

Bevezetés az R-nyelv és környezet használatába

Solyosi Norbert

Ezúton szeretnék köszönetet mondani az *R Development Core Team*-nek az R-rendszer fejlesztéséért, illetve a hasznos dokumentációkért és példákért. Az R-rendszert *alkalmazók közösségének* a hasznos levelezési archívumért és egyéb dokumentációkért. Szintén nagy segítséget nyújtottak (a segédlet témaköreinek kiválasztásában) kollégáim: *Harnos Andrea* és *Reiczigel Jenő*, valamint *Pénzes Zsolt* és *Barta Zoltán*. A szöveg nyelvi ellenőrzésében segített *Dala Sára*.

Tartalomjegyzék

Előszó	5
Bevezetés	6
Az R működésének alapelvei	6
Szintaktikai alapok	8
Utasítások	8
Függvények	9
Az adatok tartóssága és az objektumok eltávolítása	10
Szövegszerkesztők	10
Segédletek	11
help	11
apropos	15
example	15
demo	16
Segédletek a CRAN-on	16
Objektumok	19
Attribútumok	19
Objektumok kezelése	20
Adatok begépelése	20
Adatgenerálás	21
Adattároló objektumok	25
Kifejezés	34
Objektumok szerkesztése	35
Objektum-műveletek	38
Adatok olvasása, kezelése és írása	41
Munkakönyvtár	41
Adatok olvasása	41
Adatok kiírása	47
Grafika	52
A grafikai eszköz beállításai	52
Az alapterepítés grafikai függvényei	57
Interaktív vizualizáció	61
Trellis	63
Programozás R-ben	65
Vezérlők	65
Saját függvények készítése	68
Jelentések készítése	71
Sweave	71
Példa	72
Sweave beállítások	72
Objektumok használata a szövegben	76
A kódszakasz újrahasonosítása	76
Tangle vagy weave	77
Az RweaveLatex paraméterezése	77

Függelék	78
Telepítés	79
Windows	79
Linux	79
Szövegszerkesztők	85
Tinn-R	85
Emacs	85
Kate	87
Grafikus felületek	90
Windows RGui	90
R Commander	97
SciViews-R GUI	99
ODBC-kapcsolat létrehozása	111
Microsoft Excel	111
Microsoft Access	111
MySQL	114
PostgreSQL	114
Szoftverintegráció	116
R (D)COM Server	116
RPy	118

Előszó

Ezt a rövid használati útmutatót azért kezdtem el írni, hogy magyarul is elérhető legyen egy az R nyelv alkalmazásába bevezető segédlet. Természetesen a hasonló jellegű munkák, vagyis gyorsan fejlődő nyílt forráskódú rendszerek felhasználását segítő szövegek sohasem lehetnek teljesek. Így minden esetben a teljesség igénye nélkül kezdhet csak bele a szerkesztő a munkába, ami viszont – egy alapos segédletet eléképzelve – folyamatos hiányérzetet okoz. Igen sok angol nyelvű anyag áll rendelkezésre, de mindnek mások a céljai és súlypontjai. Ráadásul – e súlypontoknak megfelelően – más és más didaktikai felépítést követnek. Arról nem is beszélve, hogy hányszor egymásnak ellentmondóan...

Az én célom az volt, hogy egy pusztán *technikai bevezetőt* adjak közre, a speciális alkalmazások (pl. statisztika) elhagyásával. Ahogy már megjegyeztem, mindenképpen töredékes lesz az előálló kézirat, többen bizonyára éppen azt nem fogják megtalálni benne, amit kerestek, mások pedig esetenként más megfogalmazást tartanak helyesnek. Éppen ezért szeretném, ha ezt az R-bevezetőt jelen állapotában kiindulásnak, bővítendő alapnak tekintnék, mely folyamatosan fejlődhet, újabb részekkel egészülhet ki. Ebben a folyamatban szívesen vennék minden megjegyzést, kritikát és szövegjavaslatot.

Addig is remélem, hogy munkám hasznára válik majd néhányaknak.

Budapest, 2005. szeptember 27.

*Solymosi Norbert
Biomatematikai és Számítástechnikai Tanszék
Állatorvos-tudományi Kar
Szent István Egyetem
1078 Budapest, István u. 2.*

*E-mail: Solymosi.Norbert@aotk.szie.hu
Honlap: <http://www.univet.hu/users/nsolymosi/>*

Bevezetés

Az R egy olyan programozási nyelv és környezet, amely különösen alkalmas statisztikai számítások és grafikai megjelenítési feladatok megvalósítására. Az R-nyelv 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 szabad szoftver, ami a LESSER GNU¹ GENERAL PUBLIC LICENSE² közreadási feltételek alapján terjeszthető. Az S-nyelvvvel 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 magva egy parancsértelmező (interpreter) jellegű nyelv, szintaxisa felületesen hasonlít a C nyelvére, de tulajdonképpen egy „funkcionális programozási nyelv” a Scheme⁴ nyelvhez hasonló képességekkel. Az R-nyelv többek között lehetővé teszi ciklusok alkalmazását, illetve a moduláris programozást – függvényeken keresztül. A felhasználók által használt függvények többsége R-ben van megírva, amelyek kisebb belső primitív eljárásokat gyűjteményére épülnek. Más nyelvekben (C, C++ vagy FORTRAN) megírt eljárásokat is használhatunk a nagyobb hatékonyság érdekében, illetve kiegészítő primitíveket készíthetünk.

Az R-rendszer nagyszámú statisztikai eljárást tartalmaz. Ezek között találjuk a lineáris és generalizált lineáris modelleket, a nem-lineáris regressziós modelleket, idősoranalíziseket, klasszikus paraméteres és nem-paraméteres tesztek, klaszterezési és simítási módszereket. A statisztikai lehetőségek mellett sok függvény érhető el a rugalmas grafikai környezetben; e környezet segítségével az adatok reprezentációja igen sokféleképpen valósítható meg.

Az alaprendszerrel telepített eljárások mellett igen nagy számú csomag érhető el mind a CRAN⁵-ről, mind a Bioconductor⁶-ról, mind pedig egyéb internetes tárolókból. Az R rohamos terjedése folytán egyre többen hoznak létre egy-egy speciális adatelemzési területnek megfelelő eljárásgyűjteményeket, csomagokat, amelyeket az R-közösség rendelkezésére bocsájtanak.

Az R-nyelv fejlesztését *Robert Gentleman* és *Ross Ihaka* (Statistics Department of the University of Auckland) kezdte el. 1997. közepe óta az *R Development Core Team* módosíthatja az R forráskód CVS archívumát.

A „környezet” elnevezés arra utal, hogy inkább alaposan megtervezett és egységes rendszerről van szó, mint folyamatosan növekvő nagyon speciális és merev eszközköről (mint amilyenek a gyakrabban használt adatelemző szoftverek).

Az R nagyfokú hordozhatósággal rendelkezik, hiszen mind Windows, mind Linux, mind pedig MacOS operációs rendszerekhez elérhetők bináris telepítők. Emellett a forráskód is letölthető, így az abból történő telepítés még nagyobb rugalmasságot biztosít.

Az egyszerű interaktív programozás mellett lehetőség nyílik komplexebb megoldások fejlesztésére, illetve lehetséges integrált megoldások fejlesztése, összekapcsolás más alkalmazásokkal (pl. Microsoft Excel), illetve kombinálás más nyelvekkel (pl. L^AT_EX, Python, Visual Basic, Delphi, stb.). Ez utóbbi lehetőség segítségével saját alkalmazások fejleszthetők gyakori statisztikai vagy vizualizációs feladatok egyszerű megoldására.

Az R működésének alapelvei

Ahogy az előbbiekben láttuk, az R-nyelv egy interpretált szkript nyelv, a programkódokat nem fordítjuk bináris állománnyá a futtatáshoz, hanem az R-parancsértelmező értelmezi azokat. Az R-nyelv szintaxisa szerint megszerkesztett kódjainkat terminálon keresztül juttatjuk el az interpreterhez. Az 1–3. ábrákon látható egy Linux-os, egy DOS-os és egy Microsoft Windows-os terminál. Tulajdonképpen az 1. és a 2. egyformán működik, vagyis a beírt kódokat ENTER-rel jóváhagyva értelmezi. A 3. ábrán látható Windows-os *RGui* „terminál” az előbbieknél több, a menürendszerből elérhető funkcióval rendelkezik.

¹<http://www.gnu.hu/>

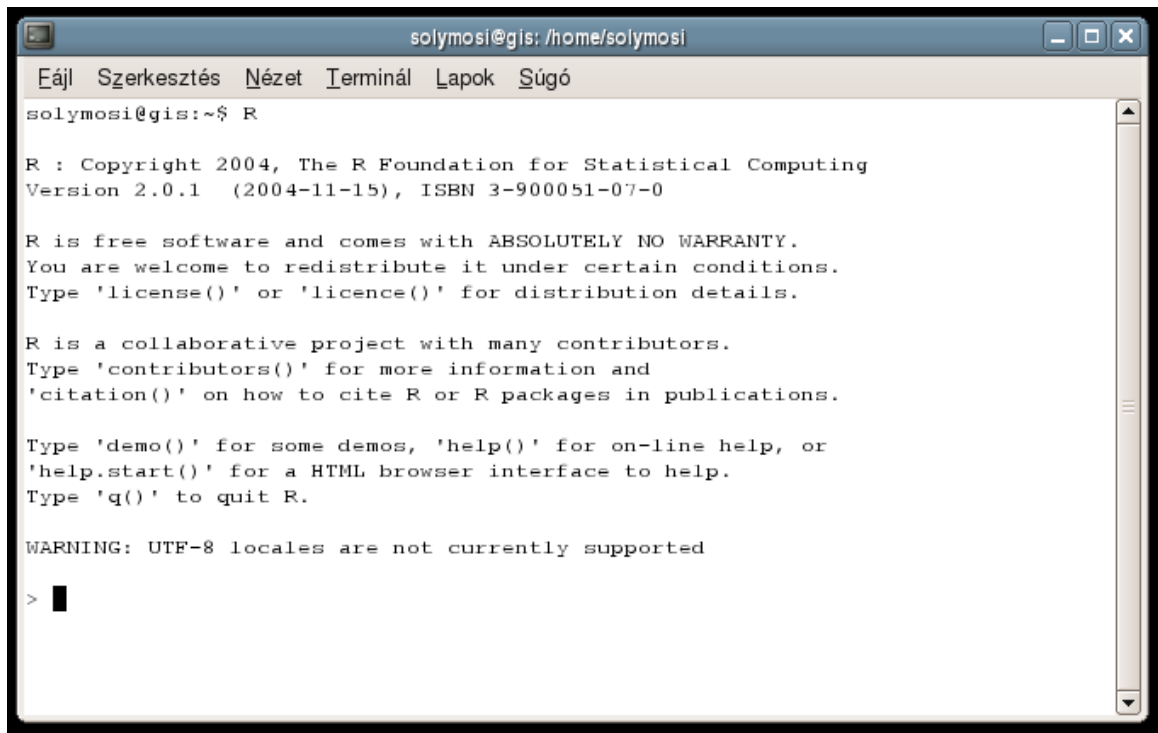
²<http://www.gnu.org/copyleft/lgpl.html>

³<http://www.insightful.com/>

⁴<http://www.plt-scheme.org/>

⁵<http://cran.r-project.org/>

⁶<http://www.bioconductor.org/>



The image shows a Linux terminal window titled "solymosi@gis: /home/solymosi". The terminal displays the R startup screen, which includes the following text:

```
solymosi@gis:~$ R

R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1 (2004-11-15), 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.

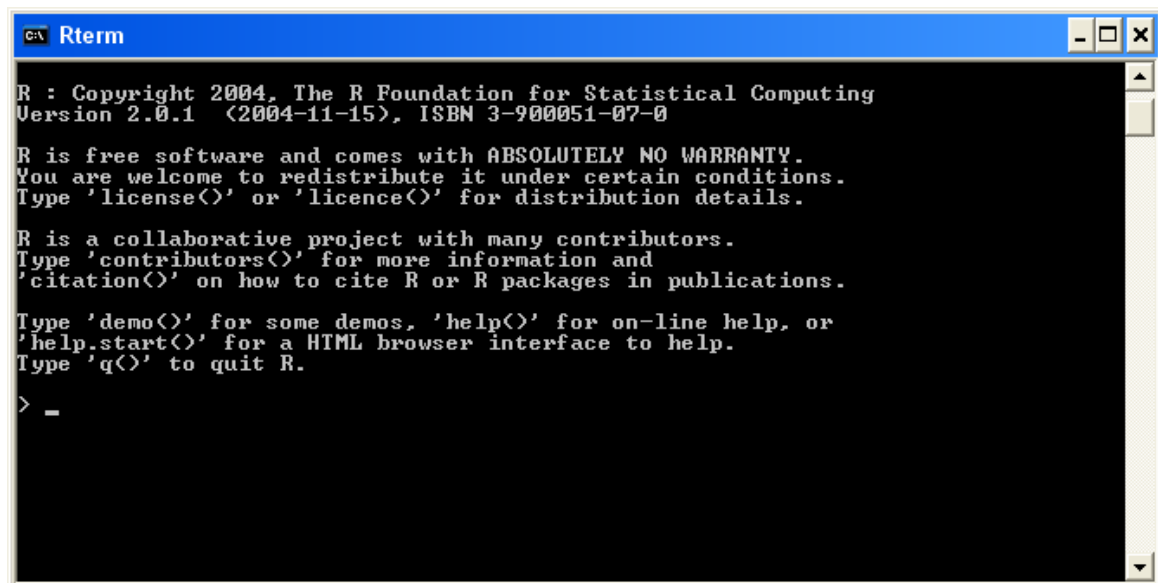
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

WARNING: UTF-8 locales are not currently supported

> █
```

1. ábra. Linux-os terminál



The image shows a DOS terminal window titled "Rterm". The terminal displays the R startup screen, which includes the following text:

```
R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1 (2004-11-15), ISBN 3-900051-07-0

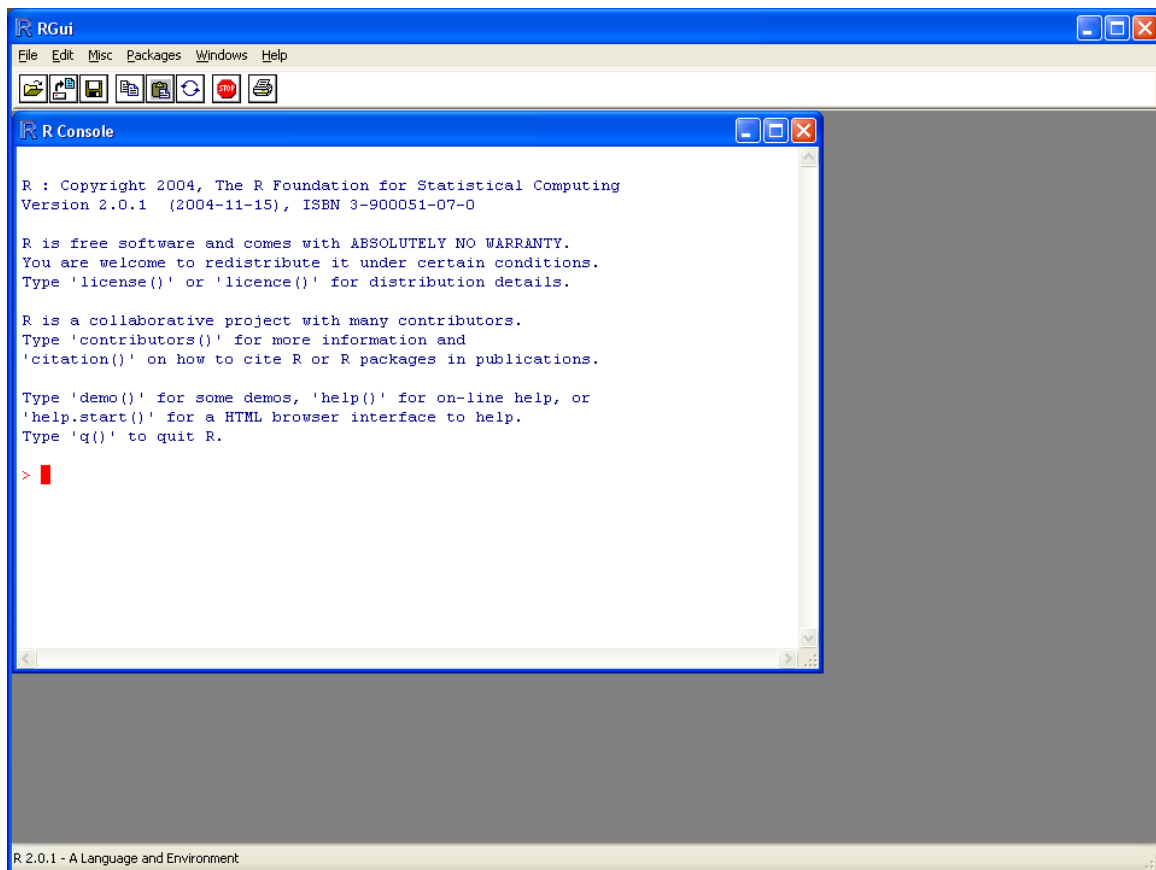
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

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 a HTML browser interface to help.
Type 'q()' to quit R.

> _
```

2. ábra. DOS-os terminál



3. ábra. Windows-os RGui

A kódok értelmezhetők *parancssori*, illetve *szkript* módban. Az előbbinél a promptban (>) beírt kódot új sor kezdésével küldhetjük el az értelmezőhöz. Az utóbbi esetében a megszerkesztett, általában több sorból álló fájlban mentett szkriptet hívjuk meg akár az R-terminálon (`source()`), akár más eszközzel (pl. R CMD BATCH). Ahogy a UNIX és Windows termiálokban általános, itt is a függőleges nyílak segítségével tudunk közlekedni az *utasítások történetében*. A már korábban lefuttatott utasítást a felfelé mutató nyíllal hívhatjuk újra és vagy újraértelmeztetjük úgy, ahogy van, vagy pedig javítjuk és az új utasítást futtatjuk le. A parancsértelmező által végrehajtott utasítások eredményei visszatérhetnek a terminálba, fájlba, adatbázisba, valamint a grafikus eszköz(ök)re (4. ábra).

Szintaktikai alapok

Az R-rendszer kis- és nagybetű érzékeny, így például az R és az r különböző szimbólumoknak számítanak, és különböző objektumokat jelenthetnek. Általában minden alfabetikus szimbólum használható a változók nevében, ezek mellett a `.` és az `_` 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, attól függően, hogy milyen operációs rendszeren, illetve milyen nyelvi beállításokkal működik a rendszerünk. Amennyiben hordozható kódot szeretnénk írni, akkor lehetőség szerint az objektumnevekben érdemes mellőzni az ékezetes betűket.

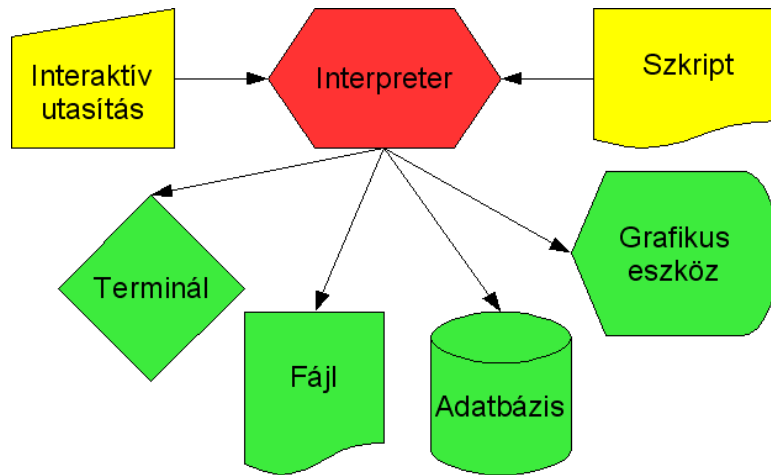
A megjegyzéseket `#`-el jelöljük, az ezt követő karakterek a sor végéig megjegyzésként értelmeződnek.

Utasítások

Az alaputasítások *kifejezések* vagy *értékadások* lehetnek. Ha egy *kifejezést értékadás* nélkül adunk meg mint utasítást, akkor az R kiértékeli és megjeleníti az értékét, ami egyben el is vész.

```
> 1 + 2
```

```
[1] 3
```

4. ábra. Parancsértelmező

A terminálban megjelentő látványból az első sor `> 1+2` a kifejezés, amit értelmezteni, futtatni szerettünk volna, a `[1] 3` sor pedig a kifejezés értékelése utáni eredmény. A szögletes zárójelek között lévő `1` azért áll ott, mert az eredményünk egy vektor és annak a sor elején álló elemének indexét jelzi.

Egy *értékkadás* esetén ugyancsak kiértékeli a kifejezést, de az értékét automatikus megjelenítés nélkül eltárolja egy objektumban. Az *értékkadás* általános szintaxisa `változó <- kifejezés`. Az értékkadás jeleként itt a `<-` használatos, de lehet használni a `=` és a `->` jelet is. Korábban csak az általunk használható 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 kiírja az `a` objektumban tárolt értéket. Fontos megjegyezni, hogy amennyiben egy változónak új értéket adunk, akkor annak a korábbi értéke törlődik és felülíródik az újjal.

```

> a <- 5
> a

[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 mindig 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, ilyenkor a prompt az alapértelmezett `>`-ről `+-`-ra változik.

Az utasításokat csoportosíthatjuk is, kapcsos zárójelek `{}` közé rendezve. Az utasítás-csoportokat esetenként *blokk*nak hívják. Egy magában álló utasítást akkor értelmez az R-környezet, ha annak szintaxisa teljes, és új sort kezdünk. A blokkot mindaddig nem értelmezi, amíg azt le nem zárjuk, és új sort nem kezdünk.

```

> {
+   a <- 1
+   b <- a + 2
+   b
+ }

[1] 3
  
```

A következőkben az *utasításon* mind a magában álló, mind a blokkba rendezett utasításokat értjük.

Függvények

Az R-ben létrehozott és kezelt egységeket *objektumoknak* nevezzük. Ezek lehetnek változók, tömbök, karakterláncok, függvények vagy ezek komplex struktúrái. Az R-rendszeren belül az objektumokon *operátorokkal* és

függvényekkel végezhetünk különböző műveleteket. A függvények a `függvény.neve(arg1, arg2, argN)` szintaxis szerint épülnek fel. A `függvény.neve` határozza meg a függvény nevét, amivel azonosítja a rendszer a meghívandó eljárás(oka)t. A zárójelek közé foglalt `argN` a függvény argumentumait jelenti. Egyes függvények esetén nem minden argumentumnak kell megadnunk értéket, mivel a függvény rendelkezik alapértelmezett értékekkel.

Az adatok tartóssága és az objektumok eltávolítása

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

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, ami automatikus betöltődés esetén adatok felcserélődéséhez vezethet. Ennek kivédése érdekében egyrészt minden elemzést külön könyvtárban tanácsos végezni, másrészt érdemes az objektumneveket specializálni.

Szövegszerkesztők

Amennyiben hosszabb szkripteket szeretnénk készíteni, a terminálban való programkódírást nehézkes és igen áttekinthetetlen. Ezért, ha ilyen feladatba fogunk, érdemes valamilyen szövegszerkesztővel megírni a kódjainkat.

Windows

Az RGui a 2.0.1-es verziótól kezdve tartalmaz egy szkript-szerkesztő eszközt, ami igen egyszerű szövegszerkesztő, kevés funkcióval. Előnye viszont, hogy a benne szerkesztett kódból egyes sorokat vagy kijelölt szakaszokat közvetlenül lehet átadni az R-terminálnak értelmezésre.

A Microsoft Windows környezetben a kellékek között elérhető *Jegyzettömb* teljes mértékben megfelel a kód szerkesztéséhez. Ha ebben szerkesztjük a szkriptünket, akkor vagy úgy tudjuk az R-értelmezőhöz eljuttatni, hogy a `source()` függvényt használjuk, vagy a szerkesztőből a vágólappra helyezett kódrészletet beillesztjük az R-terminálba. Hátránya még, hogy egyszerre csak egy állomány tudunk benne szerkeszteni.

Igen hasznos eszköz a Tinn-R⁷ szerkesztő, amivel egyszerre több fájlt kezelhetünk és ezek projektbe szervezhetők. Emellett képes kommunikálni a beállított R-terminállal. Ennek segítségével a szerkesztőből közvetlenül küldhetünk kódokat vagy egész szkript-fájlokat az R-hez. (A mellékletben rövid leírás található a Tinn-R beállításával kapcsolatban.)

Linux

Linuxon igen jó eszköz a *Kate*⁸ szerkesztő, ami rendelkezik egy terminálablakkal is, így egy alkalmazáson belül lehet szerkeszteni és futtatni is a kódokat.

Platformfüggetlen alkalmazások

Platformfüggetlen, sokféle feladatra használható eszköz az *Emacs*⁹ vagy az *Xemacs*¹⁰, mely mint szerkesztő is nagyon jól használható, de ha az *ESS*¹¹ (Emacs Speaks Statistics) csomagot is telepítjük, akkor emellett, mint az R-értelmezővel való közvetlen együttműködésre is képes.

⁷<http://www.sciviews.org/Tinn-R/>

⁸<http://kate.kde.org/>

⁹<http://www.gnu.org/software/emacs/emacs.html>

¹⁰<http://www.xemacs.org/>

¹¹<http://ess.r-project.org/>

Segédletek

Az R nagy előnye, hogy igen jól dokumentált. A működés minden pontja kontrollálható, a nyitott forráskódnak köszönhetően. Természetesen a forráskód tanulmányozásánál egyszerűbb információnyerési lehetőségek is rendelkezésre állnak az R használatával kapcsolatban felmerülő kérdések megválaszolására. Ezeket a lehetőségeket gyűjtöttem össze az alábbiakban.

help

Az R-ben a beépített sűgórendszer a UNIX `man` segédletéhez hasonlít. Ha egy adott függvénnyel kapcsolatban részletesebb információkat szeretnénk megismerni, használhatjuk a `help` parancsot. Ha például érdekelnek a `mean` függvény paraméterezésének részletei, így járhatunk el:

```
> help(mean)
```

Ugyanezt érhetjük el az egyszerűsített szintaxissal:

```
> ?mean
```

```
mean                package:base                R Documentation
```

```
Arithmetic Mean
```

```
Description:
```

```
Generic function for the (trimmed) arithmetic mean.
```

```
Usage:
```

```
mean(x, ...)
```

```
## Default S3 method:
```

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

```
Arguments:
```

```
x: An R object. Currently there are methods for numeric data frames, numeric vectors and dates. A complex vector is allowed for 'trim = 0', only.
```

```
trim: the fraction (0 to 0.5) of observations to be trimmed from each end of 'x' before the mean is computed.
```

```
na.rm: a logical value indicating whether 'NA' values should be stripped before the computation proceeds.
```

```
...: further arguments passed to or from other methods.
```

```
Value:
```

```
For a data frame, a named vector with the appropriate method being applied column by column.
```

If 'trim' is zero (the default), the arithmetic mean of the values in 'x' is computed.

If 'trim' is non-zero, a symmetrically trimmed mean is computed with a fraction of 'trim' observations deleted from each end before the mean is computed.

References:

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

See Also:

'weighted.mean', 'mean.POSIXct'

Examples:

```
x <- c(0:10, 50)
xm <- mean(x)
c(xm, mean(x, trim = 0.10))

mean(USArrests, trim = 0.2)
```

Az output szerkezete

A fent látható output szerkezete általános, néhány szóban a következő:

Az első sor első eleme a keresett objektum *neve*, a második pedig annak a *csomagnak* a neve, ami tartalmazza azt. A következő sorban lesz a dokumentáció idevonakozó szakaszának *címe*, ezt követik a leíró részek:

Description: rövid leírás

Usage: ha függvényről van szó, akkor a függvény szintaxisa az összes argumentumával, ha operátorról, akkor operátor tipikus alkalmazásának szintaxisa

Arguments: az argumentumok jelentésének leírása és használatuk szintaxisára vonatkozó megjegyzések

Details: részletesebb leírás

Value: amennyiben van ilyen, akkor a függvény vagy operátor használata nyomán keletkező objektum leírása

References: a fejlesztők által fontosnak tartott közlemény(ek) bibliográfiai adatai

Author(s): a függvény, vagy az azt tartalmazó csomag készítőinek neve

See Also: az R-dokumentációban a témával kapcsolatban javasolható egyéb szakaszok

Examples: a megértést segítő néhány példát mutat be

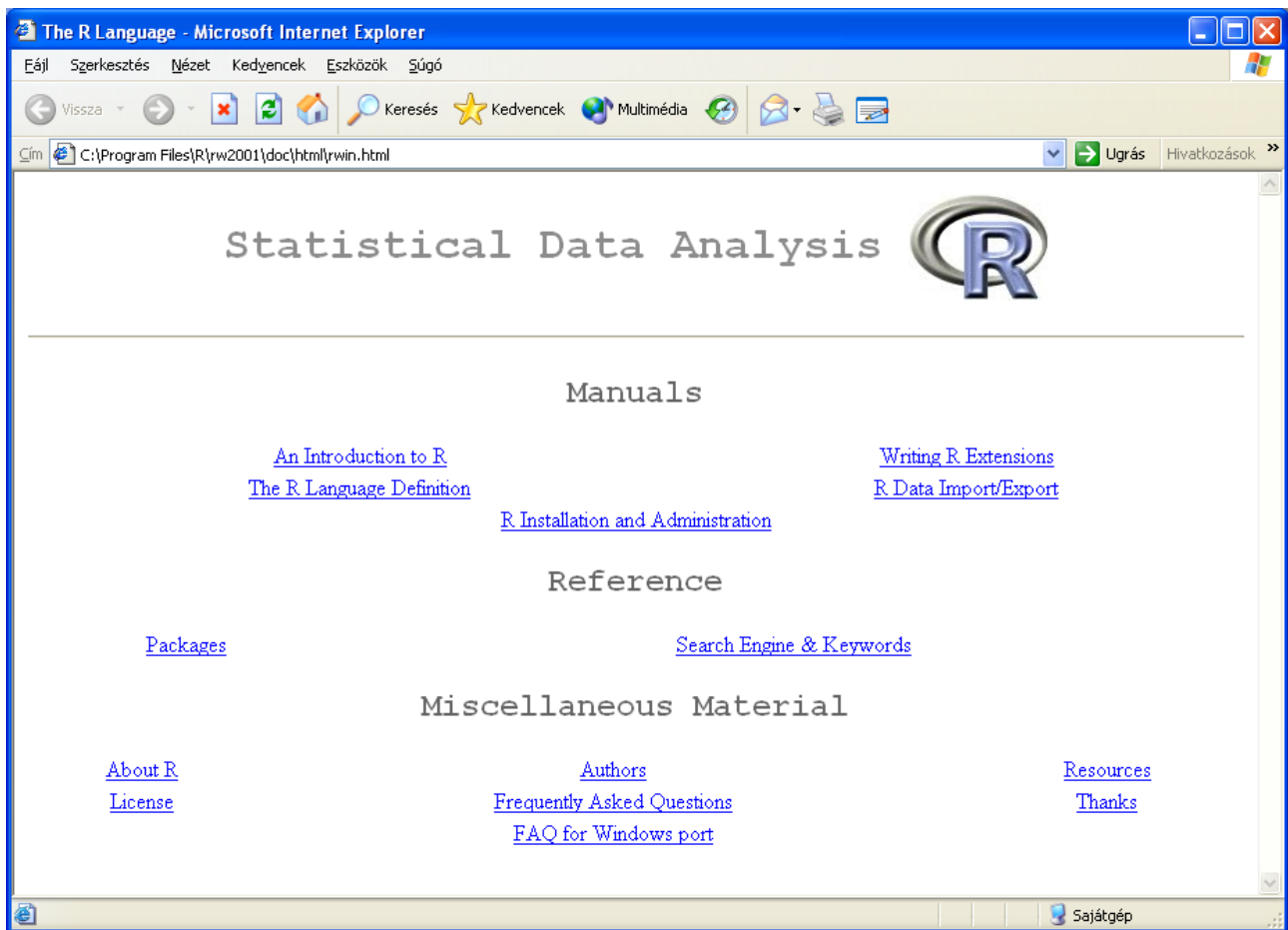
Ha speciális karaktereket tartalmazó kifejezést keresünk, a karaktereket szimpla vagy dupla idézőjelek közé kell foglalnunk:

```
> help("[")
```

Ugyanezt az eljárást használjuk egyes szavak esetén, amelyek szintaktikai jelentéssel bírnak (mint pl.: az `if`, a `for` vagy a `function`). A legtöbb R-telepítésben elérhető egy HTML sűgő rendszer is, amit a következő paranccsal tudunk meghívni:

```
> help.start()
```

A parancs eredményeként az alapértelmezett web browserünkben megjelenik a 5. ábrán látható lap, ami a *telepített* verzióval és csomagokkal kapcsolatos információk gyors elérését segíti. A képernyőn látható linkek segítségével tudunk vándorolni a számunkra fontos területek között. Az *An Introduction to R*, *Writing R Extensions*, *The R Language Definition*, *R Data Import/Export*, *R Installation and Administration* dokumentumok itt elérhető HTML verziója mellett nyomtatható pdf formátumban is elérhetők az általános telepítés után, mégpedig Windows környezetben a `C:\Program Files\R\rw2001\doc\manual` könyvtárban. A *Search Engine*



5. ábra. HTML súgó

É *Keywords* rész csak akkor működik ha telepített Java-val rendelkezünk valamint ha a browserünk Java és JavaScript támogatással rendelkezik.

A `help()` utasítást csak akkor tudjuk használni, ha pontosan ismerjük a keresett függvény nevét. Ha nem helyes adjuk meg a függvény nevét, mint a következő példában, akkor nem jutunk a várt információhoz: Ha pl. szeretnénk a *t-teszt* alkalmazásával kapcsolatos információkat 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()` függvénnyel próbáljuk megtalálni azokat a csomagokat, illetve függvényleírásokat, amelyekben ez a szóösszetétel 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 nem csak a betöltött, hanem az összes telepített csomagban keres az objektum nevére. Hozzátevé azt, 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.

```
> help(gh.test, try.all.packages=T)
```

```
Help for topic 'gh.test' is not in any loaded package but can be
found in the following packages:
```

Package	Library
gmodels	C:/PROGRA~1/R/rw2011/library

A jelzett csomag betöltése után a sűgó kiírja a használattal kapcsolatos információkat. Másik lehetőség az információ megszerzésére, ha az intező segítségével megnyitjuk a megjelölt könyvtárat, vagyis a fenti példa szerint a `C:\Program Files\R\rw2001\library` könyvtáron belül a `gmodels` alkönyvtárat. Itt a 24. ábrához hasonló szerkezetet láthatunk, amelyen belül van egy `chtml` alkönyvtár. Ez tartalmaz egy `gmodels.chm` fájlt, ami egy állományban tartalmazza a csomag teljes dokumentációját. Látható, hogy az általunk megadott `t-test` szöveget az R átalakította `t - test`-é. Most próbáljuk megkeresni a `help.search("t - test")` utasítással a keresett függvényt.

```
> 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 szóközöket elhagyjuk.

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

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

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

apropos

Az `apropos` függvénnyel a *betöltött* könyvtárak objektumainak *neveiben* kereshetünk karakteret vagy azok láncolatát. A függvény a telepített, de nem betöltött könyvtárakban nem keres.

```
> apropos("test")

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

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

```
> apropos("*.test")

[1] "ansari.test"           "bartlett.test"       "binom.test"
[4] "Box.test"              "chisq.test"          "cor.test"
[7] "fisher.test"           "fligner.test"        "friedman.test"
[10] "kruskal.test"          "ks.test"              "mantelhaen.test"
[13] "mcnemar.test"          "mood.test"            "oneway.test"
[16] "pairwise.prop.test"    "pairwise.t.test"     "pairwise.wilcox.test"
[19] "power.anova.test"       "power.prop.test"      "power.t.test"
[22] "PP.test"               "prop.test"            "prop.trend.test"
[25] "quade.test"            "shapiro.test"         "t.test"
[28] "var.test"              "wilcox.test"
```

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

```
> apropos("^test")

[1] "testVirtual"           "testPlatformEquivalence"
```

example

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 paraméterezésének 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
```

demo

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 alapsomagokhoz tartozó bemutatókat.

```
> demo()
```

```
Demos in package 'base':
```

```
is.things      Explore some properties of R objects and
                is.FOO() functions. Not for newbies!
recursion      Using recursion for adaptive integration
scoping        An illustration of lexical scoping.
```

```
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
```

```
Demos in package 'stats':
```

```
glm.vr         Some glm() examples from V&R with several
                predictors
lm.glm         Some linear and generalized linear modelling
                examples from 'An Introduction to Statistical
                Modelling' by Annette Dobson
nlm            Nonlinear least-squares using nlm()
smooth        'Visualize' steps in Tukey's smoothers
```

Ha az összes telepített csomaghoz tartozó bemutatószkriptet ki szeretnénk listázni, akkor a fenti forma helyett a `demo(package = .packages(all.available = TRUE))` utasítást használjuk. A listából kiválasztva egy demót, pl. a `graphics` csomagból az `image` bemutatót, a `demo(image)` utasítással futtathatjuk le.

