

# Adatkezelés és egyváltozós elemzések R-ben

Abari Kálmán

2022-07-22



# **Tartalomjegyzék**



# Üdvözöljük

Ez a honlap az *Adatkezelés és egy változós elemzések* c. könyv elektronikus változatát mutatja be. A nyomtatásban megjelent könyvhöz képest számos bővítést tartalmaz:

- a függelék fejezetei, például a kitűzött feladatok megoldásai is itt jelennek meg,
- az R grafikus lehetőségeit tartalmazó fejezet bővebb a *hagyományos grafika* résszel.

A teljes könyv, az adatbázisok és az R kódok megtalálhatók a következő címen<sup>1</sup>.

A könyvet Máth János lektorálta, és Friss Kinga illusztrálta.

<sup>1</sup> <https://abarik.github.io/aeer>



# Előszó

Kedves Olvasó!

Köszönjük, hogy bizalmat szavaz könyvünknek, és az R megismeréséhez ezt az utat választja. Az első lépésektről a komplett adatalemzési feladatok megoldásáig vezetjük az Olvasót, és főként kezdő vagy újrakezdő felhasználókhöz szólunk. Utunk során áttekinthjük 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:



**Egy hegység.** Az így jelölt fejezet az R alaptudás része, megismerése feltétlenül járult hozzá. A könyvben megfogalmazott célok ezen fejezetek megismerésével is elérhetők, azaz komplett adatalemzéseket hajthatunk végre csupán ezek végig olvasásával is.



**Két hegység.** Kiegészítő tudást tartalmazó fejezetek. Újabb eszközök megismerését teszik lehetővé, és/vagy hozzájárulnak az *egy hegység* fejezetek mélyebb megértéséhez.



**Három hegység.** 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 vágunkat is kielégíthessük. **Könyvünk olvasását tehát az 1. fejezet *egy hegység* alfejezetével (?? Elindulás) 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. Megismétljük a legfontosabb fogalmakat és felsoroljuk a megismert függvényeket.



**Összefoglalás.** Nem csak a fejezet áttanulmányozása után, hanem időről-időre javasoljuk a fejezet végi összefoglalások áttekintését. Ezzel leellenőrizhetjük R tudásunk frissességét, eldönthetjük, hogy érdemes-e újra átolvasni a fejezetben leírtakat. És mit jegyezzünk meg az előszóból: a könyv 11 fejezetet tartalmaz, az alfejezeteket nehézségek alapján három különböző ikon egyikével jelöltük: *egy hegység, két hegység vagy három hegység*.

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.

Örömmel fogadjuk Olvasóink észrevételeit az [abari.kalman@gmail.com](mailto:abari.kalman@gmail.com) címen.

# 1. fejezet

## Itt kezdődik



Szóval azt mondod, hogy publikációs elemzést tudok készíteni, ha előírrom ezt a könyvet?

### 1.1. Elindulás



Ebben a fejezetben:

- bemutatunk egy konkrét adatelemzési példát,
- áttekintjük a könyv tartalmát,
- lehetőséget adunk az előzetes R ismeretek felmérésére,
- és segítünk a megfelelő fejezet kiválasztására 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 lefedor három nagy terület mindegyikébe: az adatkezelésbe, a grafikus megjelenítésbe és az adatelemzésbe is. A leírtak megértéséhez a statisztikai alapismereteken 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 előismertekkel 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.

### **Bevezető példa: 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 felméréssel zárult. Az összegyűjtött adatok az `excel_2020.xlsx` állományban állnak rendelkezésre.

Nézzük az adatalemzés lépéseit és egyben könyünk felépítését!

#### **2. fejezet: Mi az R?**

A bevezető 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 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 adatalemzés végrehajtásához az *RStudio*-t ajánljuk, és azon belül pedig a projektek használatát szorgalmazzuk. A 4. fejezetben megismérjük 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 RMarkdown állományt,
4. helyezzük el a lentebb szereplő R parancsokat az RMarkdown állomány egyes csonkjában.

#### **5. fejezet: Az R nyelv.**

Az R parancsok létrehozásának vannak szabályai, amelyeket a munka során be kell tanunk. 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 adatalemzés első lépése az adatállomány beolvasása. Adataink változatos formában állhatnak rendelkezésre, a 6. fejezetben ezek beolvasására kapunk receptet.

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

```
str(felmeres)           # a dataframe szerkezete
names(felmeres)         # változónevek
unique(felmeres$modszer) # különböző értékek
```

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 elemzésekben 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   min   max range   se
#> X11     1       modern    1 13  0.65  0.20  0.33  0.95  0.63  0.05
#> X12     2 hagyományos  1 13  0.38  0.13  0.08  0.55  0.47  0.04
```

### 9. fejezet: Grafika.

A grafikus megjelenítés is a leíró statisztikai elemzés része. A 9. fejezetben részletesen 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 egy változós elemzéseket mutatjuk be.

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

```
wilcox.test(eredmeny~modszer, data=felmeres)
#>
#> Wilcoxon rank sum exact test
#>
#> data: eredmeny by modszer
#> W = 145, p-value = 0.001
#> 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 minden összes egészítő ki a következő sorokkal a leíró statisztikai elemzést, majd a *Knit* nyomógomb segítségével fordítsuk le az RMarkdown állományt. A leíró statisztikai mutatók már is

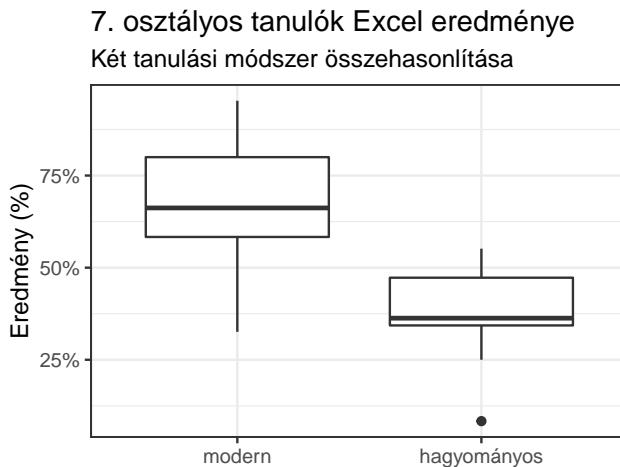
táblázatos, könnyen áttekinthető formában jelennek meg.

```
options(OutDec = ",") # a tizedesjel beállítása
st <- psych::describeBy(x = felmeres$eredmeny, group = felmeres$modszer,
                        mat=T, fast=T, digits = 2)
knitr::kable(st[-1], align = c("c", "c"), row.names = F)
```

group1	vars	n	mean	sd	min	max	range	se
modern	1	13	0,65	0,20	0,33	0,95	0,63	0,05
hagyományos	1	13	0,38	0,13	0,08	0,55	0,47	0,04

Publikációnk szerves része a magyarázó ábra. A korábban rajzolt dobozdiagramunkat csinosítuk ki a következő sorok R csonkba helyezésével. A `ggsave()` függvény a háttér-tárra 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" +
  scale_y_continuous(labels = scales::percent) + theme_bw()
ggsave(filename = "output/kep/dd.png", plot = p1)
p1
```



A bevezető példa megoldásához természetesen a hipotézisvizsgálat szöveges értékelés is hozzátarozik, 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.

### 1.1.1. Összefoglalás



Ebben az alfejezetben egy adatalemzési példát oldottunk meg, melynek segítségével illusztrálni tudtuk a további fejezetek tartalmát. A 2. fejezetben áttekinthető 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 adatalemzés szokásos lépései 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 egy változós hipotézisvizsgálatoké, az utolsó, 11. fejezet az eredmények publikálását foglalja össze.

### 1.1.2. 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. A bevezető példa (Két tanítási módszer összehasonlítása) megoldásában a hipotézisvizsgálat alapján adjunk szöveges értékelést!

## 1.2. A könyv felépítése



Ebben a fejezetben:

- bemutatjuk a könyv részletes felépítését,
- 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.

Fejezet/alfejezet	Leírás
<b>1. Itt kezdődik</b>	
1.1. Elindulás	A könyv fejezeteinek bemutatása egy konkrét adatalemzésen keresztül
1.2. A könyv felépítése	Jelen alfejezet, amelyben a könyv egyes alfejezeteit mutatjuk be röviden
1.3. Próbák lista	A könyvben szereplő egy változós statisztikai eljárások lista
<b>2. Mi az R?</b>	
2.1. Az R bemutatása	A parancssoros R jellemzői, az R nyelv, az <i>Alap R</i> és a csomag fogalma

Fejezet/alfejezet	Leírás
2.2. A modern R	Megtanuljuk a <i>Tidyverse R</i> fogalmát, megtudjuk mi a modern R
2.3. Múlt és jelen	Az R rövid története, alapelvek az R tanulásához, és az R alaptudás elemei
<b>3. Az R telepítése</b>	
3.1. Az Alap R és az RStudio telepítése	Megismerjük az <i>Alap R</i> és az <i>RStudio</i> telepítését
3.2. A Tidyverse R telepítése	A <b>tidyverse</b> csomag(gyűjtemény) telepítése
3.3. Az R frissítése	Az <i>Alap R</i> , az <i>RStudio</i> és a csomagok frissítésének módszerei
<b>4. Munka az R-ben</b>	
4.1. Az RStudio használata	Az <i>RStudio</i> jellemzői és felépítése, a projektek használata
4.2. Segítség az R használatához	Segítségkérési lehetőségek az R-ben, a beépített súgó használata
4.3. Az Alap R használata	Az <i>Alap R</i> konzolja, az <i>RGui</i> , az <i>R Commander</i> és a kötegelt üzemmód
<b>5. Az R nyelv</b>	
5.1 Adatobjektumok	Az objektumok fogalma és létrehozásuk, egyszerű kifejezések
5.2 Függvények	A függvény fogalma, a függvényhívás módja, a kifejezés teljes fogalma
5.3 Adatszerkezetek	Az R egyszerű és összetett adatszerkezetei, létrehozásuk és indexelésük
5.4 További adatszerkezetek és függvények	A dátum, tömb, táblázat és tibble adatszerkezetek kezelése
5.5 Objektumok és típusok	Az R objektum-orientált lehetőségei
<b>6. Beolvasás</b>	
6.1 Excel, SPSS és RDS állományok	A <i>rio</i> csomag lehetőségei
6.2 Tidyverse beolvasás	A <i>tidyverse</i> csomag lehetőségei
6.3 Tagolt szöveges állományok	A hagyományos R lehetőségei
<b>7. Adatmanipuláció</b>	
<b>8. Mutatók és táblázatok</b>	
<b>9. Grafika</b>	
<b>10. Hipotézisvizsgálatok</b>	
<b>11. Publikáció</b>	

### 1.2.1. Összefoglalás



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

### 1.2.2. 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 RMarkdown állományokat szerkesztünk és hajtunk végre. Mely fejezetekben találunk hasznos információkat az R ezen használatával kapcsolatban?

## 1.3. Próbák lista



Ebben a fejezetben:

- áttekintést adunk az egy- és kétváltozós hipotézisvizsgálatokról.

A 10. fejezetben bemutatjuk az egy- és kétváltozós hipotézisvizsgálatok végrehajtását. Ebben a fejezetben felsoroljuk a legfontosabb próbákat, összesen öt táblázatban soroljuk fel őket:

- egy mintát vizsgáló próbák (??, táblázat),
- páros mintát vizsgáló próbák (??, táblázat),
- két független mintát vizsgáló próbák (??, táblázat),
- több összetartozó mintát vizsgáló próbák (??, táblázat),
- több független mintát vizsgáló próbák (??, 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 variánciára vagy a valószínűségre. A 10. fejezetben foglalkozunk az eloszlásvizsgálatok közül a normalitást ellenőrző próbákkal is, így a ??, táblázat ezeket is számba veszi.

### 1.2. táblázat. Egy minta vizsgálata

Cél	Próba neve	R függvény
várható érték	egymintás u-próba egymintás t-próba	<code>BSDA::z.test()</code> <code>t.test()</code>
medián	előjel-próba	<code>BSDA::SIGN.test()</code>

Cél	Próba neve	R függvény
variancia valószínűség normalitás	Mood-féle medián-próba	
	egymintás Wilcoxon-próba	wilcox.test()
	khí-négyzet próba	chisq.test()
	Shapiro-Wilk próba	shapiro.test()
	Kolmogorov-Szmirnov próba	DescTools:LilliefTest()

**1.3. táblázat.** Páros minta vizsgálata

Cél	Próba neve	R függvény
várható érték medián	páros t-próba	t.test(paired=T)
	páros előjel-próba	BSDA::SIGN.test()
	páros Wilcoxon-próba	wilcox.test(paired=T)
variancia valószínűség	var.test()	
	McNemar-próba	mcnemar.test()
	Cohran-Q próba	mcnemar.test()

**1.4. táblázat.** Két független minta vizsgálata

Cél	Próba neve	R függvény
várható érték medián	kétmintás u-próba	BSDA::z.test()
	kétmintás t-próba	t.test()
	Welch-féle d-próba	t.test(var.equal=F)
variancia valószínűség	Mann-Whitney-próba	wilcox.test()
	F-próba	var.test()
	khí-négyzet próba	chisq.test()
	Fisher-féle egzakt próba	fisher.test()

**1.5. táblázat.** Több összetartozó minta vizsgálata

Cél	Próba neve	R függvény
várható érték	egyszempontos összetartozó mintás varianciaelemzés	ez::ezANOVA()
medián	Friedman-próba	friedman.test()

#### 1.6. táblázat.

Több független minta vizsgálata

Cél	Próba neve	R függvény
várható érték	egyszempontos varianciaelemzés Welch-féle egyszempontos varianciaelemzés	aov() oneway.test(var.equal=F)
medián variancia	Kruskal-Wallis-próba Levene-próba Bartlett-próba	kruskal.test() DescTools::LeveneTest() bartlett.test()

#### 1.3.1. 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 későbbiekbén milyen jellegű statisztikai következtetéseket tudunk levonni az R használatával.

#### 1.3.2. 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. fejezet

# Mi az R?



### 2.1. Az R bemutatása



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. Másrészről 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 rendszerünkéről, ha az R-t szeretnénk használni.
- Az R nemcsak egy statisztikai programcsomag önmagában, hanem egy teljes értékű programozási nyelv.
- Az R statisztikai módszerek szinte végletesen 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 előnyvel jár. Egyrészt a szkript állományok létrehozása és végrehajtása a statisztikai elemzések megismeretlhetőségét biztosítja, másrészről ez az oktatók és a hallgatók könnyebb kommunikációját is lehetővé teszi.
- Az R az adataelemzés eredményének sokszínű publikálását is biztosítja. Az R Markdown<sup>1</sup> formanyelv segítségével HTML, PDF és Word dokumentumot, illetve prezentációs diákat vagy akár kész cikkeket hozhatunk létre. A Shiny<sup>2</sup> csomag interaktív Webs alkalmazások építését teszi lehetővé.
- Mára az R használata szinte egyet jelent az ingyenesen elérhető RStudio<sup>3</sup> 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 felhasz-

<sup>1</sup> <https://rmarkdown.rstudio.com/>

<sup>2</sup> <http://shiny.rstudio.com/>

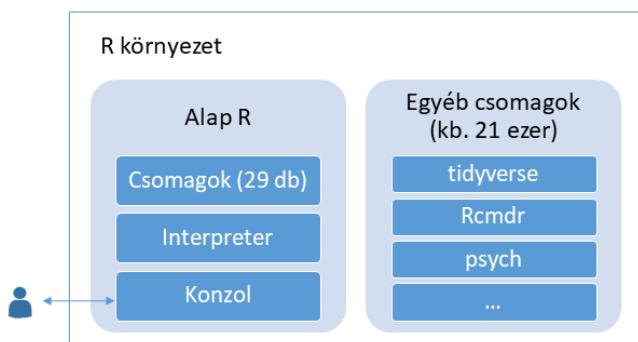
<sup>3</sup> <https://www.rstudio.com/>

nálói felületen egérrel vagy az ujjunkkal mutogatjuk el a kívánt tevékenységet. Az R egészen más megközelítést vall, használata a kezdeti lépések től 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 minden nap 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 mély-símeretére. Az R nyelv elsajátítása során bevezetjük azokat az egyszerű fogalmakat, amelyeket nem nélkülözhetők 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<sup>4</sup>. 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 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.



5

**2.1. ábra.** Az R környezet: Alap R és az egyéb csomagok

<sup>4</sup> <https://www.r-project.org/contributors.html>

Az R környezet három fő összetevőt tartalmaz (??. ábra): (1) egy konzolt, aholá a parancsainkat begépelhetjük; (2) a parancsok végrehajtásáért felelős R interpretert; (3) a csomagokat. 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 valamelyen 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. 21 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. 21 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 többi 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 gyártthatjuk.

A R környezet már igazi szoftver, terjesztésének koordinálását az R Foundation<sup>6</sup> végzi a CRAN<sup>7</sup> infrastruktúráján keresztül. Ez biztosítja, hogy számítógépunkre telepíthesük 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 tettszöleges 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 segítségnként 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 (18407 csomaggal), a Bioconductor (2140 csomaggal) és a GitHub.

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

<sup>6</sup> <https://www.r-project.org/foundation/>

<sup>7</sup> <https://cran.r-project.org/mirrors.html>

### 2.1.4. 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 interpretiert a parancsok tényleges végrehajtására, és jó néhány csomagba szervezett eljárást az adatelemzési feladatok elvégzéséhez. Az *Alap R* minden össze 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*-ból, és az ún. *egyéb csomagokból* áll.

### 2.1.5. 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 programcsomaghöz képest?
5. Milyen ingyenesen elérhető, grafikus felhasználói felülettel rendelkező statisztikai programcsomagok építenek az R-re?
6. Említettük, hogy az adatelmezési munka nem igényli az R programozói fokú ismeretét, de soroljunk fel néhány könyvet, amelyből az R programozása is megtanulható!

## 2.2. A modern R



Ebben a fejezetben:

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

A 2014-es év az R nyelv életében meghatározó változást hozott. Egyszer megjelent a **magrittr** csomagban a pipe operátor (`%>%`), amellyel olvashatóbb kódok írására nyílt lehetőség<sup>8</sup>, másrészt a pipe operátorra alapozva Hadley Wickham bemutatta a **dplyr** és

<sup>8</sup> Más programozási nyelvekben az „objektum” helyett a „változó” elnevezést használják, de a változó

**tidyR** csomagokat. Ezzel az R funkcionális<sup>9</sup> oldalát úgy erősítették meg<sup>10</sup>, 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** és **forcats**. Ezek a csomagok nem egyszerűen új funkciókkal ruhazzák fel az *Alap R* tudását, mint általában az *egyéb csomagok*. A *Tidyverse* csomagai 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 kódal.

Eddig láttuk, hogy az R használatához szükséges az *Alap R* telepítése, majd a speciális problémáknak megfelelően kiegészíthetjük az R tudását úgy, hogy telepítünk egyet vagy többet az *egyéb csomagok* kategoriájából. Választhatjuk akár a *Tidyverse* csomagjait is telepítésre, ugyanis így lehetőségünk nyílik a *Tidyverse R* használatára. Utasításaink megfogalmazásának ma ez a legmodernebb módja.

A modern R alatt lényegében azokat a funkciókat értjük, amelyek a *Tidyverse*<sup>11</sup> 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.

### 2.2.1. 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.

fogalma már foglalt a statisztikában, így szerencsésebb a memóriában tárolt adatokra objektumként hivatkozni.

<sup>9</sup> Az R egy nem túl fiatalkorú, a funkcionális programnyelvekhez hasonlóan építkező programozási nyelv, vagyis egy probléma megoldása 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 kiküszöbölni az R-ben előszeretettel használnak procedurális eszközöket (például `for` ciklusokat), de a kód olvashatoságát és karbantartását igazán ez sem könnyíti meg.

<sup>10</sup> További értékkedás operátorok a `->`, `<->`, `->>` és `=`. Ezeket nem használjuk ebben a könyvben.

<sup>11</sup> <https://www.tidyverse.org/>

### 2.2.2. Feladatok



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

## 2.3. Múlt és jelen



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<sup>12</sup> és Robert Gentleman<sup>13</sup>, 1997-től pedig egy nagyobb csapat, az R Development Core Team<sup>14</sup> 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<sup>15</sup> (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 közössége (*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/><sup>16</sup>. 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 dokumentációkhöz. A CRAN fő számítógépe Ausztriában található <https://CRAN.R-project.org/><sup>17</sup>, azonban nagyon sok naponta frissülő tükörszerver<sup>18</sup> érhető el világszerte.

<sup>12</sup> <https://www.stat.auckland.ac.nz/~ihaka/>

<sup>13</sup> <https://www.linkedin.com/in/robert-gentleman-06845098/>

<sup>14</sup> <https://www.r-project.org/contributors.html>

<sup>15</sup> <https://www.r-project.org/foundation/>

<sup>16</sup> <https://www.r-project.org/>

<sup>17</sup> <https://CRAN.R-project.org/>

<sup>18</sup> <https://cran.r-project.org/mirrors.html>

### 2.3.2. Alapelvek

Az R elsődleges célja, hasonlóan más statisztikai programcsomagokhoz, a statisztikai adatalemzés, amelyet négy lépéssel bonthatunk:

1. adatok beolvasása,
2. adatok előkészítése elemzésre,
3. adatalemzés,
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 szeretnénk megemlíteni, ami segíthet minket az utazásunk során:

- *Magabiztoság* - 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* (???. fejezet) minden R-ben jártas felhasználó számára közös. E könyv célja ennek az alaptudásnak az átadása, melynek birtokában már kellő magabiztosággal 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, minden a kellő magabiztoság 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 szórakoztatják, 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? Meggyőződéünk, ha a lentebb felsorolt témaörökkel tisztában vagyunk, akkor már magabiztos R tudással rendelkezünk, és bármilyen további R témaörök 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 az R Markdown segítségével

#### 2.3.4. 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 dokumentációkhöz.

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.

### 2.3.5. 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. fejezet

## Az R telepítése



"Ha a fejét verzed be, a játéknak vége... felbörök az agyadban, azt hiszel, amit hinni akarsz. De ha a proszt. maradvány csontszövetségekkel szemben állsz, akkor minden mely a nyúl ürge." -

### 3.1. A fő komponensek telepítése



Ebben a fejezetben:

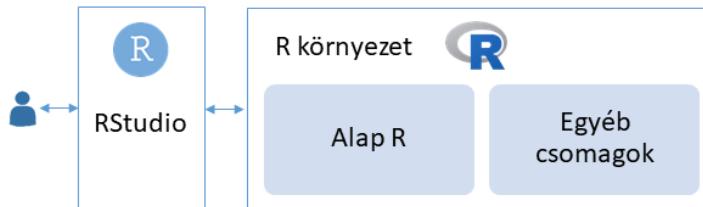
- megismerjük az *Alap R*, az *RStudio* és a csomagok telepítését.

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**

- az *Rstudio-t* használjuk,
- projekt üzemmódban dolgozunk, és
- RMarkdown állományokban rögzítjük az R parancsainkat.

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



1

**3.1. ábra.** Az R kényelmes használata

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 halmozából származnak, és telepítésekkel 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 nyugtázsával végezzük el a telepítést. A telepítést lehetőleg olyan Windows felhasználó alatt végezzük el, amelynek a neve sem ékezes 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://www.rstudio.com/products/rstudio/download/> oldalról. Az RStudio Desktop (Open Source License) változatra lesz szükségünk, töltük le és telepítük ezt a számítógéünkre. 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 (??. ábra).

### 3.1.3. Csomagok telepítése

A csomagok telepítésére az Alap R vagy az RStudio elindítása után van módunk. Érdekes 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 ?? fejezet fellapozással segítséget kaphatunk a lenti parancsok kipróbálásához, de úgy is eljárhatunk, hogy most kihagyjuk ennek a résznek az áttekintését, és később térünk vissza, amikor valóban felmerül az igény csomagok telepítésére.

Az R csomagok hivatalos helye a CRAN (The Comprehensive R Archive Network)<sup>2</sup>. A CRAN számlítogépei tárolják a nyílt forráskódú R nyelv és környezet különböző verzióinak kódjait és dokumentációját, így az összes R csomag forráskódját is. Egy bírálati folyamat után bármely felhasználó csomag 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 idézőjelben 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<sup>3</sup>, ahol alaposan tesztelt és igen jól dokumentált bioinformatikai témaúj csomagokat találunk. Az innen elérhető csomagokat – például most a **DESeq2** csomagot az RNS-szekvenálási elemzésekhez – a következő parancsokkal telepíthetjük:

<sup>2</sup> <https://cran.r-project.org/web/packages/>

<sup>3</sup> <https://www.bioconductor.org/>

```
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 szűrhatunk be az RMarkdown állományainkba. Ezzel a sorral telepíthetjük a csomagot:

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

Fontos tudnunk, hogy a csomagok telepítésére egy számítógépen egy adott R verzióban belül csak egyszer van szükség. A telepítő parancsainkat azonban érdemes megőrizni, ugyanis egy új R verzióban könnyebben tudjuk így telepíteni a korábban használt csomajainkat. Nagyon fontos, hogy a telepítő parancsok futtatása után, tegyük azokat megjegyzésbe, vagyis írunk előjük kettőskereszt (#) karaktert (részletesebb információkat a megjegyzésekrol a **???** fejezetben olvashatunk). Ezzel tudjuk megvédeni ezeket a telepítő parancsokat az újból, 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.

### 3.1.4. 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-ról telepíthetjük az `install.packages()` parancs használatával. A Bioconductor-ról vagy a GitHub-ról származó csomagok telepítéséhez más parancsokat kell használnunk.

### 3.1.5. 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



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**, **purrr**, **tibble**, **dplyr**, **tidyr**, **stringr**, **readr** és **forcats**) vannak szétosztva, mindenki 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 packages -----
#> tidyverse 1.3.0 --
#> <U+221A> ggplot2 3.3.0   <U+221A> purrr   0.3.3
```

```
#> <U+221A> tibble  3.0.0      <U+221A> dplyr    0.8.5
#> <U+221A> tidyverse 1.0.2      <U+221A> stringr 1.4.0
#> <U+221A> readr   1.3.1      <U+221A> forcats 0.5.0
#> -- Conflicts ----- tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()   masks stats::lag()
```

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:
#>
#> * lubridate (1.7.4 -> 1.7.8)
#> * purrr     (0.3.3 -> 0.3.4)
#> * xml2       (1.2.5 -> 1.3.1)
#>
#> Start a clean R session then run:
#> install.packages(c("lubridate", "purrr", "xml2"))
```

Például a fenti esetben 3 csomag frissítését 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.

### 3.2.1. Összefoglalás



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

### 3.2.2. Feladatok



1. Keressünk rá a *Tidyverse R* csomajaira, 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



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 mindenkomponenst intenzíven fejlesztik, újabb és újabb funkciókat építenek be, és az esetleges hibákat rendre javítják a frissebb változatokban. Az *Alap R* évente kb. négyeszer 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<sup>4</sup> 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 letöljük 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 hárrom lépésekben foglalhatjuk össze.

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

<sup>4</sup> <https://cran.r-project.org/>

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

2. Telepítük 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.

```
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-ról származó csomagok telepítő parancsait mindenkor 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<sup>5</sup>, töltsük le az operációs rendszerünknek megfelelő változatot és indítsuk el a telepítőt. Szerencsére a régi *RStudio* beállításait öröklí 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)
```

### 3.3.4. 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

<sup>5</sup> <https://rstudio.com/>

*RStudio* friss verziója a hivatalos honlapokról szerezhető be. A csomagok frissítéséhez használjuk az `update.packages(ask=FALSE)` parancsot.

### 3.3.5. Feladatok

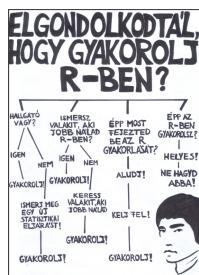


1. Az `RStudio Tools/Check for Package Updates` menüpontjával tájékozódunk a telepített csomagjaink állapotáról. Végezzük el a szükséges frissítéseket! Mit tegyünk, ha nem sikerül valamelyik csomag telepítése?
2. Ismerjük meg a telepített csomagjaink számát és forrását (CRAN vagy Bioconductor vagy GitHub)!



4. fejezet

## Munka az R-ben



## 4.1. Az RStudio használata



Ebben a fejezetben:

- megismerjük az RStudio jellemzőit és felépítését,
  - a konzolos és parancsállományos használat különbségeit,
  - a parancsállományok és az RMarkdown 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 hajthatjuk 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 kérdéseket. Korábban láttuk, hogy az *Alap R* telepítésével elérhetővé válik a *konzol*, ahol a 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 **??**. és **??.** 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 bizonnal egyet értenek azok, akik a fenti sorok begépelését és végrehajtássát 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, mint ha közvetlenül a konzolba gépelettük volna be őket. Ezeknek a szöveges állományoknak két fajtáját ismerjük meg ebben a könyvben: a *parancsállományokat* és az *RMarkdown állományokat* (a **??.** fejezetben az *RMarkdown* állományokról többet olvashatunk).

A parancsállományok és az *RMarkdown* 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 *RMarkdown* állományaiiba. 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 (??.. ábra).



1

**4.1. ábra.** Az R kényelmes használata

Az *RStudio* elsősorban a parancsok írását könnyíti meg, segítségével a parancsok létrehozásához kapunk rendkívüli segítséget. Megjegyezzük, hogy az *RStudio* egy üzleti vállalkozás neve is egyben, 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.
- **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, RMarkdown állományokat, képállományokat és dokumentációkat –, egyetlen könyvtárba foghatjuk össze, és a forráskódból relatívan hivatkozhatunk ezekre az állományokra.
- **Publikálás támogatása.** Az RMarkdown segítségével kényelmesen és reprodukálható módon hozhatunk létre például PDF, HTML és Word formanyelvű do-

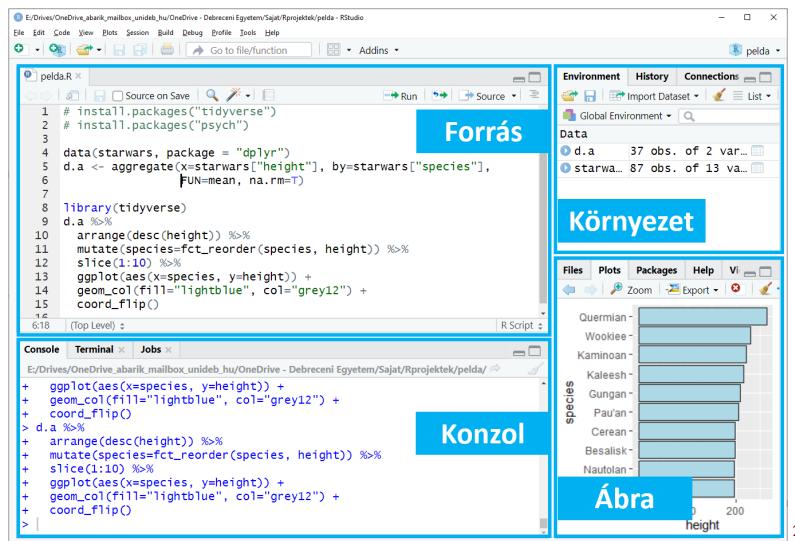
kumentumokat, vagy PDF, HTML és PowerPoint bemutatókat.

- **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 minden nap munkához szükséges ismereteket most bemutatjuk.

#### 4.1.2. Az RStudio felépítése

Az RStudio indítása után egy több panelból 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 / New R Script menüpontot, amely egy új parancsállomány létrehozását kezdeményezi. E lépés után már biztosan a négy-paneles, ???. 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, egyszerűen a kattintáshoz a 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

A legtöbb időt a *Forrás* nevű bal felső panelben töltjük, mert alapértelmezetten itt jelennek meg a parancsállományok és az RMarkdown állományok lapjai. Az R parancsainkat tehát ide írjuk. Az RStudio első indításánál ez a panel üres, de a további indítá-

soknál a korábban szerkesztett, de be nem zárt lapok automatikusan megnyílnak. Itt helyeztünk el korábban egy parancsállomány lapot a `File / New file / New 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 rárérdezünk 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 mindenkorban egyszerre egy parancs begépelésére és végrehajtására van lehetőséget kínál, lényegében egyszerre egy parancs begépelésére és végrehajtására van lehetőségünk. Ez lényegesen eltér a *Forrás* panel parancsállomány vagy *RMarkdown* 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 *RMarkdown* állományok R parancsait is valahogyan át kell ide irányítani, úgy mint-ha 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.

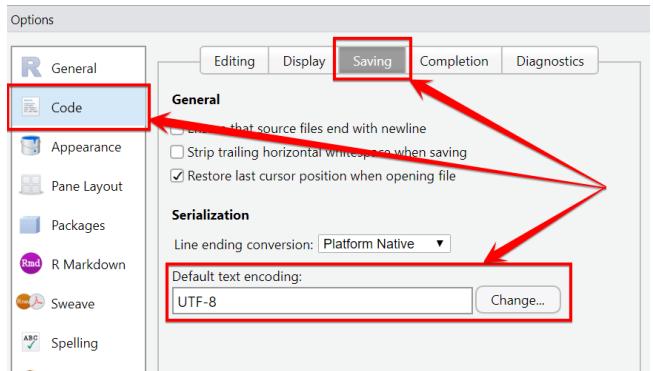
### 4.1.3. Az RStudio beállításai

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

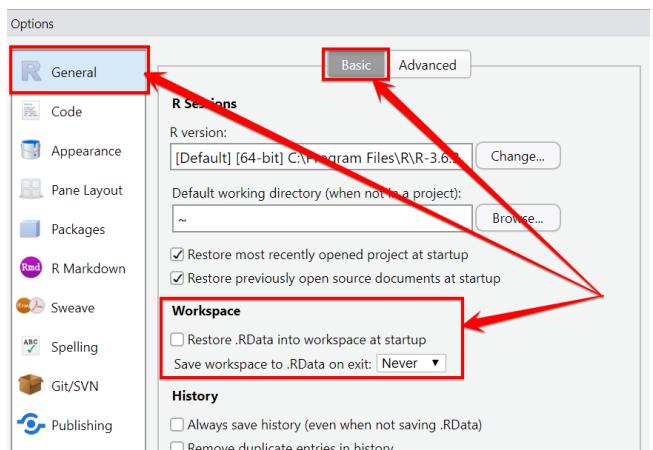
**UTF-8 kódolás beállítása.** A fenti menüpont kiválasztása után a bal oldali listából a `Code`, majd a fenti opciók közül a `Saving` opciót válasszuk. A [???](#) ábrán is látható módon, érjük el, hogy a `Default text encoding` alatt az `UTF-8` legyen kiválasztva. Fontos, hogy minden szöveges állományunk 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 ([???](#) ábra). Az *RStudio* projekt szemléletű használata mellett erre a mentési funkcióra nincs szükség.

**Az output megjelenítésének tiltása az RMarkdown lapon.** A bal oldalon az `RMarkdown` menüpont kiválasztása után vegyük ki a pipát a `Show output inline for all R Markdown`



**4.3. ábra.** Az UTF-8 beállítása az RStudio-ban



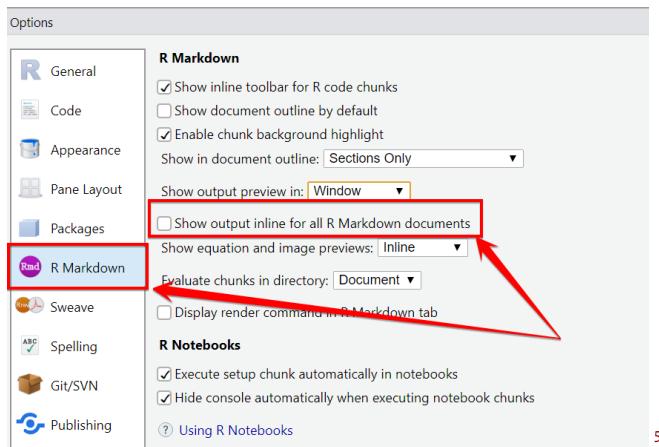
**4.4. ábra.** A munkaterület automatikus mentésének tiltása az RStudio-ban

documents elől (??.. ábra). Ez a beállítás gördülékenyebb szerkesztést biztosít az RMarkdown lapokon.

