

Reiczigel Jenő – Harnos Andrea – Solymosi Norbert

BIOSTATISZTIKA
nem statisztikusoknak

Tartalomjegyzék

Előszó	1
Köszönetnyilvánítás	4
Hogyan olvassuk ezt a könyvet?	5
Szükséges előismeretek	6
Jelölések, írásmód	7
Ismerkedés az R-rel	8
Hogyan olvassuk az R-kódokat?	9
1. Bevezetés	13
1.1. Miért tanulunk statisztikát?	13
1.2. Megjegyzések a példákhoz	16
1.3. Hétköznapi valószínűségszámítás és statisztika	20
2. A statisztika alapfogalmai	23
2.1. Populáció és minta	23
2.2. Leíró és induktív statisztika	27
2.3. Mintavételi módszerek	29
2.4. Az adatok	33
2.4.1. Adatmátrix	33
2.4.2. Adattípusok, mérési skálák	35
2.4.3. Transzformációk, származtatott változók	39
2.4.4. Hiányzó értékek	45
2.4.5. Kiugró értékek	48
3. Egy kis valószínűségszámítás	51
3.1. Események, valószínűség	51
3.2. Oddsz és logit	56
3.3. Relatív kockázat és esélyhányados	58
3.4. Valószínűségi változók	60
3.4.1. Valószínűségi változók függetlensége	68

3.5.	A statisztikában leggyakrabban használt eloszlások	68
3.5.1.	A hipergeometrikus és a binomiális eloszlás	69
3.5.2.	A Poisson-eloszlás	74
3.5.3.	A normális eloszlás	77
3.5.4.	További folytonos eloszlások	80
3.6.	A valószínűségszámítás és a statisztika kapcsolata	82
4.	Leíró statisztika	87
4.1.	Táblázatok és ábrák	87
4.1.1.	Egy változó ábrázolása	88
4.1.2.	Két változó együttesének ábrázolása	97
4.2.	Mérőszámok, statisztikák	103
4.2.1.	Egy változó jellemzése	104
4.2.2.	Két változó közötti összefüggés jellemzése	115
4.2.3.	Asszociációs mértékek	118
4.2.4.	Adattranszformációk hatása a statisztikai mérőszámokra	121
5.	Becslés	123
5.1.	Alapfogalmak	124
5.1.1.	Pontbecslés	124
5.1.2.	Intervallumbecslés	126
5.1.3.	Matematikai formalizmus	129
5.1.4.	A mintaátlag néhány fontos tulajdonsága	131
5.1.5.	Becslés pontossága	132
5.2.	Pontbecslések jósága	135
5.2.1.	Torzítatlanság	135
5.2.2.	Konziszencia	139
5.3.	Eljárások pontbecslések készítésére	139
5.3.1.	Behelyettesítéses becslés	139
5.3.2.	Maximum likelihood (ML) becslés	140
5.4.	Eljárások konfidencia-intervallumok szerkesztésére	142
5.5.	Több paraméter szimultán becslése	145
5.6.	A szükséges mintaelemszám meghatározása becsléshez	147
6.	Hipotézisvizsgálat	151
6.1.	A statisztikai hipotézisvizsgálat alapgondolata	153
6.1.1.	Az indirekt bizonyítás	154
6.1.2.	A tudomány fejlődése	155
6.1.3.	Nullhipotézis és alternatíva	156
6.1.4.	Döntés a nullhipotézisről	159

6.2.	A hipotézisvizsgálat technikai kérdései	163
6.2.1.	Próbastatisztika	164
6.2.2.	A p -érték meghatározása	166
6.2.3.	Döntés a H_0 -ról p -érték nélkül	170
6.2.4.	Egyszerű és összetett hipotézisek	174
6.2.5.	Próba ereje	177
6.3.	További témaik	182
6.3.1.	Többszörös összehasonlítások	182
6.3.2.	Tesztek és konfidencia-intervallumok	184
6.3.3.	A szükséges mintaelemszám meghatározása	185
6.3.4.	Paraméteres és nemparaméteres eljárások	187
7.	Gyakran használt statisztikai próbák	193
7.1.	Várható értékre (populációátlagokra) vonatkozó próbák	194
7.1.1.	Egy várható érték	194
7.1.2.	Két várható érték, független minták	196
7.1.3.	Két várható érték, párosított minták	200
7.1.4.	Kettőnél több várható érték	202
7.2.	Varianciáakra vonatkozó próbák	202
7.2.1.	Egy variancia	203
7.2.2.	Két variancia, független minták	203
7.2.3.	Kettőnél több variancia, független minták	205
7.3.	Eloszlásokra vonatkozó próbák	206
7.3.1.	Egy eloszlás: illeszkedésvizsgálat	206
7.3.2.	Két változó együttes eloszlása: függetlenségvizsgálat	212
7.3.3.	Két vagy több eloszlás: homogenitásvizsgálat	218
7.4.	Valószínűségekre (populációbeli arányokra) vonatkozó próbák	220
7.4.1.	Egy valószínűség	220
7.4.2.	Két valószínűség, független minták	222
7.4.3.	Két valószínűség, párosított minták	225
7.4.4.	Kettőnél több valószínűség, független minták	226
7.5.	Mediánokra vonatkozó próbák	227
7.5.1.	Egy medián	227
7.5.2.	Két vagy több medián	229
7.6.	Rangsprobák	230
7.6.1.	Wilcoxon-féle előjeles rangpróba	231
7.6.2.	Mann–Whitney-féle U-próba	235
7.6.3.	Kruskal–Wallis-féle H-próba	238
8.	Korrelációszámítás	241

8.1.	A Pearson-féle korrelációs együttható	242
8.1.1.	Hipotézisvizsgálat a Pearson-féle korrelációs együtthatóra vonatkozóan	244
8.2.	Együtthatók monoton, de nem lineáris kapcsolatokra	245
9.	Regressziószámítás	249
9.1.	A regressziószámítás szokásos kérdésfeltevései	250
9.2.	Véletlenség a magyarázó és a függő változóban	251
9.3.	Mikor használunk korreláció-, illetve regressziószámítást? .	252
9.4.	Egyszerű lineáris regresszió: I-es modell	253
9.4.1.	Hipotézisvizsgálatok	255
9.4.2.	A determinációs együttható	257
9.4.3.	Predikció a modellben	258
9.5.	Origón átmenő regresszió	261
9.6.	Egyszerű lineáris regresszió: II-es modell	263
9.6.1.	MA-regresszió	263
9.6.2.	SMA-regresszió	263
9.7.	Többszörös lineáris regresszió	266
9.7.1.	Hipotézisvizsgálatok	269
9.8.	További korrelációs mérőszámok	270
9.8.1.	A többszörös korreláció és a determinációs együttható .	271
9.8.2.	A parciális korreláció	272
9.9.	Multikollinearitás	273
9.10.	Regressziós diagnosztika	276
9.10.1.	Az illesztett modell jóságának vizsgálata	277
9.10.2.	Alkalmazhatósági feltételek vizsgálata	278
9.10.3.	Kiugró értékek és torzító pontok	281
9.10.4.	Diagnosztikus ábrák	289
9.11.	Nemlineáris kapcsolatok	290
9.11.1.	Lineárisra visszavezethető regressziók	292
9.11.2.	Példák változók transzformálásával végzett regressziókra	294
9.11.3.	Lineárisra nem visszavezethető regressziók	301
10.	Varianciaelemzés (ANOVA)	309
10.1.	A számítások	311
10.1.1.	Varianciatábla (szórásfelbontás)	315
10.2.	Csoporthatárak páronkénti összehasonlítása	317
10.3.	Többtényezős varianciaelemzés	320
10.4.	Kísérleti elrendezések	324
10.4.1.	Véletlen blokkos elrendezés	324

