus eloszlással ($G(2) = 19{,}34$; $p < 0{,}001$).

Érdemes megvizsgálni melyik csoport esetén van eltérés a hipotetikus aránytól. A `chisq.theo.multcomp()` függvényt használhatjuk ezekre az összehasonlításokra.

```r
# Khí-négyzet próba illeszkedésvizsgálatra - utóvizsgálat
# - Páronkénti összehasonlítások adott valószínűségre
library(RVAideMemoire)
chisq.theo.multcomp(x = x, p = p, p.method = "fdr")
#>
#>         Pairwise comparisons using chi-squared tests
#>
#> data:  x and p
#>
#>  observed expected     Chi  Pr(>Chi)
#>        92    95.55  0.2029 6.524e-01
#>        32    13.65 25.9666 1.042e-06 ***
#>       149   163.80  3.3431 1.012e-01
```

```
#>
#> P value adjustment method: fdr
```

```
# G-próba illeszkedésvizsgálatra - Páronkénti összehasonlítások
library(RVAideMemoire)
G.theo.multcomp(x = x, p = p, p.method = "fdr")
#>
#>         Pairwise comparisons using G-tests
#>
#> data:  x and p
#>
#>  observed expected       G    Pr(>G)
#>        92    95.55  0.2041 0.6514237
#>        32    13.65 19.1579 0.0000361 ***
#>       149   163.80  3.3004 0.1038968
#>
#> P value adjustment method: fdr
```

Az khí-négyzet próba utóvizsgálata alapján azt mondhatjuk, hogy a naponta dohányzók aránya nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}652$), míg az alkalmi dohányosok aránya szignifikánsan eltér a hipotetikus eloszlástól ($p < 0{,}001$). A nem dohányzók aránya szintén nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}101$). A G-próba utóvizsgálata hasonló eredményre vezet: a naponta dohányzók aránya nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}651$), míg az alkalmi dohányosok aránya szignifikánsan eltér a hipotetikus eloszlástól ($p < 0{,}001$). A nem dohányzók aránya szintén nem különbözik szignifikánsan a hipotetikus eloszlástól ($p = 0{,}104$).

10.6.3. Próbák kapcsolatvizsgálatra

Amennyiben két nominális változónk van, akkor a két változó kapcsolatát (vagy függetlenségét) khí-négyzet próbával is vizsgálhatjuk.

Az adatok összesített vagy nyers formában is rendelkezésre állhatnak. Tegyük fel, hogy az eddigi, fiatalok dohányzásával kapcsolatos információ mellett a középkorúak és idősek esetében is rendelkezésre állnak adatok. Vizsgáljuk meg, hogy van-e eltérés a három életkort jellemző dohányzási szokásában.

Kezdjük az összesített adatokkal. A jelen 3x3-as elrendezésnek megfelelő 9 gyakoriságí adat, még így is sok formában állhat rendelkezésre, de a legjobb, ha mátrix adatszerkezetbe visszük be őket.

```
adat <- " korosztaly naponta_dohanyzik alkalmi_dohanyos nem_dohanyzik
              fiatal                92               32           149
           kozepkoru                35               43            97
                idos                21               65            87
"
# adatmátrix beolvasása
df21 <- read.table(textConnection(adat), header=T, sep="",
                   row.names = 1)
dm21 <- as.matrix(df21)  # mátrixszá konvertálás
dm21
#>           naponta_dohanyzik alkalmi_dohanyos nem_dohanyzik
#> fiatal                   92               32           149
#> kozepkoru                35               43            97
#> idos                     21               65            87
```

```
# --= khí-négyzet próba mátrixból =--
chisq.test(x = dm21)
#>
#>  Pearson's Chi-squared test
#>
#> data:  dm21
#> X-squared = 54.502, df = 4, p-value = 4.131e-11
```

A próba eredménye alapján a három életkori csoport dohányzási szokása nem egyezik meg egymással ($\chi^2(4) = 54{,}50$; $p < 0{,}001$). Most megvizsgáljuk, hogy a három életkori csoport közül melyik csoportok között van eltérés. A `pairwiseNominalIndependence()` függvény használatával páronkénti összehasonlításokat végezhetünk. A `compare` argumentumban megadhatjuk, hogy a sorok vagy az oszlopok között szeretnénk-e páronkénti összehasonlítást végezni. Az `fisher`, `gtest` és `chisq` argumentumokban megadhatjuk, hogy milyen próbát szeretnénk végezni. Az `method` argumentumban megadhatjuk a p értékek korrekciós módszerét.

```
# Khí-négyzet próba - utóvizsgálat
# - Páronkénti összehasonlítások
library(rcompanion)
pairwiseNominalIndependence(x = dm21, compare = "row",
                            fisher = F, gtest = F, chisq = T,
                            method = "fdr", digits = 3)
#>           Comparison  p.Chisq p.adj.Chisq
#> 1 fiatal : kozepkoru 1.51e-04    2.27e-04
```

```
#> 2      fiatal : idos 4.32e-12    1.30e-11
#> 3   kozepkoru : idos 1.42e-02    1.42e-02
```

Az utóvizsgálat világossá tette, hogy minden csoport-pár esetén szignifikáns különbség van: a fiatalok és a középkorúak között $p < 0{,}0015$, a fiatalok és az idősek között $p < 0{,}001$, míg a középkorúak és az idősek között $p = 0{,}014$.

A khí-négyzet próba mellett a G-próba is használható kapcsolatvizsgálatra.

```
# --= G-próba kapcsolatvizsgálatra =--
library(RVAideMemoire)
G.test(x = dm21)
#>
#>  G-test
#>
#> data:  dm21
#> G = 55.602, df = 4, p-value = 2.43e-11
```

Az outputbók kiolvasható, hogy a három életkori csoport dohányzási szokása nem egyezik meg egymással ($G(4) = 55{,}60$; $p < 0{,}001$). Most megvizsgáljuk, hogy a három életkori csoport közül melyik csoportok között van eltérés. A `pairwiseNominalIndependence()` függvény használatával páronkénti összehasonlításokat végezhetünk.

```
# G-próba - utóvizsgálat - Páronkénti összehasonlítások
library(rcompanion)
pairwiseNominalIndependence(x = dm21, compare = "row", fisher = F, gtest = T,
    chisq = F, method = "fdr", digits = 3)
#>           Comparison  p.Gtest p.adj.Gtest
#> 1 fiatal : kozepkoru 1.51e-04    2.27e-04
#> 2      fiatal : idos 2.46e-12    7.38e-12
#> 3   kozepkoru : idos 1.37e-02    1.37e-02
```

Az utóvizsgálat alapján minden csoport-pár esetén szignifikáns különbséget talált: a fiatalok és a középkorúak között $p < 0{,}001$, a fiatalok és az idősek között $p < 0{,}001$, míg a középkorúak és az idősek között $p = 0{,}014$.

A harmadik itt bemutatott lehetőség kapcsolatvizsgálatra a Fisher-próba. A Fisher-próba a kisebb mintaszámú csoportok esetén is használható, és a p értékek korrekciójára is van lehetőség. A `fisher.test()` függvény használatával végezhetjük el a Fisher-próbát. A `simulate.p.value` argumentumban megadhatjuk, hogy szimulációval szeretnénk-e kiszámítani a p értéket. Az `conf.level` argumentumban megadhatjuk a konfidenciaintervallum szintjét.

```
# --= Fisher egzakt próba =--
fisher.test(dm21, conf.level = 0.95, simulate.p.value = T)
#>
#>  Fisher's Exact Test for Count Data with simulated p-value
#>  (based on 2000 replicates)
#>
#> data:  dm21
#> p-value = 0.0004998
#> alternative hypothesis: two.sided
```

A Fisher-próba outputja alapján a három életkori csoport dohányzási szokása nem egyezik meg egymással ($p < 0{,}001$). Most megvizsgáljuk, hogy a három életkori csoport közül melyik csoportok között van eltérés. A `pairwiseNominalIndependence()` függvény használatával páronkénti összehasonlításokat végezhetünk.

```
# Fisher egzakt próba - utóvizsgálat - Páronkénti összehasonlítások
library(rcompanion)
pairwiseNominalIndependence(x = dm21, compare = "row", fisher = T, gtest = F,
    chisq = F, method = "fdr", digits = 3)
#>           Comparison p.Fisher p.adj.Fisher
#> 1 fiatal : kozepkoru 1.52e-04     2.28e-04
#> 2      fiatal : idos 2.82e-12     8.46e-12
#> 3   kozepkoru : idos 1.43e-02     1.43e-02
```

A Fisher-próba utóvizsgálat alapján minden csoport-pár esetén szignifikáns különbséget talált: a fiatalok és a középkorúak között $p < 0{,}001$, a fiatalok és az idősek között $p < 0{,}001$, míg a középkorúak és az idősek között $p = 0{,}014$.

Eddig megvizsgáltunk három különböző lehetőséget két (független) nominális változó kapcsolatának vizsgálatára:

- Khí-négyzet próba: `chisq.test()`,
- G-próba: `G.test()`,
- Fisher-próba: `fisher.test()`.

Minden bemutatott példában konstans értékeket használtunk, de sok esetben adatmátrixban keletkeznek a bemenő adatok. Azt javasoljuk, hogy az `xtabs()` függvény használatával készítsünk egy mátrixot, amely a két nominális változó összesített adatait tartalmazza. Ez a mátrix már bemenete lehet a fenti kapcsolatvizsgálati próbáknak, pontosan úgy, ahogy azt korábban megmutattuk.

Induljunk ki egy adatmátrixból, amely a fiatalok, középkorúak és idősek dohányzási szokásait tartalmazza. Az adatmátrixban a `korosztaly` és a `dohanyzasi_szokas` változók szerepelnek.

```
adat <- "  korosztaly    dohanyzasi_szokas
               fiatal      'nem dohányzik'
            középkorú      'nem dohányzik'
            középkorú      'nem dohányzik'
            középkorú      'nem dohányzik'
            középkorú      'nem dohányzik'
                 idős      'nem dohányzik'
               fiatal  'naponta dohányzik'
            középkorú  'naponta dohányzik'
                 idős  'naponta dohányzik'
                 idős  'naponta dohányzik'
               fiatal   'alkalmi dohányos'
            középkorú   'alkalmi dohányos'
                 idős   'alkalmi dohányos'  "

# adatmátrix beolvasása
df22 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df22$korosztaly <- factor(df22$korosztaly)
df22$dohanyzasi_szokas <- factor(df22$dohanyzasi_szokas)
dplyr::glimpse(df22)  # adatmátrix szerkezete
#> Rows: 13
#> Columns: 2
#> $ korosztaly        <fct> fiatal, középkorú, középkorú, középkorú, köz~
#> $ dohanyzasi_szokas <fct> nem dohányzik, nem dohányzik, nem dohányzik,~
dm22 <- xtabs(~korosztaly + dohanyzasi_szokas, data = df22)
```

A fenti sorokkal létrehozott `dm22` mátrix már bemenete lehet a fenti kapcsolatvizsgálati próbáknak.

10.6.4. Páros kontingencia táblák

Előfordulhatnak olyan vizsgálatok, ahol ugyanazt a nominális változót két különböző helyzetben vagy időpontban mérjük. Például egy kurzus elején és a kurzus végén felteszünk

egy szakmai kérdést, és megnézzük, hogy a helyes és helytelen válaszok aránya hogyan változott.

A példa adatai egy adatmátrixban így szerepelhetnének:

```
adat <- "  elotte         utana
           helyes        helyes
           helyes        helyes
           helyes        helyes
        helytelen        helyes
        helytelen        helyes
        helytelen     helytelen
        helytelen        helyes
        helytelen     helytelen
        helytelen        helyes
        helytelen        helyes
        helytelen        helyes  "

# adatmátrix beolvasása
df23 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df23$elotte <- factor(df23$elotte)
df23$utana <- factor(df23$utana)
str(df23)  # adatmátrix szerkezete
#> 'data.frame':    11 obs. of  2 variables:
#>  $ elotte: Factor w/ 2 levels "helyes","helytelen": 1 1 1 2 2 2 2 2 2 2 ...
#>  $ utana : Factor w/ 2 levels "helyes","helytelen": 1 1 1 1 1 2 1 2 1 1 ...
```

A fenti adatok páros kontingencia táblában is megjeleníthetők. A `xtabs()` függvény használatával készíthetünk egy mátrixot, amely a két nominális változó összesített adatait tartalmazza.

```
# mátrix létrehozása
dm23 <- xtabs(~elotte+utana,data = df23)
dm23
#>            utana
#> elotte      helyes helytelen
#>   helyes         3         0
#>   helytelen      6         2
```

A McNemar-próba és McNemar–Bowker-próba alkalmas a két kondícióban mért nominális változók közötti eloszlás-eltérés vizsgálatára. A McNemar-próba a 2×2-es kontingencia táblákra vonatkozik, míg a McNemar–Bowker-próba $N \times N$-es kontingencia táblákra alkalmazható, ahol a nominális változónak kettőnél több értéke is lehet. Mindkét esetben a `mcnemar.test()` függvényt használhatjuk, és a mátrix bemenettel hívjuk meg.

```
# --= McNemar próba =--
mcnemar.test(x = dm23)
#>
#>  McNemar's Chi-squared test with continuity correction
#>
#> data:  dm23
#> McNemar's chi-squared = 4.1667, df = 1, p-value = 0.04123
```

A McNemar-próba outputja alapján a két kondícióban mért nominális változók eloszlása szignifikánsan eltér egymástól ($\chi^2(1) = 4{,}17$; $p = 0{,}041$).

Néha az adatok eleve összesítve állnak rendelkezésre, így közvetlenül is elkészíthetjük a McNemar-próba bemenő mátrixát. Nézzük most más adatokkal ugyanazt a példát, azaz a kurzus elején és a kurzus végén felteszünk egy szakmai kérdést, és megnézzük, hogy a helyes és helytelen válaszok aránya hogyan változott.

```
adat <- "  elotte  utana_helyes  utana_helytelen
    elotte_helyes             5               15
 elotte_helytelen            23                8  "
# adatmátrix beolvasása
df24 <- read.table(textConnection(adat), header=T, sep="",
                   row.names = 1)
dm24 <- as.matrix(df24)  # mátrixszá konvertálás
dm24
#>                  utana_helyes utana_helytelen
#> elotte_helyes               5              15
#> elotte_helytelen           23               8
```

```
# --= McNemar próba =--
mcnemar.test(x = dm24)
#>
#>  McNemar's Chi-squared test with continuity correction
#>
#> data:  dm24
#> McNemar's chi-squared = 1.2895, df = 1, p-value = 0.2561
```

A McNemar-próba outputja alapján a két kondícióban mért nominális változók eloszlása nem tér el egymástól ($\chi^2(1) = 1{,}29$; $p = 0{,}256$).

Említettük, hogy a McNemar-próba a 2 × 2-es kontingencia táblákra vonatkozik, míg a McNemar–Bowker-próba $N \times N$-es kontingencia táblákra alkalmazható, Nézzünk példát utóbbi esetre is.

Megkérdeztük a munkavállalókat egy motivációs beszélgetés előtt és után, hogy mennyire tartja valószínűnek, hogy 5 év múlva még itt fog dolgozni. A lehetséges válasz egy három szintű faktorba rendezhető: "igen", "nem" és "nem tudja". Hozzuk létre a szükséges mátrixot!

```
adat <- "  elotte  utana_igen  utana_nem  utana_nem_tudja
      elotte_igen           5         15                8
       elotte_nem          23          8                8
 elotte_nem_tudja          76          9                8  "

# adatmátrix beolvasása
df25 <- read.table(textConnection(adat), header=T, sep="",
                   row.names = 1)
dm25 <- as.matrix(df25)  # mátrixszá konvertálás
dm25
#>                  utana_igen utana_nem utana_nem_tudja
#> elotte_igen               5        15               8
#> elotte_nem               23         8               8
#> elotte_nem_tudja         76         9               8
```

```
# --= McNemar-Bowker próba =--
mcnemar.test(x = dm25)
#>
#>  McNemar's Chi-squared test
#>
#> data:  dm25
#> McNemar's chi-squared = 56.791, df = 3, p-value = 2.848e-12
```

A McNemar–Bowker-próba outputja alapján a két kondícióban mért nominális változók eloszlása szignifikánsan eltér egymástól ($\chi^2(3) = 56{,}79$; $p < 0{,}001$). Mivel a nominális változónknak három értéke van, ezért szignifikáns esetben a McNemar–Bowker-próba utóvizsgálatára is szükség van. Az `{rcompanion}` csomagban található `nominalSymmetryTest()` függvény használatával végezhetjük el a McNemar–Bowker-próba utóvizsgálatát. A `method` argumentumban megadhatjuk a p értékek korrekciós módszerét.

```
# McNemar–Bowker próba – utóvizsgálat
library(rcompanion)
nominalSymmetryTest(dm25)
#> $Global.test.for.symmetry
#>   Dimensions  p.value
#> 1      3 x 3 2.85e-12
#>
#> $Pairwise.symmetry.tests
#>                                                 Comparison  p.value
#> 1             elotte_igen/utana_igen : elotte_nem/utana_nem    0.256
#> 2 elotte_igen/utana_igen : elotte_nem_tudja/utana_nem_tudja 2.67e-13
#> 3   elotte_nem/utana_nem : elotte_nem_tudja/utana_nem_tudja        1
#>   p.adjust
#> 1 3.84e-01
#> 2 8.01e-13
#> 3 1.00e+00
#>
#> $p.adjustment
#>   Method
#> 1    fdr
#>
#> $statistical.method
#>         Method
#> 1 McNemar test
```

A McNemar–Bowker-próba utóvizsgálata alapján három 2×2-es mátrixra vonatkozó szimmetria kerül tesztelésre. A második eset p értéke megmutatja, hogy szignifikáns eltolódás van a “nem tudja” válaszokból az “igen” válaszok felé ($p < 0{,}001$). A másik két lehetséges esetben nincs szignifikáns eltérés a válaszok eloszlásában.