Opcionális lehetőségekkel 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.

#### 4.1.4. Az RStudio konzol

Az RStudio konzolja a Konzol panel egyik lapján található (??.. 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 közép-



**4.5. ábra.** Az output megjelenítésének tiltása az RMarkdown lapon

pontjában áll a munka során.

Közvetlenül azonban nagyon ritkán gépelünk parancsot a konzolba, erre a *Forrás* panel parancsállomány vagy RMarkdown 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 verziósáma, 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 parancssal:

```
> 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
```

Sót, 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 nyomunk `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"` karakterszorozatot, 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 gépelett 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 minden 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. Egyszer a korábban végrehajtott parancsainkat visszahívhatjuk, lapozhatunk bennük előre, hátra. Erré a `FEL/LE NYÍL` 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 szövegkiszort 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 névről 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 parancssal, 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 *parancsállománynak* vagy *szkriptállománynak* nevezzük. Ilyen szöveges állományok létrehozására tetszőleges szövegszerkesztő alkalmaz, 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 / New R Script` menüponttal történik. Parancsállományok mentésére és már létező megnyitására is van lehetőségünk a megfelelő menüpont kiválasztásával (`File / Save` és `File / Open`).

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 újsor beszúrása, ami lényegesen különbözik a konzolos használat parancs végrehajtási funkciójától. A parancsaink interaktív végrehajtáráért 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 / New 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 szövegküzorral állunk 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 szövegküzor 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 kétsége, 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 adataelemzési 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 / New R Script`) és gépeljük be a megoldást jelentő sorokat.

```
# A Csillagok háborúja c. film karaktereinek átlagos testmagassága
# Abari Kálmán
# 2022. 07. 06.

# 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 megjegyzésekkel is becsempeztünk, 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 megjegyzések rövidítéséről a `???` fejezetben olvashatunk). Futtassuk a sorokat a `Ctrl+Enter` segítségével, fassuk át a kiszámolt átlagos testmagasságokat az `output mean` oszlopában, majd mentseük 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 adataelemzési feladatunkat megoldottuk. Vajon lehet ezt ennél jobban csinálni? Igen!

#### 4.1.7. RMarkdown állományok

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

Hozzunk létre az RStudio-ban a `File / New File / R Markdown` menüponttal egy új RMarkdown állományt. A megjelenő dialógusdobozban töltük ki a `Title` és `Author` mezőket, azaz adjunk címet és szerzőt a dokumentumhoz, majd kattintsunk az `OK` gombon. A `Forrás` panelen megjelenik egy új RMarkdown lap, amely egy alapértelmezett tartalommal jön létre, és nem üresen, mint a parancsállományok esetében. Említettük, hogy az

RMarkdown állományok szerkezete kötöttebb, ez az alapértelmezett tartalom az eligazdásban segít minket. Érjük el, hogy az új RMarkdown ezeket a sorokat tartalmazza (a szerző neve a sajátunk legyen):

```
---
title: "A Csillagok háborúja c. film karaktereinek átlagos testmagassága"
author: "Abari Kálmán"
date: '2022. 07. 06.'
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

Adatok beolvasása és az átlagok kiírása

```{r}
# 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 RMarkdown állomány egy fejléccel kezdődik, amelyet a --- karakterek határolnak. A természetes nyelvű szöveget szabadon a fejléc alatti részben bárhol írhatjuk, az R parancsokat azonban ún. R csonkokban kell elhelyeznünk, amelyeket speciális kezdő és záró sorok határolnak. A ???. fejezetben részletesebben olvashatunk ezekről. Most elégünk meg annyival, hogy egy RMarkdown 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 RMarkdown állományunk második csonkjában lévő két R parancsot hajtsuk végre két Ctrl+Enter segítségével. A mini-elemzés eredménye ismét a konzolban látható.

Hogyan foglalhatnánk össze a parancsállományok és az RMarkdown állományok közötti különbséget? A ???. táblázatban láthatjuk, hogy minden 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 adataelemző R parancsokat.

Az R parancsokat szabadon írhatjuk a parancsállományokba, viszont a fejléc információt és a magyarázó szövegeket megjegyzésbe kell tenni. Az RMarkdown állományok-

ba 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 az RMarkdown állomány összehasonlítása

| Tartalom                  | Parancsállomány (.R) | RMarkdown (.Rmd) |
|---------------------------|----------------------|------------------|
| <i>Fejléc szöveg</i>      |                      |                  |
| cím, szerző, dátum        | megjegyzésbe         | fejlécbe         |
| <i>Magyarázó szöveg</i>   |                      |                  |
| természetes nyelvű szöveg | megjegyzésbe         | bárhová          |
| <i>Adatelemzés</i>        |                      |                  |
| R parancs                 | bárhová              | R csonkba        |

Valóban annyiban áll a különbség a két állománytípus között, hogy a máshová és más-hogyan írjuk az R parancsokat és az egyéb magyarázó/fejléc szövegeinket? Nem. A **??.** fejezetben részletesen bemutatjuk, hogy az RMardown állományok ereje abban van, hogy egy fordítási folyamat (*knit*-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.

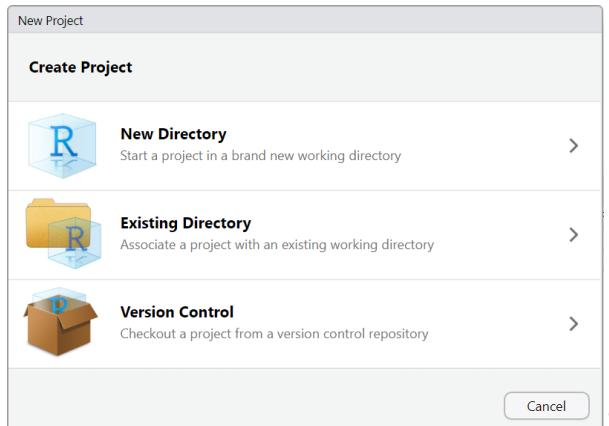
#### 4.1.8. Projektek használata

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

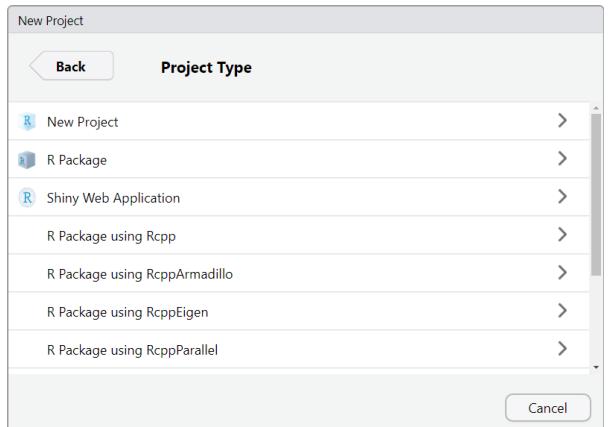
Az *RStudio* lehetőséget ad arra, hogy minden egyes adatfeldolgozási feladatunkhoz egy projektet rendeljünk. Egy *RStudio* projekt minimálisan egy projekt könyvtárat és az ebben lévő **.Rproj** kiterjesztésű projektállományt jelenti. Ezeket a következő módszerrel hozhatjuk létre. Először kattintsunk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

Két nagyon fontos dolgötörtént a fentiek hatására. Egyrészt a számítógépünkön létrejött az *elso\_projekt* projektkönyvtár, és benne az *elso\_projekt.Rproj* projektállomány, másrészt az *RStudio* ún. *projekt üzemmódba* 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



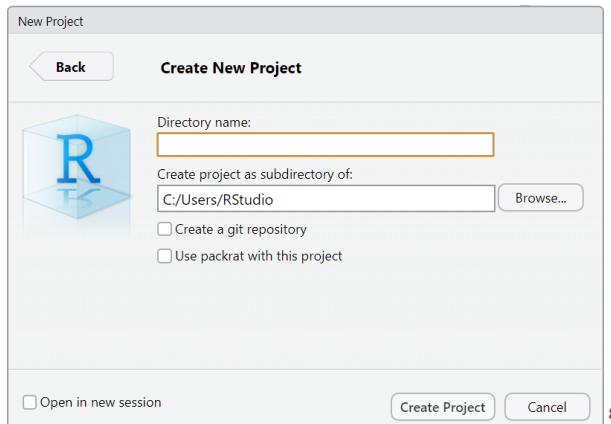
**4.6. ábra.** RStudio projekt létrehozása : 1. lépés



**4.7. ábra.** RStudio projekt létrehozása : 2. lépés

most az `első_project` feliratot látjuk, de amennyiben nincs aktív projektünk, akkor a `Project: (none)` feliratot olvashatjuk. Kerüljük a projekt nélküli állapotot.

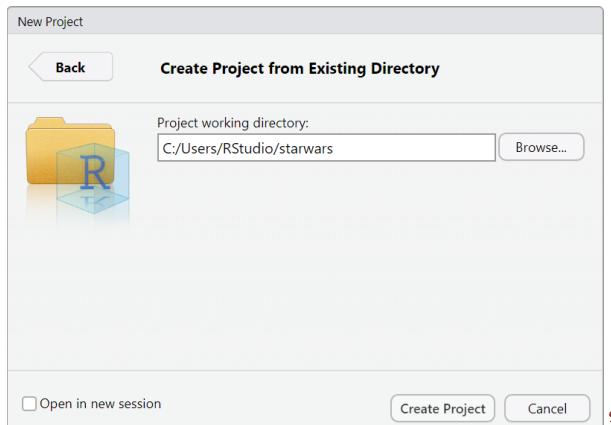
Minden adatfeldolgozási feladathoz – még a legkisebbhez is – hozunk létre projektet. minden állományt, amely a feladathoz tartozik a projektkönyvtáron belül helyezzük el. Milyen állományok jöhetsz szóba: például parancsállományok, RMarkdown állományok, adatállományok, képállományok, dokumentációk és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



**4.8. ábra.** RStudio projekt létrehozása : 3. lépés

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

Korábban létrehozott könyvtárból szintén a `File / New Project` menüpont segítségével hozhatunk létre RSudio projektkönyvtárat. Itt azonban az `Existing Directory` opciót kell kiválasztani (???. ábra). Ezt követően ennek a létező könyvtárnak az elérési útját kell megadnunk az ???. 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 felismerni, 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` / `Le` (és `Shift+Jobbra` / `Balra`): a cursor 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` / `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 cursor 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 funkciókhöz.

#### 4.1.10. Összefoglalás



Az adatelemzési munka során az RStudio -t használjuk projekt üzemmódban, miközben RMarkdown állományba gyűjtjük az elemző R parancsokat és az egyéb magyarázó és fejlec szövegeket. Ebben a fejezetben ezt a téTELmondatot töltöttük meg tartalommal. Megismertük az RStudio integrált környezetét. A Forrás panel lehetséges lapjai a parancsállomány, az RMarkdown 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 `.Rmd` kiterjesztésű RMarkdown állományba rögzítük parancsainkat. Az *RStudio* rutinszerű használatához a billentyűparancsok ismerete is hozzátarozik. A projektszemlélet az adatelemzéssel kapcsolatos állományok egyben tartásáról, és a hordozhatóság biztosításáról szól.

#### 4.1.11. Feladatok



1. Bizonyosudjunk meg róla, hogy az alapvető szövegszerkesztési ismeretek birtokában vagyunk. Ismerjük az `Insert` billentyű funkcióját? Találunk legalább 8 módszert, amely kizárolag a billentyű segítségével mozgatja a szövegkijelölésnek minden billentyűparancsait ismerjük? Milyen karaktertörlési lehetőségeket ismerünk? Ismerjük mindenhangot vágóasztal-művelet billentyűparancsát?
2. Az *RStudio* mellett minden más intergrált fejlesztőszközök 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



Ebben a fejezetben:

- megismерjük az R hivatalos dokumentációit,
- az ún. cheet-sheet-ek forrását,
- 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*<sup>10</sup>), melynek tanulmányozása rendkívül nagy lépést jelenthet az R alaptudás

<sup>10</sup> <https://cran.r-project.org/doc/manuals/r-release/R-intro.html>

megszerzéséhez. Az említett menüpont alatt találjuk még a *contributed documentation*<sup>11</sup> linket is, amely számos rövidebb, és hosszabb dokumentációt tartalmaz, angol és más nyelveken. Itt találjuk Solymosi Norbert nagyszerű magyar nyelvű R bevezetőjét<sup>12</sup> 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 téma kör legfontosabb tudnivalóit. Az *RStudio Help / Cheatsheets* menüjéből, vagy közvetlenül a <https://www.rstudio.com/resources/cheatsheets/> címről számos R téma cheat-sheet-jét érhetjük el.

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

```
help.start()
```

parancsal, ahol számos, az R nyelvet részletesen tárgyaló dokumentum közül választ-hatunk.

Ha csak egyetlen függvényteljesen 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"
```

<sup>11</sup> <https://cran.r-project.org/other-docs.html>

<sup>12</sup> <https://cran.r-project.org/doc/contrib/Solymosi-Rjegyzet.pdf>

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 lehetőséggéként 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)
```

### 4.2.1. Összefoglalás



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

### 4.2.2. 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



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 ([??.](#) ábra).
- MacOs környezetben indítsuk el az *R.app* alkalmazást, amely egyetlen konzolt tartalmaz.
- Linux környezetben az *r* parancssori futtatásával szintén egy konzolt kapunk.

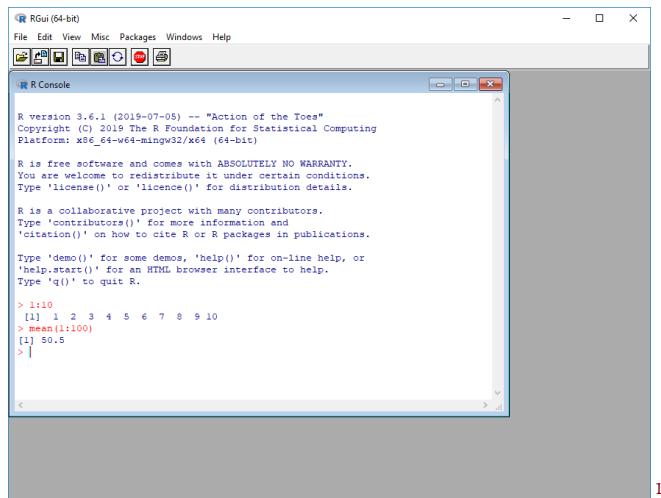
#### 4.3.1. A konzol használata

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 [??.](#) ábrán a Windows környezetben használható *RGui* alkalmazás látható, miután a konzolba két parancsot gépelünk 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 [??.](#) ábrán). Az *RGui* nagyszerű tulajdonsága, hogy támogatja a parancsállományok használatát. Az *RGui*-ban találunk menüpontokat (*File/New script*, *File/Open script* és *File/Save*),



13

**4.10. ábra.** RGui alkalmazás a konzollal Windows környezetben

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 parancsokat vezeti át a konzolba. Az RGui-ban ez a **Ctrl+R** billentyűkombinációval lehetséges – ez gyakorlatilag az RStudio-beli **ctrl+Enter** –, de az **Edit/Run line or selection** vagy az **Edit/Run all** 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 (az RMarkdown 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 eddigiek től 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*<sup>14</sup> és a *JASP*<sup>15</sup>. Mindhárom felsorolt eszközben közös, hogy grafikus felhasználói felületen mozgunk, és egérkattintással, menüben való navigálással, vezérlőelemek (rádiógombok, jelölőnégyzetek, listák, nyomógombok, beviteli mezők)

<sup>14</sup> <https://www.jamovi.org/>

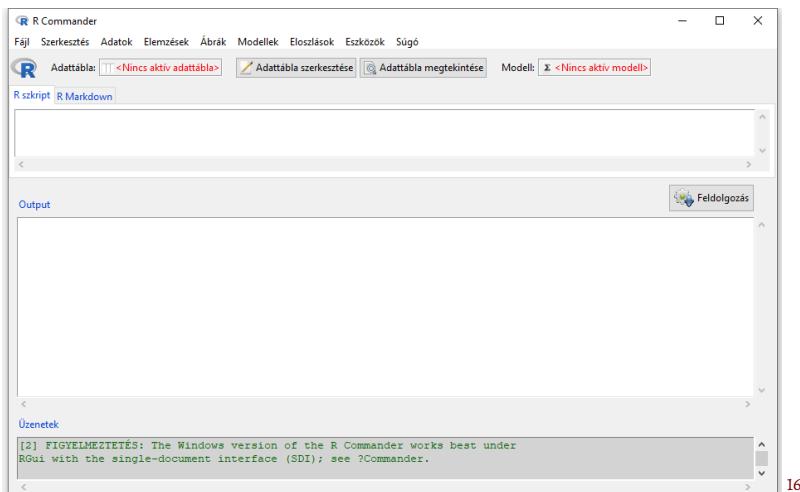
<sup>15</sup> <https://jasp-stats.org/>

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 (??. ábra), melynek felépítése fentről lefelé a következő: (1) a gazdag menürendszer, (2) az eszközötér az aktuális adatbázis (Adattábla) mezővel és az Adattábla megtékinthető gombokkal, (3) a parancslomány vagy RMarkdown lapok, (4) az output számára fenntartott szöveges mező, és (5) az üzenetek helye. Megjegyezzük, hogy a ??-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

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 egérkattintással 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 *R Commander* segítségével. Válasszuk ki a *Eszközök / Csomag telepítése* menüpontot, ha szükséges válasszunk a tükörszerverek közül, majd válasszuk ki a megjelenő listából a **dplyr** csomagot. Ezt követően töltük be a **dplyr** csomagot az *Eszközök / Csomag(ok)* betöltése menüponttal. Keressük meg a listában a **dplyr** csomagnevét 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 elmulogatott tevékenységeinknek megfelelő *R* parancsokat, illetve az output és üzenelek részben ezek végrehajtásáról is értesítést kapunk.

Még egy rendkívül fontos dollog 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 minden 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 elmulogatott tevékenységek *R* parancsait is szorgalmasan gyűjt, 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 (pluginek) 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 funk-

cíokat 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<sup>17</sup> részletesebb információkat olvashatunk.

#### 4.3.4. Kötegelt futtatás

Ha felidézzük az eddig tanultakat az R használati módjairól, akkor világos, hogy minden egyik 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 RMarkdown állományok esetén, és 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 mindenkor 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 RMarkdown 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 minden 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.

<sup>17</sup> <https://socialsciences.mcmaster.ca/jfox/Misc/Rcmdr/>

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

#### 4.3.5. 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 a `.R` kiterjesztésű parancsállományok kötegelt feldolgozását segítik. Az *R Commander* parancsok gépelése nélkül teszi lehettővé elemzések végrehajtását, minden összes menüpontot kell kiválasztani, majd a dialógusdobozban elvégezni a szükséges beállításokat. Érdemes ki próbálni a *jamovi* és a *JASP* statisztikai programokat is.

#### 4.3.6. Feladatok



1. Foglaljuk össze az R használati módjait! Soroljuk fel minden lehetőséget!
2. Hasonlítsuk össze a parancsállományok használatát *RGui*-ban és *RStudio*-ban!
3. Hasonlítsuk össze a parancsállományok és az *RMarkdown* 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 videó tutoriált az eszközök használatáról. Melyik eszköz tetszik jobban? Miben hasonlítanak és miben térnek el?



## 5. fejezet

# Az R nyelv



Az előző fejezetekben megismertük az R környezetet, az *Alap R*, az *RStudio* és a csomagok telepítését, megtanultuk a projektek, parancsállományok és RMarkdown 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.Rmd* RMarkdown állományt (`File / New File / R Markdown`) és egy *gyakorlas.R* parancsállományt (`File / New File / R Script`). A fejezet példáit egyaránt gépeljük az RMarkdown 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



Ebben a fejezetben:

- áttekintjük az egyszerű számolási lehetőségeket R-ben,
- bevezetjük az aritmetikai operátor és a kifejezés fogalmát,
- megismérjük 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 szöges zárójelben lévő sorszámot is láthatunk ([1]), amely bonyolultabb outputokban segít eligazodni. Később az ?? fejezetben visszatérünk a [1] értelmezésére.

Látjuk, ebben az esetben az R úgy viselkedik, mint egy számológép. Kiszámolja a parancssorra 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.

### 5.1. táblázat.

**Matematikai operátorok precedenciájuk csökkenő sorrendjében**

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

```
(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 kifejezések vagy – utalva arra, hogy a kifejezés értéke szám – *aritmetikai kifejezések* nevezünk. Az eddigiek alapján az aritmetikai kifejezések tehát a következő nyelvi elemeket tartalmazhatják:

- számok, amelyeket *numerikus konstansoknak* nevezünk,
- műveleti jelek, amelyeket *aritmetikai operátoroknak* nevezünk,
- és kerek zárójelek.

A fentiek alapján összetettebb aritmetikai kifejezéseket is megformálhatunk. 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
```

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

ciájú hatványozás az elsőként végrehajtott művelet.

```
-2^2      # először hatványozás, majd előjele
#> [1] -4
(-2)^2    # először előjele, 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 objektumok),
- általánosítható a kifejezés művelet része, amelyet eddig az operátorok jelenítettek meg (ezek lesznek a függvények).

Az adatrész általánosítása tehát az adatobjektum (vagy röviden objektum), a műveletek 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<sup>1</sup> kell létrehoznunk. Az objektumok révén a memoriába 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 memoriában tárolt értékével fog számolni az R.

<sup>1</sup> Más programozási nyelvekben az „objektum” helyett a „változó” elnevezést használják, de a változó fogalma már foglalt a statisztikában, így szerencsésebb a memoriában tárolt adatokra objektumként hivatkozni.

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

Minden objektumnak van neve és tartozik hozzá a memóriában egy terület, ahol a kérdezés érték tárolásra kerül. Esetünkben az objektum neve `x`, a hozzá tartozó memóriaterületen pedig a 97 értéket tárolja az R. Az objektum leegyszerűsítve tehát egy névérték párt, 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 az objektum értékének megváltoztatása.

#### 5.1.2.1. Objektumok létrehozása

Objektumot értékkadással hozhatunk létre. Az értékkadás tartalmaz egy értékkadás operátort, melynek alakja `<-` (balra nyíl), vagyis egy kisebb jel és egy mínusz előjel egymás után írva szóköz nélkül<sup>2</sup>.

Az értékkadás általános alakja:

```
objektumnév <- kifejezés    # értékkadó utasítás
```

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

Ha olyan objektumnevet szerepeltetünk az értékkadásban, amely még nem létezik, akkor az R létrehoz egy ilyen nevű új objektumot, és a hozzá tartozó memóriaterületen pedig az értékkadás 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 mindenkor, amíg ezen nem változtatunk. A munkánk során létrehozott objektumok a memória egy speciális területére, a *munkaterületre* (*workspace*) kerülnek.

<sup>2</sup> További értékkadás operátorok a `->`, `<<-`, `->>` és `a =`. Ezeket nem használjuk ebben a könyvben.

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
```

### 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, másról megkapjuk az objektum memóriában tárolt értékét.

```
a      # vajon mi az "a" objektum értéke
#> [1] 3.333
```

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` parancssal ismerhetjük meg). Hagymányosan a pont karaktert használjuk az objektumnevekben a tagolásra (például `magassag.peter` magasságának tárolására). 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 minden két 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 kissé 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, minden egyik 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 kettőskeresztsz (#) karakter használatával vezetünk be. Az R értelmező a kettőskereszttől a sor végéig tartó részt figyelmen kívül hagyja. Itt helyezhetjük el a parancsal 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 kizárolag magyarázó szövegek kerülhetnek megjegyzésbe, sokszor R parancsok végrehajtását akadályozzuk meg ezzel a módszerrel. Úgy kerülhetjük el egy parancs végrehajtását, hogy nem kell kitörönünk a parancsállományból vagy az RMarkdown állományból, egyszerűen csak megjegyzésbe kell tennünk őket. Emlékezzünk vissza, hogy az **???** fejezetben a csomagok telepítéséért felelős parancsok esetében kifezetten 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 megemlíjtük, hogy az RStudio-ban egyszerre több kijelölt sort tudunk megjegyzésbe tenni, vagy onnan kivenni a **Ctrl+Shift+C** segítségével.

### 5.1.4. 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

munkamemóriában. Ehhez az értékkadó 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 kettőskereszt (#) segítségével helyezhetünk el.

### 5.1.5. Feladatok



1. Gondoljuk át, mi lehet a következő algebrai kifejezés eredménye, majd el- lenőrizzük 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 = TRUE)` 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 RMarkdown (.Rmd) állomány szerkesztése közben?

## 5.2. Függvények



Ebben a fejezetben:

- áttekintjük a függvényhívás lehetőségeit a nevesített argumentumokkal, az alapértelmezésekkel és az argumentumok sorrendjének megváltoztatásával,
- megsmerjük a legfontosabb matematikai függvényeket,
- és pontosítjuk a kifejezés fogalmát.

Az aritmetikai kifejezéseinben 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.

```
2^0.5
#> [1] 1.414
```

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

```
sqrt(2)
#> [1] 1.414
```

A függvények valamelyen 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)
```

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 (?: 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

### 5.2. táblázat.

Az R alapvető matematikai függvényei

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

```
exp(1)      # Euler-féle szám, a természetes logaritmus alapja
#> [1] 2.718
```

parancsból. Ezt az iracionális 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 logaritmus
#> [1] 0.6931
```

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

```
log(x=2)    # 2 természetes alapú logaritmus
#> [1] 0.6931
```

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 lehetőség van, 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] 1.737
#> [1] 1.673
```

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] 1.737
#> [1] 1.673
```

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 pontosíthassuk: **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 mindenkor egy értéket eredményez, ez lesz az utasítás értéke. Még akkor is rendelkezik értékkel az utasításunk, ha az nem jelenik meg a parancssorban, például az értékkadó utasítás értéke a jobb oldali kifejezés értéke. Ezért írhatjuk a következő parancsot:

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

Amennyiben egy értékkadás, mint kifejezés értékét szeretnénk megjeleníteni a konzolban, akkor tegyük kerekzárójelbe 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 mindenike 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.

### 5.2.3. 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 véghajtása.

#### 5.2.4. Feladatok



1. Tekintsük át az **??.** táblázat utolsó oszlopában szereplő R függvényeket. Próbáljuk megjósolni a függvények visszatérési értékét. Végezzünk ellenőrzést: gépeljük be, és hajtsuk végre a matematikai függvényeket! Egészítük ki a begépelt matematikai függvényeket az argumentumok nevével, mindegyik argumentumnak adjunk nevet az **??.** táblázat első oszlopá alapján!
2. Az előző feladatban a matematikai függvények gépelése során minden *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 hanyadik sorába írnánk címet 12-es és 16-os betűméret esetén? Próbáljuk ki mindenzt 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



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 könyünk kulcsfontosságú fejezetébe, az adatszerkezetek tanulmányozásába. Legyünk alaposak az itt szereplő témaörök 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 (objektu-

### 5.3. táblázat. Numerikus konstansok írása

| 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             |

mokból), 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 adat-elemzési munka kiinduló pontja maga az adat. Eddig csak számszerű (numerikus) adatokkal találkoztunk, és azok közül is csak az egész számok leírására fókusztáltunk. 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 adat-feldolgozás során:

- *numerikus* típus, amely lehet *double* vagy *integer*, attól függően, hogy tizedestörteket 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.

A továbbiakban megismérjük, 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 ?? 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 törtrészet.

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

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

Használhatunk az R-ben exponenciális alakú és hexadecimális (16-os számrendszerű) 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  $\epsilon$  karakter vágja ketté, egy bal oldali és egy jobb oldali részre. Az exponenciális alakú szám értéke: a bal oldali rész szorozva 10 annyit adhatványával, 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.

Az R automatikusan exponenciális alakra vált túl kicsi vagy túl nagy számok konzolbeli megjelenésénél. 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
0.000001                # 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. 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.

Egy karakteres konstans tetszőleges karaktert (betűt, számjegyet, írásjeleket, szóközt stb.) tartalmazhat, de az első két megadási forma esetében azt a határolójelet el kell elkerülnünk, amelyet az illető karakteres konstans létrehozásánál használtuk. 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 jellet (\, fordított perjel) kezdődő karaktersorozatokat, amelyeket speciálisan értelmez az R. A legfontosabb escape szekvenciákat és jelentésüket az **??** táblázat tartalmazza.

Természetesen, karakteres objektumokat is létrehozhatunk.

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

**5.4. táblázat.** Néhány escape szekvencia

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

**5.5. táblázat.** Néhány karakterkezelő függvény

| Függvény                   | Leírás                 | Példa                                |
|----------------------------|------------------------|--------------------------------------|
| paste(); paste0(sep = "")  | sztringek összefűzése  | paste("a", "b", sep = "")            |
| nchar(x)                   | karaktersztring 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\nn", sep = "")       |
| grep(); grepl(); regexpr() | részsztringek keresése | grepl(pattern = "lm", x = "alma")    |
| sub(); gsub()              | részsztringek cseréje  | gsub("lm", repl = "nyj", x = "alma") |

```
#> [1] "festő"
#> [1] "Érd"
```

Karakteres operátor az R-ben nincs, de számos karakterkezelő függvény segíti a sztringek kezelését (??. 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 egy-szerűbb típus, mivel itt összesen két érték tárolására van módunk. Ez a logikai *igaz* és

**5.6. táblázat.** Relációs operátorok

| 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   |

**5.7. táblázat.** Logikai operátorok

| 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   |

a logikai *hamis* érték, amelyek az R nyelvben a `TRUE` és a `FALSE` logikai értékeket jelentik. Az R a logikai értékek 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 (???. táblázat).

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

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

### 5.3.1.4. 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 tizedestörtek esetében pontot kell használnunk az egész és a törtrész elválasztására (például 12.34, -0.04, 3.12e+12). Karakteres konstansok esetében a következő formát 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.

### 5.3.1.5. Feladatok



1. Mi a hasonlóság a következő három numerikus konstans között: 0xabc, 2748, .2748e4.
2. Az R öt előre definiált konstassal 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 válzása is létezik, soroljunk fel még legalább öt lehetőséget (?Quotes)! Milyen előnyökkel rendelkezik ez a megadási forma az idézőjelek és afordított perjel tekintetében?
5. Helyezzük el idézőjeleket karakteres konstansokban, mindenhol megadási forma mellett!
6. Próbáljuk ki az ?? táblázat karakterkezelő függvényeit! Gépeljük be az utolsó oszlopban lévő példákat, és vizsgáljuk meg a függvények visszatérési értékét.
7. Próbáljuk ki az ?? táblázat relációs operátorait! Gépeljük be a példákat és ellenőrizzük az eredményeket.
8. A logikai operátorok működéséről teljes képet kaphatunk az ?? táblázatból. Próbáljuk ki ezeket a parancsokat is!

### 5.3.2. Áttekintés

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, mindenkor látjuk, az R legalapvetőbb adatszerkezetének, a vektornak 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* objektumok esetén tér el a visszatérési értéke, `class()` ugyanis ekkor a `numeric` outputot adja.

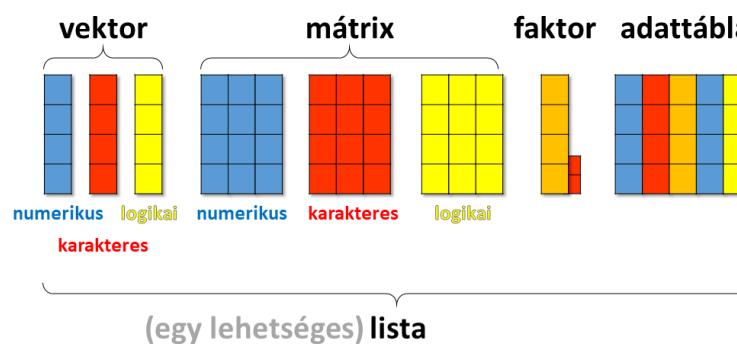
```
typeof(obj.double);      class(obj.double)
#> [1] "double"
#> [1] "numeric"
typeof(obj.integer);    class(obj.integer)
#> [1] "integer"
#> [1] "integer"
typeof(obj.karakteres); class(obj.karakteres)
#> [1] "character"
#> [1] "character"
typeof(obj.logikai);    class(obj.logikai)
#> [1] "logical"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

Az adatalemzési 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 memóriahelegen 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 adatszerkezetet kell számolnunk. Ebben a fejezetben az R leggyakrabban használt adatszerkezetet tekintjük át. Most felsoroljuk és jellemzzük 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 hosszságú vektorokat vagy faktorokat teszünk

egymás mellé, és így egy kétdimenziós szerkezetet kapunk, amelynek vannak sorai és oszlopai.

Az ?? 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 ?? ábrán egy 8 elemű lista jelenik meg, amelynek első eleme egy numerikus vektor, utolsó eleme pedig egy adattábla.



3

**5.1. ábra.** Az R legfontosabb adatszerkezetei

Az ?? 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 adat elemzések során. Mindegyik esetben megvizsgáljuk:

- 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.

### 5.8. táblázat. Adatszerkezetek

| 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   |

#### 5.3.2.1. 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 címeket feleltetünk meg. A *lista* elemi tetszőleges típusúak lehetnek. A *mátrix* ugyanúgy homogén, mint a *vektor* és a *faktor*. Az *adattábla* felfogható azonos elemszámú vektorok/faktorok listájának.

#### 5.3.2.2. Feladatok



- Próbáljuk ki az **???** táblázatban szereplő példákat. Hozzuk létre a különböző típusú objektumokat és vizsgáljuk meg a `typeof()` és `class()` függvényekkel az objektumok típusát.

### 5.3.3. Vektor

Az R legalapvetőbb adatszerkezete a *vektor*. A vektort egymás melletti (vagy alatti) célállban tárolt értékek sorozataként képzelhetjük el (**???** ábra), mely értékek mindenike 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 kizárálag *integer*, vagy kizárálag *double*, vagy kizárálag *karakteres*, vagy kizárálag *logikai* típusúak lehetnek.

### 5.3.3.1. Vektor létrehozása

Vektort legegyeszerűbben a `c()` függvényel hozhatunk létre, az argumentumlistában egymás 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()` függvényel 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 szerepeltejük. 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 írunk egy nevet és egy egyenlőségi jelet az argumentumként használt elem előtt. 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()` függvényel. Amennyiben az értékadás bal oldalán szerepeltekjük, 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()` függvényel. Az így létrehozott vektorok ugyanis valamilyen számtani sorozat egymást követő elemei, vagyis az egymás mellett lévő elemek különbsége (a lépésköz) állandó.

**5.3.3.2.1. 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, mindenkorral annyi igaz, hogy a sorozat vége a `stop` értéknél mindenkorral 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
```

**5.3.3.2.2. 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 lépésközöt 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 <- 1:20
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
```

**5.3.3.2.3. 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, mégghozzá 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 FALSE 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
```

**5.3.3.2.4. 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"  "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"  "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 összeolvashatjuk 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 nézzük 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

```

### 5.3.3.3. 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. Próbálunk 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()` függvénytel. 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 véghajtás 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 oldalon lévő vektorok elemszáma eltér, akkor az általános szabály az, hogy a rövidebb vektort az R megismétli mindaddig, míg a hosszabb vektor elemszámát el nem éri. Ha a rövidebb vektort nem egész számszor megismételve kapjuk a hosszabb vektor hosszát, akkor figyelmeztést kapunk az R-tól, 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 végrehajtásához.

```
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 közt hajtódik végre a kijelölt művelet (vagyis koordinátánkénti végrehajtás történik),
- különböző elemszámú vektorok esetében pedig először a rövidebb vektor ismétléssel kiegészül a hosszabb vektor hosszára, és ezt követi a koordinátánkénti végrehajtás.

Az operátorokon túl az `??` 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 logaritmus
#> [1] 0 1 2
x <- 1.3:10; round(x)        # 9 szám egészre kerekítve
#> [1] 1 2 3 4 5 6 7 8 9
```

#### 5.3.3.4. 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 értékhöz (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 érték `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

### 5.9. táblázat. Függvények vektorokkal

| 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            |

az jellemző, hogy a bemenő vektor elemeivel egy előre definiált műveletsorozatot hajtanak végre, például összehajtják 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 **??** táblázat tartalmazza.