10.4.2. Latinnégyzet-elrendezés	326
10.5. Az ANOVA diagnosztikája	328
10.6. Kontrasztok	328
11. Az általános lineáris modell	331
11.1. A fejezet példája	331
11.1.1. A kísérlet rövid leírása	331
11.1.2. Exploratív elemzések	333
11.2. Statisztikai modellek	337
11.3. A modell felírása	339
11.3.1. Példák különböző modellekre	340
11.3.2. Faktorok a lineáris modellben	341
11.4. A lineáris modell paramétereinek becslése	352
11.4.1. A becsült értékek és a vetítő mátrix	353
11.5. Hipotézisvizsgálat	354
11.5.1. A null- és a telített modell	354
11.5.2. Modell és részmodell összehasonlítása	355
11.5.3. Az összes magyarázó változó együttes tesztelése	356
11.5.4. Több változó szimultán tesztelése	358
11.5.5. Megjegyzések a modellek tesztelésével kapcsolatban	358
11.6. A lineáris modellek alkalmazhatóságának feltételei	360
11.6.1. Linearitás	360
11.6.2. Kiugró és torzító pontok	362
11.7. Modellválasztás	363
11.7.1. Mit értsünk a „legjobb” modellen?	364
11.7.2. A legszűkebb modell, amely nem különbözik szignifikánsan a teljes modelltől	365
11.7.3. Információs kritériumok	365
11.8. Modellszelekciós eljárások	367
11.8.1. Egyenkénti beléptetés	367
11.8.2. Egyenkénti kihagyás	368
11.8.3. Váltakozó beléptetés-kihagyás	368
11.9. Mikor használjuk az aov(), és mikor az lm() függvényt?	369
11.9.1. Négyzetösszegtípusok	370
11.10. Többszörös összehasonlítások	371
11.11. Kontrasztok az általános lineáris modellben	374
11.11.1. Kontrasztok (általános lineáris hipotézisek) becslése és tesztelése	375

12. Az R-nyelv és -környezet	385
12.1. Telepítés	385
12.2. RGui	387
12.3. A ConTEXT kódszerkesztő	388
12.3.1. Telepítés	388
12.4. Első lépések az R-rel	389
12.4.1. Függvények	391
12.4.2. Csomagok	392
12.4.3. Súgó	393
12.5. R-munkafolyamat	397
12.6. Adatok olvasása és írása	398
12.6.1. Munkakönyvtár	398
12.6.2. Adatok olvasása	398
12.6.3. Adatok írása	400
12.7. Adattároló objektumok	401
12.7.1. Vektor	401
12.7.2. Mátrix	402
12.7.3. Data frame-ek	404
12.7.4. Lista	405
12.7.5. Hivatkozás az objektumok elemeire	406
Függelék	415
A. Konfidencia-intervallumok képletei	415
A.1. Normális eloszlású változó átlaga	415
A.2. Két normális eloszlású változó átlaga közötti különbség	416
A.3. Normális eloszlású változó varianciája, illetve szórása	419
A.4. Valószínűség (populációbeli arány)	420
A.4.1. Wald-féle intervallum	420
A.5. Két valószínűség különbsége	422
A.6. Relatív kockázat	423
A.7. Esélyhányados	424
B. Statisztikai táblázatok	427
Irodalomjegyzék	437
Példák listája	439
Tárgymutató	443

12. Az R-nyelv és -környezet

Az R olyan programozási nyelv, amely különösen alkalmas statisztikai elemzések elvégzésére és azok eredményeinek grafikus megjelenítésére. A *John Chambers* által elindított S-nyelv GNU verziójaként is tekinthető. Az S-nyelvet az 1970-es években a *Bell Laboratories*-ben fejlesztették interaktív adatelemzés és vizualizáció céljából. Az R-nyelv fejlesztését *Robert Gentleman* és *Ross Ihaka* (Statistics Department of the University of Auckland) kezdte el, forráskódját 1997. közepe óta az *R Development Core Team* módosíthatja. Az R szabad szoftver, ami a LESSER GNU¹ GENERAL PUBLIC LICENSE 2.1.² közreadási feltételek alapján terjeszthető. Az S-nyelvvel való rokonság miatt az S-nyelven, illetve az S-Plus³ környezetben megírt kódok a legtöbb esetben használhatók az R-környezetben is, esetenként azonban módosításokra szorulnak.

Az R-nyelv interpretált nyelv, ami azt jelenti, hogy a programkódokat az R-interpreter értelmezi, majd az eredményeket különféle formában visszaadja. Ahhoz, hogy az R-nyelven megfogalmazott utasítások segítségével elemzéseket végezhessünk, szükségünk van az R-környezetre, aminek a telepítési lépései a következő szakasz tárgyalja.

12.1. Telepítés

Töltsük le a legfrissebb telepítő állományt az R hivatalos oldalához⁴ kapcsolódó The Comprehensive R Archive Network (CRAN) honlapjáról⁵. Erre az oldalra navigálva egy Frequently used pages című táblázatot láthatunk. A táblázat első sorának címe Download and Install R. Itt találhatók meg a különböző operációs rendszerekre készített telepítő készletekre

¹<http://www.gnu.hu/>

²<http://www.gnu.org/copyleft/gpl.html>

³<http://www.insightful.com/>

⁴<http://www.r-project.org/>

⁵<http://cran.r-project.org/>

mutató linkek. A Windows (95 and later) linkről az R for Windows című oldalra jutunk, ahol két alkönyvtárra mutató linket láthatunk, ezek közül válasszuk ki a **base-t**⁶. Ez arra az oldalra vezet minket, amelyen az aktuális, legfrissebb stabil Microsoft Windows-hoz készített R-telepítő található. A könyv írásakor ez a 2.4.1-es verzió volt, így a telepítő neve R-2.4.1-win32.exe. Erre a linkre kattintva megindul a telepítőállomány letöltése. A legfrissebb állomány alatt az old mutatóra kattintva megtaláljuk a korábbi verziókat is, esetenként ezekre is szükség lehet. Ha letöltöttük a telepítőt, akkor azt elindítva egy telepítő varázsló segítségével installálhatjuk az R-t a rendszerünkre. A telepítő futtatása során felmerülő kérdésekben kezdő felhasználó számára tanácsos az alapértelmezéseket elfogadni. Azonban van néhány pont, melyeknél szükség esetén érdemes változtatni. Ezek a következők:

- Az Összetevők kiválasztása ablakban (5. a sorban) látható egy lista, amiben kiválaszthajuk, hogy mely elemeket kívánjuk telepíteni, és melyeket nem. Azonban a lista hosszabb, mint ami az ablakban egyszerre látható, ezért érdemes azt lejjebb gördíteni. Az így megjelenő elemek között található a PDF Reference Manual, ami egyetlen állományban tartalmazza az alaptelepítés főbb csomagjaihoz tartozó függvények leírását. Alapértelmezésben ezt nem telepíti a folyamat, tehát ha szükségünk van erre az állományra, akkor jelöljük be.
- A Startup Options ablakban (6.) határozhatjuk meg azt, hogy az alapértelmezett beállításokat szeretnénk használni (*No (accept defaults)*) a telepített R indításakor, vagy módosítani szeretnénk azokat (*Yes (customized setup)*). Ha az utóbbit választjuk, akkor három további ablakban van módunk a beállításokon módosítani.
- A Display Mode ablakban (7.) beállíthatjuk, hogy az alapértelmezett (*MDI (one big window)*), vagy az (*SDI (separate windows)*) megjelenítési módot kívánjuk-e használni az R grafikus környezetében. Az utóbbi választását javasoljuk, részleteket lásd az RGui leírásánál.

A telepítés további lépéseiben tanácsos elfogadni az alapértelmezett beállításokat. Ha az installáció befejeződött, akkor a rendszerünkön van egy R-környezetünk, ami többek között tartalmazza az interpretort, különböző felhasználói felületeket (Rterm, RGui), az alaptelepítéshez tartozó csomagokat, valamint számos, a felhasználót segítő dokumentációt.