Az adatok természetesen adatmátrixból is származhatnak, így most a McNemar–Bowker-próbát erre az esetre is bemutatjuk. Az adatok az `elotte` és `utana` változókban szerepelnek, és a válaszok “igen”, “nem” és “nem tudja” lehetőségeket tartalmaznak.

```
adat <- "  elotte         utana
             igen           nem
             igen           nem
             igen           nem
             igen          igen
              nem    'nem tudja'
```

```
              nem    'nem tudja'
             igen    'nem tudja'
       'nem tudja'          igen
       'nem tudja'           nem  "

# adatmátrix beolvasása
df26 <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df26$elotte <- factor(df26$elotte)
df26$utana <- factor(df26$utana)
str(df26)  # adatmátrix szerkezete
#> 'data.frame':    9 obs. of  2 variables:
#>  $ elotte: Factor w/ 3 levels "igen","nem","nem tudja": 1 1 1 1 2 2 1 3 3
#>  $ utana : Factor w/ 3 levels "igen","nem","nem tudja": 2 2 2 1 3 3 3 1 2
```

A mátrixot az `xtabs()` függvény használatával készítjük el, amely már a `mcnemar.test()` függvény bemenete is lehet.

```
# mátrix létrehozása
dm26 <- xtabs(~elotte + utana, data = df26)
```

10.6.5. Cohran-Q próba

Ha kettőnél több kondícióban is mérjük ugyanazokat a vizsgálati személyeket, és a függő változónk dichotóm, akkor a Cohran-Q próbát használhatjuk. A Cohran-Q próba a McNemar-próba kiterjesztésének is tekinthető kettőnél több mérési szituációban.

A McNemar-próba esetében használt példát úgy fejleszthetjük tovább, hogy ugyanazt a kérdést feltesszük óra előtt, óra közben és óra végén is.

Kezdjük a széles formával.

```
adat <- "  tanulo     elotte     kozben      utana
                a     helyes  helytelen     helyes
                b     helyes     helyes     helyes
                c  helytelen     helyes     helyes
                d  helytelen  helytelen     helyes
                e  helytelen     helyes     helyes
```

```
                f  helytelen     helyes     helyes
                g  helytelen     helyes     helyes  "

# adatmátrix beolvasása
df27_szeles <- read.table(textConnection(adat), header=T, sep="")
```

```
# faktorrá alakítások
df27_szeles$tanulo <- factor(df27_szeles$tanulo)
df27_szeles$elotte <- factor(df27_szeles$elotte)
df27_szeles$kozben <- factor(df27_szeles$kozben)
df27_szeles$utana <- factor(df27_szeles$utana)
str(df27_szeles)  # adatmátrix szerkezete
#> 'data.frame':    7 obs. of  4 variables:
#>  $ tanulo: Factor w/ 7 levels "a","b","c","d",..: 1 2 3 4 5 6 7
#>  $ elotte: Factor w/ 2 levels "helyes","helytelen": 1 1 2 2 2 2 2
#>  $ kozben: Factor w/ 2 levels "helyes","helytelen": 2 1 1 2 1 1 1
#>  $ utana : Factor w/ 1 level "helyes": 1 1 1 1 1 1 1
```

A Cohran-Q próba az `{RVAideMemoire}` csomagban található `cochran.qtest()` függvény használatával végezhető el. Hosszú adatmátrixot vár, így elkészítjük a hosszú formát is.

```
df27_hosszu <- tidyr::pivot_longer(data = df27_szeles, cols = 2:4,
                                   names_to = "idopont",
                                   values_to = "valasz")
df27_hosszu$tanulo  <- factor(df27_hosszu$tanulo)
df27_hosszu$idopont <- factor(df27_hosszu$idopont)
df27_hosszu$valasz  <- factor(df27_hosszu$valasz)
```

```
# --= Cohran-Q próba =--
library(RVAideMemoire)
cochran.qtest(valasz ~ idopont | tanulo, data = df27_hosszu)
#>
#>  Cochran's Q test
#>
#> data:  valasz by idopont, block = tanulo
#> Q = 6.3333, df = 2, p-value = 0.04214
#> alternative hypothesis: true difference in probabilities is not equal to 0
#> sample estimates:
#> proba in group elotte proba in group kozben  proba in group utana
```

```
#>             0.7142857             0.2857143             0.0000000
#>
#>         Pairwise comparisons using Wilcoxon sign test
#>
#>        elotte kozben
#> kozben 0.5000      -
#> utana  0.1875    0.5
#>
#> P value adjustment method: fdr
```

A Cohran-Q próba outputja alapján a három kondícióban mért nominális változók eloszlása szignifikánsan eltér egymástól ($Q(2) = 6{,}33$; $p = 0{,}042$). A Cohran-Q próba utóvizsgálatát az `{rcompanion}` csomagban található `pairwiseMcnemar()` függvény használatával végezhetjük el. A `method` argumentumban megadhatjuk a p értékek korrekciós módszerét. A "helytelen" válaszok mintabeli arányait is megtaláljuk a 3 különböző időpontban.

```
# utóvizsgálat McNemar-próbával
library(rcompanion)
pairwiseMcnemar(valasz ~ idopont | tanulo, data = df27_hosszu,
                test = "mcnemar", method = "fdr", digits = 3)
#> $Test.method
#>      Test
#> 1 mcnemar
#>
#> $Adustment.method
#>   Method
#> 1    fdr
#>
#> $Pairwise
#>            Comparison chi.sq df p.value p.adjust
#> 1 elotte - kozben = 0    1.8  1    0.18   0.1800
#> 2  elotte - utana = 0      5  1  0.0253   0.0759
#> 3  kozben - utana = 0      2  1   0.157   0.1800
```

Az utóvizsgálat alapján az "elotte" és "utana" helyzetben a "helytelen" válaszok arányának a változása okozhatta a Cohran-Q próba szignifikáns eredményét, de ez az eltérés is csak tendenciaszintű ($p = 0{,}760$).

Összefoglalás

A valószínűségi próbák célja annak statisztikai eldöntése, hogy a megfigyelt gyakorisági eloszlás eltér-e egy előre feltételezett (hipotetikus) eloszlástól, illetve hogy két nominális változó között van-e kapcsolat. A fejezet három fő témakört ölel fel: illeszkedésvizsgálat egy vagy több valószínűségre, kapcsolatvizsgálat két nominális változó között, valamint ismételt mérésekre alkalmazható próbák, mint a McNemar- és Cohran-Q próba. Egymintás arányvizsgálat esetén binomiális próbát (`binom.test()`) vagy nagyobb minta esetén aránypróbát (`prop.test()`) alkalmazunk, ha például azt szeretnénk eldönteni, hogy egy adott kategória (pl. nők aránya) eltér-e egy feltételezett értéktől. Több kategóriás nominális változóknál egzakt multinomiális próbát (`multinomial.test()`), illetve közelítő módszerként `prop.test()`, `chisq.test()` vagy `G.test()` függvényeket használhatunk. Ha két nominális változó közötti kapcsolatot vizsgálunk, használhatunk khí-négyzet próbát (`chisq.test()`), G-próbát (`G.test()`), vagy kisebb minták esetén Fisher-próbát (`fisher.test()`). Párosított mérések esetén – például ugyanannak a személynek két különböző időpontban adott válaszai – a McNemar-próba (`mcnemar.test()`) és annak kiterjesztett változata, a McNemar-–Bowker-próba alkalmazható. Több időpontban mért dichotóm válasz esetén a Cohran-Q próba (`cochran.qtest()`) használatos, amely a McNemar-próba általánosítása.

Feladatok

1. A fejezetben használt teszteknek lehet alkalmazási feltétele. Soroljuk fel ezeket!
2. A fejezetben ugyanannak a hipotézisnek a vizsgálatára több próbát is bemutattunk. Foglaljuk össze, hogy melyik próbát mikor érdemes használni!
3. A fejezetben a nominális változók illeszkedésvizsgálatára és kapcsolatvizsgálatára fókuszáltunk. Ordinális változók esetén milyen próbákat használhatunk?

10.7. Hatásméret

Miről lesz szó? Ebben a fejezetben

- a hatásméret fogalmát és jelentőségét mutatjuk be, illetve
- a hatásméret kiszámításának különböző módszereit ismertetjük.

Az előző fejezetekben bemutattuk, hogyan használhatjuk fel az adatokat hipotézisek tesztelésére. Ezek a módszerek bináris választ adnak: vagy elutasítjuk, vagy megtartjuk (nem utasítjuk el) a nullhipotézist. Az outputok legtöbbször intervallumbecslést is tartalmaznak

a populációbeli paraméterre vonatkozóan, vagyis képet kaphatunk arról, hogy a becslések milyen tartományban tekinthetők megbízhatónak.

Azonban még egy kérdés biztosan nyitva maradt. Egy p < 0,05 eredmény önmagában még nem árulja el, hogy a megfigyelt eltérés mennyire jelentős vagy lényeges a gyakorlat szempontjából. Sőt az is előfordulhat, hogy egy nem szignifikáns eredmény mögött valójában érzékelhető nagyságú különbség húzódik – csak épp a minta túl kicsi ennek kimutatásához.

Szükség van a *hatásméret* (*effect size*) kiszámítására, amely lehetővé teszi, hogy kvantitatívan jellemezzük a hatás erősségét, függetlenül a mintanagyságtól. A hatásméret mindig a vizsgált hatás (különbség, kapcsolat, arányeltérés stb.) nagyságát fejezi ki, nem pedig annak valószínűségét vagy szignifikanciáját.

A hatásmérték mutatók lehetőséget adnak arra, hogy a statisztikai eredmények mögötti gyakorlati jelentőséget is megértsük. Például egy $d = 0{,}80$ hatásméret már erős hatásra utalhat, míg egy $d = 0{,}20$ csupán gyenge eltérést jelez, még akkor is, ha az utóbbi esetben a p érték szignifikáns lenne.

A hatásméret kiszámításához az `{effectsize}` csomagot használjuk. A korábban bemutatott próbák többségében megadjuk a hatásméret kiszámítását. Egyes esetekben többfajta számítási mód is létezik, próbálunk teljes képet nyújtani az `{effectsize}` csomag határain belül.

Az `{effectsize}` csomag a hatásméret mérőszámok értelmezéséhez is segítséget nyújt. Különböző szerzők általában négy fokozat valamelyikébe sorolják az egyes hatásméreteket: elhanyagolható hatás (very small), kis hatás (small), közepes hatás (moderate) és nagy hatás (large). Ezek a kategóriák segíthetik az eredmények megértését, de tartsuk szem előtt, hogy az egyes szerzők maguk is máshol húzzák meg a határokat és az értelmezés változhat például tudományterülettől, hipotézistől, korábbi eredményektől függően. A hatásméretek értelmezéséről további információt a csomag weboldalán találunk.

A következő alfejezetekben a hipotézisvizsgálatok során használt adatmátrixokat használjuk fel. Például a `df01` adatmátrix szerkezetének felidézéséhez vissza kell lapoznunk a 10.2.1. fejezethez. Minden esetben 2 függvényhívás tartozik a hatásméret meghatározásához: az első a hatásméret kiszámítása, a második pedig az értelmezés. Az utóbbi függvény neve tartalmazza az `interpret_` karaktersorozatot, és konkrét hatásméretet vár bemenetként, amelyet az előtte futtatott hatásméret-számító függvényből nyerünk ki.

Viszonylag kevés kísérőszöveget mutatunk be a következő részben, szándékaink szerint ezen rész egyfajta katalógusként működhet: a próba ismeretében a megfelelő részhez lapozva a hatásméret kiszámítására találunk példát.

10.7.1. T-próbák

Az t-próbák esetében a `cohens_d()` függvénnyel határozhatjuk meg a hatásméretet. Az `interpret_cohens_d()` adja meg az értelmezést.

```
# egymintás t-próba: Cohen d
effectsize::cohens_d(x = df01$pontszam, mu = 10, ci = 0.95)
#> Cohen's d |        95% CI
#> -------------------------
#> -0.15     | [-0.95, 0.66]
#>
#> - Deviation from a difference of 10.
effectsize::interpret_cohens_d(d = -0.15, rules = "cohen1988")
#> [1] "very small"
#> (Rules: cohen1988)
```

```
# kétmintás t-próba: Cohen d
effectsize::cohens_d(x = pontszam ~ modszer, data = df02,
                     pooled_sd = T, ci = 0.95)
#> Cohen's d |        95% CI
#> -------------------------
#> 0.09      | [-1.30, 1.48]
#>
#> - Estimated using pooled SD.
effectsize::interpret_cohens_d(d = 0.09, rules = "cohen1988")
#> [1] "very small"
#> (Rules: cohen1988)
```

```
# Welch-féle d: Cohen d
effectsize::cohens_d(x = pontszam ~ modszer, data = df02,
                     pooled_sd = F, ci = 0.95)
#> Cohen's d |        95% CI
#> -------------------------
#> 0.09      | [-1.30, 1.48]
#>
#> - Estimated using un-pooled SD.
effectsize::interpret_cohens_d(d = 0.09, rules = "cohen1988")
#> [1] "very small"
#> (Rules: cohen1988)
```

```
# páros t-próba, széles adatbázis: Cohen d
effectsize::cohens_d(x = df03_szeles$elotte, y = df03_szeles$utana,
                     paired=T, ci = 0.95)
#> Cohen's d |       95% CI
#> ------------------------
#> 1.70      | [0.05, 3.28]
```

10.7.2. Varianciaelemzések

```
# egyszempontos varianciaelemzés - eta-négyzet
effectsize::eta_squared(model = aov_1, partial = F,
                        generalized = F, ci = 0.95)
#> # Effect Size for ANOVA (Type I)
#>
#> Parameter | Eta2 |       95% CI
#> -----------------------------
#> modszer   | 0.67 | [0.19, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_eta_squared(es = 0.67, rules = "cohen1992")
#> [1] "large"
#> (Rules: cohen1992)
```

```
# egyszempontos varianciaelemzés - általánosított eta-négyzet
effectsize::eta_squared(model = aov_ez_1, partial = F,
                        generalized = T, ci = 0.95)
#> # Effect Size for ANOVA (Type III)
#>
#> Parameter | Eta2 (generalized) |       95% CI
#> ------------------------------------------
#> sport     |               0.66 | [0.04, 1.00]
#>
#> - Observed variables: All
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_eta_squared(es = 0.66, rules = "cohen1992")
#> [1] "large"
#> (Rules: cohen1992)
```

```
# egyszempontos varianciaelemzés - omega-négyzet
effectsize::omega_squared(model = aov_1, partial = T, ci = 0.95)
#> # Effect Size for ANOVA
#>
#> Parameter | Omega2 |       95% CI
#> ---------------------------------
#> modszer   |   0.57 | [0.05, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_omega_squared(es = 0.57, rules = "cohen1992")
#> [1] "large"
#> (Rules: cohen1992)
```

```
# egyszempontos varianciaelemzés  - epsilon-négyzet
effectsize::epsilon_squared(model = aov_1, partial = T, ci = 0.95)
#> # Effect Size for ANOVA
#>
#> Parameter | Epsilon2 |       95% CI
#> ----------------------------------
#> modszer   |     0.59 | [0.07, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_epsilon_squared(es = 0.59, rules = "cohen1992")
#> [1] "large"
#> (Rules: cohen1992)
```

10.7.3. Korreláció és regresszió

```
# korrelációszámítás - Pearson r
cor(x = df06$homerseklet, y = df06$jegkrem, method = "pearson")
#> [1] 0.8536321
effectsize::interpret_r(r = 0.94, rules = "funder2019")
#> [1] "very large"
#> (Rules: funder2019)
```

```
# korrelációszámítás - Kendall tau
cor(x = df06$homerseklet, y = df06$jegkrem, method = "kendall")
#> [1] 0.6910233
```

```
effectsize::interpret_r(r = 1, rules = "funder2019")
#> [1] "very large"
#> (Rules: funder2019)
```

```
# korrelációszámítás - Spearman rho
cor(x = df06$homerseklet, y = df06$jegkrem, method = "spearman")
#> [1] 0.8143859
effectsize::interpret_r(r = 1, rules = "funder2019")
#> [1] "very large"
#> (Rules: funder2019)
```

```
# Egyszerű lineáris regresszió: determinációs együttható
summary(lm_1)$adj.r.squared
#> [1] 0.6834691
effectsize::interpret_r2(r2 = 0.84, rules = "cohen1988")
#> [1] "substantial"
#> (Rules: cohen1988)
```