#### 5.3.3.5. 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 függvéncsalád, amellyel megvizsgálhatjuk, hogy egy tetszőleges objektum az adott típushoz tartozik-e. Ez az *is.\*()* függvéncsalád, 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.\*()* függvéncsaláddal 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.

### 5.3.3.6. Az NA hiányzó érték

Korábbi példáinkban már felbukkant a hiányzó érték, amelyet az R-ben az NA jelöl. Az adatalemzési munkánkat végigkísérlik 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
```

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 muta-

tót 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
```

### 5.3.3.7. 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
```

### 5.3.3.8. Vektor indexelése

Fontos részhez érkeztünk, érdemes kicsit lassítanunk. Már nagyon sok minden meg-tanultunk 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 vektor-elem értékét 1-gyel.

```
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 1-gyel 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 írjuk.

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 meg-változtattuk az `y` értékét, mert újra az értékadás bal oldalán szerepel az `y` objektum. Az 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 ellenőrizhet-jük, 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 értékkedés bal oldalán. Értékkedés jobb oldalán viszont felbukkan, a második sorban a `z.uj` objektum létrehozásához használtuk fel a `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 értékkedést használunk. Ez az objektum lehet a kiindulásként 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-gel, 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.

#### 5.3.3.8.1. 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 kétszerére. A változást ellenőrizhetjük a konzolban.

Ha az `x` vektort az elemszámánál nagyobb indexssel 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 példában 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()` függvényekkel (vagy bármilyen más vektorlátrehozó függvényekkel), vagy a kettőspont (`:`) operátorral, de korábban létrehozott objektumot is használhatunk indexelésre (`x[y]`).

Az indexelés során több vektorelementet egy lépésekben 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 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 mindenkor egész szám, de az R megengedi, hogy tört értékeket tartalmazó indexvektort szerepeljünk 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
```

**5.3.3.8.2. Indexelés karakteres vektorokkal** ♦ Amennyiben egy vektor elemei rendelkeznek névvel, akkor karakteres indexvektorokat is használhatunk az indexeléshez. Ez meglehetősen nagy könnyebbéget 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.

**5.3.3.8.3. 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 jelezük, 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. Az `integer(0)` az üres vektort jelöli.

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
```

**5.3.3.8.4. 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 pozícióra 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.

### 5.3.3.9. 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.

```
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 műveettel 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 elemknél 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 problémákat. 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. Generálunk 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] 35 0 9 3 5 16 23 13 1 37
pontszamok[pontszamok > 36] # vektor szűrése
#> [1] 37 38 39 40 38 37 37 40 37 37 37
```

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

Más relációs operátorokat (**??.** táblázat) is használhatunk a szűrésben, sőt logikai operátorok (**??.** táblázat) segítségével tetszőleges természetes nyelven megfogalmazott feltételt át tudunk fordítani R logikai kifejezésbe. A logikai operátorokat tartalmazó logikai kifejezéseket összetet *logikai kifejezéseknek* nevezzük. Írassuk ki a `pontszámokat` 36 és 39 között, majd 3 és 6 között, és végül mindezeket együttesen.

```
pontszamok[pontszamok>=36 & pontszamok<=39]
#> [1] 37 38 39 38 37 37 37 37 37
pontszamok[pontszamok>=3 & pontszamok<=6]
#> [1] 3 5 6 5 4 4
pontszamok[(pontszamok>=36 & pontszamok<=39) | (pontszamok>=3 & pontszamok<=6)]
#> [1] 3 5 37 38 6 39 5 38 37 37 4 37 4 37 37
```

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 van nem 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 adatalemzés 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 értékkel. 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 minden hiányzó értéket azonos számmal szeretnénk felülírni, akkor elegendő lenne a `x[is.na(x)] <- 7` kifejezés is.

Korábban már említettük a `(??).` 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 minden két oldalán a logikai kifejezésekhez fűzzük hozzá a `& !is.na(x)` kifejezést, így tudjuk az `NA` értékeket eltávolítani az értékadás minden két oldaláról.

```
x <- c(33, NA, 32, 38, NA, 37)
x[x < 36 & !is.na(x)] <- x[x < 36 & !is.na(x)] + 1
```

### 5.3.3.10. Vektor rendezése

Egy vektor elemeit növekvő vagy csökkenő sorrendbe rendezhetjük. Az R-ben a vektor elemeit a `sort()` vagy az `order()` függvényel 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
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()` függvénytel, amely a bemeni vektor elemeit fordított sorrendben sorolja fel, szintén elérhetjük a csökkenő rendezettséget.

Ha a `sort()` függvénnnyel 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.

```
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.

### 5.3.3.11. 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()` függvényel,

vagy a kettőspont (:) operátorral készíthetünk. Megbeszéltük a vektorok közötti műveletek véghajtásának fő szabályát: ismétlésel 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()` függvénnnyel is elvégezhetjük.

### 5.3.3.12. 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-öt 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 létrehozha-

tunk 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 `??.` 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 minden egyik értéket 1-gyel, csak az 5-öket csökkentsük 1-gyel.

### 5.3.4. Mátrix

A mátrix adatszerkezet egyetlen lényeges dolgban különbözik a vektortól: a mátrix kétdimenziós, sorokba és oszlopokba szervezi az elemeket, míg a vektor egydimenziós (érdemes visszalapozni a `??.` 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  $4 \times 5$ -ös. A következő sorban az `ncol=10` argumentum már egy 40 elemű mátrix létrehozását kezdeményezi ( $4 \times 10$ -es), így az `1:20` vektor ismétlésével állnak elő a szükséges elemek. (Figyelmezhető, hogy ha a szükséges mátrixelemekszám 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 értékekkel 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 `??.` 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 operátor (`[]`) 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
#>      [,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  

#>      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  $3 \times 4$ -es mátrixot, amely 3 tanuló átlagát tartalmazza 4 tantárgyból.

```
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  

#>      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 átlaga
#> [1] 0.4618802
```

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

```
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.

```
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
```

#### 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 oszlovektorokból
#> [,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 elő 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 lépésként 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
```

#### 5.3.4.5. 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.

#### 5.3.4.6. Feladatok



1. Hozzunk létre egy csupa 1-ből álló mátrixot, amelynek 3 sora és 2 oszlopa van!
2. Hozzunk létre egy  $3 \times 4$ -es karakteres mátrixot, amely 12 különböző kezsznevét tartalmaz!
3. Hozzunk létre egy  $3 \times 4$ -es logikai mátrixot, amelynek 1. és 3. sora TURE a 2. sora pedig FALSE értékeit 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!

### 5.3.5. Faktor

A faktor adattípus nagyon hasonló a vektorhoz, ugyanis minden faktor egy speciális *integer* vektor, a faktor tehát homogén és egydimenziós adatszerkezet. Faktorokat előszörban 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 címkek között (az ??, ábrán ezt egy piros kis téglalappal jelöltük). Egy faktorelem értéke csak ezekből a címkekkel kerülhet ki, ami nagy fokú 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” címkeket veheti fel, más értéket nem (az NA hiányzó érték természetesen lehet faktorelem értéke is). A munka során minden a címkekkel 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()` függvényteljesítőjével. A faktor létrehozásánál minden gondoskodunk a faktor lehetséges értékeinek, vagyis a faktor címkei megadásáról. A címkeket néha (`factor()`)szinteknek (`levels`) is nevezünk. Mivel a kategorikus változóink lehetséges értékeit 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 kiiratása

#> [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 feltárolnak az `unclass(x.f)` outputjában. A szám-címke összerendelést magunk szabályozhatjuk, ha a `factor()` függvény `levels=` argumentumában módosítunk a sorrendet.

```
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 összerende-lést `levels=c("nő", "férfi")` paraméterrel (1-`"nő"`, 2-`"férfi"`)-re változtattuk. Láthat-juk, a címkék sorrendje a faktor értékeitől független, mégis fontos szerepet kap majd a táblázatok és ábrák megjelenítésénél, tehát érdemes rá odafigyni.

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.

```
(x.f.1 <- factor(c("férfi", "férfi", "nő", "férfi", "nő")))
#> [1] férfi férfi nő     férfi nő
#> Levels: férfi nő
(x.f.2 <- factor(c("férfi", "Férfi", "nő", "férfi", "nő")))
#> [1] férfi Férfi nő     férfi nő
#> Levels: férfi Férfi nő
(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 korábban látottakhoz, 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 min-tába, 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 minden össze egyet. A fenti hibák a `levels` szerepeltetésével könnyen kiküszöböltetők.

```
(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`- címkei kövessék ezt a sorrendet.

```
x <- c(1, 1, 0, 1, 0)      # numerikus vektor létrehozása, 0-nő, 1-férfi
(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 minden esetben "nő", az 1 "férfi" címkéhez fog vezetni. Az `x.f.1` és `x.f.2` faktorok minden 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 vektorok.

### 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 (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 = TRUE` 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()` függvénytel 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.

```
(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. 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 címkekét 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.

### 5.3.5.5. 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 végzettséggű, 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()` függvényel beszűdesebb faktorszinteket hoztunk létre.

### 5.3.5.6. 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.

### 5.3.5.7. Feladatok



- Hozzuk létre azt a karakteres vektort, amely a férfi, nő karakteres konstan-sokat, úgy helyezi el egymás mellett, hogy a 7 darab férfi érték után 13 db nő címke következik! Hány elemű az így létrejött vektor?
- Egy vizsgálatban az első 10 személy neme férfi, a többi 8 neme nő volt. Hoz-zuk létre azt a faktort, amely leírja a neme változót!
- Egy vizsgálatban városi („V”) és falusi („F”) fiatalok vettek részt! A megkér-dezettek településtípusa rendre a következő volt: F, F, V, F, V, V, V, F. Hoz-zuk létre azt a faktort, amely leírja a településtípus változót!
- 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. 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 kü-lönböző 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 (???. 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()` függvénytel 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 elnevezhetünk 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()` függvénytel 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 1 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 elérünk, 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 az második paraméterben szereplő függvény visszatérési értékei. Az `sapply()` hasonlóan jár el, de a visszatérési értéke egy vektor.

#### 5.3.6.4. Összefoglalás



A lista az R legszabadabb adatszerkezete, egydimenziós és inhomogén. Listát a `list()` függvénytel hozhatunk létre, melynek argumentumában tetszőleges adatszerkezetű objektumokat felsorolhatunk, ezek alkotják a lista egyes elemeit. Lista indexelése a `[]`, `[[]]` és `$` operátorokkal is lehetséges.

#### 5.3.6.5. 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 (???. 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()` függvénytel hozhatunk létre, amely azonos hosszú vektorokat vagy faktorokat vár az argumentumában. A `data.frame()` tehát listaszerűen konstruálja az adattáblát.

```
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  $3 \times 2$ -es. Mindkét argumentumot elneveztük (`nev`, `pont`), ezekből oszlopnevek lesznek. Az oszlopok elnevezéséhez az objektumneveknél használt szabályokat vegyük figyelembe (??. 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.

```
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 oszlobból álló adattáblát készítettünk (`df2`  $6 \times 3$ -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 legszerencsesebb. Kényelmesebb, ha az *RStudio* adatbázis ablakában jelenítjük meg az adattábla tartalmát, amit a *Környezet* panel megfelelő adatbázisnevén való kattintással és vagy a `View()` parancssal kezdeményezhetünk. Próbáljuk ki a `View(df)` és `View(df2)` függvényhívásokat.

Hasznos információ 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)           # sornévek
#> [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")  # sornévek 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 listák nál megtanult indexelési formákat jelentik. Az általános indexelése 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 oszlopkordinátákat 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-ból adattábla, majd adattábla
#>   pont.1
#> 4      6
#> 3      8
#> 2      7
#> 1      6
df2[4:1, 1:2]$tipus        # df2-ból adattábla, majd faktor
#> [1] B A B A
#> Levels: A B
df2$pont.2[1:3]            # df2-ból 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 valamelyen 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`" címkkel 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.

```
# 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-r-nyelv-9-summary}



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()` függvénnnyel lehetséges, ahol az argumentumban az oszlopokat alkotó vektorokat és faktorokat kell felsorolni. Az adattábla indexelése a mátrixoknál és a listáknál tanultak alapján lehetséges.

### 5.3.7.6. Feladatok



1. Hozzunk létre egy  $30 \times 3$ -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 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



Ebben a fejezetben:

- megismérjük a `tömb`, `táblázat`, `dátum`, `idő`, `időtartam` és `tibble` adatszerkezeteket,
- valamint a munkaterület és munkakönyvtár kezelésének függvényeit.

Az R legfontosabb adatszerkezetet megismertük az előző fejezetben. Az adatelemzés kiindulópontja az `adattábla`, amely a `mátrix` és a `lista` adatszerkezet előnyeit egyesíti, lényegében vektorok és faktorok egymásutánja. A munka során azonban találkozhatunk három vagy több dimenzióba szervezett adatokkal (`tömb` és `táblázat`), valamint szükség lehet `dátum`, `idő` és `időtartam` kezelésére is. A `Tidyverse` R megújította az adattáblát, és bevezette a saját `tibble` típusát az adatok szokásos 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, amelynek különböző mértékegységekben mu-

### 5.10. táblázat. Adatszerkezetek (folytatás)

| 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 |

tatja két időpont közötti különbséget.

- *tibble* - Speciális adattábla, amely a `Tidyverse` R része, és megkönnyíti az adatok kezelését.

Az `??`. táblázatban már korábban bemutattuk az R legfontosabb adatszerkezeteit, a Z `??`. 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.

## 5.4.1. Tömbök és táblázatok

A *tömb* a mátrix általánosításával nyerhető adatszerkezet. Az azonos típusú adatokat a mátrix két dimenzió mentén rendezи 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 dimenzióméretek mentén rendezи 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  $2 \times 3 \times 2$ -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  $\times$  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 használhatunk. Ha  $\times 3$  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()` függvénytel 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 ki-csít kényelmesebb formula argumentumot. Az R formula olyan kifejezés, amely tartalmaz egy tilde (~) karaktert, és annak két oldalán rendszerint egy adattábla oszlopnevei jelennének 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 képezzünk (`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 álnak 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 `tаб3` a kiinduló pontunk, amely egy táblázat. A táblázatot a korábban tanult módszerrel gyakoriságokat tartalmazó adattáblává alakítjuk, majd eseteket (nyers adatokat) tartalmazó adattáblává.

```

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
df3 <- as.data.frame(df3) # adattáblából 3D táblázat
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
#> 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 átalakítás 2 sora:

```

```
df.long <- df3[rep(row.names(df3), df3$Freq), c("Var1", "Var2", "Var3")]
rownames(df.long) <- seq_along(rownames(df.long))
head(df.long) # az első 6 sor kiírása
#>   Var1 Var2 Var3
#> 1 Female Left Freq
#> 2 Female Left Freq
#> 3 Female Left Freq
#> 4   Male Left Freq
#> 5   Male Left Freq
#> 6   Male Left Freq
```

### 5.4.2. Dátum és idő

Az adatalemzés 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           # datum kiírása
#> [1] "2022-07-22"
typeof(datum)       # datum típusa
#> [1] "double"
class(datum)        # datum típusa
#> [1] "Date"
unclass(datum)      # datum alapja
#> [1] 19195
```

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 van, 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
```

**5.4.2.1.1. 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<sup>4</sup>) alak mellett sok olyan forma létezik, amelyben elválasztó karakterként a perjel vagy a pont szerepel, valamint az év-hó-nap hármás sorrendje is változhat. A konkrét dátum értelmezéséhez az `as.Date()` függvény `format=` argumentumát kell helyesen beállítani. A használható kódokat a `??.` táblázat tartalmazza.

#### 5.11. táblázat. Formátumkódok a dátumokban

| 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 verzióink 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
```

<sup>4</sup> [https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601)

```
#> [1] "2020-04-12"
as.Date("2020. április 12.", format="%Y. %B %d.") # magyar hónapnével
#> [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")` parancssal vizsgálhatjuk meg, hogy milyen környezetben dolgozunk. A `sys.setlocale("LC_TIME", "C")` utasítás észak-amerikai beállításokra vált az R, 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 hónapnével
#> [1] "2020-04-12"
as.Date("12 April 2020", format="%d %B %Y")         # angol hónapnével
#> [1] "2020-04-12"
Sys.setlocale("LC_TIME", lct) # magyar helyi beállítás visszatöltése
#> [1] "Hungarian_Hungary.utf8"
```

**5.4.2.1.2. 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"
```

**5.4.2.1.3. Dátum konvertálása** ♦ Ha már van egy *dátum* típusú objektumunk, akkor azt változatos módon jeleníthetjük meg a `format()` függvény segítségével, amely egy-szerű 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"
```

#### 5.4.2.2. 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()` függvényel lehetséges, és ez az általunk dátum-idő típusnak tekintett *POSIXct* objektummal tér vissza:

```
ido <- Sys.time()      # pontos dátum-idő, POSIXct típusú objektum
ido          # idő kírása
#> [1] "2022-07-22 10:32:00 CEST"
typeof(ido)       # idő típusa
#> [1] "double"
class(ido)        # idő típusa
#> [1] "POSIXct" "POSIXt"
unclass(ido)      # idő alapja
#> [1] 1658478720
```

A fentiek alapján úgy tűnhet, hogy a *POSIXct* objektum egész másodperceket tárol csupán, de ez nem így van. Az alapértelmezett megjelenítések módosítva láthatóvá válnak a tört másodpercek is:

```
op <- options(digits.secs = 6, digits = 16)
ido          # POSIXct kiírása változott digits.sec=6 miatt
#> [1] "2022-07-22 10:32:00.4746 CEST"
unclass(ido)  # double kiírása változott digits=6 miatt
#> [1] 1658478720.4746
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()` függvényel 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()` függvényel beállíthatjuk a kívánt időzónát, amely a legtöbb esetben 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] "2022-07-22 08:32:00 UTC"
Sys.setenv(TZ=tz)    # alapértelmezett időzóna visszaállítása
```

A másik lehetőség, hogy már egy létező *POSIXct* objektumon végezzük 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] "2022-07-22 10:32:00 CEST"
ido.utc   # UTC-vel
#> [1] "2022-07-22 08:32:00 UTC"
```

**5.4.2.2.1. Dátum-idő létrehozása karakteres adatból** ♦ Amennyiben karakteres formában rendelkezésre áll egy időpont, akkor minden össze az egyes komponensek jelentését kell elmagyaráznunk az *as.POSIXct()* függvény *format=* argumentumában. Szabványos idő megadása esetén ((ISO 8601<sup>5</sup>)) 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 (??. táblázat) kibővül a ?? táblázatban szereplő időre vonatkozó kódokkal, így ezeket is használhatjuk a *POSIXct* objektum létrehozása során. Teljes listát az *?strptime* súgójában olvashatunk.

| Formátum kód | Jelentés              |
|--------------|-----------------------|
| %H           | óra (00-23)           |
| %I           | óra (01-12)           |
| %M           | perc (00-59) neve     |
| %p           | AM/PM jelzése         |
| %S           | másodperc (00-59)     |
| %Z           | időzóna (csak output) |

<sup>5</sup> [https://en.wikipedia.org/wiki/ISO\\_8601](https://en.wikipedia.org/wiki/ISO_8601)

**5.4.2.2.2. 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"
```

**5.4.2.2.3. Dátum-idő konvertálása** ♦ A `POSIXct` objektum `dátum` típusúvá konvertálható az `as.Date()` függvénnyel, illetve a `format()` függvény segítségével tetszőleges formájú karakteres dátumot/időt nyerhetünk ki az objektumból.

```
ido <- Sys.time() # pontos dátum-idő, helyi beállításnak megfelelően
ido # dátum-idő objektum
#> [1] "2022-07-22 10:32:00 CEST"
as.Date(ido) # dátum objektum
#> [1] "2022-07-22"
format(ido, format="%Y. %m. %d.") # magyar dátum
#> [1] "2022. 07. 22."
format(ido, format="%Y. %B %d. %H.%M.%S") # magyar dátum-idő
#> [1] "2022. július 22. 10.32.00"
format(ido, format="%Y. %b %d. %H:%M:%S") # magyar dátum-idő
#> [1] "2022. júl. 22. 10:32:00"
format(ido, format="%Y. %m. %d. %H:%M:%S %Z") # magyar dátum-idő
#> [1] "2022. 07. 22. 10:32:00 CEST"
```

### 5.4.2.3. Műveletek és az időtartam

A többnyire szöveges formában megjelenő dátumok és dátum-idők R objektummá alkításának 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 `diffftime()` 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 7797 days
diffftime(Sys.Date(), as.Date("2001-03-17")) # ua.
#> Time difference of 7797 days
diffftime(Sys.Date(), as.Date("2001-03-17"), unit="hours") # órákban
```

```
#> Time difference of 187128 hours
as.numeric(difftime(Sys.Date(), as.Date("2001-03-17"), unit="hours")) # számként
#> [1] 187128
```

Ne feledjük, hogy az időtartam is egy típus az R-ben (*difftime* osztály), ahogyan a következő sorokban ez megfigyelhetjük:

```
diffdt <- difftime(Sys.Date(), as.Date("2001-03-17"), unit="hours")
typeof(diffdt) # diffdt típusa
#> [1] "double"
class(diffdt) # diffdt típusa
#> [1] "difftime"
unclass(diffdt) # diffdt alapja
#> [1] 187128
#> attr(,"units")
#> [1] "hours"
```

A *difftime()* működik dátum-idővel is, és természetesen két dátum vagy időpont között a szokásos műveletek is elvégezhetők. A Google Ürlap időbényeg 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 időbényeg a korábbi
#> [1] TRUE
# hány másodperc telt el a két válasz között
as.numeric(difftime(idobelyeg.2, idobelyeg.1, unit="sec"))
#> [1] 3302
```

#### 5.4.2.4. Összefoglalás



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.

#### 5.4.2.5. Feladatok



1. Konvertáljuk dátummá a következő két sztringet: "6November2020", "2013-02-29"! Utóbbi esetben mi lehet a hiba oka?
2. 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!
3. 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 napotot kell még várni az üstökös érkezésére.

#### 5.4.3. Tibble

A Tidyverse R használata során az adatainkat *tibble* típusú objektumban tároljuk. Használatához töltük be a **tidyverse** csomagot.

```
library(tidyverse)
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
#> 7 A         8        7
#> 8 B         9        8
```

A tibble objektumok alaptípusa lista, de az osztálytípusok között megjelennek a tibble-re specifikus osztályok is. A `tbl_df` osztály jelenléte hozzá magával azokat az új tulajdonságokat és lehetőségeket, amit a Tidyverse R központi adatszerkezetévé teszi ezt az objektumtípushoz.

```
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); class(df)
#> [1] "list"
#> [1] "tbl_df"     "tbl"        "data.frame"
```

A tibble és a data frame típusú objektumok között az átjárhatóságot az `as_tibble()` és az `as.data.frame()` függvény biztosítja.

```
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
#> 7 A         8      7
#> 8 B         9      8
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:

```
is_tibble(df); is.data.frame(df)
#> [1] TRUE
#> [1] TRUE
```

Melyek a data frame 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 data frame típusú objektumok. Tibble esetében csak az egy hosszú vektorok ismétlése megengedett. Tehát ez a konstrukció működik:

```
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.

```
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 több oszlop neve alul jelenik meg. Az oszlopnevek alatt az oszlop típusa is megjelenik.

A negyedik eltérés a tibble indexeléséhez kötődik. Az `[` operátor használata során minden esetben tibble típusú objektumot kapunk, nem kaphatunk vektort, azaz nem törtenhet dimenzióvesztés.

```
df[, 2]
#> # A tibble: 8 x 1
#>   pont.1
#>   <int>
#> 1     6
#> 2     7
#> 3     8
#> 4     9
#> 5     6
#> 6     7
#> 7     8
#> 8     9
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]
#> [1] 6
```

#### 5.4.4. 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()
```

A munkaterületről objektumot az `rm()` parancssal távolíthatunk el, például a

```
rm(fib.0)      # fib.0 törlése
ls()
#> [1] "a"          "alap"        "D"           "datum"
#> [5] "df"          "df.long"     "df1"         "df2"
#> [9] "df3"         "diff"        "egyutthato.a" "egyutthato.b"
#> [13] "egyutthato.c" "eset.1"     "eset.2"       "eset.3"
#> [17] "fib.1"       "fib.2"       "fiu"          "foglalkozas"
#> [21] "hazas"       "ido"         "ido.utc"      "idobelyeg.1"
#> [25] "idobelyeg.2" "import_example" "isk.vegz"    "isk.vegz.f"
#> [29] "iteletek"     "lakohely"    "llct"         "magassag"
#> [33] "mat"          "nev"         "nevek"        "obj.double"
#> [37] "obj.integer" "obj.karakteres" "obj.logikai" "op"
#> [41] "peter.bmi"    "peter.magassaga" "peter.sulya"  "pontszamok"
#> [45] "pulzus.atlag" "Pulzus.atlag"   "regi.v.1"    "regi.v.2"
#> [49] "survey"        "szamok"      "tab1"         "tab2"
#> [53] "tab3"          "tab4"        "table.kiir"   "tan.ido"
#> [57] "tipus"         "tomb"        "tz"           "uj.v"
```

```
#> [61] "v.v.d"           "v.v.i"           "v.v.k"           "v.v.l"
#> [65] "van.kocsija"    "vekt"            "x"               "x.d"
#> [69] "x.f"              "x.f.1"            "x.f.2"            "x.f.3"
#> [73] "y"                "z"               "z.uj"
```

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()
```

segítségével történik.

#### **5.4.5. A munkakönyvtár függvényei**

Az R használata során mindenkor van egy kitüntetett, aktuális könyvtárunk, amelyet munkakönyvtárnak nevezünk. A munkakönyvtár célja, hogy az állományok nyitása és menete során, ha nem használunk külön könyvtárhivatalkot, akkor ez lesz az alapértelmezett könyvtár.

A munkakönyvtár az R-ben lekérdezhető ill. 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/peldap könyvtárra. Figyeljük meg, hogy az elérési útban perjelet (/) használtunk.

Megjegyezzük, hogy az RStudio projekt üzemmódú használata során nincs szükség a munkakönyvtár beállítására a `setwd()` parancssal, sőt, kerüljük a használatát. A munkakönyvtárunk a munka során végig maradjon meg az alapértelmezetten beállított könyvtár, maga a projektkönyvtár.

A munkakönyvtár jelentőségét tovább növeli, hogy az R indításakor ebben a könyvtárban 2 állomány létezését figyeli: \* .Rhistory (a visszahívatott 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.

#### 5.4.6. Csomagkezelő függvények

Korábban megbeszéltük, hogy a csomagok adatobjektumokat és függvényeket tartalmaznak. Az ún. egyéb csomagok (számuk kb. 17000) 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()` függvénnel.

```
search()
detach("tools:rstudio")
```

```
## [1] ".GlobalEnv"      "package:stats"    "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"       "package:base"
```

A fenti eredményben a `package` karakterszorozattal kezdődő elemek mutatják, hogy melyek az éppen betöltött csomagok. A listában nem szereplő, de korábban telepített csomagok betöltéséhez használjuk a `library()` vagy `require()` függvényeket.

```
library(MASS)
require(foreign)
search()
```

```
## [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)
```

parancsal kérdezhetők le. Betöltött csomagok esetében használhatjuk az

```
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()` függvényel távolíthatunk el a memóriából:

```
detach(package:foreign)
detach(package:MASS)
```

Ha a használni kívánt csomag még nincs telepítve a számítógépünkre, akkor az `@ref(Csomagok_telepítése)`. 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:

```
install.packages("DescTools")
install.packages("psych")
```

A számítógépünön telepített csomagokról az `installed.packages()` függvény ad tájékoztatást. Amennyiben a

```
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

```
update.packages()
```

parancsot.

#### 5.4.7. Feladatok



1. Írassuk ki a munkaterület objektumait!
2. Hozzunk létre egy `pulzus` nevű objektumot és újra írassuk ki a munkaterület objektumneveit!
3. Távolítsuk el a `pulzus` objektumot a munkaterületről!
4. Határozzuk meg az aktuális munkakönyvtárat!
5. Növeljük meg a betű méretét az RGui, az R Commander és az R Studio alkalmazásokban is!

6. Vizsgáljuk meg, hogy a számítógépünkön van-e telepítve a **DescTools** csomag, ha nincs telepítésük! Derítsük ki, hogy a **DescTools** csomagnak mi a célja? Sorolunk fel három függvényt és adattáblát ebből a csomagból! Távolítsuk el a memóriából a **DescTools** csomagot!
7. Telepítük a számítógépünkre a következő csomagokat: **HSAUR2**, **psych**, **prettyR**, **descr** és **pastecs**!

## 5.5. Haladó nyelvi elemek

### 5.5.1. Objektumok és típusok

Az R-ben használható objektumok név-érték párok, vagyis minden objektumnak van neve és értéke. Objektumok alatt ebben a könyvben az adatobjektumokat értjük, bár már említettük, hogy valójában a függvények is objektumoknak tekinthetők az R-ben, hiszen a függvénynek is van neve, és értéke, az utóbbi pedig utasítások sorozata. Az R-ben minden objektum, például az eddig vizsgált vektorok, 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`, `level` és `class` attribútumok jelentőségét is.

Egy objektum összes attribútuma az `attributes()` függvényel kérdezhetők le. Ha a `names` attribútumra vagyunk kíváncsiak a `names()` függvényt is használhatjuk. Ha létrehozunk egy `x` numerikus vektort, akkor

```
x <- 1:5      # integer vektor
attributes(x)  # x attribútumainak kiírása
#> NULL
names(x)       # x name attribútumának kiírása
#> NULL

x <- numeric(0)
mode(x) <- "list"
class(x)
#> [1] "list"
length(x)
#> [1] 0

attr(x, "length") <- "integer"
```

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 állíttottunk 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 `names` attribútum beállítható a `names()` függvényel 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.

```
x
#> elégtelen  elégséges   közepes       jó      jeles
#>      1        2        3        4        5
x[c("közepes", "jó")]
#> közepes     jó
#>      3        4
```

Rögzítük 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 minden idézőjelet kell használnunk.

```
x[1]
#> elégtelen
```

```
#>           1
x["1"]
#> <NA>
#>   NA
x[c(1,3)]; x[c("0", "2")]
#> eléglesen közepes
#>           1           3
#> <NA> <NA>
#>   NA   NA
```

Egyetlen attribútum lekérdezésére és beállítására az `attr()` függvényt is használhatjuk. Az `attr()` függvényben meg kell adnunk az elérődő attribútum nevét is.

```
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` argumentum

```
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
```

### Amennyiben

```
x <- 1:12                      # integer vektor
attr(x, "dim") <- c(2,6)        # integer mátrix (2x6-os)
dimnames(x) <- 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"
```

### Osztályok

A faktor ennek megfelelően tartalmaz egy `levels` attribútumot, amely a faktor különböző értékeit (szintjeit) sorolja fel. A faktorok `class` attribútumának értéke pedig `factor`.

```
f <- factor(c("a", "b", "a"))
attributes(f)
#> $levels
#> [1] "a" "b"
#>
#> $class
```

```
#> [1] "factor"
levels(f)
#> [1] "a" "b"
class(f)
#> [1] "factor"
class(f) <- NULL
#attributes(f)<- NULL
#f
unclass(f)
#> [1] 1 2 1
#> attr(,"levels")
#> [1] "a" "b"
```

A `class()` függvény az objektum `class` argumentumával tér vissza. Azok az objektumok, amelyek nem rendelkeznek `class` argumentummal, a `class()` visszatérési értéke

- "numeric", ha az objektum *integer* vagy *double* vektor
- "array" és/vagy "matrix", ha az objektum rendelkezik `dim` attributummal
- más esetben a `typeof()` visszatérési értékével.

Az `unclass()` visszatérési értéke az az objektum, amelynek a `class` attribútumát eltávolították.

A korábban tárgyat típusok mindegyike osztály: *Date*, *difftime*, *POSIXct*, *POSIXlt*, *table*.

```
x <- as.Date("2020-03-12")
attributes(x)
#> $class
#> [1] "Date"
class(x)
#> [1] "Date"

x <- Sys.Date()-as.Date("2020-03-12")
x
#> Time difference of 862 days
attributes(x)
#> $class
#> [1] "difftime"
#>
#> $units
#> [1] "days"
class(x)
#> [1] "difftime"

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"

x <- as.POSIXlt(x)
x
#> [1] "2020-12-02 12:00:00 GMT"
attributes(x)
#> $names
#> [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"   "isdst"
#
#> $class
#> [1] "POSIXlt" "POSIXt"
#
#> $tzone
#> [1] "GMT"
class(x)
#> [1] "POSIXlt" "POSIXt"

x <- table(sample(LETTERS[1:3], 100, replace = T))
x
#>
#> A  B  C
#> 31 36 33
attributes(x)
#> $dim
#> [1] 3
#
#> $dimnames
#> $dimnames[[1]]
#> [1] "A" "B" "C"
#
#>
#> $class
#> [1] "table"
class(x)
#> [1] "table"

```

A listaelemek nevét a `x` lis `names` attribútuma tartalmazza, segítségével a többi elemnek

is adhatunk értéket:

```
names(x)
#> [1] "A" "B" "C"
names(x)[c(1,2)] <- c("a", "b")
names(x)
#> [1] "a" "b" "C"
x
#> a b C
#> 31 36 33
```

### 5.5.1.1. Értékek kizárása

A faktor létrehozásánál gondoskodhatunk bizonyos értékek kizárásról, olyan értékekről, amelyeket nem szeretnénk a faktorban felsorolni:

```
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:

```
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>
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 kizárhatsunk az `NA`-n kívül.

Nézzük, hogyan tekint az R az adattáblára.

```
#typeof(df); class(df); length(df)
#is.list(df); is.matrix(df); is.data.frame(df)
```

Az adattáblák alaptípusa `list`, osztálytípusa pedig `data.frame` a hossza pedig az alkotó (oszlop)vektorok/faktorok száma. Az adattáblára tehát tekinthetünk úgy, mint egy listára, melynek elemei az adattábla oszlopai lesznek.

### Feladat

234 (23)4 2(34)



## 6. fejezet

# Beolvasás



### 6.1. Alapvető formátumok



Ebben a fejezetben áttekintjük:

- a táblázatkezelők állományainak beolvasását,
- a tagolt szöveges állományok fogalmát és
- azok beolvasását az *Alap R*-ben.

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 beolvasását táblázatkezelők (pl. Microsoft Excel, LibreOffice Calc) állományaiból és tagolt szöveges állományokból.