⁶<http://cran.r-project.org/bin/windows/base/>

12.2. RGui

A Microsoft Windows környezetben az RGui az egyik felhasználói felület, amelyen keresztül az R-nyelven megírt kódjainkat eljuttathatjuk az R-interpreterhez, illetve az onnan visszakapott eredményeket szöveges és grafikus formában megjeleníthetjük. Az RGui három ablakot tartalmaz, melyeknek saját, a többtől kisebb-nagyobb mértékben eltérő menürendszerére van, amiben az adott ablakban elérhető, gyakrabban használt funkciók vannak (csoporthoz kötve) felsorolva. A három ablak elnevezése: *R Console*, *R Editor* és *R Graphics*.

Az ablakok vagy SDI (Single Document Interface) vagy MDI (Multiple Document Interface) rendszerben kezelhetők. Az előzőben mindegyik ablak saját menüvel, míg az utóbbi esetében mindegyik ablak közös menüfelülettel rendelkezik. Az MDI esetén a „szülőablak” menürendszerében megjelenő funkciók attól függnek, hogy éppen melyik ablak aktív (a felső kék szegély élénkebb, mint a többinél). A telepítés során beállíthatjuk, hogy melyik rendszert szeretnénk használni, de a későbbiekbén is módosíthatjuk azt.

R Console

Ez az elem tekinthető az RGui központi részének. A konzolon keresztül juttathatunk el R-kódokat a parancsértelmezőhöz. A szöveges eredmények alapértelmezésben szintén a konzolra íródnak ki.

R Editor

Az RGui a 2.0.1-es verziótól kezdve tartalmazza ezt a kódszerkesztő eszközt, ami egy egyszerű szövegszerkesztő, kevés funkcióval.

Előnye, hogy a benne szerkesztett kódból egyes sorokat vagy kijelölt szakaszokat közvetlenül lehet átadni az R-konzolnak futtatásra.

A programozás közben gyakran ejthetünk szintaktikai hibákat. Az R-interpreter jelzi, hogy hiba van a szkriptben, azonban általában nem mutat rá arra, hogy mi a hiba. A hiba megkeresését segíti az olyan kódszerkesztő használata, ami az R-nyelv szintaxisának különböző elemeit (zárójelek, függvénynevek, objektumnevek) jól látható módon elkülnöíti. A kódszerkesztőknek ezt a funkcióját szintaxis-kiemelésnek (*highlighting*) nevezzük. Az R

Editor sajnos nem képes az R-szintaxis kiemelésére, és számos fontos karakter (pl. #, &) csak igen nehézkesen írható be magyar területi és nyelvi beállítás mellett.⁷

Hosszabb kódok szerkesztésére érdemes olyan kódszerkesztőt használni, ami alkalmas az R-szintaxis kiemelésére, mint például a későbbiekben ismertetett ConTEXT szerkesztő.

R Graphics

Ebben az ablakban jelennek meg az ábrák, amelyek a menürendszer segítségével különböző vektor- és pixelgrafikus formátumokban fájlba menthetők.

12.3. A ConTEXT kódszerkesztő

A ConTEXT olyan kódszerkesztő, amelyet programozók munkájának megkönnyítésére fejlesztettek. Használata ingyenes. Igen sok programozási nyelv szintaxisának megfelelő szövegkiemelés (*highlighter*) telepíthető hozzá, közöttük az R-nyelvnek megfelelő is. Előnye, hogy egyszerre több szkriptet is szerkeszthetünk ugyanabban az alkalmazásban, és ezek projektbe is szervezhetők. Hátránya, hogy az RGuival nem tud kommunikálni, így szkriptjeinket be kell másolnunk a konzolba, vagy pedig a `source()` függvényt használhatjuk futtatásukhoz.

12.3.1. Telepítés

A szoftver honlapjáról⁸ töltsük le a telepítőkészletet. Ezt a `download` fülön keresztül elérhető lapról végezhetjük el. Ott található egy DOWNLOAD link, amire rákattintva elindul a letöltés.

A letöltött `ConTEXTsetup.exe` állomány elindítását követően egy telepítési varázsló segítségével végezhető el a program installálása. Tanácsos az alapértelmezések elfogadása. Az alaptelepítésben a kódszerkesztőn nem képes az R-nyelv szintaxisának kiemelésére, ehhez le kell töltenünk és telepítenünk egy kiegészítést.

Ugyanarról az oldalról, ahonnan korábban az alkalmazást letöltöttük, válasszuk ki a `highlighters [Q..T]` elemet. Az ennek következtében megjelenő új oldalon keressük meg a R-Script feliratot. Emellett található egy

⁷Közvetlenül könyvünk elkészülte előtt jelent meg az R 2.5.0-ás verziója, amiben már nem jelent problémát az említett jelek beírása, a magyar területi beállítás mellett sem.

⁸<http://www.context.cx/>

R version 2.4.1 (2006-12-18)

Copyright (C) 2006 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.

Type 'q()' to quit R.

>

12.1. ábra. R-konzol

download gomb, amire rákattintva megindul a `rscript.chl` állomány letöltése. Ha a letöltés befejeződött, az állományt másoljuk be a ConTEXT telepítési könyvtárának **Highlighters** alkönyvtárába. A ConTEXT alapértelmezett telepítési könyvtára a `C:\Program Files\ConTEXT`. Természetesen ha a kód szerkesztő telepítésénél ettől eltérőt adtunk meg, akkor az azon belüli **Highlighters** alkönyvtárba kell másolnunk a `.chl` állományt. Az ezek után újraindított ConTEXT eszköztárában látható legördülő listából többek között kiválaszthatjuk az R-t is. Jelenleg a kiemelés azt foglalja magába, hogy számos függvény nevét felismeri, és azokat más színnel jelöli, valamint a zárójelpárok nyitó és záró elemeit a többitől eltérően színezi. További lehetőség, hogy a kódban kijelölt szakaszokat a **CTRL+SHIFT+C** billentyűkombinációval megjegyzéssé alakíthatjuk, illetve abból aktív kóddá változtathatjuk.

12.4. Első lépések az R-rel

Az RGui elindítása után a konzolon a 12.1. ábrán látható tartalom jelenik meg. Ahogy korábban említettük, az R-nyelven megfogalmazott kódjainkat az R-konzolon keresztül juttatjuk el az R-interpreterhez. A konzolon a sor elején álló > jelzi azt, hogy az R várja az utasításainkat (*prompt*).

Próbáljuk ki egy egyszerű művelet elvégzését az R segítségével. Gépeljük be az **1+2** összeadást, majd nyomjuk meg az ENTER billentyűt! (Az R-konzolban a beírt kódot úgy futtathatjuk le, hogy ENTER-t nyomunk.) Az összeadás eredményeként az alábbiakat láthatjuk a konzolon:

```
> 1 + 2
```

```
[1] 3
```

A konzol első sorában az **> 1+2** a kifejezés, amit futtatni kívántunk, a [1] 3 sor pedig a futtatás eredménye. A szögletes zárójelek között lévő **1-hez** hasonló indexeknek akkor van jelentősége, ha az eredményünk több sorban fér el, ilyenkor minden sor előtt áll egy sorszám, ami az adott sor első elemének sorszámát (indexét) jelzi.

A kódok számos utasítást tartalmazhatnak, amelyek alapvetően *kifejezések* vagy *értékkadások* lehetnek. Az előző példában az **1+2** egy értékkadás nélküli kifejezés volt. Ha egy kifejezést értékkadás nélkül adunk meg mint utasítást, akkor az R kiértékeli és megjeleníti az értékét, de az eredményt nem tudjuk közvetlenül felhasználni további feladatokhoz.