Segédletek a CRAN-on

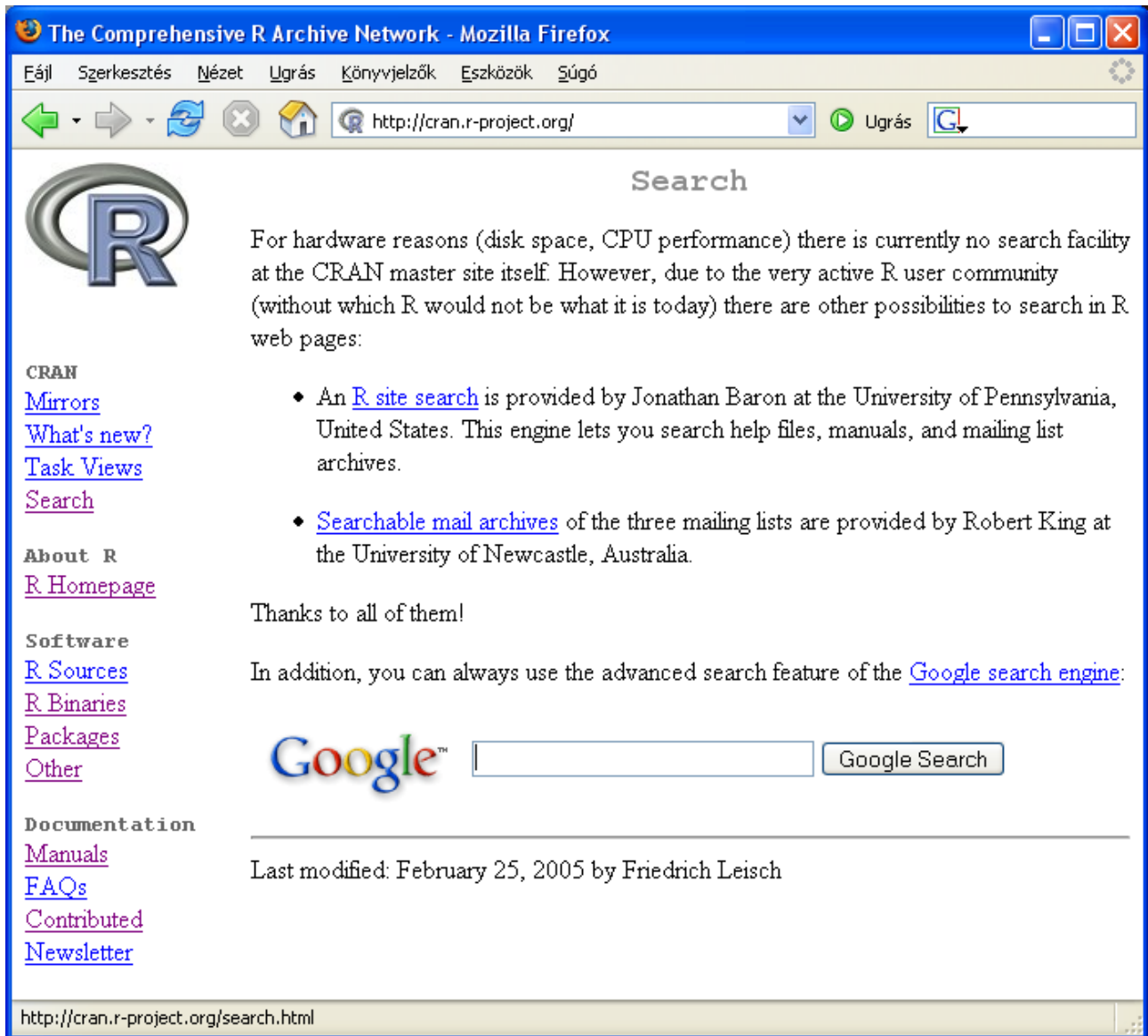
Gyakran feltett kérdések (GYIK)

A gyakran feltett kérdések¹² és azok archivált válaszai sok esetben nyújthatnak célzott segítséget.

Keresés

Az eddigiekből látható, hogy amennyiben valamely függvénnyel vagy egyéb objektummal kapcsolatban szeretnénk információhoz jutni, a fenti lehetőségekkel csupán a gépünkre telepített csomagok dokumentációjában tudunk keresgélni. Azonban a legtöbb esetben a csomagoknak csak egy része van telepítve gépünkre, vagyis az R eljárásainak csak töredékéről szerezhetünk információkat. A CRAN széleskörű keresési lehetőséget kínál (6. ábra), aminek segítségével minden függvényről, egyéb objektumról begyűjthetők a kívánt információk.

¹²<http://cran.r-project.org/>



6. ábra. Keresési felület a CRAN-on

R News

Az R News évente három alkalommal jelenik meg és az R-rel kapcsolatos újdonságokról számol be.

Levelezési listák

Számos levelezési lista érhető el. Ezek igen nagy aktivitással működnek. Az általános célú levelezési listák: `R-announce`, `R-packages`, `R-help` és `R-devel` listák. Egyes speciális érdeklődés területnek megfelelően létrehozott listák: `R-sig-DB`, `R-SIG-Debian`, `R-sig-Epi`, `R-sig-finance`, `R-sig-Geo`, `R-sig-gR`, `R-SIG-GUI`, `R-SIG-Mac`, `R-sig-QA`.

Objektumok

Az R-en belül mind az adatstruktúrák, mind pedig a függvények *objektumként* jelennek meg. Az *R-munkafolyamatban* jelenlévő objektumokat kilistázhatjuk a `ls` vagy az `objects` függvényekkel. Az egyes objektumok *típusára*, illetve *módjára* vonatkozó információkat kiolvashatjuk függvények segítségével. A `typeof(x)` függvény segítségével az `x` objektum *típusát* olvashatjuk ki. Néhány gyakrabban előforduló érték az 1. táblázatban olvasható. Az `x` objektum S-nyelvel kompatibilis *módját* a `mode(x)` függvénnyel olvashatjuk ki. A `storage.mode(x)` függvény pedig az `x` objektum tárolási módját adja vissza. Ez utóbbi akkor fontos, ha valamilyen egyéb nyelven írt függvényt hívunk meg (C, FORTRAN, stb.) és a műveletek elvégzése előtt fontos ellenőrizni, hogy a formátum megfelel-e az adott eljárás argumentum-elvárásainak. Az egyes *vektorok* típusára és módjára jellemző értékeket a 2. táblázat tartalmazza.

Attribútumok

A NULL kivételével minden objektumhoz hozzárendelhetők *attribútumok*. Az attribútumok egy lista elemeként vannak eltárolva. A lista elemeit az `attributes` függvénnyel olvashatjuk, illetve az `<-attributes`-al írhatjuk. Egyedi komponenseket az `attr` függvénnyel olvashatunk, illetve az `<-attr`-al írhatunk. Egyes attribútumok sajátos elérési funkcióval rendelkeznek (pl. a faktoroknál a `levels`), amennyiben ilyen elérhető, érdemes ezeket használni az adott feladatra. A mátrixok és a tömbök egyszerű vektorok `dim` és `dimnames` attribútumokkal kiegészítve.

names

A `names` attribútum az adott objektum egyes elemeire utaló „címke”, amellyel egyben hivatkozni is lehet az adott elem(ek)re. Az adott objektumból kiolvashatjuk a neveket a `names` függvénnyel. Ugyanakkor a `<-names` formában írhatjuk is azokat, természetesen ekkor ügyelni kell a típusra és a méretekre. Egy dimenziós tömbök esetében a `names` attribútum tulajdonképpen a `dimnames[[1]]`-el egyezik meg.

dim

A `dim` attribútumot a tömbök létrehozására vezették be. A tömbök tulajdonképpen vektorok, amelyek „oszlopban” tárolják a vektor adatait, a tömb kiterjedéseit pedig a `dim` attribútumban egész számokból álló vektorként adjuk meg. Az R ellenőrzi, hogy a dimenziókban megadott kiterjedési hosszak megfelelnek-e a vektor hosszának. A dimenziók mérete lehet akár nulla is. A vektor nem egyezik meg az egydimenziós tömbbel, mivel az utóbbi rendelkezik `dim` attribútummal, míg az előbbi nem.

1. táblázat. Fontosabb `typeof` visszatérési értékek

érték	jelentése
NULL	Null
symbol	változó neve
closure	függvény
logical	logikai értékekből álló vektor
integer	egész számokból álló vektor
double	lebegőpontos számokból álló vektor
complex	komplex adatokból álló vektor
character	karaktervektor
list	lista
raw	bináris vektor

2. táblázat. A *típus, mód és tárolási mód* kombinációk

<code>typeof</code>	<code>mode</code>	<code>storage.mode</code>
logical	logical	logical
integer	numeric	integer
double	numeric	double
complex	complex	complex
character	character	character

3. táblázat. Típus-konverziók

forrás	konverzió	eredmény
0 1 2 NA	as.character	"0" "1" "2" NA
0 1 2 NA	as.logical	FALSE TRUE TRUE NA
0 1 2 NA	as.complex	0+0i 1+0i 2+0i NA
FALSE TRUE NA	as.character	"FALSE" "TRUE" NA
FALSE TRUE NA	as.numeric	0 1 NA
FALSE TRUE NA	as.complex	0+0i 1+0i NA
'0' '1' 'a' NA	as.logical	NA NA NA NA
'0' '1' 'a' NA	as.numeric	0 1 NA NA
'0' '1' 'a' NA	as.complex	0+0i 1+0i NA NA
0 1.2 2.3 NA	as.logical	FALSE TRUE TRUE NA
0 1.2 2.3 NA	as.numeric	0.0 1.2 2.3 NA
0 1.2 2.3 NA	as.character	"0" "1.2" "2.3" NA

dimnames

A tömbök egyes dimenziói elnevezhetők a `dimnames` attribútumban tárolt nevekkal. A neveket egy szöveges vektorokból álló listában adhatjuk meg.

class

Az R beépített osztály-rendszere a `class` attribútumon keresztül kezelhető. A `class` attribútum szöveges vektor, azokat az osztályokat tartalmazza, amelyekből az adott objektum származik.

tsp

A `tsp` az idősorobjektumok attribútuma, azok paramétereit tárolja (`start`, `end` és `frequency`).

Objektumok kezelése

Objektumok létrehozása

Ahogy már a korábbiakban láttuk, létrehoztunk objektumokat értékadással. Ekkor azonban az objektum *módja, típusa* általánosan lesz meghatározva. Az objektumot úgy is létrehozhatjuk, hogy *módját, típusát, méretét*, stb. előre meghatározzuk. Ez a lehetőség igen hasznos lehet az objektumokkal való manipulációk során. Például létrehozhatunk üres objektumokat és módosíthatjuk elemeiket, ami hatékonyabb, mint a `c()` függvénnyel egyszerre feltölteni a vektort. Az elemek módosításában az indexeket is használhatjuk.

Az adattároló objektumokat (lásd alább) feltölthetjük adatokkal, adatfájlok beolvasásával, adatok generálásával, illetve adatsorok billentyűzetről való bevitelével. Az adatfájlok olvasásáról és írásáról a következő fejezetben lesz szó. Az alábbiakban (az egyes objektumtípusok ismertetése előtt) az adatgenerálásról írok, mivel az adattároló objektumok ismertetése előtt ez célszerűnek látszik.

Adatok begépelése

A számításainkban vagy a grafikai megjelenítésekben használandó adatainkat, ha nem túl nagy mennyiségről van szó, akkor gyorsan begépelhetjük, többféleképpen is:

c

A `c` függvény értékeket vagy objektumokat fűz össze *vektor*rá vagy *listá*vá. Alapértelmezésben a megadott értékeket *vektor*rá fűzi össze.

```
c(..., recursive=FALSE)
```

A `c` függvény argumentumainak leírása:

...	Az összefűzendő értékek vagy objektumok, amelyeket vesszővel választunk el.
<code>recursive</code>	Ha az értéke <code>TRUE</code> és az objektumok között van lista is, akkor a lista minden elemét egy vektor elemeivé alakítja és a végleges objektum vektor lesz. Ha <code>FALSE</code> és az összefűzendő objektumok egyik eleme lista, akkor az eredményként létrejövő objektum is <i>lista</i> lesz.

A `c` függvény segítségével létrehozhatunk egy egyszerű vektort:

```
> a <- c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
> a
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Üres vektort is létrehozhatunk:

```
> a <- c()
> a
```

```
NULL
```

scan

A `scan` függvény részletesebb leírását az adatfájlok olvasása és írása résznél lehet megtalálni, itt egy egyszerűbb alkalmazását láthatjuk. Segítségével a következő módon hozhatunk létre hasonló vektort:

```
> a <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
10:
Read 9 items
> a
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Az egyes értékeket szóközzel elválasztva gépeljük be egymás után.

Adatgenerálás

Szabályos sorozatok

A következő függvények használhatók:

seq

Szabályos sorozatok generálására használhatjuk.

```
from:to
a:b
```

```
seq(from, to)
seq(from, to, by=)
seq(from, to, length=)
seq(along=)
seq(from)
```

A `seq` függvény argumentumainak leírása:

<code>from</code>	A sorozat induló értéke.
<code>to</code>	A sorozat záróértéke.
<code>by</code>	A sorozat növekedésének/csökkenésének léptéke.
<code>length</code>	A létrehozandó sorozat hossza, elemszáma.
<code>along</code>	Az itt megadott objektum hosszának megfelelő hosszúságú sorozatot hoz létre.
<code>a,b</code>	Egyenlő hosszúságú „faktorok”.

Szabályos, egész számokból álló sorozatot generál a következő utasítás, amiben a 1-től 10-ig terjedő vektor lesz:

```
> a <- 1:10
> a
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> a <- 1:10 - 1
> a
[1] 0 1 2 3 4 5 6 7 8 9
```

A második sorozatnál azt láthatjuk, hogy előbb generál 1-től 10-ig egy sorozatot és utána, az eredményvektor mindegyik tagjából kivon egyet.

```
> a <- 1:(10 - 1)
> a
[1] 1 2 3 4 5 6 7 8 9
```

Ez utóbbi esetben a sorozat kezdő értéke 1 lett, mivel nem a vektor mindegyik eleméből vontunk ki egyet, hanem a szekvencia maximális értékéből, amit a `:` utáni zárójellel adtunk meg.

```
> a <- seq(1, 5, 0.5)
> a
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

, ahol az `seq` függvény első argumentuma a kezdete, a második a vége, a harmadik pedig a növekvénye a sorozatnak. Más módon is előállítható az előző sorozat:

```
> a <- seq(length = 9, from = 1, to = 5)
> a
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

, ahol a `length` a sorozat elemszámát, a `from` a kezdetét, a `to` pedig a végét jelenti.

rep

Vektorok vagy listák elemeit ismétli és ezáltal hoz létre sorozatot. Az általános `rep` függvénynél gyorsabb az egyszerűsített `rep.int` változat.

```
rep(x, times, ...)

## Default S3 method:
rep(x, times, length.out, each, ...)

rep.int(x, times)
```

A `rep` függvény argumentumainak leírása:

<code>x</code>	egy vektor (bármilyen formában), vagy pairlist vagy 'POSIXct' vagy 'POSIXlt' vagy 'date' objektum
<code>times</code>	Nem negatív egész szám(ok)ból álló vektor, ami megadja, hogy az <code>x</code> -et, illetve elemeit hányszor ismétlje meg a függvény. Ha a vektor hossza 1, akkor az abban megadott számszor ismétli meg az <code>x</code> -et. Ha a hosszúsága egyezik a <code>x</code> hosszúságával, akkor az egyező indexű elemeinek értékének megfelelő számban ismétli <code>x</code> adott elemét.

<code>length.out</code>	Azt adhatjuk meg vele, hogy az eredményvektor milyen hosszú legyen.
<code>each</code>	Az <code>x</code> minden elemét megismétli az itt megadott egész számnak megfelelően.
<code>...</code>	további argumentumok

Néhány példa:

```
> a <- c(1,2,3)
> b <- rep(a, 3)
> b
```

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

A `b` vektort úgy hozza létre, hogy az `a` vektort háromszor megismétli.

```
> b <- rep(a, c(3,2,1))
> b
```

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

Itt látható az, hogy ha a `times` argumentumban megadott érték hossza nem 1 és megegyezik az `a` vektor hosszával, akkor az egymásnak megfelelő indexű szorzóval ismétli meg a forrásvektor elemeit. Esetünkben az első elemet háromszor, a második elemet kétszer, a harmadik elemet pedig egyszer illeszti be az eredményvektorba.

sequence

A `sequence` függvény segítségével az argumentumban megadott értékekkel végződő sorozatokat generálhatunk. Tulajdonképpen úgy, mintha a `seq(from, to)` függvényben a `from` mindig 1 lenne és csak a `to` értéket adnánk meg. Azzal együtt, hogy itt egyszerre több `to` értéket megadhatunk.

```
> a <- sequence(c(4,5))
> a
```

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

```
> a <- sequence(4:5)
> a
```

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

A fenti két példa ugyanazt az eredményt adja, a második valamivel egyszerűbben. Mindkét esetben ugyanaz a vektor a függvény argumentuma, csak más formában adjuk meg.

gl

A `gl` függvény faktorokat hoz létre a megadott szinteknek megfelelően.

```
gl(n, k, length = n*k, labels = 1:n, ordered = FALSE)
```

A `gl` függvény argumentumainak leírása:

<code>n</code>	Egész szám, ami megadja a szintek számát.
<code>k</code>	Egész szám, ami az ismétlések számát határozza meg.
<code>length</code>	Az eredmény hosszát megadó egész szám.
<code>labels</code>	A faktor szintjeinek elnevezésére szolgáló vektor.
<code>ordered</code>	Logikai érték, ami azt határozza meg, hogy az eredményt rendezze-e vagy sem a függvény.

```
> a <- gl(3, 5)
> a
```

```
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
```

```
> a <- gl(3, 5, length = 30)
> a
```

```
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
```

```
> a <- gl(2, 6, label = c("Male", "Female"))
> a
```

```
[1] Male Male Male Male Male Male Female Female Female Female
[11] Female Female
Levels: Male Female
```

```
> a <- gl(2, 10)
> a
```

```
[1] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
```

```
> a <- gl(2, 1, length = 20)
> a
```

```
[1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
```

```
> a <- gl(2, 2, length = 20)
> a
```

```
[1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels: 1 2
```

Véletlen sorozatok

A statisztikában igen hasznos eljárások azok, amelyek segítségével véletlen adatok állíthatók elő. Az R-nyelvben nagyszámú sűrűségfüggvény áll rendelkezésre erre a feladatra. E függvények általános formája `...func(n, p1, p2,` ahol a `func` a valószínűségi függvényt jelzi, `n` a generálandó elemszámot és `p1, ...` a valószínűség függvény paraméterei (4. táblázat). A függvények nevében szereplő `...` helyére `d` (*sűrűségfüggvény*), `p` (*eloszlásfüggvény*), `q` (*kvantilis függvény*) vagy `r` (*véletlenszám-generálás*) betű kerülhet.

4. táblázat. Véletlen sorozatok

szabály	függvény
béta	<code>...beta</code>
binomiális	<code>...binom</code>
Cauchy	<code>...cauchy</code>
χ^2	<code>...chisq</code>
exponenciális	<code>...exp</code>
Fisher-Snedecor (F)	<code>...f</code>
gamma	<code>...gamma</code>
Gaussian (normális)	<code>...norm</code>
geometrikus	<code>...geom</code>
hypergeometrikus	<code>...hyper</code>
logisztikus	<code>...logis</code>
lognormális	<code>...lnorm</code>
negative binomiális	<code>...nbinom</code>
Pearson (c2)	<code>...chisq</code>
Poisson	<code>...pois</code>
Student(t)	<code>...t</code>
uniform	<code>...unif</code>
Weibull	<code>...weibull</code>
Wilcoxon's statistics	<code>...wilcox, ...signrank</code>

Adattároló objektumok

Vektor

A vektorokat alkotják *numerikus*, *karakter*, *komplex* vagy *logikai* adattípusok. Ugyanazon vektoron belül *többféle típus* NEM használható. Korábban már láttuk, hogy vektorokat létrehozhatunk többféle módon is az adatgeneráló függvények vagy a `c` függvény segítségével. A `vector` függvénnyel is létrehozhatunk vektorokat.

```
vector(mode = "logical", length = 0)
```

A `vector` függvény argumentumainak leírása:

<code>mode</code>	E kulcsszó, arra utal, hogy milyen típusú adatok tárolására szolgál a készítendő vektor.
<code>length</code>	Nem negatív egész szám, amivel beállítjuk, hogy hány elemet tartalmazzon a vektor.

A létrehozott vektor értékei attól függenek, hogy milyen *módot* állítottunk be: `0` ha numerikus, `FALSE` ha logikai vagy `"` ha karakteres. A `vector` függvény, aminek két argumentuma van (`mode` és `length`), létrehoz egy vektort,

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

```
[1] 0 0 0 0 0
```

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

```
[1] FALSE FALSE FALSE FALSE FALSE
```

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

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

Ugyanezt érhetjük el egyetlen argumentum (`length`) megadásával, ha a `numeric`, a `logical` vagy a `character` függvényeket használjuk.

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

```
[1] 0 0 0 0 0
```

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

```
[1] FALSE FALSE FALSE FALSE FALSE
```

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

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

Faktor

A faktorokat alkotják *numerikus* vagy *karakter* adattípusok. Ugyanazon faktoron belül *többféle típus* NEM használható. A `factor` függvény nem pusztán egy kategoriális változóból álló vektort hoz létre, hanem a változó szintjeit is kigyűjti.

```
factor(x, levels = sort(unique.default(x), na.last = TRUE),
      labels = levels, exclude = NA, ordered = is.ordered(x))
```

A `factor` függvény argumentumainak leírása:

<code>x</code>	Vektor, karakter vagy numerikus.
<code>levels</code>	Vektor, amely azon értékekből állhat, amelyekből a <code>x</code> felépül. (Alapértelmezésben a <code>x</code> vektor növekvő sorba állított egyedi értékei.)

labels	Értéke vagy a levels vektor hosszúságával megegyező hosszúságú címkéket tartalmazó vektor, vagy 1 hosszúságú karaktervektor.
exclude	Vektor, ami azokat az értékeket tartalmazza, amelyeket el szeretnénk távolítani a létrehozandó faktorból. Ennek a vektornak ugyanolyan típusúnak kell lennie, mint az x vektornak.
ordered	Logikai érték. Annak meghatározására, hogy a levels rendezve legyenek.

Most létrehozunk egy vektort és azt faktorrá alakítjuk:

```
> a <- rep(c(1, 2, 3), 3)
> a
```

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

```
> r <- factor(a)
```

Vessünk egy pillantást az eredményül kapott r faktor belső szerkezetére, a fix(r) utasítással:

```
structure(as.integer(c(1, 2, 3, 1, 2, 3, 1, 2, 3)), .Label = c("1", "2", "3"),
class = "factor")
```

```
> r <- factor(x, levels = 1:5)
> r
```

```
[1] 1 2 3 1 2 3 1 2 3
Levels: 1 2 3 4 5
```

```
> r <- factor(a, labels = c("a", "b", "c"))
> r
```

```
[1] a b c a b c a b c
Levels: a b c
```

```
> r <- factor(a, exclude = 3)
> r
```

```
[1] 1 2 <NA> 1 2 <NA> 1 2 <NA>
Levels: 1 2
```

A levels függvénnyel kiolvashatjuk a faktor szintjeit:

```
> levels(r)
```

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

A labels utasítással kigyűjthetők a faktor lehetséges szintjei:

```
> labels(r)
```

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

Ezen értékek megegyeznek a seq(along=a) függvény eredményével:

```
> seq(along=a)
```

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

Tömb

A tömböket alkotják *numerikus*, *karakter*, *komplex* vagy *logikai* adattípusok. Ugyanazon tömbön belül *többféle típus* NEM használható. Az array függvény a forrásadatokból (data) létrehoz egy tömböt, ami a dim argumentum által meghatározott dimenziójú.

```
array(data = NA, dim = length(data), dimnames = NULL)
```

Az array függvény argumentumainak leírása:

<code>data</code>	A tömb feltöltésére szolgáló vektor. Ha nem adunk meg adatokat, akkor üres tömböt hoz létre
<code>dim</code>	Egy vagy több elemet tartalmazó egész szám vektor, aminek elemei a az egyes dimenziók maximális indexét adják meg.
<code>dimnames</code>	A dimenziók nevét adhatjuk meg ezzel a lista típusú argumentummal. Ha van neve az egyes dimenzióknak, akkor azon keresztül is lehet rájuk hivatkozni.

Amennyiben a forrásadat kevesebb elemből áll, mint amennyit a dimenziók meghatároznak, a függvény a hiányzó elemeket feltölti a forrásadatokból.

```
> a <- rep(c(1, 2, 3), 3)
> r <- array(data = a, dim = c(2, 4))
> r

      [,1] [,2] [,3] [,4]
[1,]    1    3    2    1
[2,]    2    1    3    2

> nevek <- list(c(1, 2), c("a", "b", "c", "d"))
> r <- array(data = a, dim = c(2, 4), dimnames = nevek)
> r

  a b c d
1 1 3 2 1
2 2 1 3 2
```

A többdimenziós tömbök „sík” kontingencia-táblázattá alakítására egyszerű eszköz a `ftable` függvény.

```
> ftable(Titanic, row.vars = 1:3)

      Survived No Yes
Class Sex  Age
1st  Male  Child    0  5
      Adult  118 57
      Female Child    0  1
      Adult   4 140
2nd  Male  Child    0  11
      Adult  154 14
      Female Child    0  13
      Adult   13 80
3rd  Male  Child   35 13
      Adult  387 75
      Female Child   17 14
      Adult   89 76
Crew Male  Child    0  0
      Adult  670 192
      Female Child    0  0
      Adult    3 20
```

Mátrix

A mátrixokat alkotják *numerikus, karakter, komplex* vagy *logikai* adattípusok. Ugyanazon mátrixon belül *többféle típus* NEM használható. A `matrix` függvénnyel lehet létrehozni mátrixot, ami tulajdonképpen egy kétdimenziós vektor.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

A `matrix` függvény argumentumainak leírása:

<code>data</code>	Az adatokat tartalmazó vektor. Ha nem adjuk meg, akkor egy üres mátrixot hoz létre.
<code>nrow</code>	A sorok számát adhatjuk meg vele. Rövidítése <code>nr</code> .

<code>ncol</code>	Az oszlopok számát adhatjuk meg segítségével. Rövidítve <code>nc</code> .
<code>byrow</code>	Ha az értéke az alapértelmezett <code>FALSE</code> , akkor oszlopfolytonosan, egyébként pedig sorfolytonosan tölti fel a mátrixot adatokkal.
<code>dimnames</code>	Egy listában a dimenziók nevét adhatjuk meg, hasonlóan a tömbhöz, itt viszont csak két dimenzió van.

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)
> m
```

```
      [,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 feltöltődni.

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

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

Az oszlopok és sorok neveit a `dimnames` argumentum segítségével határozhatjuk meg, amit a tömbnél látottak szerint listaként kell megadni.

Mátrixot az `array` függvénnyel is létrehozhatunk. További mátrix-képzési lehetőség, hogy egy vektorból hozunk létre mátrixot a `dim` függvény segítségével:

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

[1] 1 2 3 4 5 6

> dim(a)

NULL

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

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

data.frame

A `data.frame`-eket alkotják *numerikus, karakter, komplex* vagy *logikai* adattípusok. Ugyanazon `data.frame`-en belül *HASZNÁLHATÓ többféle típus*. A *data.frame* olyan adattábla, aminek alkotó oszlopai vektorként foghatók fel. Fájlból beolvasott adattáblák eredményei általában ilyen objektumként jelennek meg, de létrehozhatjuk a `data.frame` függvénnyel is.

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE)
```

A `data.frame` függvény argumentumainak leírása:

```
...
```

Vagy csak az értékeket adjuk meg, ilyenkor ha azok rendelkeznek névvel, akkor a táblázat mezőnevei „öröklök” ezeket. Vagy névvel adjuk meg az értékeket, ilyenkor ez e név fog szerepelni a táblázat fejlécében.

<code>row.names</code>	Segítségével a sorok neveit adhatjuk meg. Ha egyetlen értéként adjuk meg, akkor ezzel azt határozzuk meg, hogy melyik oszlop tartalmazza azokat az értékeket, amelyeket a sorok elnevezésére szánunk. Az adott oszlopot megadhatjuk a sor-számával, illetve a nevével is. Ha vektorként adjuk meg az értékét, akkor annak hossza meg kell hogy egyezzen a sorok számával. Az alapértelmezett érték <code>NULL</code> .
<code>check.rows</code>	Ha az értéke <code>TRUE</code> , akkor ellenőrzi, hogy a sorok hosszának és elnevezéseinek egyezőségét.
<code>check.names</code>	Az alapértelmezett <code>TRUE</code> érték mellett ellenőrzi a mezőneveket: megfelelnek-e a változók elnevezési szabályainak, illetve, hogy nincsenek-e duplumok.

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)
> r
```

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

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

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

Ha viszont a hosszabb nem osztható a rövidebbel, 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. Az adattáblába beépülő vektorok oszlopok lesznek, amiknek a neve alapértelmezésben a vektor neve lesz (ezt módosíthatjuk).

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

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

A `row.names` argumentum segítségével a sorokat is elnevezhetjük, a bemeneti objektum vektorként adandó meg, és a hosszának meg kell egyeznie a táblázat oszlopainak hosszával.

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

```
  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
```

Lista

A listákat alkotják *numerikus, karakter, komplex, logikai* adattípusok, illetve *függvény* és *kifejezés*. Ugyanazon listán belül *többféle típus* HASZNÁLHATÓ. A listát a *data.frame*-hoz hasonlóan hozhatjuk létre a `list` függvénnyel. Általában azt mondhatjuk, hogy semmilyen megkötés nincsen az alkotóelemekkel kapcsolatban. Nem számít, hogy az egyes építőelemek (vektorok, listák, mátrixok stb.) milyen méretűek. Azt viszont érdemes megjegyezni, hogy az alkotóelemek nevét nem építi be automatikusan a `list` függvény a listába.

```
list(...)
```

A `list` függvény argumentumainak leírása:

```
...                Objektumok, bármilyen.
```

```
> lista1 <- list(x, y)
> lista2 <- list(A = x, B = y)
> lista1
```

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

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

```
> lista2
```

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

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

```
> names(lista1)
```

```
NULL
```

```
> names(lista2)
```

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

Idősor

A idősorokat alkotják *numerikus, karakter, komplex* vagy *logikai* adattípusok. Ugyanazon idősoron belül HASZNÁLHATÓ *többféle típus*. A `ts` függvény segítségével vektorból vagy mátrixból hozhatunk létre egy *idősor* objektumot. A függvény beállítási lehetőségei a következők:

```
ts(data = NA, start = 1, end = numeric(0), frequency = 1,
    deltat = 1, ts.eps = getOption("ts.eps"), class = , names = )
```

<code>data</code>	numerikus vektor vagy mátrix. Amennyiben <code>data.frame</code> a bemenő adat, akkor a <code>data.matrix</code> függvénnyel numerikus függvénné alakítja a <code>ts</code> függvény
<code>start</code>	az első megfigyelés időpontja. Akár egy egész szám, akár egy két számból álló vektor, amely megad egy természetes időegységet és egy 1-gyel kezdődő mintaszámot
<code>end</code>	az utolsó megfigyelés időpontja, <code>start</code> -hoz hasonlóan.
<code>frequency</code>	az időegységen belüli megfigyelések száma.
<code>deltat</code>	két megfigyelési időpont közti mintavételi része (pl.: 1/12 a havonkénti adatokhoz). Vagy csak a <code>frequency</code> , vagy csak a <code>deltat</code> adható meg.
<code>ts.eps</code>	az idősor összehasonlítási toleranciája. A gyakoriságok egyenlőnek tekintendők, ha az abszolút különbségeik kisebbek, mint <code>ts.eps</code> értéke.
<code>class</code>	az eredményhez rendelt osztály. Az alapértelmezett érték <code>ts</code> egy egyszerű idősorhoz, vagy <code>c("mts", "ts")</code> többszörös idősorhoz.
<code>names</code>	karaktervektor, ami a többszörös idősorok neveit adja meg, az alapértéke a <code>data</code> oszlopnevei vagy „ <i>Series 1</i> ”, „ <i>Series 2</i> ”, ...

Néhány példa a `ts` függvény paraméterezésére:

```
> ts(1:10, start = 1974)
```

```
Time Series:
Start = 1974
End = 1983
Frequency = 1
 [1] 1 2 3 4 5 6 7 8 9 10
```

```
> ts(1:20, start = c(1974, 8), frequency = 4)
```

```
      Qtr1 Qtr2 Qtr3 Qtr4
1975          1
1976     2    3    4    5
1977     6    7    8    9
1978    10   11   12   13
1979    14   15   16   17
1980    18   19   20
```

```
> ts(1:20, start = c(1974, 8), frequency = 12)
```

```
      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1974          1  2  3  4  5
1975     6  7  8  9 10 11 12 13 14 15 16 17
1976    18 19 20
```

Hivatkozás az objektumok elemeire

Indexelés

Az indexelési rendszer nagyon rugalmas és hatékony eszköz az egyes adattároló objektumok elemeinek kiolvasására, akár numerikus, akár logikai adattípusokról van szó. Az indexeket az objektum után írt szögletes zárójellel adjuk meg.

AZ INDEXELÉS NEM 0-RÓL HANEM 1-RÓL INDUL!!!

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

```
[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 tehetjük meg. Az `x` mátrixból egy elemet az `x[i, j]` utasítással olvashatunk ki, ahol *i* a mátrix sorát, *j* pedig a sorát 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
```

A mátrixhoz hasonlóan a tömböknél is használható az idexekkel való olvasás, a dimenziók szerint bővítve az indexek számát. Pl. egy háromdimenziós tömb esetén egy elemre az `x[i,j,k]` hivatkozhatunk.

Az indexek segítségével nem csak kiolvashatunk értékeket a tömbökből, hanem lekérdezéseket is végezhetünk az objektumokból, illetve azok elemeit is módosíthatjuk.

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

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

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

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

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

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

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

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

Ahogy látható a példából, az objektumból eltávolíthatunk elemeket, sorokat, oszlopokat. Az objektumok elemei közül lekérdezhethetjük a bizonyos feltételeknek megfelelőket.

```
> 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 idexeket gyűjtöttük ki. Az egyes feltételeknek megfelelő elemeket felül is írhatjuk.

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

```
> x
```



```

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

```

A `data.frame`-eken hasonlóan hajthatjuk végre a lekérdezéseket. A listák esetében az indexek többretegűek 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)
> lista

```

```

[[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 *gyökér-elemeire* dupla szögletes zárójelek közé zárt indexszel hivatkozhatunk. Az első gyökér-elem egy mátrix, annak az első oszlopát a következő módon hivatkozhatjuk:

```

> lista[[1]][, 1]

[1] 1 2 3

```

A lista harmadik gyökér-eleme 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.

Névvel való hivatkozás

A nevek attribútumok, amelyek több fajtája is lehet (nevek, oszlopnevek, sornevek, dimenziónevek). Többek között arra is alkalmasak, hogy objektumok elemeire hivatkozhassunk. 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 objektumnak. 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 névvel való hivatkozás során az objektum nevének és a \$ 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 újabbára való kapcsolódás az előzőről való automatikus lekapcsolást is jelent. A lekapcsolásra használhatjuk a `detach` függvényt is.

Objektumok konvertálása

Az R nagyszámú átalakítási lehetőséggel rendelkezik a különböző objektumtípusok között. Az átalakítások előtt célszerű tájékozódni: milyen típusú objektummal állunk szemben. Erre használható az `is.valami()` függvény, aminél a `valami` az objektum-típust jelenti (pl. `is.list`, `is.matrix`, `is.data.frame`, stb.). A típusok közötti konverziót az `as.valami` függvénnyel valósíthatjuk meg. Az egyes adattípusok közötti átalakítások eredményei a ?? táblázatban láthatók.

```
> faktor <- factor(c(1, 10))
> faktor
[1] 1 10
Levels: 1 10
> as.numeric(faktor)
[1] 1 2
```

Ha egy faktort úgy akarunk numerikussá konvertálni, hogy megmaradjanak a szintjei, először karakterré kell alakítanunk és azután numerikussá:

```
> as.numeric(as.character(faktor))
[1] 1 10
```

Kifejezés

A kifejezés (`expression`) az objektumok között alapvető jelentőségű az R-környezetben. Egy kifejezés tulajdonképpen egy karaktorsorozat, amit az R értelmez. Minden értelmezhető utasítás *kifejezés*. Amikor egy utasítást közvetlenül az R-terminálba írunk be, az értelmeződik, amennyiben a szabályoknak megfelelő. Sokszor hasznos lehet egy kifejezés előállításának anélkül, hogy értelmeztetnénk. Az `expression` függvény ezt teszi lehetővé. A létrehozott kifejezést az `eval()` függvénnyel tudjuk végrehajtani.

```
expression(...)
```

```

> x <- 3
> y <- 2.5
> z <- 1
> kifejezes1 <- expression(x/(y + exp(z)))
> kifejezes1

expression(x/(y + exp(z)))
> eval(kifejezes1)

[1] 0.5749019

```

A kifejezéseket többek között használhatjuk arra is, hogy függvényeket jelenítsünk meg ábráinkon. Néhány függvény használható a kifejezéseken, mint argumentum. Így például a `D()` a *parciális deriváltat* eredményez:

```

> D(kifejezes1, "x")
1/(y + exp(z))
> D(kifejezes1, "y")
-(x/(y + exp(z))^2)
> D(kifejezes1, "z")
-(x * exp(z)/(y + exp(z))^2)

```

Objektumok szerkesztése

Objektumok diagnosztikája

Az adattárolásra szolgáló objektumok tulajdonságainak megismerése, kiírása gyakran hasznos lehet. Néhány, erre szolgáló függvény bemutatása következik.

length

A `length` függvény segítségével az objektum „hosszát”, elemszámát olvashatjuk ki. A függvény az egyes objektumoknál eltérő elemeket olvas. Vektorok, tömbök és mátrixok esetében az objektumot alkotó értékek darabszámát, listák esetén a listát alkotó „gyökér-elemek” számát adja meg. A `data.frame`-nél pedig az oszlopok számát jelenti.

summary

A `summary` függvény az egyes objektumok összesítő leíró adatait adja vissza.