#### 6.1.1. Táblázatkezelők

A táblázatkezelők saját állományai (pl. `.xlsx` és `.ods`) kezelhetők a legkényelmesebben az adatelemzés során. Az adatbevitel és a rögzített adatok karbantartása, későbbi mó-

dosítása ebben a formátumban a legegyszerűbb, ráadásul a számítógépes tesztek sokszor ilyen típusú állományokba írják a válaszokat. Ezek az állományok (munkafüzetek) munkalapokból állnak, így akár több adatbázist is tárolhatunk egyetlen munkafüzetben. Egy Excel vagy LibreOffice Calc adatbázis 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 adatbázisok beolvasását 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`, `agatha_christie_m.ods`), amelynek mindenki 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
head(ac.1[1:3], n=3)
#>   megjelenes.eve           cim.magyar           cim.angol
#> 1       1930    Gyilkosság a paplakban The Murder at the Vicarage
#> 2       1942 Holttest a könyvtárszobában The Body in the Library
#> 3       1942      A láthatatlan kéz      The Moving Finger
ac.1 <- import(file = "adat/agatha_christie_m.ods") # LibreOffice Calc
head(ac.1[1:3], n=3)
#>   megjelenes.eve           cim.magyar           cim.angol
#> 1       1930    Gyilkosság a paplakban The Murder at the Vicarage
#> 2       1942 Holttest a könyvtárszobában The Body in the Library
#> 3       1942      A láthatatlan kéz      The Moving Finger
```

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 adatbázis 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()` függvényünk működésére. Nem maga az `import()` végzi a közvetlen beolvasást, hanem okosan kiválasztja a beolvasandó állománykiterjesztése 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 paramétereinek is eltérhet, és esetünkben el is tér. A `file=` argumentum közös, és természetesen a munkalap

sorszámát meg kell adnunk, amely minden esetben a `sheet=2`-vel történik.

```
ac.2 <- import(file = "adat/agatha_christie_m.xlsx", sheet=2)
head(ac.2, n=3)
#> megjelenes.eve           cim.magyar           cim.angol
#> 1         1930      Gyilkosság a paplakban The Murder at the Vicarage
#> 2         1942 Holttest a könyvtárszobában   The Body in the Library
#> 3         1942      A láthatatlan kéz       The Moving Finger
#> szereplo
#> 1 Miss Marple
#> 2 Miss Marple
#> 3 Miss Marple
ac.2 <- import(file = "adat/agatha_christie_m.ods", sheet=2, range="F7:I52")
head(ac.2, n=3)
#> megjelenes.eve           cim.magyar           cim.angol
#> 1         1930      Gyilkosság a paplakban The Murder at the Vicarage
#> 2         1942 Holttest a könyvtárszobában   The Body in the Library
#> 3         1942      A láthatatlan kéz       The Moving Finger
#> szereplo
#> 1 Miss Marple
#> 2 Miss Marple
#> 3 Miss Marple
```

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 a tartomány pontos megadása szükséges a sikeres beolvasásához.

```
ac.3 <- import(file = "adat/agatha_christie_m.xlsx", sheet=3, skip=6)
head(ac.3, n=3)
#> megjelenes.eve           cim.magyar           cim.angol
#> 1         1930      Gyilkosság a paplakban The Murder at the Vicarage
#> 2         1942 Holttest a könyvtárszobában   The Body in the Library
#> 3         1942      A láthatatlan kéz       The Moving Finger
#> szereplo
#> 1 Miss Marple
#> 2 Miss Marple
#> 3 Miss Marple
ac.3 <- import(file = "adat/agatha_christie_m.ods", sheet=3, range="F7:I52")
head(ac.3, n=3)
#> megjelenes.eve           cim.magyar           cim.angol
#> 1         1930      Gyilkosság a paplakban The Murder at the Vicarage
#> 2         1942 Holttest a könyvtárszobában   The Body in the Library
#> 3         1942      A láthatatlan kéz       The Moving Finger
#> szereplo
#> 1 Miss Marple
```

```
#> 2 Miss Marple
#> 3 Miss Marple
```

A 4. munkalapon már rendkívül terhelt az adatbázisunk a környező zavaró celláktól, 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")
head(ac.4, n=3)
#>   megjelenes.eve           cim.magyar           cim.angol
#> 1      1930    Gyilkosság a paplakban The Murder at the Vicarage
#> 2      1942 Holttest a könyvtárszobában The Body in the Library
#> 3      1942     A láthatatlan kéz       The Moving Finger
#>   szereplo
#> 1 Miss Marple
#> 2 Miss Marple
#> 3 Miss Marple
ac.4 <- import(file = "adat/agatha_christie_m.ods", sheet=4, range="F7:I52")
head(ac.4, n=3)
#>   megjelenes.eve           cim.magyar           cim.angol
#> 1      1930    Gyilkosság a paplakban The Murder at the Vicarage
#> 2      1942 Holttest a könyvtárszobában The Body in the Library
#> 3      1942     A láthatatlan kéz       The Moving Finger
#>   szereplo
#> 1 Miss Marple
#> 2 Miss Marple
#> 3 Miss Marple
```

Jegyezzük meg, ha csak tehetjük, adatainkat táblázatkezelő programmal hozzuk létre és annak saját formátumában (xlsx vagy ods) tároljuk.

### 6.1.2. Tagolt szöveges állományok

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 elégendő, 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 zájják. 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 minden 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 adatértékkal, 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 tehát oszlopneveket tartalmaz. Ez nem kötelező, elképzelhető, hogy az első sorban már közvetlenül adatértékek vannak.

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 valóban felfedezhetjük a rendezettséget, a fejlécet 4 oszlopnévvvel, 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 tekinjen megjegyzésnek (a kettős kereszttet) 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 a 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 tanuljuk meg:

- `file=`  
A beolvasandó állomány elérési útja.
- `sep=`  
Az elválasztó 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 a pont a tizedes elválasztó, így a `dec=".,"`-ra van szükségünk.
- `header=`  
Ha van fejléc a szöveges állományban, akkor `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 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.

Végezzük el az `egyetem.csv` tagolt szöveges állomány beolvasását, amelynek első néhány sora a következő:

```
hallgato;Height;neme;lebekves;felkelés;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 állo-

mány, amelynek a tartalmát a következő parancsok egy `d.read` adattáblában helyezik el. Az állományban az adatokat (és az oszlopneveket is) a pontosvessző (;) választja el, a numerikus értékekben pedig tizedesvessző szerepel.

```
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/egyetem.csv"
d.df <- read.table(file = data.file, header=T,
                     sep=";",
                     dec=",",
                     strip.white = T,
                     stringsAsFactors = F,
                     fileEncoding = "latin2")
str(d.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 ...
#> $ felkeltes: num  5.5 8 7.5 8.5 9 8.5 7.5 7.5 7 8.5 ...
#> $ Drink   : chr  "víz" "üdítő" "tej" "víz" ...
```

### 6.1.3. Összefoglalás



A statisztikai munka első lépése az adatbázisok, adatmátrixok beolvasása. Adatankat 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 a `read.table()` függvénnnyel.

### 6.1.4. Feladatok



1. Olvassuk be a [https://onlinestatbook.com/2/case\\_studies/data/leniency.xls](https://onlinestatbook.com/2/case_studies/data/leniency.xls) Excel állományt, állapítsuk meg hány sora és oszlopa van.
2. Olvassuk be a <https://vincentarelbundock.github.io/Rdatasets/csv/DAAG/socsupport.csv> tagolt szöveges állományt, állapítsuk meg hány sora és oszlopa van.

## 6.2. A Tidyverse R és az inline beolvasás



Ebben a fejezetben áttekintjük

- az inline beolvasás eseteit és
- a tagolt szöveges állományok *Tidyverse R* beolvasását.

### 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 minden esetben *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)
d.tbl <- read_delim(file = data.file, col_names=T,
                     delim=";",
                     locale=locale(decimal_mark=",", encoding = "latin2"),
                     trim_ws= T)
str(d.tbl)
#> #> spec_tbl_df [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 parancsállományokban 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] 137.2
# 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"))
d.df <- data.frame(magassag, nem) # data frame létrehozása
d.tbl <- tibble(magassag, nem)      # tibble létrehozása
```

A létrehozott adattáblák is nagyon hasonlítanak egymásra, és felhasználásuk is azonos módon történik:

```
d.df    # data frame
#> magassag nem
#> 1     132 fiú
#> 2     143 lány
#> 3     129 lány
#> 4     145 fiú
d.tbl  # tibble
#> # A tibble: 4 x 2
#>   magassag     nem
#>   <dbl> <fct>
#> 1     132 fiú
#> 2     143 lány
#> 3     129 lány
#> 4     145 fiú
mean(d.df$magassag)           # testmagasság átlaga
#> [1] 137.2
mean(d.tbl$magassag)          # testmagasság átlaga
#> [1] 137.2
table(d.df$nem, useNA = "ifany") # gyakorisági táblázat a nemre
#>
#>   fiú lány
#>   2     2
```

```
table(d.tbl$nem, useNA = "ifany") # gyakorisági táblázat a nemre
#>
#>   fiú lány
#>   2     2
```

Nem kell feltétlenül az adattáblát alkotó oszlopokat külön numerikus vagy faktor oszlopokban előzőleg elkészíteni, ezeket a `data.frame()` vagy `tibble()` argumentumába közvetlenül is beírhatjuk:

```
d.df <- data.frame(
  magassag = c(132, 143, 129, 145),
  nem = factor(c("fiú", "lány", "lány", "fiú"), levels=c("fiú", "lány"))
)

d.tbl <- 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 oszlopnevét vessző választ el egymástól a `tribble()` argumentumában.

```
d.tbl <- tribble(
  ~magassag, ~nem,
  132, "fiú",
  143, "lány",
  129, "lány",
  145, "fiú"
)
d.tbl
#> # 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()` függvénnnyel gondoskodnunk kell.

```
d.tbl$nem <- factor(d.tbl$nem, levels=c("fiú", "lány"))
d.tbl
#> # 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 kódba, ennek megfelelően állítottuk be az elválasztó karaktereket. A `read.table()` esetében körbevettük a `textConnection()` függvényteljesítményét a beillesztett adatokat, erre a *Tidyverse R* `read_delim()`-jénél már nincs szükség. A `nem` faktorról konvertálásáról se felejtkezzünk el.

```
d.df <- read.table(file = textConnection(
  magassag nem
  132 fiú
  143 lány
  129 lány
  145 fiú
), header=T, sep=" ")
d.df$nem <- factor(d.df$nem, levels=c("fiú", "lány"))
d.df
#>   magassag nem
#> 1     132 fiú
#> 2     143 lány
#> 3     129 lány
#> 4     145 fiú
d.tbl <- read_delim(
  magassag nem
  132 fiú
  143 lány
  129 lány
  145 fiú
, col_names=T, delim="")
d.tbl$nem <- factor(d.tbl$nem, levels=c("fiú", "lány"))
d.tbl
#> # A tibble: 4 x 2
#>   magassag nem
#>   <dbl> <fct>
#> 1     132 fiú
```

```
#> 2      143 lány
#> 3      129 lány
#> 4      145 fiú
```

### 6.2.3. Összefoglalás



swwwwww

### 6.2.4. Feladatok



sadadasda

## 6.3. Kiírás és más lehetőségek



Ebben a fejezetben áttekintjük

- a tagolt szöveges állományok kiírását
- objektumok olvasását és írását bináris állományokba,
- más statisztikai programcsomagok adatállományainak olvasását és írását,
- és
- a fix széles mezővel rendelkező állományok olvasását.

### 6.3.1. Tagolt szöveges állomány kiírása

Adattáblák és mátrixok kiírására a `write.table()` függvényt használhatjuk az *Alap R*-ból és a `write_delim()` függvényt a *Tidyverse R*-ból. Mindkét függvény egy-egy függvény-család reprezentánsa, de az *Alap R*-ból elegendő ismerni az említett tagot, a *Tidyverse R*-ból pedig az említetten kívül a `write_csv2()` függvényt. Néhány új argumentummal kell megismерkednünk, 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 mindenkor függvény esetében.

Az első példában a `write.table()` függénnel egy mátrixot és a korábban létrehozott `d.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
write.table(x.mat, "output/adat/x_mat.txt", col.names=F, row.names=F) # kiírása
# adattábla kiírása
write.table(x = d.df, file = "output/adat/df_out.txt", sep = "\t", quote = F,
            dec = ",", row.names = F, col.names = T, fileEncoding = "UTF-8")
```

A Tidyverse 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 = d.tbl, file = "output/adat/tbl_out.txt", delim = "\t", col_names = T)
# pontosvesszővel tagolt szöveges állomány létrehozása
write_csv2(x = d.tbl, file = "output/adat/tbl_out.csv", col_names = T)
```

### 6.3.2. R objektumok írása és olvasása

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 beszélünk, pedig a munka során ezek mentése és beolvasása is érdekes lehet.

Egy objektum értékét eltárolhatjuk szöveges állományban a `dput()` függvényel, és visszaolvashatjuk a `dget()`-tel:

```
library(MASS)
dput(x = survey, file = "output/adat/dput_out.txt") # survey kiírása txt-be
d.df <- dget(file = "output/adat/dput_out.txt")      # survey beolvasása txt-ből
```

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 tudjunk lementeni és visszatölteni egy objektumot.

```
# survey kiírása bináris állományba
saveRDS(object = survey, file = "output/adat/survey.rds")
# survey beolvasása bináris állományból
d.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
d.df <- read_rds(file = "output/adat/survey_2.rds")
```

Egyszerre több objektumok 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 figyelmezhetünk 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. Más típusú adatállományok

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 pozícióban 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 megfelelő 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"
d.df <- import(file = data.file, sep=";", header=T, dec=",")
str(d.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\xeddz" "\xfc\cd\xedt\xf5" "tej" "v\xeddz" ...
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/nepesseg.sav"
d.df <- import(file = data.file)
str(d.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 "Elveszületések 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"
d.tbl <- import(file = data.file, setclass = "tbl")
str(d.tbl)
#> #> 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 esetle-

gesen mi is bővíthatjuk az `export()` argumentumlistáját. A következő példákban pontosvesszővel tagolt, SPSS, SAS, XLSX és RDS adatállományokat hozunk létre:

```
export(x = d.df, file = "output/adat/rio_out.csv", dec=",", sep=";") # CSV
export(x = d.df, file = "output/adat/rio_out.sav")                      # SPSS
export(x = d.df, file = "output/adat/rio_out.sas7bdat")                  # SAS
export(x = d.df, file = "output/adat/rio_out.xlsx")                       # Excel
export(x = d.df, file = "output/adat/rio_out.ods")                        # LibreOffice Calc
export(x = d.df, file = "output/adat/rio_out.RDS")                         # RDS
```

### 6.3.4. Adatok csomagokban

Az adatalemzési munkánk R-ben az adattáblák létrehozásával kezdődik. Az adatokat külső adatállományból többféle módszerrel beolvashatjuk, illetve inline módon is létrehozhatjuk (lásd @ref(#beolvas-alapveto) fejezet). Azonban számos csomag tartalmaz saját adattáblát, amelyeket használhatunk az R megismerése során is. 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
data(package = .packages(all.available = TRUE)) # a telepített csomagokban lévő adattáblák
```

Amennyiben egy adattáblára szükségünk van egy csomagból, akkor a csomag betöltése nélkül is elérhetjük az adattáblát:

```
data(survey, package="MASS")
head(survey)

#>      Sex Wr.Hnd NW.Hnd W.Hnd     Fold Pulse    Clap Exer Smoke Height      M.I
#> 1 Female  18.5  18.0 Right  R on L     92 Left Some Never  173.0 Metric
#> 2   Male   19.5  20.5 Left   R on L    104 Left None Regul  177.8 Imperial
#> 3   Male   18.0  13.3 Right  L on R     87 Neither None Occas    NA    <NA>
#> 4   Male   18.8  18.9 Right  R on L     NA Neither None Never  160.0 Metric
#> 5   Male   20.0  20.0 Right Neither   35 Right Some Never  165.0 Metric
#> 6 Female  18.0  17.7 Right  L on R     64 Right Some Never  172.7 Imperial
#>      Age
#> 1 18.25
#> 2 17.58
#> 3 16.92
#> 4 20.33
#> 5 23.67
#> 6 21.00
```

A szokásos eljárás azonban a csomag betöltése, amely után az adattábla nevét szabádon 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 (tidyverse be van töltve)
help(smiths, package="tidyverse") # a smiths leírása (tidyverse 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 állományban minden sorban azonos, rögzített szélességükkel hivatkozunk rájuk.

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 pozíció a nevet, 12 pozíció a telefonszámot, 3 pozíció az egy tizedesre pontos életkort soroljuk fel.

Ideiglenesen hozzuk létre ezt az állományt magunk is a `cat()` függvénnnyel. A `tempfile()` függvényt használjuk egy a rendszerünkben érvényes ideiglenes állomány nevének meghatározására. A `cat()` függvénnnyel 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",
    " Ági+3630459785921,2",
```

```
" Zoltán+3630459785942,4",
" Bea+3630459785938,6", sep="\n")
```

A beolvasást az *Alap R* `read.fwf()` függvényével végezzük el először. A `width=` paraméterben 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()` függvénnnyel 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.

```
d.df <- read.fwf(file = file, widths=c(7,12,4), header=T, sep=";", dec=",",
                   colClasses=c("character", "character", "double"),
                   fileEncoding = "UTF-8")
d.df
#>      nev      telefon kor
#> 1 Ági +36304597859 21.2
#> 2 Zoltán +36304597859 42.4
#> 3 Bea +36304597859 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)
d.tbl <- read_fwf(file = file, skip=1,
                   col_positions = fwf_widths(widths = c(7, 12, 4),
                                               col_names =
                                               c("nev", "telefon", "kor")),
                   locale=locale(decimal_mark=",", encoding = "UTF-8"),
                   col_types = "ccd")
d.tbl
#> # A tibble: 3 x 3
#>   nev      telefon        kor
#>   <chr> <chr>       <dbl>
#> 1 Ág     i+3630459785 921
#> 2 Zoltá n+3630459785 942
#> 3 Bea    +36304597859 38.6
```

### 6.3.6. Összefoglalás



asdasd

### 6.3.7. Feladatok



1 A `cat()` függvénytel a `dput()`-hoz hasonlóan szöveges állományba írhatjuk egy karakteres, numerikus vagy logikai vektor értékét. Mindkét függvénytel végezzük el a kiírást, és vessük össze a kapott szöveges állományok tartalmát! 1. A Kaggle egyik adatbázisában<sup>1</sup> 4000 videójáték értékelése található. Töltsük le a CSV adatállományt, és nyissuk meg. Keressük meg az R-bloggers<sup>2</sup> oldalon az adatállományhoz kapcsolódó cikket, és próbálunk ki néhány elemző parancsot. A blogger melyik csomag, melyik függvényével végezte a beolvasást? 1. Keressük fel és tanulmányozzuk a Great R packages for data import, wrangling and visualization<sup>3</sup> oldalt! A bevezetésben lefektetett alapelvek közül melyiket erősít meg ez az oldal? 1. Töltsünk le 10 érdekesnek tűnő adatállományt a 19 Places to Find Free Data Sets for Data Science Projects<sup>4</sup> oldalról, és nyissuk meg őket!



## 7. fejezet

# Adatmanipuláció



### 7.1. Adatkezelés az Alap R-ben



Ebben a fejezetben :

Ebben a fejezetben az adattáblák manipulációját tekintjük át, melyek az adatkezelés szempontjából a legfontosabb R objektumok. Mint korábban láttuk, a mátrixhoz hasonlóan sorokat és oszlopokat tartalmaz, illetve a listához hasonlóan elemekből, még hozzá azonos hosszúságú oszlopvektorokból épül fel (??. ábra). Az adattábla kettős eredete jelentősen megkönyíti az ilyen adatok kezelését.

Az adattábla sorai egyedeikre (személyek, tárgyak, dolgok stb.) vonatkozó megfigyelések, az oszlopok pedig a megfigyelt tulajdonságok. A statisztikához közelebbi fogalakkal, 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 tárolá-

### 7.1. táblázat. Információt kérő függvények

| Függvény                      | Leírás              | Példa   |
|-------------------------------|---------------------|---|
| <code>str(object)</code>      | szerkezet kiírása   | <code>str(df)</code>                          |
| <code>dim(x)</code>           | x dimenziói         | <code>dim(df)</code>                          |
| <code>nrow(x)</code>          | x sorainak száma    | <code>nrow(df)</code>                         |
| <code>ncol(x)</code>          | x oszlopainak száma | <code>ncol(df)</code>                         |
| <code>names(x)</code>         | x elemeinek neve    | <code>names(df)</code>                        |
| <code>colnames(x)</code>      | x oszlopnevei       | <code>colnames(df)</code>                     |
| <code>rownames(x)</code>      | x sornevei          | <code>rownames(df)</code>                     |
| <code>head(x,n=6)</code>      | x első sorai        | <code>head(df)</code>                         |
| <code>tail(x,n=6)</code>      | x utolsó sorai      | <code>tail(df)</code>                         |
| <code>View(x)</code>          | x teljes tartalma   | <code>View(df)</code>                         |
| <code>class(x)</code>         | x típusa            | <code>class(df); class(df)\$oszlop</code>     |
| <code>length(x)</code>        | x hossza            | <code>length(df); length(df)\$oszlop</code>   |
| <code>unique(x)</code>        | x különböző értékei | <code>unique(df)\$oszlop</code>               |
| <code>table(...,useNA)</code> | gyakorisági tábla   | <code>table(df)\$oszlop, useNA='ifany'</code> |
| <code>summary(object)</code>  | leíró adatok        | <code>summary(df); summary(df)\$oszlop</code> |

sára a numerikus vektort. Természetesen adattáblában karakteres és logikai vektorok is szerepelhetnek, sőt dátumokat és időpontokat is kezelhetünk az adattáblában.

#### 7.1.1. Információ megtekintése

Az adatbázis beolvasása (@ref(#beolvasas)) után következik az információk begyűjtése a beolvasott adatokról. A legfontosabb információkérő függvényeket a ?? táblázat tartalmazza. Az információ megszerzésének célja az egyszerű tájékozódáson kívül a beolvasás helyességének ellenőrzése: 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.

Az adatalemzési munkánk során a beolvasás előtt már sok ismeretünk összegyűlt az adatbázisról, de most tegyük úgy, mintha egy ismeretlen `flow.xlsx` adatbázist kellene felfedeznünk.

```
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 kapcsolatban él" "Egyedülálló" ...
```

```
#> $ 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"          "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$alkatoi.tev) # az alkatoi.tev változó típusa
#> [1] "character"
unique(flow$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 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
#> 1 2 2 4 22 18 4 4 3 5 4 1 2 1 1 1 1 1 2 1 1 3 3 4 5 3
#> 69
#> 1
```

Az adatbázis tetszőleges részét megjeleníthetjük a konzolban a szokásos indexelés segítségével:

```
d.df[sorindex, oszlopindex]
```

Például 4 sorból az első 5 oszlopot így tekinthetjük meg:

```
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 felőli részben (ezt a View(flow) parancssal is kezdeményezhetjük), de a flow előtti ikonon kattintva megjeleníthetjük az adatbázis str()-ból ismert szerkezetét.

További kényelmi lehetőség, ha a *tibble* típus kényelmes 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   flow.4
#>   <chr>        <dbl> <chr> <chr>          <chr>     <dbl>    <dbl>    <dbl>    <dbl>
#> 1 Igen         26   Férfi  Egyedülálló  Egyetem      2       4       5       5
#> 2 Igen         20   Nő    Egyedülálló  Gimnázi~     4       4       5       5
#> 3 Nem          22   Nő    Élettársi kapcs~ Gimnázi~     4       4       3       4
#> 4 Nem          21   Férfi Egyedülálló  Gimnázi~     3       4       3       4
```

```
#> 5 Igen      21 Nő   Egyedülálló   Gimnázi~   5   5   5   2
#> 6 Igen      25 Nő   Elvált     Egyetem    5   5   5   5
#> 7 Nem       53 Nő   Özvegy     Gimnázi~   5   5   3   3
#> 8 Nem       21 Nő   Egyedülálló   Gimnázi~   4   4   4   4
#> 9 Nem       22 Nő   Élettársi kapcs~ Gimnázi~   5   4   4   5
#> 10 Nem      21 Nő   Egyedülálló   Gimnázi~   5   5   5   5
#> # ... with 90 more rows, and 16 more variables: 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. Oszlopok kezelése

### 7.1.2.1. 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 meggysíthatják a további munkát, és fordítva, a kevésbe beszédes, következetlen, a túl rövid vagy túl hosszú változónevek akadályozhatják a sikeres adatelemzést. A változónevek minden legyenek beszédesek, csak az angol ábécé kisbetűit és számjegyeket használjunk, több részből álló neveket aláhúzással (.) esetleg ponttal (.) tagoljuk (R kódolási stílus<sup>1</sup>).

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.

```
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"
names(bandak)[2] <- "rajongo"      # 2. oszlop nevének átírása
names(bandak)[3:5] <- c("alakulas","felbomlas","orszag") # 3-5. oszlopok átnevezése
```

<sup>1</sup> <https://style.tidyverse.org/syntax.html>

```
names(bandak)[names(bandak)=="banda"] <- "nev" # a "banda" nevű oszlop átnevezése
bandak
#>      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
```

### 7.1.2.2. Oszlop indexelése

Az Alap R-ben az oszlopok indexelése a `[` vagy `[[` operátor segítségével történhet. Mivel az adattáblák örököltek 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 használjuk. 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 minden napos a statisztikai munka során, így ezeket az indexelési formákat ismernünk kell.

```
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.   :17.0   Min.   :1.00
#> 1st Qu.:21.0   1st Qu.:3.00
#> Median :23.0   Median :4.00
#> Mean   :29.3   Mean   :4.01
#> 3rd Qu.:33.0   3rd Qu.:5.00
#> Max.   :69.0   Max.   :5.00
# "flow" szót tartalmazó oszlopok száma
ncol(flow[grep(pattern = "flow", 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 akádályozhatjuk 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
```

### 7.1.2.3. 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 már is új oszlopsorrendet határoztunk meg.

```
# a korábbi 1. oszlop (alkotoi.tev) átkerül a 3. oszlopra
flow <- flow[, c(2, 3, 1, 4:25)]
flow[1:2, 1:3]
#> kor    nem alkotoi.tev
#> 1 26 Férfi      Igen
#> 2 20   Nő       Igen
```

Természetesen oszlopsorszámok helyett változóneveket is használhatunk.

```
bandak
#>      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
```

#### 7.1.2.4. Oszlopok létrehozása és törlése

Láttuk korábban, hogy a `cbind()` segítségével oszlopokat adhatunk a meglévő adatbázisunkhoz. Például a meglévő bandák 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)
```

Egyetlen oszlop beszúrására is van lehetőségünk, és hasonlóan törölhetünk egyetlen oszlopot is:

```
adattábla$új.oszlopnév # új oszlop beszúrása a dataframe végére
adattábla$oszlopnév <- NULL # oszlop törlése
```

Szűrjuk 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
```

#### 7.1.2.5. Típuskonverzió

Az oszlopok nevének és sorrendjének optimális beállítása után meg kell vizsgálnunk, hogy az oszlopaink típusa megfelel-e az általa reprezentált statisztikai változók méreti skálájának. Nem léphetünk tovább az elemzés felé, amíg ez az összefüggés nem teljesül.

|                        |         |
|------------------------|---------|
| Változó mérési skálája | R típus |
| nominális              | faktor  |

| Változó mérési skálája | R típus                               |                             |
|------------------------|---------------------------------------|-----------------------------|
| ordinális              | faktor (rendezett)                    |                             |
| intervallum            | numerikus vektor                      |                             |
| arány                  | numerikus vektor                      |                             |
| Cél mérési skála       | Típuskonverzió                        | R függvény                  |
| - nominális            | - numerikusból faktor                 | - factor(x)                 |
| - ordinális            | - karakteresből faktor                | - ordered(x)                |
|                        | - numerikusból faktor<br>(rendezett)  |                             |
|                        | - karakteresből faktor<br>(rendezett) |                             |
| - intervallum          | - karakteresből numerikus             | - as.numeric(x)             |
| - arány                | - faktorból numerikus                 | -                           |
|                        |                                       | as.numeric(as.character(x)) |

A típuskonverzió két leggyakoribb esetével foglalkozunk itt. Az egyik faktorrá konver-tálás teszi a típuskonverziók legnagyobb részét. Ezt a `factor()` függvénytel végezzük el. A kiinduló változónk lehet numerikus vagy karakteres.

```
factor(c(1, 1, 2, 1, 1, 1, 2))
#> [1] 1 1 2 1 1 1 2
#> Levels: 1 2
factor(c("Dohányzik", "Dohányzik", "Nem dohányzik"))
#> [1] Dohányzik Dohányzik Nem dohányzik
#> Levels: Dohányzik Nem dohányzik
```

Mindkét esetben előfordulhat, hogy a kiinduló változóban lévő egyedi numerikus értékek

A numerikus típuskonverzió kevésbé gyakori, de előfordulhat.

Például

```
kor <- c(44, 39, "55 év", 38)
kor[3] <- "55"
as.numeric(kor)
#> [1] 44 39 55 38
```

#### 7.1.2.6. Transzformáció

Számos esetben szükség lehet az adattábla oszlopaiban lévő értéke átalakítására (transzformálására). Az értékeket vagy helyben (ugyanabban az oszlopban) változtat-juk meg, vagy új oszlopként szúrjuk be az adattáblába. Adatok transzformálásához

tekintsük a women adattáblát, amely a weight változójában font-ban mért értékeket tartalmaz. Ezt alakítsuk át kg-ban mért adatokká egy új oszloppal:

```
data(women); women
#>   height weight
#> 1     58    115
#> 2     59    117
#> 3     60    120
#> 4     61    123
#> 5     62    126
#> 6     63    129
#> 7     64    132
#> 8     65    135
#> 9     66    139
#> 10    67    142
#> 11    68    146
#> 12    69    150
#> 13    70    154
#> 14    71    159
#> 15    72    164
```

```
women$suly<-round(women$weight*0.45)
women
#>   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
#> 7     64    132   59
#> 8     65    135   61
#> 9     66    139   63
#> 10    67    142   64
#> 11    68    146   66
#> 12    69    150   68
#> 13    70    154   69
#> 14    71    159   72
#> 15    72    164   74
```

Ugyanezt az eredményt a transform() függvény segítségével is elérhetjük, ahol a subset()-hez hasonlóan némileg egyszerűbben hivatkozhatunk az adattábla változóira. Most alakítsuk át height változót inch-ről cm-re.

```
transform(women, magassag=round(height*2.45))
#>   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
#> 7     64     132   59    157
#> 8     65     135   61    159
#> 9     66     139   63    162
#> 10    67     142   64    164
#> 11    68     146   66    167
#> 12    69     150   68    169
#> 13    70     154   69    172
#> 14    71     159   72    174
#> 15    72     164   74    176
```

Amennyiben a fenti példákban nem új változónevek az átalakítás célpontjai, hanem már létező oszlopok, akkor helyben végezzük a transzformációt:

```
transform(women, height=height-10)
#>   height weight suly
#> 1     48     115   52
#> 2     49     117   53
#> 3     50     120   54
#> 4     51     123   55
#> 5     52     126   57
#> 6     53     129   58
#> 7     54     132   59
#> 8     55     135   61
#> 9     56     139   63
#> 10    57     142   64
#> 11    58     146   66
#> 12    59     150   68
#> 13    60     154   69
#> 14    61     159   72
#> 15    62     164   74
```

A változók átalakításának másik gyakori esete, amikor az eredetileg folytonos változót kategórikus változóvá alakítjuk. A `cut()` függvény segítségével numerikus vektorból faktort állíthatunk elő.

```
cut(1:10,3)
#> [1] (0.991,4] (0.991,4] (0.991,4] (0.991,4] (4,7]      (4,7]      (4,7]
#> [8] (7,10]     (7,10]     (7,10]
#> Levels: (0.991,4] (4,7] (7,10]
```

A fenti példában a 10 elemű bemenő vektortból 3 szintű faktort hoztunk létre. Ha az intervallumok határát magunk szeretnénk megadni, akkor a második (breaks) argumentumban egy vektort kell megadnunk:

```
cut(1:10,breaks=c(0,2,10))
#> [1] (0,2]  (0,2]  (2,10] (2,10] (2,10] (2,10] (2,10] (2,10] (2,10]
#> Levels: (0,2] (2,10]
```

A létrejövő faktor szintjei az intervallumok leírásából állnak, természetesen ezeket megváltoztathatjuk, csak a `labels`= paramétert kell használnunk:

```
cut(1:10,breaks=c(0,2,7,10),label=c("gyenge","közepes","erős"))
#> [1] gyenge gyenge közepes közepes közepes közepes erős     erős
#> [10] erős
#> Levels: gyenge közepes erős
```

Adattáblák esetében a `cut()` függvény használatára láthatunk egy példát:

```
transform(women,height=cut(height,breaks=c(0,60,70,100),labels=c("alacsony","közepes","magas")))
#> height weight suly
#> 1 alacsony    115   52
#> 2 alacsony    117   53
#> 3 alacsony    120   54
#> 4 közepes     123   55
#> 5 közepes     126   57
#> 6 közepes     129   58
#> 7 közepes     132   59
#> 8 közepes     135   61
#> 9 közepes     139   63
#> 10 közepes    142   64
#> 11 közepes    146   66
#> 12 közepes    150   68
#> 13 közepes    154   69
#> 14 magas      159   72
#> 15 magas      164   74
```

`car csomag recode()` függvénye

### 7.1.2.7. Faktor változó összefoglalás

### 7.1.2.8. Numerikus változó összefoglalás

### 7.1.3. Sorok kezelése

Sokszor előfordul, hogy egy adattábla valamely változójának értékeivel szeretnénk a sorokat elnevezni, illetve fordítva, az adattábla sorneveit oszlopvektorban szeretnénk látni. A **fivethirtyeight** csomag `unisex_names` adattáblája adattábla állományból történő beolvasása során a `read.table()` függvényben a "row.names=n" argumentum megadásával a szöveges állomány `n.` oszlopából nyerjük a sorok neveit. Az `mtcars` adattábla sorneveit a következő parancs segítségével vihetjük be változóba:

```
data(unisex_names, package = "fivethirtyeight")
head(unisex_names)

#> # A tibble: 6 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
#> 6 Jaime  109870.    0.562      0.438  0.124
```

```
rownames(unisex_names) <- unisex_names$name
head(unisex_names)

#> # A tibble: 6 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
#> 6 Jaime  109870.    0.562      0.438  0.124
```

```
mtcars2<-data.frame(name=rownames(mtcars),mtcars, row.names=1:32)
mtcars2[1:10,]

#>          name  mpg cyl disp  hp drat    wt  qsec vs am gear carb
#> 1 Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
#> 2 Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
#> 3 Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
#> 4 Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
#> 5 Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
```

```
#> 6      Valiant 18.1   6 225.0 105 2.76 3.460 20.22 1 0 3 1
#> 7      Duster 360 14.3   8 360.0 245 3.21 3.570 15.84 0 0 3 4
#> 8      Merc 240D 24.4   4 146.7 62 3.69 3.190 20.00 1 0 4 2
#> 9      Merc 230 22.8   4 140.8 95 3.92 3.150 22.90 1 0 4 2
#> 10     Merc 280 19.2   6 167.6 123 3.92 3.440 18.30 1 0 4 4
```

A fordított irányhoz a következő parancsot kell használnunk:

```
mtcars3<-mtcars2          # új adattábla
rownames(mtcars3)<-mtcars3$name # sornevek meghatározása
mtcars3<-mtcars3[2:11]       # a felesleges első oszlop törlése
mtcars3[1:10,]
#>           mpg cyl disp hp drat    wt  qsec vs am gear
#> Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46 0 1 4
#> Mazda RX4 Wag    21.0   6 160.0 110 3.90 2.875 17.02 0 1 4
#> Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61 1 1 4
#> Hornet 4 Drive   21.4   6 258.0 110 3.08 3.215 19.44 1 0 3
#> Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02 0 0 3
#> Valiant          18.1   6 225.0 105 2.76 3.460 20.22 1 0 3
#> Duster 360        14.3   8 360.0 245 3.21 3.570 15.84 0 0 3
#> Merc 240D         24.4   4 146.7 62 3.69 3.190 20.00 1 0 4
#> Merc 230          22.8   4 140.8 95 3.92 3.150 22.90 1 0 4
#> Merc 280          19.2   6 167.6 123 3.92 3.440 18.30 1 0 4
```