Az értékkadás esetén ugyancsak kiértékeli a kifejezést, de az értékét automatikus megjelenítés nélkül eltárolja egy objektumban. Az értékkadás általános szintaxisa: **objektum <- kifejezés**. Az értékkadás jeleként itt a <- használatos, de használható a = és a -> jel is. Korábban csak az általunk használt jel működött minden esetben.

```
> a <- 1 + 2  
> a
```

```
[1] 3
```

Itt már két utasítást használtunk, az első sor elvégzi az összeadást és az értékkadást, a második sor pedig kiíratja az **a** objektumban tárolt értéket.

Ugyanezt végrehajthatjuk egy sorban is, amennyiben az első sort zárójelek közé tesszük.

```
> (a <- 1 + 2)
```

```
[1] 3
```

Ahogy látható, az összeadás és az értékkadás mellett a kiíratás is megtörtenik a kód lefuttatásával.

Az objektumok létrehozásánál és használatakor figyelembe kell vennünk, hogy az R-nyelv kis- és nagybetűérzékeny, így például az A és az a különböző szimbólumnak számít, és különböző objektumokat jelölhet. Az objektumok elnevezésében használhatunk betűket és számokat, ezek mellett a . és az _ jelet is néhány megkötéssel. A nevek vagy .-tal vagy betűvel kezdődhetnek, ha .-tal kezdődik egy név, a második karakter nem lehet szám. Az ékezetes betűk használata változó sikerű lehet, attól függően, hogy milyen operációs rendszeren, illetve milyen nyelvi beállításokkal működik a rendszerünk. Amennyiben olyan kódot szeretnénk írni, ami más gépeken is biztosan lefuttatható, akkor lehetőség szerint az objektumnevekben érdemes mellőzni az ékezetes betűket.

Fontos megjegyezni, hogy amennyiben egy objektumnak új értéket adunk, akkor annak a korábbi értéke törlődik, és felülíródik az újjal. A korábban létrehozott a objektum tartalmát az alábbi utasítás felülírja:

```
> (a <- 5)
```

```
[1] 5
```

Ha több utasítást adunk meg, az R azokat egymás után értelmezi. Az egyes utasításokat vagy pontosvesszővel, vagy sortöréssel választhatjuk el. Amennyiben az értelmező egy szintaktikailag teljes utasítást talál, akkor azt értelmezi, és az eredményt visszaadja. A pontosvessző minden az utasítás végét jelzi. Ha a bevitt utasítás szintaktikailag nem teljes, és egy új sort kezdünk, az értelmezés nem fut le. Amennyiben interaktív üzemmódban dolgozunk, a prompt az alapértelmezett >-ről +-ra változik.

12.4.1. Függvények

Az R-rendszeren belül az objektumokon *operátorokkal* és *függvényekkel* végezhetünk különböző műveleteket. Az előző példákban láttuk az operátorok (lásd 12.1. táblázat) használatát.

12.1. táblázat. Aritmetikai operátorok

operátor	jelentés	kifejezés	eredmény
+	összeadás	2+3	5
-	kivonás	5-2	3
*	szorzás	5*2	10
/	osztás	10/2	5
[^]	hatvány	2 ³	8

A függvények a `fuggveny.neve(arg1, arg2, ...)` szintaxis szerint épülnek fel. A `fuggveny.neve` határozza meg a függvény nevét, amivel azonosítja a rendszer a meghívandó eljárás(okat). A zárójelek közé foglalt `arg1, arg2, ...` a függvény argumentumait jelenti. Az argumentumok azok az objektumok (adatok, más függvények, kifejezések), amelyekből mint bemenő információkból a függvény létrehozza az eredményt. Általában nem minden argumentumnak kell értéket adnunk, mivel a függvény rendelkezhet alapértelmezett értékekkel. Az alábbi példában a `length()` függvény segítségével megszámolhatjuk, hogy a korábban létrehozott `a` objektum hány elemet tartalmaz:

```
> length(a)
```

```
[1] 1
```

A `length()` függvény egyetlen argumentuma a vizsgálandó objektum neve.

12.4.2. Csomagok

Az R-környezetben a függvények könyvtárakban (*library*), csoportosítva vannak összegyűjtve. Az alaptelepítéssel számos függvény telepítődik az R könyvtárrendszerébe.

További nagyon nagy számú függvény érhető el a CRAN, a Bioconductor, az Omegahat és egyéb tárolókból. A függvények csomagokba (*package*) rendezve érhetők el, a hozzájuk tartozó dokumentációval együtt.

Csomagok telepítése

Az RGui *Packages* menüpont *Select repositories...* elemét kiválasztva a megjelenő ablakban megjelennek azok a tárolók, ahonnan (az interneten keresztül) csomagokat telepíthetünk.

Ezek beállítása után ugyancsak a *Packages* menüből válasszuk ki az *Install package(s)...* pontot. Az ekkor megjelenő *CRAN mirror* listát tartalmazó ablakban jelöljünk ki egy tükröt. Ajánlott a térben legközelebbi kiválasztása. Az *OK* gombra kattintva kis idő múlva megjelenik egy újabb ablak, ami az elérhető csomagok listáját tartalmazza. A kívánt listaelemre kattintva az kék hátterű lesz, ami azt jelenti, hogy telepítésre kiválasztottuk. Ha egyszerre több csomagot is telepíteni kívánunk, akkor a **CTRL** billentyűt lenyomva tartva kell a csomagok nevére kattintanunk. Az *OK* gomb megnyomásával elindul a csomag(ok) letöltése és telepítése.

Csomagok betöltése

Egy csomag telepítése után az ahhoz tartozó függvények rendelkezésünkre állnak, azonban mielőtt használnánk valamelyiket, a `library()` függvénytel be kell töltenünk az adott csomagot:

```
> library(lattice)
```

Ezzel az utasítással betöltjük a `lattice` könyvtárat, ami ugyan az alaptelepítéshez tartozik, de nem töltődik be automatikusan az R indításakor.

12.4.3. Súgó

Azt, hogy egy függvénynek milyen argumentumai vannak, illetve hogyan használható, a `help()` függvény segítségével kérdezhetjük le az R-súgóból.

A `help()` függvény számos argumentummal rendelkezik, azonban az alapértelmezett egyéb argumentumok mellett elegendő megadnunk a lekérdezendő objektum nevét:

```
> help(length)
```

Ekkor az eredmény nem a konzolba íródik ki, hanem Compiled HTML súgó formájában jelenik meg. Rövidebb formában is megadhatjuk az alábbiak szerint:

```
> ?length
```

A `help()` utasítást csak akkor tudjuk használni, ha pontosan ismerjük a keresett függvény nevét. Ha nem helyesen adjuk meg a függvény nevét (mint a következő példában), akkor nem jutunk a várt információhoz. Ha pl. szeretnénk a *t-teszt* alkalmazásával kapcsolatos információkhoz jutni, és nem tudjuk a függvény pontos nevét, megpróbálhatjuk a `help(t-test)` utasítást.

```
> help(t-test)
```

```
No documentation for 't - test' in specified packages and libraries:
you could try 'help.search("t - test")'
```

Az üzenetben közli velünk az R, hogy a *betöltött* csomagok között nem talált ilyen függvényt, és felajánlja, hogy a `help.search()` függvénytel próbáljuk megtalálni azokat a csomagokat, illetve függvényleírásokat, amelyekben ez a szöösszetétel szerepel. Míg az alapbeállításokkal a `help()` csak az aktuálisan *betöltött* csomagok között keres, addig a `help.search()`

az összes *telepített* R-könyvtárban. Amennyiben a `help()` függvényben a `try.all.packages` argumentumot TRUE-ra állítjuk, akkor nemcsak a betöltött, hanem az összes telepített csomagban keres az objektum nevére. Ha éppen nincsen betöltve a telepített csomag az R-be, akkor nem fogja megjeleníteni az objektumhoz tartozó leírást, csak azt adja meg, hogy mely csomag tartalmazza azt. Látható, hogy az általunk megadott **t-test** szöveget az R átalakította **t - test**-é. Most próbáljuk megkeresni a szóban forgó függvényt a **t - test** kulcsszóval.

```
> help.search("t - test")
```

```
No help files found with alias or concept or title matching 't - test'
using fuzzy matching.
```