```

summary(object, ...)

> summary(valtozo.lista)

  Length Class  Mode
[1,] 5      -none- character
[2,] 3      -none- numeric
[3,] 2      -none- numeric

```

Ahogy példánk mutatja a `valtozo.lista` leíró adatai közül kiolvasható, hogy az egyes vektorok milyen hosszúak, illetve milyen *módúak*.

str

Az `str` függvény teljesen részletes képet ad az adott R-objektum *struktúrájáról*. A `summary` függvényhez képest alternatív diagnosztikai eljárásként használható.

```

> str(valtozo.lista)

List of 3
 $ : chr [1:5] "y" "x" "c" "v" ...
 $ : num [1:3] 1 2 3
 $ : num [1:2] 1.2 2.3

```

A `str` az előző függvényhez (`summary`) képest kiírja az objektum típusát is és az egyes vektorok első elemeit is.

edit

Az `edit` függvény egy szövegszerkesztőt vagy a `data.entry`-t hívja meg az adott R-objektum szerkesztésére.

```
edit(name = NULL, file = "", title = NULL,
     editor = getOption("editor"), ...)

vi(name = NULL, file = "")
emacs(name = NULL, file = "")
pico(name = NULL, file = "")
xemacs(name = NULL, file = "")
xedit(name = NULL, file = "")
```

Az `edit` függvény argumentumainak leírása:

<code>name</code>	A szerkeszteni kívánt és nevesített R-objektum neve. Ha nincs megadva, akkor a <code>file</code> által meghatározott objektum lesz megnyitva szerkesztésre.
<code>file</code>	Egy fájlnev, amelybe a szerkesztett változat ki lesz írva.
<code>title</code>	A szerkesztőben címként megjelenő felirat.
<code>editor</code>	Meghatározhatju, hogy mely szövegszerkesztőt hívja meg az R. Windowson az alapértelmezés a <i>notepad</i> . Megadható más szerkesztő is, de természetesen csak akkor fog hiba nélkül működni, ha telepítettük a rendszeren (pl. <i>Tinn-R</i>).
<code>...</code>	További argumentumokat adhatunk meg más eljárásokba, vagy azokhoz.

fix

A `fix` függvény az `edit` függvényt hívja meg az adott objektum szerkesztésére, azonban (az `edit`-tel ellentétben) a változásokat *el is menti az objektumban*.

```
fix(x, ...)

x          A szerkesztendő R-objektum.
...       Az edit-nél használható további argumentumok.
```

Data Editor

Az előző adatbeviteli lehetőségek mellett, még az is lehetséges, hogy a `data.entry`, a `dataentry`, illetve a `de` függvények segítségével *grafikus felületen keresztül* töltsünk fel adattároló objektumokat adatokkal.

```
data.entry(..., Modes = NULL, Names = NULL)
dataentry(data, modes)
de(..., Modes = list(), Names = NULL)
```

A `data.entry`, a `dataentry` és `de` függvények argumentumainak leírása:

<code>...</code>	Változók listája. Jelenleg numerikusnak, vagy karakternek kell lennie, vagy ezekből álló listának.
<code>Modes</code>	A változóknak megfelelő <i>módok</i> .
<code>Names</code>	A változókhoz használt nevek.
<code>data</code>	Numerikus és/vagy karakter vektorokból álló lista.
<code>modes</code>	A <code>data</code> hosszának megfelelő lista, ami megadja a változók <i>módját</i> .

A későbbi függvények bemutatásához létrehozunk néhány adattároló objektumot.

```
> i <- c('y','x','c','v','b')
> j <- c(1,2,3)
> k <- c(1.2,2.3)
> valtozo.lista <- list(i,j,k)
```

Adatbevitelhez a `valtozo.lista` lista formátumú objektumába gépeljük be a következőt:

```
> de(valtozo.lista, Names=c('i','j','k'))
```

Ha a megjelenő táblázatban beírunk a `j` oszlopba egy új értéket, mondjuk 4-et, akkor az alábbi lista íródik ki a terminálba:

```
$valtozo.lista
$valtozo.lista$i
[1] "y" "x" "c" "v" "b"
```

```
$valtozo.lista$j
[1] 1 2 3 4
```

```
$valtozo.lista$k
[1] 1.2 2.3
```

Habár a terminálban megjelenik a `j` vektor új eleme, a 4, a `valtozo.lista` objektum nem változott meg. Ezt a következő diagnosztikai eljárással lehet ellenőrizni:

```
> str(valtozo.lista)
List of 3
 $ : chr [1:5] "y" "x" "c" "v" ...
 $ : num [1:3] 1 2 3
 $ : num [1:2] 1.2 2.3
```

Most próbáljuk ki a `de` helyett a `data.entry` függvényt az új érték beírásához:

```
> data.entry(valtozo.lista, Names=c('i','j','k'))
```

```
$valtozo.lista
$valtozo.lista$i
[1] "y" "x" "c" "v" "b"
```

```
$valtozo.lista$j
[1] 1 2 3 4
```

```
$valtozo.lista$k
[1] 1.2 2.3
```

Az `str` függvénnyel ellenőrizzük, hogy a `valtozo.lista` objektumban történt-e változás.

```
> str(valtozo.lista)
List of 3
 $ i: chr [1:5] "y" "x" "c" "v" ...
 $ j: num [1:4] 1 2 3 4
 $ k: num [1:2] 1.2 2.3
```

Látható, hogy (az előző példával ellentétben) a beírt új érték bekerül a `valtozo.lista` objektumba. A függvény-csoport harmadik tagja a `dataentry`, aminél a lista formájú adatok mellett mindenképpen meg kell határozni (ugyancsak lista formájában) az egyes vektorok *módját* is.

```
> dataentry(valtozo.lista,list('character','numeric','numeric'))
```

Az előzőekhez képest a megjelenő táblázat mezőfeliratait: `var0`, `var1`, `var2`. Ha egy új értéket adunk a `var1` oszlophoz, a táblázat bezárása után a terminálba (az alábbiak szerint) kiíródik:

```
$var0
[1] "y" "x" "c" "v" "b"
```

```
$var1
[1] 1 2 3 4
```

```
$var2
[1] 1.2 2.3
```

Az `str` függvénnyel ellenőrizzük, hogy a `valtozo.lista` objektumban történt-e változás.

```
> str(valtozo.lista)
List of 3
 $ i: chr [1:5] "y" "x" "c" "v" ...
 $ j: num [1:4] 1 2 3
 $ k: num [1:2] 1.2 2.3
```

Vagyis az eredmény hasonló, mint a `de` esetében, az objektum nem változott meg.

5. 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

Objektum-műveletek

Aritmetikai műveletek

Ha a vektorokon végezzük a klasszikus aritmetikai műveleteket (5. táblázat), fontos, hogy figyeljünk néhány specialitásra:

```
> x <- 1:4
> x + 3
```

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

Az x vektor minden eleméhez hozzáadott 3-at az utasítás.

```
> x <- 1:4
> y <- rep(1, 4)
> z <- x + y
> z
```

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

Két egyenlő hosszúságú vektort adtunk össze.

```
> x <- 1:4
> y <- 1:2
> z <- x + y
> z
```

```
[1] 2 4 4 6
```

Két különböző hosszúságú vektor esetén akkor hajtható végre valamilyen aritmetikai művelet, ha a rövidebb vektor elemeinek számával osztható a hosszabb vektor elemeinek a száma (mint előző példánkban). Ebben az esetben az R a rövidebb vektort addig ismétli, amíg annak a hossza el nem éri a hosszabb vektor hosszát. Amennyiben az oszthatóság feltétele nem teljesül, a feladatot ugyan végrehajtja, de figyelmeztetést kapunk:

```
> x<-1:3
> y<-1:2
> z<-x+y
Warning message:
longer object length
      is not a multiple of shorter object length in: x + y
> z
[1] 2 4 4
```

Gyakrabban használt függvények

subset

Segítségével vektorokból vagy data.frame-okból válogathatunk le részeket, általunk meghatározott szempontok szerint.

```
subset(x, ...)

## Default S3 method:
subset(x, subset, ...)

## S3 method for class 'data.frame':
subset(x, subset, select, drop = FALSE, ...)
```

A `subset` függvény argumentumainak leírása:

<code>x</code>	Az adatobjektum, amiből a leválogatást végeznénk.
<code>subset</code>	Logikai kifejezés.
<code>select</code>	E kifejezés meghatározza, hogy mely oszlopok adatait válogassa le a függvény.
<code>drop</code>	Ha <code>TRUE</code> , akkor a lehető legalacsonyabb dimenzióknak megfelelően fogja össze az eredményt.
<code>...</code>	További argumentumok.

Példák:

```
> a <- 1:20
> subset(a, a > 10)

[1] 11 12 13 14 15 16 17 18 19 20
```

Az `a` vektorból leválogattuk a 10-nél nagyobb értékű elemeket. A továbbiakban a `data.frame`-et használó példákban az `airquality` adatállományt fogjuk használni. Az alaptáblázat 6 oszlopból és 153 rekordból áll:

```
> dim(airquality)

[1] 153  6
```

Az alábbi példában az látható, hogy két szempont szerint végzünk szűrést: a `Temp` oszlop tartalma nagyobb, mint 80, valamint a `Month` oszlop értéke 9. Az eredménytáblázatban csak azok a sorok jelennek meg, amelyekre ez a két feltétel igaz. A kiindulási hat oszlop helyett az eredménytáblázatban csak két oszlop lesz (`Ozone`, `Wind`).

```
> lekerdezes <- subset(airquality, Temp > 80 & Month==9, select = c(Ozone, Wind))
> dim(lekerdezes)

[1] 9 2
```

A dimenziók lekérdezése után láthatjuk, hogy csak 9 rekord felelt meg a feltételeknek.

split és unsplit

A függvény egy faktorban megadott értékek szerint az adott vektort vagy `data.frame`-ot szétválogatja, illetve összeilleszt ilyen módon létrejött listákat.

```
split(x, f)
split(x, f) <- value
unsplit(value, f)
```

A `split` és `unsplit` függvények argumentumainak leírása:

<code>x</code>	A feldarabolandó vektor vagy <code>data.frame</code> .
<code>f</code>	A csoportokat meghatározó faktor, de lehet faktorokból álló lista is.
<code>value</code>	Vektorokból vagy <code>data.frame</code> -okból álló lista, ami kompatibilis az <code>x</code> -el. Ha a hosszúságok nem egyezők, akkor a <i>recycling</i> lép működésbe.

sort.list

Segítségével növekvő vagy csökkenő sorrendbe lehet rendezni adatokat, illetve sorbarendezhetünk táblázatokat is, úgy, hogy a sorok egyben maradnak.

```
sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE,
         method = c("shell", "quick", "radix"))
```

A `sort.list` függvény argumentumainak leírása:

<code>x</code>	Vektor.
<code>partial</code>	Részleges rendezéshez használt elemek indexeinek vektora.
<code>decreasing</code>	Logikai érték, ami ha <code>TRUE</code> , akkor csökkenő, ha <code>FALSE</code> , akkor növekvő sorba rendezi az adatokat.
<code>na.last</code>	A hiányzó értékek NA kezelését meghatározó argumentum. Ha <code>TRUE</code> , akkor a hiányzó értékek a sor végére, ha <code>FALSE</code> , akkor az elejére kerülnek. Ha az értéke <code>NA</code> , akkor a hiányzó értékeket eltávolítja.
<code>method</code>	A részleges rendezés módszere.

Példák: A korábban előállított `lekerdezes` táblázat rekordjai nem rendezettek:

6. táblázat. Mátrix-függvények

```

%%
crossprod
diag
dim, ncol, nrow
dimnames
eigen
kappa
qr
solve
svd
t
upper.tri, lower.tri

```

```
> lekerdezes
```

```

      Ozone Wind
124    96  6.9
125    78  5.1
126    73  2.8
127    91  4.6
128    47  7.4
129    32 15.5
134    44 14.9
143    16  8.0
146    36 10.3

```

Az alábbi példában az Ozone oszlop alapján növekvő sorrendbe rendezzük a táblát:

```
> lekerdezes[sort.list(lekerdezes$Ozone),]
```

```

      Ozone Wind
143    16  8.0
129    32 15.5
146    36 10.3
134    44 14.9
128    47  7.4
126    73  2.8
125    78  5.1
127    91  4.6
124    96  6.9

```


Adatok olvasása, kezelése és írása

Munkakönyvtár

Ha adatállományokkal dolgozunk, sokszor fájlkból 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")
```

Ahogy a példából is látszik az út megadásánál (akár Windows, akár Linux környezetben dolgozunk) a könyvtárak elválasztására a `/` jelet MUSZÁJ használni. Ez Linuxon nem jelent újdonságot, viszont DOS, illetve Windows esetén az elérési utak megadásánál az elválasztóként a `\` jelet használják.

AMIKOR AZ R-BEN AKARUNK MEGADNI FÁJLELÉRÉSI ÚTVONALAT,
AKKOR CSAK A / JELET HASZNÁLHATJUK!

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` függvénnyel végezhetjük el.

```
> getwd()
```

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

Adatok olvasása

Microsoft Excel állományok olvasása

Annak ellenére, hogy a Microsoft Excel adattárolási formátum széles körben elterjedt az R alapsomag 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 telepítsünk a számítógépünkön. Az első példában bemutatott megoldáshoz szükséges, hogy mielőtt lefuttatjuk, létrehozzunk egy ODBC-kapcsolatot (a példában `ODBCexcel` elnevezésűt). 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 szögletes zárójeleknek a fenti példában megadott szintaxis szerint jelen kell lennie.

A második megoldás szintaktikailag egyszerűbben adja ugyanazt az eredményt.

FONTOS MEGJEGYZENI, HOGY AZ ODBC-KAPCSOLATON KERESZTÜL AZ EXCEL TÁBLÁZATOK NEM MÓDOSÍTHATÓK, CSAK OLVASHATÓK!

A gregmisc könyvtár read.xls függvényének segítségével

Ahhoz, hogy ezt a függvényt tudjuk használni, nem elegendő a *gregmisc* csomagot telepíteni, szükség van arra, hogy *Perl* is legyen telepítve gépünkön. Az *ActivePerl*¹³ telepítése után a gépünkön lesz egy használható *Perl*.

```
> library(gregmisc)
> adat <- read.xls("d:/excel.xls", 1, perl="C:/perl/bin/perl.exe")
```

A függvény első argumentumával megadjuk az adott excel fájlt, a másodikkal a *munkalap* sorszámát, a harmadikkal pedig a *perl.exe* elérési útvonalát határozzuk meg.

Excel-állomány CSV-formátumba alakítása

Ahogy a későbbiekben látni fogjuk, az R több függvény segítségével is képes a *comma separated value* (.csv) állományok olvasására. Így az Excel-állományok használatának az egyik lehetősége az, ha átalakítjuk .csv állománnyá.

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

A *xls2csv*¹⁴ alkalmazás segítségével szintén elvégezhetjük az állomány átalakítását. Mivel nem kell telepíteni, csupán a tömörített állományt kell kicsomagolni, olyan gépeken is használható, amin nincsen telepítési jogosultságunk. A következő kóddal (DOS) egy .xls állományt alakíthatunk át .csv fájlá.

```
D:\catdoc-0.94>xls2csv.exe -q 1 -c ; d:\excel.xls > d:\excel.csv
```

Az itt látható paraméterezésnél több is lehetséges, de az R-hez való átalakításnak ez is teljesen megfelel. A *-q* után álló *1* azt jelenti, hogy csak a szöveges cellák lesznek idézőjelek közé foglalva. A *-c* után álló *;* az oszlopokat elválasztó karakter megadására szolgál.

A foreign könyvtár adatállomány-kezelő függvényei

A *foreign* könyvtár függvényei lehetőséget adnak arra, hogy néhány statisztikai szoftver csomagok adatformátumait olvashassuk, illetve írassuk.

Adatok olvasása ASCII állományokból

readLines

Szöveges állományokból soronként olvashatunk ki adatokat a *readLines* függvény segítségével.

```
readLines(con = stdin(), n = -1, ok = TRUE)
```

A *con* argumentumban egy fájlt adunk meg. Az *n* segítségével adhatjuk meg, hogy hány sort olvasson be a megadott fájlból a függvény. Ha *n* értéke az alapértelmezett *-1*, akkor a teljes szöveges állományt beolvassa. A harmadik *ok* argumentumot amennyiben *n* 0-nál kisebb mindenképpen az alapértelmezett *TRUE*-ra kell állítani, különben hibát generál a függvény.

¹³<http://www.perl.com/download.csp>

¹⁴<http://www.45.free.net/~vitus/ice/catdoc/#download>

7. táblázat. Foreign csomag függvények

függvény	rövid leírás
<code>data.restore</code>	S3 bináris állományt olvas
<code>lookup.xport</code>	SAS XPORT formátumú könyvtárból olvas ki információkat
<code>read.dbf</code>	DBF állományt olvas
<code>read.dta</code>	Stata bináris állományt olvas
<code>read.epiinfo</code>	Epi Info adatállományt olvas
<code>read.mtp</code>	Minitab Portable Worksheet-et olvas
<code>read.octave</code>	Octave szöveges adatállományt olvas
<code>read.S</code>	S3 bináris állományt olvas
<code>read.spss</code>	SPSS adatállományt olvas
<code>read.ssd</code>	a <code>read.xport</code> segítségével egy táblát olvas ki SAS Permanent Dataset-ből
<code>read.systat</code>	egy táblát olvas ki a Systat File-ből
<code>read.xport</code>	SAS XPORT formátumú könyvtárat olvas
<code>write.dbf</code>	DBF állományt ír
<code>write.dta</code>	Stata bináris formátumú állományt ír
<code>write.foreign</code>	táblázatot ír ki más statisztikai eszköz számára olvasható formában

Karakterhatárolt állományok

Karakterhatárolt állománynak nevezem azokat az ASCII állományokat, amelyek adatokat karakter határolt értékek¹⁵ formájában tárolnak (*csv*). A karakterhatárolt állományok R-be való beolvasását leginkább a `read.table` függvénnyel, illetve származékaival valósíthatjuk meg. Ezek paraméterezése látható az alábbiakban.

```
read.table(file, header = FALSE, sep = "", quote = "\"", dec = ".",
           row.names, col.names, as.is = FALSE, na.strings = "NA",
           colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#")

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".", fill = TRUE, ...)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=";", fill = TRUE, ...)

read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".", fill = TRUE, ...)

read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=";", fill = TRUE, ...)
```

A `read.csv` függvény lényegében abban különbözik a `read.table`-tól, hogy az alapértelmezett mezőelválasztó a vessző (`,`). A `read.csv2` függvénynél a mezőelválasztó a pontosvessző (`;`), a tizedeseket elválasztó jel pedig *nem pont* (`.`) hanem *vessző* (`,`). A `read.delim` függvénynél a mezőelválasztó a `\t` vagyis tabulátor, a tizedeseket elválasztó jel pedig *pont* (`.`). A `read.delim2` függvénynél a mezőelválasztó ugyancsak `\t`, viszont a tizedeseket elválasztó jel *nem pont* (`.`) hanem *vessző* (`,`). Az egyes függvények közötti argumentumbeállítási eltéréseket a 8. táblázat mutatja.

8. táblázat. A `read.table` függvénycsoport különbségei

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

A `read.table` függvény argumentumainak leírása:

file A beolvasandó fájl neve. Ha nem állítottuk be a munkakönyvtárként azt a könyvtárat, ami tartalmazza az adott fájlt, akkor a teljes utat meg kell adnunk.

¹⁵<http://gisfgyelo.geocentrum.hu/kisokos/kisokos.csv.html>

<code>header</code>	Ha az alapértelmezett <code>FALSE</code> értékre van állítva, akkor a táblázat első sorát nem fejlécként, hanem első adatsorként olvassa be. <code>TRUE</code> esetén viszont a táblázatunk első sorát fejlécként olvassa be.
<code>sep</code>	Az egyes mezőket elválasztó karaktert határozhatjuk meg. Az alapértelmezett <code>" "</code> (<i>white space</i>) határoló mezőként értelmezi az egy vagy több <i>szóközt</i> , a <i>tabo(ka)t</i> , vagy az <i>új sorokat</i> .
<code>quote</code>	A szöveges mezők jelzésére szolgáló karaktert így adhatjuk meg. Az alapértelmezett érték a <code>\\" jelsor. Ez a beállítás azt jelenti, hogy akár <code>"</code>, akár <code>'</code> jelek fogják közre a szövegeket a táblában, a függvény a beolvasott táblázatban szöveggként, de a jelek nélkül fogja tárolni azokat.</code>
<code>dec</code>	A lebegőpontos értékeket tartalmazó mezők beolvasánál tizedesjelként értelmezendő jelet határozza meg. Az alapértelmezés a pont <code>.</code> .
<code>row.names</code>	A sorok neveit határozhatjuk meg vele. Megadhatjuk többféleképpen is. Az egyik lehetőség, hogy egy vektorban adjuk meg a sorneveket, ebben az esetben figyelni kell arra, hogy a vektor hossza megegyezzen a sorok számával. A sorneveket úgy is meghatározhatjuk, hogy megadjuk a táblázatnak azt az oszlopát, amelyik tartalmazza a neveket. Az adott oszlopot meghatározhatjuk egy számmal (ami az oszlop sorszáma), vagy az oszlop nevével. Ha nem állítjuk be a sorneveket, akkor egyszerűen automatikusan sorszámozva lesznek.
<code>col.names</code>	Az oszlopnevek megadására szolgál. Az oszlopok számának megfelelő hosszúságú vektor formájában adható meg. Ha <code>header</code> argumentumot <code>FALSE</code> -ra állítottuk, akkor alapértelmezésben az oszlopok nevei a <code>V</code> és az oszlop sorszámból jönnek létre.
<code>as.is</code>	A <code>read.table</code> függvény alapértelmezésben a szöveges mezőket faktorrá alakítja. Ez az argumentum lehetőséget nyújt az átalakítás kontrollálására. Az alapértelmezése <code>FALSE</code> . Ha <code>TRUE</code> -ra állítjuk, akkor a szöveges mezők szövegesként lesznek beolvasva és nem alakítódnak át faktorrá.
<code>na.strings</code>	Vektorként megadható listája azon értékeknek, melyek esetén a függvény hiányzó értéket kell, hogy beszúrjon a helyükre a végleges táblázatba. Alapértelmezett értéke <code>"NA"</code> .
<code>colClasses</code>	Lehetőséget nyújt arra, hogy az egyes mezők adattípusát megváltoztassuk a beolvasás során. Egy vektorban sorolhatjuk fel (az oszlopok sorrendjében) az átalakítás eredményeként várt típusokat. Ha valamelyik mezőn nem akarunk átalakítást végezni, akkor annak <code>NA</code> értéket adunk meg. Az alapértelmezett érték <code>NA</code> .
<code>nrows</code>	A beolvasandó sorok maximális számát határozhatjuk meg vele. Ha értéke negatív, akkor az egész táblát beolvassa a függvény. Alapértéke <code>-1</code> .
<code>skip</code>	Az állomány elején beolvasás nélkül „átugrandó” sorok száma. Alapértelmezése <code>0</code> .
<code>check.names</code>	Az alapértelmezett <code>TRUE</code> -érték mellett a mezőneveket ellenőrzi, hogy megfelelnek-e a változók elnevezési szabályainak.
<code>fill</code>	Ha ezt <code>TRUE</code> -ra állítjuk, akkor (ha van olyan sora forrásállománynak, ami kevesebb mezőt tartalmaz) a függvény feltölti üres cellákkal, a sor végére illetve azokat. Alapértelmezésben <code>!blank.lines.skip</code> .
<code>strip.white</code>	Ha a <code>sep</code> argumentumot beállítottuk, és ha ennek az értékét <code>TRUE</code> -ra állítjuk, akkor a szöveges mezők elején, illetve végén lévő szóközöket törli. Alapértelmezésben <code>FALSE</code> .
<code>blank.lines.skip</code>	Ha az alapértelmezett <code>TRUE</code> értékre van állítva, akkor a forrásfájlból nem olvassa be az üres sorokat, átugorja őket.
<code>comment.char</code>	A megjegyzéseket megelőző, jelölő karaktert határozhatjuk meg vele. Alapértelmezésben <code>#</code> .

Rögzített szélességű mezők

Olyan ASCII fájllokból is olvashatunk adatokat, amelyekben nem karakterek határolják el az egyes mezőket. A mezők szélessége ilyenkor rögzített karakterszámú. Ilyen feladat esetén a `read.fwf` függvény nyújt segítséget.

```
read.fwf(file, widths, header = FALSE, sep = "\t", as.is = FALSE,
        skip = 0, row.names, col.names, n = -1, buffersize = 2000,
        ...)
```

A `read.fwf` függvény `read.table` függvénytől *eltérő* argumentumainak leírása:

<code>widths</code>	Az egyes mezők méretét határozhatjuk meg segítségével. Amennyiben egy rekord egy sorban helyezkedik el, akkor egy vektorban kell megadnunk, a mezők hosszúságát meghatározó karakterhosszban. Ha a rekordjaink többsorosak, akkor listaként kell megadnunk ezt az argumentumot.
<code>sep</code>	Itt nem a forrásfájl belső mezőválasztó karaktert jelenti, sőt nem is szabad, hogy az itt megadott jel szerepeljen a forrásállományban. Tulajdonképpen belső használatra szolgáló, szeparáló karakter.
<code>n</code>	Megyegezzik a <code>read.table</code> függvény <code>nrows</code> argumentumával.
<code>bufferize</code>	Az egyszerre beolvasandó sorok számának beállítására szolgál.
<code>...</code>	További <code>read.line</code> argumentumokat használhatunk, köztük a <code>na.strings</code> és <code>colClasses</code> függvényeket is.

scan

A `read.table` és a `read.fwf` függvények tulajdonképpen a `scan` függvényre épülnek, azonban ez utóbbi közvetlenül is használható. Míg a korábbi függvények visszatérési objektuma `data.frame`, addig a `scan` vektort vagy listát ad vissza.

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
      quote = if (sep=="\n") "" else "\"", dec = ".",
      skip = 0, nlines = 0, na.strings = "NA",
      flush = FALSE, fill = FALSE, strip.white = FALSE,
      quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,
      comment.char = "")
```

A `scan` függvény `read.fwf` és `read.table` függvényektől eltérő argumentumainak leírása:

<code>file</code>	Hasonlóan az előzőkhöz, a beolvasandó állományt adjuk meg vele. Ha azonban az értéke az alapértelmezett "", akkor a billentyűzetről olvassa be a begépett adatokat a meghatározott objektumba. A billentyűzetről való adatbevitel befejezését vagy egy új sor kezdésével, vagy egy EOF jel segítségével érhetjük el. Ez utóbbit Windowson <code>Ctrl-D</code> , Linuxon <code>Ctrl-Z</code> billentyűkombinációval adhatjuk meg.
<code>what</code>	A beolvasandó adatok típusát határozza meg. Ha listában adjuk meg, akkor úgy értelmezi a függvény, hogy a fájl sorai rekordok, és a listában meghatározott adattípusok sorrendben a „mezők”-nek felelnek meg. A támogatott típusok: <code>logical</code> , <code>integer</code> , <code>numeric</code> , <code>complex</code> , <code>character</code> , <code>raw</code> és <code>list</code> . A <code>list</code> olyan elemeket kell, hogy tartalmazzon, amelyek az előző hat típusnak, vagy <code>NULL</code> -nak felelnek meg.
<code>nmax</code>	A beolvasandó <i>adatok elemszámának</i> maximuma. Ha a <code>what</code> lista, akkor a maximálisan beolvasandó <i>rekordok</i> száma. Amennyiben nem pozitív értéként adjuk meg, a teljes adatállományt beolvassa.
<code>n</code>	A beolvasandó <i>adatok elemszámának</i> maximuma. Alapértelmezésben nincsen korlátozva.
<code>nlines</code>	Ha pozitív szám, akkor a maximálisan beolvasandó <i>sorok</i> számát határozza meg.
<code>flush</code>	Ha az értéke <code>TRUE</code> , akkor a függvény az utolsó mező olvasása után a sor végére ugrik. Ez lehetővé teszi, hogy a az utolsó mező után megjegyzéseket helyezhessünk el, és így kizárjuk azt, hogy egy sorban több mint egy rekord legyen.
<code>quiet</code>	Ha az értéke az alapértelmezett <code>FALSE</code> , akkor a függvény minden elem beolvasása után kiír egy sort a terminálba, jelezve azt, hogy hány elemet olvasott már be.
<code>multi.line</code>	A függvény csak akkor veszi figyelembe, ha a <code>what</code> lista. Ebben az esetben, ha <code>FALSE</code> -ra állítjuk, akkor minden rekord egy sorba lesz beillesztve.

dget

A `dput` függvénnyel kiírt objektum visszaolvasására használható függvény (a `dget(file)` szintaxis szerint), ahol a `file` az objektumot tartalmazó állomány.

Magyarítás

Előfordulhat, hogy munkánk során olyan ASCII-állományokat olvasunk be, amelyekben magyar ékezetes betűket tartalmazó karakterláncok, szavak fordulnak elő. Ilyenkor, ha a karakterek kódolási beállítása nem megfelelő, előfordulhat, hogy ezek az ékezetes karakterek a beolvasás után nem a várt alakban jelennek meg. Az aktuálisan működő kódolást a következő módon olvashatjuk ki:

```
> Sys.getlocale(category = "LC_CTYPE")
```

```
[1] "Hungarian_Hungary.1250"
```

Az R alapbeállításában a magyar nyelvhez az `Hungarian_Hungary.1250` kódolást használja, ami a korábbi példákban bemutatott `.csv` állomány beolvasásakor a következő eredményt adja:

```
> read.csv2('d:/excel.csv', header=T)
```

```
      a b c
1 '\366' 1 NA
2 '\341' 2 76
3 '\355' 3 23
4 '\365' 4 34
5   'û' 5 54
6 '\351' 6 60
```

Látható, hogy a betűk nagyrészt hibásan kódolta az R. Az alapkódolás helyett az 1251 vagy 1252 kódot használva karaktereink helyesen jelennek meg a terminálban. A beállítás a következő módon történik:

```
> Sys.setlocale("LC_CTYPE", "Hungarian_Hungary.1252")
```

```
> read.csv2('d:/excel.csv', header=T)
```

```
      a b c
1 'ö' 1 NA
2 'á' 2 76
3 'í' 3 23
4 'ó' 4 34
5 'ű' 5 54
6 'é' 6 60
```

A kódolás beállítását érdemes a munkafolyamat elején elvégezni, mert ha egyszer a helytelen beállítással elvégeztünk már a beolvasást, akkor az a kódolás működik a munkafolyamatban továbbra is.

Adatbázisok

Miért használjunk adatbázist?

Az R-ben az adatobjektumok a memóriában helyezkednek el és esetleg több változatban is jelen lehetnek egy munkafolyamatban. Mivel az objektumok a memóriában foglalnak helyet, ezért az R (jelenlegi formájában) a túl nagy adatállományok kezeléséhez nem a legjobb eszköz. Néhány száz megabájtos objektumok gyorsan okozhatnak memória-túlsordulást.

Az R nem támogatja az adatok konkurrens kezelését: ha több felhasználó dolgozik ugyanazzal az adatállománnyal, az egyik által létrehozott változtatás nem jelenik meg a másikonál.

Az adatbáziskezelő rendszerek (DBMS), és különösen a relációs DBMS-ek (RDBMS) ezen hiányosságokon képesek segíteni, főbb előnyeik a következők:

1. Gyors hozzáférés nagy adatbázisok egyes részeihez
2. Az adatbázison belül összesítő táblázatok és kereszttáblák létrehozására igen hatékony eszköz.
3. Az adatokat sokkal hatékonyabb formában lehet tárolni adatbázisokban, mint egyszerű táblázatokban vagy az R `data.frame` formátumában.
4. Amellett, hogy egyszerre több felhasználó férhet hozzá az adatbázisban tárolt adatokhoz, ez biztonságos kapcsolaton keresztül történhet, az illetéktelenek kizárásával.

Az adatbázisban tárolt adatokból az R-környezetbe nem kell teljes táblázatokot behívni és ezzel terhelni a memóriát. Az adatbázisban el lehet végezni bizonyos előmunkálatokat a felhasználandó adatokon és csak a statisztikai elemzésben valóban résztvevő vagy ábrázolandó adatok kerülnek az R-be, így erőforrásokat szabadítva fel.

Az R-hez többféle közvetlen interfészt fejlesztettek adatbázisok eléréséhez (pl. RMySQL, RPostgreSQL) ezek kisebb-nagyobb mértékben követik az új R-változatokat. Rugalmasabb adatbázis elérést tesz lehetővé az RODBAC csomag, ami ODBC -kapcsolaton keresztül tud adatbázisokat olvasni és írni. Az alábbiakban egy igen egyszerű megoldást mutatunk be, ahol egy *Microsoft Access* adatbázisból egy táblázatot olvasunk ki. Itt az ODBC-kapcsolat nem kíván meg azonosítást és jelszót – más kapcsolatoknál (pl. PostgreSQL¹⁶) ez nem nélkülözhető.

¹⁶<http://www.postgresql.org/>

```
> library(RODBC)
> db <- odbcConnect('adatbazisom')
> tablazat <- sqlQuery(db, 'SELECT * FROM d_virus_emission')
```

Látható, hogy az adatbázis megnyitása után, azon SQL lekérdezéseket lehet futtatni, a lekérdezés eredményeként visszatérő objektum `data.frame`.

SQLite

Az *SQLite*¹⁷ adatbázis-formátum nagy hordozhatóságot tesz lehetővé, mivel az adatbáziskezeléshez nem szükséges szerver. Nem túl nagy adatbázisok kezeléséhez hasznos formátum. Platformfüggetlen és ingyenes. Több teszt is azt bizonyította, hogy gyorsabb a *MySQL*¹⁸ illetve *PostgreSQL* szervereknél. Az SQLite-adatbázisok tervezéséhez, kezeléséhez remek grafikus felülettel rendelkező, platformfüggetlen ingyenes szoftver a *SQLite Database Browser*¹⁹. Ha valaki próbált már több tábla összekapcsolásával SQL-lekérdezéseket szerkeszteni, akkor tudja, hogy milyen nagy segítséget nyújthat egy grafikus SQL-szerkesztő. Az *Open Office 1.1*²⁰ verziójához letölthető egy kiegészítés²¹, aminek telepítése után az *Open Office Calc* alkalmazással kapcsolódni lehet *SQLite*-adatbázisokhoz és azokban könnyedén szerkeszthetünk grafikus felületen többtáblás lekérdezéseket. Sajnos jelenleg ez csak Linux alatt működik, de ígérik Windows alatt is működő verzióját is. Az alábbi kód egy SQLite adatbázisból SQL kód segítségével olvas ki egy táblázatot.

```
> library(RSQLite)
> meghajto <- dbDriver("SQLite")
> kapcsolat <- dbConnect(meghajto, dbname = "d:/vtr.db")
> eredmeny <- dbSendQuery(kapcsolat, "select * from alpha")
> adat <- fetch(res, n = -1)
```

Adatok kiírása

write

A megadott objektumot (x) ASCII-állományba írja ki. Általában mátrixokra használatos, amiket érdemes transzponálni a kiírás előtt.

```
write(x, file = "data",
      ncolumns = if(is.character(x)) 1 else 5,
      append = FALSE)
```

A `write` függvény argumentumainak leírása:

<code>x</code>	A kiírandó adat.
<code>files</code>	Vagy a célfájl megadó karakterlánc, vagy egy <i>kapcsolat</i> , amin keresztül kiíródik az adat. Lehet "" is az értéke, akkor a már korábban beállított <i>kapcsolatba</i> ír ki a függvény.
<code>ncolumns</code>	Lehetővé teszi az oszlopok számának meghatározását a kiírandó adatokban.
<code>append</code>	Ha TRUE értékre van állítva, akkor a <code>file</code> argumentumban megadott fájl tartalmához <i>hozzáfűzi</i> az adatokat, ha az alapértelmezett FALSE, akkor <i>felülírja</i> az állományt.

```
> x <- matrix(1:10, ncol=5)
> x <- t(x)
> write(x, "write-al_kirva.txt")
```

Ha `data.frame`-re használjuk, előtte érdemes átalakítanunk mátrixszá.