### 7.1.3.1. Rendezés

A vektorok rendezésénél már megismertük az `order()` függvényt (3.1.8. fejezet), amelyet adattáblák rendezésére is használhatunk. Az `mtcars` adattábla sorait a fogyasztási adatok (`mpg` változó) alapján növekvő sorrendbe rendezhetjük, ha a sorok indexelésére az `order()` függvény visszatérési értékét használjuk:

```
order(mtcars$mpg)
#> [1] 15 16 24 7 17 31 14 23 22 29 12 13 11 6 5 10 25 30 1 2 4 32 21 3 9
#> [26] 8 27 26 19 28 18 20
```

A fenti indexeket a sorkoordináta helyére írva, megkapjuk a rendezett adattáblát (helytakarékosából az első 10 sorát írjuk ki):

```
mtcars[order(mtcars$mpg)[1:10],]
#>           mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98 0 0 3 4
#> Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82 0 0 3 4
#> Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41 0 0 3 4
```

```
#> Duster 360      14.3   8 360.0 245 3.21 3.570 15.84 0 0 3 4
#> Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42 0 0 3 4
#> Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.60 0 1 5 8
#> Merc 450SLC     15.2   8 275.8 180 3.07 3.780 18.00 0 0 3 3
#> AMC Javelin      15.2   8 304.0 150 3.15 3.435 17.30 0 0 3 2
#> Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87 0 0 3 2
#> Ford Pantera L   15.8   8 351.0 264 4.22 3.170 14.50 0 1 5 4
```

Rendezési szempontnak a sorneveket is használhatjuk:

```
mtcars[order(rownames(mtcars))[1:10],]
#>                      mpg cyl disp hp drat    wt  qsec vs am gear carb
#> AMC Javelin      15.2   8 304.0 150 3.15 3.435 17.30 0 0 3 2
#> Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98 0 0 3 4
#> Camaro Z28        13.3   8 350.0 245 3.73 3.840 15.41 0 0 3 4
#> Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42 0 0 3 4
#> Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61 1 1 4 1
#> Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87 0 0 3 2
#> Duster 360         14.3   8 360.0 245 3.21 3.570 15.84 0 0 3 4
#> Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50 0 1 5 6
#> Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47 1 1 4 1
#> Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90 1 1 4 1
```

Rendezésnél egynél több változót is figyelembe vehetünk, ekkor az `order()` függvényben több változónévet kell felsorolnunk vesszővel elválasztva:

```
mtcars[order(mtcars$mpg, mtcars$disp)[1:10],c("mpg", "disp")]
#>                      mpg   disp
#> Lincoln Continental 10.4 460.0
#> Cadillac Fleetwood  10.4 472.0
#> Camaro Z28           13.3 350.0
#> Duster 360            14.3 360.0
#> Chrysler Imperial    14.7 440.0
#> Maserati Bora        15.0 301.0
#> Merc 450SLC          15.2 275.8
#> AMC Javelin          15.2 304.0
#> Dodge Challenger     15.5 318.0
#> Ford Pantera L       15.8 351.0
```

Csökkenő sorrendű rendezéshez használhatjuk az `order()` függvény "decreasing=TRUE" argumentumát, vagy a `rev()` függvényt. Több rendezési szempont esetén ha keverni szeretnénk a rendezési irányokat, akkor numerikus oszlopvektorok előtt a mínusz (-) jellel fordíthatjuk meg a rendezés irányát csökkenőre.

```
mtcars[order(mtcars$mpg,-mtcars$disp)[1:10],c("mpg","disp")]
#>          mpg   disp
#> Cadillac Fleetwood 10.4 472.0
#> Lincoln Continental 10.4 460.0
#> Camaro Z28         13.3 350.0
#> Duster 360         14.3 360.0
#> Chrysler Imperial  14.7 440.0
#> Maserati Bora      15.0 301.0
#> AMC Javelin        15.2 304.0
#> Merc 450SLC        15.2 275.8
#> Dodge Challenger   15.5 318.0
#> Ford Pantera L     15.8 351.0
```

#### 7.1.4. Adattábla szűrése

Sokszor előfordul, hogy az adattábla sorait egy vagy több változó (oszlop) értéke szerint szeretnénk leválogatni. Az adattábla indexelése során a sorkoordináta helyén logikai kifejezést szerepeltetünk. Ha például le szeretnénk kérdezni, azokat a sorokat, amelyekben a fogyásztás értéke kisebb mint 15 mérföld/gallon, akkor a következő logikai kifejezést használhatjuk:

```
mtcars$mpg<15
#> [1] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
#> [13] FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE TRUE
#> [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

A fenti logikai vektorban pontosan azokban a pozíciókban szerepel TRUE érték, amelyik sorban fogyásztás értéke kisebb mint 15 mérföld/gallon. Ha ezt szerepeltetjük a sorkoordináta helyén, a kívánt sorokhoz jutunk:

```
mtcars[mtcars$mpg<15,]
#>          mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Duster 360     14.3   8 360 245 3.21 3.570 15.84  0  0    3    4
#> Cadillac Fleetwood 10.4   8 472 205 2.93 5.250 17.98  0  0    3    4
#> Lincoln Continental 10.4   8 460 215 3.00 5.424 17.82  0  0    3    4
#> Chrysler Imperial  14.7   8 440 230 3.23 5.345 17.42  0  0    3    4
#> Camaro Z28      13.3   8 350 245 3.73 3.840 15.41  0  0    3    4
```

Több váltózón alapuló feltétel megadásához összetett logikai kifejezést kell írnunk:

```
mtcars[mtcars$mpg<15 & mtcars$disp>400,]
#>          mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Cadillac Fleetwood 10.4   8 472 205 2.93 5.250 17.98  0  0    3    4
```

```
#> Lincoln Continental 10.4   8   460 215 3.00 5.424 17.82  0   0     3   4
#> Chrysler Imperial    14.7   8   440 230 3.23 5.345 17.42  0   0     3   4
```

Adattáblák szűrését egyszerűsíti a subset() függvény, amely az első paraméterében egy adattáblát, második paraméterében pedig a szűrést jelentő logikai kifejezést várja. A fenti szűrés subset() függvény használatával:

```
subset(mtcars, mpg<15 & disp>400)
#>                      mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Cadillac Fleetwood 10.4   8   472 205 2.93 5.250 17.98  0   0     3   4
#> Lincoln Continental 10.4   8   460 215 3.00 5.424 17.82  0   0     3   4
#> Chrysler Imperial    14.7   8   440 230 3.23 5.345 17.42  0   0     3   4
```

A subset() függvény egy select argumentumot is tartalmazhat, melynek segítségével a szűrés eredményében megjelenő oszlopokat határozhatjuk meg:

```
subset(mtcars, mpg<15 & disp>400, select=c("mpg","disp"))
#>                      mpg disp
#> Cadillac Fleetwood 10.4  472
#> Lincoln Continental 10.4  460
#> Chrysler Imperial    14.7  440
```

### 7.1.5. Hiányzó értékeket tatalmazó sorok eltávolítása

Az NA értéket is tartalmazó adattáblánkból az na.omit() függvény használatával távolíthatjuk el azokat a sorokat, amelyekben a hiányzó érték előfordul.

```
data(mtcars)
mtcars[c(2,5,7),1]<-NA
mtcars[1:10,]
#>                      mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Mazda RX4        21.0   6 160.0 110 3.90 2.620 16.46  0   1     4   4
#> Mazda RX4 Wag    NA    6 160.0 110 3.90 2.875 17.02  0   1     4   4
#> Datsun 710       22.8   4 108.0  93 3.85 2.320 18.61  1   1     4   1
#> Hornet 4 Drive   21.4   6 258.0 110 3.08 3.215 19.44  1   0     3   1
#> Hornet Sportabout NA    8 360.0 175 3.15 3.440 17.02  0   0     3   2
#> Valiant          18.1   6 225.0 105 2.76 3.460 20.22  1   0     3   1
#> Duster 360        NA   8 360.0 245 3.21 3.570 15.84  0   0     3   4
#> Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1   0     4   2
#> Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1   0     4   2
#> Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1   0     4   4
```

```
na.omit(mtcars)[1:10,]
#>          mpg cyl disp hp drat    wt  qsec vs am gear carb
#> Mazda RX4   21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
#> Datsun 710  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
#> Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
#> Valiant     18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
#> Merc 240D   24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
#> Merc 230    22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
#> Merc 280    19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
#> Merc 280C   17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
#> Merc 450SE   16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
#> Merc 450SL   17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
```

### 7.1.6. Adattáblák indexelése

Az adattáblák indexelésére a mátrixok és listák indexelési eljárásait is használhatjuk. A 3.5. fejezetben már áttekintettük az Alap R lehetőségeit.

```
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/metacritic_games.csv"
library(tidyverse)
d.tbl <- read_delim(file = data.file, delim = ",")
d.tbl
#> # A tibble: 5,699 x 15
#>   game           platform developer genre number_players rating release_date
#>   <chr>          <chr>      <chr>    <chr>        <chr>       <chr>
#> 1 Portal 2       PC         Valve  So~ Acti~ <NA>        E10+     Apr 18, 2011
#> 2 The Elder Scroll~ PC       Bethesda~ Role~ No Online Mul~ M      Nov 10, 2011
#> 3 The Legend of Z~ 3DS      GREZZO   Misc~ No Online Mul~ E10+     Jun 19, 2011
#> 4 Batman: Arkham C~ PC       Rockstea~ Acti~ <NA>        T      Nov 21, 2011
#> 5 Super Mario 3D L~ 3DS      Nintendo Acti~ No Online Mul~ E      Nov 13, 2011
#> 6 Deus Ex: Human R~ PC       Nixxes S~ Acti~ No Online Mul~ M      Aug 23, 2011
#> 7 Pushmo            3DS      Intellig~ Misc~ 1 Player      E      Dec  8, 2011
#> 8 Total War: Shogu~ PC       Creative~ Stra~ Online Multip~ T      Mar 15, 2011
#> 9 FIFA Soccer 12     PC       Electron~ Spor~ <NA>        E      Sep 27, 2011
#> 10 Battlefield 3      PC      EA DICE   Acti~ <NA>        M     Oct 25, 2011
#> # ... with 5,689 more rows, and 8 more variables: positive_critics <dbl>,
#> #   neutral_critics <dbl>, negative_critics <dbl>, positive_users <dbl>,
#> #   neutral_users <dbl>, negative_users <dbl>, metascore <dbl>,
#> #   user_score <dbl>
d.df <- as.data.frame(d.tbl)
head(d.df)
#>           game platform      developer
#> 1      Portal 2      PC      Valve Software
#> 2 The Elder Scrolls V: Skyrim      PC Bethesda Game Studios
```

```
#> 3 The Legend of Zelda: Ocarina of Time 3D      3DS          GREZZO
#> 4                  Batman: Arkham City      PC  Rocksteady Studios
#> 5                  Super Mario 3D Land      3DS          Nintendo
#> 6      Deus Ex: Human Revolution      PC      Nixxes Software
#>   genre      number_players rating release_date positive_critics
#> 1     Action           <NA>  E10+ Apr 18, 2011      51
#> 2  Role-Playing No Online Multiplayer      M Nov 10, 2011      32
#> 3  Miscellaneous No Online Multiplayer  E10+ Jun 19, 2011      84
#> 4 Action Adventure           <NA>  T Nov 21, 2011      27
#> 5     Action No Online Multiplayer      E Nov 13, 2011      81
#> 6     Action No Online Multiplayer      M Aug 23, 2011      52
#>   neutral_critics negative_critics positive_users neutral_users negative_users
#> 1         1             0        1700       107        19
#> 2         0             0        1616       322       451
#> 3         1             0        283        20         5
#> 4         0             0        240        34        27
#> 5         1             0        251        39        11
#> 6         0             0        520       112        78
#>   metascore user_score
#> 1     95        90
#> 2     94        82
#> 3     94        90
#> 4     91        87
#> 5     90        84
#> 6     90        85
```

### 7.1.7. Válogatás az sorokból

A sorok leválogatatása nagyon fontos eszköz a kezünben ahhoz, hogy az elemzésünket a kívánt mintaelemekkel folytathassuk.

#### 7.1.7.1. Sorok indexelése és a szűrés Alap R-ben

A sorok indexelésére a `[` operátort használhatjuk, a vessző előtti részbe, a sorkoordinátákba írhatunk neumerikus vektort, pozitív és negatív értékekkel, logikai vektort és karakteres vektort, bár ez utóbbi használta nagyon ritkkán fordul elő.

A példákhoz a **fivethirtyeight** csomag `unisex_names` adattáblájából indulunk ki, amely amerikában használt uniszex neveket tartalmaz.

```
data("unisex_names", package = "fivethirtyeight")
unisex_names
#> # A tibble: 919 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
#> 6 Jaime    109870.    0.562    0.438  0.124
#> 7 Peyton   94896.     0.434    0.566  0.133
#> 8 Kerry    88964.     0.484    0.516  0.0321
#> 9 Jody     80401.     0.352    0.648  0.296
#> 10 Kendall 79211.    0.372    0.628  0.255
#> # ... with 909 more rows
```

A sorkoordinátákba írjuk a sorok válogatását biztosító lehetőségeket:

```
unisex_names[2, ]
#> # A tibble: 1 x 5
#>   name    total male_share female_share   gap
#>   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
#> 1 Riley    154861.    0.508    0.492  0.0153
unisex_names[1:3, ]
#> # A tibble: 3 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
unisex_names[-(1:3), ]
#> # A tibble: 916 x 5
#>   name    total male_share female_share   gap
#>   <chr>     <dbl>     <dbl>     <dbl>     <dbl>
#> 1 Jackie   132929.    0.421    0.579  0.158
#> 2 Avery    121797.    0.335    0.665  0.330
#> 3 Jaime    109870.    0.562    0.438  0.124
#> 4 Peyton   94896.     0.434    0.566  0.133
#> 5 Kerry    88964.     0.484    0.516  0.0321
#> 6 Jody     80401.     0.352    0.648  0.296
#> 7 Kendall  79211.    0.372    0.628  0.255
#> 8 Payton   64152.     0.334    0.666  0.331
#> 9 Skyler   53486.     0.646    0.354  0.292
#> 10 Frankie  51288.    0.624    0.376  0.247
#> # ... with 906 more rows
```

A `head()` és `tail()` függvény is a sorokból válogat, az első 6, illetve utolsó 3 sort így kap-hatjuk meg:

```
head(unisex_names, n = 6)
#> # A tibble: 6 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
#> 6 Jaime   109870.    0.562     0.438  0.124
tail(unisex_names, n = 3)
#> # A tibble: 3 x 5
#>   name    total male_share female_share   gap
#>   <chr>   <dbl>     <dbl>      <dbl>   <dbl>
#> 1 Gwin    101.      0.562     0.438  0.124
#> 2 Yacine  100.      0.545     0.455  0.0892
#> 3 Aeon    100.      0.465     0.535  0.0703
```

Bizonyos esetekben szükség lehet az adattábla soraiból véletlen módon néhányat kiválasztani. Ekkor a `sample()` függvényt használjuk:

```
unisex_names[sample(x = 1:nrow(unisex_names), size = 10), ]
#> # A tibble: 10 x 5
#>   name    total male_share female_share   gap
#>   <chr>   <dbl>     <dbl>      <dbl>   <dbl>
#> 1 Merion  127.      0.471     0.529  0.0587
#> 2 Deondrea 154.     0.435     0.565  0.131
#> 3 Avry    731.      0.496     0.504  0.00762
#> 4 Taygen   274.     0.445     0.555  0.109
#> 5 Lin     1435.     0.404     0.596  0.193
#> 6 Shamari 1601.     0.420     0.580  0.159
#> 7 Arlis   1462.     0.663     0.337  0.327
#> 8 Vernis  173.      0.666     0.334  0.332
#> 9 Maysen   689.      0.419     0.581  0.162
#> 10 Evann   634.      0.546     0.454  0.0918
```

A `sample()` függvény alapértelmezés szerint visszatevés nélküli választ véletlen értékeket az első paraméterből (esetünkben 919 számból 10-et), de ha a `replace=TRUE` argumentumot használjuk, akkor visszatevéssel fog választani:

```
d.tbl <- unisex_names[sample(x = 11:14, size = 5, replace = T), ]
d.tbl
#> # A tibble: 5 x 5
#>   name    total male_share female_share   gap
```

```
#>   <chr>    <dbl>    <dbl>    <dbl> <dbl>
#> 1 Pat      44782.    0.369    0.631 0.262
#> 2 Payton   64152.    0.334    0.666 0.331
#> 3 Frankie  51288.    0.624    0.376 0.247
#> 4 Frankie  51288.    0.624    0.376 0.247
#> 5 Frankie  51288.    0.624    0.376 0.247
```

A `sample()` függvény fenti paramétere zére mellett biztosan előfordul sorismétlés az új `d.tbl` adattáblában. Az sorismétléseket tartalmazó adattáblák kezelésére a `duplicated()` és a `unique()` függvényeket használhatjuk. A sorismétlések felderítésére a `duplicated()` függvényt használjuk:

```
duplicated(d.tbl)
#> [1] FALSE FALSE FALSE  TRUE  TRUE
d.tbl[duplicated(d.tbl), ]
#> # A tibble: 2 x 5
#>   name    total male_share female_share   gap
#>   <chr>    <dbl>    <dbl>    <dbl> <dbl>
#> 1 Frankie  51288.    0.624    0.376 0.247
#> 2 Frankie  51288.    0.624    0.376 0.247
```

A sorismétlések eltávolítására a `unique()` függvényt használhatjuk:

```
unique(d.tbl)
#> # A tibble: 3 x 5
#>   name    total male_share female_share   gap
#>   <chr>    <dbl>    <dbl>    <dbl> <dbl>
#> 1 Pat      44782.    0.369    0.631 0.262
#> 2 Payton   64152.    0.334    0.666 0.331
#> 3 Frankie  51288.    0.624    0.376 0.247
```

Ha a sorkoordinátákba logikai vektort írunk, akkor szűrést valósítunk meg:

```
unisex_names[unisex_names$gap < 0.003, ]
#> # A tibble: 10 x 5
#>   name    total male_share female_share     gap
#>   <chr>    <dbl>    <dbl>    <dbl> <dbl>
#> 1 Kris     24956.    0.499    0.501 0.00275
#> 2 Camdyn   3275.     0.500    0.500 0.000154
#> 3 Christian 2685.    0.501    0.499 0.00227
#> 4 Kam      729.      0.501    0.499 0.00235
#> 5 Jonel    613.      0.499    0.501 0.00237
#> 6 Kodee    533.      0.500    0.500 0.000316
#> 7 Callaway 292.      0.500    0.500 0.000455
```

```
#> 8 Bless      280.      0.500      0.500 0.0000717
#> 9 Nike       206.      0.500      0.500 0.000782
#> 10 Tkai      143.      0.500      0.500 0.000570
```

## 7.2. Adatkezelés Tidyverse R-ben



Ebben a fejezetben:

### 7.2.1. A %>% operátor

Egy probléma megoldása több lépésben történik. A survey adatbázisból keressük ki a lányok testmagasságának az át gyakran függvényeket kell egymásba ágyaznunk. 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)      # vektor előállítása
out <- log(x)        # természetes alapú logaritmus kiszámolása
out <- sum(out)      # átlag kiszámolása
out                         # eredmény kiírása
#> [1] 1.792
```

A részeredmények tárolására az `out` objektumot használtuk fel. A köztes eredmények tárolása azonos vagy eltérő nevek alatt szokásos gyakorlat a hagyományos R használata során.

A fenti lehetőség mellett választhatjuk a függvények egymásba ágyazását is:

```
x <- c(1, 2, 3)      # vektor előállítása
sum(log(x))           # eredmény kiírása
#> [1] 1.792
```

A fenti példa jóval tömörebb és nem szükséges köztes objektumok létrehozása, a hagyományos R-ben nagyon gyakori összetett függvényhívásokkal (rész)problémák megoldása.

A **magrittr** csomag kínál egy harmadik lehetőséget, a pipe operátor (`%>%`) használatával. Az operátor egyben a Tidyverse R része is.

```
library(tidyverse)
x <- c(1, 2, 3)      # vektor előállítása
x %>% log() %>% sum() # eredmény kiírása
#> [1] 1.792
```

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 karbantarható 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 függvényvel 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] 2.303
c(10, 100) %>% log()      # ekvivalens alak: log(x=c(10, 100))
#> [1] 2.303 4.605
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 oprátor 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 a pont `.` helyörző segítségével expliciten jelezük, hogy hová kerül a pipe operátor bal oldaláról eérkező adat. Például a karakterstring cserére használatos `sub()` függvény `x=` argumentuma alapértelmezés szerint a harmadik:

```
sub(pattern = "s", replacement = "z", x = letters) # s cseréje z-re
#> [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "z"
#> [20] "t" "u" "v" "w" "x" "y" "z"
```

A következő függvényhívás az argumentumok nevesítése miatt szintén működőképes, az pipe-ban `x=letters` argumentumhívás valósul meg.

```
letters %>% sub(pattern = "s", replacement = "z")
#> [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "z"
#> [20] "t" "u" "v" "w" "x" "y" "z"
```

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.

```
letters %>% sub("s", "z") # Nem ezt akartuk. Valójában ez hívódik: sub(pattern=letters, replacement="s", x="")
```

Ha tehát nem az első argumentumban szeretnénk szerepelni a pipe bal oldaláról érkező adatot, akkor a . helyőrzőt kell használnunk:

```
letters %>% sub("s", "z", .) # letters %>% sub(pattern = "s", replacement = "z", x=letters)
#> [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "z"
#> [20] "t" "u" "v" "w" "x" "y" "z"
```

```
unisex_names %>% slice(1:3)
#> # A tibble: 3 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
unisex_names %>% slice(n())          # csak az utolsó sor
#> # A tibble: 1 x 5
#>   name      total male_share female_share     gap
#>   <chr>    <dbl>      <dbl>       <dbl>   <dbl>
#> 1 Aeon     100.      0.465      0.535  0.0703
unisex_names %>% slice((n()-4):n()) # utolsó 5 sor
#> # A tibble: 5 x 5
#>   name      total male_share female_share     gap
#>   <chr>    <dbl>      <dbl>       <dbl>   <dbl>
#> 1 Eaden    102.      0.573      0.427  0.146
#> 2 Inioluwa 101.      0.353      0.647  0.295
#> 3 Gwin     101.      0.562      0.438  0.124
#> 4 Yacine   100.      0.545      0.455  0.0892
#> 5 Aeon     100.      0.465      0.535  0.0703
unisex_names %>% slice(sample(1:n(), size=4)) # véletlen 4 sor
#> # A tibble: 4 x 5
#>   name      total male_share female_share     gap
#>   <chr>    <dbl>      <dbl>       <dbl>   <dbl>
#> 1 Kadian   125.      0.349      0.651  0.302
#> 2 Tonnie   989.      0.388      0.612  0.224
#> 3 Christian 2685.     0.501      0.499  0.00227
#> 4 Kalani   4012.     0.411      0.589  0.178
d.tbl <- unisex_names %>% slice(sample(21:24, size=5, replace = T )) # véletlen 4 sor
d.tbl
#> # A tibble: 5 x 5
#>   name      total male_share female_share     gap
#>   <chr>    <dbl>      <dbl>       <dbl>   <dbl>
#> 1 Blair   23160.     0.490      0.510  0.0195
#> 2 Carey   24790.     0.523      0.477  0.0465
#> 3 Carey   24790.     0.523      0.477  0.0465
```

```

#> 4 Kris 24956.      0.499      0.501 0.00275
#> 5 Blair 23160.     0.490      0.510 0.0195

d.tbl %>% distinct()  # a sorok egyedivé tétele
#> # A tibble: 3 x 5
#>   name    total male_share female_share     gap
#>   <chr>   <dbl>      <dbl>      <dbl>      <dbl>
#> 1 Blair  23160.     0.490      0.510 0.0195
#> 2 Carey  24790.     0.523      0.477 0.0465
#> 3 Kris   24956.     0.499      0.501 0.00275

unisex_names %>% sample_n(3)          # véletlen 3 sor
#> # A tibble: 3 x 5
#>   name    total male_share female_share     gap
#>   <chr>   <dbl>      <dbl>      <dbl>      <dbl>
#> 1 Mell    197.       0.639      0.361 0.278
#> 2 Bobie   436.       0.338      0.662 0.324
#> 3 Gracin  537.       0.658      0.342 0.316
unisex_names %>% sample_frac(0.01)  # véletlen 1%-nyi sor
#> # A tibble: 9 x 5
#>   name    total male_share female_share     gap
#>   <chr>   <dbl>      <dbl>      <dbl>      <dbl>
#> 1 Kris   24956.     0.499      0.501 0.00275
#> 2 Rylin  1406.      0.410      0.590 0.181
#> 3 Kodie  2070.      0.548      0.452 0.0956
#> 4 Shiva  581.       0.592      0.408 0.183
#> 5 Tajae  425.       0.403      0.597 0.194
#> 6 Jaidan 1354.      0.595      0.405 0.190
#> 7 Amen   624.       0.665      0.335 0.331
#> 8 Skylen 161.       0.401      0.599 0.198
#> 9 Ocean  2471.      0.485      0.515 0.0298

unisex_names %>% filter(gap<0.003, grepl(pattern = "^[C", x = name) | grepl(pattern = "^[K", x = name))
#> # A tibble: 6 x 5
#>   name    total male_share female_share     gap
#>   <chr>   <dbl>      <dbl>      <dbl>      <dbl>
#> 1 Kris   24956.     0.499      0.501 0.00275
#> 2 Camdyn 3275.      0.500      0.500 0.000154
#> 3 Christian 2685.     0.501      0.499 0.00227
#> 4 Kam    729.       0.501      0.499 0.00235
#> 5 Kodee  533.       0.500      0.500 0.000316
#> 6 Callaway 292.      0.500      0.500 0.000455

```

```
unisex_names %>% filter_all(any_vars(.>0.4))

#> # A tibble: 919 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
#> 6 Jaime   109870.    0.562      0.438  0.124
#> 7 Peyton  94896.     0.434      0.566  0.133
#> 8 Kerry   88964.     0.484      0.516  0.0321
#> 9 Jody    80401.     0.352      0.648  0.296
#> 10 Kendall 79211.    0.372      0.628  0.255
#> # ... with 909 more rows
```

```
smiths %>% select(kor=age, suly=weight)

#> # A tibble: 2 x 2
#>   kor   suly
#>   <dbl> <dbl>
#> 1 33    90
#> 2 NA    NA

smiths %>% rename(kor=age, suly=weight)

#> # A tibble: 2 x 5
#>   subject   time   kor   suly height
#>   <chr>     <dbl> <dbl> <dbl>   <dbl>
#> 1 John Smith     1    33    90    1.87
#> 2 Mary Smith     1    NA    NA    1.54
```

```
library(fivethirtyeight)
```

### 7.2.1.1. Oszlopok válogatása a Tidyverse R-ben

A Tidyverse R a `select()` függvényt kínálja az oszlopok indexelése. A `select()` függvény argumentumába az Alap R-ben látott fenti lehetőség mindegyikét használhatjuk, némi könnyítéssel:

```
# library(tidyverse)
# smiths.tbl %>% select(1:3)
# smiths.tbl %>% select(1, 3, 5)
# smiths.tbl %>% select(-1, -3, -5)
# smiths.tbl %>% select(subject, height)
```

Láthatjuk a `select()` elbírja a pozitív vagy negatív indexek közvetlen megadását, nem kell vektorban felsorolni őket, illetve az oszlopneveket idézőjel nélkül is megadhatjuk. További lehetőség, hogy az oszlopneveket úgy is használhatjuk, mintha számok lennének, a kettőspont operátor és a negatív előjel is használható velük:

```
# smiths.tbl %>% select(subject:age)
# smiths.tbl %>% select(~subject, -age)
# smiths.tbl %>% select(-(subject:age))
```

Az oszlopok között az oszlopnevek alapján is lehet válogatni, ehhez a Tidyverse R számos függvényt ajánl:

```
# smiths.tbl %>% select(starts_with("ti"))
# smiths.tbl %>% select(ends_with("t"))
# smiths.tbl %>% select(contains("ei"))
# d.df <- data.frame(id=1, kerd.1=10, kerd.2=20, kerd.3=30)
# d.df
# d.df %>% select(num_range("kerd.", 1:3))
```

Válogathatunk az oszlopok közül az oszlop adataira vonatkozó logikai értéket szolgáltató függvények segítségével. Erre a `select_if()` ad lehetőséget.

```
# smiths.tbl %>% select_if(is.numeric)
# smiths.tbl %>% select_if(is.character)
```

### 7.3. Haladó adatkezelés



Ebben a fejezetben:

## 8. fejezet

# Mutatók és táblázatok



### 8.1. Alap R lehetőségei



Ebben a fejezetben áttekintjük

- az *Alap R* mutatószámoló és
- táblázat készítő függvényeit
- 

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 fogalkozunk, 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 például 237 egyetemista testmagasságát tartalmazza a `Height` oszlopában. Számoljuk ki ezt a két fontos mutatót:

```
data(survey, package = "MASS") # survey betöltése
mean(survey$Height, na.rm = T) # testmagasság átlaga
#> [1] 172.4
sd(survey$Height, na.rm = T) # testmagasság szórása
#> [1] 9.848
```

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] 96.97
min(survey$Height, na.rm = T) # minimum
#> [1] 150
max(survey$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$Height, na.rm = T) # ferdeség
#> [1] 0.2174
moments::kurtosis(survey$Height, na.rm = T) # csúcsosság
#> [1] 2.587
```

#### 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 ez a működés lényege. Most három numerikus változó (kézméret, testmagasság és életkor) legfontosabb statisztikai mutatóit írtajuk ki:

```
# mutatók 3 változóra: apply()
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = mean, na.rm = T)
#> Wr.Hnd Height Age
#> 18.67 172.38 20.37
apply(survey[, c("Wr.Hnd", "Height", "Age")], MARGIN = 2, FUN = sd, na.rm = T)
#> Wr.Hnd Height Age
#> 1.879 9.848 6.474
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 16.75
#> 25% 17.5 165 17.67
#> 50% 18.5 171 18.58
#> 75% 19.8 180 20.17
#> 100% 23.2 200 73.00

# mutatók 3 változóra: sapply()
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = mean, na.rm = T)
#> Wr.Hnd Height Age
#> 18.67 172.38 20.37
sapply(survey[, c("Wr.Hnd", "Height", "Age")], FUN = sd, na.rm = T)
#> Wr.Hnd Height Age
#> 1.879 9.848 6.474
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 16.75
#> 25% 17.5 165 17.67
#> 50% 18.5 171 18.58
#> 75% 19.8 180 20.17
#> 100% 23.2 200 73.00
```

### 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$Wr.Hnd) # kézményet mutatói
#>   Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> 13.0    17.5   18.5 18.7   19.8 23.2    1
summary(survey$Height) # testmagasság mutatói
#>   Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
#> 150     165    171   172   180 200    28
summary(survey$Age)    # életkor mutatói
#>   Min. 1st Qu. Median Mean 3rd Qu. Max.
#> 16.8    17.7   18.6 20.4   20.2 73.0
```

### 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 minden össze 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 faktor egy vagy több csoportjában képes statisztikai mutató meghatározására. Fontos, hogy a statisztikai mutató egyetlen ér-

tékkel térjen vissza (pl. `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
#> 17.60 19.74
tapply(survey$Wr.Hnd, INDEX = survey[,c("Sex", "Smoke")], FUN = mean, na.rm=T)
#>           Smoke
#> Sex      Heavy Never Occas Regul
#> Female 17.90 17.66 16.82 17.48
#> Male    20.32 19.61 19.83 20.27
tapply(survey$Wr.Hnd, INDEX = survey[,c("Sex", "Smoke", "Exer")], FUN = mean, na.rm=T)
#> , , Exer = Freq
#>
#>           Smoke
#> Sex      Heavy Never Occas Regul
#> Female 17.73 17.53 15.92 19.55
#> Male    18.93 19.75 20.27 20.86
#>
#> , , Exer = None
#>
#>           Smoke
#> Sex      Heavy Never Occas Regul
#> Female   NA 17.59 18.50   NA
#> Male     23.2 19.29 17.95 19.5
#>
#> , , Exer = Some
#>
#>           Smoke
#> Sex      Heavy Never Occas Regul
#> Female 18.15 17.77 17.77 16.10
#> Male    23.00 19.47 20.50 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. Paramétereinek 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 17.60
#> 2 Male 19.74
by(survey[, "Wr.Hnd"], INDICES = survey[, "Sex"], drop=F), FUN = mean, na.rm=T)
#> Sex: Female
#> [1] 17.6
#> -----
#> Sex: Male
#> [1] 19.74
aggregate(survey[, "Wr.Hnd"], by=survey[, c("Sex", "Smoke")], FUN = mean, na.rm=T)
#>      Sex Smoke      x
#> 1 Female Heavy 17.90
#> 2 Male Heavy 20.32
#> 3 Female Never 17.66
#> 4 Male Never 19.61
#> 5 Female Occas 16.82
#> 6 Male Occas 19.83
#> 7 Female Regul 17.48
#> 8 Male Regul 20.27
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] 20.32
#> -----
#> Sex: Female
#> Smoke: Never
#> [1] 17.66
#> -----
#> Sex: Male
#> Smoke: Never
#> [1] 19.61
#> -----
#> Sex: Female
#> Smoke: Occas
#> [1] 16.82
#> -----
#> Sex: Male
#> Smoke: Occas
#> [1] 19.83
#> -----
#> Sex: Female
```

```
#> Smoke: Regul
#> [1] 17.48
#> -----
#> Sex: Male
#> Smoke: Regul
#> [1] 20.27
aggregate(survey[, "Wr.Hnd"], by=survey[, c("Sex", "Smoke", "Exer")], FUN = mean, na.rm=T)
#>      Sex Smoke Exer      x
#> 1 Female Heavy Freq 17.73
#> 2 Male Heavy Freq 18.93
#> 3 Female Never Freq 17.53
#> 4 Male Never Freq 19.75
#> 5 Female Occas Freq 15.92
#> 6 Male Occas Freq 20.27
#> 7 Female Regul Freq 19.55
#> 8 Male Regul Freq 20.86
#> 9 Male Heavy None 23.20
#> 10 Female Never None 17.59
#> 11 Male Never None 19.29
#> 12 Female Occas None 18.50
#> 13 Male Occas None 17.95
#> 14 Male Regul None 19.50
#> 15 Female Heavy Some 18.15
#> 16 Male Heavy Some 23.00
#> 17 Female Never Some 17.77
#> 18 Male Never Some 19.47
#> 19 Female Occas Some 17.77
#> 20 Male Occas Some 20.50
#> 21 Female Regul Some 16.10
#> 22 Male Regul Some 19.45
by(survey[, "Wr.Hnd"], INDICES = survey[, c("Sex", "Smoke", "Exer")], FUN = mean, na.rm=T)
#> Sex: Female
#> Smoke: Heavy
#> Exer: Freq
#> [1] 17.73
#> -----
#> Sex: Male
#> Smoke: Heavy
#> Exer: Freq
#> [1] 18.93
#> -----
#> Sex: Female
#> Smoke: Never
#> Exer: Freq
```

```
#> [1] 17.53
#> -----
#> Sex: Male
#> Smoke: Never
#> Exer: Freq
#> [1] 19.75
#> -----
#> Sex: Female
#> Smoke: Occas
#> Exer: Freq
#> [1] 15.92
#> -----
#> Sex: Male
#> Smoke: Occas
#> Exer: Freq
#> [1] 20.27
#> -----
#> Sex: Female
#> Smoke: Regul
#> Exer: Freq
#> [1] 19.55
#> -----
#> Sex: Male
#> Smoke: Regul
#> Exer: Freq
#> [1] 20.86
#> -----
#> 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] 19.29  
#> -----  
#> 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] 17.77  
#> -----  
#> Sex: Male  
#> Smoke: Never  
#> Exer: Some  
#> [1] 19.47  
#> -----  
#> Sex: Female
```