Sajnos így sem tudtunk meg semmit a *t-teszt* használatáról. Most próbáljuk meg úgy, hogy a kötőjel két végéről a szóközöt elhagyjuk.

```
> help.search("t-test")
```

<code>bartlett.test(stats)</code>	Bartlett Test for Homogeneity of Variances
<code>fisher.test(stats)</code>	Fisher's Exact Test for Count Data
<code>pairwise.t.test(stats)</code>	Pairwise t tests
<code>power.t.test(stats)</code>	Power calculations for one and two sample t tests
<code>t.test(stats)</code>	Student's t-Test

Végre megkaptuk a súgórendszer azon elemeit, amelyek tartalmaznak a megadott keresési feltételhez hasonló karakterláncot. Látható, hogy az eredményként megjelenő listában a sorok az R-objektum nevével kezdődnek, szorosan ezután következik az azt tartalmazó könyvtár neve, majd pedig az R-dokumentáció belüli elnevezése. Ezek közül már ki tudjuk választani azt az elemet, amit kerestünk (*Student's t-Test*), és a `help(t.test)` segítségével kiírathatjuk a dokumentációját.

Az `apropos()` függvénytel a *betöltött* könyvtárak objektumainak *neveiben* kereshetünk karaktereket vagy azok láncolatát. A függvény a telepített, de nem betöltött könyvtárakban nem keres.

```
> apropos("test")
```

[1] "ansari.test"	"bartlett.test"
[3] "binom.test"	"Box.test"
[5] "chisq.test"	"cor.test"
[7] "fisher.test"	"fligner.test"
[9] "friedman.test"	"kruskal.test"

```
[11] "ks.test"                  "mantelhaen.test"
[13] "mauchley.test"            "mauchly.test"
[15] "mcnemar.test"              "mood.test"
[17] "oneway.test"                "pairwise.prop.test"
[19] "pairwise.t.test"            "pairwise.wilcox.test"
[21] "power.anova.test"           "power.prop.test"
[23] "power.t.test"                "PP.test"
[25] "prop.test"                  "prop.trend.test"
[27] "quade.test"                 "shapiro.test"
[29] "t.test"                      "var.test"
[31] "wilcox.test"                 "testVirtual"
[33] "testPlatformEquivalence"
```

Amennyiben csak azokat az objektumokat keressük, amelyek nevének az elején szerepel a keresett karakterlánc, így tehetjük meg:

```
> apropos("^test")
[1] "testVirtual"                  "testPlatformEquivalence"
```

Amennyiben csak azokat az objektumokat keressük, amelyek nevének a végén szerepel a keresett karakterlánc, a következő szerint végezhetjük el:

```
> apropos("^.{1,}.test")
[1] "ansari.test"                  "bartlett.test"
[3] "binom.test"                   "Box.test"
[5] "chisq.test"                   "cor.test"
[7] "fisher.test"                   "fligner.test"
[9] "friedman.test"                 "kruskal.test"
[11] "ks.test"                      "mantelhaen.test"
[13] "mauchley.test"                 "mauchly.test"
[15] "mcnemar.test"                  "mood.test"
[17] "oneway.test"                   "pairwise.prop.test"
[19] "pairwise.t.test"                "pairwise.wilcox.test"
[21] "power.anova.test"              "power.prop.test"
[23] "power.t.test"                  "PP.test"
[25] "prop.test"                     "prop.trend.test"
[27] "quade.test"                    "shapiro.test"
[29] "t.test"                        "var.test"
[31] "wilcox.test"
```

Az `example()` függvény szintén segíthet egyes függvények használatának elsajátításában. Kipróbálhatjuk vele azokat a példákat, amelyeket a szerzők beépítettek az egyes csomagokba. Ez igazán hasznos lehet egyes függvények paramétereinek tanulmányozásában.

```
> example(fisher.test)
```

```
fshr.t> TeaTasting <- matrix(c(3, 1, 1, 3), nr = 2,
  dimnames = list(Guess = c("Milk", "Tea"),
  Truth = c("Milk", "Tea")))

fshr.t> fisher.test(TeaTasting, alternative = "greater")

Fisher's Exact Test for Count Data

data: TeaTasting
p-value = 0.2429
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
0.3135693      Inf
sample estimates:
odds ratio
6.408309
```

Egyes csomagokhoz szkripteket mellékelnek az eljárások bemutatására. Ezek a `demo` függvény segítségével lefuttathatók, és áttekintést nyújtanak a könyvtár alkalmazásának lehetőségeiről. A `demo()` utasítással, argumentum nélkül kilistázhatjuk az alapcsomagokhoz tartozó bemutatókat.

```
> demo()

...
Demos in package 'graphics':

Hershey      Tables of the characters in the
               Hershey vector fonts
Japanese     Tables of the Japanese characters
               in the Hershey vector fonts
graphics     A show of some of R's graphics
               capabilities
image        The image-like graphics builtins
               of R
persp         Extended persp() examples
plotmath      Examples of the use of
               mathematics annotation
...
...
```

Ha az összes telepített csomaghoz tartozó bemutatószkriptet ki szeretnék listázni, akkor a fenti forma helyett a

```
> demo(package = .packages(all.available = TRUE))
```

utasítást használjuk. A listákból kiválasztva egy demót (pl. a `graphics` csomagból az `image` bemutatót), a `demo(image)` utasítással futtathatjuk le.

Az eddigiekből látható, hogy amennyiben valamely függvénytel vagy egyéb objektummal kapcsolatban szeretnénk információhoz jutni, a fenti lehetőségekkel csupán a gépünkre telepített csomagok dokumentációjában tudunk keresgálni. Azonban a legtöbb esetben a csomagoknak csak egy része van telepítve gépünkre, vagyis az R eljárásainak csak töredékéről szerezhetünk információkat. Széles körű internetes keresési lehetőséget kínál a CRAN keresője⁹, segítségével minden függvénnyről, egyéb objektumról begyűjthetők a kívánt információk.

12.5. R-munkafolyamat

Egy R-munkafolyamat (*session*) során a létrehozott objektumok név szerint vannak tárolva. Az `objects()` vagy az `ls()` függvényekkel kiírathatók a konzolra az R-ben aktuálisan tárolt objektumok nevei. A tárolt objektumokat együttesen munkaterületnek (*workspace*) nevezzük. A már feleslegessé vált objektumokat az `rm()` függvénytel távolíthatjuk el úgy, hogy a függvény paramétereként az `objektum(ok)` nevét adjuk meg.

Ahogy a UNIX és Windows terminálokban általános, itt is a függőleges nyilak segítségével tudunk közlekedni az *utasítások történetében*. A már korábban lefuttatott utasítást a felfelé mutató nyíllal hívhatjuk újra, és vagy újrafuttatjuk úgy, ahogy van, vagy pedig javítjuk, és a módosított utasítást futtatjuk le.

A létrehozott objektumokat tárolhatjuk egy következő munkafolyamat számára. minden R-munkafolyamat végén, a kilépéskor az RGui felajánlja a munkaterület mentését. Amennyiben mentjük az objektumainkat, azok egy `.RData`, a munkafolyamatban használt összes utasítás pedig egy `.Rhistory` kiterjesztésű fájlba íródik ki. Amikor újraindítjuk az R-t, a mentett munkaterület betöltődik (az elemzések folytathatósága végett). Emellett az utasítások története is betöltődik. Ez igen zavaró is lehet, mivel gyakori, hogy különböző elemzési munkafolyamatokban is ugyanolyan, egyszerű neveket használunk, és ez automatikus betöltődés esetén adatok felcserélődéséhez vezethet. Ennek kivédése érdekében egyrészt minden elemzést külön könyvtárban tanácsos végezni, másrészt érdemes különböző változóneveket használni.