```
> adat <- read.table("tabla.txt")
> adat <- as.matrix(adat)
> adat <- t(adat)
> write(adat, "write-al_kirva.txt")
```

¹⁷<http://www.sqlite.org/>

¹⁸<http://www.mysql.com/>

¹⁹<http://sqlitebrowser.sourceforge.net/>

²⁰<http://www.openoffice.org/>

²¹<http://dba.openoffice.org/drivers/sqlite/index.html>

write.table

E függvény segítségével az `x` objektumot (`data.frame`) írathatjuk ki egy fájlba, amiben karakterhatárolt szövegként tárolódik.

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
           eol = "\n", na = "NA", dec = ".", row.names = TRUE,
           col.names = TRUE, qmethod = c("escape", "double"))
```

A `write.table` függvény `write` függvénytől *eltérő* argumentumainak leírása:

<code>quote</code>	Vagy logikai értéket kell megadni, vagy numerikus vektort. Ha az értéke <code>TRUE</code> , akkor a karakter és faktor oszlopok adatai idézőjelek közé zárva lesznek kiírva. Ha numerikus vektorként adjuk meg, akkor a vektorban azoknak az oszlopoknak a sorszámát adjuk meg, amelyek tartalmát idézőjelek közé szeretnénk foglalni. Mindkét esetben mind az oszlop-, mind a sornevek idézőjelekkel lesznek övezve. Ha az értéke <code>FALSE</code> , akkor egy cella sem lesz „idézéjelezve”.
<code>sep</code>	Ezzel állíthatjuk be, hogy az adatállományban az egyes oszlopokat milyen határoló karakter válassza el.
<code>eol</code>	A sor végét jelző karakter(ek).
<code>na</code>	A hiányzó adatot jelző karakterlánc.
<code>dec</code>	A tizedesjelként használatos karaktert határozza meg.
<code>row.names</code>	Ha az értéke <code>TRUE</code> , akkor a sorok nevei is ki lesznek írva a célállományba, ha <code>FALSE</code> , akkor nem. Figyelni kell arra, hogy ha kiíratjuk a sorneveket, akkor ugyan egy újabb oszlopként fog az megjelenni, de <i>nem lesz az oszlopnak neve</i> . Ez az adatok későbbi visszaolvasásánál hibát eredményezhet.
<code>col.names</code>	Az oszlopok nevének kiíratását meghatározó logikai érték. Ha <code>TRUE</code> , akkor kiíródnak, ha <code>FALSE</code> , akkor nem.
<code>qmethod</code>	Meghatározhatjuk, hogy a dupla idézőjelek (") hogyan jelenjenek meg a kiírt állományban. Az alapértelmezett <code>"escape"</code> C-stílusban <code>\</code> formában írja ki. A másik lehetőség a <code>"double"</code> megduplázza a jeleket. Mindkét érték rövidíthető is az első betűikkel.

```
> adat <- read.table("tabla.txt")
> write.table(adat, "write.table-val_kirva.txt")
```

save

A `save` függvénnyel a megadott objektumokat bináris állományba lehet kiíratni, elmenteni egy későbbi R munkafolyamathoz. Az eredményként kapott fájlt a `load` függvénnyel tölthetjük be egy újabb munkafolyamatba.

```
save(..., list = character(),
      file = stop("'file' must be specified"),
      ascii = FALSE, version = NULL, envir = parent.frame(),
      compress = FALSE)
```

A `save` függvény argumentumainak leírása:

<code>...</code>	A kiírandó objektumok neveit soroljuk itt fel.
<code>list</code>	Egy karaktervektorban megadhatjuk a mentendő objektumok <i>mentési elnevezéseit</i> .
<code>file</code>	Vagy egy kapcsolat vagy egy fájlnev, ahova az objektumokat íratnánk ki. Ha a <code>version</code> értéke 1, akkor fájlnevként kell megadni ezt az argumentumot.
<code>ascii</code>	Ha az értéke <code>TRUE</code> , akkor az objektumok ASCII formátumban lesznek kiírva. Hasznos lehet különböző géptípusok közötti adatátvitelnél. Az alapértelmezett <code>FALSE</code> érték bináris kiírást eredményez.
<code>version</code>	A munkakörnyezet formátumának verziójára utal. Ha az értéke az alapértelmezett <code>NULL</code> , akkor a futó verzió szerint menti el. Értéke 1 az R 0.99.0 verziójától az 1.3.1-ig. Az alapértelmezett érték 2 (az R 1.4.0 verziószámától kezdődően).
<code>envir</code>	Azt határozhatjuk meg vele, hogy mely környezetben keresse a mentésre kijelölt objektumokat.
<code>compress</code>	Ha fájlba mentünk, akkor lehetőség van a kiírt állomány tömörítésére ezzel a logikai argumentummal. Ha kapcsolaton keresztül írunk ki, vagy a <code>version</code> értéke 1, akkor nincs lehetőség a tömörítésre.


```
> save(adat, file="save-vel")
> load("save-vel")
```

save.image

Az előző függvényhez hasonlóan bináris állományba írja ki az objektumokat, de nem csak az argumentumként megadottakat, hanem minden objektumot, ami a munkakörnyezetben található.

```
save.image(file = ".RData", version = NULL, ascii = FALSE,
           compress = FALSE, safe = TRUE)
```

A `save.image` függvény `save` függvénytől eltérő argumentumainak leírása:

<code>safe</code>	Az alapértelmezett TRUE beállítás esetén először készül egy átmeneti állomány, és ha a kiírás ebbe sikeres, akkor ez neveződik át a végleges állománnyá. Bár ez több lemezterületet vesz igénybe, a munkafolyamat adatait biztonságosan kezeli.
-------------------	---

A `save(list = ls(all=TRUE), file = "minden_objektum.RData")` utasítással `save.image` függvény eredményével egyező eredményt érhetjük el. Amennyiben az R-ből `q("yes")` utasítással lépünk ki, akkor is hasonló mentés történik, de akkor egy `.RData` fájlba íródik ki minden objektumunk. Ez a fájl a következő R indításkor automatikusan be is töltődik! Ha Windows RGui-t használunk, akkor a kilépéskor az R rákérdez, hogy akarjuk-e menteni a munkakörnyezetet (32. ábra), amennyiben jóváhagyjuk, akkor egy (a későbbiekben automatikusan betöltődő) `.RData` fájlba menti el a munkakörnyezet objektumait.

dput

R-objektumot tudunk vele kiírni egy ASCII-állományba. Az objektum olvasására használható a `dget(file)` függvény.

```
dput(x, file = "", control = "showAttributes")
```

A `dput` függvény argumentumainak leírása:

<code>x</code>	A kiírandó objektum.
<code>file</code>	Vagy egy karakterlánc, ami a célfájlra mutat vagy egy kapcsolat. Ha "" értéket adunk meg, akkor a konzolra írja az objektumot.
<code>control</code>	A deparsing folyamat paramétereizhető a segítségével. (A részleteket lásd a <code>.deparseOpts</code> leírásánál.)

```
> dput(adat, file="dput-tal")
> adat2 <- dget("dput-tal")
```

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.

```
dump(list, file = "dumpdata.R", append = FALSE,
     control = "all", envir = parent.frame(), evaluate = TRUE)
```

A `dump` függvény (előzőekben még le nem írt) argumentumai:

<code>list</code>	Listában meghatározott egy vagy több kiírandó objektum neve.
<code>evaluate</code>	

```
> dump(ls(), file = "dump-pal.R")
> adat2 <- source('dump-pal.R')
```

sink

E függvény segítségével az R-utasítások outputjai egy ASCII-fájlba íródnak ki.

```
sink(file = NULL, append = FALSE, type = c("output", "message"),
     split = FALSE)
```

```
sink.number(type = c("output", "message"))
```

A `sink` függvény további argumentumainak magyarázata:

<code>file</code>	Vagy a célfájlt megadó karakterlánc, vagy egy <i>kapcsolat</i> , amin keresztül kiíródik az adat.
<code>type</code>	Az alapértelmezett R-outputban a beállítása <code>"output"</code> . Ha átállítjuk <code>"message"</code> -re, akkor csak prompt-, és a figyelmeztetés/hiba üzenetek jelennek meg a terminálban.
<code>split</code>	Ha az értéke <code>TRUE</code> , akkor az output a terminál mellett az új sinkbe is kiíródik.

```
> sink("sink-ke1.txt")
```

Az utasítás végrehajtása után lefuttatott parancsok eredményeként előállt outputok a terminál helyett a `sink-ke1.txt` fájlba íródnak ki. Az `unlink(sink-ke1.txt)` utasítással törölhetjük a sink-fájlunkat.

history

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

```
loadhistory(file = ".Rhistory")
savehistory(file = ".Rhistory")
history(max.show = 25, reverse = FALSE)
```

A `history` függvény további argumentumainak magyarázata:

<code>max.show</code>	A maximálisan megjelenített sorok száma. Ha <code>Inf</code> értéket adunk meg, akkor az összes elérhető sort visszaadja.
<code>reverse</code>	Ha értéke <code>FALSE</code> , akkor a parancsok futtatásának sorrendjében listázza ki azokat, ha <code>TRUE</code> , akkor visszafelé. Ez utóbbi esetben azonban hibásan jelenhetnek meg a többsoros utasítások.

xtable

Az `xtable` könyvtár `xtable` függvényével *L^AT_EX*-, vagy *HTML*- formátumba alakíthatunk át táblázatokat, amiket később fájlba is írhatunk további dokumentumokba való beágyazás céljából.

```
xtable(x, caption=NULL, label=NULL, align=NULL, vsep=NULL, digits=NULL,
       display=NULL, ...)
```

A `xtable` függvény argumentumainak magyarázata:

<code>x</code>	Olyan R -objektum, amelynek osztálya a <code>methods(xtable)</code> -ban megtalálható.
<code>caption</code>	A táblázat címét megadó karakterlánc. Ha az alapértelmezett <code>NULL</code> , akkor nem ad címet.
<code>label</code>	A <i>L^AT_EX</i> -táblázat esetén a címkében szereplő elnevezés. Az alapértelmezett <code>NULL</code> nem hoz létre címkét.
<code>align</code>	E karaktervektorral azt határozzuk meg, hogy az egyes oszlopok hogyan legyenek rendezve. A jobbra igazítást a <code>r</code> , a balra igazítást a <code>l</code> , a középre igazítást pedig a <code>c</code> karakter jelzi. Az oszlopok függőleges elválasztására használható a <code>jel</code> .
<code>vsep</code>	Karaktervektor, aminek a hossza vagy egy, vagy pedig az oszlopok száma plusz 2 (egy a bal, egy pedig a jobb széléhez a táblának). Bármelyik, a <i>L^AT_EX</i> -ben elfogadott elválasztó karakter használható. <i>HTML</i> -módban nem működik.
<code>digits</code>	Numerikus vektor, aminek hossza megegyezik az oszlopok számával. Mindegyik elem az adott oszlopban lévő lebegőpontos számok tizedeshelyeinek a számát jelzi. Ha <code>data.frame</code> az <code>x</code> , akkor mivel a sornevek egy plusz oszlopot képeznek, a vektor hossza eggyel több, mint a <code>ncol(x)</code> .
<code>display</code>	Karaktervektor, aminek a hossza megegyezik az oszlopok számával, illetve <code>data.frame</code> esetén eggyel több, mint az <code>ncol(x)</code> értéke. Az egyes karaktereket a <code>formatC</code> függvény értelmezi (9. táblázat).
<code>...</code>	Kiegészítő argumentumok (jelenleg nincs ilyen).

9. táblázat. A `formatC` értékformáló kódjai

kód	típus	formátum
d	egész szám	
f	valós szám	xxx.xxx
e, E	valós szám	n.ddde+nn vagy n.dddE+nn
g, G	valós szám	n.dddde+nn vagy n.ddddE+nn
fg	valós szám	xxx.xxx
s	szöveg	

Grafika

Az R-környezet a nagyszámú statisztikai eljárás mellett a grafikai lehetőségek tárházát is nyújtja. A statisztikai elemzések különböző, nagy rugalmassággal kezelhető grafikus megjelenítése mellett saját ábratípusainkat is meg tudjuk tervezni.

A grafikai eljárásokat használhatjuk *interaktív* és *batch* módban. Az utóbbi általában az előbbi segítségével alaposan megtervezett grafikák rutinszerű elkészítésére használatos.

Az ábrákat az R valamely úgynevezett *grafikai eszköz meghajtó* (graphics device driver) segítségével hozza létre. Attól függően, hogy a számos meghajtó (10. táblázat) közül melyiket használjuk, az ábrák megjeleníthetők a *képernyőn*, illetve *fájlba* írhatók. Mielőtt egy ábrát készítünk, el kell indítanunk egy meghajtót. Ha nem állítjuk be ennek típusát, akkor az R automatikusan egy *grafikai ablakot* nyit meg az ábrázoláshoz. Ez tulajdonképpen ugyanaz, mintha Windowson kiadnánk a `windows()` utasítást. A grafikával kapcsolatos eljárások három főbb csoportba oszthatók:

- A *magas szintű grafikai* eljárások létrehoznak egy ábrát a grafikus eszközön, annak több elemével együtt (pl.: tengelyek, címkék, feliratok).
- Az *alacsony szintű grafikai* eljárások segítségével kiegészítő információkat jeleníthetünk meg az aktív grafikai eszközön lévő ábránkon (pontok, vonalak, címkék).
- Az *interaktív grafikai* lehetőségek segítségével az aktív grafikai eszközön lévő ábrához adhatunk újabb információt megjelenítő elemeket, vagy arról értékeket olvashatunk le. Mindezt az egér segítségével.

A grafikai eszköz beállítása

Ha az R-környezet alapbeállítását használjuk, akkor egy grafikus ablakba rajzoljuk a parancsokban megadott ábráinkat. Ha a munkafolyamat során több ábrát is készítünk, és szeretnénk, hogy ezek visszanezethetők legyenek, akkor vagy elmentjük azokat külön-külön fájlokba, vagy rögzítjük a grafikai történetben.

Több grafikai eszköz

Arra is van lehetőség, hogy több grafikus ablakunk legyen és a munkafolyamat során készülő ábrák külön ablakokban egyszerre láthatók legyenek. A 10. táblázat függvényeinek segítségével tudunk új grafikus eszközt megnyitni ábráink készítésére. Példaképpen nyissunk meg egyszerre több eszközt:

```
> windows()
> pdf()
> postscript()
> png()
> jpeg()
> windows()
> windows()
```

Létrehoztunk tehát hét eszközt, amire ábrákat készíthetünk. Ha egyszerre több grafikai eszközt használunk, akkor figyelni kell arra, hogy egyszerre csak az egyik eszköz lesz aktív. Az aktuálisan kiadott utasítások mindig az aktív eszközre lesznek kirajzolva. Ha létrehozunk egy új eszközt, akkor az lesz az aktív, míg a többit inaktivizáljuk. Fontos, hogy figyeljünk arra, hogy ha a létrehozott eszköz *grafikus ablak* (pl.: `windows()`), akkor az R-környezet *fókusza* arra kerül át. Ez azt jelenti, hogy amikor kiadtuk a konzolon az utasítást, az ENTER megnyomásáig a fókusz ott volt, azután pedig már a grafikus ablakon lesz. Ennek gyakorlati jelentősége az, hogy hiába kezdünk el gépelni vagy beilleszteni újabb kódokat, azok nem kerülnek be a konzolra, mert az aktív ablak a grafikus ablak. A fókuszt az egérrel a konzolra kattintva tudjuk visszahelyezni. A munka során lekérdezhethetjük, hogy milyen grafikus eszközeink vannak megnyitva. Ez a `dev.list()` függvénnyel lehetséges.

10. táblázat. Grafikai meghajtók

utasítás	rövid leírása
X11()	A grafikus ablak X11 window rendszereken való használatához (pl. Linux).
windows()	A grafikus ablak Windowson való használatához.
quartz()	A grafikus ablak MacOS X környezetben való használatához.
postscript()	PostScript printeren való nyomtatáshoz, vagy PostScript típusú fájlba íráshoz.
pdf()	PDF fájlba való íráshoz.
png()	PNG pixelgrafikus állomány létrehozásához.
jpeg()	JPEG pixelgrafikus fájl készítéséhez.
bitmap()	Bitmap fájlba írja a képet.
pictex()	A $\text{T}_{\text{E}}\text{X}$, illetve $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ állományokba beilleszthető formában írja ki az ábrát egy <code>.tex</code> állományba. A <code>\usepackage{pictex}</code> szükséges.
xfig()	Xfig grafikát hoz létre.
bmp()	BMP állományba írja az ábrát.
win.metafile()	Windows Metafájlba írja ki az ábrát.
win.print()	A nyomtatóra küldi az ábránkat.

```
> dev.list()
```

```

      windows          pdf          postscript
      2                3                4
png:Rplot%03d.png jpeg:75:Rplot%03d.jpg          windows
      5                6                7
      windows
      8

```

Azt hogy éppen melyik grafikus eszköz aktív, a `dev.cur()` utasítással tudjuk lekérdezni.

```
> dev.cur()
```

```
windows
      8
```

Ha az aktív eszközünk egy grafikus ablak, akkor annak címsorában a `R Graphics: Device 8 (ACTIVE)` felirat is jelzi aktív voltát. Arra is van mód, hogy egy adott grafikus eszköz előtti, illetve utáni eszközt lekérdezzük, erre szolgál a `dev.prev()`, illetve a `dev.next()` utasítás. Ha az aktív státuszt egy másik eszközre szeretnénk átállítani, akkor a `dev.set(which = k)` függvényt használjuk. A `k` argumentumban adhatjuk meg az aktív eszköz számát. Azonban a konkrét szám helyett használhatjuk az előtte (`dev.prev()`), illetve utána (`dev.next()`) *relatív hivatkozást* is.

```
> dev.set(which = dev.next())
```

```
windows
      2
```

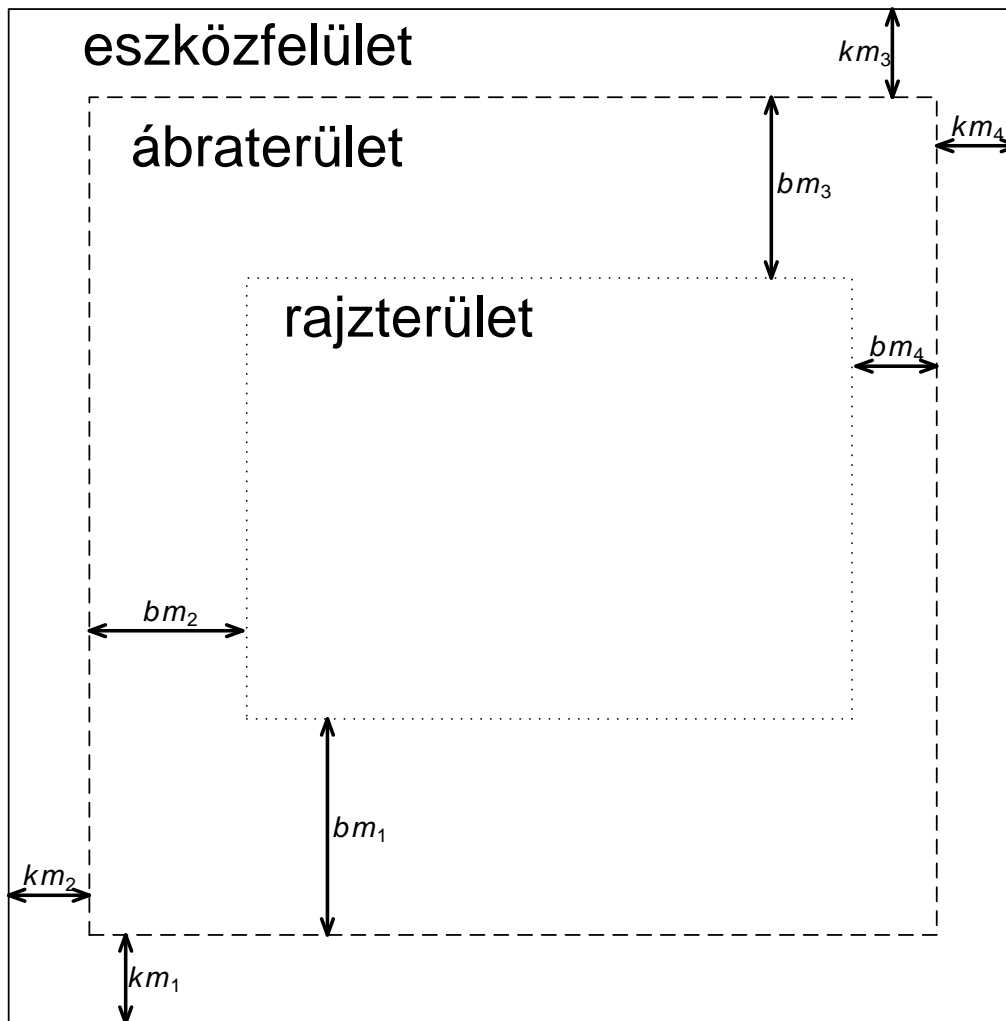
Ahogy látható, a 2. számú eszköz lett az aktív. Ez azt is bemutatja, hogy ha az utolsó eszközről a következőre ugunk, akkor az az első lesz. Ha egy eszközre már nincs szükségünk, bezárhatjuk a `dev.off(k)` utasítással, amiben a `k` argumentum az eszköz számára utal.

```
> dev.off(2)
```

```
pdf
      3
```

A törlés után a sorban következő eszköz lesz az aktív. Ha nem grafikus ablak a bezárt eszköz, akkor annak tartalma fájlként lesz mentve a bezárás után. Ezek a fájlok a *munkakönyvtárba* lesznek mentve.

Az adott eszköz tartalmát át tudjuk másolni a `dev.copy(device, ..., which = dev.next())` utasítással egy általunk meghatározott eszközre. Hasonló eredményt érhetünk el a `dev.print(device = postscript, ...)` paranccsal is, azzal a különbséggel, hogy ebben az esetben a forráseszköz be is záródik. A `dev.copy2eps(...)`



7. ábra. Grafikai eszköz részei

függvény egy speciális esete az előzőknek, mivel ennek segítségével EPS állományba írhatjuk ki az eszközön készített ábránkat. A `dev.control(displaylist = c("inhibit", "enable"))` segítségével az adott eszközön rögzíthetjük az egymás után megjelenő ábrákat, így visszanezhetjük azokat. Ha a `displaylist` argumentumot "inhibit" értékre állítjuk, akkor kikapcsoljuk a rögzítést, ha "enable" értékre, akkor bekapcsoljuk. Ha rögzíteni akarjuk a képeket, akkor az ábra létrehozása előtt kell ezt az utasítást beállítanunk. A `dev.copy` függvény csak bekapcsolt rögzítés esetén működik.

A grafikai felület szerkesztése

A 7. ábrán láthatók a grafikai eszköz részei és azok alapbeállítás szerinti elrendezése. Az R-környezetben lehetőségünk van a felület részeinek, illetve elrendezésüknek átszabására. A grafikai felület testreszabására használhatjuk a `layout`, és a `split.screen` függvényt és a *grafikai paramétereiket*. (A grafikai paraméterek leírása az 58. laptól olvasható.)

layout

A `layout` függvény az *eszközfelületet* *ablakokra* „darabolja fel”, az argumentumban megadott mátrixnak, illetve az oszlopszélesség és sormagasság értékeinek megfelelően.

```
layout(mat, widths = rep(1, dim(mat)[2]), heights = rep(1, dim(mat)[1]),
       respect = FALSE)
layout.show(n = 1)
lcm(x)
```

A `layout` függvény argumentumainak rövid leírása:

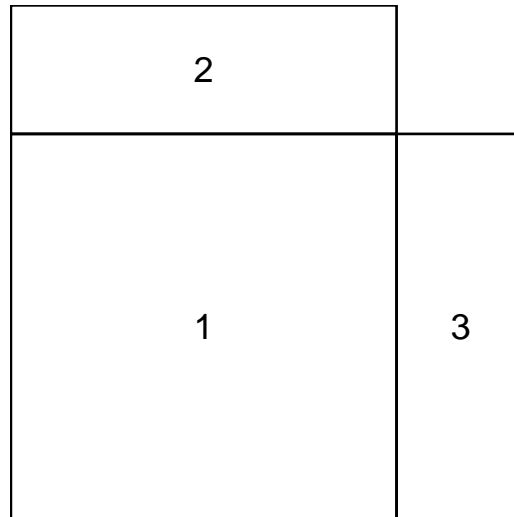
```

> m <- matrix(c(2, 0, 1, 3), 2, 2, byrow = TRUE)
> m

      [,1] [,2]
[1,]    2    0
[2,]    1    3

> nf <- layout(m, widths = c(3, 1), heights = c(1, 3), TRUE)
> layout.show(nf)

```



8. ábra. A grafikai felület átszabása a `layout` függvénnyel I.

<code>mat</code>	Mátrix formájában adhatjuk meg a kialakítandó <i>alablakok</i> számát, aminek minden cellája 0 vagy pozitív egész szám lehet. Egy szám többször is szerepelhet a mátrixban, viszont hiányos sorozat esetén hibát generál a függvény. A 0 értékű cellának megfelelő alablakba nem kerül majd ábra.
<code>widths</code>	Vektorban adhatjuk meg az oszlopok szélességét. Relatív szélességet numerikus vektorban adhatunk meg. Az abszolút szélességet centiméterben, az <code>lcm</code> függvény segítségével adhatjuk meg.
<code>heights</code>	Az oszlop magasságát adhatjuk meg e függvény segítségével, a <code>widths</code> argumentumhoz hasonlóan.
<code>respect</code>	Vagy logikai értéként adjuk meg, vagy mátrixként. Az utóbbi esetben a mátrix méretének meg kell egyeznie a <code>mat</code> argumentumban megadott mátrix méretével. A mátrix cellák értéke 0 vagy 1 lehet.
<code>n</code>	A kirajzolandó ábrák száma.
<code>x</code>	Azt a dimenziót adhatjuk meg vele, ami centiméterben lesz értelmezve.

A 8. ábrán látható grafikai felületszerkezet a felette látható mátrixra épül. Megfigyelhető, hogy a jobb felső sarokban nem jön létre *alablak*, ennek az oka, hogy az alapmátrixban a második cella értéke 0. Az is megfigyelhető, hogy a `layout` függvény `widths` argumentuma `c(3,1)` értéket vett fel, aminek az lesz az eredménye az ábrán, hogy a bal oszlopban elhelyezkedő két cella (2,1) háromszor olyan széles, mint a jobb oszlopban lévő cella (0,3). A sorok magasságában tapasztalható különbségek a `heights` értékei miatt keletkeztek. A létrehozott új szerkezetbe ezek után berajzolhatók az ábrák. Azt, hogy az adott ábra melyik *alablakba* kerüljön, a rajzutasítások sorrendjében határozhatjuk meg. Az előző példában létrehozott *alablakokba* rajzolásra látható példa a 9. ábrán.

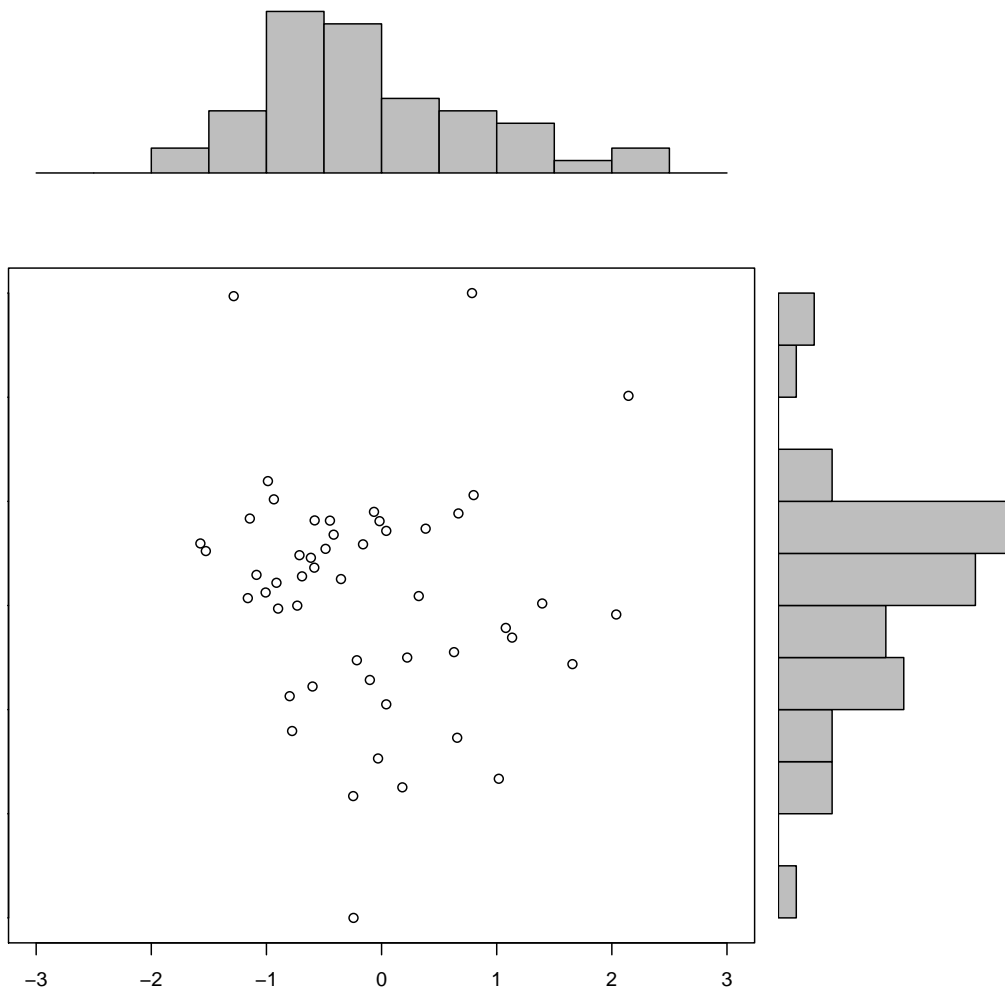
`split.screen`

A `split.screen` függvénnyel az eszközfelületet részekre „vághatjuk”. Ezek külön-külön képernyőként kezelhetők, rajzolhatók és törölhetők. A `screen` segítségével kiválaszthatjuk azt az *alablakot*, amelyikkel dolgozni szeretnénk. Az `erase.screen` törli a meghatározott képernyőt, a `close.screen` pedig törli a meghatározott ablak definícióját.

```

> x <- pmin(3, pmax(-3, rnorm(50)))
> y <- pmin(3, pmax(-3, rnorm(50)))
> xhist <- hist(x, breaks = seq(-3, 3, 0.5), plot = FALSE)
> yhist <- hist(y, breaks = seq(-3, 3, 0.5), plot = FALSE)
> top <- max(c(xhist$counts, yhist$counts))
> xrange <- c(-3, 3)
> yrange <- c(-3, 3)
> plot(x, y, xlim = xrange, ylim = yrange, xlab = "", ylab = "")
> barplot(xhist$counts, axes = FALSE, ylim = c(0, top), space = 0)
> barplot(yhist$counts, axes = FALSE, xlim = c(0, top), space = 0,
+       horiz = TRUE)

```



9. ábra. A grafikai felület átszabása a layout függvénnyel II.


```

split.screen(figs, screen, erase = TRUE)
screen(n = , new = TRUE)
erase.screen(n = )
close.screen(n, all.screens = FALSE)

```

A függvények argumentumainak rövid leírása:

<code>figs</code>	Az oszlopok és a sorok számát meghatározó, kételemű numerikus vektor. Négyoszlopos mátrixként is megadható a képernyő szerkezete. Ha mátrix, akkor minden sora leír egy képernyőt, annak bal- és jobb oldalára, tetejére és aljára vonatkozó értékekkel. A cellák értéke NDC egységben értendő, vagyis a bal alsó sarok 0 és a jobb felső 1.
<code>screen</code>	E számmal határozzuk meg azt, hogy mely képernyőt vágja újabb képernyőkre a függvény. Ha ezt nem határozzuk meg, akkor az egész grafikai eszközre vonatkozik a művelet.
<code>erase</code>	Logikai érték, ami arra vonatkozik, hogy a kiválasztott képernyő törlődjék-e.
<code>n</code>	E számmal meghatározzuk, hogy melyik ablakot készítse elő a rendszer a rajzolásra, törlésre vagy definíció törlésre.
<code>new</code>	Logikai érték, ami ha TRUE, akkor a rajzolás előtt az ablak törlődik.
<code>all.screens</code>	Logikai érték arra vonatkozóan, hogy az összes képernyődefiníció törlődjön-e.

Az alapterepítés grafikai függvényei

Magas szintű grafika

A magas szintű grafikai függvények minden esetben új ábrát generálnak az éppen aktív grafikai eszközön, egyszersmind az adott grafikai eszköz addigi tartalma törlődik. E függvények eredményeként a tengelyek, címkék és feliratok automatikusan jelennek meg, ha azokat alapértelmezésben használjuk.

Az alapterepítésben elérhető magas szintű grafikai függvények: `assocplot`, `barplot`, `boxplot`, `coplot`, `contour`, `curve`, `dotchart`, `filled.contour`, `fourfoldplot`, `hist`, `image`, `interaction.plot`, `matplot`, `mosaicplot`, `pairs`, `persp`, `pie`, `plot`, `qqnorm`, `qqplot`, `stars`, `sunflowerplot`, `symbols`, `termplot`, `ts.plot`. A függvények argumentumai igen nagy számúak is lehetnek. A részletekre nem kitérve a közös argumentumok rövid leírása alább olvasható:

<code>add</code>	Ha értéke TRUE, akkor lehetővé teszi, hogy alacsony szintű grafikai függvényekkel az ábrát elérjük. Nem minden eljárást tesz lehetővé.
<code>axes</code>	Ha FALSE értéket adunk meg, akkor nem generál tengelyeket az ábránkhöz, így lehetővé téve, hogy magunk szerkesztette tengelyekkel (<code>axis</code>) lássuk el a későbbiekben. Alapértelmezésben TRUE.
<code>log</code>	Az általa meghatározott tengely(ek)e)t log-transzformálja. Ha értéke "x", akkor az x-en, ha "y", akkor az y tengelyen végzi el az átalakítást. Ha "xy", akkor mindkettőn. Sok de nem minden ábratípuson működik.
<code>main</code>	Az ábra címe, ami felülre és középre lesz kiírva (nagybetűkkel).
<code>sub</code>	Alcím, ami az x tengely alá kerül kisebb betűkkel.
<code>type</code>	Értékével a grafikánk rajzát állíthatjuk be: <ul style="list-style-type: none"> "p" pontokat rajzol "l" vonalakat rajzol "b" vonalakkal összekötött pontokat rajzol "o" a pontok fölé rajzolja a vonalakat "h" a pontokból függőleges vonalat húz az x tengelyhez "s", "S" lépcsőzetes vonalrajz "n" Nem rajzol ábrát. A tengelyeket ugyan felrajzolja, de azon kívül nincs semmi a grafikus eszközön. Viszont lehetőséget ad arra, hogy alacsony szintű grafikai eljárással rajzoljunk rá.
<code>xlab</code> , <code>ylab</code>	Az x, illetve y tengely feliratát határozhatjuk meg, alapértelmezésben a megjelölt objektum neve.

Alacsony szintű grafikai utasítások

Az alapterepítés alacsony szintű grafikai függvényei: `abline`, `arrows`, `axis`, `contour`, `grid`, `legend`, `lines`, `mtext`, `points`, `polygon`, `rect`, `segments`, `qqline`, `text`, `title`.