10.7.4. Nemparaméteres próbák

```
# Egymintás Wilcoxon.-próba: rang-biszeriális korreláció
effectsize::rank_biserial(df07$felkeszultseg, mu=3, ci = 0.95)
#> r (rank biserial) |       95% CI
#> --------------------------------
#> 1.00              | [1.00, 1.00]
#>
#> - Deviation from a difference of 3.
effectsize::interpret_rank_biserial(1, rules = "funder2019")
#> [1] "very large"
#> (Rules: funder2019)
```

```
# Mann-Whitney próba
effectsize::p_superiority(x = felkeszultseg ~ oktato, data=df08,
                          parametric = F, verbose = F)
#> Pr(superiority) |       95% CI
#> --------------------------------
#> 0.00            | [0.00, 0.00]
```

```
#>
#> - Non-parametric CLES
# - r
effectsize::rank_biserial(x = felkeszultseg ~ oktato, data=df08,
                          parametric = F, verbose = F)
#> r (rank biserial) |         95% CI
#> ------------------------------------
#> -1.00             | [-1.00, -1.00]
```

```
# Kruskal-Wallis próba
effectsize::rank_epsilon_squared(x = felkeszultseg ~ oktato,
                                 data=df10, ci = 0.95)
#> Epsilon2 (rank) |       95% CI
#> -------------------------------
#> 0.73            | [0.73, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_epsilon_squared(0.73, rules = "cohen1992")
#> [1] "large"
#> (Rules: cohen1992)
```

```
# Friedman-próba - Kendall-W
effectsize::kendalls_w(x = felkeszultseg ~ idopont | hallgato,
              data = df11_hosszu, ci = 0.95)
#> Kendall's W |       95% CI
#> --------------------------
#> 0.66        | [0.58, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_kendalls_w(0.66, rules = "landis1977")
#> [1] "substantial agreement"
#> (Rules: landis1977)
```

10.7.5. Valószínűség próbái

```
## Illeszkedésvizsgálat
effectsize::fei(x = table(df20), p = c(0.35, 0.05, 0.6), ci = 0.95)
#> Fei  |       95% CI
```

```
#> -------------------
#> 0.31 | [0.17, 1.00]
#>
#> - Adjusted for uniform expected probabilities.
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::cohens_w(x = table(df20), p = c(0.35, 0.05, 0.6), ci = 0.95)
#> Cohen's w |       95% CI
#> ------------------------
#> 1.33      | [0.73, 4.36]
#>
#> - One-sided CIs: upper bound fixed at [4.36~].
effectsize::pearsons_c(x = table(df20), p = c(0.35, 0.05, 0.6), ci = 0.95)
#> Pearson's C |       95% CI
#> --------------------------
#> 0.80        | [0.59, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
```

```
# kapcsolatvizsgálat: 2x2
effectsize::phi(x = dm23, adjust = F, ci = 0.95)
#> Phi  |       95% CI
#> --------------------
#> 0.29 | [0.00, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_phi(0.48, rules = "funder2019")
#> [1] "very large"
#> (Rules: funder2019)
```

```
effectsize::oddsratio(x = dm23, adjust = F, ci = 0.95)
#> Odds ratio |   95% CI
#> ----------------------
#> Inf        | [ , Inf]
effectsize::interpret_oddsratio(OR = 0.12, rules = "cohen1988")
#> [1] "large"
#> (Rules: cohen1988)
```

```
# kapcsolatvizsgálat: NxN
effectsize::cramers_v(x = dm22, adjust = F, ci = 0.95)
#> Cramer's V |       95% CI
```

```
#> ------------------------
#> 0.29       | [0.00, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
effectsize::interpret_cramers_v(0.29, rules = "funder2019")
#> [1] "medium"
#> (Rules: funder2019)
```

```
# McNemar-próba
effectsize::cohens_g(x = dm24, ci = 0.95)
#> Cohen's g |        95% CI
#> -------------------------
#> 0.11      | [-0.05, 0.24]
effectsize::interpret_cohens_g(0.1, rules = "cohen1988")
#> [1] "small"
#> (Rules: cohen1988)
```

```
# McNemar-Bowker próba
effectsize::cohens_g(x = dm25, ci = 0.95)
#> Cohen's g |       95% CI
#> ------------------------
#> 0.28      | [0.20, 0.34]
effectsize::interpret_cohens_g(0.28, rules = "cohen1988")
#> [1] "large"
#> (Rules: cohen1988)
```

Összefoglalás

A hatásméret (effect size) egy számértékkel kifejezett mutató, amely azt jelzi, hogy mekkora a vizsgált hatás – függetlenül attól, hogy az statisztikailag szignifikáns-e. Különösen fontos akkor, amikor a minta túl kicsi (és ezért a p érték nem szignifikáns), vagy épp túl nagy (és egy elhanyagolható hatás is szignifikánsnak tűnik). A hatásméretek értelmezése során hasznos támpontot adnak az ún. szabályrendszerek, amelyek a hatásokat elégségesen kis, kis, közepes és nagy kategóriákba sorolják. Ezeket azonban mindig a kutatási kontextushoz illeszkedve kell értékelni. A hatásméretek kiszámításához és értelmezéséhez a `{effectsize}` csomag kínál egységes és széles körben alkalmazható eszközöket. A csomag nemcsak számolja, hanem az `interpret_*()` függvényeken keresztül segít értelmezni is az eredményeket. A fejezet katalógusszerűen mutatja be, hogyan számíthatjuk ki a hatásméretet az adott próbatípushoz tartozó függvénnyel, és hogyan értelmezzük a kapott értékeket. A példák t-próbáktól kezdve varianciaanalízisen, korre-

láción és nemparaméteres próbákon át egészen a valószínűségi és kontingenciatáblás elemzésekig terjednek.

🎯 Feladatok

1. A `{pwr}` csomag `cohen.ES()` függvénye a klasszikus (Cohen, 1988) példákat szolgáltatja kicsi, közepes és nagy hatásméret esetén a csomagban található tesztekhez. Határozzuk meg és rendszerezzük ezeket az értékeket!
2. Az előző fejezetben bemutatott hipotézisvizsgálatok egy részét lefedtük a hatásméret számításával ebben a fejezetben. Mely eljárások maradtak ki, és milyen hatásméret-számításokat használhatunk ezekhez?

10.8. Statisztikai erő és mintanagyság 🤔

ℹ Miről lesz szó? Ebben a fejezetben

- a statisztikai erő fogalmát, jelentőségét és összefüggéseit ismertetjük, valamint
- a statisztikai erő és mintanagyság kiszámításának különböző módszereit mutatjuk be.

A statisztikai erővel kapcsolatos elemzés fontos része a kísérlet megtervezésének. Lehetővé teszi például, hogy meghatározzuk azt a mintanagyságot, amely egy adott nagyságú hatás kimutatásához szükséges. Egy kísérlet megtervezésekor valóban kulcskérdés, hogy mekkora mintával dolgozzunk. Ha túl kicsi a minta, akkor könnyen lehet, hogy létező hatást nem tudunk kimutatni.

A mintaméret egy olyan négyes tagja, amelynek minden alkotója a másik 3 birtokában meghatározható:

1. A *minta mérete* a vizsgálatba bevont egyedek száma. Nagyobb mintaméret nagyobb statisztikai erővel jár, és kisebb hatásméret is kimutatható.
2. A *hatásmérték*, vagyis az alternatív hipotézisnek megfelelő hatás nagysága a populációban, például a várható értékek eltérése szórásban kifejezve (Cohen-d), vagy a korreláció nagysága (r) két változó kapcsolatvizsgálatában. Minél nagyobb a valóságos hatás mértéke, annál könnyebben észlelhető a hatás, azaz annál nagyobb a próba ereje, és annál kevesebb elemű mintára van szükségünk a hatás kimutatásához.
3. A *próba ereje* $(1 - \beta)$, azaz a nullhipotézis helyes elutasításának valószínűsége. Képlettel $1 - \beta$ formában írható, ahol β a másodfajú hiba valószínűsége. A másodfajú hiba a nullhipotézis hamis megtartásának valószínűsége (hamis negatív eset). Minél nagyobb

az erő, annál valószínűbb, hogy észlelhető egy létező hatás. A próba erejét tipikusan 0,8-ra állítjuk.

4. A *szignifikancia szint* (α) annak a valószínűsége, hogy hibásan elutasítják a nullhipotézist. Ezt nevezik elsőfajú hibának is (hamis pozitív eset). Minél alacsonyabb a szignifikancia szint, annál valószínűbb, hogy elkerüljük a hamis pozitív eredményt. Az α standard beállítása 0,05.

A statisztikai erővel kapcsolatos vizsgálatokhoz a `{pwr}` csomagot fogjuk használni. A 10.1. táblázat összefoglalja a csomag függvényeit, megmutatja a paraméterezésüket és megnevezi azt a próbát, ahol a függvényt felhasználhatjuk. A függvények paraméterei utalnak a 4 alkotóra, az `n=` mindig a mintaelemszámra, a `power=` a statisztikai erőre, a `sig.level=` a szignifikancia szintre utal. A negyedik összetevő, a hatásmérték próbánként változó paraméternevet jelent. Például t-próbák esetében a `d=` paraméter a Cohen-d hatásmértéket jelenti, a `pwr.r.test()` függvényben az `r=` a korrelációs együtthatóra utal.

10.1. táblázat: A `{pwr}` csomag függvényei a paraméterezésükkel és a próbák megnevezésével *(saját szerkesztés)*

Függvény	Milyen vizsgálatban használjuk
`pwr.t.test(n, d, sig.level, power, type)`	t-próba (egymintás, páros, kétmintás azonos elemszámok)
`pwr.t2n.test(n1, n2, d, sig.level, power)`	kétmintás t-próba (eltérő elemszámok)
`pwr.anova.test(k, n, f, sig.level, power)`	egyszempontos varianciaelemzés
`pwr.r.test(n, r, sig.level, power)`	korrelációs együttható
`pwr.f2.test(u, v, f2, sig.level, power)`	lineáris modell
`pwr.p.test(h, n, sig.level, power)`	egymintás valószínűség
`pwr.2p.test(h, n, sig.level, power)`	két valószínűség, azonos elemszámok
`pwr.2p2n.test(h, n1, n2, sig.level, powe)`	két valószínűség, eltérő elemszámok
`pwr.chisq.test(w, N, df, sig.level, power)`	khí-négyzet próba

Amennyiben a 10.1. táblázat függvényeivel a szükséges mintaelemszám megállapítása a célunk, akkor a másik három mennyiséget meg kell határoznunk, míg a mintaelemszámra

vonatkozó paramétert `NULL`-ra kell állítanunk (`n=NULL`). Ezt a módszer kell használnunk abban az esetben is, ha a más összetevőt keresünk: a keresett paramétert `NULL`-ra állítjuk, a másik hármat pedig értékkel látjuk el a függvény hívása során.

A mintaelemszám kiszámításához szükséges statisztikai erő és szignifikancia szint általában a kísérlet körülményeiből könnyen meghatározható, standard értékük a `power=0.8` és `sig.level=0.05` paraméterekkel állítható be a 10.1. táblázat függvényeiben. A mintaelemszám számításához szükséges harmadik összetevő, a hatásméret meghatározásához azonban tapasztalat, elegendő háttér információ és hasonló témában publikált tanulmányok is szükségesek lehetnek. Könnyebbség, hogy létezik ökölszabály, amelyek megkülönböztetik a hatásméret legalább három kategóriáját, így beszélünk kis, közepes és nagy hatásméretről. Sajnos ezek a szabályok tudományterülettől függhetnek, ráadásul különböző statisztikai próbák eltérő hatásméret mérőszámmal rendelkeznek, így a kategória határok tesztenként változnak.

10.8.1. T-próbák

A t-próbákkal kapcsolatos mintaelemszám meghatározásához a `pwr.t.test()` függvényt használjuk, a hatásmérték a `d=` argumentummal adható meg, amely a Cohe-d hatásmértéket jelenti.

Számoljuk ki, hogy egymintás t-próba esetén mekkora mintanagyság szükséges 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,5) és 0,05-ös szignifikanciaszintet alkalmazunk. A `pwr.t.test()` függvényt használjuk

```
# egymintás t-próba
library(pwr)
pwr::pwr.t.test(n = NULL, d = 0.5, sig.level = 0.05, power = 0.8,
                type="one.sample", alternative = "two.sided")
#>
#>      One-sample t test power calculation
#>
#>               n = 33.36713
#>               d = 0.5
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
```

A kalkulált mintaelemszám felfelé kerekítve 34. Láthatjuk, hogy egymintás t-próbához a `type="one.sample"` argumentum megadása szükséges.

Páros t-próba esetén a `type="paired"` argumentumot kell használnunk.

```
# páros t-próba
pwr::pwr.t.test(n = NULL, d = 0.5, sig.level = 0.05, power = 0.8,
                type="paired", alternative = "two.sided")
#>
#>      Paired t test power calculation
#>
#>               n = 33.36713
#>               d = 0.5
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
#>
#> NOTE: n is number of *pairs*
```

Láthatjuk, hogy 0,5-ös hatásméret és 0,05-ös szignifikanciaszint esetén a 0,8-as statisztikai erőhöz szintén 34 fős mintára van szükség.

Kétmintás t-próba a `type="two.sample"` argumentummal végezhető.

```
# kétmintás t-próba, azonos csoportlétszámok
pwr::pwr.t.test(n = NULL, d = 0.5, sig.level = 0.05, power = 0.8,
                type="two.sample", alternative = "two.sided")
#>
#>      Two-sample t test power calculation
#>
#>               n = 63.76561
#>               d = 0.5
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
#>
#> NOTE: n is number in *each* group
```

Láthatjuk, hogy 0,5-ös hatásméret és 0,05-ös szignifikanciaszint esetén a 0,8-as statisztikai erőhöz mindkét csoportban 64 főre van szükség.

Amennyiben eltérő csoportlétszámmal dolgozunk, akkor valamelyik csoport létszámának ismeretében a másik csoport létszáma meghatározható.

```r
# kétmintás t-próba, eltérő csoportlétszámok
pwr::pwr.t2n.test(n1 = NULL, n2 = 45, d = 0.5, sig.level = 0.05,
             power = 0.8, alternative = "two.sided")
#>
#>      t test power calculation
#>
#>              n1 = 108.3923
#>              n2 = 45
#>               d = 0.5
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
```

A fenti output alapján azt mondhatjuk, hogy amennyiben az egyik csoport létszáma 40 fő, 0,5-ös a hatásméret és 0,05-ös a szignifikanciaszint, akkor a 0,8-as statisztikai erőhöz a másik csoportban 109 főre van szükség.

10.8.2. Varianciaelemzés

Egyszempontos varianciaelemzés esetén a `pwr.anova.test()` függvényt használjuk.

Számoljuk ki egyszempontos varianciaelemzés esetén 5 csoporttal számolva, mekkora mintanagyság szükséges minden csoportban 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,25) és a 0,05 szignifikanciaszintet alkalmazunk.

```r
# egyszempontos varianciaelemzés
pwr.anova.test(k = 5, n = NULL, f = 0.25, sig.level = 0.05,
               power = 0.8)
#>
#>      Balanced one-way analysis of variance power calculation
#>
#>               k = 5
#>               n = 39.1534
#>               f = 0.25
#>       sig.level = 0.05
#>           power = 0.8
#>
#> NOTE: n is number in each group
```

Látható, hogy csoportonként 40 fős mintára van szükség. A hatásmérték az `f=` argumentumban adható meg és a jelentése a Cohen-féle F. A Cohen-féle F értéke a független változó összes szintjének egyfajta standardizált átlagos hatása a populációban. A Cohen-féle F értéke 0 és plusz végtelen között bármi lehet. Nulla, ha a populációbeli várható értékek egyenlők egymással az egyes csoportokban, de bármeddig nőhet mivel az átlagok szórása az egyes csoportokon belüli átlagos szóráshoz képest bármilyen nagy lehet. Cohen azt javasolta (Cohen, 1988), hogy a 0,10, 0,25 és 0,40 értékek kis, közepes és nagy hatásméreteket jelentenek.

10.8.3. Korreláció és regresszió

A korrelációszámításhoz kötődő mintaelemszám meghatározásához a `pwr.r.test()` függvényt használjuk.

Számoljuk ki mekkora mintanagyság szükséges minden csoportban 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,5) és a 0,05 szignifikanciaszintet alkalmazunk.

```
# korrelációszámítás
pwr.r.test(n = NULL, r = 0.3, sig.level = 0.05, power = 0.8,
           alternative = "two.sided")
#>
#>      approximate correlation power calculation (arctangh transformation)
#>
#>               n = 84.07364
#>               r = 0.3
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
```

Látható, hogy 29 elemű mintára van szükségünk. A hatásmérték `r=` megadásához a korrelációs együttható értékét használjuk. Cohen szerint a 0,1, 0,3, és 0,5-ös *r* értékek kis, közepes és nagy hatásméreteket jelentenek.