```
#> Smoke: Occas
#> Exer: Some
#> [1] 17.77
#> -----
#> 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 17.60 165.7 20.41
#> 2   Male 19.74 178.8 20.33
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 17.90 166.9 22.73
#> 2   Male Heavy 20.32 180.6 20.28
#> 3 Female Never 17.66 165.7 20.03
#> 4   Male Never 19.61 178.3 20.51
#> 5 Female Occas 16.82 167.4 21.91
#> 6   Male Occas 19.83 178.6 18.93
#> 7 Female Regul 17.48 159.8 22.87
#> 8   Male Regul 20.27 182.2 20.40
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 17.73 167.2 25.86
#> 2   Male Heavy Freq 18.93 179.7 20.46
#> 3 Female Never Freq 17.53 167.2 19.73
#> 4   Male Never Freq 19.75 180.3 20.47
```

```

#> 5 Female Occas Freq 15.92 166.9 19.75
#> 6 Male Occas Freq 20.27 178.4 19.26
#> 7 Female Regul Freq 19.55 167.0 19.79
#> 8 Male Regul Freq 20.86 182.6 21.99
#> 9 Male Heavy None 23.20 171.0 20.92
#> 10 Female Never None 17.59 163.0 20.40
#> 11 Male Never None 19.29 173.6 22.15
#> 12 Female Occas None 18.50      NaN 41.58
#> 13 Male Occas None 17.95 176.0 17.92
#> 14 Male Regul None 19.50 177.8 17.58
#> 15 Female Heavy Some 18.15 166.6 18.04
#> 16 Male Heavy Some 23.00 193.0 18.92
#> 17 Female Never Some 17.77 164.9 20.19
#> 18 Male Never Some 19.47 176.7 20.17
#> 19 Female Occas Some 17.77 168.3 18.94
#> 20 Male Occas Some 20.50 182.9 18.67
#> 21 Female Regul Some 16.10 156.2 24.92
#> 22 Male Regul Some 19.45 182.8 18.33

by(survey[, "Wr.Hnd"], INDICES = survey[, c("Sex", "Smoke")], FUN = shapiro.test)
#> Sex: Female
#> Smoke: Heavy
#>
#>   Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.86, p-value = 0.2
#>
#> -----
#> Sex: Male
#> Smoke: Heavy
#>
#>   Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.84, p-value = 0.1
#>
#> -----
#> Sex: Female
#> Smoke: Never
#>
#>   Shapiro-Wilk normality test
#>
#> data: dd[, ]

```

```
#> W = 0.95, p-value = 5e-04
#>
#> -----
#> Sex: Male
#> Smoke: Never
#>
#> Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.99, p-value = 0.5
#>
#> -----
#> Sex: Female
#> Smoke: Occas
#>
#> Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.97, p-value = 0.9
#>
#> -----
#> Sex: Male
#> Smoke: Occas
#>
#> Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.92, p-value = 0.3
#>
#> -----
#> Sex: Female
#> Smoke: Regul
#>
#> Shapiro-Wilk normality test
#>
#> data: dd[, ]
#> W = 0.9, p-value = 0.4
#>
#> -----
#> Sex: Male
#> Smoke: Regul
#>
#> Shapiro-Wilk normality test
#>
```

```
#> data: dd[x, ]
#> W = 0.93, p-value = 0.3
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   Min.   :16.9
#> 1st Qu.:17.0   1st Qu.:163   1st Qu.:17.5
#> Median  :17.5   Median  :167   Median  :18.4
#> Mean    :17.6   Mean    :166   Mean    :20.4
#> 3rd Qu.:18.5   3rd Qu.:170   3rd Qu.:20.0
#> Max.    :20.8   Max.    :180   Max.    :73.0
#> NA's    :16
#> -----
#> Sex: Male
#>      Wr.Hnd        Height         Age
#> Min.   :14.0   Min.   :155   Min.   :16.8
#> 1st Qu.:18.5   1st Qu.:173   1st Qu.:17.9
#> Median  :19.5   Median  :180   Median  :18.9
#> Mean    :19.7   Mean    :179   Mean    :20.3
#> 3rd Qu.:21.0   3rd Qu.:185   3rd Qu.:20.3
#> Max.    :23.2   Max.    :200   Max.    :70.4
#> 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 leírására szolgáltak. A gyakorisági táblázatok 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 jobbkazes, egy diáknak pedig nem jegyezték fel a kezességet. Hasonló eredményt kapunk, ha a táblázatok létrehozásra 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:

```
table(survey$Pulse, useNA = "ifany")
#>
#>   35   40   48   50   54   55   56   59   60   61   62   63   64   65   66   67
#>   1    1    2    2    1    1    1    1    12   1    4    1    9    6    6    1
#>   68   69   70   71   72   73   74   75   76   78   79   80   81   83   84   85
#>   16   1    13   2    14   1    5    5    13   4    3    18   1    4    5    4
#>   86   87   88   89   90   92   96   97   98   100  104 <NA>
#>   3    2    4    1    8    6    3    1    1    2    2    2    45
xtabs(~Pulse, data = survey, addNA = T)
#> Pulse
#>   35   40   48   50   54   55   56   59   60   61   62   63   64   65   66   67
#>   1    1    2    2    1    1    1    1    12   1    4    1    9    6    6    1
#>   68   69   70   71   72   73   74   75   76   78   79   80   81   83   84   85
#>   16   1    13   2    14   1    5    5    13   4    3    18   1    4    5    4
#>   86   87   88   89   90   92   96   97   98   100  104 <NA>
#>   3    2    4    1    8    6    3    1    1    2    2    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:

```
sort(table(survey$Pulse, useNA = "ifany")), decreasing = T)
#>
#> <NA>  80   68   72   70   76   60   64   90   65   66   92   74   75   84   62
#>   45   18   16   14   13   13   12   9    8    6    6    6    5    5    5    4
#>   78   83   85   88   79   86   96   48   50   71   87   100  104  35   40   54
#>   4    4    4    4    3    3    3    2    2    2    2    2    2    1    1    1
#>   55   56   59   61   63   67   69   73   81   89   97   98
#>   1    1    1    1    1    1    1    1    1    1    1    1    1    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:

```
table(survey$Sex, survey$W.Hnd, useNA = "ifany") # kezesség gyakorisági táblázata (2D)
#>
#>      Left Right <NA>
#> Female    7   110    1
#> Male     10   108    0
#> <NA>     1     0    0
ftable(table(survey$Sex, survey$W.Hnd, survey$Exer, useNA = "ifany")) # kezesség gyakorisági táblázata (3D)
```

```
#>          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.

```
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 fenti 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.

```
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()` függvényel átalakítjuk a mátrixunkat.

```
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
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 `prop.table()` függvényt használhatjuk.

```
kezesseg <- table(survey$W.Hnd)
prop.table(kezesseg)
#>
#> Left    Right
#> 0.07627 0.92373
```

```
100*prop.table(kezesseg)
#>
#>   Left  Right
#> 7.627 92.373
```

A könnyebb értelmezhetőség kedvéért használjuk a `round(digits=1)` függvényt:

```
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.

```
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.

```
apply(koff.csalad, MARGIN=1, FUN=sum)
#>      Házas Házas.volt Egyedül.él
#>      3029       141       718
apply(koff.csalad, MARGIN=2, FUN=sum)
#>      0   1-150 151-300    >300
#>     906   1910    742     330
margin.table(koff.csalad, margin=1)
#>      Házas Házas.volt Egyedül.él
#>      3029       141       718
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:

```
survey$Smoke <- ordered(survey$Smoke, levels=c("Never", "Occas", "Regul", "Heavy"))
dohanyzas <- table(survey$Smoke)
cumsum(round(100*prop.table(dohanyzas), digits=0))
#> Never Occas Regul Heavy
#>     80     88     95    100
```

## 8.2. Tidyverse R lehetőségei



Ebben a fejezetben áttekintjük

- a *Tidyverse R* mutatószámoló és
- táblázat készítő lehetőségeit
- 

Az R-ben létezik egy `|>` operátor, amely jelentősen megkönnyíti a kód olvasását.

```
10 |> sqrt() # sqrt(10)
#> [1] 3.162

survey %>%
  group_by(Sex) %>%
  summarise(mean_height = mean(Height, na.rm = TRUE))
```

```
#> # A tibble: 3 x 2
#>   Sex      mean_height
#>   <fct>     <dbl>
#> 1 Female      166.
#> 2 Male        179.
#> 3 <NA>        172
```

## 8.3. Haladó lehetőségek



Ebben a fejezetben áttekintjük

- a mutatók és táblázatok kiírásának kényelmes lehetőségeit

Ebben a fejezetben a **psych**, **DescTools** és a **summarytools** lehetőségeit tekintjük át. Mindhárom csomag rendkívül kényelmessé teszi a mutatók és a gyakorisági táblázatok kiírását. Mindhárom csomag képes több változót kezelní, több faktorra csoportosítva több statisztikai mutatót megjeleníteni, tehát a legnagyobb szabadságot adják a kutató kezébe.

### 8.3.1. Mutatók

#### 8.3.1.1. A teljes adattábla leírása

```
library(tidyverse)
survey %>% psych::describe(fast = T)
#>    vars   n   mean    sd    min   max range   se
#> Sex      1 236    NaN    NA    Inf -Inf -Inf    NA
#> Wr.Hnd   2 236  18.67  1.88  13.00 23.2 10.20 0.12
#> NW.Hnd   3 236  18.58  1.97  12.50 23.5 11.00 0.13
#> W.Hnd    4 236    NaN    NA    Inf -Inf -Inf    NA
#> Fold     5 237    NaN    NA    Inf -Inf -Inf    NA
#> Pulse    6 192  74.15 11.69  35.00 104.0 69.00 0.84
#> Clap     7 236    NaN    NA    Inf -Inf -Inf    NA
#> Exer     8 237    NaN    NA    Inf -Inf -Inf    NA
#> Smoke    9 236    NaN    NA    Inf -Inf -Inf    NA
#> Height   10 209 172.38  9.85 150.00 200.0 50.00 0.68
#> M.I      11 209    NaN    NA    Inf -Inf -Inf    NA
#> Age      12 237  20.37  6.47  16.75  73.0 56.25 0.42
DescTools::DescToolsOptions(digits=2)
survey %>% DescTools::Desc(plotit = F)
#> -----
#> Describe . (data.frame):
```

```

#>
#> data frame: 237 obs. of 12 variables
#>      168 complete cases (70.9%)
#>
#>   Nr  ColName  Class          NAs       Levels
#>   1   Sex       factor        1 (0.4%)  (2): 1-Female, 2-Male
#>   2   Wr.Hnd    numeric       1 (0.4%)
#>   3   NW.Hnd    numeric       1 (0.4%)
#>   4   W.Hnd     factor        1 (0.4%)  (2): 1-Left, 2-Right
#>   5   Fold      factor        .           (3): 1-L on R, 2-Neither, 3-R
#>                                on L
#>   6   Pulse     integer       45 (19.0%)
#>   7   Clap      factor        1 (0.4%)  (3): 1-Left, 2-Neither,
#>                                3-Right
#>   8   Exer      factor        .           (3): 1-Freq, 2-None, 3-Some
#>   9   Smoke     ordered, factor 1 (0.4%)  (4): 1-Never, 2-Occas,
#>                                3-Regul, 4-Heavy
#>  10  Height    numeric       28 (11.8%)
#>  11  M.I       factor        28 (11.8%) (2): 1-Imperial, 2-Metric
#>  12  Age       numeric       .
#>
#>
#> -----
#> 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)
#>
#> -----
#> 3 - NW.Hnd (numeric)
#>
#>      length      n     NAs unique      0s     mean   meanCI'
#>       237     236      1     68      0  18.58   18.33
#>      99.6%  0.4%
#>          0.0%           18.83
#>
#>      .05     .10     .25 median     .75     .90     .95
#>    15.50   16.30  17.50   18.50  19.72  21.00   22.22
#>
#>      range      sd   vcoef      mad      IQR     skew     kurt
#>    11.00    1.97    0.11    1.63    2.22    0.02    0.44
#>
#> lowest : 12.5, 13.0 (2), 13.3, 13.5, 15.0
#> highest: 22.7, 23.0, 23.2 (2), 23.3, 23.5
#>
#> heap(?): remarkable frequency (8.9%) for the mode(s) (= 18)
#>
#> ' 95%-CI (classic)
#>
#> -----
#> 4 - W.Hnd (factor - dichotomous)
#>
#>      length      n     NAs unique
#>       237     236      1      2
#>      99.6%  0.4%
#>
#>      freq     perc lci.95 uci.95'
#> Left      18    7.6%    4.9%   11.7%
#> Right     218   92.4%   88.3%  95.1%
#>
#> ' 95%-CI (Wilson)
#>

```

```

#> -----
#> 5 - Fold (factor)
#>
#>   length     n     NAs unique levels  dupes
#>     237     237      0      3      3      y
#>       100.0%  0.0%
#>
#>   level   freq   perc  cumfreq  cumperc
#> 1 R on L    120  50.6%      120  50.6%
#> 2 L on R     99  41.8%      219  92.4%
#> 3 Neither     18   7.6%      237 100.0%
#>
#> -----
#> 6 - Pulse (integer)
#>
#>   length     n     NAs unique      0s   mean  meanCI'
#>     237     192      45      43      0  74.15  72.49
#>       81.0%  19.0%           0.0%           75.81
#>
#>   .05   .10   .25 median   .75   .90   .95
#>  59.55  60.00  66.00    72.50  80.00  90.00  92.00
#>
#>   range     sd vcoef     mad    IQR   skew   kurt
#>   69.00  11.69  0.16   11.12  14.00 -0.02   0.33
#>
#> lowest : 35, 40, 48 (2), 50 (2), 54
#> highest: 96 (3), 97, 98, 100 (2), 104 (2)
#>
#> heap(?): remarkable frequency (9.4%) for the mode(s) (= 80)
#>
#> ' 95%-CI (classic)
#>
#> -----
#> 7 - Clap (factor)
#>
#>   length     n     NAs unique levels  dupes
#>     237     236      1      3      3      y
#>       99.6%  0.4%
#>
#>   level   freq   perc  cumfreq  cumperc
#> 1 Right    147  62.3%      147  62.3%
#> 2 Neither    50  21.2%      197  83.5%
#> 3 Left     39  16.5%      236 100.0%
#>
```

```

#> -----
#> 8 - 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%
#>
#> -----
#> 9 - Smoke (ordered, factor)
#>
#>   length     n     NAs unique levels  dupes
#>     237     236      1      4      4      y
#>           99.6%  0.4%
#>
#>   level   freq    perc  cumfreq  cumperc
#> 1 Never   189  80.1%     189  80.1%
#> 2 Occas   19   8.1%     208  88.1%
#> 3 Regul   17   7.2%     225  95.3%
#> 4 Heavy   11   4.7%     236 100.0%
#>
#> -----
#> 10 - Height (numeric)
#>
#>   length     n     NAs unique          os      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)
#>
#> -----

```

```

#> 11 - M.I (factor - dichotomous)
#>
#>   length      n      NAs unique
#>   237     209      28      2
#>           88.2% 11.8%
#>
#>   freq    perc  lci.95  uci.95'
#> Imperial    68 32.5%  26.5% 39.2%
#> Metric      141 67.5%  60.8% 73.5%
#>
#>   ' 95%-CI (Wilson)
#>
#> -----
#> 12 - Age (numeric)
#>
#>   length      n      NAs unique      0s    mean  meanCI'
#>   237     237      0      88      0 20.37 19.55
#>       100.0%  0.0%      0.0%          21.20
#>
#>   .05     .10     .25 median     .75    .90    .95
#> 17.08 17.22 17.67 18.58 20.17 23.58 30.68
#>
#>   range      sd vcoef      mad      IQR skew kurt
#> 56.25  6.47  0.32     1.61    2.50  5.16 33.47
#>
#>   lowest : 16.75, 16.92 (3), 17.0 (2), 17.08 (7), 17.17 (11)
#> highest: 41.58, 43.83, 44.25, 70.42, 73.0
#>
#>   ' 95%-CI (classic)
survey %>% summarytools::dfSummary()
#> Data Frame Summary
#> survey
#> Dimensions: 237 x 12
#> Duplicates: 0
#>
#> -----
#> No Variable            Stats / Values        Freqs (% of Valid) Graph          Valid      Missing
#> -----
#> 1  Sex                  1. Female          118 (50.0%) IIIIIIIIII 236      1
#>     [factor]             2. Male           118 (50.0%) IIIIIIIIII (99.6%) (0.4%)
#>
#> 2  Wr.Hnd               Mean (sd) : 18.7 (1.9) 60 distinct values : . 236      1
#>     [numeric]             min < med < max:           : : . (99.6%) (0.4%)
#>                           13 < 18.5 < 23.2           . : : :
```

```

#>                               IQR (CV) : 2.3 (0.1) : : : : :
#>                               . : : : : : : : :
#>
#> 3   NW.Hnd           Mean (sd) : 18.6 (2)    68 distinct values : 236
#>      [numeric]          min < med < max: : . (99.6%)
#>                  12.5 < 18.5 < 23.5 : : :
#>                               IQR (CV) : 2.2 (0.1) . : : : .
#>
#> 4   W.Hnd            1. Left             18 ( 7.6%) I 236
#>      [factor]          2. Right            218 (92.4%) IIIIIIIIIIIIIIIIIIII (99.6%)
#>
#> 5   Fold              1. L on R        99 (41.8%) IIIIIIII 237
#>      [factor]          2. Neither       18 ( 7.6%) I (100.0%)
#>                  3. R on L        120 (50.6%) IIIIIIIIII
#>
#> 6   Pulse             Mean (sd) : 74.2 (11.7) 43 distinct values . : 192
#>      [integer]          min < med < max: : : (81.0%)
#>                  35 < 72.5 < 104 : : .
#>                               IQR (CV) : 14 (0.2) : : :
#>
#> 7   Clap              1. Left             39 (16.5%) III 236
#>      [factor]          2. Neither       50 (21.2%) III (99.6%)
#>                  3. Right          147 (62.3%) IIIIIIIIIIIIII
#>
#> 8   Exer              1. Freq            115 (48.5%) IIIIIIII 237
#>      [factor]          2. None            24 (10.1%) II (100.0%)
#>                  3. Some           98 (41.4%) IIIIIIII
#>
#> 9   Smoke             1. Never          189 (80.1%) IIIIIIIIIIIIIIII 236
#>      [ordered, factor]  2. Occas          19 ( 8.1%) I (99.6%)
#>                  3. Regul          17 ( 7.2%) I
#>                  4. Heavy          11 ( 4.7%) I
#>
#> 10  Height            Mean (sd) : 172.4 (9.8) 67 distinct values : 209
#>      [numeric]          min < med < max: : . . (88.2%)
#>                  150 < 171 < 200 : : : : :
#>                               IQR (CV) : 15 (0.1) : : : : : : .
#>
#> 11  M.I               1. Imperial      68 (32.5%) IIIIII 209
#>      [factor]          2. Metric        141 (67.5%) IIIIIIIIIIIIII (88.2%)
#>
```

```
#> 12   Age           Mean (sd) : 20.4 (6.5)    88 distinct values : 237      0
#>      [numeric]      min < med < max:          : (100.0%) (0.0%)
#>                  16.8 < 18.6 < 73            : 
#>                  IQR (CV) : 2.5 (0.3)        : 
#>                  : . 
#> -----
```

### 8.3.1.2. Több numerikus változó

```
num.valtozok <- survey[, c("Wr.Hnd", "NW.Hnd", "Height")]
num.valtozok %>% psych::describe()
#>     vars   n   mean   sd median trimmed   mad   min   max range skew
#> Wr.Hnd   1 236 18.67 1.88   18.5  18.61  1.48  13.0 23.2 10.2 0.18
#> NW.Hnd   2 236 18.58 1.97   18.5  18.55  1.63  12.5 23.5 11.0 0.02
#> Height   3 209 172.38 9.85 171.0 172.19 10.08 150.0 200.0 50.0 0.22
#>             kurtosis   se
#> Wr.Hnd     0.30 0.12
#> NW.Hnd     0.44 0.13
#> Height    -0.44 0.68
num.valtozok %>% DescTools::Desc(plotit = F)
#> -----
#> Describe . (data.frame):
#>
#> data frame: 237 obs. of 3 variables
#>     208 complete cases (87.8%)
#>
#>     Nr  ColName Class   NAs       Levels
#>     1   Wr.Hnd numeric  1 (0.4%)
#>     2   NW.Hnd numeric  1 (0.4%)
#>     3   Height numeric 28 (11.8%)
#>
#> -----
#> # 1 - 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)
#>
#> -----
#> 2 - NW.Hnd (numeric)
#>
#>      length      n     NAs   unique      0s     mean   meanCI'
#>      237       236       1       68       0  18.58   18.33
#>      99.6%   0.4%                  0.0%           18.83
#>
#>      .05      .10      .25 median      .75      .90      .95
#>      15.50   16.30   17.50   18.50   19.72   21.00   22.22
#>
#>      range      sd vcoef      mad     IQR     skew     kurt
#>      11.00   1.97   0.11     1.63   2.22   0.02   0.44
#>
#> lowest : 12.5, 13.0 (2), 13.3, 13.5, 15.0
#> highest: 22.7, 23.0, 23.2 (2), 23.3, 23.5
#>
#> heap(?): remarkable frequency (8.9%) for the mode(s) (= 18)
#>
#> ' 95%-CI (classic)
#>
#> -----
#> 3 - 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)
num.valtozok %>% summarytools::descr()
#> Descriptive Statistics
#> num.valtozok
#> N: 237
#>
#>          Height   NW.Hnd   Wr.Hnd
#> -----
#>      Mean    172.38   18.58   18.67
#>      Std.Dev     9.85    1.97    1.88
#>      Min     150.00   12.50   13.00
#>      Q1      165.00   17.50   17.50
#>      Median    171.00   18.50   18.50
#>      Q3      180.00   19.75   19.80
#>      Max     200.00   23.50   23.20
#>      MAD      10.08    1.63    1.48
#>      IQR      15.00    2.22    2.30
#>      CV       0.06    0.11    0.10
#>      Skewness    0.22    0.02    0.18
#>      SE.Skewness  0.17    0.16    0.16
#>      Kurtosis   -0.44    0.44    0.30
#>      N.Valid    209.00   236.00   236.00
#>      Pct.Valid   88.19    99.58   99.58
```

### 8.3.2. Táblázatok

#### Egydimenziós táblázatok

```
# faktor
survey$W.Hnd %>% DescTools::Desc(plotit = F)
#> -----
#> . (factor - dichotomous)
#>
#>   length      n     NAs unique
#>   237      236      1      2
#>   99.6%    0.4%
#>
#>   freq     perc   lci.95  uci.95'
#> Left      18    7.6%    4.9%   11.7%
#> Right     218   92.4%   88.3%  95.1%
#>
#> ' 95%-CI (Wilson)
survey$W.Hnd %>% summarytools::freq()
```

```

#> Frequencies
#> survey$W.Hnd
#> Type: Factor
#>
#>          Freq % Valid % Valid Cum. % Total % Total Cum.
#> -----
#>     Left      18    7.63      7.63    7.59    7.59
#>     Right     218   92.37    100.00   91.98   99.58
#>     <NA>      1        0.42      0.42   100.00
#>     Total     237   100.00    100.00   100.00   100.00
#> rendezett faktor
survey$Smoke %>% DescTools:::Desc(plotit = F)
#> -----
#> . (ordered, factor)
#>
#>   length      n      NAs unique levels  dupes
#>   237      236       1       4       4       y
#>   99.6%  0.4%
#>
#>   level   freq   perc  cumfreq  cumperc
#> 1 Never    189  80.1%    189   80.1%
#> 2 Occas     19   8.1%    208   88.1%
#> 3 Regul     17   7.2%    225   95.3%
#> 4 Heavy      11   4.7%    236 100.0%
survey$Smoke %>% summarytools:::freq()
#> Frequencies
#> survey$Smoke
#> Type: Ordered Factor
#>
#>          Freq % Valid % Valid Cum. % Total % Total Cum.
#> -----
#>     Never    189  80.08    80.08   79.75   79.75
#>     Occas     19   8.05    88.14    8.02   87.76
#>     Regul     17   7.20    95.34    7.17   94.94
#>     Heavy      11   4.66   100.00    4.64   99.58
#>     <NA>      1        0.42      0.42   100.00
#>     Total     237   100.00   100.00   100.00   100.00

```

### Kétdimenziós táblázatok

```

# faktor
library(magrittr)
survey %>% DescTools:::Desc(Sex~W.Hnd, data=., plotit = F)
#> -----

```

```

#> Sex ~ W.Hnd (.)
#>
#> Summary:
#> n: 235, rows: 2, columns: 2
#>
#> Pearson's Chi-squared test (cont. adj):
#> X-squared = 0.24, df = 1, p-value = 0.6
#> Fisher's exact test p-value = 0.6
#> McNemar's chi-squared = 82, df = 1, p-value <2e-16
#>
#> estimate lwr.ci upr.ci'
#>
#> odds ratio      0.687  0.252  1.871
#> rel. risk (col1) 0.706  0.278  1.792
#> rel. risk (col2) 1.027  0.956  1.103
#>
#>
#> Contingency Coeff.   0.048
#> Cramer's V          0.048
#> Kendall Tau-b      -0.048
#>
#>
#>      W.Hnd   Left   Right    Sum
#> Sex
#>
#> Female   freq      7     110    117
#>           perc   3.0%  46.8%  49.8%
#>           p.row  6.0%  94.0%   .
#>           p.col 41.2%  50.5%   .
#>
#> Male     freq     10     108    118
#>           perc   4.3%  46.0%  50.2%
#>           p.row  8.5%  91.5%   .
#>           p.col 58.8%  49.5%   .
#>
#> Sum     freq     17     218    235
#>           perc   7.2%  92.8% 100.0%
#>           p.row  .     .     .
#>           p.col  .     .     .
#>
#>
#> -----
#> ' 95% conf. level
survey %$% summarytools::ctable(x = Sex, y = W.Hnd)

```

```
#> Cross-Tabulation, Row Proportions
#> Sex * W.Hnd
#> Data Frame: survey
#>
#> -----
#>          W.Hnd      Left     Right    <NA>   Total
#> Sex
#> Female       7 ( 5.9%) 110 (93.2%) 1 ( 0.8%) 118 (100.0%)
#> Male         10 ( 8.5%) 108 (91.5%) 0 ( 0.0%) 118 (100.0%)
#> <NA>        1 (100.0%) 0 ( 0.0%) 0 ( 0.0%) 1 (100.0%)
#> Total        18 ( 7.6%) 218 (92.0%) 1 ( 0.4%) 237 (100.0%)
#> -----
```

### 8.3.2.1. Mutatók csoportokra

```
survey$Height %>% psych::describeBy(x = ., g = survey[,c("Sex", "W.Hnd")], mat=T)
#>      item group1 group2 vars  n  mean      sd median trimmed  mad  min max
#> X11     1 Female  Left    1  6 167.7  4.920 170.0  167.7 2.224 160.0 172
#> X12     2  Male   Left    1  9 180.5 11.070 180.0  180.5 7.028 165.0 200
#> X13     3 Female  Right   1 95 165.4  6.063 166.4  165.7 5.337 150.0 178
#> X14     4  Male   Right   1 97 178.7  8.142 180.0  179.0 7.413 154.9 196
#>      range   skew kurtosis   se
#> X11 11.98 -0.5684 -1.7387 2.0085
#> X12 35.00  0.2830 -1.1371 3.6899
#> X13 28.00 -0.3992 -0.4297 0.6221
#> X14 41.06 -0.3472 -0.1779 0.8267
num.valtozok %>% psych::describeBy(x = ., g = survey[,c("Sex", "W.Hnd")], mat=T)
#>      item group1 group2 vars  n  mean      sd median trimmed  mad  min max
#> Wr.Hnd1   1 Female  Left    1  7 18.16  1.4478 18.50  18.16 0.7413 15.4
#> Wr.Hnd2   2  Male   Left    1 10 19.99  1.6455 19.65  19.93 1.3343 17.5
#> Wr.Hnd3   3 Female  Right   1 110 17.60  1.2353 17.50  17.63 1.1119 13.0
#> Wr.Hnd4   4  Male   Right   1 107 19.72  1.7658 19.50  19.71 1.4826 14.0
#> NW.Hnd1   5 Female  Left    2  7 17.94  0.9467 18.00  17.94 0.7413 16.4
#> NW.Hnd2   6  Male   Left    2 10 19.84  1.5138 19.75  19.93 1.4826 17.0
#> NW.Hnd3   7 Female  Right   2 110 17.47  1.3714 17.55  17.51 1.0378 12.5
#> NW.Hnd4   8  Male   Right   2 107 19.70  1.8351 19.50  19.72 1.4826 13.3
#> Height1   9 Female  Left    3  6 167.67  4.9199 170.00  167.67 2.2239 160.0
#> Height2  10  Male   Left    3  9 180.54 11.0697 180.00  180.54 7.0275 165.0
#> Height3  11 Female  Right   3 95 165.41  6.0631 166.40  165.65 5.3374 150.0
#> Height4  12  Male   Right   3 97 178.67  8.1425 180.00  178.96 7.4130 154.9
#>      max range   skew kurtosis   se
#> Wr.Hnd1 20.0  4.60 -0.66791 -0.72047 0.5472
#> Wr.Hnd2 23.0  5.50  0.29434 -0.95330 0.5204
```

```
#> Wr.Hnd3 20.8 7.80 -0.43157 1.34878 0.1178
#> Wr.Hnd4 23.2 9.20 -0.06857 0.01661 0.1707
#> NW.Hnd1 19.5 3.10 0.02948 -0.92181 0.3578
#> NW.Hnd2 22.0 5.00 -0.32153 -1.02247 0.4787
#> NW.Hnd3 20.7 8.20 -0.64755 1.83733 0.1308
#> NW.Hnd4 23.5 10.20 -0.27990 0.67647 0.1774
#> Height1 172.0 11.98 -0.56840 -1.73871 2.0085
#> Height2 200.0 35.00 0.28304 -1.13707 3.6899
#> Height3 178.0 28.00 -0.39916 -0.42966 0.6221
#> Height4 196.0 41.06 -0.34723 -0.17790 0.8267
survey %>% DescTools::Desc(Height ~ Sex * W.Hnd, data=., plotit = F)
#> -----
#> Height ~ Sex (.)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 2
#>
#>
#>      Female     Male
#> mean    165.69 178.83
#> median   166.75 180.00
#> sd       6.15   8.38
#> IQR      7.44   12.21
#> n        102    106
#> np      49.04% 50.96%
#> NAs       16     12
#> 0s        0      0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 97, df = 1, p-value <2e-16
#>
#>
#> Warning:
#> Grouping variable contains 1 NAs (0.422%).
#>
#> -----
#> Height ~ W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 2
#>
#>
#>      Left     Right
#> mean    175.18 172.11
```

```

#> median 172.00 170.59
#> sd       10.67   9.78
#> IQR      11.22  15.00
#> n        16     192
#> np      7.69% 92.31%
#> NAs      2      26
#> 0s       0      0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 0.89, df = 1, p-value = 0.3
#>
#>
#> Warning:
#> Grouping variable contains 1 NAs (0.422%).
#>
#> -----
#> Height ~ Sex:W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 207 (87.3%), missings: 30 (12.7%), groups: 4
#>
#>
#>      Female:Left    Male:Left   Female:Right   Male:Right
#> mean      167.67      180.54      165.41      178.67
#> median     170.00      180.00      166.40      180.00
#> sd         4.92       11.07       6.06       8.14
#> IQR        6.00       7.62       7.47      12.28
#> n          6          9          95         97
#> np        2.90%      4.35%      45.89%      46.86%
#> NAs        1          1          15         11
#> 0s         0          0          0          0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 99, df = 3, p-value <2e-16
#>
#>
#> Warning:
#> Grouping variable contains 2 NAs (0.844%).
#> survey %>% DescTools::Desc(Height + Age ~ Sex * W.Hnd, data=., plotit = F)
#> -----
#> Height ~ Sex (.)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 2

```

```
#>
#>
#>           Female     Male
#> mean      165.69   178.83
#> median    166.75   180.00
#> sd        6.15    8.38
#> IQR       7.44   12.21
#> n         102     106
#> np       49.04%  50.96%
#> Nas       16      12
#> 0s        0       0
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 97, df = 1, p-value <2e-16
#>
#>
#> Warning:
#>   Grouping variable contains 1 NAs (0.422%).
#>
#> -----
#> Height ~ W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 208 (87.8%), missings: 29 (12.2%), groups: 2
#>
#>
#>           Left     Right
#> mean      175.18   172.11
#> median    172.00   170.59
#> sd        10.67    9.78
#> IQR       11.22   15.00
#> n         16     192
#> np       7.69%  92.31%
#> Nas       2      26
#> 0s        0       0
#>
#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 0.89, df = 1, p-value = 0.3
#>
#>
#> Warning:
#>   Grouping variable contains 1 NAs (0.422%).
#>
#> -----
```

```
#> Height ~ Sex:W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 207 (87.3%), missings: 30 (12.7%), groups: 4
#>
#>
#>      Female:Left   Male:Left   Female:Right   Male:Right
#> mean       167.67      180.54      165.41      178.67
#> median      170.00      180.00      166.40      180.00
#> sd          4.92       11.07       6.06       8.14
#> IQR         6.00        7.62        7.47       12.28
#> n            6           9          95          97
#> np         2.90%      4.35%      45.89%      46.86%
#> NAs          1           1          15          11
#> 0s            0           0           0           0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 99, df = 3, p-value <2e-16
#>
#>
#> Warning:
#> Grouping variable contains 2 NAs (0.844%).
#>
#> -----
#> Age ~ Sex (.)
#>
#> Summary:
#> n pairs: 237, valid: 236 (99.6%), missings: 1 (0.4%), groups: 2
#>
#>
#>      Female     Male
#> mean      20.41    20.33
#> median     18.42    18.88
#> sd         6.91     6.07
#> IQR        2.48     2.37
#> n          118      118
#> np        50.00%   50.00%
#> NAs          0        0
#> 0s            0        0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 4.3, df = 1, p-value = 0.04
#>
#>
```

```

#> Warning:
#>   Grouping variable contains 1 NAs (0.422%).
#>
#> -----
#> Age ~ W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 236 (99.6%), missings: 1 (0.4%), groups: 2
#>
#>
#>      Left    Right
#> mean    18.63   20.53
#> median   18.50   18.58
#> sd       1.29    6.72
#> IQR      1.52    2.62
#> n        18     218
#> np      7.63%  92.37%
#> Nas      0      0
#> 0s      0      0
#>
#> Kruskal-Wallis rank sum test:
#> Kruskal-Wallis chi-squared = 0.72, df = 1, p-value = 0.4
#>
#>
#> Warning:
#>   Grouping variable contains 1 NAs (0.422%).
#>
#> -----
#> Age ~ Sex:W.Hnd (.)
#>
#> Summary:
#> n pairs: 237, valid: 235 (99.2%), missings: 2 (0.8%), groups: 4
#>
#>
#>      Female:Left   Male:Left  Female:Right  Male:Right
#> mean      18.36     18.53     20.57     20.50
#> median    18.50     18.46     18.42     18.92
#> sd        1.02      1.22      7.12      6.31
#> IQR       1.63      1.08      2.67      2.42
#> n         7         10       110       108
#> np      2.98%     4.26%    46.81%   45.96%
#> Nas      0         0         0         0
#> 0s      0         0         0         0
#>

```