Interaktív grafikai lehetőségek

Az előző két grafikai függvénycsoportnál az egyes rajzelemek megjelenítését parancsok segítségével tudjuk elérni. Az alaptelepítésben vannak olyan függvények, amelyek segítségével az ábránkról információkat tudunk leolvasni, illetve kiegészíthetjük feliratokkal, rajzelemekkel.

identify

E függvény segítségével az egérmutató pozícióját tudjuk kiolvasni, ha a bal egérgombot megnyomjuk. Ha az adott x, y koordinátához közel van rajzolt pont, akkor annak indexét jeleníti meg a pont mellett.

```
identify(x, y = NULL, labels = seq(along = x), pos = FALSE,
        n = length(x), plot = TRUE, offset = 0.5, ...)
```

x,y	Egy szórásdiagram pontjainak koordinátái. Meg lehet adni objektumot is, ami a koordinátákat tartalmazza.
labels	Lehetőség van arra, hogy a koordinátákkal megegyező elemszámú vektorban megadjunk címkeket a pontokhoz.
pos	Ha az értéke TRUE, akkor a visszatérési értékhez hozzárendelődik egy érték, ami a címke relatív pozícióját adja meg (1 = alatta, 2 = balra, 3 = felette, 4 = jobbra).
n	Az azonosítandó pontok maximális száma.
plot	Ha az értéke TRUE, a címkek megjelennek az ábrán, különben nem.
offset	A címkeket elválasztó távolság karakter-szélességben megadott mértéke.
...	További grafikai paraméterek

Az alábbi példával létrehozunk egy ábrát, amelyen véletlen pontok láthatók. Ha valamelyik közelébe kattintunk, akkor az adott pont indexéhez tartozó nagybetű jelenik meg mellette.

```
> x <- rnorm(26,0,1)
> y <- rnorm(26,0,1)
> plot(x,y)
> identify(x,y,labels=LETTERS)
```

Ha meghatároztuk a n argumentumot, akkor annak elérése után a kurzor újra aktív lesz a konzolon. Ha nem határoztunk meg ilyen korlátot, akkor úgy nyerhetjük vissza a kurzorunkat, hogy a konzolra kattintunk (a fókusz át helyezzük) és megnyomjuk a ESC billentyűt.

locator

A bal egérgombbal való kattintással megadott pozíciót adja vissza, illetve ezen adatok felhasználásával az ábrát pontokkal, szimbólumokkal vagy vonalakkal egészíthetjük ki.

```
locator(n = 512, type = "n", ...)
```

n	Az azonosítandó pontok száma.
type	Az értéke "n" (nem rajzol), "p" (pontot rajzol), "l" (vonalat rajzol) vagy "o" (pontot és vonalat rajzol) lehet.
...	Egyéb grafikai paraméterek adhatók meg.

```
> x <- rnorm(26,0,1)
> y <- rnorm(26,0,1)
> plot(x,y)
> locator(n=3,type="p",pch=13)
```

A fenti példával a véletlen pontokból álló ábránkra három pontot rajzolhatunk az egér bal gombját használva. A pontok pch=13 kódú szimbólumként jelennek meg (11. ábra)

Grafikai paraméterek

A grafikai paraméterek beállításához, illetve lekérdezéséhez a par függvényt használhatjuk. Az egyes grafikai paramétereket a par függvény argumentumaként állíthatjuk be paraméternév = érték formában, de megadhatjuk listaként is. Az aktuális paraméterbeállításokat a par() vagy a par(no.readonly=TRUE) utasításokkal kérdezhetjük le. A csak olvasható és nem írható argumentumok neve előtt a * jel látható.

adj	A szöveges elemek igazítását állíthatjuk be vele. A 0 érték balra igazít, az 1 jobbra, míg a 0.5 középre. Megadhatjuk $c(x,y)$ formában is, ekkor a vízszintes és függőleges irányban külön állíthatjuk be ezt a tulajdonságot.
ann	Ha FALSE értéket adunk meg, akkor a magas szintű függvényeknél nem jelennek meg feliratok. Az alapértelmezett érték TRUE.
ask	Logikai argumentum. Ha az értékét TRUE-ra állítjuk, akkor egy új rajz létrejötté előtt, a felhasználótól jóváhagyást kér.
bg	A háttér színét állíthatjuk be vele.
bty	Karakterlánc, aminek segítségével meghatározhatjuk, hogy az ábrát határoló doboz milyen vonallal legyen kirajzolva. "o" teljes keretet rajzol "1" baloldali és alsó oldalakat rajzol "7" jobboldali és felső oldalakat rajzol "c" baloldali, alsó és felső oldalakat rajzol "u" baloldali, alsó és jobboldali oldalakat rajzol "]" alsó, jobboldali és felső oldalakat rajzol "n" nem rajzol keretet
cex	Számérték, ami a megjelenített szöveg, illetve szimbólum méretét állítja be, az alapértelmezett értékhez (1) viszonyítva.
cex.axis	A tengelyfeliratok méretének az aktuálshoz viszonyított nagyítási mértéke.
cex.lab	Az x és y címkék méretének az aktuálshoz viszonyított nagyítási mértéke.
cex.main	A főcím méretének az aktuálshoz viszonyított nagyítási mértéke.
cex.sub	Az alcím méretének az aktuálshoz viszonyított nagyítási mértéke.
*cin	Hüvelykben megadott karakterméret (szélesség, magasság).
col	A rajzoláshoz használatos szín.
col.axis	A tengelyfelirathoz használt szín.
col.lab	Az x és y címkékhez használt szín.
col.main	A főcímhez használt szín.
col.sub	Az alcímhez használt szín.
*cra	Az alapértelmezett karakterméret pixelben (szélesség, magasság).
crt	Számérték, amivel meghatározhatjuk, hogy egy karakter hány fokkal legyen elforgatva. Nem túl intelligens argumentum, mivel csak a 90 fok többszörösét képes értelmezni.
*csi	Az alapértelmezett karaktermagasság, hüvelykben.
*cxy	Az alapértelmezett karakterméret (szélesség, magasság) a felhasználói mértékegységben. A $\text{par}("cxy")=\text{par}("cin")/\text{par}("pin")$. Megjegyzendő, hogy a $c(\text{strwidth}(ch), \text{strheight}(ch))$ használata az adott ch karakterláncához sokkal pontosabb.
*din	A grafikai eszköz dimenziói (szélesség, magasság) hüvelykben.
err	Hibaüzenet. <i>Nem működik!</i>
family	A rajzhoz használt betűcsalád neve. Minden grafikus eszközön egyforma, bár néhány nem engedi az átállítását. Az alapértelmezett érték ". Standard értékei "serif", "sans", "mono" és "symbol". Egyes eszközökön más családok is használhatók.
fg	Az ábra előterének rajzolásához használt szín. Ezel a színnel fog megjelenni a keret, a tengely.
fig	Egy $c(x1, x2, y1, y2)$ formában megadható NDC vektor, ami meghatározza az <i>ábraterületet</i> az eszközön. Ha beállítjuk, akkor új rajz jön létre. Így, ha egy már meglévőhöz szeretnénk hozzáadni, akkor a $\text{new}=\text{TRUE}$ beállításra is szükségünk lesz.
fin	Az ábraterület dimenziói (szélesség, magasság) hüvelykben. Ha beállítjuk, akkor új rajz jön létre.
font	Egész szám, ami meghatározza, hogy milyen betűt használunk a szövegünkben. Az 1 normál, a 2 vastag, a 3 dőlt és a 4 vastag dőlt betűt eredményez.
font.axis	A tengelyfeliratokhoz használt betű.
font.lab	A címkékhez használt betű.
font.main	A főcímhez használt betű.
font.sub	Az alcímhez használt betű.
gamma	Gamma korrekció. (A részletek a hsv függvénynél találhatók.)

lab	Egy $c(x, y, len)$ formátumú numerikus vektor, ami a tengelyfeliratokat módosítja. Az x és y elemben azt határozzuk meg, hogy közelítőleg hány jel legyen az egyes tengelyeken. Az alapértelmezett érték $c(5, 5, 7)$. Jelenleg a len nem működik.
las	A tengelyek címkeinek elhelyezkedési irányát határozhatjuk meg a numerikus értékének beállításával: 0 mindig párhuzamos a tengellyel, ez az alapértelmezés 1 mindig horizontális a felirat 2 a felirat mindig merőleges a tengelyre 3 a felirat mindig függőleges
lend	A vonalvég stílusát határozhatjuk meg. Vagy számként (0 = <i>lekerekített</i> , 1 = <i>vágott</i> , 2 = <i>szögletes</i>), vagy karakterláncként (" round " = <i>lekerekített</i> , " butt " = <i>vágott</i> , " square " = <i>szögletes</i>) adhatjuk meg.
lheight	A szövegsorok magasságszorzója. Az alapérték 1.
ljoin	A vonalak találkozásait beállító argumentum, ami lehet szám (0 = <i>lekerekített</i> , 1 = <i>félderékszög</i> , 2 = <i>ferde</i>), vagy karakterlánc (" round " = <i>lekerekített</i> , " mitre " = <i>félderékszög</i> , " bevel " = <i>ferde</i>).
lmitre	A vonal szögellésének limitje. Az értékének 1-nél nagyobbaknak kell lennie, alapértelmezetten 10. Nem mindegyik grafikus eszköz fogadja el.
lty	A vonal típusát határozhatjuk meg a segítségével. Megadhatjuk számként (0 = <i>lát-hatatlan</i> , 1 = <i>folyamatos</i> , 2 = <i>szaggatott</i> , 3 = <i>pontozott</i> , 4 = <i>pontozott-szaggatott</i> , 5 = <i>hosszú-szaggatott</i> , 6 = <i>hosszú-rövid szaggatott</i>), illetve karakterként (" blank " = <i>lát-hatatlan</i> , " solid " = <i>folyamatos</i> , " dashed " = <i>szaggatott</i> , " dotted " = <i>pontozott</i> , " dotdash " = <i>pontozott-szaggatott</i> , " longdash " = <i>hosszú-szaggatott</i> , " twodash " = <i>hosszú-rövid szaggatott</i>) kódolva. A vonalrészek hosszát meg lehet határozni egy maximálisan 8 karakterből álló karakterláncsal is. A $c(1:9,"A":"F")$ karakterek közül állíthatjuk össze a karakterláncot.
lwd	Vonal vastagságát megadó pozitív szám, ami alapértelmezésben 1.
mai	A rajzterület margóméreteit <i>hüvelykben</i> meghatározó vektor. A 7. ábrán a belső margók jelölésének megfelelően kell megadni: $c(bm_1, bm_2, bm_3, bm_4)$.
mar	A rajzterület margóméreteit <i>sorszámokban</i> meghatározó vektor. A 7. ábrán a belső margók jelölésének megfelelően kell megadni: $c(bm_1, bm_2, bm_3, bm_4)$. Az alapértelmezett értéke $c(5, 4, 4, 2) + 0.1$.
mex	A mex a margókon használatos koordináták leírására szolgáló karakter méretét növelő faktor. Nem a karakter méretét változtatja, hanem a mai és mar , illetve az oma és omi közötti konverziót határozza meg.
mfcol, mfrow	A grafikus felület felosztására használhatjuk, a vektor formában ($c(nr, nc)$) megadott értékek segítségével. Eredményeként az $nr*nc$ tömbnek megfelelő sor- és oszlopszámú képernyőszerkezet jön létre. A tömb celláiba külön-külön helyezhetünk el ábrákat.
mfg	Az $mfcol$ és $mfrow$ paraméterek által meghatározott tömb elemekre hivatkozhatunk a $c(i, j)$ formájú vektor segítségével. A meghatározott cellának megfelelő felületre kerül a következő rajz. Az S-el való kompatibilitás végett a $c(i, j, nr, nc)$ forma is használható.
mgp	A tengelycím, tengelycímke és a tengelysor margósora mex egységben. Az alapértelmezés $c(3, 1, 0)$.
mkh	Ha a rajzolandó szimbólum pch értéke szám, akkor ezzel az argumentummal határozható meg a magassága hüvelykben mérve. Jelenleg nem működik.
new	Logikai érték, melyet ha az alapértelmezett FALSE értékről TRUE-ra állítunk, akkor a következő, magasszintű függvényvel készített ábra rárajzolódik az aktív eszközön már meglévő rajzra. Ellenkező esetben, minden magasszintű rajzolás előtt törlődik a felület.
oma	A külső margókat <i>sorszámokban</i> meghatározó vektor (7. ábra). A vektort $c(km_1, km_2, km_3, km_4)$ formában kell megadni.
omd	A külső margókat NDC (<i>normalized device coordinates</i>) egységben megadó vektor, amit a $c(x1, x2, y1, y2)$ formában kell megadni (7. ábra).
omi	A külső margókat <i>hüvelykben</i> meghatározó vektor (7. ábra). A vektort $c(km_1, km_2, km_3, km_4)$ formában kell megadni.

pch	Vagy egy szimbólumkódot használunk (0-25), vagy egy karaktert adunk meg pont jelölésére. A 0-24 közötti kódok és a megfelelő szimbólumok a 11. ábrán láthatók.
pin	Az aktuális rajzterület dimenziói (szélesség, magasság), hüvelykben megadva.
plt	Segítségével az aktuális ábraterületen koordinátákkal határozhatjuk meg a rajzterületet. Vektor formában kell megadni $c(x_1, x_2, y_1, y_2)$.
ps	Egész számmal adhatjuk meg a karakterek vagy szimbólumok pontméretét.
pty	A rajzterület alakját határozhatjuk meg. Ha "s" értéket vesz fel, akkor <i>négyzet</i> alakú, ha "m" értékű, akkor a maximális rajzterületet biztosító téglalap alakú rajzterületet kapunk.
smo	A körök és körívek simításával kapcsolatos argumentum. Nem működik.
srt	Karakterláncok elforgatását adhatjuk meg fokban (lásd még <code>crt</code>).
tck	A rajzterület szélességéhez, illetve magasságához viszonyítva adhatjuk meg a rácselemek <i>hosszát</i> . Ha értéke 1, akkor a teljes rajzterületet behálózza, egy grided hoz létre, az alapértelmezett érték NA. Ha pozitív számot adunk meg, akkor a rajzterületre, ha negatívot, akkor azon kívülre húzza a vonalakat.
tcl	A rácselemek méretét a szövegsor magasságának arányában adhatjuk meg. Az alapértelmezett érték -0.5. Ha NA értéket adunk meg, akkor annak következtében a <code>tck = -0.01</code> értékű lesz (S alapértelmezett).
tmag	A főcím méretének a rajz egyéb felirataihoz viszonyított növelését meghatározó szám.
type	E karakterrel megadhatjuk a <i>rajzolás</i> típusát. A részleteket lásd az 57. lapon.
usr	A rajzterület felhasználó által beállítható szélső koordinátái, amiket $ac(x_1, x_2, y_1, y_2)$ formában kell megadni. Ha a <code>xlog</code> értéke TRUE, akkor az <code>x</code> határértékei <code>10^{par("usr")}[1:2]</code> .
xaxp	Az <code>x</code> tengely szélső értékű jelölőinek koordinátáit adja meg $c(x_1, x_2, n)$ formában. Ha az <code>xlog</code> értéke FALSE, az <code>n</code> egész szám, ami azt adja meg, hogy a két megadott <code>x</code> érték között hány szakasz legyen.
xaxs	Az <code>x</code> tengely intervallumának számítási stílusát meghatározó argumentum. A lehetséges értékek: "r", "i", "e", "s", "d". Azonban jelenleg csak az "r" és az "i" használható. A stílusok mindegyike az adattartományon vagy a <code>xlim</code> értékeken alapszik. Az "r" (reguláris) módszer az először 4%-kal megnagyobbítja az adattartományt, és ehhez hoz létre egy jól illeszkedő címkéjű tengelyt. Az "i" (internal) az eredeti adattartományhoz hoz létre egy jól illeszkedő címkéjű tengelyt.
xaxt	E karakterrel meghatározható az <code>x</code> tengely stílusa. Az "s" érték az alaértelmezés, használható az "l", illetve az "e" érték is, de ezek eredménye ugyanaz lesz, mint az "s"-nél. Ha "n" értéket adunk meg, akkor létrehozza a tengelyt, de nem rajzolja ki.
xlog	Ha a FALSE alapértelmezett értéket TRUE-ra állítjuk, akkor az <code>x</code> tengelyen logaritmus skálát fog használni.
xpd	Logikai vagy NA értéket vehet fel. Ha FALSE, akkor a <i>rajzterületre</i> , ha TRUE, az <i>ábraterületre</i> , NA esetén pedig az egész <i>eszközfelületre</i> rajzol.
yaxp	A <code>xaxp</code> argumentumhoz hasonló.
yaxs	Az <code>y</code> tengely intervallumának számítási stílusát meghatározó argumentum. Részletek az <code>xaxs</code> argumentumnál olvashatók.
yaxt	Az <code>y</code> tengely stílusát meghatározó karakter. Részletek az <code>xaxt</code> argumentumnál olvashatók.
ylog	Az <code>y</code> tengely skáláját állíthatja át. Részletek az <code>xlog</code> argumentumnál olvashatók.

A 10. ábrán látható három grafikai paraméter átállítása és azok eredménye. A `mfrow = c(2, 2)` beállítás négy egyenlő részre osztja a grafikai eszköz felületét. Ahhoz, hogy a négy rajzterület négyzet alakú legyen, a `pty` értékét "s"-re változtattuk. A `bty = "n"` hatására a rajzterületek körül nem jelennek meg keretek.

Interaktív vizualizáció

Számos fejlesztés áll rendelkezésre interaktív vizualizációs feladatok megoldására. Egyesek telepítéséhez szükséges az R-környezeten kívül egyéb környezet vagy meghajtók. Az `iplots`²² és a `KLIMT`²³ JAVA környezetet igénylő eszköz, az R-rel való kommunikációjukhoz szükséges az `rJava` csomag is. Az `xgobi` újabb változata a

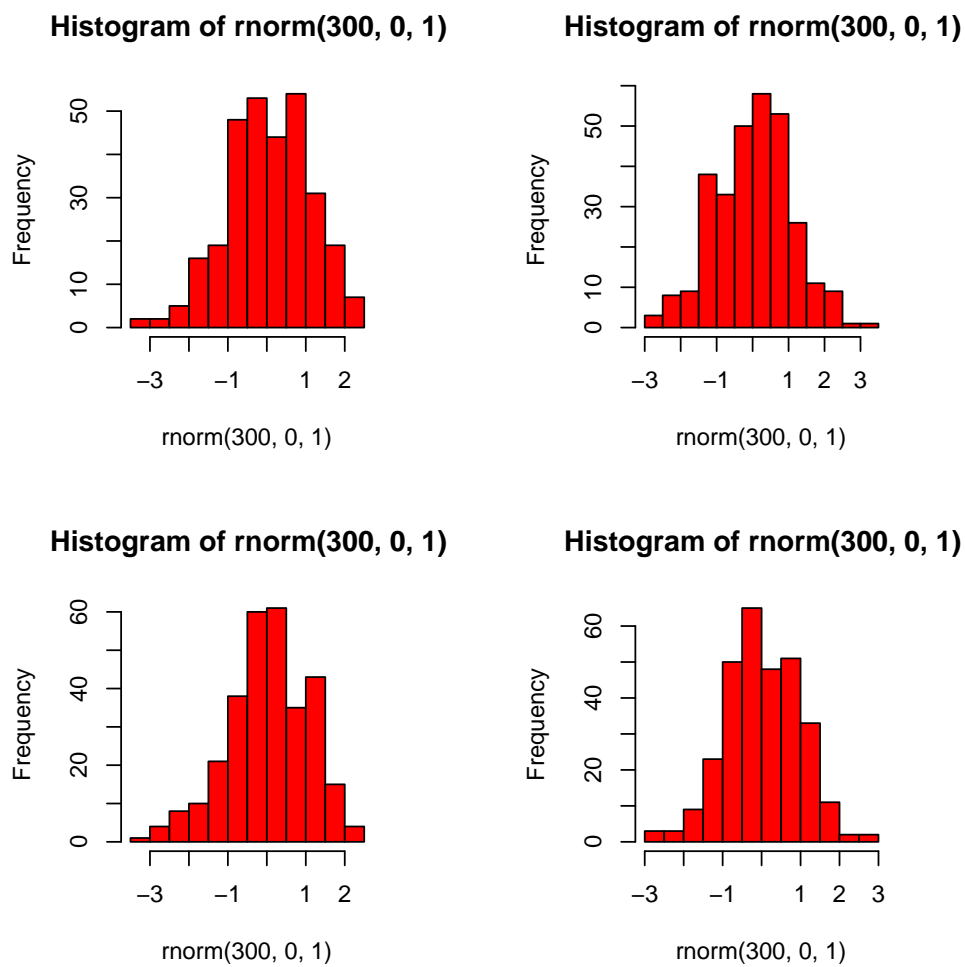
²²<http://www.rosuda.org/iPlots/>

²³<http://www.rosuda.org/KLIMT/>

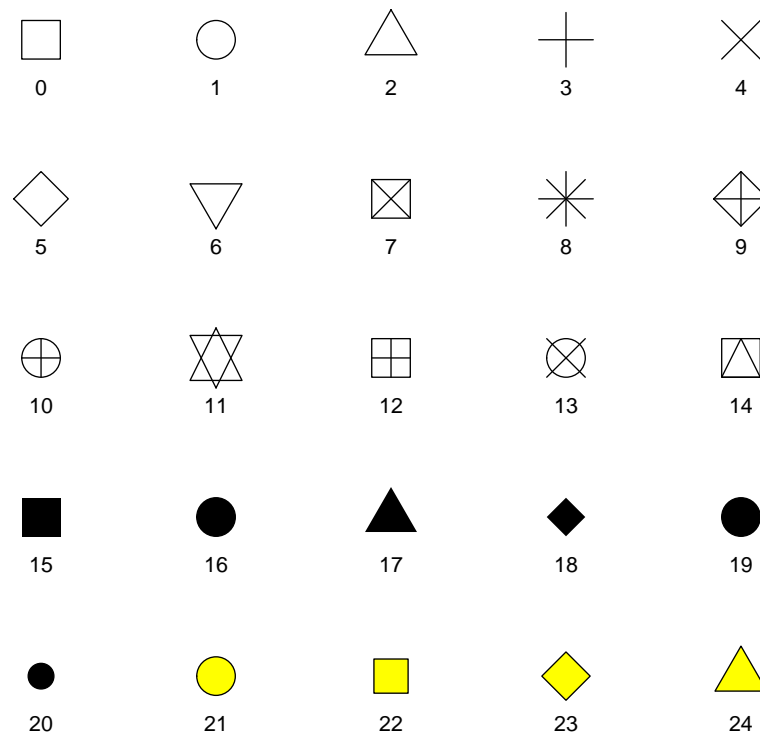
```

> grafikai.parameterek <- par(mfrow = c(2, 2), pty = "s", bty = "n")
> hist(rnorm(300, 0, 1), col = "red")
> hist(rnorm(300, 0, 1), col = "red")
> hist(rnorm(300, 0, 1), col = "red")
> hist(rnorm(300, 0, 1), col = "red")
> par(grafikai.parameterek)

```



10. ábra. par példa



11. ábra. A pch kódok 0-tól 24-ig

`ggobi`²⁴ vizualizációs rendszerrel is kialakítható együttműködés a `Rggobi` csomag telepítésével. Az `OpenGL`²⁵ környezetre épülő `rgl` és `djmrgl` csomagok segítségével nem csak létrehozhatunk háromdimenziós grafikákat, felületeket, hanem azokat térben forgathatjuk, nagyíthatjuk és mozgathatjuk. Példa az `rgl` csomag lehetőségeire:

```
> library(rgl)
> data(volcano)
> y <- 2 * volcano
> x <- 10 * (1:nrow(y))
> z <- 10 * (1:ncol(y))
> ylim <- range(y)
> ylen <- ylim[2] - ylim[1] + 1
> colorlut <- terrain.colors(ylen)
> col <- colorlut[ y-ylim[1]+1 ]
> rgl.clear()
> rgl.bg(color="white")
> rgl.surface(x, z, y, color=col)
```

Trellis

A Trellis grafikai környezetet eredetileg többváltozós adatállományok változói között fennálló kapcsolatok, interakciók exploratív vizualizációjára fejlesztették ki²⁶, és az *S/S-plus* környezetben implementálták először. Az R-környezetben a `lattice` és ennek alapját jelentő `grid` könyvtárak tartalmazzák azokat a függvényeket, amelyekkel Trellis típusú vizualizációt valósíthatunk meg. Itt is beszélhetünk magas, illetve alacsony szintű grafikai függvényekről, ezek azonban eltérnek az alapsomag grafikai függvényeitől. A `trellis.device()` használható a *trellis* grafikai eszköz megnyitására.

²⁴<http://www.ggobi.org>

²⁵<http://www.opengl.org>

²⁶<http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis>

Magas szintű függvények

A következő magas szintű grafikai függvények használhatók a trellis grafikai ábrák létrehozásában: `barchart`, `bwplot`, `cloud`, `contourplot`, `densityplot`, `dotplot`, `histogram`, `levelplot`, `parallel`, `qq`, `qqmath`, `rfs`, `splom`, `stripplot`, `tmd`, `wireframe`, `xyplot`.

Alacsony szintű függvények

Az előző függvényekkel létrehozott ábrákat kiegészíthetjük egyéb grafikai elemekkel, amire a következő függvényeket használhatjuk: `larrrows`, `llines`, `lplot.xy`, `lpoints`, `lsegments`, `lsegments`, `ltext`, `panel.arrows`, `panel.lines`, `panel.points`, `panel.segments`, `panel.text`.

Grafikai paraméterek

Az egyes grafikai paraméterek aktuális értékét a `trellis.par.get()` függvény segítségével kérdezhethetjük le. Az egyes paramétereket átállíthatjuk a `lset`, a `canonical.theme` vagy a `trellis.par.set()` függvények segítségével. Az utóbbi megoldás használata javasolható leginkább.

Programozás R-ben

Az R az egyszerűbb-bonyolultabb utasítások mellett lehetőséget nyújt komplex feladatok megvalósítására is. Ezen feladatok általában nem oldhatók meg egy-egy utasítás lefuttatásával, hanem programozást igényelnek, amikor is több (esetleg *vezérlőkön* keresztül egymásra épülő) utasítást használunk *ciklusokba* rendezve, *feltételes elágazásokba* terelve a folyamatokat. Amellett, hogy az R gazdag függvénytárát használhatjuk ezekben a programokban, magunk is készíthetünk függvényeket. Érdeemes saját függvényt készíteni olyan gyakran előforduló, összetett feladatok megoldására, amire nem találtunk kész függvényt az R-közösség készletében. Ha több ilyen függvényt készítettünk már és ezeket gyakran szeretnénk használni, akkor létrehozhatunk *saját csomagot* is, ami ezeket tartalmazza.

Az R-programok írásához érdemes olyan szövegszerkesztőt használni, ami együttműködik az R-környezettel (Emacs, Xemacs). A programkód szerkesztéséhez nagy segítséget nyújt, ha a szövegszerkesztőnk rendelkezik az R-nyelvnek megfelelő szintaktikai kiemeléssel (Tinn-R).

Vezérlők

Itt vezérlőkön a ciklusszervező eszközöket és a feltételes utasításokat értjük. Mindkét vezérlőtípus esetén gyakran kell megfogalmaznunk *feltételeket*. Ezekhez az R-ben, mint más programozási nyelvekben is, használunk összehasonlító, illetve logikai operátorokat (11. és 12. táblázat).

Ciklusok

A ciklusképzés vagy *looping* azt jelenti, hogy egy utasítást vagy blokkot ismételten futtatunk le. Kifejezetten a ciklusok kezelésére az R három utasítással rendelkezik, ezek a **for**, a **while** és a **repeat**. Mindhárom utasítás az utoljára értelmezett utasítás értékét adja vissza. Itt is lehetséges, habár ritkán alkalmazott megoldás, hogy az eredményt egy *objektumnak* adják át. A cikluson belüli folyamatellenőrzésre további két beépített szerkezet érhető el: a **next** és a **break**. A **break** és a **next** utasításokkal kiléphetünk egy ciklusból, illetve a ciklus következő elemére ugorhatunk. Mindkettőre igaz, hogy az utánuk álló utasítások nem értelmeződnek.

Az R-ben elérhetők egyéb utasítások is, amelyek tulajdonképpen hurkolásokat végeznek, ilyenek az **apply**, a **lapply**, a **mapply** és a **tapply**. Egyébként több operátor, különösen az aritmetikaiak *vektorizáltak*, így ciklus nélkül is minden elemen végrehajtódik a művelet.

A ciklusszervező függvények szintaxisa:

```
for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

A ciklusszervező függvények argumentumainak leírása:

11. táblázat. Összehasonlító operátorok

operátor	jelentés
<	kisebb mint
>	nagyobb mint
<=	kisebb vagy egyenlő
>=	nagyobb vagy egyenlő
==	egyenlő
!=	nem egyenlő

12. táblázat. Logikai operátorok

operátor	jelentés
<code>! x</code>	nem
<code>x & y</code>	és
<code>x && y</code>	és
<code>x y</code>	vagy
<code>x y</code>	vagy
<code>xor(x, y)</code>	exkluzív vagy

<code>cond</code>	Egyelemű vektor, amiben meghatározzuk a feltételt. Az értéke nem lehet NA. Ha több elemből álló vektorként adjuk meg, akkor (hibaiüzenet mellett) a vektor első elemében meghatározott feltétel szerint fut le a függvény.
<code>var</code>	Egy változó neve.
<code>seq</code>	Egy vektor (beleértve a listát is).
<code>expr</code>	Kifejezés(ek), ha több soros kifejezés-sorozat, akkor blokkban kell elhelyezni.

for

A `for` ciklusban egy vektor hosszában határozhatjuk meg, hogy az adott utasítás(oka)t hányszor ismételve meg az R-környezet .

```
for (változó in vektor)
  utasítás1
```

A vektor lehet *vektor* vagy *lista* is. A függvény mintegy végigfut a vektoron és minden eleménél végrehajtja az `utasítás1` parancsot. A ciklus befejezésekor a `változó` nevű változó továbbra is elérhető lesz, és értéke a vektor utolsó elemével egyezik meg.

```
> for (i in 1:3) cat(i, '\n')
```

```
1
2
3
```

repeat

A `repeat` utasítás mindaddig ismétli az adott utasítás értelmezését, amíg az szükséges. Ez a lehetőség egyben veszélyes is mert, könnyen vezethet végtelen ciklushoz. A szintaxis a következő:

```
repeat utasítás
```

Az utasításnak blokkot kell formálnia. Ahhoz, hogy kontrollálni tudjuk a folyamatot, a blokkon belül el kell helyezni egy kilépési feltételt is.

```
> i <- 0
> repeat {
+   i <- i + 1
+   cat(i, '\n')
+   if (i == 3) break}
```

```
1
2
3
```

while

Hasonló az előzőhöz, azonban magában a függvényben lehetőség van a folyamat kontrollálására.

```
while ( feltételes utasítás ) utasítás
```

A ciklusban elhelyezett utasítások addig ismétlődnek, amíg a feltételes utasításban meghatározott feltétel igaz, ha az hamis, akkor a ciklus befejeződik. Ha az utasítás soha nem értékelődik, akkor NULL értéket ad vissza, egyébként pedig mindig az utoljára lefuttatott utasítás eredményét.

```
> i <- 1
> while (i < 4) {
+   cat(i, '\n')
+   i <- i + 1}

1
2
3
```

Feltételes utasítások

if

Az `if`, illetve `if/else` utasítás feltételesen értelmez két utasítást. Amennyiben a megadott feltétel értéke igaz, akkor az első utasítás értelmeződik, egyébként a második. A szintaxis:

```
if ( feltételes utasítás )
  utasítás I.
else
  utasítás II.
```

Ha a feltételes utasítás eredménye nem logikai vagy numerikus, akkor hibaiüzenetet kapunk. Az `if/else` utasítás lehetőséget ad numerikus problémák (pl. a negatív szám logaritmus) elkerülésére. Minthogy az `if/else` egy utasítás, lehetőség van arra, hogy az eredménye értékadás során átadódjék egy objektumnak. A következő két példa ugyanazt az eredményt adja:

```
> n <- 2
> if (n > 0)
+ {k <- n} else
+ {k <- 0}
> k <- if (n > 0) n else 0
> k

[1] 2
```

Az `else` kitétel nem kötelező. Amennyiben az `if` utasítás nincsen blokkban, és van `else` kitétel, akkor annak ugyanabban a sorban kell szerepelnie, mint az `if`-nek. Ha nem így van, akkor a szintaktikailag teljes sor az új sor hatására értelmeződik. Ha az `if` utasítás blokkban van, és `else` kitételet is használunk, akkor az `else` az `if`-et lezáró kapcsos zárójellel (`}`) egy sorban kell, hogy kezdődjék. Az `if/else` utasításokat egymásba is ágyazhatjuk:

```
if ( feltételes utasítás I.)
  utasítás I.
else if ( feltételes utasítás II.)
  utasítás II.
else if ( feltételes utasítás III.)
  utasítás III.
else
  utasítás IV.
```

Az `else if` utasítások számának nincsen korlátja.

ifelse

Az `ifelse` függvény használata egyszerűsíti az `if/else` kifejezés-kombinációt. Míg a korábbi `if` függvényt alternatív kifejezés nélkül is lehet használni, az `ifelse`-t csak azzal együtt. Segítségével ciklusba ágyazás nélkül lehet objektumok elemeit tesztelni, és értéküktől függő értékadásokat elvégeztetni.

```
ifelse(test, yes, no)
```

A ciklusszervező függvények argumentumainak leírása:

<code>test</code>	A feltételt meghatározó kifejezés.
<code>yes</code>	Ha feltétel eredménye <code>TRUE</code> .
<code>no</code>	Ha feltétel eredménye <code>FALSE</code> .

Az `if/else` függvény használatával egyező eredményt adó formula:

```
> n <- 2
> k <- ifelse(n > 0, n, 0)
> k
```

```
[1] 2
```

Ha egy elemnél többel rendelkező objektumon használjuk a függvényt, akkor „megspórolhatunk” egy ciklust:

```
> sor <- c(1,2,1,3,1,4)
> k <- ifelse(sor < 2, 1, 2)
> k
```

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

switch

Míg a korábbi feltételes függvények legfeljebb két alternatíva között tesznek különbséget, a `switch` több alternatívát is lehetővé tesz.

```
switch(EXPR, ...)
```

A `switch` függvény argumentumainak leírása:

EXPR	E kifejezés értékeli a numerikus vagy szöveges értékeket.
...	Az alternatívák listája.

Példák:

```
> x <- rnorm(123, mean=0, sd=1)
> switch("mean",
+   mean = cat('átlag: ', mean(x), '\n'),
+   median = cat('medián: ', median(x), '\n'),
+   trimmed = cat('trimmelt átlag: ', mean(x, trim = .1), '\n'))
```

```
átlag: 0.0910466
```

Saját függvények készítése

A `function` függvény segítségével létrehozhatunk saját függvényeket.

```
function( arglist ) expr
return(value)
```

A `function` és `return` függvények argumentumainak leírása:

arglist	Egy vagy több nevet sorolhatunk fel, amelyek a függvény argumentumai lesznek. Meg lehet adni egyszerűen a nevet, vagy a nevet és a hozzá kapcsolódó kifejezést együtt (<code>name=expression</code>). Arra is van lehetőség, hogy nem adunk meg egy argumentumot sem.
expr	Egy vagy több kifejezés, amit a függvényünk végre fog hajtani.
value	A függvény visszatérési értéke, amely egy kifejezés vagy egy objektum.

Az alábbi egyszerű példában látható, hogy először meg kell szerkesztenünk, definiálnunk a függvényt, annak argumentumaival, illetve a függvény „belsejében működő” utasításokkal együtt. Miután megszerkesztettük a függvényünket, be is kell töltenünk, ami tulajdonképpen a `function` függvény futtatását jelenti. Vegyük észre, hogy itt az értékadáshoz nem a `<-`, hanem a `=` jelet használjuk. Miután betöltöttük az új függvényt, az a szokásos módon meghívható. A függvényünket egyszerűen végrehajthatjuk, vagy (értékadás útján) egy objektumnak adhatjuk át az értékét. Függvény definiálásának és használatának lépései:

1. A függvény definiálása és betöltése:

```
> elso.fuggvenyem <- function(x)
+ {
+   x + 1
+ }
```

13. táblázat. String-függvények

<u>függvény neve</u>
cat
deparse
formatC
grep
match, pmatch
nchar
parse
paste
strsplit
sub, gsub
substring
<u>toupper, tolower</u>

2. A függvény meghívása:

```
> elso.fuggvenyem(23)

[1] 24
```

Ebben az egyszerű példában a függvény által végrehajtott műveletek (összeadás) eredménye nem túl elegáns, érdemes lenne egyértelműbbé tenni az outputját.

A következő példában egy olyan függvényt definiálunk, mely egy numerikus vektorból néhány főbb leíró statisztikát számít, és azok eredményét egy könnyen értelmezhető outputba írja ki.

```
> leiro.statisztikak <- function(x)
+ {
+   leirok = paste('Elemszám:', '\t', length(x), '\n',
+ 'Átlag:', '\t', '\t', round(mean(x),3), '\n',
+ 'Szórás:', '\t', '\t', round(sd(x),3), '\n',
+ 'Varianscia:', '\t', round(var(x),3), '\n',
+ 'Minimum:', '\t', round(min(x),3), '\n',
+ 'Maximum:', '\t', round(max(x),3), '\n', sep='')
+   return(leirok)
+ }
```

Ebben a függvényben további függvényeket használtunk. A `paste` string-függvény (13. táblázat) segítségével az argumentumként megadott elemeket karakterláncá alakítja és összefűzi. A `paste` függvényben kétféle speciális karaktert kódoló karakterlánc is látható: a `\t` és a `\n`. Az előző a *tab*-ot, az utóbbi az *új sort* kódolja (14. táblázat). A függvényünkben használt leíró statisztikák számítását végző függvények és a `round` függvény (15. táblázat) eredményei szintén a `paste` argumentumai. A függvényünk tulajdonképpen összefűzi a szöveges elemeket, és a szöveges elemmé alakított számított értékeket a `leirok` objektumba írja be. Vegyük észre, hogy az értékadásnál nem a szokásos `<-`, hanem a `=` jelet használtuk. Az utolsó sorban lévő `return(leirok)` parancs a függvény visszatérési értékét határozza meg. Ez azt jelenti, hogy ha meghívjuk a `leiro.statisztikak` függvényünket, akkor annak eredménye a `leirok` objektum tartalma lesz.