Egyszerű lineáris regresszió esetén a `pwr.f2.test()` függvényt használhatjuk.

Számoljuk ki mekkora mintanagyság szükséges 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,15) és a 0,05 szignifikanciaszintet alkalmazunk.

```
# egyszerű lineáris regresszió
pwr.f2.test(u = 1, v = NULL, f2 = 0.15, sig.level = 0.05, power = 0.8)
#>
#>      Multiple regression power calculation
```

```
#>
#>               u = 1
#>               v = 52.315
#>              f2 = 0.15
#>       sig.level = 0.05
#>           power = 0.8
```

A függvény paraméterében az `f2=` a Cohen-féle F^2 értéket jelenti, ami a korábban látott Cohen-féle F négyzete. Cohen szerint a 0,02, 0,15 és 0,35 értékek kis, közepes és nagy hatásméreteket reprezentálnak.

Egyszerű lineáris regresszió esetén, mivel csak egy prediktorváltozó van, a számláló szabadsági fokainak száma 1, így a függvényben az `u=1` beállítást használjuk. Mivel a nevező szabadsági fokainak számára vagyunk kíváncsiak, a `v=` argumentumot `NULL`-ra állítottuk. A mintaelemszám meghatározásához a nevező szabadsági fokainak számát 2-vel meg kell növelnünk. Ennek megfelelően az ajánlott mintaelemszám 55.

10.8.4. Valószínűség próbái

10.8.4.1. Egyetlen valószínűségre vonatkozó próba

Egyetlen valószínűségre vonatkozó próba esetén a `pwr.p.test()` függvényt használjuk.

Számoljuk ki mekkora mintanagyság szükséges 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,5) és a 0,05 szignifikanciaszintet alkalmazunk.

```
# binomiális próba
pwr.p.test(h=0.5, n=NULL, sig.level=0.05, power=0.80,
           alternative="two.sided")
#>
#>      proportion power calculation for binomial distribution (arcsine
#>        transformation)
#>
#>               h = 0.5
#>               n = 31
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
```

Látjuk, hogy legalább 32 elemű mintával kell dolgoznunk. A hatásmérték megadására a `h=` argumentumot használjuk, amely Cohen-féle H értéket vár. Cohen szerint a 0,2, 0,5 és 0,8 értékek kis, közepes és nagy hatásméreteket reprezentálnak.

10.8.4.2. Két független valószínűségre vonatkozó próba

Két populációbeli arány összehasonlítása során is kiszámolhatjuk a szükséges mintaelemszámot.

```
# két populációbeli arány összehasonlítása, azonos csoportlétszámok
pwr.2p.test(h=0.5, n=NULL, sig.level=0.05, power=0.80,
            alternative="two.sided")
#>
#>      Difference of proportion power calculation for binomial distribution
#>       (arcsine transformation)
#>
#>               h = 0.5
#>               n = 63
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
#>
#> NOTE: same sample sizes
```

Látjuk, hogy mindkét mintában 63 elemre van szükség. A `h=` paraméterben továbbra is a kimutatni kívánt hatásmértéket kell megadnunk a Cohen-féle H mérőszámmal meghatározva.

Amennyiben eltérő mintaelemszámmal dolgozunk a két minta setében, akkor a `pwr.2p2n.test()` függvényt használjuk. Az egyik csoport 55 elemű.

```
# két populációbeli arány összehasonlítása, eltérő csoportlétszámok
pwr.2p2n.test(h=0.5, n1=55, n2=NULL, sig.level=0.05, power=0.80,
              alternative="two.sided")
#>
#>      difference of proportion power calculation for binomial distribution
#>        (arcsine transformation)
#>
#>               h = 0.5
#>              n1 = 55
```

```
#>              n2 = 73
#>       sig.level = 0.05
#>           power = 0.8
#>     alternative = two.sided
#>
#> NOTE: different sample sizes
```

Látjuk a másik csoportban 74 személyre van szükségünk.

10.8.4.3. Khí-négyzet próba

Khí-négyzet próbák esetén a `pwr.chisq.test()` függvényt használhatjuk, de két lényegesen eltérő szituációban is alkalmazhatjuk.

- Egyetlen nominális változó illeszkedésvizsgálata során a nominális változó szintjeinek száma (k) határozza meg a `df=` értékét: `df=k-1`.
- Kapcsolatvizsgálat esetén a két nominális változó szintjeinek száma (k és s) alapján: `df=(k-1)*(s-1)`.

A `w=` hatásmérték paraméter a Cohen-féle W értéket jelenti. Kis hatás: w = 0,10; közepes hatás: w = 0,30; nagy hatás: w = 0,50.

Az első példa az illeszkedésvizsgálatra vonatkozik. Egy 4 szintű nominális változó eloszlását vetjük össze egy konstans eloszlással. Számoljuk ki a szükséges mintaelemszámot 0,8-as statisztikai erő eléréséhez, ha a hatás mérete közepes (0,5) és a 0,05 szignifikanciaszintet alkalmazunk.

```
# khí-négyzet próba, illeszkedésvizsgálat
pwr.chisq.test(w=0.3, N=NULL, df = (4-1), sig.level = 0.05, power=0.8)
#>
#>      Chi squared power calculation
#>
#>               w = 0.3
#>               N = 121.1396
#>              df = 3
#>       sig.level = 0.05
#>           power = 0.8
#>
#> NOTE: N is the number of observations
```

A második példa a kapcsolatvizsgálatra vonatkozik. Számoljuk ki a szükséges mintaelemszámot 0,8-as statisztikai erő eléréséhez, ha a hatás mérete kicsi (0,1) és 0,05-ös szignifikanciaszintet alkalmazunk. A két nominális változó szintjeinek száma 5 és 6.

```
# khí-négyzet próba, kapcsolatvizsgálat
pwr.chisq.test(w=0.1,df=(5-1)*(6-1),power=0.80,sig.level=0.05)
#>
#>      Chi squared power calculation
#>
#>               w = 0.1
#>               N = 2096.079
#>              df = 20
#>       sig.level = 0.05
#>           power = 0.8
#>
#> NOTE: N is the number of observations
```

Összefoglalás

Egy jól megtervezett vizsgálat esetén alapvető kérdés, hogy mekkora minta szükséges ahhoz, hogy egy valós hatást megbízhatóan kimutathassunk. Túl kicsi minta esetén előfordulhat, hogy egy meglévő hatást nem tudunk érzékelni, ezzel szemben túl nagy mintával feleslegesen pazaroljuk az erőforrásainkat. A statisztikai erő (más néven a próba ereje) és a mintanagyság, a hatás nagysága és a szignifikanciaszint egymással összefüggő mennyiségek. Bármelyik kiszámítható, ha a másik három ismert. Ezek kiszámítására a `{pwr}` csomag függvényeit használtuk.

Feladatok

1. A fejezetben a statisztikai erő és mintanagyság kiszámítására kizárólag a `{pwr}` csomag függvényeit használtuk. Milyen R-beli és R-en kívüli alternatívák léteznek még?

10.9. Jamovi az R-ben 🤯

> **Miről lesz szó? Ebben a fejezetben**
>
> - megismerjük a `{jmv}` csomag egy-parancsos megközelítésmódját,
> - és példán keresztül szemléltetjük ezt a lehetőséget.

A hipotézisvizsgálat összetett tevékenység, nem pusztán a próbát megvalósító függvényhívásból áll (például az `aov()` hívásából egyszempontos varianciaelemzésnél), hanem körbeveszik leíró statisztikai, feltételvizsgáló, hatásmérték számoló, vagy utóvizsgálatokat végző parancsok is. Feladatunk teljesítéséhez rendszerint több függvényhívásra van szükségünk, és ezek újabb és újabb csomagok betöltését és karbantartását igénylik, amely rendkívül időigényessé tehetik a folyamatot.

Vannak azonban olyan csomagok, amelyek leegyszerűsítik a fenti folyamatot, és az ígérik, hogy akár egyetlen függvényhívás outputjából egy teljes statisztikai hipotézisvizsgálat eredményét ki tudjuk olvasni. Ilyen csomag a `{jmv}`, amely a grafikus felhasználói felülettel rendelkező *jamovi* statisztikai programcsomag funkcióinak elérését biztosítja számunkra R-ből. A *jamovi* külön oldalt tart fent, amely bemutatja a `{jmv}` csomag aktuálisan elérhető funkcióit. Jelenleg több olyan függvény érhető el, amely a könyvünkben bemutatott próbákhoz szorosan kapcsolódik, és komplett elemzések végrehajtását támogatja. Ezek a következők:

- `descriptives()` - leíró statisztikai elemzés,
- `ttestOneS()` - egymintás t-róba és egymintás Wilcoxon-próba,
- `ttestPS()` - páros t-róba és páros Wilcoxon-próba,
- `ttestIS()` - kétmintás t-próba, Welch-féle d próba és Mann–Whitney próba,
- `anovaRM()` - összetartozó mintás egyszempontos varianciaelemzés,
- `anovaOneW()` - egyszempontos varianciaelemzés és Welch-féle varianciaelemzés,
- `anovaNP()`- Kruskal–Wallis próba,
- `anovaRMNP()` - Friedman-próba,
- `corrMatrix()` - korrelációszámítás,
- `linReg()` - regressziószámítás,
- `propTest2()` - binomiális-próba,
- `propTestN()`- illeszkedésvizsgálat khí-négyzet próbával,
- `contTables()` - kapcsolatvizsgálat khí-négyzet próbával,
- `contTablesPaired()` - McNemar-próba, McNemar–Bowker próba.

A fenti listában az egyes függvényhívások fő statisztikai próbáit neveztük meg, de az output jóval gazdagabb a központi próba eredményénél. Éppen ez az újdonság, miszerint a próbával kapcsolatos összes szóba jöhető elemző tevékenység egyetlen függvényhívással kiíratható.

Két példán mutatjuk be ezt a lehetőséget a korábban bemutatott `df02` és `df22` adatbázisokon. Az első példában a `ttestIS()` függvényhívásával egy kétmintás t-próbát hajtunk végre, amelyhez a Welch-féle d próba és a Mann–Whitney próba is hozzátartozik. A második példában a `contTables()` függvényhívásával egy kapcsolatvizsgálatot hajtunk végre, amelyhez kérhetjük többek között a soronkénti százalékos eloszlást és a Fisher-féle egzakt próbát is.

Kezdjük a két független csoport összehasonlítását a `ttestIS()` függvényhívásával.

```
library(jmv)
ttestIS(formula = pontszam~modszer,
        data=df02,
        welchs = TRUE,
        mann = TRUE,
        norm = TRUE,
        eqv = TRUE,
        desc = TRUE
        )
#>
#>  INDEPENDENT SAMPLES T-TEST
#>
#>  Independent Samples T-Test
#>  ──────────────────────────────────────────────────────────────
#>                                  Statistic    df     p
#>  ──────────────────────────────────────────────────────────────
#>    pontszam    Student's t            0.13    6.0    0.90
#>                Welch's t              0.13    4.9    0.90
#>                Mann-Whitney U          6.0           0.69
#>  ──────────────────────────────────────────────────────────────
#>    Note. H□ μ□<sub>A</sub> ≠ μ□<sub>B</sub>
#>
#>
#>  ASSUMPTIONS
#>
#>  Normality Test (Shapiro-Wilk)
#>  ──────────────────────────────
#>                W       p
#>  ──────────────────────────────
#>    pontszam    0.95    0.68
#>  ──────────────────────────────
#>    Note. A low p-value
#>    suggests a violation of
```

```
#>    the assumption of
#>    normality
#>
#>
#>  Homogeneity of Variances Test (Levene's)
#>  ──────────────────────────────────────────
#>                F       df    df2    p
#>  ──────────────────────────────────────────
#>    pontszam    0.34     1      6    0.58
#>  ──────────────────────────────────────────
#>    Note. A low p-value suggests a
#>    violation of the assumption of
#>    equal variances
#>
#>
#>  Group Descriptives
#>  ──────────────────────────────────────────────────────────────
#>                Group    N    Mean    Median    SD    SE
#>  ──────────────────────────────────────────────────────────────
#>    pontszam    A        4      40        40    23     11
#>                B        4      38        32    14    6.8
#>  ──────────────────────────────────────────────────────────────
```

Két nominális változó kapcsolatát a `contTables()` függvénnyel vizsgálhatjuk.

```
# átnevezzük a szinteket
levels(df22$dohanyzasi_szokas) <- c("alkalmi", "naponta", "nem doh.")

# kontingencia táblázat
jmv::contTables(
    formula = ~ korosztaly:dohanyzasi_szokas,
    data = df22,
    pcRow = TRUE,
    fisher = TRUE,
    phiCra = TRUE,
    gamma = TRUE,
    taub = TRUE,
    exp = TRUE
)
#>
```

```
#>  CONTINGENCY TABLES
#>
#>  Contingency Tables
#>  ──────────────────────────────────────────────────────────────────────────
#>    korosztaly                    alkalmi    naponta    nem doh.    Total
#>  ──────────────────────────────────────────────────────────────────────────
#>    fiatal        Observed              1          1           1        3
#>                  Expected           0.69       0.92        1.4      3.0
#>                  % within row         33         33          33      100
#>
#>    idős          Observed              1          2           1        4
#>                  Expected           0.92       1.23        1.8      4.0
#>                  % within row         25         50          25      100
#>
#>    középkorú     Observed              1          1           4        6
#>                  Expected           1.38       1.85        2.8      6.0
#>                  % within row         17         17          67      100
#>
#>    Total         Observed              3          4           6       13
#>                  Expected           3.00       4.00        6.0     13.0
#>                  % within row         23         31          46      100
#>  ──────────────────────────────────────────────────────────────────────────
#>
#>
#>  χ² Tests
#>  ──────────────────────────────────────────────
#>                            Value    df    p
#>  ──────────────────────────────────────────────
#>    χ²                        2.2     4    0.71
#>    Fisher's exact test                    0.87
#>    N                          13
#>  ──────────────────────────────────────────────
#>
#>
#>  Nominal
#>  ──────────────────────────────
#>                       Value
#>  ──────────────────────────────
#>    Phi-coefficient      NaN
#>    Cramer's V          0.29
```

```
#>  ──────────────────────────
#>
#>
#>  Gamma
#>  ──────────────────────────────────────────────────
#>    Gamma    Standard Error    Lower    Upper
#>  ──────────────────────────────────────────────────
#>     0.41              0.34    -0.27      1.0
#>  ──────────────────────────────────────────────────
#>
#>
#>  Kendall's Tau-b
#>  ──────────────────────────────────
#>    Kendall's Tau-B    t      p
#>  ──────────────────────────────────
#>                0.28    1.1    0.28
#>  ──────────────────────────────────
```

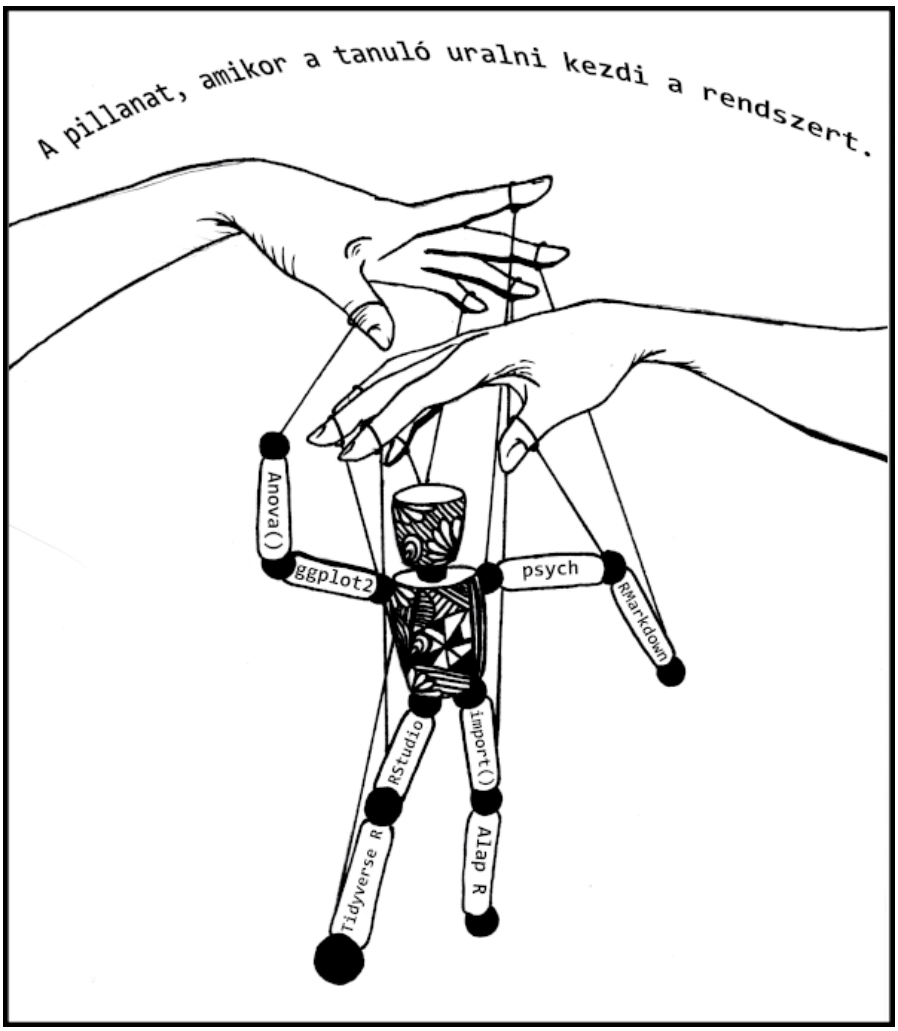
Összefoglalás

A statisztikai hipotézisvizsgálatok gyakran több lépésből állnak: leíró statisztikák, feltételvizsgálatok, hatásméret-számítás, utóvizsgálatok. Ezek rendszerint több csomag és függvény használatát igénylik, ami időigényes és bonyolult lehet. A `{jmv}` csomag ezt a folyamatot egyszerűsíti: egy függvényhívással komplex statisztikai elemzést végezhetünk, amely nemcsak a fő statisztikai próbát tartalmazza, hanem automatikusan mellékeli az ahhoz kapcsolódó információkat is.