⁹<http://cran.r-project.org/search.html>

12.6. Adatok olvasása és írása

12.6.1. Munkakönyvtár

Ha különböző fájlokkal (pl.: adat, kép) dolgozunk, sokszor fájlok ból olvasunk, illetve azokba írunk ki adatokat. Ilyenkor meg kell adnunk a használt fájlok elérési útvonalát. Ha az elérési útvonalban több alkönyvtár is előfordul, akkor az út hosszú lehet, és adott esetben többször is meg kell adni, vagyis nehézkes. Az R lehetőséget ad arra, hogy meghatározzuk a *munkakönyvtárat*, amiben dolgozunk. Így elegendő a munkakönyvtáron belüli fájlnevek megadása, a teljes útvonal nélkül. A munkakönyvtár megadására a `setwd()` függvényt használjuk.

```
> setwd("d:/munka")
```

Az út megadásánál a könyvtárak elválasztására a / vagy a \\ jelet kell használni. (A Windows-on elérési utak megadásánál elválasztóként használatos \ jel itt nem használható!) A munkakönyvtárat nem csak kódként adhatjuk meg, hanem az RGui *File* menüjének *Change dir...* pontjára kattintva megjelenő ablakon keresztül is.

Előfordulhat, hogy egyszerre több könyvtárban lévő állományokkal is dolgozunk, ebben az esetben hasznos, ha tudjuk, hogy éppen mi az aktuális munkakönyvtár. Az aktuális munkakönyvtár kiolvasását a `getwd()` függvénytel végezhetjük el.

```
> getwd()
```

```
[1] "d:/munka"
```

12.6.2. Adatok olvasása

Microsoft Excel állományok olvasása

Annak ellenére, hogy a Microsoft Excel adattárolási formátum széles körben elterjedt, az R-alapcsomag jelenleg nem tartalmaz eljárást az ilyen fájlok olvasására. Ezen állományok olvasása többféleképpen is megvalósítható.

ODBC segítségével Az RODBC könyvtár segítségével több módon is olvashatjuk Excel-munkafüzetünket. Az első lépés egy *kapcsolat kialakítása*, ezek lehetőségét mutatják a következő, egyenértékű kódok:

```
> library(RODBC)
> kapcsolat <- odbcConnect('ODBCexcel')
```

```
> kapcsolat <- odbcDriverConnect("DRIVER=Microsoft Excel Driver (*.xls);  
+      DBQ = d:/excel.xls")  
> kapcsolat <- odbcConnectExcel("d:/excel.xls")
```

Mindhárom megoldáshoz szükséges, hogy a *Microsoft Excel Driver*-t telepítsük a számítógépünkön. Az első példában bemutatott megoldáshoz szükséges, hogy mielőtt lefuttatjuk, létrehozzunk egy ODBC-kapcsolatot (a példában `ODBCExcel` elnevezésű). A második és a harmadik megoldás nem igényel ilyen előzetes beállítást. A létrehozott kapcsolatról le lehet kérdezni, hogy milyen táblázatokat tartalmaz.

```
> sqlTables(kapcsolat)
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	
1	d:\\\\excel	<NA>	Munka1\$	SYSTEM TABLE	<NA>
2	d:\\\\excel	<NA>	Munka2\$	SYSTEM TABLE	<NA>
3	d:\\\\excel	<NA>	Munka3\$	SYSTEM TABLE	<NA>

A kialakított *kapcsolaton* keresztül az alábbi két módon is kiolvashatjuk az egyes *munkalapokban* tárolt adatokat.

```
> adat <- sqlQuery(kapcsolat, "select * from [Munka1$]")  
> adat <- sqlFetch(kapcsolat, "Munka1")
```

Mindkét példában a Munka1 nevű munkalap adattartalmát olvastuk ki, és adtuk át az `adat` objektumnak.

Az első példa azt mutatja be, hogy egy SQL-lekérdezés segítségével hogyan olvashatjuk az adott munkalapot. Nagyon fontos, hogy az SQL-kódban a \$-jelnek és a szöglletes zárójeleknek a fenti példában megadott szintaxis szerint jelen kell lennie.

A második megoldás szintaktikailag egyszerűbben adja ugyanazt az eredményt.

Fontos megjegyezni, hogy az ODBC-kapcsolaton keresztül az Excel táblázatok nem módosíthatók, csak olvashatók!

Excel-állomány CSV-formátumba alakítása. Mivel az R több függvény segítségével is képes a *comma separated value* (.csv) állományok olvasására, az Excel-állományok használatának egyik lehetősége az, ha átalakítjuk .csv állománnyá, majd azt olvassuk be az R-be.

Ha a gépünkön fut *Microsoft Excel*, *Open Office* vagy más irodai programcsomag, amelynek van táblázatkezelő alkalmazása, akkor annak segítségével elmenthetjük .csv kiterjesztéssel az adott .xls állományt.

A karakterhatárolt állományok beolvasását leginkább a `read.table()` függvénytel, illetve származékaival valósíthatjuk meg. Az egyes függvények közötti argumentumbeállítási eltéréseket a 12.2. táblázat mutatja.

12.2. táblázat. A `read.table()` függvény változatainak különbsége.

függvény	sep	dec	quote	fill
<code>read.table()</code>	""	.	\"	<code>!blank.lines.skip</code>
<code>read.csv()</code>	,	.	\"	TRUE
<code>read.csv2()</code>	;	,	\"	TRUE
<code>read.delim()</code>	\t	.	\"	TRUE
<code>read.delim2()</code>	\t	,	\"	TRUE

12.6.3. Adatok írása

A `write()` függvény a megadott objektumot (`x`) ASCII állományba írja ki. Általában mátrixokra használatos, amiket érdemes transponálni a kiírás előtt, amit a `t()` függvénytel hajthatunk végre. Ha `data.frame`-re használjuk, előtte alakítsuk mátrixszá (`as.matrix()`).

A `write.table()` függvény segítségével az `x` objektumot (`data.frame`) írhatjuk ki egy fájlba, karakterhatárolt szövegként.

A `save()` függvénytel a megadott objektumokat bináris állományba lehet kiíratni, egy későbbi R-munkafolyamatban való alkalmazásra elmenteni. Az eredményként kapott fájlt a `load()` függvénytel tölthetjük be egy újabb munkafolyamatba.

A `save.image()` az előző függvényhez hasonlóan bináris állományba írja ki az objektumokat, de nemcsak az argumentumként megadottakat, hanem minden objektumot, ami a munkakörnyezetben található. Ugyanezt az eredmény érhetjük el a

```
> save(list = ls(all=TRUE), file = " minden_objektum.RData")
```

utasítással. Amennyiben az R-ból `q("yes")` utasítással lépünk ki, akkor is hasonló mentés történik, de akkor egy `.RData` fájlba íródik ki minden objektumunk. Ez a fájl az R következő indításakor automatikusan be is töltődik! Ha Windows RGui-t használunk, akkor a kilépéskor az R rákérdez, hogy akarjuk-e menteni a munkakörnyezetet, amennyiben jóváhagyjuk, akkor egy, a későbbiekben automatikusan betöltődő `.RData` fájlba menti el a munkakörnyezet objektumait.

A `dput()` segítségével egy R-objektumot tudunk kiírni egy ASCII állományba. Az objektum olvasására használható a `dget()` függvény. A `dump()`

a `list` argumentumban megadott objektumokat egy ASCII fájlba írja ki, amit a `source()` függvény forrásaként lehet használni. Ha a `list` argumentumnak `ls()` értéket adunk, akkor a munkakörnyezet összes objektumát kiírja az `.R` fájlba.

Az R-utasítások outputjai egy ASCII fájlba írhatók ki a `sink()` alkalmazásával. Az utasítás végrehajtása után lefuttatott parancsok eredményeként előállt outputok a terminál helyett az argumentumban megadott fájlba íródnak ki. Az `unlink()` utasítással tudjuk törölni a sink-fájlunkat.

A fenti mentési lehetőségek az objektumokra koncentrálnak, de nem rögzítik a munkafolyamatban használt parancsokat, illetve azok sorrendjét. A `savehistory()` utasítással menthetjük a lefuttatott utasításokat, sorrendjükben egy ASCII fájlba. A mentett parancstörténetet a `loadhistory()` utasítással tölthetjük be egy új R-munkakörnyezetbe.

12.7. Adattároló objektumok

Az R-környezetbe beolvasott vagy ott létrehozott adatainkat különböző típusú objektumokban tárolhatjuk. Az eljárásainkban használt adataink különböző típusuak lehetnek, így pl.: `logical`, `integer`, `double`, `complex`, `character` stb. A következőkben leírunk néhány fontosabb adattároló objektumtípust.

12.7.1. Vektor

A vektor olyan adattároló objektum, amiben adataink (legyenek azok numerikusak, szövegesek vagy logikaiak) *rendezetten*, *egydimenziós* formában tárolhatók. Ugyanazon vektoron belül csak *egyfélé* adattípus tárolható.