```
> x <- rnorm(123, mean=0, sd=1)
> res <- leiro.statisztikak(x)
> cat(res)
```

```
Elemszám:      123
Átlag:         -0.081
Szórás:        0.985
Varianscia:    0.97
Minimum:       -2.157
Maximum:       2.247
```

A példában generálunk egy véletlen számokat tartalmazó vektort, ami a függvény meghívásakor annak argumentuma lesz. Függvényünk eredményét egy értékadáson keresztül beírjuk a `res` objektumba. A `res` objektumot a `cat` függvénnyel formázzuk, és kihatjuk a terminálba.

14. táblázat. Speciális karakterek

kódolás	eredménye
\'	apoztróf
\"	idézőjel
\n	új sor
\r	soremelés
\t	horizontális tab
\b	backspace
\a	hangjelzés
\f	oldaltörés
\v	függőleges tab
\\	backslash

15. táblázat. Általános függvények

R-függvény	rövid leírása
sqrt	négyzetgyök
log	természetes logaritmus
log10	10 alapú logaritmus
exp	exponenciális
abs	abszolút érték
round	a legközelebbi egész számra kerekít
ceiling	felfelé kerekít
floor	lefelé kerekít
sin, cos, tan	szinusz, koszinusz, tangens
asin, acos, atan	arkusz szinusz, arkusz koszinusz, arkusz tangens
sum(x)	x elemeinek összege
prod(x)	x elemeinek szorzata
max(x)	x legnagyobb értéke
min(x)	x legkisebb értéke
which.max(x)	x melyik eleme x legnagyobb értéke
which.min(x)	x melyik eleme x legkisebb értéke
range(x)	x terjedelme, meggyevezik a $c(\min(x), \max(x))$ vektorral
length(x)	x elemszáma
mean(x)	x lemeinek átlaga
median(x)	x elemeinek mediánja
var(x) vagy cov(x)	x elemeinek varianciája ($n-1$ alapú), ha x egy mátrix vagy data.frame variancia-kovariancia mátrix az eredmény
cor(x)	ha x mátrix vagy data.frame, akkor korrelációs mátrix (ha vektor, akkor 1)
var(x,y) vagy cov(x,y)	kovariancia x és y között, ha x és y mátrix vagy data.frame, akkor azok oszlopai között
cor(x,y)	lineáris korreláció x és y között, vagy korrelációs mátrix, ha mátrixok vagy data.frame-ok

x numerikus vektor

Jelentések készítése

Gyakori igény, hogy a statisztikai eredmények (akár szöveges, akár grafikus formában) egy dokumentumban összefoglalva jelenjenek meg. A korábbiakban láthattuk, hogy az R-környezetben végzett műveletek eredményei kimenthetők fájllokba, adatbázisokba, illetve a vágólapra. Az így exportált részek beilleszthetők szövegszerkesztő, kiadványszerkesztő szoftverek dokumentumaiba. A \LaTeX tördelési rendszerhez az R többféle kimenettel is rendelkezik (pl.: `pic`, `table`), a két környezet együttes alkalmazása igen hatékony lehet. Az elterjedtebb irodai szoftvercsomagok (pl. *Open Office*, *KOffice*, *StarOffice* vagy *MS Office*) szövegszerkesztő eszközeivel szintén készíthetjük jelentéseket, az R-környezetből származó kimenetek felhasználásával.

Ha több ábrát, illetve szöveges kimenetet kívánunk beilleszteni a készülő jelentésünkbe, és ezt „manuálisan” szeretnénk megvalósítani, akkor hosszadalmas, több hibalehetőséget hordozó utat választunk. Gondoljunk arra, hogy ha több képet is beillesztünk, akkor azok elnevezésében akkurátusan kell eljárni, de ha ezt még meg is tettük, a szoftverekben elérhető fájlból való beszúrási rutinokban működésből fakadóan könnyen más képet illesztünk be az adott helyre, mint amit szerettünk volna. Ha ezt meg is oldottuk, és (nagyon figyelmesen) minden ábrát a helyére tudtuk tenni, szükség lehet az elemzések, így az ábrák elkészítésének módosított vagy más adatokon alapuló megismétlésére. Ekkor pedig kezdhetjük előlről az egészet, a keveredések veszélyével terhelve. Szerencsére létezik egy eszköz, ami az R-környezetben is elérhető és lehetővé teszi, hogy olyan dokumentumokat hozzunk létre, amelyekben a programkódotól kezdve, az eredményeken és ábrákon keresztül, az értelmezésig minden egészét dinamikusan kezelhető. Ez a **Sweave**.

Sweave

A *Sweave* lehetőséget ad arra, hogy a *dokumentációs* szövegrész(ek)e)t és az *R-kódo*(ka)t egy *noweb* szintaxisú forrásállományban szerkesszük, majd az R-értelmezőn lefuttatva \LaTeX ²⁷ állományt kapjunk vissza eredményül az alábbi elemekkel:

- dokumentációs szöveg
- R-input és/vagy
- R-output (szöveg vagy grafika)

Ez a megoldás lehetőséget biztosít arra, hogy jelentésünket újrageneráljuk, ha megváltoztak a forrásadataink, illetve emellett az analízisben használt kódot, eljárást is dokumentálhatjuk ugyanazon jelentésben. Azon R-felhasználóknak, akik a \LaTeX -ben is dolgoznak, további előny, hogy nem kell új szintaxist és szoftverkezelést tanulniuk.

Noweb fájlok

A *noweb* (Ramsey, 1998) olyan dokumentáló-programozási eszköz, ami lehetővé teszi programozási forráskód és a rá vonatkozó dokumentáció kombinálását egyetlen fájlban. Különböző szoftverek lehetővé teszik a dokumentáció és/vagy a forráskód kivonását. A *noweb* fájl egyszerű szöveges állomány, ami tartalmazza a programkódot és a dokumentációs szakaszokat (*chunk*):

Dokumentációs szakasz Olyan sorral kezdődik, aminek az első karaktere @, amit szóköz vagy új sor követ. A sor további része megjegyzés lesz, vagyis nem értelmeződik. Általában a dokumentációs szakasz jelölőnyelven írt szöveg, pl. \LaTeX .

Kód szakasz Az első sora `<<name>>=` szöveggel kezdődik, a sor folytatása szintén megjegyzés lesz és nem értelmeződik.

Az első szakasz alapértelmezésben mindig dokumentáció.

²⁷<http://www.inf.unideb.hu/~matex/>

Sweave-állományok

A Sweave-forrásfájlok szabályos *noweb* fájlok néhány kiegészítő szintaktikai elemmel, amelyek lehetővé teszik különböző kiegészítő beállítási lehetőségek alkalmazását a végső output formázásnak érdekében. Hagyományosan a *noweb*-fájlok kiterjesztése `.nw`, amely a Sweave-fájloknál szintén lehetséges. A Sweave-fájlokat általában `.rnw`, `.Rnw`, `.snw` és `.Snw` kiterjesztésekkel használják, jelezve azt, hogy *noweb* stílusú Sweave-fájlok. A továbbiakban `.rnw` kiterjesztést használunk.

Példa

Egy egyszerű Sweave fájlt mutat a 12. ábra, amelyben a \LaTeX -fájlba két kódrész van beágyazva. Ha beállítottuk a munkakönyvtárunkat, akkor a következő kódot kell futtatnunk az R-környezetben:

```
> Sweave('sweavepelda.rnw')
```

```
Writing to file sweavepelda.tex
Processing code chunks ...
 1 : echo term verbatim
 2 : term verbatim eps pdf
```

You can now run LaTeX on sweavepelda.tex

A Sweave a `.rnw` állományból létrehozott egy \LaTeX -fájlt, amit a 13. ábrán láthatunk. Az első különbség a két dokumentum között, hogy a `Sweave.sty` \LaTeX stílus betöltését szolgáló utasítás (C:/R/rw2001/share/texmf/Sweave) automatikusan beíródik a \TeX állományunkba. Ez teszi lehetővé, hogy az `Sinput` és `Soutput` környezeteket a \LaTeX értelmezni tudja. A dokumentációs szöveg változatlan formában átmásolódik a `sweavepelda.rnw` fájlból a `sweavepelda.tex` állományba. A kódrészeket azonban (annak függvényében, hogy azok inputok vagy outputok) a `Sinput`, illetve `Soutput` környezetekbe illeszti be a Sweave. Természetesen az outputok úgy jönnek létre, hogy az inputokat értelmezi az R. A `Sinput` és `Soutput` környezeteknek `\begin{Schunk}` és `\end{Schunk}` által határoltan kell a \TeX állományban szerepelniük.

A második kódszakasz egy olyan Sweave lehetőséget mutat be, ami kiegészítés a *noweb* szintaxishoz képest: a kódrész neve segítségével a Sweave-vel utasításokat tudunk közölni. Ezek segítségével kontrollálhatjuk a végleges outputot.

- A kódrészt úgy jelöltük meg, hogy ábra-szakaszként értelmezze (`fig=TRUE`), így a Sweave létrehoz egy EPS és egy PDF állományt, amelyek a kódrészben lévő kódnak megfelelő grafikát tárolja. Továbbá beszúr egy `\includegraphics{sweavepelda-001}` utasítást a \LaTeX -állományba.
- Az `echo=FALSE` argumentum azt állítja be, hogy az R-input ne kerüljön bele a végleges dokumentumba (nem lesz `Sinput` környezet).

Sweave beállítások

A beállítási lehetőségek segítségével meghatározhatjuk, hogy az `.rnw` fájlban tárolt kódrészek és azok outputjai (szöveg, ábra), hogyan íródjanak át a `.tex` állományba. Minden opció ugyanolyan formájú `argumentum=érték`, ahol az érték lehet szám, szöveg vagy logikai érték. Egyszerre több argumentum is beállítható (vesszővel elválasztva), mindegyik argumentumnak értéket kell adnunk (ezek nem tartalmazhatnak vesszőt vagy egyenlőségjelet). A logikai argumentumok értékadásánál használható a `true`, a `false`, illetve ezek kezdőbetűi (`t`, `f`), a nagybetűs változatok is működnek.

Az `.Rnw` fájlban az opciók a következőképpen adhatók meg:

1. A kódrész kezdetén a szögletes zárójelek (`<<<>>`) közé helyezhetjük el a beállítandó argumentumokat, az így megadott beállítások csak az adott kód chunkra vonatkoznak.
2. A dokumentumban bárhol elhelyezhető a következő utasítás:

```
\SweaveOpts{arg1=érték1, arg2=érték2, ..., argN=értékN}
```

, ami módosítja az alapbeállításokat az utasítás utáni dokumentum-szakaszra vonatkozóan. Ennek megfelelően, ha a dokumentum elején helyezzük el ezt az utasítást, akkor az az összes kódszakaszra vonatkozóan átállítja az alapbeállításokat.


```
\documentclass[a4paper]{paper}

\usepackage{graphicx}
\usepackage[latin2]{inputenc}
\usepackage[magyar]{babel}
\usepackage[T1]{fontenc}

\title{Sweave-példa}

\begin{document}
\maketitle
```

Ebben a példában a `\LaTeX{}` dokumentumunkba két kódot illesztettem be. Az alábbi kódrészben véletlen adatok generálódnak és az `\verb|adat|` objektumnak adódnak át.

```
<<>>=
```

```
adatok <- rnorm(800,0,1)
```

```
@
```

A második kódrész létrehoz egy hisztogrammot.

```
\begin{figure}[h]
\begin{center}
<<fig=TRUE, echo=FALSE>>=
hist(adatok, main="", col="red", ylab="gyakoriság")
@
\caption{Példa ábra}
\label{swxplhist}
\end{center}
\end{figure}
```

Ide jöhetne egy szöveg, amiben összefoglalhatnám az ábra alapján megfogalmazható következtetéseket.

```
\end{document}
```

12. ábra. sweavepelda.rnw

```
\documentclass[a4paper]{paper}
```

```
\usepackage{graphicx}
```

```
\usepackage[latin2]{inputenc}
```

```
\usepackage[magyar]{babel}
```

```
\usepackage[T1]{fontenc}
```

```
\title{Sweave-példa}
```

```
\usepackage{C:/R/rw2001/share/texmf/Sweave}
```

```
\begin{document}
```

```
\maketitle
```

Ebben a példában a `\LaTeX{}` dokumentumunkba két kódot illesztettem be.

Az alábbi kódrészben véletlen adatok generálódnak és az `\verb|adat|` objektumnak adódnak át.

```
\begin{Schunk}
```

```
\begin{Sinput}
```

```
> adatok <- rnorm(800, 0, 1)
```

```
\end{Sinput}
```

```
\end{Schunk}
```

A második kódrész létrehoz egy hisztogrammot.

```
\begin{figure}[h]
```

```
\begin{center}
```

```
\includegraphics{sweavepelda-001}
```

```
\caption{Példa ábra}
```

```
\label{swxplhist}
```

```
\end{center}
```

```
\end{figure}
```

Ide jöhetne egy szöveg, amiben összefoglalhatnám az ábra alapján megfogalmazható következtetéseket.

```
\end{document}
```

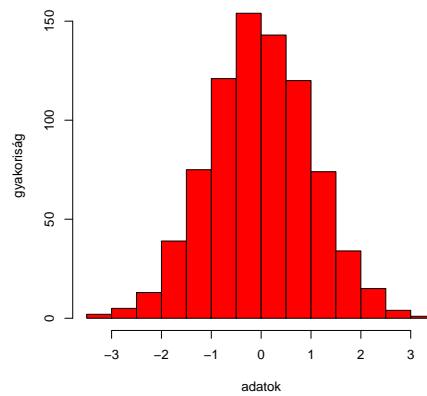
13. ábra. sweavepelda.tex

Sweave-példa

Ebben a példában a \LaTeX dokumentumunkba két kódot illeszttem be. Az alábbi kódrészben véletlen adatok generálódnak és az `adat` objektumnak adódnak át.

```
> adatok <- rnorm(800, 0, 1)
```

A második kódrész létrehoz egy hisztogrammot.



1. ábra. Példa ábra

Ide jöhetne egy szöveg, amiben összefoglalhatnám az ábra alapján megfogalmazható következtetéseket.

A meghajtótól függően különböző opciók használhatók. Minden meghajtó képes kezelni legalább a következő opciókat (az értékek az alapértelmezett értékek):

engine=S: karakterlánc, ami leírja, hogy melyik S motor elérhető a kód chunkok értelmezésére. A lehetséges értékek pl.: S, R, S3 vagy S4. Mindegyik meghajtó csak a kód chunkokat értelmezi, a többi részt figyelmen kívül hagyja.

split=FALSE: logikai érték. Ha TRUE, akkor az output több fájlba kerül, ha FALSE, akkor egy fájl lesz az eredmény. A részletek meghajtótól függetlenek.

label: szöveges címke, ami a fájlnevek készítéséhez használandó, ha a **split=TRUE** értékre van állítva. Ha a **label** értékét **label.engine** formában adjuk meg, akkor a kiterjesztés el lesz távolítva, mielőtt további alkalmaznánk (pl.: a **hello.S** címke **hello**-vá lesz egyszerűsítve).

Az első (de csak az első) argumentum értéke állhat magában az argumentum neve nélkül, ekkor az **label**-ként értelmeződik:

```
<<hello.S, split=FALSE>>
```

, ami ugyanaz, mint a következő:

```
<<split=FALSE, label=hello.S>>
```

, de a

```
<<split=FALSE, hello.S>>
```

forma hibát fog generálni.

Objektumok használata a szövegben

Korlátozottan lehetőség, de van arra is, hogy az R-objektumok értékét a dokumentációban „beágyazva” használjuk. Ha az **Sexpr{kifejezés}** környezetben helyezük el az adott objektum nevét, akkor a szövegben annak az értéke meg fog jelenni. A kifejezés lehet objektum vagy valamilyen kifejezés, művelet. Mielőtt ezt használjuk, a kódszakaszban érvényes forrása kell, hogy legyen. A kifejezésben használhatunk R-függvényeket, viszont a kapcsos zárójel nem alkalmazható. Amennyiben ilyen kifejezés használata szükséges, akkor a kódszakaszban kell elvégeztetni és az eredményét meghívni a **\Sexpr** utasítással.

A kódszakasz újrahasznosítása

A névvel rendelkező kódszakaszok újrahasznosíthatók a dokumentumon belül. Álljon itt egy egyszerű példa:

```
<<a>>=
x <- 10
@
```

```
<<b>>=
x + y
@
```

```
<<c>>=
<<a>>
y <- 20
<<b>>
@
```

, ami egyenértékű a következő kóddal:

```
<<c>>=
x <- 10
y <- 20
x + y
@
```

A kódszakaszra utaló operátor (<<>>) csak név argumentummal rendelkezik, más Sweave-opció nem használható benne.

Tangle vagy weave

A Sweave-rendszert két S-függvénnyel érhetjük el, ezek a `Stangle()` és a `Sweave`, mindkettő része az alap R-telepítésnek. A `Stangle` az `.rnw` állományból csak a kódszakaszokat olvassa ki és értelmezi, majd egy vagy több fájlba kiírja. A `Sweave()` futtatja a kódrészeket az S-motoron és az eredményekkel, illetve a dokumentációs szöveggel összefűzi egy állományba. A `Stangle()` függvény a `Rtangle` meghajtót, míg a `Sweave` `RweaveLatex`-ot használ.

Az RweaveLatex paraméterezése

Az `RweaveLatex` meghajtó az alábbiakban leírt beállítási lehetőségeket támogatja a kódrészek felügyeletéhez:

<code>echo</code>	Ha az értéke az alapértelmezett <code>TRUE</code> , akkor az outputban meg fog jelenni az R-kód is. Egyébként nem.
<code>eval</code>	Ha az alapértelmezett <code>TRUE</code> helyett <code>FALSE</code> értéket adunk meg, akkor az adott kódszakaszt nem értelmezi az R.
<code>results</code>	Egy karakterláncként adhatjuk meg az outputban szereplő karakterek megjelenítésének típusát. Az alapértelmezés <code>verbatim</code> , ha <code>tex</code> -re állítjuk, akkor a <code>TEX</code> stílusnak megfelelően jelenik meg az output szövege. Ha <code>hide</code> értéket adunk meg, akkor nem generál outputot, viszont a kódszakaszt értelmezi.
<code>print</code>	Ha az alapértelmezett <code>FALSE</code> értéket <code>TRUE</code> -ra állítjuk, akkor a kódszakasz minden kifejezése még az értelmezés előtt be lesz illesztve a <code>print()</code> függvénybe, így a kifejezések értékei láthatók lesznek az outputban.
<code>term</code>	Ha az alapértelmezett <code>TRUE</code> értékű, akkor az értékadások értéke nem lesz megjelenítve, míg az objektumoké igen. Ha viszont <code>FALSE</code> értéket adunk meg, akkor csak azok az értékek lesznek kiírva az outputba, amelyek esetében a <code>print</code> vagy a <code>cat</code> utasítást használtuk.
<code>split</code>	Ha az alapértelmezett <code>FALSE</code> helyett <code>TRUE</code> értéket adunk meg, akkor minden kódrésznek megfelelően külön fájlba íródnak az outputok.
<code>strip</code>	Ha az értéke az alapértelmezett <code>TRUE</code> , akkor az üres sorokat az output elejéről és végéről eltávolítja, ha <code>FALSE</code> , akkor nem foglalkozik velük.
<code>prefix</code>	Ha az alapértelmezett <code>TRUE</code> értéket használjuk, akkor a létrehozott ábrákhoz és a szöveges outputokhoz egy általános kiterjesztést illeszt.
<code>prefix.string</code>	Alapértelmezésben <code>.Snw</code> .
<code>include</code>	Logikai értéke arra utal, hogy az adott kódrész által kódolt szöveges és grafikus output egy helyen legyen a végleges dokumentumban, vagy sem. Ha <code>FALSE</code> értéket adunk meg, akkor a szöveges és a grafikus eredmények külön outputként lesznek a <code>.tex</code> állományba kiírva. Az alapértelmezése <code>TRUE</code> .
<code>fig</code>	Logikai értéke arra utal, hogy a kódrész ábrát kódol, vagy sem. Az alapértelmezése <code>FALSE</code> .
<code>eps</code>	Ha az értéke <code>TRUE</code> , akkor az ábrát elmenti <code>.eps</code> állományba, ha <code>FALSE</code> , akkor nem. Az alapértelmezés <code>TRUE</code> .
<code>pdf</code>	Ha az értéke <code>TRUE</code> , akkor az ábrát elmenti <code>.pdf</code> állományba, ha <code>FALSE</code> , akkor nem. Az alapértelmezés <code>TRUE</code> .
<code>width</code>	Az ábra szélességét határozza meg hüvelykben, az alapértelmezés 6.
<code>height</code>	Az ábra magasságát határozza meg hüvelykben, az alapértelmezés 6.

Függelék

Telepítés

Windows

Windowsra a telepítőkészlet egyetlen bináris állomány, ami a <http://cran.r-project.org/> oldalról letölthető. Telepíthető a Windows 95, 98, ME, NT4.0, 2000 and XP operációs rendszerekre. A bináris telepítő állomány (pl. `rw2001.exe`) telepítési képernyőit mutatja a 15-22. ábra.

Az alapértelmezett telepítési hely a `C:\Program Files\R\` könyvtár, amelyen belül létrehoz a telepített verzióknak megfelelő könyvtárstruktúrát (23. ábra). Egyszerre több verzió is futhat az adott operációs rendszeren. Az alap-telepítőkészlettel néhány csomag is telepítésre kerül (`base`, `datasets`, `graphics`, `grDevices`, `grid`, `methods`, `splines`, `stats`, `stats4`, `tcltk`, `tools`, `utils`). Ezek a csomagok mint könyvtárak kerülnek bejegyzésre az R fájlstruktúrájába, a `library` könyvtárba. (A `stats` csomag belső könyvtárszerkezetét mutatja a 24. ábra.) A csomag fáján belül szereplő `html` könyvtár tartalmaz egy lefordított `.html` fájlt, ami az adott csomag súgója, egy fájlba rendezve, igen hasznos lehet a könyvtár funkcióinak tanulmányozásában.

Csomagok telepítése

- A CRAN-ról illetve a Bioconductor oldaláról közvetlenül telepíthetünk csomagokat:
 - A Windows RGui `Packages` menüjéből kiválasztjuk a `Install package(s)...` almenüt, aminek következtében megjelenik a 38. ábrán látható lista, amivel megadhatjuk azt a CRAN tüköroldalt, ahonnan telepíteni szeretnénk, ami után az R-verzióinkhoz elérhető csomagok listája jelenik meg egy újabb űrlapon (40. ábra). A listából kiválasztva a kívánt csomagot, az telepítődik. Ebben az esetben azok a csomagok telepítődnek, amelyekről a kiválasztott könyvtár működése függ. Egyszerre több csomagot is ki lehet választani.
- A CRAN-ról letölthetők `.zip` kiterjesztéssel különböző csomagok. Ezek telepítése a következő módon valósítható meg:
 - A Windows RGui `Packages` menüjéből kiválasztjuk az `Install package(s) from local zip files...` almenüt. A megjelenő fájlkezelő segítségével kiválasztjuk a csomagot tartalmazó, letöltött, zippelt állományt.

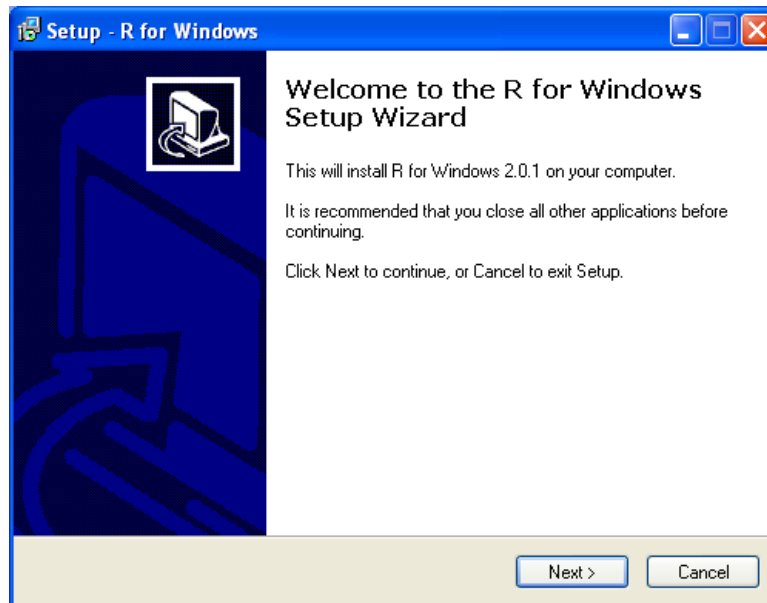
Linux

A bináris állományok elérhetők a CRAN-on néhány disztribúcióhoz, amelyek egyszerűen telepíthetők az adott platformon. A forráskód szintén letölthető és a következő módon telepíthető:

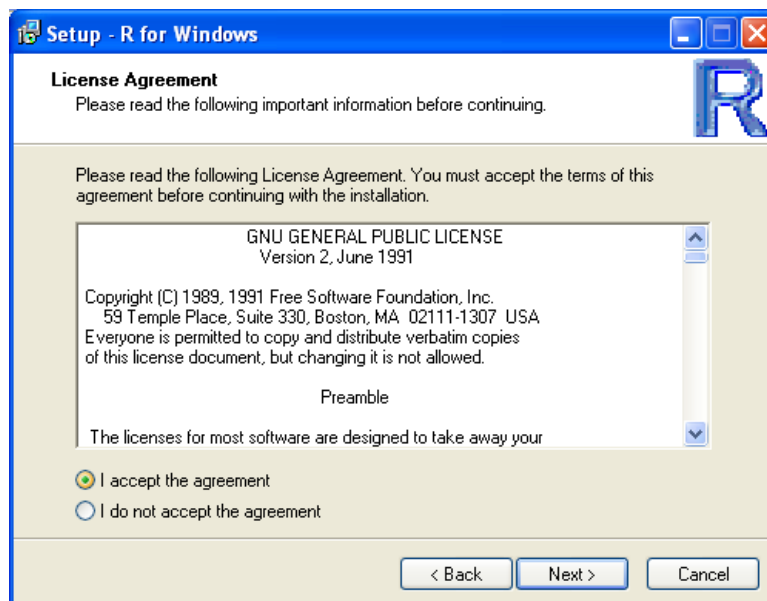
```
./configure
make
make install
```

Csomagok telepítése

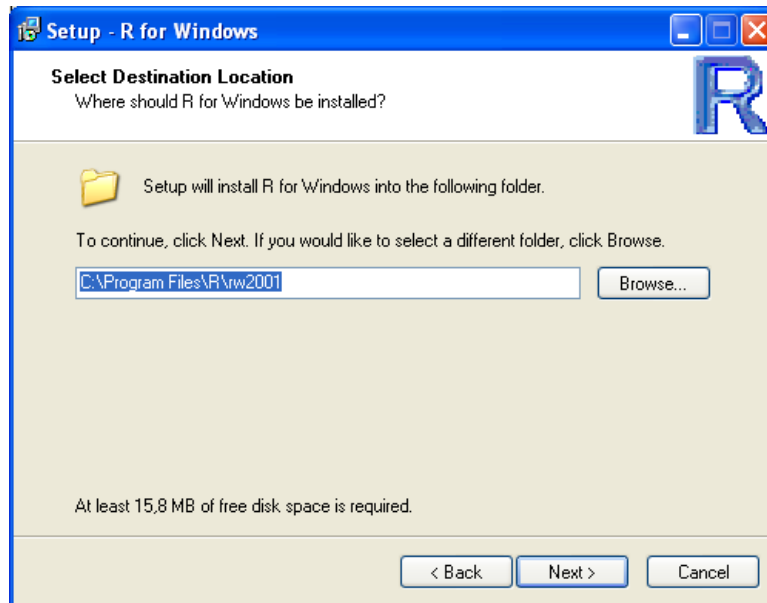
Vagy bináris telepítőt töltünk le a telepített R verzióinkhoz, vagy forrásból telepítünk. Ez utóbbit egy terminálban `root`-ként hajthatjuk végre az `R CMD INSTALL` csomag utasítással, ahol a `csomag` a letöltött és telepítendő csomagunk helye és neve. Előfordul, hogy bizonyos csomagok telepítése feltételezi más csomagok telepítetttségét.



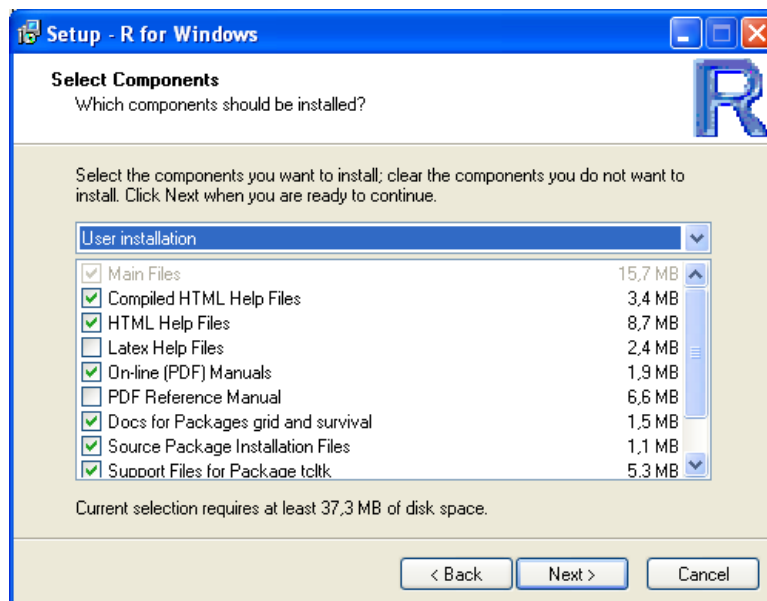
15. ábra. Windows telepítési képernyő 1.



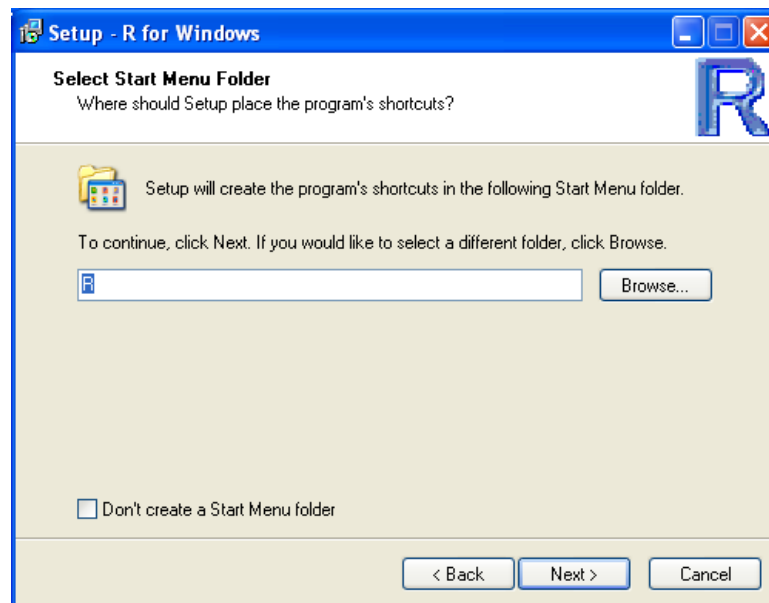
16. ábra. Windows telepítési képernyő 2.



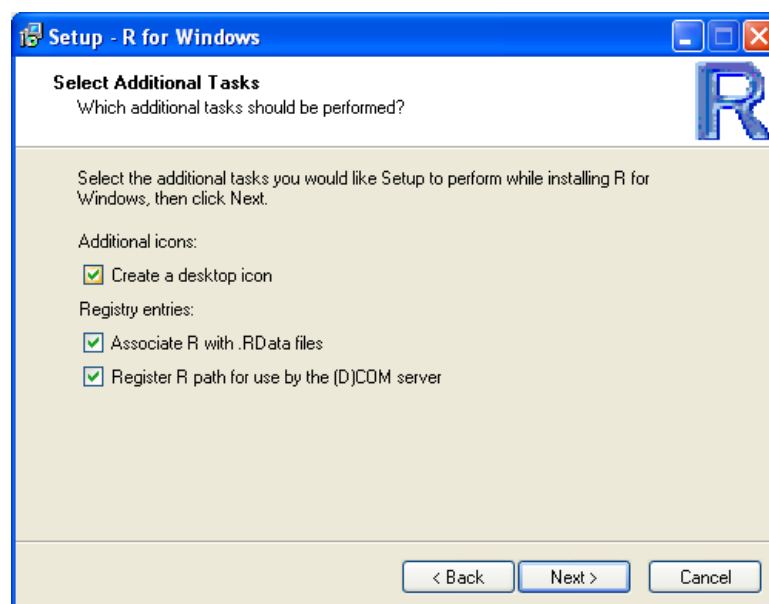
17. ábra. Windows telepítési képernyő 3.



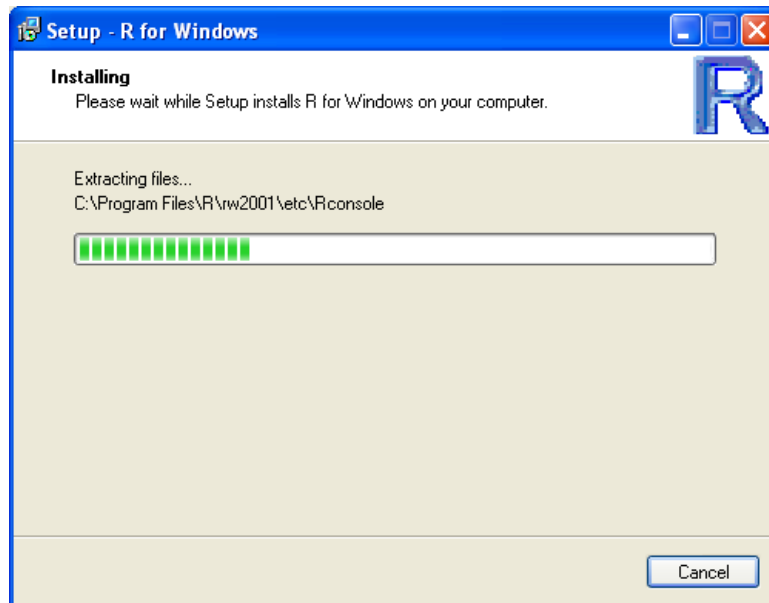
18. ábra. Windows telepítési képernyő 4.



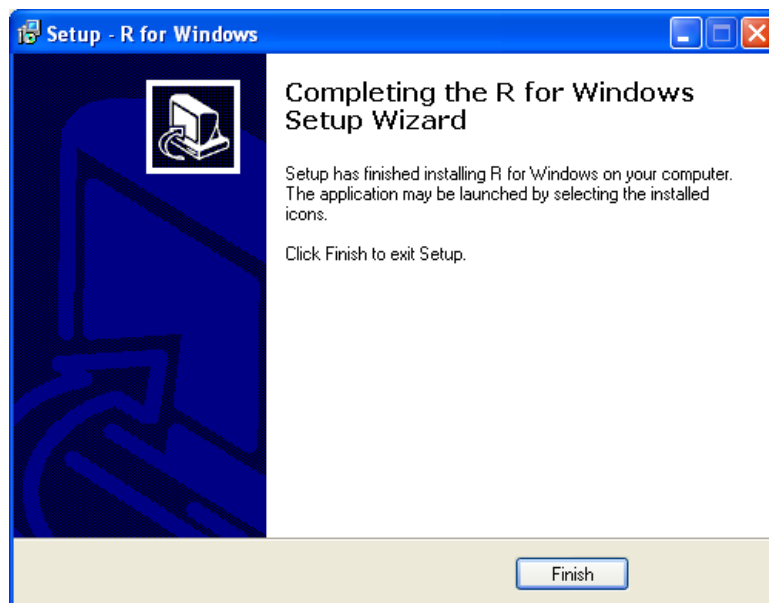
19. ábra. Windows telepítési képernyő 5.



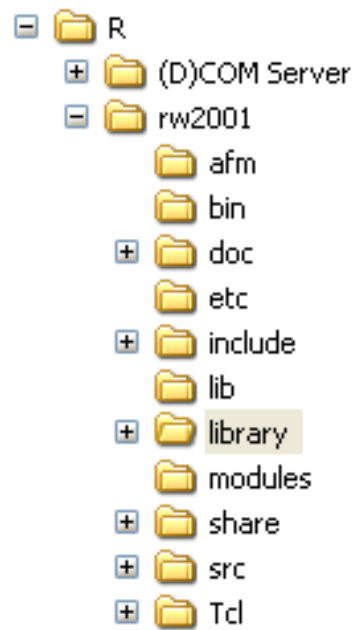
20. ábra. Windows telepítési képernyő 6.



21. ábra. Windows telepítési képernyő 7.



22. ábra. Windows telepítési képernyő 8.



23. ábra. Windows telepítési fa

24. ábra. A *stats* csomag fastruktúrája

Szövegszerkesztők

Tinn-R

A Tinn-R egyszerű szövegszerkesztő R-szkript szerkesztését, és az R-interpreterrel való „felhasználóbarát” együttműködést könnyíti meg. Előnyei:

- Egyszerre több szkript fájljal is lehet dolgozni
- Az R-szintaxisnak megfelelő szövegkiemelés
- Menüből kezelhető R-műveletek
- Együttműködik R-felületekkel:
 - Rgui
 - R Konzol
 - SciViews R Konzol (99. lap)
- Az R mellett lehetőséget nyújt való együttműködésre az S-Plus-szal is
- Egyéb nyelveknek megfelelő szövegkiemelésre is képes, így pl. az adatbázisokkal való munka során használatos SQL-kódok szerkesztésére is igen hasznos eszköz

A Tinn-R letöltése és telepítése után bizonyos beállításokat el kell végeznünk ahhoz, hogy a telepített R-rendszerrel együttműködhessen:

1. Az **Options** menüből a **Main** almenü alatt található **Application** almenüt választva a 25. ábrán látható űrlap jelenik meg
2. Két beállítást érdemes megváltoztatni az űrlapon:
 - A **Starting comment** elnevezésű szövegdobozba az R-nelvnek megfelelő **#** jelet kell beírni
 - Az **Rgui** feliratú gombra kattintva, a megjelenő fájlkezelő segítségével kiválasztjuk azt a parancsértelmező felületet, amit a későbbiekben használnánk

Emacs

Az *Emacs* szövegszerkesztőről, illetve használatáról számos ismertető érhető el az interneten²⁸, ezért itt erre nem térünk ki, csupán azt szeretném bemutatni, hogy az R-környezettel hogyan lehet összekapcsolni.

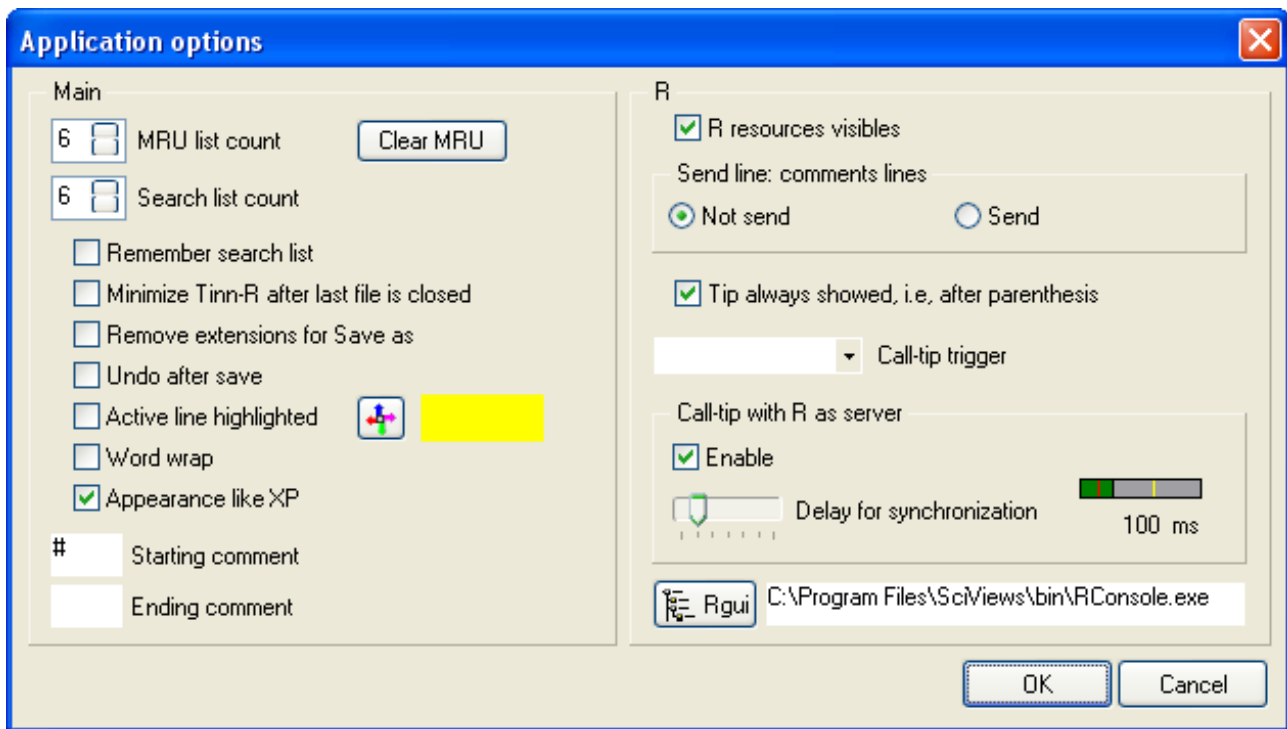
Telepítés

A telepítéshez le kell töltenünk a szoftver telepítőállományát²⁹. A tömörített állományt ki kell csomagolnunk egy könyvtárba, jó, ha az elérési útvonalban nincsen szóköz, mondjuk legyen **C:\emacs**. A kicsomagolás után ajánlott, de nem feltétlenül szükséges a **bin** alkönyvtárban lévő **addpm.exe** futtatása. Ez elvégez néhány beállítást, többek között a *Programok* közé betesz egy a **C:\emacs\bin\runemacs.exe** fájlra mutató linket. Tanácsos, és a későbbi *ESS* telepítéshez szükséges is egy **.emacs** vagy **_emacs** fájl telepítése is. Ebben az *Emacs* különböző beállításait a *Lisp*³⁰ nyelv alkalmazásával testreszabhatjuk. Ha valaki nem jártas ebben a nyelvben,

²⁸<http://www.cs.elte.hu/linfo/Szoveg/Emacs/>

²⁹<ftp://ftp.gnu.org/gnu/emacs/windows/emacs-21.3-fullbin-i386.tar.gz>

³⁰<http://cons.org>



25. ábra. Tinn-R beállítási űrlap

akkor le is lehet tölteni különböző feladatoknak megfelelően optimalizált beállítási állományokat³¹. Az *Emacs* a beállítási állományt először a HOME könyvtárban keresi, ha ott nem találja, akkor a C:/ gyökérkönyvtárban próbálkozik. Ha valamely könyvtárban talál *.emacs* és *_emacs* állományt is, akkor az előzőt fogja használni, az utóbbit figyelmen kívül hagyja. Szerintem a legegyszerűbb, ha a gyökérkönyvtárba másoljuk a fájlt.

ESS

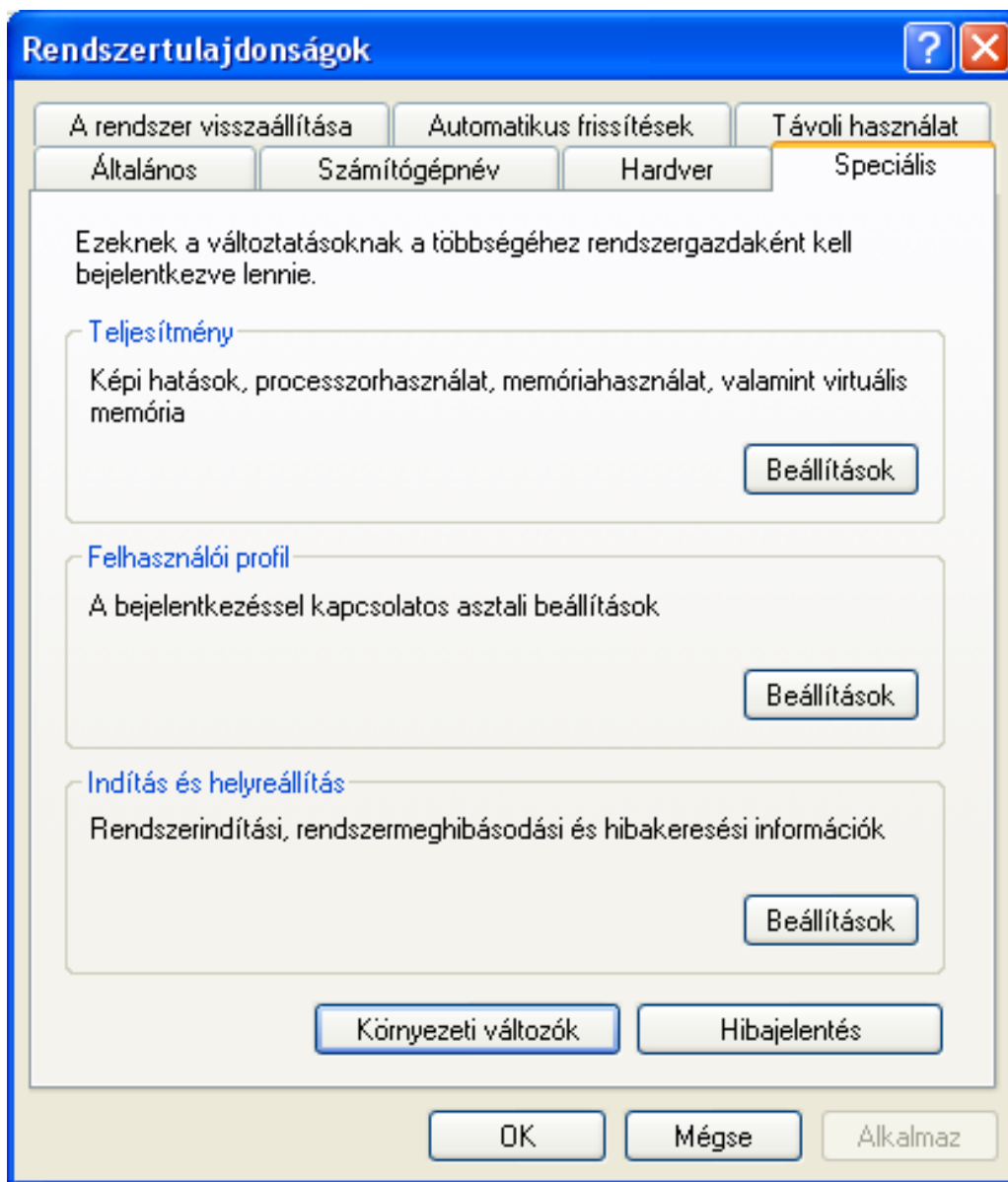
Az *Emacs Speaks Statistics (ESS)* egy általános interfész, amelyen keresztül az emacs kapcsolatot tud teremteni statisztikai szoftverekkel. Jelenleg lehetőség van az S, az R, a SAS, a BUGS, a Stata és az XLisp-Stat statisztikai alkalmazásokkal, környezetekkel való együttműködésre. Az *ESS* szabadon letölthető az internetről³² és számos kimerítő dokumentáció³³ érhető el a használatával kapcsolatban, itt most csak az installálásra térnék ki. A telepítés lépései:

- A letöltött tömörített állományt (pl. *ess-5.2.10.zip*) csomagoljuk ki az *Emacs* könyvtárba. (Az előző példánál maradva az C:\emacskönyvtárba.) A példa szerint C:\emacskönyvtár\ess-5.2.10 lesz az *ESS* telepítési könyvtára. Természetesen a verziószám változhat.
- A Microsoft Windows környezeti változói között a PATH-ban szerepelnie kell a használni kívánt R-konzol elérési útjának. Ezt a Windows 9x operációs rendszerek esetén az c:\autoexec.bat fájlban tudjuk beállítani, a következő sor beillesztésével: `path=%PATH%;C:\program~1\R\rw2001\bin`. Természetesen az `rw...bin` részben a kipontozott helyen azt a verziószámot kell megadnunk, amit használni kívánunk. Ha Windows NT/2000/XP környezetben dolgozunk, akkor a beállítást a következő lépésekben tudjuk megoldani: a *Start* menüből a *Beállítások* közül kiválasztjuk a *Vezérlőpultot*. A megjelenő parancsikonok közül ki kell választanunk a *Rendszer* feliratút. A megjelenő *Rendszertulajdonságok* elnevezésű űrlapon kiválasztjuk a *Speciális* címkével rendelkező fület (26. ábra). Itt a *Környezeti változók* gombra kattintva megjelenik a *Környezeti változók* című űrlap (27. ábra), amin ki kell választanunk a *Rendszerváltozók* feliratú (alsó) listából a *Path* sort. A *Szerkesztés* gombra kattintva előtűnő *Rendszerváltozók szerkesztése* ablakban (28. ábra) a *Változó értéke* mezőben keressük meg az R-környezetre vonatkozó bejegyzést. Ha nem találunk, akkor a sor végére, az utolsó elemről pontosvesszővel elválasztva írjuk be a C:\program~1\R\rw2001\bin utat. Fontos, hogy a szóközöket lehetőleg mellőzzük, ezért ha a C:\Program Files könyvtárban van az R telepítés, akkor ehelyett a könyvtárnév helyett használjuk a C:\program~1 elnevezést. Ha beírtuk az utat, akkor az *OK* gombbal jóváhagyjuk, majd az aktívá váló (27. és 28. ábrán látható) űrlapon ugyancsak ezt tesszük.

³¹<http://www.dotfiles.com>

³²<http://ess.r-project.org/downloads/ess/>

³³<http://ess.r-project.org/>



26. ábra. Környezeti változó beállítása I.

- A `.emacs` vagy `_emacs` állományunkba beillesztjük a `(load "C:/emacs/ess-5.2.10/lisp/ess-site")` sort.

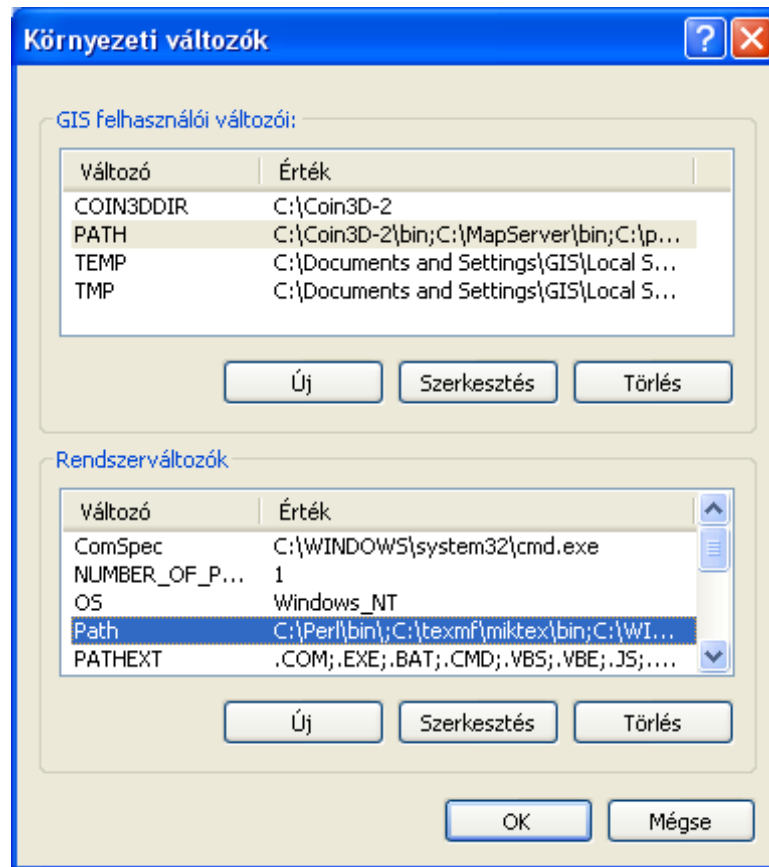
Ezzel az *Emacs-Ess-R* integráció készen áll a munkára. Ha most elindítjuk az *Emacs*-unkat, és lenyomjuk az ALT-x gombkombinációt, akkor megjelenik az alsó képernyő sorban az M-x karaktorsor és a kurzor villog. Ha most beírjuk a kis vagy nagy R-betűt, a sor tartalma a következőre változik:

```
ESS [S(R): Rterm] starting data directory? c:/emacs/bin/
```

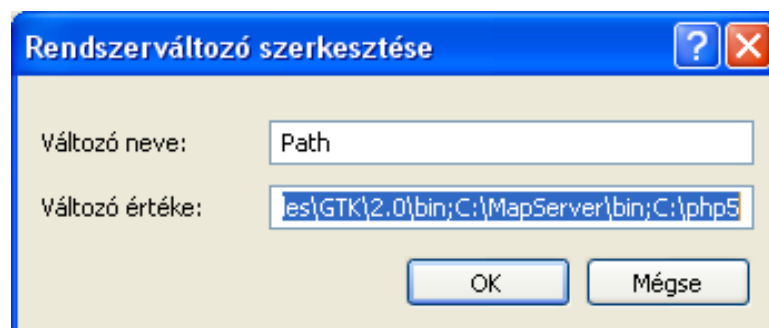
Ha ezt jóváhagyjuk ENTER-rel, akkor a 29. ábrán lévő képernyőn látható felületen kezdődhet meg a munka.

Kate

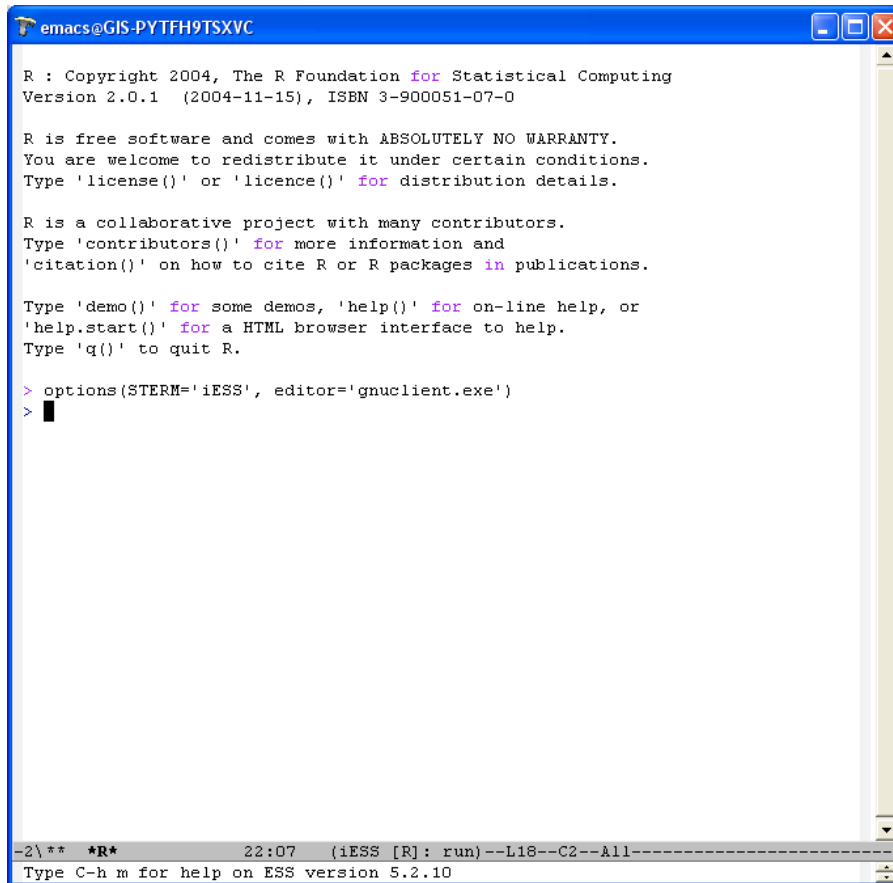
Linuxon egyszerűen használható szövegszerkesztő a *Kate*, ami lehetővé teszi az egyszerre több állománnyal való munkát. Az R-nyelvnek megfelelő szintaxis kiemelésre képes. Minden egyéb beállítás nélkül használhatjuk a Kate alsó ablakában látható felületet, mint R-terminált. Ha a rendszerünkön telepítve van az R-környezet és a Kate is, akkor a Kate elindítása után megjelenő felület terminál ablakába elegendő beírni az R utasítást és megnyomni az ENTER-t, aminek következtében máris van a szövegszerkesztőn belül egy R-környezetünk (30. ábra).



27. ábra. Környezeti változó beállítása II.



28. ábra. Környezeti változó beállítása III.



```

emacs@GIS-PYTFH9TSXVC

R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1 (2004-11-15), 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.

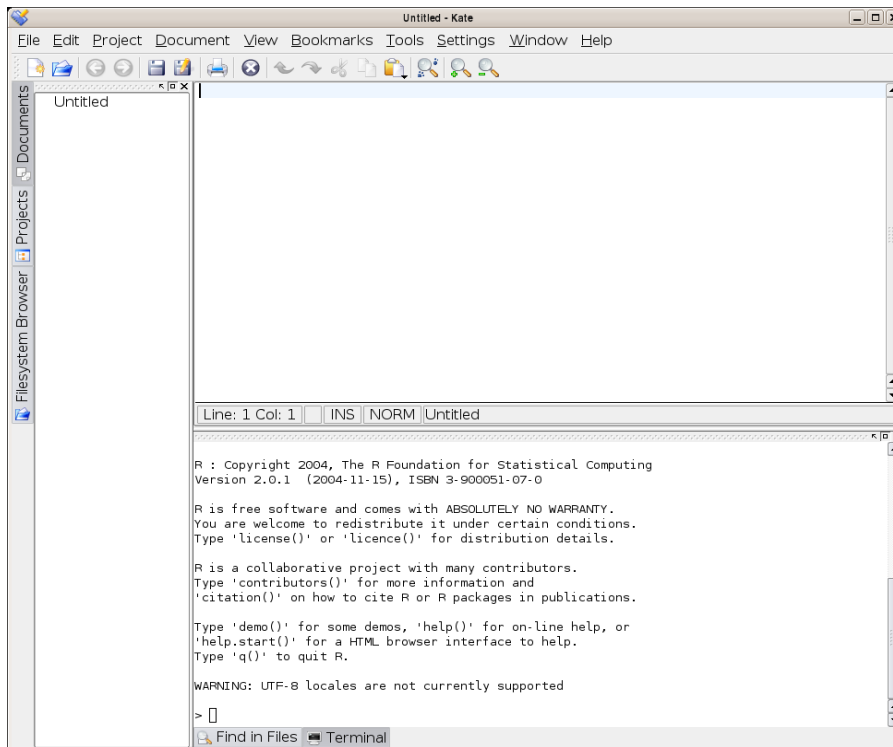
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

> options(STERM='iESS', editor='gnuclient.exe')
> █

```

-2** *R* 22:07 (iESS [R]: run) --L18--C2--All-----
Type C-h m for help on ESS version 5.2.10

29. ábra. R az Emacs-ben



```

Untitled - Kate
File Edit Project Document View Bookmarks Tools Settings Window Help

Untitled

Line: 1 Col: 1 | INS | NORM | Untitled

R : Copyright 2004, The R Foundation for Statistical Computing
Version 2.0.1 (2004-11-15), 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 a HTML browser interface to help.
Type 'q()' to quit R.