Feladatok

1. Keressünk a `{jmv}` csomaghoz hasonló példákat, olyan csomagokat, amelyek egyetlen függvényhívással komplett statisztikai elemzést végeznek. Készítsünk egy rövid összefoglalót a csomagokról és a funkcióikról!
2. Végezzük el az előző fejezetekben bemutatott próbákat a `{jmv}` csomag segítségével! Készítsünk egy statisztikai jelentést (HTML riportot) a kapott eredményekről (lásd 11. fejezet)!

11. Publikáció



A statisztikai adatelemzés hosszadalmas folyamatának utolsó, és egyben legizgalmasabb lépése a kapott eredmények megosztása másokkal. Nem meglepő, hogy a statisztikai programcsomagok ezt a publikációs lépést is támogatják. Az *Alap R* a *Quarto* dokumentumkezelő rendszerrel kiegészítve kiváló eszköz publikációkész riportok elkészítésére. Ráadásul mindezt a reprodukálható kutatás maximális támogatása mellett végezhetjük. Egyetlen dokumentumban transzparens módon eltárolhatjuk a

- kutatási kérdéseinket és hipotéziseinket,
- az adatbeolvasó, előkészítő és elemző R parancsainkat,
- a kutatási eredményeket bemutató ábráinkat és táblázatainkat,
- valamint a szöveges magyarázatainkat és hivatkozásainkat,

amelyek teljes értékűvé teszik a kutatási beszámolónkat. Ezen dokumentum és az adatállományok birtokában bárki képes lesz a kutatási eredményeink későbbi reprodukálására.

A *Quarto* egy nyílt forráskódú publikációs rendszer, amely lehetővé teszi tudományos dokumentumok, prezentációk, weboldalak, blogok és könyvek létrehozását. A *Quarto* támogatja többek között az R és Python programozási nyelveket, így dinamikus és reprodukálható tartalmak készítését is lehetővé teszi.

A *Quarto* a *Pandoc markdown*-t használja a természetes nyelvű szövegek formázott létrehozásához, támogatja a LaTeX képleteket és a hivatkozásokat is. Ezen felül a *Quarto* kereszthivatkozásokkal segíti az ábrák és táblázatok szövegbe integrálását.

A *Quarto* támogatja a különböző formátumokba történő exportálást, beleértve a *HTML*, *PDF*, *MS Word* és *ePub* formátumokat, így a felhasználók széles körben terjeszthetik munkáikat. A *Quarto* célja, hogy egységes és rugalmas platformot biztosítson dokumentumok erőfeszítés nélküli létrehozására, elősegítse a tudás megosztását és a reprodukálható kutatást.

11.1. Quarto HTML dokumentum 🙂

> **i** Miről lesz szó? Ebben a fejezetben
>
> - bemutatjuk a *Quarto* dokumentum felépítését és
> - a HTML dokumentum létrehozása során használható beállításokat.

Korábban megállapítottuk, hogy a parancsállományok `.R` kiterjesztésű szöveges fájlok, amelyek R parancsokat vagy `#` jellel kezdődő megjegyzéssorokat tartalmaznak. Az *RStudio* azonban további lehetőséget kínál az R kód és a természetes nyelvű szöveg integrált tárolására. Ezek közé tartozik a *Quarto* dokumentum, amely az *R Markdown* formátum közvetlen

utódja. Ebben a könyvben kizárólag a *Quarto* formátumot tárgyaljuk, amely jelentős átfedést mutat az *R Markdown* dokumentumokkal, de számos új funkcióval is kiegészíti elődjét.

Hozzunk létre egy új *Rstudio* projektet, és benne egy *Quarto* állományt a `File / New File / Quarto Document...` menüpont segítségével. A dialógusdobozban felkínált opciókat fogadjuk el. A `Source` panelben egy alapértelmezett tartalommal kitöltött *Quarto* állományt látunk. Helyettesítsük azt a következő sorokkal:

````
---
title: "Kutatási beszámoló"
date: 2025-04-30
format: html
---

# Bevezetés

Ebben a dokumentumban a `tidyverse` csomag `starwars` adatkészletét fogjuk elemezni.

# Adatok betöltése és előkészítése

```{r}
#| label: setup
library(tidyverse)

# Az adatok áttekintése
glimpse(starwars)
```

# Alapvető leíró statisztikák

Az adatkészlet **`r nrow(starwars)`** karakter adatait tartalmazza.

```{r}
#| label: summary-stats
summary(starwars |> select(height, mass, birth_year))
```

## Magasság és tömeg közötti kapcsolat

Vizsgáljuk meg a karakterek magassága és tömege közötti kapcsolatot egy ábrával:
````

````
```{r}
#| label: height-mass-plot
#| fig-cap: "A Star Wars karakterek magassága és tömege közötti kapcsolat"
ggplot(starwars, aes(x = height, y = mass)) + geom_point(na.rm = TRUE) +
  labs(title = "Magasság és tömeg", x = "Magasság (cm)", y = "Tömeg (kg)") +
  theme_minimal()
```
````

Mentsük el az állományt `quarto_pelda.qmd` néven, majd nyomjuk meg a `Ctrl-Shift-K` billentyűkombinációt (használhatjuk a `Render` gombot is a panel tetején). Ezzel a `.qmd` állomány fordítását (renderelését) kezdeményezzük HTML formátumba. A jobb alsó panelben meg is jelenik a HTML dokumentum, és a projekt könyvtárunkban a `quarto_pelda.html` állomány is létrejött. A HTML állomány tartalmazza a formázott természetes nyelvű szöveget, az R input parancsokat, valamint az R parancsok outputját, legyen az szöveges vagy ábra jellegű.

A *Quarto* állományok `.qmd` kiterjesztésű egyszerű szöveges állományok (az *R Markdown* állományok `.Rmd` kiterjesztésűek voltak). Egy *Quarto* állomány három részből épül fel:

- minden *Quarto* állomány egy *fejrésszel* kezdődik, amit a --- sorok határolnak,
- az állományban a fejrész után bárhol elhelyezhetünk *természetes nyelvű szöveget*, amelyet a *Pandoc markdown* szabályai szerint formázhatunk,
- a természetes nyelvű szövegek között, bárhol, R parancsok is elhelyezhetők, de azokat speciális határolók (```) közé kell írnunk, ezek az ún. *R csonkok*.

11.1.1. A fejléc

A *Quarto* fejléce az állomány metaadatait írja le YAML formátumban. Egy szokásos, kicsit kibővített fejléc, magyar nyelvű HTML állományok létrehozásához a következő lehet:

```
---
title: "Kutatási beszámoló"
subtitle: "A kérdőíves vizsgálat tanulságai 2021-2024"
author: "Abari Kálmán"
date: today
execute:
  echo: true
  warning: false
editor_options:
  chunk_output_type: console
format:
```

```
  html:
    theme: Cerulean
    title-block-banner: true
    embed-resources: true
    self-contained-math: true
    toc: true
    toc-location: left
    toc-expand: 2
    number-sections: true
    number-depth: 3
lang: hu
author-title: Szerző
published-title: Dátum
toc-title: Tartalomjegyzék
---
```

A fejléc a dokumentum címét (`title`), alcímét (`subtitle`) és szerzőjét (`author`) állítja be, majd a dátumot az aktuális napi dátumra állítja a `date: today` megadásával. Az `execute` attribútum alatti

- `echo: true` engedélyezi a kód megjelenítését (alapértelmezés szerint csak a kimenet jelenik meg),
- `warning: false` letiltja a figyelmeztető üzenetek megjelenítését.

A `chunk_output_type: console` azt határozza meg, hogy amikor a kódot az *RStudio*-ban futtatjuk (azaz amikor a `Ctrl-Shift-R` billentyűkombinációval "csak" futtatjuk, és nem a `Ctrl-Shift-K` billentyűkombinációval rendereljük), a kimenet szokásos módon az *RStudio* konzoljában jelenjen meg. A másik lehetőség a `chunk_output_type: inline`, amely a kimenetet a kód alatt jeleníti meg. Ezt az opciót lehetőség szerint ne használjuk, mivel a kimenet megjelenítése a konzolban sokkal kényelmesebb és áttekinthetőbb.

A generált állomány formátumára vonatkozó közvetlen beállítások a `format:` alatt találhatók:

- `html:` - a dokumentum HTML formátumban fog létrejönni, a további beállítások mind erre a HTML állományra vonatkoznak,
- `theme: Cerulean` - a HTML dokumentum megjelenésének témája `Cerulean` lesz, amely egyike a *Quarto* által támogatott Bootswatch témáknak, amelyek előre definiált stílusokat és színvilágot biztosítanak a weboldalak számára,
- `title-block-banner: true` - engedélyezi a címblokk mögötti háttérszín beállítását a dokumentum elején, amely most a `theme` értékétől fog függeni,

- `embed-resources: true` - az erőforrásokat (például képeket, stíluslapokat, betűtípusokat) beágyazza a HTML fájlba, így nem lesz szükség külön külső fájlokra; ez a beállítás kulcsfontosságú az önálló és könnyen hordozható HTML állomány előállításához,
- `self-contained-math: true` - a matematikai képleteket beágyazza, így azok offline is megjeleníthetők lesznek,
- `toc: true` - a HTML dokumentum tartalomjegyzéket fog megjeleníteni,
- `toc-location: left` - a tartalomjegyzék a bal oldalon fog elhelyezkedni,
- `toc-expand: 2` - a tartalomjegyzékben az első két szint automatikusan kibontva jelenik meg,
- `number-sections: true` - a dokumentum címeit számozással látjuk el,
- `number-depth: 3` - a számozás mélysége maximum 3 szint mélységig terjed (pl. 1., 1.1., 1.1.1.).

A YAML fejléc utolsó része a nyelvi és metaadat beállításokkal kapcsolatos:

- `lang: hu` - a dokumentum nyelve magyar lesz,
- `author-title: Szerző` - az "Author" címke helyett "Szerző" jelenik meg a HTML dokumentumban,
- `published-title: Dátum` - a publikálás dátumát "Dátum" címkével jelöljük,
- `toc-title: Tartalomjegyzék` - a tartalomjegyzék címe magyarul fog szerepelni, azaz "Tartalomjegyzék" lesz.

A fenti YAML konfiguráció biztosítja számunkra, hogy a *Quarto* dokumentum egy szépen formázott, magyar nyelvű, számozott fejezetekkel és bal oldali tartalomjegyzékkel rendelkező HTML fájl legyen, amely offline is teljesen működőképes lesz.

11.1.2. Pandoc markdown

A *Quarto* dokumentum természetes nyelvű szövegrészét a *Pandoc markdown* szabályai szerint formázhatjuk. A *Pandoc markdown* a széles körben használt *Markdown* formanyelvet egészíti ki néhány új funkcióval, azonban a cél közös: formázott szöveg létrehozása a lehető legegyszerűbben. Mindössze néhány speciális karakterrel kell kiegészítenünk a szöveget, és fordítás után máris kész a formázott dokumentum. Ilyen karakterek például a `#` a címek formázásához, a `*` és a `**` a *dőlt* és **félkövér** szöveghez, a `~~` az ~~áthúzott~~ szöveghez, a `^` és a `~` a felső- és alsóindexekhez, a `|` a táblázatokhoz, a `[]()` a hivatkozásokhoz, a `![ ]()` a képek beillesztéséhez, a `$` a LaTex képletekhez (például a mintabeli szórás kiszámítása: $s = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$), és így tovább.

Egy rövid összefoglalót találunk a következő részben a *Pandoc markdown* szabályairól, de a teljes leírásért látogassuk meg a Pandoc dokumentációját. Érdemes a következő részt a *Quarto*

dokumentumunkban is kipróbálni, azaz másoljuk be a következő sorokat a `quarto_pelda.qmd` állományba (a fejléc után), majd rendereljük le a dokumentumot a szokásos billentyűkombinációval: `Ctrl-Shift-K`. Legyünk óvatosak, az utolsó sorban lévő kép beillesztése csak akkor fog sikerülni, ha a projektkönyvárunkban gondoskodunk egy `images` könyvtárról és benne egy `quarto.png` képállományt is elhelyezünk. A kívánt cél elérése érdekében javasolt a fenti lépéseket végrehajtani, ellenkező esetben töröljük a `![Quarto logó](images/quarto.png)` sort a *Quarto* dokumentumból.

```
# Szöveg formázása

*dőlt* **félkövér** ~~áthúzott~~ `kód`

felsőindex^2^ alsóindex~2~

# Címek

# 1. szintű cím (főcím)

## 2. szintű cím (alcím)

### 3. szintű cím (al-alcím)


# Felsorolások

- Egyszerű listaelem

- Második listaelem

    - elem 2.1

    - elem 2.2.

- Harmadik listaelem

# Számozás

1. Első elem

2. Második elem\
   Ide írhatunk bármit, az a 2. elemhez fog tartozni
```

```
3. Harmadik elem


# Képletek (LaTeX szintaxis szerint)


- Inline matematikai képlet: $E = mc^2$

- Blokkszintű képlet\
  $$\sum_{i=1}^{n} x_i$$

# Táblázatok

| Név  | Kor | Város   |
|------|----|----------|
| Anna | 25 | Budapest |
| Béla | 30 | Debrecen |

# Hivatkozások és képek

Egyszerű link: <https://google.com>

Link beillesztése: [Google](https://google.com)

Kép beillesztése: ![Quarto logó](images/quarto.png)
```

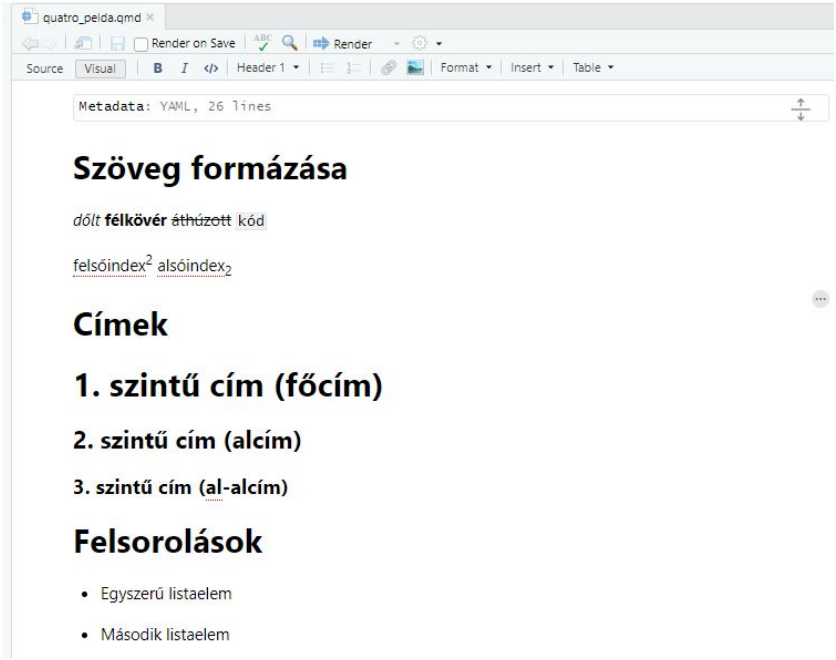
A formázási lehetőségek magabiztos használata érdekében érdemes a fenti sorokat módosítani, újabb felsorolt elemeket, hivatkozásokat, formázott szövegrészeket is létrehozni. Fontos a táblázatok és a képletek használatának gyakorlása is, mivel ezek a leggyakrabban használt elemek a tudományos dokumentumokban.

11.1.2.1. A vizuális szerkesztő

Egy vizuális szerkesztőt is biztosít számunkra az *RStudio*, amely segítségével a *Quarto* dokumentumokat még egyszerűbben szerkeszthetjük. Ha már magabiztosan írunk *Markdown* dokumentumokat, akkor a vizuális szerkesztő használatára könnyű lesz áttérni, hiszen nem különbözik lényegesen a *Google Docs* vagy az *MS Word* használatától.