```

#> Kruskal-Wallis rank sum test:
#>   Kruskal-Wallis chi-squared = 5.7, df = 3, p-value = 0.1
#>
#>
#> Warning:
#>   Grouping variable contains 2 NAs (0.844%).
num.valtozok %>% summarytools::stby(
  data = .,
  INDICES = survey[, c("W.Hnd", "Sex")],
  FUN = summarytools:::descr,
  stats = "common"
)
#> Descriptive Statistics
#> num.valtozok
#> N: 7
#>
#>           Height  NW.Hnd  Wr.Hnd
#> -----
#>      Mean    167.67   17.94   18.16
#> Std.Dev     4.92    0.95   1.45
#> Min       160.02   16.40   15.40
#> Median     170.00   18.00   18.50
#> Max       172.00   19.50   20.00
#> N.Valid     6.00    7.00    7.00
#> Pct.Valid   85.71  100.00  100.00
#>
#> N: 110
#>
#>           Height  NW.Hnd  Wr.Hnd
#> -----
#>      Mean    165.41   17.47   17.60
#> Std.Dev     6.06    1.37   1.24
#> Min       150.00   12.50   13.00
#> Median     166.40   17.55   17.50
#> Max       178.00   20.70   20.80
#> N.Valid    95.00  110.00  110.00
#> Pct.Valid   86.36  100.00  100.00
#>
#> N: 10
#>
#>           Height  NW.Hnd  Wr.Hnd
#> -----
#>      Mean    180.54   19.84   19.99
#> Std.Dev    11.07    1.51    1.65

```

```
#>           Min   165.00   17.00   17.50
#>           Median 180.00  19.75  19.65
#>           Max 200.00  22.00  23.00
#>           N.Valid    9.00   10.00   10.00
#>           Pct.Valid  90.00 100.00 100.00
#>
#> N: 108
#>
#>           Height  NW.Hnd  Wr.Hnd
#> -----
#>           Mean   178.67  19.70  19.72
#>           Std.Dev  8.14   1.84   1.77
#>           Min    154.94  13.30  14.00
#>           Median 180.00  19.50  19.50
#>           Max    196.00  23.50  23.20
#>           N.Valid  97.00 107.00 107.00
#>           Pct.Valid 89.81  99.07  99.07
```

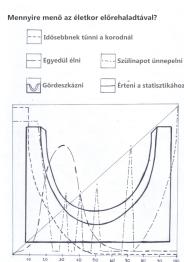
### Feladatok

RcmdrMisc - numSummary jmv - descriptives



## 9. fejezet

# Grafika az R-ben



Ebben a fejezetben áttekintjük:



- az R grafikus rendszerei
- a hagyományos grafika alapfogalmai
- magasszintű és alacsonyszintű rajzfüggvények a hagyományos grafikában
- a ggplot2 rendszer alapelve
- ábrák létrehozása ggplot2-ben
- ábrák mentése háttértárra

Az R-ben több grafikus rendszer közül választhatunk, amikor az ábráink 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 létrehozásáért felelősek, az alacsony szintű függvényekkel pedig kisebb-nagyobb grafikus elemeket helyezhetünk a meglévő ábrára. Mindig „rárajzolunk” a meglévő elemekre, a kézőbbi módosításra vagy törlésre nincs lehetőségünk.

- A **grid** csomagból elérhető grafikus rendszer grafikus primitívek rendkívül gazdag tárháza. Segítségükkel grafikus objektumokat építhetünk, amelyek az ábráról független reprezentációval rendelkeznek, így azok később módosíthatók. A saját koordináta-rendszерrel rendelkező viewport-ok rendszere tetszőlegesen bonyolult ábrák létrehozását segíti. A grid rendszer maga nem tartalmaz statistikai rajzfüggvényeket, de számos, a grid csomagra épülő rendszer igen (pl. lattice és 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.

Jelen könyvben csak ggplot2 grafikus rendszert ismertetjük.

## 9.1. A modern grafika (ggplot2)

A ggplot2 grafika teljesen más megközelítést használ az ábrák létrehozásához, mint az Alap R garfikus rendszerei. Nevét Leland Wilkinson Grammar of Graphics könyve után kapta, amely a grafika általános formális és strukturált leírására tett kísérletet. A ggplot2 grafika a Hadley Wickham által készített **ggplot2** csomaggal érhető el. A ggplot2 ábrák létrehozása némi tanulás után átlátható és következetes lesz, maguk az elkészült grafikák gyönyörűek és alkalmasak azonnali publikására.

Egy ggplot2 grafika három összetevőn alapul:

- *adat*: A megjelenítés egy adattáblában alapul (tibble vagy data frame). Az adatoszlopok ennek megfelelően többnyire numerikus vektorok vagy faktorok.
- *megjelenés*: Az ábrán megjelenő elemek lehetséges paraméterei, például pont esetén az x és y koordináta, a szín, a méret vagy alak, oszlopok esetén például a magasság és kitöltő szín.
- *ábra elem*: Az ábrán megjelenő elemek, például pont, oszlop vagy vonal. Ezek határozzák meg végső soron az ábra típusát, így létrehozhatunk például pontdiagramot, oszlopdiagramot vagy hisztogramot, és vonal diagramot, de ezekből egy ábrán akár többet is felhasználhatunk.

Nézzünk egy egyszerű példát! Hozzunk létre egy pontdiagramot a következő paranccsal:

```
ggplot(data=d.tbl, mapping=aes(x=pont.1, y=pont.2)) + geom_point()
```

A legtöbb ggplot2 ábra létrehozása a `ggplot()` függvényel indul, ahol a `data=` argumentum az *adat* részt tartalmazza, a `mapping=` argumentum pedig *megjelenés* paramétereit

sorolja fel. Ehhez az `aes()` függvényt használjuk, amely adatoszlopok és a megjelenéshez tartozó paraméterek közötti kapcsolatot is megadja egyben. Az `ábra elemeit +` operátorral adjuk hozzá a `ggplot()` függvényhez, ez most a pontelemeket létrehozó `geom_point()`.

A fenti parancs `adat` része egy 3 sorból és 4 oszloból álló adattábla, amely egy karakteres (`csoport`) és három numerikus oszlopot (`pont.1`, `pont.2`, `pont.3`) tartalmaz.

```
library(tibble)
d.tbl <- tribble(
  ~csoport, ~pont.1, ~pont.2, ~pont.3,
  "AA", 15, 42, 12,
  "BB", 20, 28, 18,
  "CC", 35, 12, 21
)
```

Az ábra létrehozásához a `ggplot()` függvényt és a pontelemek hozzáadásához a `geom_point()` függvényt használjuk, az utóbbiban nincs is szükség argzmentumok megadására. A `ggplot()` első argumentuma a `data=d.tbl`, amely az adatösszetevőt tisztázza, a `mapping=` argumentumban az `aes()` függvényben az `x` és `y` megjelenítési paramétereiről gondoskodunk, a `pont.1` és `pont.2` oszlopok megadásával. Az `x` és `y` megjelenési paraméterek természetesen a hozzáadott pontok `x` és `y` koordinátáira vonatkoznak.

Természetesen a pontelemek helyett használhatunk oszlopelemeket is a `geom_col()` segítségével, itt az `x` és `y` megjelenítési paraméterek az oszlop `x` koordinátáját és az oszlop magasságát jelentik.

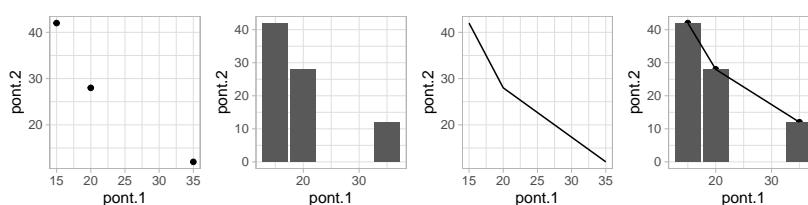
```
ggplot(data=d.tbl, mapping=aes(x=pont.1, y=pont.2)) + geom_col()
```

Ha a `geom_line()` ábraelemeket adjuk az ábrához, akkor az `x` és `y` paraméterek a vonalak létrehozásához használt pontok `x` és `y` koordinátái.

```
ggplot(data=d.tbl, mapping=aes(x=pont.1, y=pont.2)) + geom_line()
```

Egyetlen ábrán akár mindenáron ábraelem (pont, oszlop, vonal) is megjelenhet:

```
ggplot(data=d.tbl, mapping=aes(x=pont.1, y=pont.2)) + geom_point() + geom_col() + geom_line()
```

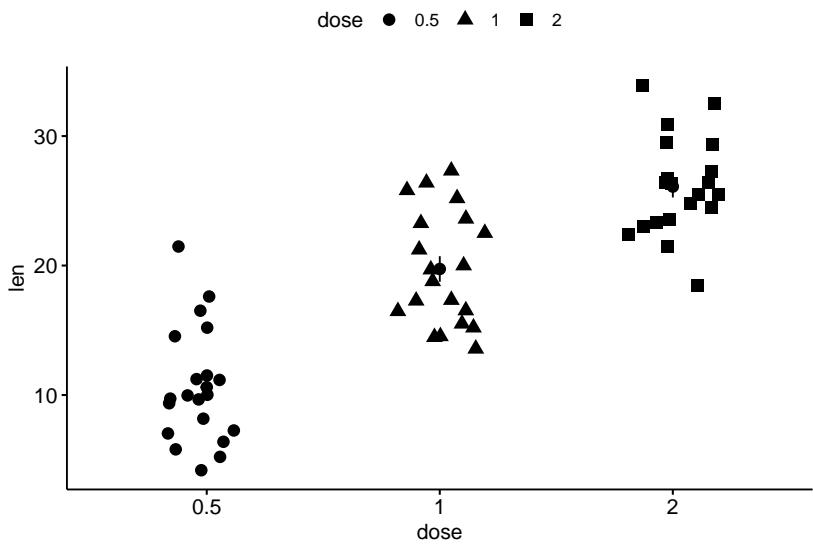


```

library(ggpubr)
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot with summary statistics: mean_se
# ++++++
# Change point shapes by groups: "dose"
ggstripchart(df, x = "dose", y = "len",
  shape = "dose", size = 3,
  add = "mean_se")

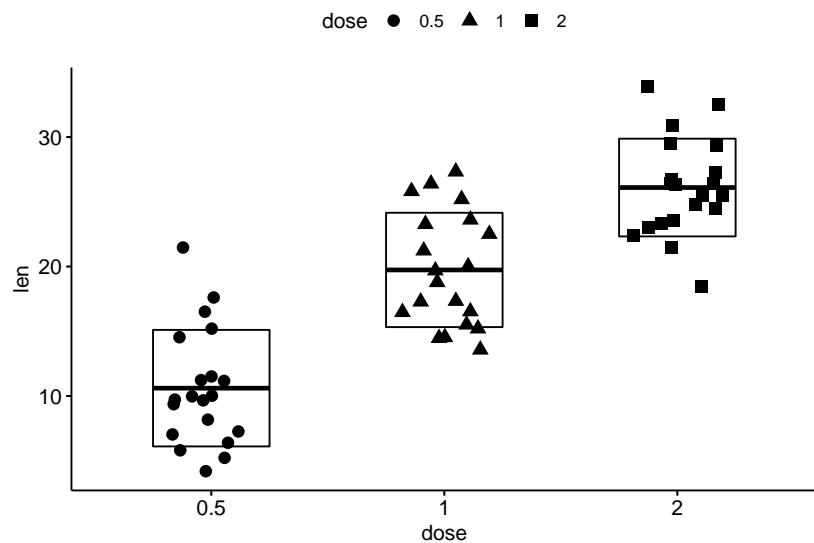
```



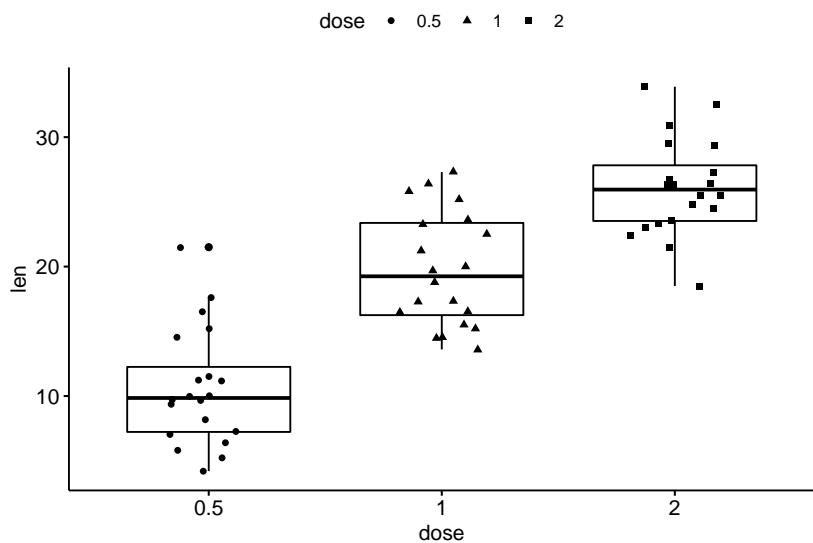
```

# Use mean_sd
# Change error.plot to "crossbar"
ggstripchart(df, x = "dose", y = "len",
  shape = "dose", size = 3,
  add = "mean_sd", add.params = list(width = 0.5),
  error.plot = "crossbar")

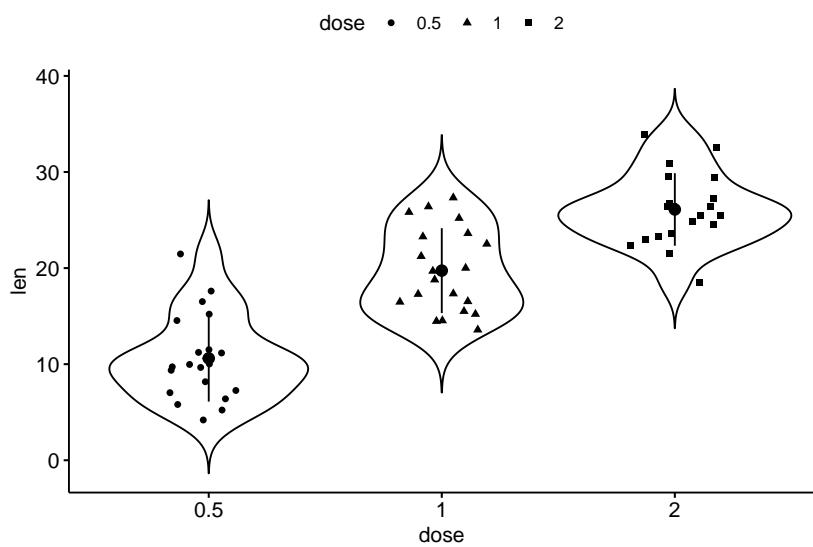
```



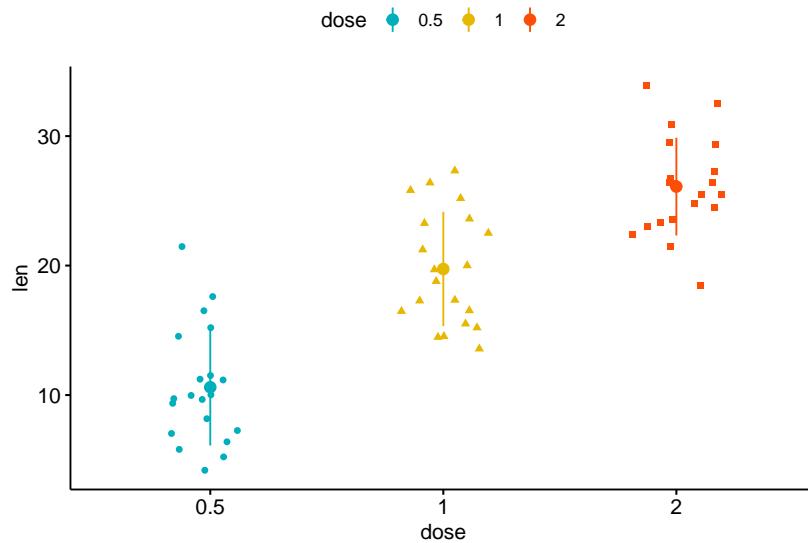
```
# Add summary statistics
# ++++++
# Add box plot
ggstripchart(df, x = "dose", y = "len",
  shape = "dose", add = "boxplot")
```



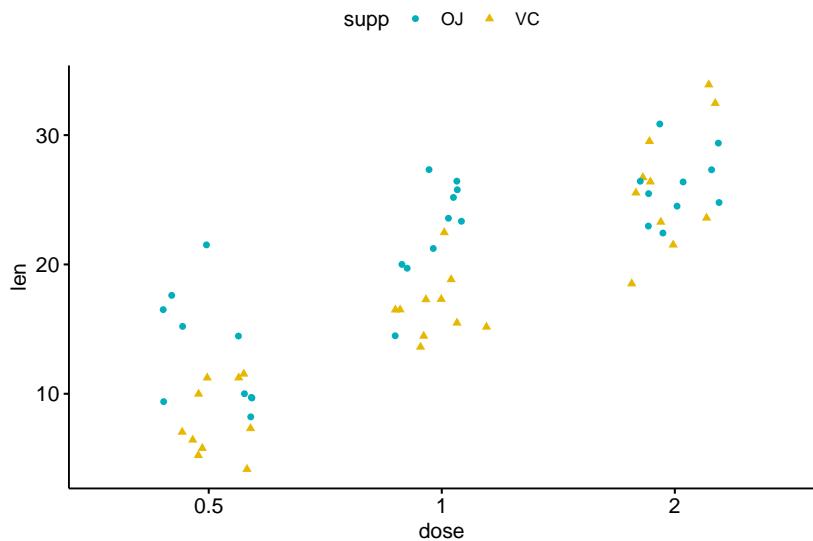
```
# Add violin + mean_sd
ggstripchart(df, x = "dose", y = "len",
shape = "dose", add = c("violin", "mean_sd"))
```



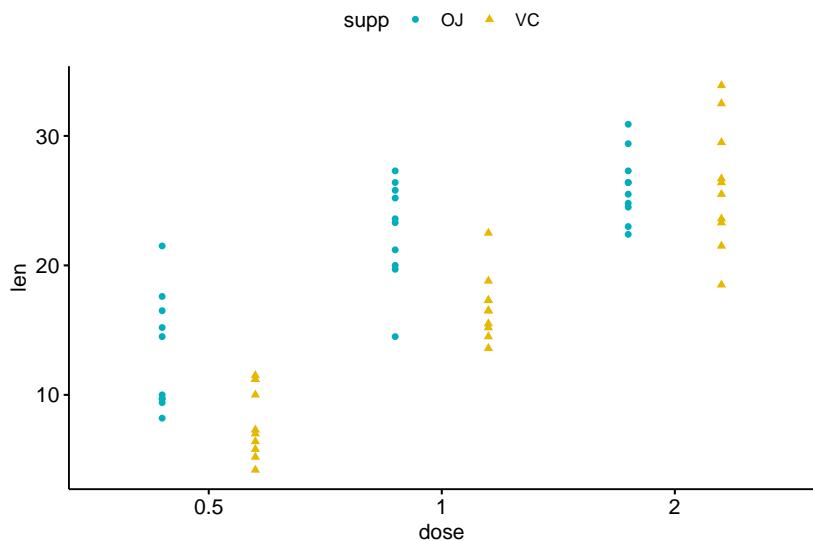
```
# Change colors
# ++++++
# Change colors by groups: dose
# Use custom color palette
ggstripchart(df, "dose", "len", shape = "dose",
  color = "dose", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "mean_sd")
```



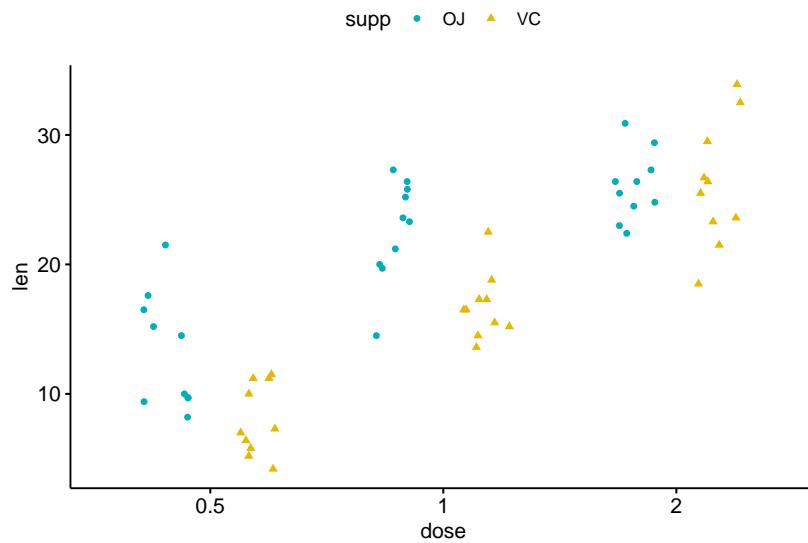
```
# Plot with multiple groups
# ++++++
# Change shape and color by a second group : "supp"
ggstripchart(df, "dose", "len", shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"))
```



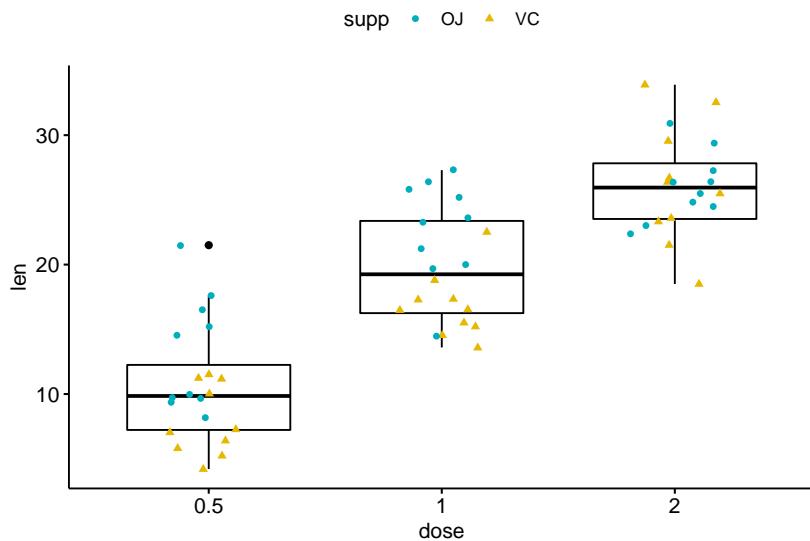
```
# Adjust point position
ggstripchart(df, "dose", "len", shape = "supp",
             color = "supp", palette = c("#00AFBB", "#E7B800"),
             position = position_dodge(0.8) )
```



```
# You can also use position_jitterdodge()
# but fill aesthetic is required
ggstripchart(df, "dose", "len", shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"),
  position = position_jitterdodge() )
```



```
# Add boxplot
ggstripchart(df, "dose", "len", shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"),
  add = "boxplot", add.params = list(color = "black") )
```



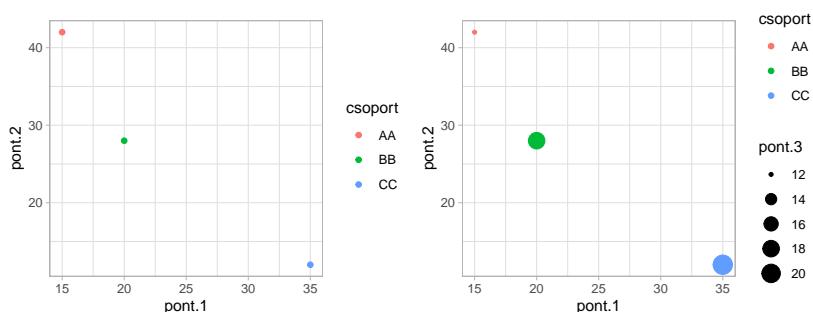
Az eddigiek alapján világos, hogy a megjelenési paraméterek

| Ábra elemek (geom_*) függvény) | Kötelező megjelenési paraméter       | Opcionális megjelenési paraméter                         |
|--------------------------------|--------------------------------------|--|
| geom_abline()                  | slope, intercept                     | alpha, color, linetype, size                             |
| geom_hline()                   | yintercept                           | alpha, color, linetype, size                             |
| geom_vline()                   | xintercept                           | alpha, color, linetype, size                             |
| geom_area()                    | x, ymin, ymax                        | alpha, colour, fill, group, linetype, size               |
| geom_col()                     | x, y                                 | alpha, colour, fill, group, linetype, size               |
| geom_bar()                     | x, y                                 | alpha, colour, fill, group, linetype, size               |
| geom_boxplot()                 | x, lower, middle, upper, ymax, ymin) | alpha, color, fill, group, linetype, shape, size, weight |
| geom_density()                 | x, y                                 | alpha, color, fill, group, linetype, size, weight        |
| geom_dotplot()                 | x, y                                 | alpha, color, fill, group, linetype, stroke              |
| geom_histogram()               | x                                    | alpha, color, fill, linetype, size, weight               |
| geom_jitter()                  | x, y                                 | alpha, color, fill, shape, size                          |
| geom_line()                    | x, y                                 | alpha, color, linetype, size                             |

| Ábra elemek (geom_*) függvény) | Kötelező megjelenési paraméter | Opcionális megjelenési paraméter                                      |
|--------------------------------|--------------------------------|---|
| geom_point()                   | x, y                           | alpha, color, fill, shape, size                                       |
| geom_ribbon()                  | x, ymax, ymin                  | alpha, color, fill, linetype, size                                    |
| geom_smooth()                  | x, y                           | alpha, color, fill, linetype, size, weight                            |
| geom_text()                    | label, x, y                    | alpha, angle, color, family, fontface, hjust, lineheight, size, vjust |

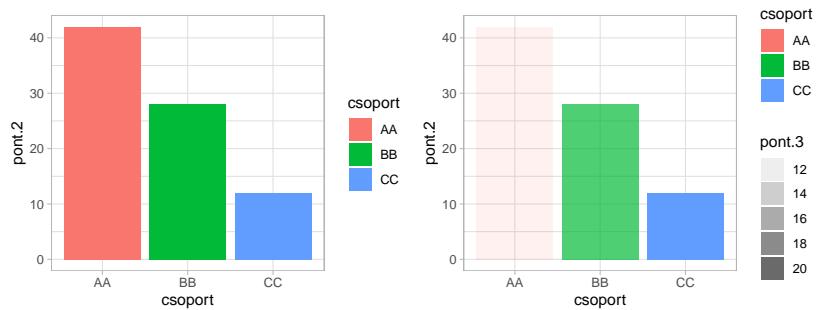
Más megjelenési paraméterekre is lehetünk hatással. Ha a szín és a pontméret jellemzőket is be akarjuk állítani, akkor az `aes()` függvényben a `colour=` és a `size=` argumentumoknak az adattábla egy-egy oszlopát kell megadnunk. Ilyen esetben jelmagyarázat is megjelenik, amelynek az alapértelmezett pozícióját most expliciten is kiírtuk: `theme(legend.position = "right")`.

```
library(ggplot2)
library(gridExtra)
p1 <- ggplot(d.tbl, aes(x=pont.1, y=pont.2, colour=csoport)) + geom_point() + theme(legend.position = "right")
p2 <- ggplot(d.tbl, aes(x=pont.1, y=pont.2, colour=csoport, size=pont.3)) + geom_point() + theme(legend.position = "right")
grid.arrange(p1, p2, ncol=2)
```



Oszlopdiagram esetén használhatjuk a `fill=` vagy az `alpha=` argumentumokat, amelyekkel az egyes oszlopok kitöltőszínét és átlátszóságát tudjuk beállítani:

```
p1 <- ggplot(d.tbl, aes(x=csoport, y=pont.2, fill=csoport)) + geom_col()
p2 <- ggplot(d.tbl, aes(x=csoport, y=pont.2, fill=csoport, alpha=pont.3)) + geom_col()
grid.arrange(p1, p2, ncol=2)
```



Néhány ábraelem esetében nem ilyen egyértelmű az `aes()` függvényben megadott oszlopvektorok és a ábraelem megjelenési paramétere közötti összefüggés.

Tekintsük például a dobozdiagrammot, amelyben a fenti táblázat szerint a `geom_boxplot()` ábraelem ígyenli az `x`, `lower`, `middle`, `upper`, `ymax`, `ymin` megjelenési paraméterek beállítását. Azonban a lenti árákat sokkal egyszerűbb `aes()` paraméterezéssel hoztuk létre, szó sincs benne a fent felsoroltak megadásáról.

```

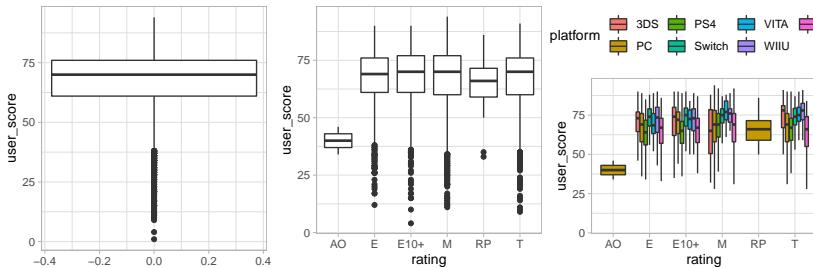
data(survey, package = "MASS")
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/metacritic_games.csv"

d.tbl <- read_delim(file = data.file, delim = ",")
d.tbl <- d.tbl %>% select(game, platform, genre, rating, metascore, user_score)
d.tbl

#> # A tibble: 5,699 x 6
#>   game                  platform genre rating metascore user_score
#>   <chr>                 <chr>   <chr> <chr>    <dbl>      <dbl>
#> 1 Portal 2              PC      Acti~ E10+     95       90
#> 2 The Elder Scrolls V: Skyrim PC      Role~ M        94       82
#> 3 The Legend of Zelda: Ocarina of T~ 3DS     Misc~ E10+     94       90
#> 4 Batman: Arkham City      PC      Acti~ T        91       87
#> 5 Super Mario 3D Land      3DS     Acti~ E        90       84
#> 6 Deus Ex: Human Revolution PC      Acti~ M        90       85
#> 7 Pushmo                  3DS     Misc~ E        90       83
#> 8 Total War: Shogun 2      PC      Stra~ T        90       84
#> 9 FIFA Soccer 12           PC      Spor~ E        89       71
#> 10 Battlefield 3           PC     Acti~ M        89       76
#> # ... with 5,689 more rows

p1 <- ggplot(d.tbl, aes(y=user_score)) + geom_boxplot()
p2 <- ggplot(na.omit(d.tbl), aes(x=rating, y=user_score)) + geom_boxplot()
p3 <- ggplot(na.omit(d.tbl), aes(x=rating, y=user_score, fill=platform)) + geom_boxplot(outlier.shape = NA)
grid.arrange(p1, p2, p3, ncol=3)

```



A kötelező megjelenési paramétereket a `geom_boxplot()` által meghívott `stat_boxplot()` függvény biztosítja, így nekünk ezekről nem kell gondoskodni, elegendő egyetlen numerikus vektort (`y=user_score`) vagy egy kategorikus és egy numerikus vektort (`x=rating, y=user_score`) megadni az `aes()` függvényben.

Léteznek tehát olyan grafikus elemek, amelyek változtatás nélkül képesek a bemenő adatok alapján a megjelenési paramétereket beállítani, és léteznek olyanok is, amelyek a bemenő adatokat áttranszformálják egy köztes adattáblába, és a megjelenési paraméterek konkrét értékét innen veszik. A következő táblázat ezeket a statisztikai transzformációkat foglalja össze, csak azokat az ábraelemeket soroljuk fel, ahol történik transzformáció:

| Ábraelem                      | Transzformáció               | Létrehozott változók   |
|-------------------------------|------------------------------|--|
| <code>geom_boxplot()</code>   | <code>stat_boxplot()</code>  | <code>width, ymin, lower, notchlower, middle, notchupper, upper, ymax</code> |
| <code>geom_bar()</code>       | <code>stat_count()</code>    | <code>count, prop</code>   |
| <code>geom_density()</code>   | <code>stat_density()</code>  | <code>density, count, scaled, ndensity</code>                                |
| <code>geom_histogram()</code> | <code>stat_bin()</code>      | <code>count, density, ncount, ndensity</code>                                |
| és                            |                              |  |
| <code>geom_freqpoly</code>    |                              |  |
| <code>geom_violin()</code>    | <code>stat_ydensity()</code> | <code>density, scaled, count, violinwidth, n, width</code>                   |
| <code>geom_smooth()</code>    | <code>stat_smooth()</code>   | <code>y, ymin, ymax, se</code>   |

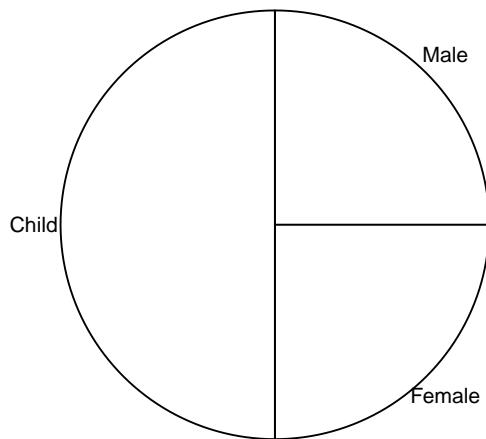
```
# egyezes <- d.tbl$platform
# tbl <- data.frame(egyezes=egyezes)
# ggplot(tbl, aes(x=egyezes, group=1)) + geom_bar(aes(y = ..count..)) +
#   geom_text(aes(label = scales::percent(..prop..), y= ..prop.. ), stat= "count", vjust = -.5) +
#   scale_y_continuous(labels = scales::percent) +
#   coord_cartesian(ylim = c(0, 1101)) + labs(x="Egyező gének száma", y="Előfordulás")
#
#
# ggplot(d.tbl, aes(x=factor(platform), y=user_score)) + geom_boxplot() +
#   geom_text(aes(label = scales::percent(..lower..)), stat= "boxplot", vjust = -.5)
```

```
library("ggpubr")
library(ggthemes)
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)
head(df[, c("wt", "mpg", "cyl")], 3)
#>          wt  mpg cyl
#> Mazda RX4   2.620 21.0   6
#> Mazda RX4 Wag 2.875 21.0   6
#> Datsun 710   2.320 22.8   4

# ++++++
df <- data.frame(
  group = c("Male", "Female", "Child"),
  value = c(25, 25, 50))

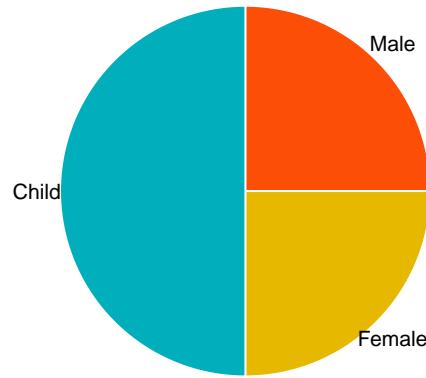
head(df)
#>   group value
#> 1   Male    25
#> 2 Female    25
#> 3   Child    50