```
> (a <- 1:5)

[1] 1 2 3 4 5

Ezzel létrehoztunk egy a vektort, ami egész számokat tartalmaz, egytől ötig. Amennyiben nem ehhez hasonló vektorokat szeretnénk létrehozni, akkor használhatjuk a c() függvényt is.

> (a <- c(9,4,6,7,1,2,5))

[1] 9 4 6 7 1 2 5
```

A függvény argumentumai azok a számok vagy szöveges elemek lesznek, amelyeket a létrejövő objektumban tárolni szeretnénk.

A `vector()` függvénytel is létrehozhatunk vektorokat. Ebben az esetben a `mode` argumentummal meg kell adnunk azt, hogy milyen típusú adatokat fog tartalmazni a vektor. Ha nem adjuk meg a típust, akkor logikai vektort fog létrehozni az R. További megadandó paraméter a `length`, amivel a vektor elemeinek a számát határozzuk meg.

```
> (a <- vector(mode = "numeric", length = 5))

[1] 0 0 0 0 0

> (a <- vector(mode = "logical", length = 5))

[1] FALSE FALSE FALSE FALSE FALSE

> (a <- vector(mode = "character", length = 5))

[1] "" "" "" "" "
```

Látható, hogy a függvény a megadott típusnak megfelelő „üres” vektort hoz létre. Ugyanezt érhetjük el egyetlen argumentum (`length`) megadásával, ha a `numeric()`, a `logical()` vagy a `character()` függvényeket használjuk.

```
> (a <- numeric(length = 5))

[1] 0 0 0 0 0

> (a <- logical(length = 5))

[1] FALSE FALSE FALSE FALSE FALSE

> (a <- character(length = 5))

[1] "" "" "" "" "
```

12.7.2. Mátrix

A mátrixok kétdimenziós adattárolásra alkalmas objektumok, amelyeket alkothatják numerikus, karakter vagy logikai adattípusok. Ugyanazon mátrixon belül csak *egyfélé* típus használható. Mátrixot több függvénytel is létrehozhatunk, ilyen a `matrix()`. A mátrix képzésénél a sorok számát az `nrow` (rövidítve `nr`), az oszlopok számát az `ncol` (rövidítve `nc`) argumentummal adjuk meg. Legalább az egyiket meg kell adnunk!

```
> a <- 1:6
> (m <- matrix(a, nr = 3))
```

```
[,1] [,2]
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

Látható, hogy a mátrix képzésekor az adatforrást *oszlopfolytonosan* tölti be a `matrix()` függvény. Ha a `byrow` argumentumot az alapértelmezett `FALSE` helyett `TRUE`-ra állítjuk, akkor mátrixunk *sorfolytonosan* fog feltölődni.

```
> (m <- matrix(a, nr = 3, byrow = T))
```

```
[,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
```

A `dimnames` argumentum segítségével adhatjuk meg a sorok és az oszlopok nevét, *lista* formában. A sorok és oszlopok nevének utólagos megváltoztatására használhatjuk a `rownames()`, illetve a `colnames()` függvényeket.

Mátrixot az `array()` függvénytel is létrehozhatunk. További lehetőség, hogy egy vektorból hozunk létre mátrixot a `dim()` függvény segítségével:

```
> (a <- 1:6)
```

```
[1] 1 2 3 4 5 6
```

```
> dim(a)
```

```
NULL
```

```
> (dim(a) <- c(3, 2))
```

```
[,1] [,2]
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

12.7.3. Data frame-ek

A data.frame-eket alkothatják numerikus, karakter vagy logikai adattípusok. Egy data.frame-en belül *többféle* típus is használható. A data.frame olyan adattábla, aminek alkotó oszlopai vektorként foghatók fel. Fájlból beolvassott adattáblák eredményei általában ilyen objektumként jelennek meg, de létrehozhatjuk a `data.frame()` függvényvel is. Az adattábla létrehozásakor ügyeljünk arra, hogy az alkotó vektorok egyforma hosszúságúak legyenek. Amennyiben az egyik vektor rövidebb a másiknál, és a hosszabb vektor hossza osztható a rövidebb vektor hosszával, akkor a függvény a rövidebb vektor ismétlésével kipótolja a különbséget.

```
> x <- 1:4
> n <- 10
> M <- c(10, 35)
> y <- 2:4
> (r <- data.frame(x, n))
```

```
x  n
1 1 10
2 2 10
3 3 10
4 4 10
```

```
> (r <- data.frame(x, M))

x  M
1 1 10
2 2 35
3 3 10
4 4 35
```

Ha viszont a hosszabb elemszáma nem osztható a rövidebb elemszámával, akkor hibát generál a függvény.

```
r <- data.frame(x,y)
```

```
Error in data.frame(x, y) : arguments imply differing number of rows: 4, 3
```

Amennyiben az adattábla egy oszlopa nem vektor, hanem faktor, arra is vonatkozik, hogy azonos hosszúságúnak kell lennie a többi oszloppal. Az adattábla beépülő vektorok oszlopok lesznek, melyeknek a neve alapértelmezésben a vektor neve lesz (ezt módosíthatjuk).

```
> (r <- data.frame(oszlop1 = x, oszlop2 = n))
```

	oszlop1	oszlop2
1	1	10
2	2	10
3	3	10
4	4	10

A `rows.names` argumentum segítségével a sorokat is elnevezhetjük, a vektorként adandó meg, és a hosszának meg kell egyeznie a táblázat sorainak számával.

```
> (r <- data.frame(oszlop1 = x, oszlop2 = n, row.names = c("a",
+      "b", "c", "d")))
```

	oszlop1	oszlop2
a	1	10
b	2	10
c	3	10
d	4	10

A mátrixhoz hasonlóan a `data.frame` is rendelkezik `dim` argumentummal.

```
> dim(r)
```

```
[1] 4 2
```

12.7.4. Lista

A lista olyan adattároló objektum, ami bármely típusú objektumokból felépülhet. Az adattároló objektumokon (vektorok, listák, mátrixok stb.) kívül tartalmazhatnak függvényeket és kifejezéseket is. Ugyanazon listán belül többféle adattípus használható. A listát a `list()` függvénytel hozhatjuk létre. Általában azt mondhatjuk, hogy semmilyen megkötés nincsen az alkotóelemekkel kapcsolatban. Nem számít, hogy az egyes építőelemek milyen méretűek. Viszont arra érdemes figyelni, hogy a `data.frame`-mel ellentétben az alkotóelemek nevét nem építi be automatikusan a `list()` függvény a listába.