Ha korábban még nem használtunk *Markdown* típusú dokumentumokat, akkor a vizuális szerkesztő használata segíthet a *Markdown* formanyelv gyorsabb elsajátításában is. Mégis azt javasoljuk, hogy a *Quarto* dokumentumok írásához kezdetben inkább a szöveges szerkesztőt használjuk, és csak akkor térjünk át a vizuális szerkesztőre, ha már magabiztosan használjuk a *Markdown* formanyelvet. A vizuális szerkesztő használata ugyanis bizonyos esetekben több időt emészt fel, és a *Markdown* formanyelv elsajátítása sokkal hasznosabb lehet a későbbiekben.

A *Quarto* dokumentumok szerkesztése közben a panel bal felső sarkában tudunk a `Source` és a `Visual` nézet között váltani. A `Source` nézetben a szöveges szerkesztőt látjuk, a `Visual` nézetben pedig a formázott dokumentumot (11.1 ábra). A `Visual` nézetben a dokumentumot a böngészőben látható formában tekinthetjük meg, és a változtatások azonnal láthatóak lesznek. Erre látunk példát a 11.1 ábrán, ahol a korábban elkészített `quarto_pelda.qmd` állományt a vizuális szerkesztővel szerkesztjük.



11.1. ábra: Vizuális szövegszerkesztő az *RStudio*-ban (Posit team, 2025)

11.1.3. R csonkok

Az R csonkok teszik igazán dinamikussá, élővé a *Quarto* dokumentumot. A HTML létrehozása során ugyanis az R csonkba írt parancsok is végrehajtódnak, és a futási eredmények is bekerülnek a végső dokumentumba, legyen az egy szöveges mutató, táblázat vagy ábra.

R csonkot a természetes nyelvű szövegek közé bárhová beszúrhatunk, ennek 3 módját választhatjuk:

- billentyűparanccsal: `Ctrl-Alt-I`,
- az *Insert* gombbal az *RStudio* szerkesztő eszköztárában,
- kézi gépeléssel a csonk kezdő és záró sorával:
  - kezdősor: ```` ```{r} ````
  - zárósor: ```` ``` ````

Nézzünk egy példát egy egyszerű R csonkra:

````
```{r}
#| label: gyakorlas-01

1 + 1  # összeadás

# egy ábra
plot(women)
```
````

Egy R csonk tehát speciális kezdő és záró sorral rendelkezik, a kettő között található a csonk tartalma. A csonk tartalma két részből áll:

- tetszőleges számú csonkbeállítással kezdhetünk, melyek mindegyikét a `#|` karakterkombináció vezeti be,
- ezt követi egy vagy több R parancs, de akár megjegyzések is beszúrhatunk a szokásos `#` karakterrel kezdődően.

A fenti példában a csonk címkéjét állítjuk be (`#| label: gyakorlas-01`), ami lényegében a csonk azonosítását végzi. Fontos tudni, hogy egyedi értékre van szükségünk a teljes *Quarto* állományra nézve, nem fordulhat elő duplikáció a címke nevében. A csonk címkéjének beállítása legalább két okból fontos:

- egy hosszabb *Quarto* állományban könnyebben navigálhatunk a különböző csonkok között a szerkesztő panel alján található legördülő kódnavigátor segítségével,
- táblázatokat és ábrákat azonosíthatunk ezzel a címkével, így kereszthivatkozások létrehozásához használhatjuk fel őket.

A `label` beállítás mellett számos további opció is rendelkezésre áll, melyek segítségével testre szabhatjuk a csonkot. A több tucat beállítási lehetőségek közül azok a legfontosabbak, amelyek a csonkban lévő kód futtatását és az eredmények megjelenítését szabályozzák:

11.1. táblázat: A csonkban megadható beállítás, mely kimeneteket tiltja le *(saját szerkesztés)*

Beállítás	Kód futása	Kód	Output	Ábra	Üzenet	Figyelmeztetés
`eval: false`	X		X	X	X	X
`include: false`		X	X	X	X	X
`echo: false`		X				
`results: hide`			X			
`fig-show: hide`				X		
`message: false`					X	
`warning: false`						X

- `eval: false` - megakadályozza a kód futtatását, így eredmény sem fog megjelenni,
- `include: false` - futtatja a kódot, de nem jeleníti meg az R kódot vagy az outputot a végső dokumentumban,
- `echo: false` - megakadályozza az R kód megjelenítését a végső dokumentumban,
- `message: false` és `warning: false` - megakadályozza az üzenetek és figyelmeztetések megjelenítését a végső dokumentumban,
- `results: hide` és `fig-show: hide` - megakadályozza a kimenetek és ábrák megjelenítését a végső dokumentumban.

A 11.1 táblázat összefoglalja, mely kimeneteket tiltják le az egyes beállítások a csonk elején.

Ha a végső HTML állományban csak egy ábrát szeretnénk megjeleníteni, akkor a szokásos csonkmegjelenés a következő lehet:

````
```{r}
#| label: fig-egyszeru-abra-01
#| fig.cap: "Egy egyszerű ábra"
#| echo: false
#| message: false
#| warning: false

# egy ábra
plot(women)
```
````

A fenti példában tiltjuk az R kód megjelenítését (`echo: false`), az üzenetek és figyelmeztetések megjelenítését (`message: false`, `warning: false`), így csak az ábra fog megjelenni a végső HTML állományban. A csonk címkéje (`label`) a későbbi kereszthivatkozásokhoz szükséges, a `fig.cap` beállítás pedig az ábra címét adja meg.

Összefoglalás

A *Quarto* dokumentumok három részből állnak: a fejrészből, a természetes nyelvű szövegből és az R csonkokból. A fejrészben a dokumentum metaadatait állítjuk be, a természetes nyelvű szövegben a *Pandoc markdown* szabályait követve formázhatjuk a szöveget, az R csonkokban pedig R parancsokat futtathatunk és azok outputját megjeleníthetjük a végső HTML dokumentumban. Az R csonkokban számos beállítás segítségével testre szabhatjuk a kód futtatását és az output megjelenítését.

Feladatok

1. Hogyan állíthatjuk be a *Quarto* HTML dokumentum számára a fejezetet sorszámozásánál a pontra végződést, azaz az "1" és "1.1" helyet az "1." és "1.1." megjelenítését.
2. Milyen témákat használhatunk még a HTML dokumentumok megjelenítése során a fejezetben használt `Cerulean` témán kívül?
3. Hol találunk részletes leírást a *Markdown* és a *Quarto* dokumentumok formázási lehetőségeiről?
4. Miért mondhatjuk, hogy a *Quarto* dokumentum egy fordított parancsállomány?
5. Mit jelent az inline R kód futtatása a *Quarto* dokumentumokban, hogyan használjuk ezeket? Nézzen utána a *Quarto* dokumentációjában!

11.2. Hivatkozások 🤔

Miről lesz szó? Ebben a fejezetben

- megismerjük a *Quarto* dokumentumokban használható szövegközi hivatkozások és a dokumentum végén megjelenő irodalomjegyzék létrehozásának szabályait, valamint
- a dokumentumon belüli kereszthivatkozásokat táblázatokra és ábrákra.

Egy tudományos írás nem nélkülözheti a bibliográfiai hivatkozásokat, hiszen ezek segítségével tudjuk a kutatásunkat a korábbi eredményekhez kapcsolni, és a saját eredményeinket a szakirodalommal összehasonlítani. Legalább ennyire fontos az is, hogy a saját a táblázatainkra és ábráinkra is megfelelően hivatkozzunk az írásunkban, hiszen ezek is a tudományos írások alapvető elemei. Ebben a fejezetben tehát a *Quarto* dokumentumokban használható hivatkozásokat ismerjük meg.

11.2.1. Bibliográfia

A tudományos írásokban két helyen fordulnak elő a hivatkozások:

- a szövegközi hivatkozásként: a hivatkozásokat a szövegben a hivatkozott gondolat vagy eredmény után zárójelben vagy a szerző neve után zárójelben adjuk meg,
- a bibliográfiában: az írásunk végén a hivatkozott művek teljes listáját adjuk meg, amely tartalmazza a hivatkozott művek szerzőit, címét, kiadási évét és egyéb fontos adatokat.

Tegyük fel, hogy a tudományos írásunkban egy cikkre és egy könyvre szeretnénk a fenti módon hivatkozni. A következő lépések szükségesek a hivatkozások megadásához:

1. Összeállítjuk a bibliográfiát egy `.bib` fájlban. Ehhez érdemes igénybe venni valamilyen hivatkozáskezelőt, amely nagyban megkönnyíti a hivatkozások kezelését. Ilyen lehet a *Zotero*, a *Mendeley* vagy a *JabRef*.
2. A `.bib` fájlt bemásoljuk a projektkönyvtárunkba és a *Quarto* dokumentum fejrészében a `bibliography:` beállítással hivatkozunk rá.
3. Gondoskodunk a hivatkozási stílus beállításáról, amely a `csl:` beállítással történik, szintén a fejrészben.
4. A szövegközi hivatkozásokat a `@` karakterrel és a hivatkozás azonosítójával adjuk meg, ahol az azonosító a `.bib` fájlban található egyedi azonosító.
5. A tudományos írás végén megjelenő bibliográfia generálásáról automatikusan gondoskodik a *Quarto* dokumentum, de ennek a beállítását is finomíthatjuk.

Nézzük a fenti lépéseket a gyakorlatban. Először is készítsünk egy `mestint.bib` fájlt és mentsük el a projektkönyvtárunkba. Legalább a következő két hivatkozást helyezzük el benne:

```
@Book{harari2015sapiens,
  title={Sapiens. Az emberiség rövid története},
  author={Harari, Yuval Noah},
  year={2023},
  publisher={Animus Kiadó}
}

@Article{Dillion2023,
  author       = {Dillion, Danica and Tandon, Niket and Gu, Yuling and Gray, Kurt},
  date         = {2023},
  journaltitle = {Trends in Cognitive Sciences},
  title        = {Can AI language models replace human participants?},
  doi          = {10.1016/j.tics.2023.04.008},
```

```
  issn         = {1364-6613},
  number       = {7},
  pages        = {597--600},
  volume       = {27},
  publisher    = {Elsevier BV},
}
```

A `.bib` fájlokban minden egyes publikációhoz egyedi azonosítót kell megadni, amelyet a hivatkozásokban használunk majd. Ezek most a `harari2015sapiens` és a `Dillion2023` azonosítók lesznek.

Egészítsük ki a `quarto_pelda.qmd` állomány fejlécét a következő beállításokkal:

```
bibliography: mestint.bib
csl: https://raw.githubusercontent.com/citation-style-language/styles/master/apa.csl
```

A `bibliography:` beállítás a `.bib` fájlra mutat, amely a bibliográfiai adatokat tartalmazza, a `csl:` beállítás pedig a hivatkozási stílust határozza meg. A hivatkozási stílusokat a Citation Style Language weboldalról tölthetjük le, ahol több száz különböző stílus közül választhatunk. Most online módon írjuk elő az APA stílust használatát.

A szövegközi hivatkozásokat tipikusan `[@azonosító]` formátumban adjuk meg a szövegben, ahol a forrásazonosító a `.bib` fájlban található egyedi azonosító, de léteznek olyan variációk, amelyek többféle szövegközi hivatkozást is lehetővé tesznek. Helyezzük el ezeket a sorokat a `quarto_pelda.qmd` állományban:

```
- A szokásos szövegközi hivatkozás [@harari2015sapiens].
- Akár több műre is hivatkozhatunk a szövegben [@harari2015sapiens; @Dillion2023].
- Megadhatunk oldalszámot is [@Dillion2023, p. 2].
- Megadhatunk több oldalt is [@harari2015sapiens, pp. 12-15].
- Megadhatunk tetszőleges szöveget is [lásd @harari2015sapiens, pp. 12-15].
- Más jellegű szövegközi hivatkozások is megadhatók:
    - @harari2015sapiens
    - @Dillion2023 [p. 2]
    - Harari népszerű könyvében [-@harari2015sapiens] megjegyzi
```

Fordítás után a HTML állományban ezek a sorok a 11.2 ábrán látható módon jelennek meg.

Ha végiglapozzuk a fordítás után kapott HTML állományt, akkor a dokumentum végén a bibliográfiai listát is megtaláljuk. Azonban direkt módon is gondoskodhatunk az irodalomjegyzékről, amelyet a következő módon hozhatunk létre a *Quarto* dokumentumokban:

- A szokásos szövegközi hivatkozás (Harari, 2023).
- Akár több műre is hivatkozhatunk a szövegben (Dillion és mtsai., 2023; Harari, 2023).
- Megadhatunk oldalszámot is (Dillion és mtsai., 2023, p. 2).
- Megadhatunk több oldalt is (Harari, 2023, pp. 12-15).
- Megadhatunk tetszőleges szöveget is (lásd Harari, 2023, pp. 12-15).
- Más jellegű szövegközi hivatkozások is megadhatók:
  - Harari (2023)
  - Dillion és mtsai. (2023, p. 2)
  - Harari népszerű könyvében (2023) megjegyzi

11.2. ábra: Szövegközi hivatkozások a HTML szövegben *(saját szerkesztés)*

```
# Irodalomjegyzék

::: {#refs}
:::
```

A 11.3 ábra a fenti kódrészlet hatását mutatja be. Az irodalomjegyzék így már felveszi a dokumentumban használt formázási stílust.

12. Irodalomjegyzék

Dillion, D., Tandon, N., Gu, Y., & Gray, K. (2023). Can AI language models replace human participants? *Trends in Cognitive Sciences*, *27*(7), 597–600. https://doi.org/10.1016/j.tics.2023.04.008

Harari, Y. N. (2023). *Sapiens. Az emberiség rövid története*. Animus Kiadó.

11.3. ábra: Bibliográfia a tudományos szöveg végén *(saját szerkesztés)*

11.2.2. Táblázatok

Táblázatok létrehozásához alapvetően két megközelítést használhatunk a *Quarto* dokumentumokban:

- A *Pandoc markdown* szabályai alapján, vagy
- R csonk segítségével is létrehozhatunk táblázatokat.

A *Pandoc markdown* szabályait követve táblázatokat például a következő módon hozhatunk létre:

```
| Név   | Kor | Város    |
|-------|-----|----------|
| Anna  | 25  | Budapest |
| Béla  | 30  | Debrecen |

: Egy egyszerű táblázat (1) {#tbl-st-01}
```

11.2. táblázat: Egy egyszerű táblázat (1)

Név	Kor	Város
Anna	25	Budapest
Béla	30	Debrecen

A fenti példában egy egyszerű táblázatot hozunk létre a *Pandoc markdown* szabályai szerint. A táblázatokat a | karakterekkel és a --- sorokkal hozhatjuk létre, ahol a | karakterekkel határoljuk a cellákat, a --- sorok pedig a cellák fejlécét határozzák meg. A táblázat címét és címkéjét a : és a `{#}` karakterrel adjuk meg, amely a későbbi kereszthivatkozásokhoz szükséges.

R csonk segítségével táblázatot például így hozhatunk létre:

```
```{r}
#| label: tbl-st-02
#| tbl-cap: "Egy egyszerű táblázat (2)"
#| message: false
#| warning: false

data.frame(Név = c("Anna", "Béla"),
               Kor = c(25, 30),
               Város = c("Budapest", "Debrecen")) |>
  knitr::kable(escape = F, booktabs = T, align="lcc")
```
```

11.3. táblázat: Egy egyszerű táblázat (2)

Név	Kor	Város
Anna	25	Budapest
Béla	30	Debrecen

A fenti példában egy egyszerű táblázatot hozunk létre az R csonkok segítségével. Táblázatokat a `knitr::kable()` függvény segítségével hozhatjuk létre, amely számos beállítási lehetőséget kínál a táblázatok testreszabására. A táblázat címét a `tbl-cap:` beállítással adjuk meg, amely a `label:` beállítás a későbbi kereszthivatkozásokhoz szükséges.

A fenti táblázatokra a szövegből a következő módon hivatkozhatunk:

```
A példákat a [-@tbl-st-01]. és a [-@tbl-st-02]. táblázatokban láthatjuk.
```

A példákat a 11.2. és a 11.3. táblázatokban láthatjuk.

A `{kintr}` csomag `kable()` függvényének tudása kibővíthető a `{kableExtra}` csomag számos további függvényével, amelyek igazán látványos és praktikus táblázatok létrehozására szolgálnak. Érdemes más táblázatkészítő csomagokat is megismerni, például a `{gt}` csomagot, amely mára szinte iparági szabvány lett a táblázatok készítésében.

11.2.3. Ábrák

Az ábrák létrehozásához két eltérő megközelítést használhatunk a *Quarto* dokumentumokban, attól függően, hogy rendelkezésre áll-e egy kész képállomány, vagy a képet a fordítási folyamat során, az R csonkba illesztett parancsok segítségével szeretnénk-e létrehozni.

Amennyiben a kész képállomány rendelkezésre áll, például a `quarto.png` állomány a projekt könyvtár `images` alkönyvtárában, akkor az ábra végső dokumentumba illesztésére még mindig 2 módunk van.