# Basic pie charts
# ++++++
ggpie(df, "value", label = "group")
```



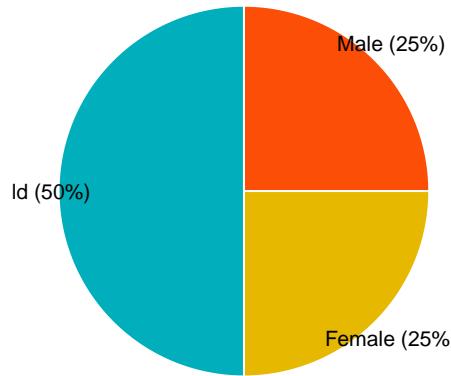
```
# Change color
# ++++++
# Change fill color by group
# set line color to white
# Use custom color palette
ggpie(df, "value", label = "group",
      fill = "group", color = "white",
      palette = c("#00AFBB", "#E7B800", "#FC4E07") )
```

group Child Female Male



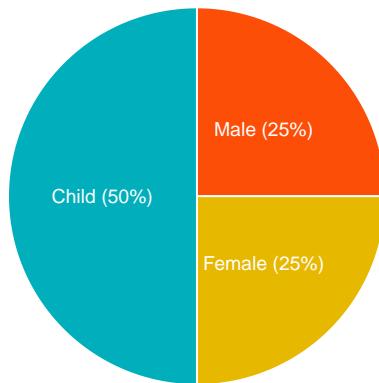
```
# Change label
# ++++++
# Show group names and value as labels
labs <- paste0(df$group, " (", df$value, "%)")
ggpie(df, "value", label = labs,
      fill = "group", color = "white",
      palette = c("#00AFBB", "#E7B800", "#FC4E07"))
```

group Child Female Male



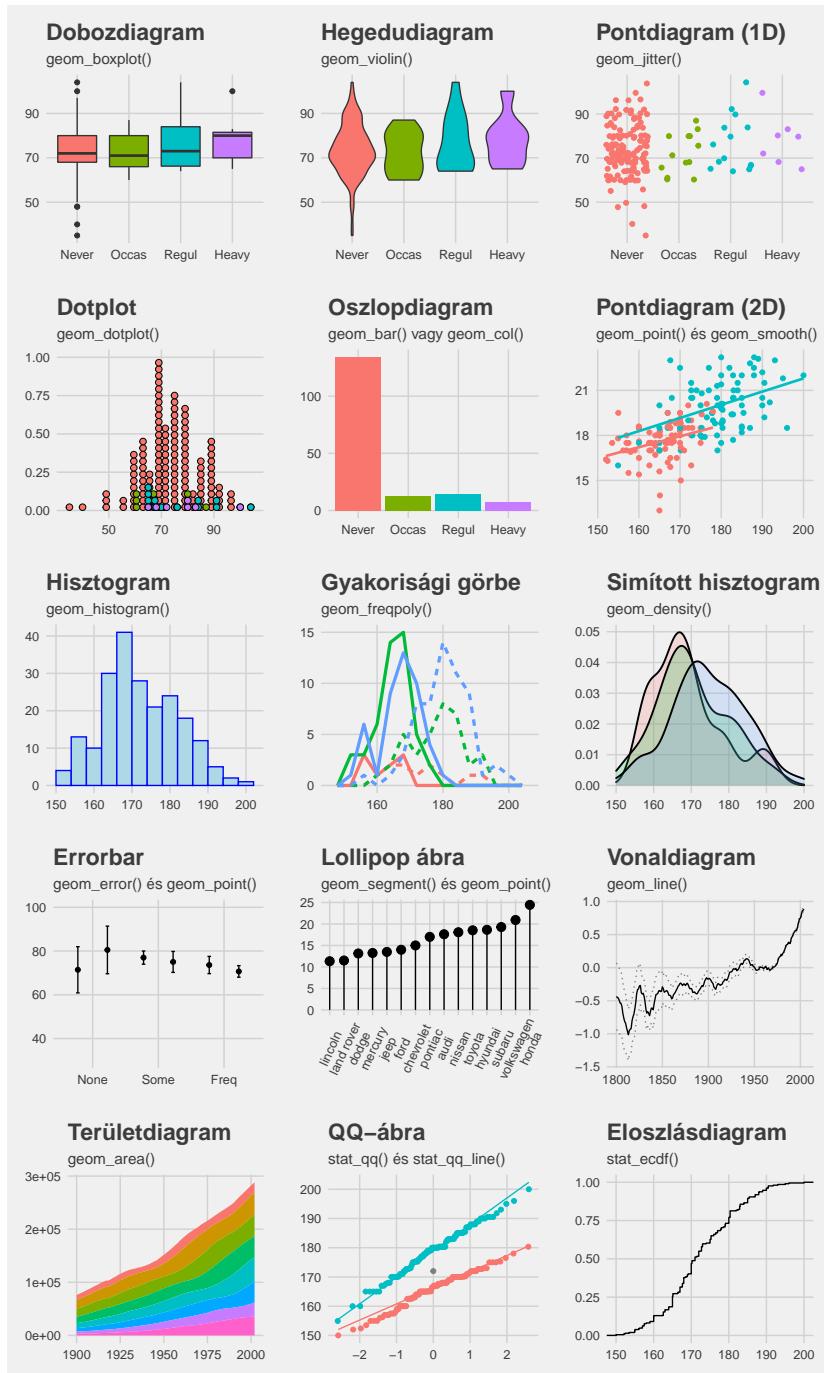
```
# Change the position and font color of labels
ggpie(df, "value", label = labs,
       lab.pos = "in", lab.font = "white",
       fill = "group", color = "white",
       palette = c("#00AFBB", "#E7B800", "#FC4E07"))
```

group Child Female Male



A következő példákhoz a következő csomagok betöltésére van szükség:

```
library(tidyverse)
library(ggplot2)
library(gridExtra)
library(ggthemes)
library(RColorBrewer)
```



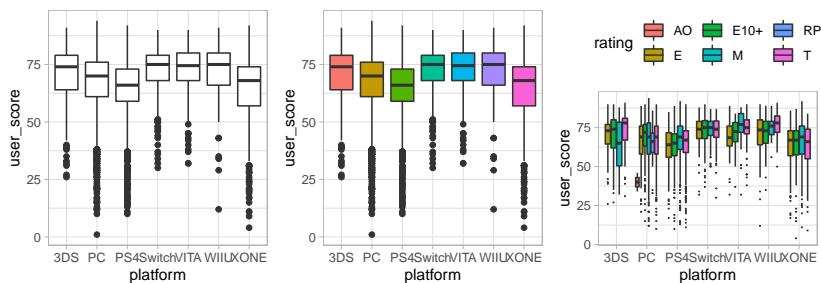
### 9.1.1. Dobozdiagram

```
data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/metacritic_games.csv"

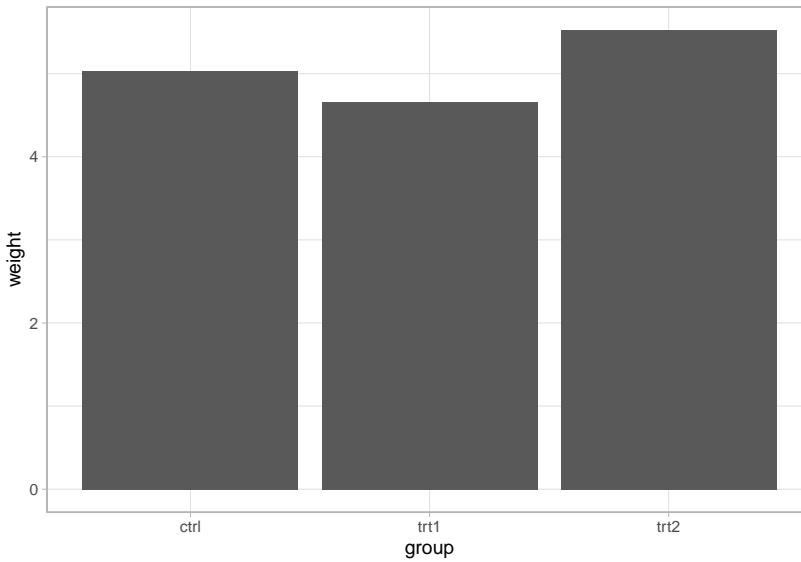
d.tbl <- read_delim(file = data.file, delim = ",")
d.tbl <- d.tbl %>% select(game, platform, genre, rating, metascore, user_score)
d.tbl

#> # A tibble: 5,699 x 6
#>   game          platform genre rating metascore user_score
#>   <chr>        <chr>    <chr>  <chr>    <dbl>      <dbl>
#> 1 Portal       PC       Acti~ E10+     95       90
#> 2 The Elder Scrolls V: Skyrim PC       Role~ M        94       82
#> 3 The Legend of Zelda: Ocarina of T~ 3DS      Misc~ E10+     94       90
#> 4 Batman: Arkham City   PC       Acti~ T        91       87
#> 5 Super Mario 3D Land   3DS      Acti~ E        90       84
#> 6 Deus Ex: Human Revolution PC       Acti~ M        90       85
#> 7 Pushmo        3DS      Misc~ E        90       83
#> 8 Total War: Shogun 2   PC       Stra~ T        90       84
#> 9 FIFA Soccer 12       PC       Spor~ E        89       71
#> 10 Battlefield 3      PC       Acti~ M        89       76
#> # ... with 5,689 more rows

p1 <- ggplot(d.tbl, aes(x=platform, y=user_score)) + geom_boxplot()
p2 <- ggplot(d.tbl, aes(x=platform, y=user_score, fill=platform)) + geom_boxplot() + theme(legend.position =
p3 <- ggplot(na.omit(d.tbl), aes(x=platform, y=user_score, fill=rating)) + geom_boxplot(outlier.size = 0) +
grid.arrange(p1, p2, p3, ncol=3)
```



```
library(gcookbook) # Load gcookbook for the pg_mean data set
ggplot(pg_mean, aes(x = group, y = weight)) +
  geom_col()
```



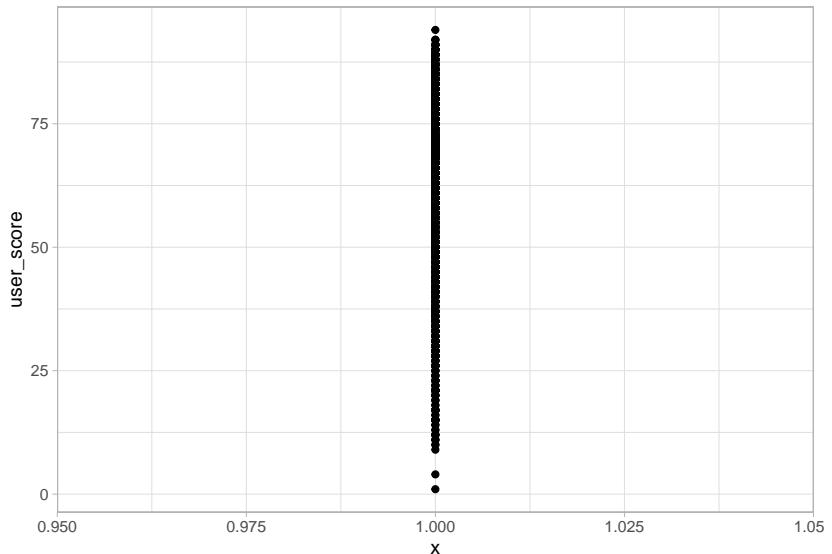
```

data.file <- "https://github.com/abarik/rdata/raw/master/r_alapok/metacritic_games.csv"

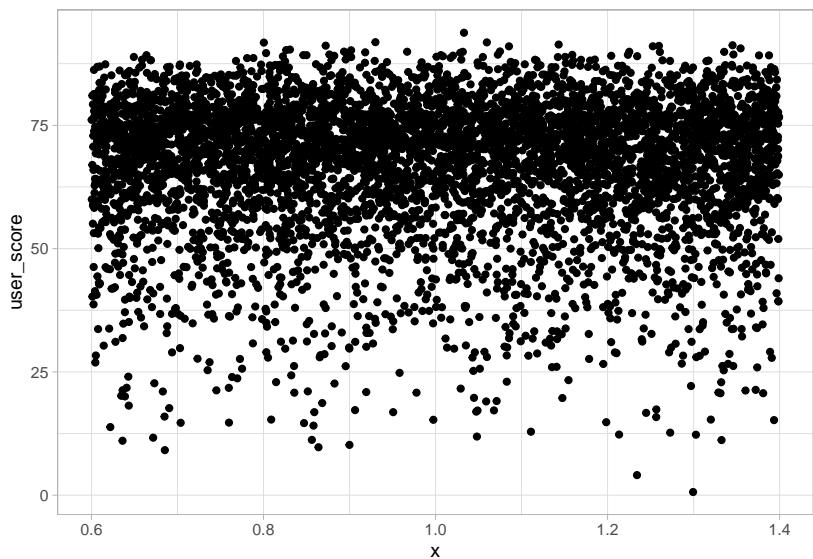
d.tbl <- read_delim(file = data.file, delim = ",")
d.tbl <- d.tbl %>% select(game, platform, genre, metascore, user_score)
d.tbl
#> # A tibble: 5,699 x 5
#>   game                  platform genre  metascore user_score
#>   <chr>                 <chr>   <chr>    <dbl>     <dbl>
#> 1 Portal 2              PC      Action     95       90
#> 2 The Elder Scrolls V: Skyrim PC      Role-P~    94       82
#> 3 The Legend of Zelda: Ocarina of Time 3D 3DS    Miscel~    94       90
#> 4 Batman: Arkham City      PC      Action~    91       87
#> 5 Super Mario 3D Land     3DS    Action     90       84
#> 6 Deus Ex: Human Revolution PC      Action     90       85
#> 7 Pushmo                  3DS    Miscel~    90       83
#> 8 Total War: Shogun 2     PC      Strate~    90       84
#> 9 FIFA Soccer 12          PC      Sports     89       71
#> 10 Battlefield 3          PC      Action     89       76
#> # ... with 5,689 more rows

ggplot(data=d.tbl, aes(x=1, y=user_score)) + geom_point()

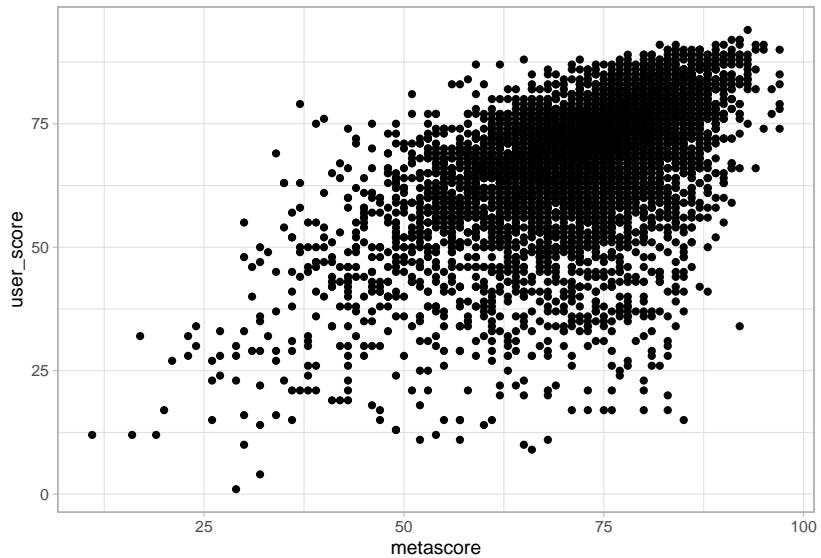
```



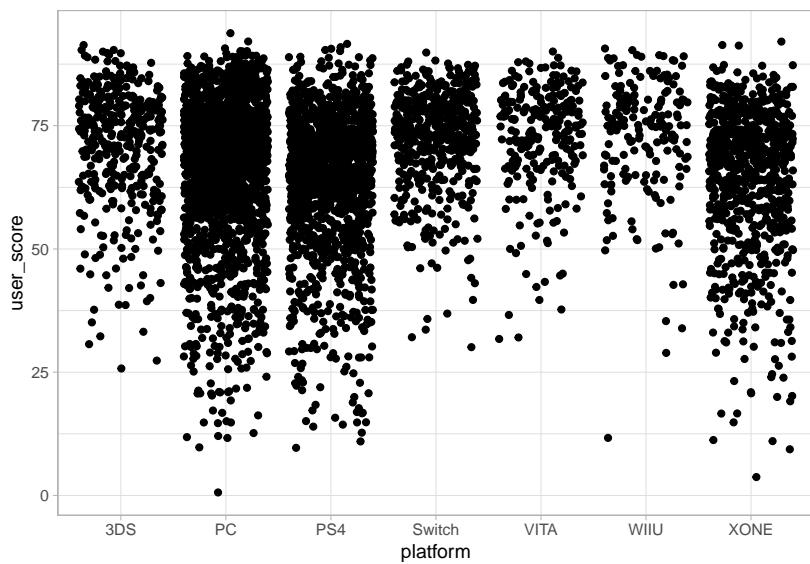
```
ggplot(data=d.tbl, aes(x=1, y=user_score)) + geom_jitter()
```



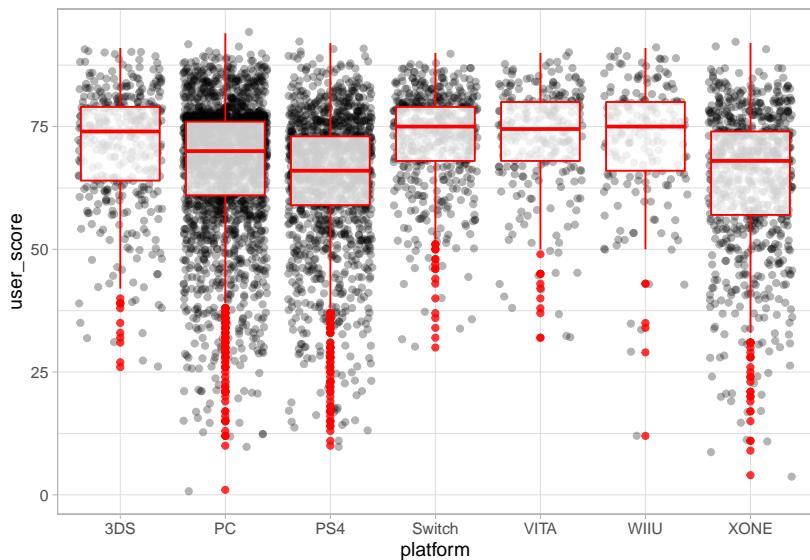
```
ggplot(data=d.tbl, aes(x=metascore, y=user_score)) + geom_point()
```



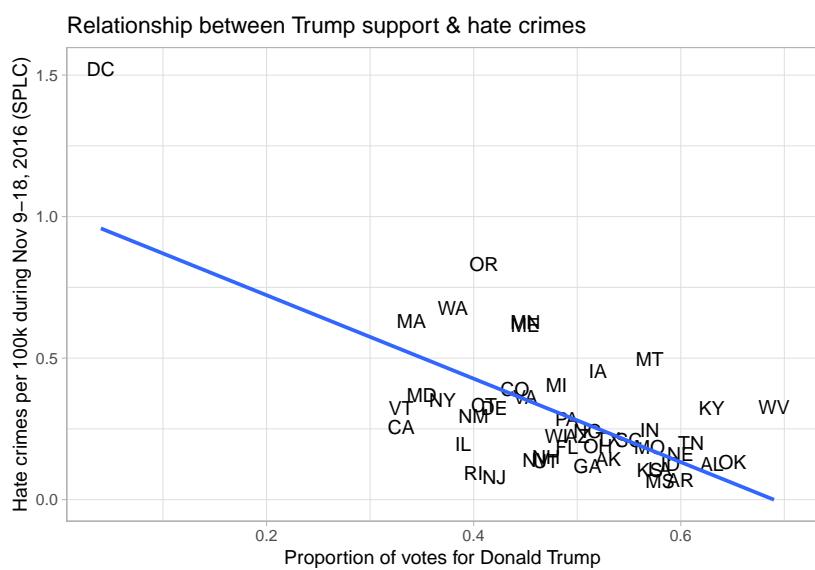
```
ggplot(data=d.tbl, aes(x=platform, y=user_score)) + geom_jitter()
```



```
ggplot(data=d.tbl, aes(x=platform, y=user_score)) + geom_jitter(alpha=0.3) + geom_boxplot(alpha=0.8, col="red")
```



```
library(fivethirtyeight)
ggplot(hate_crimes, aes(x = share_vote_trump, y = hate_crimes_per_100k_splc)) +
  geom_text(aes(label = state_abbrev)) +
  geom_smooth(se = FALSE, method = "lm") +
  labs(x = "Proportion of votes for Donald Trump",
       y = "Hate crimes per 100k during Nov 9-18, 2016 (SPLC)",
       title = "Relationship between Trump support & hate crimes")
```



## 1. feladat. Hisztogram.

Rajzolunk hisztogramot a MASS csomag survey adattáblájának Height oszlopára! Vessük össze a normális eloszlás sűrűségfüggvényével is!

### 9.1.2. Hisztogram rajzolása

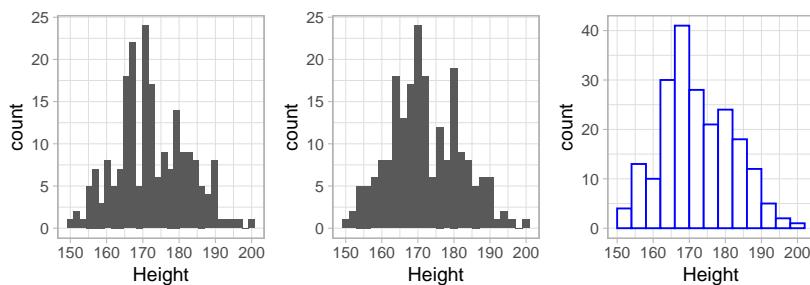
```
data(survey, package = "MASS") # a survey beolvasása

# p1 - alapértelmezett hisztogram
p1 <- ggplot(data=survey, aes(x=Height)) + geom_histogram()

# p2 - hisztogram: binwidth=2
p2 <- ggplot(data=survey, aes(x=Height)) + geom_histogram(binwidth=2)

# p3 - hisztogram: binwidth=4 és színek beállítása
p3 <- ggplot(data=survey, aes(x=Height)) +
  geom_histogram( binwidth=4, colour = "blue", fill = "white")

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)
```



### 9.1.3. Gyakorisági poligon, simított hisztogram és összevetés a normális eloszlás sűrűségfüggvényével

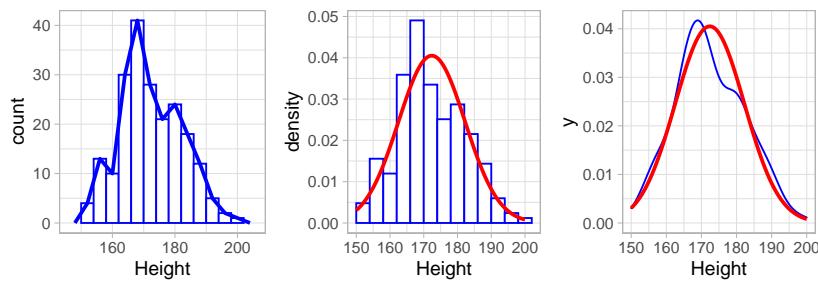
```
data(survey, package = "MASS") # a survey beolvasása

# p1 - hisztogram és gyakorisági poligon
p1 <- ggplot(data=survey, aes(x=Height)) +
  geom_histogram(colour = "blue", fill = "white", binwidth=4) +
  geom_freqpoly(binwidth = 4, size=1, colour="blue")
```

```
# p2 - hisztogram és a normális eloszlás sűrűségfüggvénye
p2 <- ggplot(data=survey, aes(x=Height)) +
  geom_histogram(aes(y = ..density..), colour="blue", fill="white", binwidth=4) +
  stat_function(fun=dnorm, args = list(mean=mean(survey$Height, na.rm=T),
                                         sd=sd(survey$Height, na.rm = T)),
                colour="red", size=1)

# p3 - simított hisztogram és a normális eloszlás sűrűségfüggvénye
p3 <- ggplot(data=survey, aes(x=Height)) +
  geom_density(colour="blue") +
  stat_function(fun=dnorm, args = list(mean=mean(survey$Height, na.rm=T),
                                         sd=sd(survey$Height, na.rm = T)),
                colour="red", size=1)

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)
```



## 2. feladat. Hisztogram csoportokra.

Rajzolunk hisztogramot a MASS csomag survey adattáblájának Height oszlopára az Exer különböző csoportjaiban!

### 9.1.4. Hisztogram csoportokra

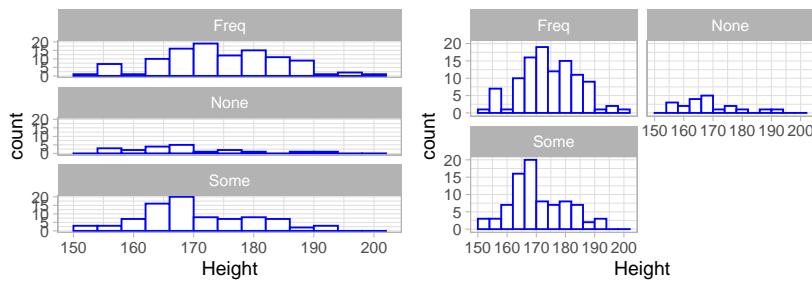
```
data(survey, package = "MASS") # a survey beolvasása

# p1 - hisztogramok egymás alá
p1 <- ggplot(data=survey[!is.na(survey$Exer),], aes(x=Height)) +
  geom_histogram(colour = "blue", fill = "white", binwidth=4) +
  facet_wrap(~ Exer, nrow = 3)

# p2 - hisztogramok táblázatszerűen
p2 <- ggplot(data=survey[!is.na(survey$Exer),], aes(x=Height)) +
  geom_histogram(colour = "blue", fill = "white", binwidth=4) +
```

```
facet_wrap(~ Exer, nrow = 2)

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, ncol=2)
```



### 9.1.5. Gyakorisági poligon és simított hisztogram csoportokra, de egy ábrán

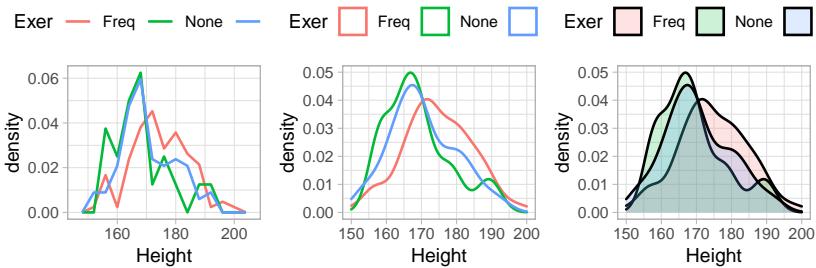
```
data(survey, package = "MASS") # a survey beolvasása

# p1 - gyakorisági poligonok egy ábrán
p1 <- ggplot(data=survey[!is.na(survey$Exer),], aes(x=Height, y=..density.., colour = Exer)) +
  geom_freqpoly(binwidth = 4, size=0.7) + theme(legend.position="top")

# p2 - simított hisztogramok egy ábrán
p2 <- ggplot(data=survey, aes(x=Height, colour = Exer)) + geom_density(size=0.7) +
  theme(legend.position="top")

# p3 - simított hisztogram kitöltéssel egy ábrán
p3 <- ggplot(data=survey, aes(x=Height, fill = Exer)) + geom_density(alpha=0.2, size=0.7) +
  theme(legend.position="top")

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)
```



### 3. feladat. Dobozdiagram.

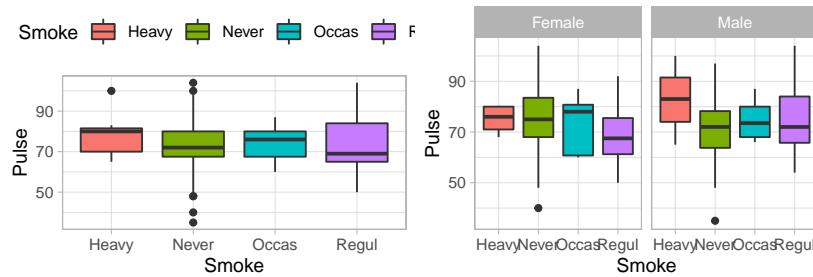
Rajzolunk hisztogramot a MASS csomag survey adattáblájának Pulse oszlopára a Smoke egyes csoportjaiban, valamint vegyük figyelembe a Sex változó értékeit is!

```
data(survey, package = "MASS") # a survey beolvasása

# p1 - dobozdiagram a Smoke csoportjaira
p1 <- ggplot(data=survey[!is.na(survey$Smoke),], aes(x=Smoke, y=Pulse, fill=Smoke)) +
  geom_boxplot() + theme(legend.position="top")

# p2 - dobozdiagram a Smoke csoportjaira a Sex figyelembevételével
p2 <- ggplot(data=survey[!is.na(survey$Smoke) & !is.na(survey$Sex),],
  aes(x=Smoke, y=Pulse, fill=Smoke)) +
  geom_boxplot() + guides(fill=FALSE) + facet_wrap(~ Sex, nrow = 1)

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, ncol=2)
```



### 4. feladat. Átlagok ábrázolása.

Rajzolunk hisztogramot a MASS csomag survey adattáblájának Height oszlopára!

### 9.1.6. Átlagok egy faktor esetén

```

data(survey, package = "MASS") # a survey beolvasása

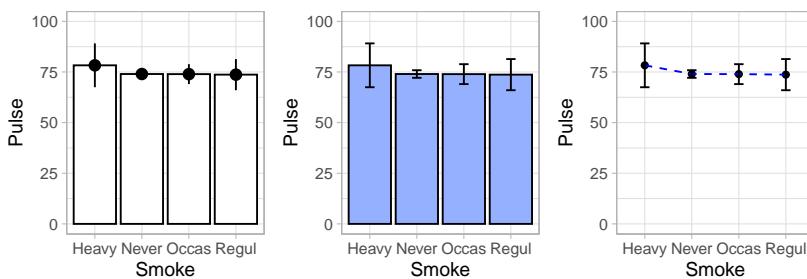
# p1 - oszlopdiagram az átlagokra 95%-os konfidencia intervallummal I.
p1 <- ggplot(data=survey[!is.na(survey$Smoke),], aes(x=Smoke, y=Pulse)) +
  stat_summary(fun.y=mean, geom="bar", fill="white", colour="black") +
  stat_summary(fun.data=mean_cl_normal, geom="pointrange") +
  coord_cartesian(ylim = c(0, 100))

# p2 - oszlopdiagram az átlagokra 95%-os konfidencia intervallummal II.
p2 <- ggplot(data=survey[!is.na(survey$Smoke),], aes(x=Smoke, y=Pulse)) +
  stat_summary(fun.y=mean, geom="bar", fill="#95b0ff", colour="black") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  coord_cartesian(ylim = c(0, 100))

# p3 - vonaldiagram az átlagokra 95%-os konfidencia intervallummal
p3 <- ggplot(data=survey[!is.na(survey$Smoke),], aes(x=Smoke, y=Pulse)) +
  stat_summary(fun.y=mean, geom="point") +
  stat_summary(fun.y=mean, geom="line", aes(group=1), colour="blue", linetype="dashed") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  coord_cartesian(ylim = c(0, 100))

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)

```



### 9.1.7. Átlagok két faktor esetén

```

data(survey, package = "MASS") # a survey beolvasása

# p1 - oszlopdiagram az átlagokra 95%-os konfidencia intervallummal I.
p1 <- ggplot(data=survey[!is.na(survey$Exer) & !is.na(survey$Sex),],
  aes(x=Exer, y=Pulse, fill=Sex)) +

```

```

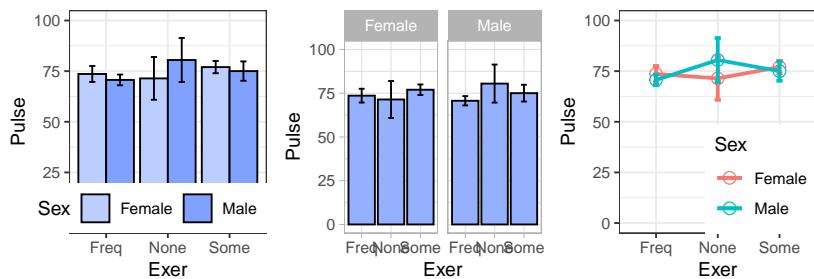
stat_summary(fun.y=mean, geom="bar", position="dodge", colour="black") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar",
  position=position_dodge(width=0.90), width=0.2) +
  scale_fill_manual("Sex", values = c("Female"="#bccdff", "Male"="#81a1ff")) +
  coord_cartesian(ylim = c(0, 100)) + theme_bw() +
  theme(legend.justification=c(1,0), legend.position=c(1,0),
  legend.direction="horizontal")

# p2 - oszlopdiagram az átlagokra 95%-os konfidencia intervallummal II.
p2 <- ggplot(data=survey[!is.na(survey$Exer) & !is.na(survey$Sex),],
  aes(x=Exer, y=Pulse)) +
  stat_summary(fun.y=mean, geom="bar", fill="#95b0ff", colour="black") +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", width=0.2) +
  coord_cartesian(ylim = c(0, 100)) +
  facet_wrap(~ Sex, nrow = 1)

# p3 - vonaldiagram az átlagokra 95%-os konfidencia intervallummal
p3 <- ggplot(data=survey[!is.na(survey$Exer) & !is.na(survey$Sex),],
  aes(x=Exer, y=Pulse, colour=Sex)) +
  stat_summary(fun.y=mean, geom="point", size=3, shape=21, fill="white") +
  stat_summary(fun.data=mean_cl_normal, geom="line", size=1, aes(group=Sex)) +
  stat_summary(fun.data=mean_cl_normal, geom="errorbar", size=1, width=0.1) +
  coord_cartesian(ylim = c(0, 100)) +
  theme_bw() + theme(legend.justification=c(1,0), legend.position=c(1,0))

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)

```



##### 5. feladat. Kétdimenziós pontdiagram.

Rajzolunk kétdimenziós pontdiagramot a MASS csomag survey adattáblája alapján a Height és NW.Hnd változók kapcsolatára. Vegyük figyelembe a Sex változót is!

```

data(survey, package = "MASS") # a survey beolvasása

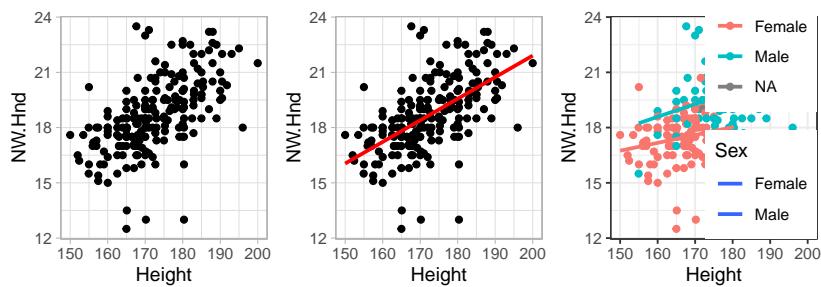
# p1 - kétdimenziós pontdiagram
p1 <- ggplot(data=survey, aes(x=Height, y=NW.Hnd)) + geom_point()

# p2 - kétdimenziós pontdiagram regressziós egyenessel
p2 <- ggplot(data=survey, aes(x=Height, y=NW.Hnd)) +
  geom_point() + geom_smooth(method = "lm", se=F, colour="red")

# p3 - kétdimenziós pontdiagram csoportonkénti regressziós egyenessel
p3 <- ggplot(data=survey, aes(x=Height, y=NW.Hnd, colour=Sex)) +
  geom_point() + geom_smooth(method = "lm", se=F, aes(fill=Sex)) +
  theme_bw() + theme(legend.justification=c(1,0), legend.position=c(1,0))

# a fenti ábrák megjelenítése
grid.arrange(p1, p2, p3, ncol=3)

```



#### 6. feladat. Egydimenziós pontdiagram.

Rajzolunk egydimenziós pontdiagramot a MASS csomag survey adattáblája alapján a Height változóra. Vegyük figyelembe a Sex változót is!

```

data(survey, package = "MASS") # a survey beolvasása

# p1 - egydimenziós pontdiagram
p1 <- ggplot(data = survey, aes(x = Exer, y = Height)) + geom_point()

# p2 - egydimenziós pontdiagram véletlen x elmozdulással
p2 <- ggplot(data = survey, aes(x = Exer, y = Height)) + geom_point(position = "jitter")

# p3 - egydimenziós pontdiagram véletlen x elmozdulással és csoportok jelölése
p3 <- ggplot(data = survey, aes(x = Exer, y = Height)) +
  geom_point(aes(colour=Sex), position = "jitter") +
  theme(legend.position="top")

```