```
> (lista1 <- list(x, y))
```

```
[[1]]
[1] 1 2 3 4
```

```
[[2]]
[1] 2 3 4
```

```
> (lista2 <- list(A = x, B = y))
```

```
$A  
[1] 1 2 3 4
```

```
$B  
[1] 2 3 4
```

```
> names(lista1)
```

```
NULL
```

```
> names(lista2)
```

```
[1] "A" "B"
```

12.7.5. Hivatkozás az objektumok elemeire

Számos esetben van arra szükség, hogy adatainknak egy adott részét külön kezeljük, lekérdezzük, felülírjuk. Ekkor szükségünk lesz arra, hogy adott elemeket azonosítani tudjunk, azokra hivatkozhassunk. Az adattároló objektumok elemeire hivatkozhatunk az elemek *indexének*, illetve *nevének* segítségével.

Indexelés

Az indexelési rendszer nagyon rugalmas és hatékony eszköz az egyes adattároló objektumok elemeinek kiolvasására. Az indexeket az objektum után írt szöglletes zárójellel adjuk meg.

Az indexeléssel kapcsolatban fontos megjegyezni, hogy (ellentétben számos programozási nyelvvel) az R-ben az indexelés *nem 0-ról, hanem 1-ről* indul!

```
> (x <- 1:3)
```

```
[1] 1 2 3
```

Ha az x vektor harmadik elemét szeretnénk kiolvasni, egyszerűen megtehetjük az x[3] utasítással.

```
> x[3]
```

```
[1] 3
```

Ha mátrixból vagy data.frame-ből szeretnénk kiolvasni értékeket, azt két index alkalmazásával lehetjük meg. Az `x` mátrixból egy elemet az `x[i, j]` utasítással olvashatunk ki, ahol `i` a mátrix sorát, `j` pedig az oszlopát jelölő index. Egy egész sor olvasásához az `x[i,]`, egy egész oszlopéhoz pedig az `x[, j]` parancsot használhatjuk.

```
> x <- matrix(1:9, nc = 3)
> x
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	8
[3,]	3	6	9

```
> x[2, 2]
```

```
[1] 5
```

```
> x[2, ]
```

```
[1] 2 5 8
```

```
> x[, 2]
```

```
[1] 4 5 6
```

Ahogy alábbi példákból látható, az objektumból el is távolíthatunk elemeket, sorokat, oszlopokat:

```
> x[-1, ]
```

	[,1]	[,2]	[,3]
[1,]	2	5	8
[2,]	3	6	9

A `x[-1,]` utasítással az `x` mátrix első *sorát* kihagyva kérdeztük le az adatokat.

```
> x[, -1]
```

	[,1]	[,2]
[1,]	4	7
[2,]	5	8
[3,]	6	9

A `x[, -1]` kihagyja az első *oszlopot*, és a többi oszlopot kérdezi le.

```
> x[-1, -1]
```

```
[,1] [,2]  
[1,] 5 8  
[2,] 6 9
```

Itt kihagytuk az első oszlopot és az első sort.

```
> x[-c(1, 3), ]
```

```
[1] 2 5 8
```

Végül kihagytuk a mátrix 1. és a 3. sorát.

Habár nem tartozik szorosan az indexsel való hivatkozás téma körébe, mégis itt mutatjuk be, hogy hogyan gyűjthetünk ki elemeket egy adott objektumban azon az alapon, hogy azok megfelelnek-e valamely feltételnek.

```
> x[x >= 5]
```

```
[1] 5 6 7 8 9
```

A mátrixból azokat az értékeket gyűjti ki, amelyek öttel egyenlők vagy nagyobbak.

```
> which(x >= 5)
```

```
[1] 5 6 7 8 9
```

A feltételnek megfelelő elemek *indexeit* is kigyűjthetjük, látszólag ugyanaz az eredmény, de míg az előző példában az értékeket, itt az indexeket gyűjtöttük ki. Az egyes feltételeknek megfelelő, vagy adott indexű elemeket felül is írhatjuk.

```
> (x[x >= 5] <- 10)
```

```
[,1] [,2] [,3]  
[1,] 1 4 10  
[2,] 2 10 10  
[3,] 3 10 10
```

A data.frame-eken a mátrixokhoz hasonlóan hajthatjuk végre a lekérdezéseket.

A listák esetében az indexek többszintűek lehetnek, álljon itt néhány példa:

```
> x <- matrix(1:9, nc = 3)
> y <- 1:5
> allista <- list(c("a", "b", "c"), c(8, 5, 2, 4, 1, 3))
> (lista <- list(x, y, allista))

[[1]]
 [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

[[2]]
[1] 1 2 3 4 5

[[3]]
[[3]][[1]]
[1] "a" "b" "c"

[[3]][[2]]
[1] 8 5 2 4 1 3
```

A lista *gyökérelemeire* dupla szögletes zárójelek közé zárt indexsel hivatkozhatunk. Az első gyökérelem egy mátrix, amelynek az első oszlopát a következő módon kérdezhetjük le:

```
> lista[[1]][, 1]
```

```
[1] 1 2 3
```

A lista harmadik gyökéreleme egy másik lista A lista második vektorának harmadik elemére a következő módon hivatkozhatunk:

```
> lista[[3]][[2]][3]
```

```
[1] 2
```

Ahogy látható, a listaelemeken belül a vektoroknál és mátrixoknál látott hivatkozást használjuk.

Névvkel való hivatkozás

Az objektumokhoz tartozhatnak nevek, ezeknek több fajtája is lehet (nevek, oszlopnevek, sornevek, dimenziónevek). Ahhoz, hogy nevek segítségével hivatkozzunk elemekre, tudnunk kell, hogy milyen nevek vannak az objektumban. Az objektumban előforduló neveket több módon is kiolvashatjuk, ennek egyik módja a `names()` függvény alkalmazása.

```
> names(lista)
```

NULL

Látható, hogy a korábban létrehozott listánk nem tartalmaz neveket. A névadást megtehetjük az objektum létrehozásakor, de utólag is. Az előbb használt `names()` függvény segítségével értéket is adhatunk az objektumknak. A névadáshoz az objektum méretével megegyező hosszúságú vektort kell használnunk, a fenti példában használt `lista` 3 elemű, tehát egy 3 elemből álló vektorban kell megadnunk a listaelemek neveit.

```
> names(lista) <- c("r", "t", "z")
```

Ha most kiolvassuk a lista elemeinek nevét, a következő eredményt kapjuk:

```
> names(lista)
```

```
[1] "r" "t" "z"
```

Most, hogy a listaelemeknek van már neve, tudunk név szerint hivatkozni rájuk. Az objektum nevét és az elem nevét egy `$` jel választja el:

```
> lista$r
```

```
 [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Ha huzamosabban dolgozunk egy adattároló objektummal, akkor a névvkel való hivatkozás során az objektum nevének és a `$`-jel többszörös begépelése feleslegesnek tűnhet. Ezért lehetőség van arra, hogy az adott objektumra „rákapcsolódhassunk”, és így a munka során az objektum nevét nem kell minden alkalommal megadnunk. Erre szolgál az `attach()` függvény. Az előző példa az `attach()` függvény használatával:

```
> attach(lista)
> r

 [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Egyszerre egy adattároló objektumra kapcsolódhatunk, egy újabbra való kapcsolódás az előzőről való automatikus lekapcsolást is jelent. A lekapcsolásra használhatjuk a `detach()` függvényt is.

Hasonló módon hivatkozhatunk nevek segítségével a `data.frame` objektumok oszlopaira is. Ott az oszlopneveket a `colnames()` függvény segítségével olvashatjuk ki. A `data.frame`-ek esetén szintén használható az `attach()` függvény.

A mátrixok esetén is van lehetőség a névvel való hivatkozásra, azonban annak módja eltér az előzőekben bemutatottaktól.

```
> m <- matrix(1:9,nc=3)
> colnames(m) <- letters[1:3]
> m

  a b c
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

Ha az `a` oszlop adatait szeretnénk lekérdezni, akkor használhatjuk az alábbi eljárást:

```
> m[, 'a']

[1] 1 2 3
```