Az első módszer a *Pandoc markdown* szintaxis szerinti beillesztés, amely a legkevésbé rugalmas megoldás, de a legegyszerűbb is.

```
![Egy egyszerű ábra (1)](images/quarto.png){#fig-sf-01 width=20%}
```

11.4. ábra: Egy egyszerű ábra (1)

A második módszer szerint használhatjuk egy új R csonkban a `knitr::include_graphics()` függvényt is:

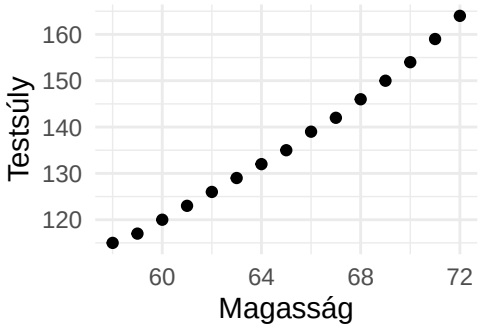
```{r}
#| label: fig-sf-02
#| fig.cap: "Egy egyszerű ábra (2)"
#| out.width: 15%
#| echo: false
#| message: false
#| warning: false

# egy ábra
knitr::include_graphics("images/quarto.png")
```

11.5. ábra: Egy egyszerű ábra (2)

Az eddigiektől lényegesen eltérő helyzet, amikor nem áll rendelkezésre képállomány. Ilyenkor a képet a fordítás során hozzuk létre, azaz ábra készítéséhez R kódot kell használunk:

```{r}
#| label: fig-sf-03
#| fig.cap: "Egy egyszerű ábra (3)"
#| fig.width: 2.6
#| fig.asp: 0.7
#| echo: false
#| message: false
#| warning: false

# egy ábra
library(ggplot2)
p1 <- ggplot(women, aes(x=height, y=weight)) +
  geom_point() +
  labs(x="Magasság", y="Testsúly") +
  theme_minimal()
p1
```

Bármelyik módszert is választjuk, az ábrákra a szövegből a következő módon hivatkozhatunk:



11.6. ábra: Egy egyszerű ábra (3)

```
A fenti 3 módszer eredménye a [-@fig-sf-01]., [-@fig-sf-02]. és [-@fig-sf-03]. ábrán.
```

A fenti 3 módszer eredménye a 11.4., 11.5. és 11.6. ábrán.

A fenti 3 ábrát 3 különféle módon hoztuk létre.

- A 11.4. ábrát a *Pandoc markdown* szabályai alapján szúrtuk be és a `{#fig-sf-01 width=20%}` beállítással azonosítót rendeltünk a képhez, valamint a szélességét is beállítottuk. Ez a képek beszúrásának legegyszerűbb módja, egyben a legkevesebb beállítási lehetőséget is kínálja.
- A 11.5. ábrát is kész képállományra alapoztuk, de ez már szélesebb körű beállítási lehetőséget rejt, köszönhetően az R csonk beállításainak. A következő ábrára vonatkozó beállításokat használtuk:
  - `label: fig-sf-02` - az ábra azonosítóját itt adtuk meg,
  - `fig.cap: "Egy egyszerű ábra (2)"` - az ábra címét itt adtuk meg,
  - `out.width: 15%` - az ábrát átméretezzük az erdeti kép méretének 15%-ára, és mivel a méretezés megőrzi a képarányt, ezért a szélesség megadása elegendő,
- A 11.6. ábrát az R csonkban futtatott parancsok segítségével hoztuk létre. Az ábra azonosítóját, a címét itt is megadtuk az R csonkban, továbbá:
  - a `fig.width: 2.6` - az ábra szélességét itt adtuk meg inch-ben mérve,
  - a `fig.asp: 0.7` - az ábra magasság/szélesség arányát itt adtuk meg.

Láttuk, hogy amennyiben R csonkot használunk az ábrák létrehozásához – akár kész kép, akár generáló R parancsok esetén –, a kép megjelenését számos módon befolyásolhatjuk. A fenti eseteken túl szokásos még:

- az ábrák igazítását a `fig.align` beállítással szabályozhatjuk, amely az ábra igazítását határozza meg (értékei: `left`, `right`, `center`),
- az ábrák átméretezése történhet a `fig.width` és a `fig.height` beállításokkal, amelyek az ábra szélességét és magasságát adják meg inch-ben mérve.

Az ábrák publikációját még egy függvény támogatja. A `{ggplot2}` csomag `ggsave()` függvényével a `ggplot()` parancsokkal készült ábrát a háttértárra menthetjük. A `p1` objektumunk épp egy ilyen ábrát tartalmaz. A `ggsave()` függvény a `filename=` argumentuma adja meg a fájl nevét, amelybe az ábrát menteni szeretnénk. Az állomány választott kiterjesztése fogja eldönteni a kimeneti képformátumot. A `plot=` paraméterben megadhatjuk, hogy melyik ábrát szeretnénk menteni. A `width=` és `height=` beállításokkal megadhatjuk az ábra szélességét és magasságát inch-ben mérve. A `dpi=` beállítással pedig a felbontást (dots per inch) adhatjuk meg, míg a `scale=` beállítással kicsinyíthetjük vagy nagyíthatjuk az ábrát.

A `p1` ábránk mentését a következő módon végezhetjük el:

```
# ábra mentése háttértárra
ggsave(filename = "output/images/egyszeru_abra_3.png", plot = p1, width = 5,
    height = 4, dpi = 300, scale = 0.9)
```

Összefoglalás

A *Quarto* dokumentumokban a szövegközi hivatkozások és a bibliográfiai hivatkozások létrehozásához szükség van az összegyűjtött irodalmakat tartalmazó `.bib` fájlra, amelyre a QMD fejrészében a `bibliography:` beállítással hivatkozunk. A hivatkozások stílusát a `csl:` beállítással határozzuk meg, amely a hivatkozási stílus fájlra mutat. A szövegközi hivatkozásokat tipikusan `[@citation-key]` formátumban adjuk meg a szövegben, az irodalomjegyzék helyét a `::: {#refs} :::` karaktersorozatok jelölik ki. Táblázatokat és ábrákat a *Pandoc markdown* szabályai szerint, vagy R csonk segítségével is létrehozhatunk. Utóbbi több lehetőséget kínál, ugyanis a csonk beállításai között a `#| label:` a táblázat/ábra azonosítóját a `|# tbl-cap:`/`|# fig.cap:` a táblázat/ábra címét adja meg. Magyar szövegből szokásos a `[-@tbl-key]` és `[-@fig-key]` formátumú hivatkozás az adott táblázatra, illetve ábrára.

Feladatok

1. A korábban említett `knitr::kable()` + `{kableExtra}` pár és a `{gt}` csomag mellett számos további csomag is segíti a táblázatok létrehozását. Melyek ezek?
2. A `{gt}` csomag használatával készítse el a fejezetben használt táblázatot!
3. Készítsen egy HTML dokumentumot, amely ábrával és képpel is bemutatja a top 10 legtöbb bevételt hozó Pixar filmet. Használjuk a Pixar Box Office Varázslat oldalon

található adatbázist és elemzési segítséget. A HTML dokumentum tartalmazzon hivatkozásokat az adatbázisra és a használt csomagokra, illetve magára az R-re is, továbbá a beillesztett képre és az ábrára is hivatkozzunk a szövegből. A dokumentum végén szerepeljen irodalomjegyzék is.

11.3. Más Quarto formátumok 😲

Miről lesz szó? Ebben a fejezetben

- a *Quarto* változatos tartalmi és kimeneti formátumait ismerjük meg,
- a PDF és MS Word formátumú dokumentumokat és prezentációkat, valamint
- a könyvek létrehozásának lehetőségeit ismertetjük.

A *Quarto* dokumentumok többfajta tartalmi és kimeneti formátumot támogatnak. Eddig az egy dokumentumra épülő, kimenetként a HTML formátumot használó lehetőségre szűkítettük a *Quarto* bemutatását. Egyszerűbb esetekben ez tökéletes választás lehet, de a *Quarto* ennél sokkal többet tud. Ha a *Quarto* segítségével készíthető számtalan állományt csoportosítani szeretnénk, akkor két megközelítést alkalmazhatunk:

Figyelhetünk a létrehozandó dokumentum *tartalmára*, strukturális felépítésére. A felépítésre vonatkozó szempontok alapján minimálisan a következő tartalmi típusokat különböztethetjük meg:

- **Dokumentum**
  Folyamatos szöveges tartalom, világos tagolással. Ez az egydokumentumos megközelítés jól használható cikkek, riportok, jegyzetek, rövidebb dokumentációk készítésére.
- **Prezentáció**
  Diákra osztott, tömör, vizuális hangsúlyú tartalom. Kiváló eszköz konferenciákra, oktatásra, szakmai bemutatókra készítendő tartalom esetén.
- **Könyv**
  Több fejezetre és alfejezetre tagolt, összetett, hosszú tartalom. Segítségével tankönyvek, kézikönyvek, részletes dokumentációk hozhatók létre.
- **Weboldal**
  Több egymással összekapcsolt dokumentumból (oldalakból) álló, navigációval ellátott tartalom. Online tudásbázisok, dokumentációs portálok, blogok és komplex oktatási oldalak készítésére használjuk.

A másik megközelítés szerint kiemelhetjük az elkészítendő állomány végleges *fájlformátumát*. A kimeneti formátumokat a következő csoportokba sorolhatjuk:

- **Webes formátumok** (HTML)
  Könnyen hordozható, webböngészőkben megjeleníthető, akár interaktív, dinamikus elemekkel gazdagítható fájlformátum.
- **Nyomtatható formátumok** (PDF)
  Rögzített oldaltördelésű, nyomtatásra, offline olvasásra optimalizált fájlformátum, amelynek formázása stabil és kötött.
- **Irodai szerkeszthető formátumok** (MS Word, ODT, PPTX)
  További szerkesztésre alkalmas, széles körben használt, népszerű fájlformátumok.
- **E-könyv formátumok** (ePub)
  Digitális könyvolvasókra optimalizált, reszponzív, újratördelhető, kényelmes olvasást lehetővé tevő fájlformátum.

A 11.4 táblázat a tartalmi típusok és támogatott technikai formátumok tipikus kapcsolatait mutatja be.

11.4. táblázat: A *Quarto* dokumentumok tartalmi típusai és támogatott technikai formátumai *(saját szerkesztés)*

Quarto dokumentumtípus	HTML	PDF	MS Word	ODT	ePub	PPTX
Dokumentum	✅	✅	✅	✅	✅	
Prezentáció	✅	✅				✅
Könyv	✅	✅	✅		✅	
Weboldal	✅					

11.3.1. Dokumentum és prezentáció

A **Dokumentum** és a **Prezentáció** tartalmi típusok esetén a *Quarto* dokumentumokat többféle formátumban is elmenthetjük. A **Dokumentum** esetén a HTML, PDF, MS Word és ODT formátumok is támogatottak, a **Prezentáció** esetén pedig a HTML, PDF és a PPTX formátumok is elérhetőek. Ez a két tartalmi típus egymáshoz nagyon hasonló munkamenetet támogat, amely lényegében megegyezik a korábban látott egydokumentumos HTML dokumentumok készítésének folyamatával:

1. Létrehozzuk a QMD állományt a projektkönyvtárunkban

- dokumentum esetén a `File / New File / Quarto Document...` menüpontot választjuk és el is mentjük `dok_pelda.qmd` néven
- prezentáció esetén a `File / New File / Quarto Presentation...` menüpontot választjuk és `prez_pelda.qmd` néven mentjük el.

2. A megjelenő dialógus dobozban beállítjuk a dokumentum vagy prezentáció címét, szerzőjét, továbbá

    - dokumentum esetén kiválasztjuk a kimeneti fájlfomátumot: HTML, PDF vagy MS Word (telepített MS Word esetén)
    - prezentáció esetén kiválasztjuk a kimeneti fájlfomátumot: HTML (Reveal JS alapon), PDF (Beamer alapon, telepített LaTex esetén) vagy a PPTX (telepített PowerPoint esetén)

3. Tovább specifikálhatjuk a dokumentum vagy prezentáció beállításait a YAMl fejlécben, majd elkészítjük a dokumentum vagy prezentáció tartalmát.

4. A dokumentum vagy prezentáció fordításához a `Render` gombot használjuk, vagy a `Ctrl + Shift + K` billentyűkombinációt.

A QMD állomány létrehozása után érdemes áttekinteni a fejléc tartalmát. A fejléc tartalmát HTML dokumentum esetén már megismertük (lásd a 11.1.1 fejezetet), azonban a PDF és MS Word dokumentumok esetén speciális beállításokat is megadhatunk.

A PDF dokumentum fejléce tipikusan következőképpen néz ki:

```
---
title: "PDF dokumentum"
format:
  pdf:
    pdf-engine: lualatex
    documentclass: scrreport
    toc: true
    number-sections: true
    fontfamily: libertinus
    keep-tex: true
    cap-location: top
lang: hu
language: hungarian.yml
bibliography: mestint.bib
csl: https://raw.githubusercontent.com/citation-style-language/styles/master/apa.csl
---
```

A YAML fejrészben a `format:pdf` alatt állíthatjuk be a PDF dokumentumra specifikus formázását. A PDF dokumentumok esetén a következő beállításokat használhatjuk:

- `pdf-engine`: a PDF dokumentum fordításához használt motor, amely lehet `pdflatex`, `lualatex` vagy `xelatex`,
- `documentclass`: a LaTeX dokumentum osztálya, amely lehet `scrartcl`, `scrreprt`, `scrbook`, `article`, `report`, `book`,
- `toc`: a tartalomjegyzék megjelenítését írjuk elő,
- `number-sections`: a fejezetek sorszámozását állítjuk be,
- `fontfamily`: a dokumentum alap betűtípusát állítjuk be (vonatkozni fog a szövegre és a fejlécekre is),
- `keep-tex`: a fordítás során a LaTeX állományok megtartása,
- `cap-location`: a kép és táblázat feliratok elhelyezése, amely lehet `top` vagy `bottom`.

Számos további beállítás is elérhető a PDF dokumentumok esetén, amelyekről a Quarto dokumentáció ad részletes tájékoztatást. Vegyük figyelembe, hogy a PDF dokumentumok esetén a LaTeX környezet telepítése szükséges, amit könnyen elvégezhetünk a MiKTex vagy a TinyTex oldalon található útmutatások segítségével.

Az MS Word állományok esetén a YAML fejléc tipikusan a következőképpen néz ki:

```yaml
---
title: "Dokumentum 1.0"
format:
  word:
    toc: true
    number-sections: true
    number-offset: 1
    reference-doc: template.docx
---
```

A fenti fejléc legnagyobb újdonsága, hogy sablondokumentumot használunk a `reference-doc:` beállítással, amely a dokumentum formázását határozza meg. A sablon dokumentumot a Microsoft Word programban készíthetjük elő, amelyet `template.docx` néven projektkönyvárunkban mentünk el. A sablon dokumentumban a végleges Word állomány megjelenését előkészítjük, gondoskodunk a címsorok, felsorolások, számozások, képek és táblázatok formázásáról. A fordítás során a QMD állomány tartalma ebbe a sablonba kerül beillesztésre. Ez nagyszerű lehetőség a dokumentumok formázásának testreszabására, hiszen a sablon dokumentumban a Word programban elérhető összes formázási lehetőséget használhatjuk.

Amennyiben PowerPoint prezentációt szeretnénk készíteni, akkor a YAML fejléc a következőképpen nézhet ki:

```
---
title: "Kiselőadás 1.0"
author: "Abari Kálmán"
format:
  pptx:
    incremental: true
    reference-doc: template.pptx
---
```

Ahogyan a fenti fejléc is mutatja, prezentációk készítéséhez is érdemes sablont használni, amely formailag meghatározza a prezentáció megjelenését. A sablon dokumentumot a Microsoft PowerPoint programban készíthetjük elő, amelyet `template.pptx` néven projektkönyvárunkban mentünk el. Meghatározzuk a diák háttérszínét, a betűtípusokat, a címsorok és szövegek formázását, a táblázatok és ábrák megjelenését. A fordítás során a QMD állomány tartalma ebbe a prezentációs sablonba kerül beillesztésre.

A *Quarto* prezentációk esetében, a fejlécet követő tartalmat a diasor felépítésének megfelelően kell megadnunk. A diasor felépítése a következőképpen nézhet ki egy, a mindennapos testmozgás fontosságáról szóló diasor esetén. A fenti fejléc utáni tartalom a következő lehetne:

```
# A mindennapos testmozgás fontossága

## Miért fontos a mindennapos testmozgás? {.incremental}

- A mindennapos testmozgás segít megőrizni a testi és lelki egészséget.
- A rendszeres testmozgás csökkenti a stresszt és javítja a hangulatot.

## Hogyan kezdjünk neki? {.nonincremental}

- Kezdjük kis lépésekkel, például napi 10-15 perces sétával.
- Fokozatosan növeljük a mozgás időtartamát és intenzitását.

## Milyen sportágakat válasszunk?

:::: {.columns}

::: {.column width="60%"}

| Sportág | Előnyök |
```

```
|---------|---------|
| Futás | Jó állóképesség, zsírégetés |
| Úszás | Kíméli az ízületeket, teljes testet átmozgat |
| Kerékpározás | Kíméli az ízületeket, szép lábakat formál |
| Jóga | Stresszoldás, hajlékonyság növelése |

:::

::: {.column width="40%"}

[Quarto](images/quarto.png){#fig-quarto width=50%}

:::

::::
```

Prezentációk összeállításánál figyelembe kell venni, hogy a tartalmat a diákra kell osztani:

- a címdiákat a `#` karakterrel kezdődő sorral jelöljük,
- a tartalommal rendelkező diákat a `##` karakterrel kezdődő sorral jelöljük, és többnyire felsorolásokat használunk,
- a felsorolt elemek megjelenését kattintásra is előírhatjuk, ezt a fejlécben lévő `incremental: true` beállítással minden diára előírhatjuk, de diánkénti megadásra is van lehetőség a `{.incremental}` és `{.nonincremental}` beállításokkal,
- a dia tartalmát megoszthatjuk a `{.column}` beállítással, amely a dián belüli oszlopok létrehozására szolgál.

Összefoglalóan azt mondhatjuk, hogy a **Dokumentum** és a **Prezentáció** tartalmi típusok esetén a *Quarto* dokumentumokat többféle formátumban is elmenthetjük, és ezt a fejlécben található `format:` beállítással irányíthatjuk. Lehetséges értékei a `html`, `pdf`, `docx`, `odt` és `pptx`. A kiválasztott fájlformátum a fejléc további beállításaira is hatással van, más opciók közül választunk ha a dokumentumot HTML, Word vagy PDF formátumban szeretnénk elkészíteni, és perezentációk esetén is különböznek a beállításai lehetőségek HTML, PDF és PPTX esetén.

11.3.2. Könyv

A könyv tartalmi típus esetén a *Quarto* dokumentumokat többféle formátumban is elmenthetjük. Szóba jöhet a HTML, PDF, MS Word és ODT formátum is. A könyv létrehozásához speciális projektet érdemes létrehozni, ugyanis számos állomány létrejön a folyamat végén,

amelyek nagyszerű kiindulópontok lehetnek a teljes könyv létrehozásához. A lépések a következők:

1. Induljunk el a `File / New Project...` menüponttal, majd válasszuk a `New Directory` lehetőséget és a `Quarto Book` opciót. Határozzuk meg a könyvünk projektkönyvtárának a nevét (legyen `book_pelda`) és helyét, majd kattintsunk a `Create Project` gombra.
2. Számos állomány létrejött, melyek egyik része a könyvhöz kapcsolódó beállításokért, másik része a könyv tartalmáért felelős. Ezeknek az állományoknak a tartalmát érdemes áttekinteni, illetve meghatározni. Végső soron könyvünk tartalmáért mi leszünk a felelősek. Később részletesen bemutatjuk a lehetőségeket, de most megtehetjük, hogy nem változtatunk semmin.
3. A könyv fordítását a `Ctrl-Shift-B` billentyűkombinációval kezdeményezhetjük, vagy a jobb felső panel `Build` fülén a `Render Book` gombját is használhatjuk, ahol egyenként is eldönthetjük, hogy melyik formátumban szeretnénk elkészíteni a könyvet (például PDF vagy HTML formátumban).

Ha követtük a fenti lépéseket, akkor a könyvünk máris elkészült, az alapértelmezett tartalommal. A `book_pelda` nevű projektkönyvtárunk `_book` alkönyvtárában megtaláljuk az `index.html` állományt, amely a HTML alapú könyvünk kezdőlapja, illetve a `book_pelda.pdf` állományt, amely PDF formátumban tartalmazza a teljes könyvünket (telepített LaTeX esetén).

A könyvünk testreszabásához tekintsük át projektkönyvtárunk tartalmát:

- `_quarto.yml` - a könyv beállításait tartalmazó fájl, amelyben a könyv címét, szerzőjét, nyelvét és egyéb beállításait adhatjuk meg,
- `index.qmd` - a könyv kezdőlapját/kezdőoldalát tartalmazó fájl, ez lesz az első fájl, amelyet a fordításkor az R feldolgoz; lényegében itt kezdődik a könyvünk, könnyen lehet, hogy az előszóval.
- `intro.qmd`, `summary.qmd`, `references.qmd` - a könyv további fejezeteit tartalmazó fájlok, amelyek a könyv tartalmát alkotják
- `cover.jpg` - a könyv borítóképét tartalmazó fájl, amelyet a `cover-image:` beállítással hivatkozunk meg a YAML fejlécben,
- `references.bib` - a könyvben használt irodalomjegyzék fájlja, amelyet a `bibliography:` beállítással hivatkozunk meg a YAML fejlécben.

A fenti alapértelmezetten létrehozott állományokon túl érdemes kiegészíteni a projektkönyvtárunkat a következő könyvtárakkal és fájlokkal:

- `docs/` - alkönyvtár a projektkönyvtárban a könyv tartalmának tárolására, a `_book` helyett,
- `images/` - alkönyvtár a projektkönyvtárban, amelyben a könyvbe beszúrandó képeket tárolhatjuk,

- `hungarian.yml` - a magyar nyelvű fordítást tartalmazó fájl, amelyet a `language:` beállítással hivatkozható meg a YAML fejlécben; a kiinduló példányát, amelyik még nem tartalmaz magyar nyelvű fordítást, a Document Language oldalról tölthetjük le, és a szükséges fordítást magunk végezzük el,
- érdemes átnevezni és esetlegesen új fájlokkal kiegészíteni az alapértelmezett `intro.qmd`, `summary.qmd`, `references.qmd` fájlokat, hogy a könyvünk tartalmát jobban tükröző állománynéveket kapjunk.

A könyvünk beállításait a `_quarto.yml` fájlban végezhetjük el. A fájl tartalma hasonló az eddig egydokumentumos QMD állományok fejlécéhez (azaz a YAML fejléchez), de a könyv esetén további beállítások is elérhetőek. A könyv YAML fejlécének tipikus tartalma a következőképpen nézhet ki:

```yaml
project:
  type: book
  output-dir: docs

lang: hu
language: hungarian.yml

bibliography: references.bib
biblio-style: apalike
csl: https://raw.githubusercontent.com/citation-style-language/styles/master/apa.csl

book:
  title: "Miért fontos a mozgás?"
  author: "Abari Kálmán"
  subtitle: "A mozgásformák összahasonlítása"
  description: "A mozgásformák és a napi rutin szerepe az egészségmegőrzésben."
  date: today
  cover-image: cover.jpg
  search: true

  chapters:
    - index.qmd
    - part: Alapozás
      chapters:
        - 01-Bevezetes.qmd
        - 02-Mozgas.qmd
        - 03-Mozgas-osszetevok.qmd
    - part: Mozgásformák
```

```
      chapters:
        - 04-Mozgasformak.qmd
        - 05-Osszehasonlitas.qmd
    - part: Mi kell tennem?
      chapters:
        - 06-Napi-1-2-orat.qmd
        - 07-Napi-rutin.qmd
    - 08-Irodalomjegyzek.qmd
  appendices:
    - app-01-Tablazatok.qmd
    - app-02-Kerdesek.qmd
format:
  html:
    theme: cosmo
  pdf:
    documentclass: scrreprt
```

A fenti YAML fejlécben a `project:` rész újdonság, innen derül ki, hogy valójában *Quarto* könyvet készítünk (`type: book`). Az `output-dir:` beállítás határozza meg, hogy a fordítás során létrejövő fájlok hova kerüljenek. A `lang:` beállítás a nyelvet határozza meg, míg a `language:` beállítás a nyelvi fájlra mutat, amely a címkék fordítását tartalmazza (például `Author` helyett a `Szerző` címke fog megjelenni). A `bibliography:` beállítás az irodalomjegyzékre mutat, míg a `biblio-style:` beállítás az irodalomjegyzék stílusát határozza meg. A `csl:` beállítás a hivatkozási stílus fájlra mutat.

A `book:` beállítás a könyv címét, szerzőjét, alcímét, leírását és a borítókép fájlját határozza meg. A `search: true` beállítás a keresőmező megjelenítését írja elő. A `chapters:` beállítás a könyv fejezeteit és alfejezeteit határozza meg, ahol a `part:` beállítás a fejezetek csoportosítására szolgál. Az `appendices:` beállítás az esetleges függelékek megadására szolgál.

A fenti beállítások hatására a fordítás után a 11.7. ábrán látható HTML könyvet kapjuk eredményül.

Saját könyvünk sikeres felépítéséhez számos példát találunk az interneten. Az egyik ilyen forrás az R for Data Science (2e), amelyet a könyv tartalmának megismerése mellett, annak szerkezetének tanulmányozására is ajánlok. A könyv teljes tartalma, annak szerkezete a `_quarto.yml` állománnyal együtt a könyv GitHub oldalán érhető el: https://github.com/hadley/r4ds/.

11.7. ábra: Egy lehetséges *Quarto* könyv első oldala, HTML formátumban [Allaire és Dervieux (2024); *az illusztráció egy MI által (ChatGPT-4o) generált kép*]

Összefoglalás

A *Quarto* az egydokumentumos HTML állományoknál jóval több kimeneti formátumot és tartalmi típust is támogat. Létrehozhatunk dokumentumot HTML, PDF, MS Word és ODT formátumban, prezentációt pedig HTML, PDF és PPTX formátumban. Az adott fájlformátum beállításait a YAMl fejlécben kell elvégeznünk. Prezentáció készítésénél még arra kell figyelnünk, hogy a tartalmat diákra kell osztani. A többdokumentumos formátumok közül a könyv lehetőséget mutattuk be, amelyre egy külön *Quarto Book* opciót biztosít az *RStudio* projektlétrehozó mechanizmusa. Egy *Quarto* könyv QMD fájlok sorozata, amely a könyv tartalmát jelenti. A könyv az `index.qmd` fájltól indul, majd tetszőleges számú QMD állomány következik. A sorrendet és az egyéb beállítási lehetőségeket a `_quarto.yml` fájl tartalmazza.

Feladatok

1. Az előző rész egyik feladata HTML dokumentum elkészítését írta elő, amely ábrával és képpel is bemutatja a top 10 legtöbb bevételt hozó Pixar filmet (forrás: Pixar Box Office Varázslat). Alakítsuk át ezt a HTML állományt:

- PDF dokumentummá,
- MS Word dokumentummá,
- HTML prezentációvá,
- PDF prezentációvá,
- PPTX prezentációvá,
- *Ouarto* könyv formátummá.

2. Készítsünk egy *Ouarto* könyvet a 10. fejezetben bemutatott hipotézisvizsgálatokból!

Utószó

Az R elsajátítása nem egy hétvégi kaland, hanem egy fokozatosan építkező, hosszú távú tanulási folyamat. Ezalatt az ember nem csupán egy programnyelv parancsait ismeri meg, hanem egy új szemléletmódot is kialakít: azt, hogyan gondolkodjunk adatokról, problémákról és megoldásokról. Aki eljutott a könyv ezen pontjáig, az már kétségtelenül túl van az első hullámvölgyeken, a hibaüzenetek okozta tanácstalanságon, az első sikerélményen egy hibás kódsor kijavítása után, a csomagtelepítés buktatóin, és azon a különleges elégedettségen, amit egy szép, elegáns megoldás kidolgozása jelentett.

Ha az Olvasó most úgy érzi, hogy az R-rel való kapcsolata „csak most kezd igazán izgalmassá válni", akkor biztos lehet benne, hogy jó úton halad. A könyv itt véget ér, de az R nyelv valódi ereje nem csupán azokban az eszközökben rejlik, amelyeket e lapokon keresztül megismert, hanem azokban a felfedezésekben, amelyeket ezután, önállóan fog megtenni. Ez a könyv arra törekedett, hogy ne csupán parancsokat ismertessen, hanem gondolkodási mintákat is közvetítsen: hogyan érdemes megközelíteni egy kérdést, miként lehet az adatokat értelmezhető formába alakítani, és hogyan lehet mindezt úgy megvalósítani, hogy a kód átlátható, újrafelhasználható és örömmel írható legyen. Ez a tanulási folyamat időt és türelmet igényel — de ha eddig eljutott, akkor nagy valószínűséggel már érzi is, hogy az R nem csupán egy eszköz a munkához, hanem egyre inkább a saját gondolkodási eszköztárának része.

Biztos lehet benne az Olvasó, hogy az R elsajátításába fektetett erőfeszítések sokszorosan megtérülnek. A nyelv mára nem csupán a tudományos kutatás és az egyetemi oktatás meghatározó eszköze, hanem fontos helyet szerzett magának az ipari adatelemzésben, az üzleti döntéstámogatásban, a statisztikai modellezésben és a vizualizációs megoldásokban is. Emellett egyre jelentősebb szerepet tölt be a mesterséges intelligencia, a gépi tanulás gyakorlati alkalmazásaiban: a prediktív modellezéstől kezdve a természetes nyelvfeldolgozáson át a big data elemzésekig számos területen használják. Az R közössége folyamatosan bővül, és a nyelv körüli ökoszisztéma egyre gazdagabbá válik. A több mint 22 ezer csomag számos területen segíti az adatkezelést, a statisztikai elemzést és a szép ábrák készítését, szinte biztos lehet benne, hogy saját érdeklődési területén is talál olyan csomagot, amely megkönnyíti a munkáját.

Hogyan tovább? A könyvben megismert receptet érdemes követni: minden adatfeldolgozási feladat – legyen az akármennyire apró vagy hatalmas – egy új kihívás, amelyhez rendelünk egy *RStudio* projektet, és a megoldást jelentő R sorokat egy *Quarto* dokumentumban rögzítjük. Addig ügyeskedünk, míg a megoldást meg nem találjuk, és a kívánt HTML (PDF, Word) dokumentumot meg nem kapjuk. Ha ezt a tevékenységet elég sokáig gyakoroljuk, és a könyvben bemutatott példák és megoldandó feladatok mellett saját projekteken is dolgozunk, akkor tudásunk egyre mélyebb és szélesebb lesz, és elérjük azt a szintet, amikor már nem csupán a könyvben bemutatott példák megoldása megy játszi könnyedséggel, hanem saját ötleteinket is megvalósíthatjuk. Higgyen az R-ben és első sorban saját magában!

Sok sikert és örömet kívánok az R-rel való további utazáshoz!

Irodalom

Allaire, J. és Dervieux, C. (2024). *quarto: R Interface to 'Quarto' Markdown Publishing System*. https://CRAN.R-project.org/package=quarto

Beckerman, A., Childs, D. és Petchey, O. (2017). *Getting Started with R*. Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198787839.001.0001

Brace, N., Snelgar, R. és Kemp, R. (2016). *SPSS for Psychologists: And Everybody Else*. Red Globe Press.

Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum.

Comtois, D. (2025). *summarytools: Tools to Quickly and Neatly Summarize Data*. https://CRAN.R-project.org/package=summarytools

Csapó, G., Csernoch, M. és Abari, K. (2020). Sprego: case study on the effectiveness of teaching spreadsheet management with schema construction. *Education and Information Technologies*, *25*(3), 1585–1605. https://doi.org/10.1007/s10639-019-10024-2

Dag, O., Dolgun, A. és Konar, N. M. (2018). onewaytests: An R Package for One-Way Tests in Independent Groups Designs. *The R Journal*, *10*(1), 175–199. https://doi.org/10.32614/rj-2018-022

Dalgaard, P. (2008). *Introductory Statistics with R*. Springer. http://books.google.co.uk/books?id=YI0kT8cuiVUC

Dancey, C. P. és Reidy, J. (2011). *Statistics Without Maths for Psychology*. Prentice Hall. https://books.google.hu/books?id=dTKdcQAACAAJ

Dienes, Z. (2016). *Mitől tudomány a pszichológia?* Akadémiai Kiadó.

Fox, J., Marquez, M. M. és Bouchet-Valat, M. (2024). *Rcmdr: R Commander*. https://github.com/RCmdr-Project/rcmdr

Iannone, R., Cheng, J., Schloerke, B., Hughes, E., Lauer, A., Seo, J., Brevoort, K. és Roy, O. (2024). *gt: Easily Create Presentation-Ready Display Tables*. https://CRAN.R-project.org/package=gt

Mangiafico, S. S. (2016). *Summary and Analysis of Extension Program Evaluation in R*. https://rcompanion.org/handbook/

Neuwirth, E. (2022). *RColorBrewer: ColorBrewer Palettes*. https://CRAN.R-project.org/package=RColorBrewer

Posit team. (2025). *RStudio: Integrated Development Environment for R*. Posit Software, PBC. http://www.posit.co/

R Core Team. (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rasch, D., Kubinger, K. D. és Yanagida, T. (2011). *Statistics in Psychology Using R and SPSS*. Wiley.

Reiczigel, J., Harnos, A. és Solymosi, N. (2007). *Biostatisztika nem statisztikusoknak*. Pars Kft. http://biostatkonyv.hu/

Vargha, A. (2000). *Matematikai statisztika pszichológiai, nyelvészeti és biológiai alkalmazásokkal*. Pólya Kiadó.

Venables, W. N. és Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer New York. https://doi.org/10.1007/978-0-387-21706-2

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wilkinson, L. (2005). *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag.

Xie, Y. (2024). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. https://yihui.org/knitr/

Zhu, H. (2024). *kableExtra: Construct Complex Table with 'kable' and Pipe Syntax*. https://CRAN.R-project.org/package=kableExtra