WARNING: UTF-8 locales are not currently supported

> █

```

Find in Files Terminal

30. ábra. Kate

Grafikus felületek

Az R-értelmezővel alapértelmezett kommunikációt parancsoron keresztül folytathatunk. Habár ez az interfész nagyon rugalmas, sokaknak (ma már) teljesen idegen. Egyes csoportok több grafikus felhasználói felületet (GUI)³⁴ is létrehozta, ezek egy része megvásárolható, más része ingyenesen letölthető. Az alábbiakban röviden ismertetek néhány ingyenesen elérhető GUI-t.

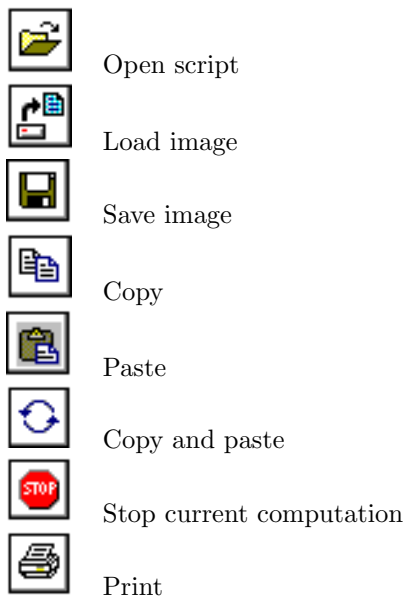
A grafikus felületeken belül az ablakok két formában jelenhetnek meg: *többdokumentumos* (MDI), illetve *egy-dokumentumos* (SDI) ablakrendszerekről beszélhetünk. Az előző esetében az ablakok egy „szülőablakon” belül helyezkednek el és az egyes „leányablakok” menürendszere a „szülőablakon” érhető el, így az elérhető menürendszer aszerint változik, hogy melyik „gyermekablak” aktív.

Windows RGui

A Microsoft Windowsra készített R-környezet telepítése után az alapértelmezett R-konzol a Windows RGui. Az indítóikkal való szoftverindítás után ennek a GUI-nak az MDI-stílusú felülete jelenik meg. A környezetben kezelhető ablakok a *konzol*, az *R-editor*, a *grafikai ablak*, illetve a *Data Editor*. A 2.1.1. verzió menürendszerének rövid bemutatása olvasható az alábbiakban.

Konzol

A Windows-os RGui konzol-eszköztára



menürendszer

File

<i>Source R code...</i>	Egy korábban megszerkesztett kódot nyit meg és egyben értelmezi is. (Ugyanezt az eredményt a <code>source()</code> utasítással érhetjük el.)
<i>New script</i>	A kódok szerkesztésére szolgáló R-editort nyitja meg, új kód szerkesztésére.
<i>Open script...</i>	Az <i>R-editorban</i> megnyit egy korábban szerkesztett kód-fájlt.
<i>Display file(s)...</i>	Megtekintésre megnyitja a kiválasztott ASCII-fájlt.

³⁴<http://www.r-project.org/GUI>

<i>Load Workspace...</i>	Egy korábbi <i>munkaterület</i> mentett „képét” tölthetjük be, ami tartalmazza az összes objektumot. Utasítása: <code>load()</code> .
<i>Save Workspace...</i>	A <i>munkaterület</i> -ben tárolt objektumokat kiírja egy <i>.RData</i> kiterjesztésű állományba. Utasítása: <code>save.image()</code> .

<i>Load History...</i>	Egy korábbi <i>munkaterület</i> utasításait tölti be. Utasítása a <code>loadhistory()</code> .
<i>Save History...</i>	A <i>munkaterület</i> utasításait menti el egy fájlba, aminek az ajánlott kiterjesztése: <i>.Rhistory</i> . Utasítása: <code>savehistory()</code> .

<i>Change dir...</i>	A munkakönyvtár meghatározását végezhetjük el segítségével (33. ábra). Ennek megfelelő utasítás a <code>setwd()</code> .
----------------------	--

<i>Print...</i>	A <i>konzol tartalmát</i> nyomtatja ki.
<i>Save to File...</i>	A <i>konzol tartalmát</i> egy ASCII fájlba menti ki.

<i>Exit</i>	Kilép az RGui-ből.
-------------	--------------------

Edit

<i>Copy</i>	A konzolban kijelölt szöveget a vágólapra másolja. (CTRL+C)
<i>Paste</i>	A vágólap tartalmát a konzolba illeszti. (CTRL+V)
<i>Paste commands only</i>	A konzol vágólapra helyezett részeiből csak az utasításokat illeszti be a prompthoz.
<i>Copy and Paste</i>	A kijelölt szöveget egy lépésben a vágólapra másolja és onnan beilleszti a konzol promptjába.
<i>Select all</i>	A konzol teljes tartalmát kijelöli. (a CTRL+A nem működik)
<i>Clear console</i>	A <i>konzol tartalmát</i> törli.

<i>Data editor...</i>	A <i>munkaterület</i> -ben tárolt dataframe vagy mátrix nevét megadva (34. ábra) megnyithatjuk azt az adatszerkesztő űrlapon (35. ábra). Utasítása: <code>fix()</code> .
-----------------------	--

<i>GUI preferences...</i>	Az RGui megjelenését megváltoztathatjuk, az erre szolgáló űrlapot (31. ábra) hívja meg e menüpont.
---------------------------	--

Misc

<i>Stop current computation</i>	Az éppen futó folyamatot leállítja.
---------------------------------	-------------------------------------

<i>Buffered output</i>	
------------------------	--

<i>List objects</i>	A <i>munkaterület</i> -ben tárolt objektumok nevét jeleníti meg a konzolban. Utasítása: <code>ls()</code> .
<i>Remove all objects</i>	A <i>munkaterület</i> -ben tárolt összes objektumot törli. Utasítása: <code>rm(list=ls(all=TRUE))</code> .
<i>List search path</i>	Megjeleníti a keresési utakat a konzolban. Ezek a már betöltött könyvtárakat reprezentálják, vagyis azokat a helyeket, ahol a kiadott utasításnak megfelelő függvényt kereshet az R-értelmező. Utasítása: <code>search()</code> .

Packages

<i>Load package...</i>	A megjelenő űrlap segítségével (37. ábra) a telepített csomagok (könyvtárak) közül kiválaszthatunk egyet, és be is tölthetjük.
------------------------	--

<i>Set CRAN mirror...</i>	A megjelenő űrlap (38. ábra) segítségével beállíthatunk azt a CRAN tükörodalt, amit csomagok telepítéséhez vagy frissítéséhez kívánunk használni.
---------------------------	---

<i>Select repositories...</i>	A megjelenő űrlap (39. ábra) segítségével beállíthatjuk a munkafolyamat során használni kívánt internetes tárhelyeket.
<i>Install package(s)...</i>	Ha korábban nem állítottunk be a 38. ábrán látható űrlapon CRAN tükörodalt, akkor először ezt kell megtennünk a megjelenő felületen. A megjelenő űrlap (40. ábra) segítségével az aktuális tárhelyen elérhető csomagokat telepíthetjük.
<i>Update packages...</i>	A már telepített csomagok frissíthetők az aktuális tárhelyről. Utasítása: <code>update.packages()</code> .
<hr/>	
<i>Install package(s) from local zip files...</i>	Helyi <code>.zip</code> fájlból telepít csomagot.

Windows

<i>Cascade</i>	Az ablakokat lépcsőzetesen rendezi.
<i>Tile</i>	Az ablakokat mozaikszerűen rendezi.
<i>Arrange Icons</i>	A minimalizált ablakokat egymás mellé rendezi.
<hr/>	
<i>✓ 1 R Console</i>	

Help

<i>Console</i>	A konzolon alkalmazható billentyű-kombinációk leírását mutatja meg.
<hr/>	
<i>FAQ on R</i>	Az R-rel kapcsolatos gyakori kérdések és válaszok.
<i>FAQ on R for Windows</i>	A Windows-on futó R-el kapcsolatos gyakori kérdések és válaszok.
<i>Manuals (in PDF)</i>	Kézikönyvek az R használatával kapcsolatban.
	<i>An Introduction on R</i>
	<i>R Reference Manual</i>
	<i>R Data Import/Export</i>
	<i>R Language Definition</i>
	<i>Writing R Extensions</i>
	<i>R Installation and Administration</i>
<hr/>	
<i>R functions (text)...</i>	Függvény keresése a teljes név alapján (42. ábra). Utasítása: <code>help()</code> .
<i>Html help</i>	HTML sűgó megjelenítése. Utasítása: <code>help.start()</code> .
<i>Search help...</i>	Szöveg keresése a címekben, nevekben, leírásokban (43. ábra). Utasítása: <code>help.search()</code> .
<i>search.r-project.org...</i>	A 44. ábrán látható űrlap beviteli mezőjébe gépelt szöveget kereshetjük a levelezési listák és egyéb dokumentációk szövegeiben.
<hr/>	
<i>Apropos...</i>	Apropos keresése a függvények nevében (41. ábra). Utasítása: <code>apropos()</code> .
<hr/>	
<i>R Project home page</i>	Az R-projekt honlapját nyitja meg. (http://www.r-project.org/)
<i>CRAN home page</i>	A CRAN honlapját nyitja meg. (http://cran.r-project.org/)
<hr/>	
<i>About</i>	Névjegy.

Popup menü

<i>Copy</i>	<i>CTRL+C</i>	A konzolban kijelölt szöveget a vágólapra másolja.
<i>Paste</i>	<i>CTRL+V</i>	A vágólap tartalmát a konzolba illeszti.
<i>Paste commands only</i>		A konzol vágólapra helyezett részeiből csak az utasításokat illeszti be a prompthoz.
<i>Copy and Paste</i>	<i>CTRL+X</i>	A kijelölt szöveget egy lépésben a vágólapra másolja és onnan beilleszti a konzol promptjába.
<hr/>		
<i>Clear window</i>	<i>CTRL+L</i>	Törli a konzol tartalmát.

Select all

A konzol teljes tartalmát tartalmát kijelöli.

*Buffered output
Stay on top**CTRL+W*

R editor

Az R-editor eszköztára



Open script



Save script



Run line or selection



Return focus to Console



Print

menürendszer

File

*New script**CTRL+N*

A kódok szerkesztésére szolgáló R editort nyitja meg, új kód szerkesztésére.

*Open script...**CTRL+O*Az *R-editorban* megnyit egy korábban szerkesztett kód-fájlt.*Save**CTRL+S*Az *R-editor* tartalmát menti ASCII fájlként.*Save as...*Az *R-editor* tartalmát „menti másként”.*Print...*Az *R-editor* tartalmát kinyomtatja.*Close script*Bezárja az *R-editor*t.*Exit*

Kilép az RGui-ból.

Edit

*Undo**CTRL+Z*

Visszavonás.

*Cut**CTRL+X*

Kivágja és a vágólapra helyezi a kijelölt szöveget.

*Copy**CTRL+C*

A vágólapra helyezi a kijelölt szöveget.

*Paste**CTRL+V*A vágólapra helyezett tartalmat beilleszti az *R editorba*.*Delete*

Törli a kijelölt szöveget.

*Select all**CTRL+A*Az *R-editor* teljes tartalmát kijelöli.*Clear console**CTRL+L*Az *R-editor* teljes tartalmát törli.*Run line or selection**CTRL+R*

Ha nincs kijelölve kódrész, akkor az aktuális sort, ha ki van jelölve szöveg, akkor azt illeszti a konzolba, ami értelmezi azt.

*Run all*Az *R editor* teljes tartalmát bemásolja a konzolba, ami azt lefuttatja.*Find...**CTRL+F*Szöveg keresése az *R-editorban*.*Replace...**CTRL+H*Szöveg cseréje az *R-editorban*.*GUI preferences...*

Az RGui megjelenését megváltoztathatjuk, az erre szolgáló űrlapot (31. ábra) hívja meg a menüpont.

Packages

<i>Load package...</i>	A megjelenő űrlap segítségével (37. ábra) a telepített csomagok (könyvtárak) közül kiválaszthatunk egyet és be is tölthetjük.
<i>Set CRAN mirror...</i>	A megjelenő űrlap (38. ábra) segítségével beállíthatunk egy CRAN-tüköroldalt, amit csomagok telepítéséhez vagy frissítéséhez kívánunk használni.
<i>Select repositories...</i>	A megjelenő űrlap (39. ábra) segítségével beállíthatjuk a munkafolyamat során használni kívánt internetes tárhelyeket.
<i>Install package(s)...</i>	Ha korábban nem állítottunk be a 38. ábrán látható űrlapon CRAN-tüköroldalt, akkor először ezt kell megtennünk a megjelenő felületen. A megjelenő űrlap (40. ábra) segítségével az aktuális tárhelyen elérhető csomagokat telepíthetünk.
<i>Update packages...</i>	A már telepített csomagok frissíthetők az aktuális tárhelyről. Utasítása az <code>update.packages()</code> .
<i>Install package(s) from local zip files...</i>	Helyi <code>.zip</code> fájlból telepít csomagot.

Windows

<i>Cascade</i>	Az ablakokat lépcsőzetesen rendezi.
<i>Tile</i>	Az ablakokat mozaikszerűen rendezi.
<i>Arrange Icons</i>	A minimalizált ablakokat egymás mellé rendezi.

1 R Console
✓ 2 Untitled - R Editor

Help

<i>Console</i>	A konzolon alkalmazható billentyű-kombinációk leírását mutatja meg.
<i>FAQ on R</i>	Az R-rel kapcsolatos gyakori kérdések és válaszok.
<i>FAQ on R for Windows</i>	A Windows-on futó R-el kapcsolatos gyakori kérdések és válaszok.
<i>Manuals (in PDF)</i>	Kézikönyvek az R használatával kapcsolatban. <i>An Introduction on R</i> <i>R Reference Manual</i> <i>R Data Import/Export</i> <i>R Language Definition</i> <i>Writing R Extensions</i> <i>R Installation and Administration</i>
<i>R functions (text)...</i>	Függvény keresése a teljes név alapján (42. ábra). Utasítása: <code>help()</code> .
<i>Html help</i>	HTML sűgó megjelenítése. Utasítása: <code>help.start()</code> .
<i>Search help...</i>	Szöveg keresése a címekben, nevekben, leírásokban (43. ábra). Utasítása: <code>help.search()</code> .
<i>search.r-project.org...</i>	A 44. ábrán látható űrlap beviteli mezőjébe gépelt szöveget kereshetjük a levelezési listák és egyéb dokumentációk szövegeiben.
<i>Apropos...</i>	Apropos keresése a függvények nevében (41. ábra). Utasítása: <code>apropos()</code> .
<i>R Project home page</i>	Az R-projekt honlapját nyitja meg. (http://www.r-project.org/)
<i>CRAN home page</i>	A CRAN honlapját nyitja meg. (http://cran.r-project.org/)
<i>About</i>	Névjegy.

Popup menü

<i>Run line or selection</i>	<i>Ctrl+R</i>	Ha nincs kijelölve kódrész, akkor az aktuális sort, ha ki van jelölve szöveg, akkor azt illeszti a konzolba, ami értelmezi azt.
<hr/>		
<i>Undo</i>	<i>Ctrl+Z</i>	Visszavonás.
<hr/>		
<i>Cut</i>	<i>Ctrl+X</i>	Kivágja és a vágólapra helyezi a kijelölt szöveget.
<i>Copy</i>	<i>Ctrl+C</i>	A vágólapra helyezi a kijelölt szöveget.
<i>Paste</i>	<i>Ctrl+V</i>	A vágólapra helyezett tartalmat beilleszti az <i>R-editorba</i>
<i>Delete</i>		Törli a kijelölt szöveget.
<i>Select all</i>	<i>Ctrl+A</i>	Az <i>R-editor</i> teljes tartalmát kijelöli.

Grafikai ablak

A grafikai ablak eszköztára



Copy to the clipboard as a metafile



Print



Return focus to console

menürendszer*File*

<i>Save as</i>		
<i>Metafile...</i>		Mentés metafájlként.
<i>Postscript...</i>		Mentés postscriptként.
<i>PDF...</i>		Mentés PDF-ként.
<i>PNG...</i>		Mentés PNG-ként.
<i>BMP...</i>		Mentés BMP-ként.
<i>Jpeg</i>		
<i>50% quality...</i>		Mentés 50%-os minőségű JPEG-ként.
<i>75% quality...</i>		Mentés 75%-os minőségű JPEG-ként.
<i>100% quality...</i>		Mentés 100%-os minőségű JPEG-ként.
<i>Copy to the clipboard</i>		
<i>as a Bitmap</i>	<i>CTRL+C</i>	Másolás bitmapként a vágólapra.
<i>as a Metafile</i>	<i>CTRL+W</i>	Másolás metafájlként a vágólapra.
<hr/>		
<i>Print...</i>	<i>CTRL+P</i>	A grafika nyomtatása.
<hr/>		
close Device		A grafikai ablak bezárása.

History

<i>Recording</i>		Ha a ✓-jellel megjelöltük, akkor az R-környezet automatikusan rögzíti a grafikus ablak tartalmát a „történet”-be.
<hr/>		
<i>Add</i>	<i>INS</i>	A grafikai „történet”-hez hozzáadjuk a grafikát.
<i>Replace</i>		
<hr/>		
<i>Previous</i>	<i>PgUp</i>	A grafikai „történet”-ben az előző képet tölti be a grafikai ablakba.
<i>Next</i>	<i>PgDown</i>	A grafikai „történet”-ben a következő képet tölti be a grafikai ablakba.
<hr/>		
<i>Save to variable...</i>		
<i>Get from variable...</i>		

Clear history

A grafikai ablak „történetének” törlése.

*Resize**✓ R mode**Fit to window**Fixed size**Windows**Cascade*

Az ablakokat lépcsőzetesen rendezi.

Tile

Az ablakokat mozaikszerűen rendezi.

Arrange Icons

A minimalizált ablakokat egymás mellé rendezi.

*1 R Console**✓ 2 2 R Graphics: Device 2 (ACTIVE)***Popup menü***Arrange Icons*

A minimalizált ablakokat egymás mellé rendezi.

Copy as metafile

Másolás metafájlként a vágólagra.

Copy as bitmap

Másolás bitmapként a vágólagra.

Save as metafile. . .

Mentés metafájlként.

Save as postscript. . .

Mentés postscriptként.

Stay on top

Print. . .

A grafika nyomtatása.

Adatszerkesztő**menürendszer***File**Close*

Az adatszerkesztő bezárása.

*Windows**Close*

Az adatszerkesztő bezárása.

Cascade

Az ablakokat lépcsőzetesen rendezi.

Tile

Az ablakokat mozaikszerűen rendezi.

Arrange Icons

A minimalizált ablakokat egymás mellé rendezi.

*1 R Console**✓ 2 Data Editor**Edit**Copy**Ctrl+C*

A kijelölt cellák tartalmát a vágólagra helyezi.

*Paste**Ctrl+V*

A vágólapon lévő adatokat, a kijelölt cellákba illeszti.

*Delete**DEL*

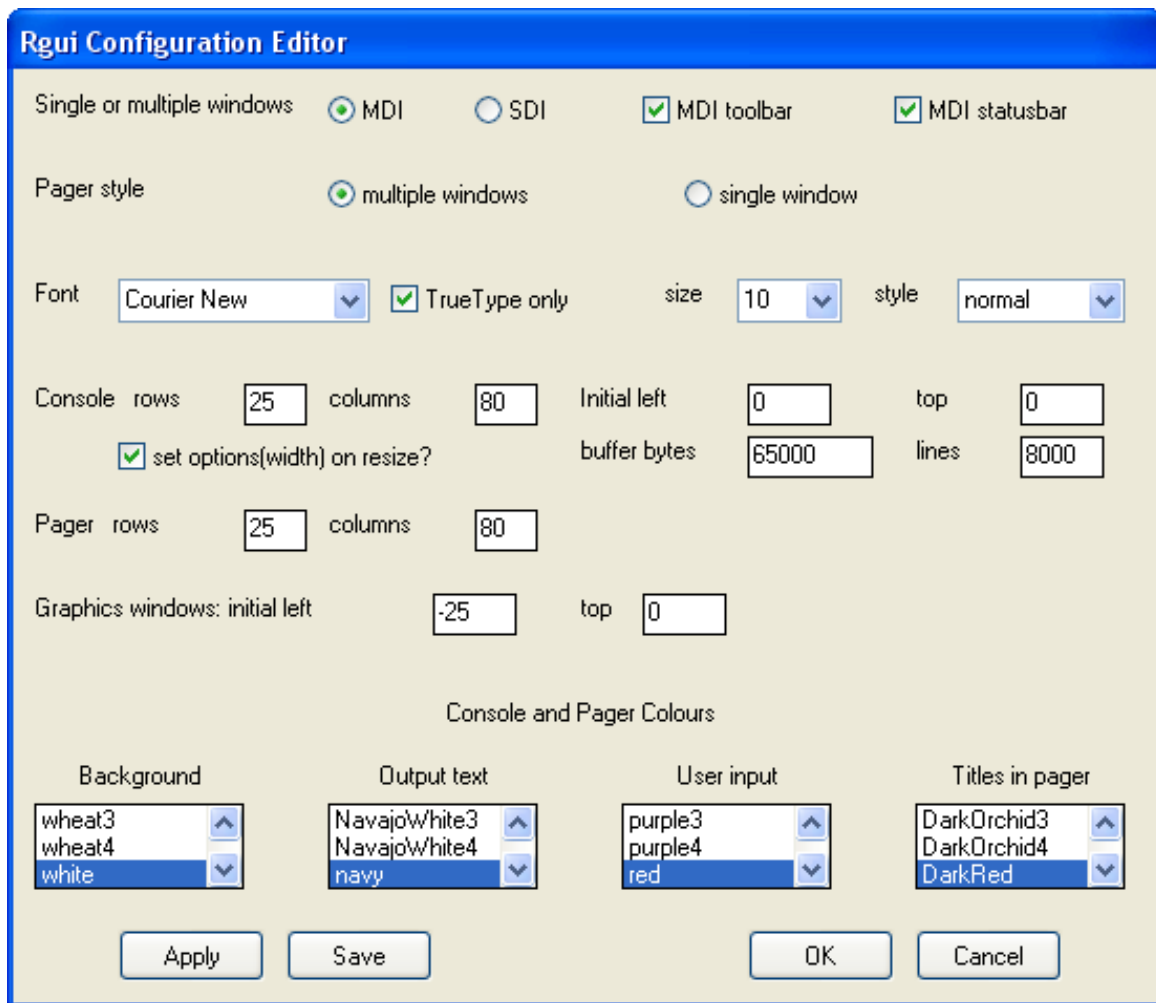
A kijelölt cella tartalmát törli.

Cell widths. . .

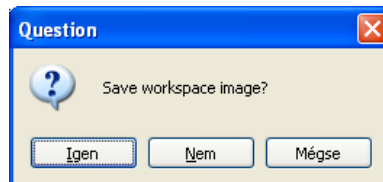
A cellák szélességét állíthatjuk be (36. ábra).

*Help**Data editor*

Az adatszerkesztő használatával kapcsolatos információkat jelenít meg.



31. ábra. Windows-os RGui beállításait módosító felület



32. ábra. A Windows RGui-ból való kilépéskor jelenik meg a munkakörnyezet mentésére kérdező párbeszédablak

Popup menü

Help

Copy selected cell

Paste to selected cell

Autosize column

Stay on top

Close

Az adatszerkesztő használatával kapcsolatos információkat jelenít meg.

A kijelölt cella tartalmát a vágólapra másolja.

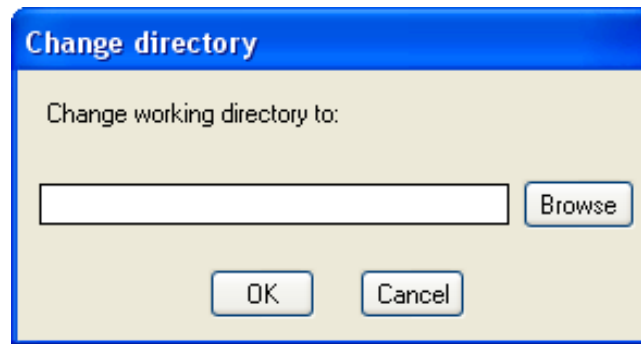
A vágólap tartalmát a kijelölt cellába másolja.

Az oszlopok szélességének automatikus méretezése.

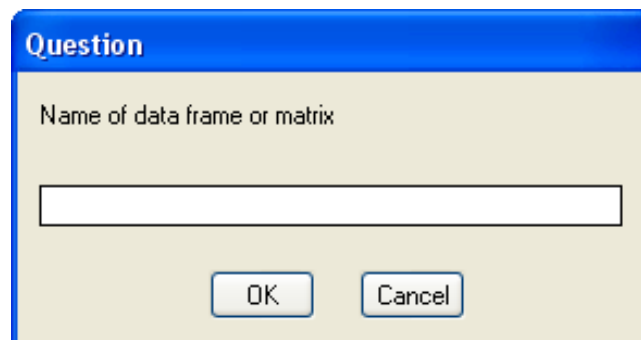
Adatszerkesztő bezárása.

R Commander

Azok részére, akik a grafikus felületű statisztikai szoftverekhez szoktak, többen is fejlesztettek GUI-kat (Graphical User Interface). Az R commander ezek közül egy ingyenes megoldás, ami részben hasonlít az S-Plus felhasználói felületéhez. Tulajdonképpen a `Rcmdr` is egy csomag, ami letölthető és/vagy telepíthető a CRAN-ról. Ahhoz, hogy hibátlanul fusson a `Rcmdr`, telepítenünk kell még más csomagokat is, ezek a következők: `Hmisc`, `quadprog`,



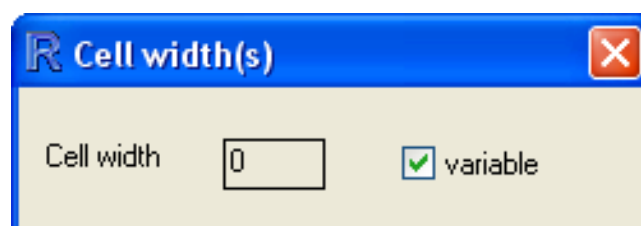
33. ábra. A munkakönyvtár beállítását segítő űrlap



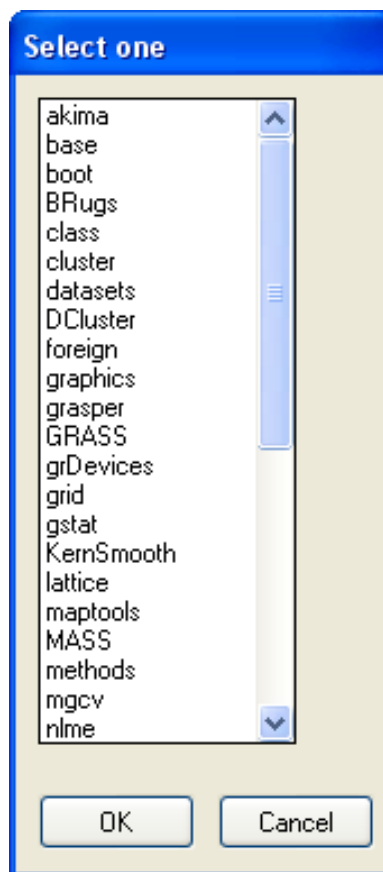
34. ábra. Adatszerkesztőbe beolvasandó adatállomány megadását segítő űrlap

	day	time	temp	activ	var5
1	346	840	36.33	0	
2	346	850	36.34	0	
3	346	900	36.35	0	
4	346	910	36.42	0	
5	346	920	36.55	0	
6	346	930	36.69	0	
7	346	940	36.71	0	
8	346	950	36.75	0	

35. ábra. Adatszerkesztő űrlap



36. ábra. A cellák szélességének beállítását segítő űrlap



37. ábra. Csomagbetöltést segítő űrlap

oz, leaps, chron, fBasis, its, tseries, quantreg, DAAG, abind, car, effects, lmtest, multcomp, mvtnorm, relimp, rgl, sandwich, strucchange, zoo. A `library(Rcmdr)` paranccsal tölthetjük be az R commandert és a hozzá szükséges csomagokat.

A 45. ábrán lévő felületen látható két szöveges terület. A felső *kódszerkesztő* területen belül tudjuk megszerkeszteni a kódunkat, amiből a kijelölt kódrészeket a **Submit** gomb megnyomásával küldjük el az értelmezőnek. Sajnos hiányzik belőle az R-nyelv szintaxis-kiemelése. A lefutott kódok és azok eredményei az alsó *eredmény-szerkesztőben* jelennek meg. A GUI felső szegélyén egy menürendszer található, amiből több statisztikai, grafikai és kiegészítő eljárásához érhetünk el egyszerű módon párbeszédablakokat (46. ábra).

SciViews-R GUI

A *SciViews-R* olyan alkalmazásgyűjtemény, amely grafikus felhasználói felületet (GUI) biztosít az R-környezethez. A gyűjteményben a *SciViews R Console*, a *SciViews R Report* és a *Tinn-R* alkalmazások találhatók. Jelenleg csak Windowson használható.

Telepítés

Az ingyenesen letölthető³⁵ telepítőállomány futtatása során egy „varázsló” vezeti a felhasználót a folyamaton keresztül. Az aktuálisan legfrisebb verzió telepítésekor figyelemmel kell lenni arra, hogy milyen R-verziókkal tud együttműködni a *SciViews-R*. Az itt bemutatott példákban a 0.8–8 verziót használtam, ami az R-környezet 2.1.X verzióival tud együttműködni. Továbbá az R-környezettel való együttműködéshez az R-telepítésnek a `C:\Program Files\R\rw2011` könyvtárban kell lennie. A *SciViews-R* alapértelmezett telepítési könyvtára pedig a `C:\Program Files\SciViews`, amit jobb így hagyni. A *SciViews-R* telepítési folyamat befejezése után a telepítési könyvtárban, a `bin` alkönyvtárban belül található a `RConsole.exe`, a `RReport.exe` és a `Tinn-R.exe` alkalmazás. Ha elindítjuk a `RConsole.exe` futtatható állományt, akkor megjelenik egy SDI-stílusú RGui és a 47. ábrán látható párbeszédablak. Arra kérdez rá, hogy telepíteni kívánjuk-e a *SciViews* csomagot. Ennek telepítése nélkül nem fog működni a *SciViews-R*. Ha az *Igen* gombra kattintunk, akkor megjelenik a 48. ábrán látható lista, amiből kiválaszthatjuk azt a forrást, ahonnan a telepítendő csomagot be szeretnénk szerezni. Az

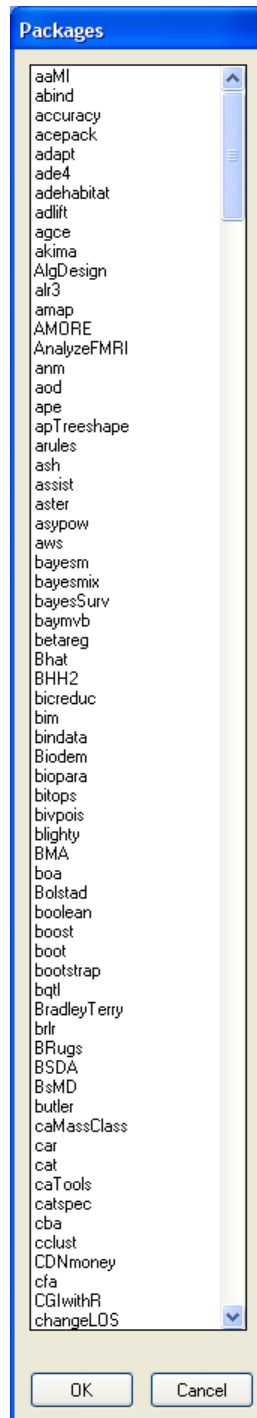
³⁵<http://www.sciviews.org/SciViews-R/>



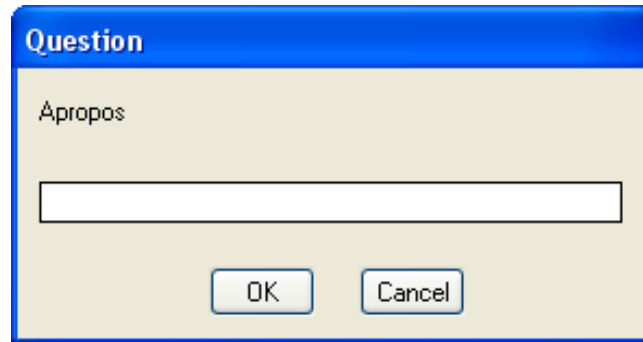
38. ábra. CRAN tüköroldal beállítását segítő űrlap



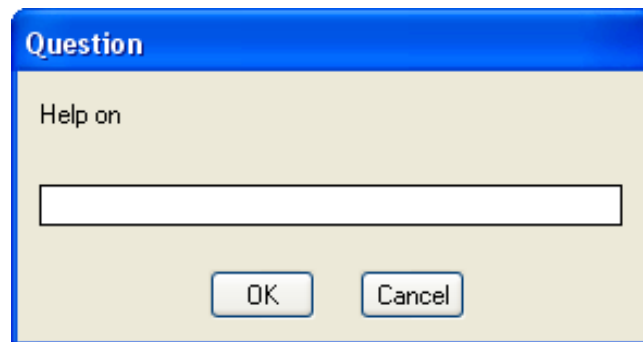
39. ábra. Select repositories...



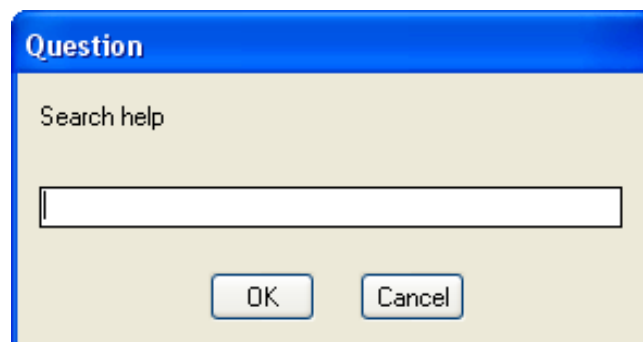
40. ábra. A CRAN-ról való csomagtelepítést segítő űrlap



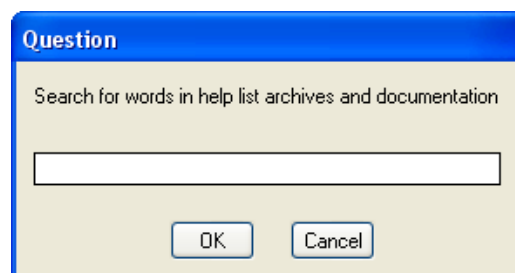
41. ábra. Apropos keresést segítő űrlap



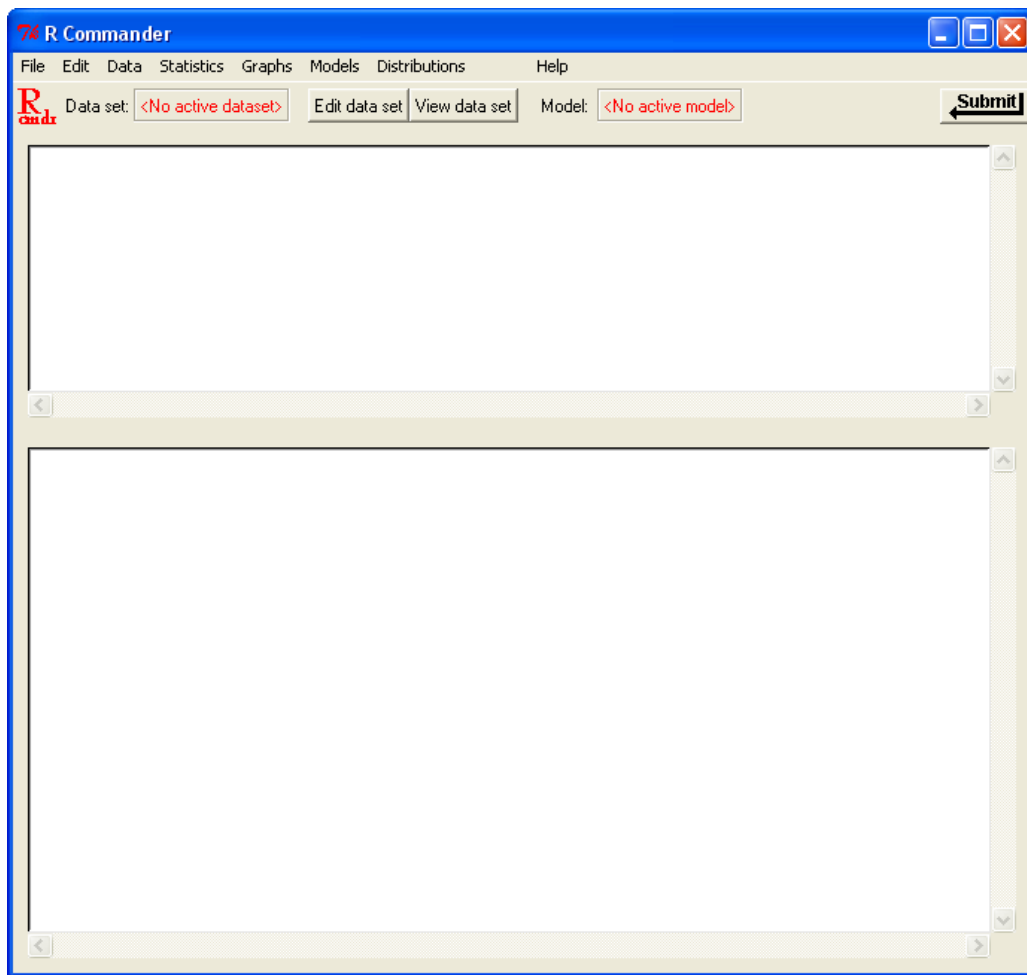
42. ábra. Függvény keresését segítő űrlap



43. ábra. Szöveg keresését segítő űrlap



44. ábra. search.r-project.org...



45. ábra. Az R Commander induló képernyője

OK gomb lenyomása után az adott forrásból telepíti a csomagot a rendszer. Természetesen a csomagtelepítést az R-környezetből is elvégezhetjük, a szokásos módokon.

A megfelelő működéshez szükséges, hogy telepítsük a R2HTML-csomagot is, ha az aktuális R-környezetünkben az ehhez a csomaghoz szükséges egyéb könyvtárak (`acepack`, `chron`) nincsenek még meg, akkor azokat is telepítenünk kell.

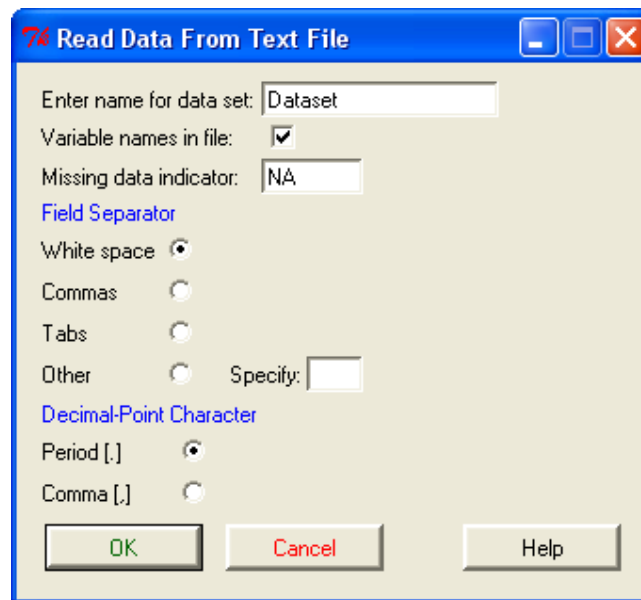
További telepítésre ajánlott csomagok: `Rcmdr`, `tcltk2`, `wxPython`, `RSPython`, `Hmisc`, valamint ezek működéséhez szükséges további csomagok.

SciViews R Console

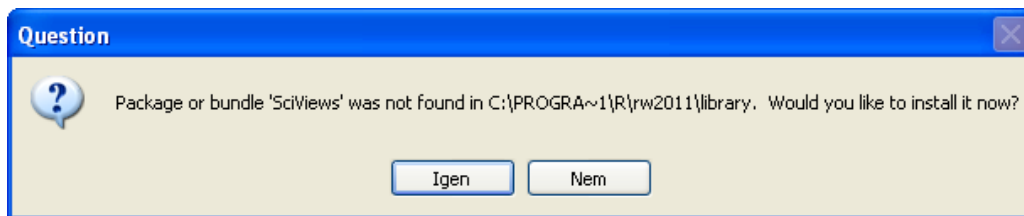
A *SciViews R Console* elindítása után rövid időre először a 49. ábrán látható konzol jelenik meg, ami a Windows RGui SDI-stílusú konzolja. Amint ezen a konzolon látható, az R betölti az összes szükséges könyvtárat, eltűnik az előbbi felület és megjelenik az *SciViews R Console* (50. ábra). A *SciViews R Console* elindítható úgy is, ha az *Windows RGui*-t SDI-módra állítjuk és egyszerűen a terminálban töltjük be az `svGUI` csomagot (`library(svGUI)`). Ha ezt MDI módban próbáljuk meg, akkor nem jelenik meg eredményként a *SciViews R Console*.

Az 50. ábrán látható *SciViews R konzolon* elérhető elemek:

- Menürendszer, mely nagyrészt megegyezik a Windows RGui konzol-menüjével.
- Eszköztár, ami szintén megegyezik a Windows RGui konzol-eszköztárával.
- Terminál, ami nem különbözik az egyéb környezetekben megszokottaktól. Popup-menüje megegyezik a Windows RGui-nál látottal.
- Kódszerkesztő. Ez az elem igazán nagy segítséget jelent rövidebb szkriptek szerkesztésében. Egyetlen hátránya, hogy egyszerre csak egy szkriptállományt tudunk kezelni vele.



46. ábra. Szöveges állományok beolvasását segítő ablak



47. ábra. SciViews-csomag telepítése I.

- Dokknak nevezzük a konzol jobb felső részén látható területet, amiben hasznos segédletek érhetők el, ezzel megkönnyítve a munkát.

Kódszerkesztő

A kódszerkesztő ismeri az R-nyelv szintaxisát, illetve az egyes telepített függvények argumentumlistáját. Ebből következik, hogy amikor beírjuk egy függvény nevét, és nyitunk egy zárójelet, akkor a függvény neve mellett megjelenik a függvény és a hozzá tartozó argumentumnevek. Alapértelmezésben a kódszerkesztő a *SciViews R konzolba* van ágyazva, viszont ha jobban szeretünk külön ablakokban dolgozni, akkor abból ki is emelhető. Ennek előnye, hogy az egyes ablakokban nagyobb felületet láthatunk egyszerre, hátránya, hogy az egyes ablakok fedhetik egymást. Az eszköztárból elérhető műveletek közül a következők működnek a jelenlegi verzióban (0.8.8):

New R Script CTRL+SHIFT+N

Új szkriptet hoz létre, egyben, ha kívánjuk, az aktuálist el is menti.

Open R Script... CTRL+SHIFT+O

Megnyit egy már korábban létrehozott szkriptet.

Save Script... CTRL+SHIFT+S

Menti az aktuális szkriptet.

Execute (F5)

A kijelölt kódrészt vagy az aktív sort elküldi az R-parancsértelmezőre, aminek következtében a terminálba íródik a lefuttatott kód és annak eredménye(i).

A kódszerkesztő popup menüje:

Cut

A kijelölt szövegrészt kivágja és a vágólapra helyezi.

Copy

A kijelölt szövegrészt a vágólapra helyezi.

Paste

A vágólapról a kurzor helyétől kezdődően beilleszti.

Find...

A szkripten belül lehet keresni karakter(sor)t.

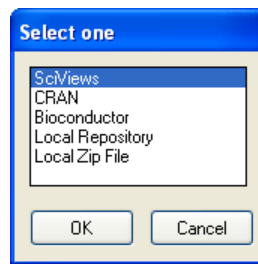
Replace...

Adott karakter(sor)t felcserélhetünk egy másikkal.

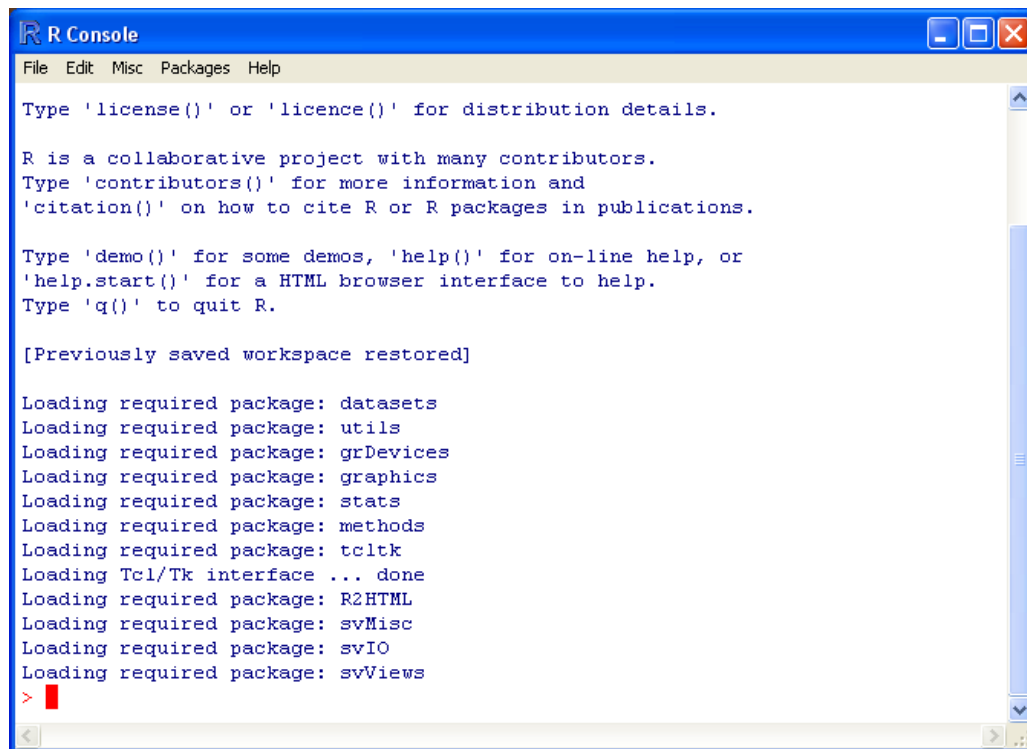
Show Whitespace

Nagybetűssé alakítja a kijelölt szakaszban lévő betűket.

Make Uppercase



48. ábra. SciViews-csomag telepítése II.



49. ábra. R konzol

Make Lowercase

Kisbetűssé alakítja a kijelölt szakaszban lévő betűket.

Undo

Művelet visszavonása.

Redo

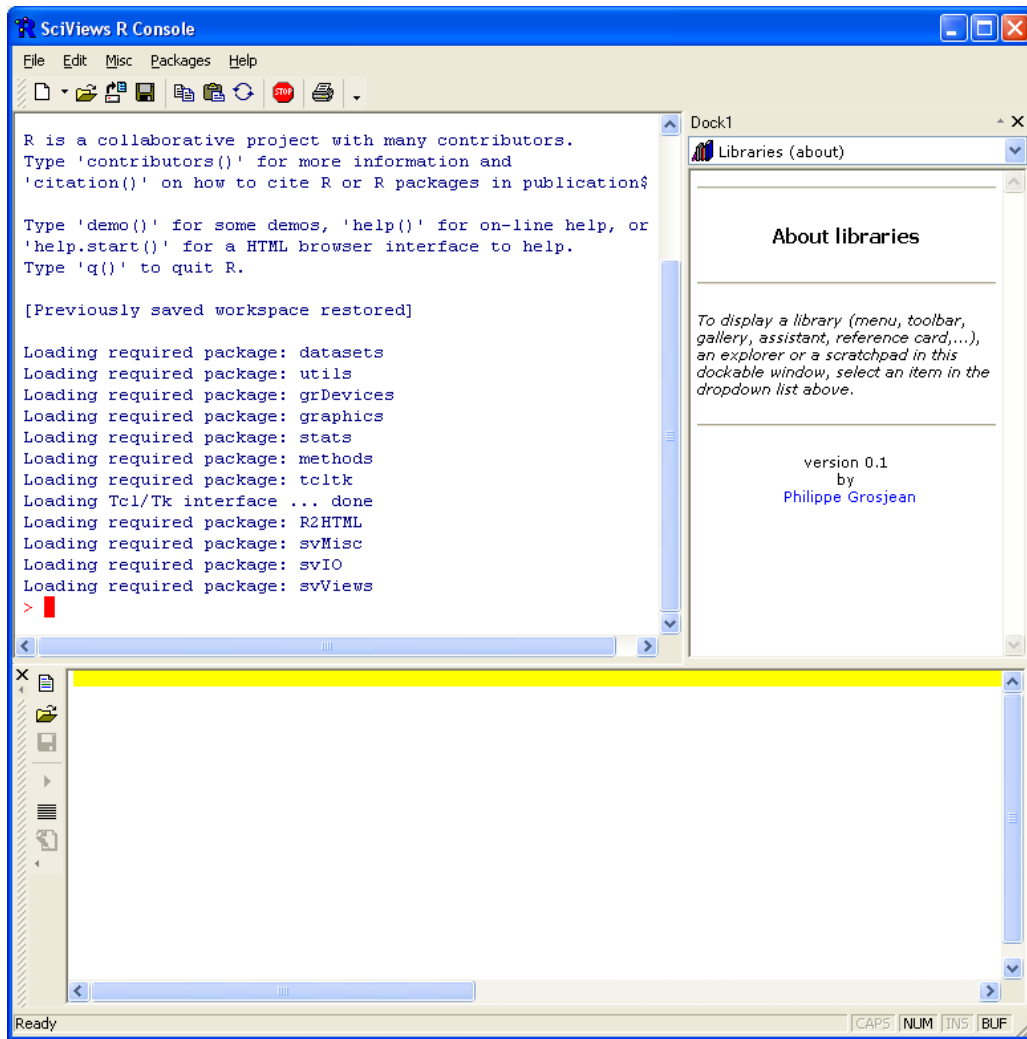
Visszavonás visszavonása.

Properties...

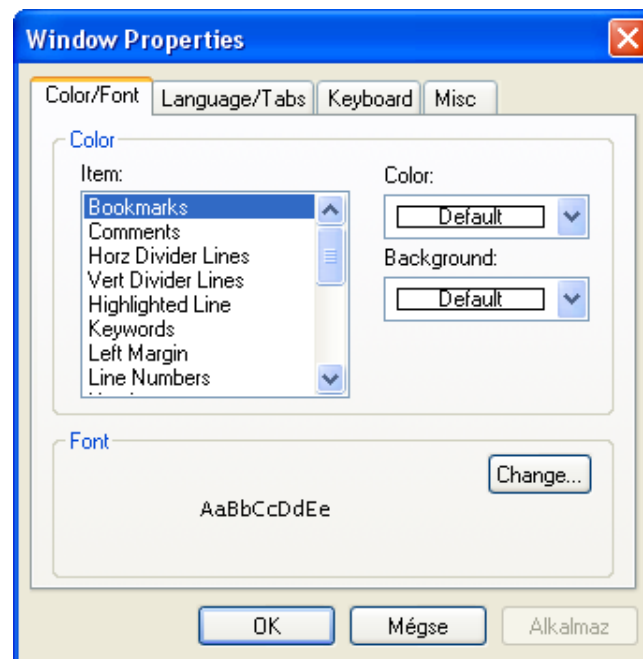
A szkriptszerkesztő tulajdonságait állíthatjuk be a megjelenő űrlap segítségével (51. ábra).

Dokk

A dokk(ok)ban sokféle gyors segédletet helyeztek el. Egyzerre öt dokkot nyithatunk meg, és mindegyikben különböző eszköztárakat tehetünk közvetlenül elérhetővé. A dokkok (a szkriptszerkesztőhöz hasonlóan) használhatók a konzolba ágyazva vagy külön ablakokban. Egyes eszköztárakból kiválaszthatunk elemeket (pl.: függvények), amelyek beíródnak vagy a *szkriptszerkesztőbe* vagy a *terminálba*. Hogy melyik részbe illesztődnek be az adott objektumok, attól függ, hogy a kurzor hol áll, a *szkriptszerkesztőn* vagy a *terminálon*. Az egyes eszköztárakat a dokk felső szegélyén található listából választhatjuk ki, név szerint:



50. ábra. SciViews R konzol

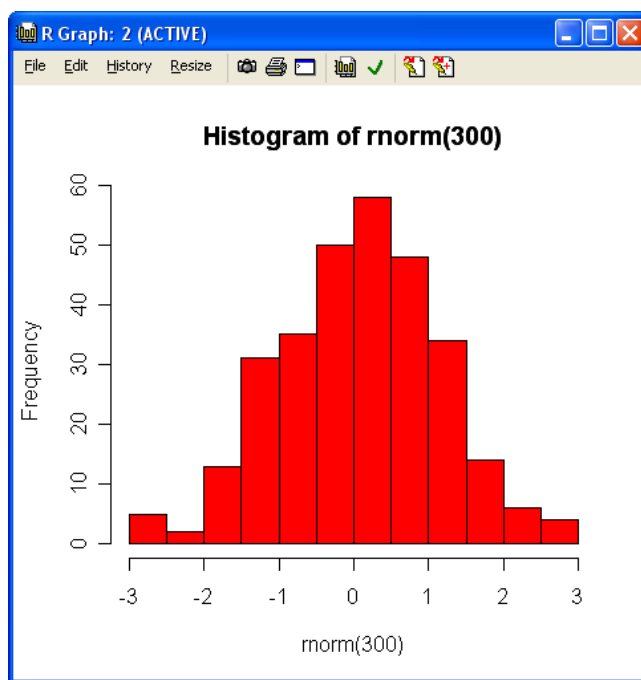


51. ábra. SciViews szkriptszerkesztő beállító felület

<i>Libraries (about)</i>	Névjegy (50. ábra).
<i>Default</i>	SciViews dokumentációk érhetők el a megnyíló felületről.
<i>R Commander menu</i>	Az R-kommander menüjéből elérhető grafikus felületeket hívhatunk meg ezen menüpontok segítségével, anélkül, hogy az R-kommander konzolát is megnyitnánk.
<i>R reference card</i>	Gyakrabban használt R-függvények téma szerint csoportosított gyűjteménye jelenik meg, ha ezt a listaelemet választjuk ki. Ha valamely függvény nevére kattintunk, akkor az aktív egységbe beilleszti a függvény nevét, illetve egyes függvényeknél argumentumokat is.
<i>Colors</i>	Egy színskálából grafikusán választhatunk színeket, ezek színskódként íródnak be az aktív területre.
<i>Web Links</i>	Az R nyelvvel és környezettel kapcsolatos keresési felület, valamint hasznos internetes kapcsolatok, illetve pdf-dokumentumok gyűjteménye.
<i>Documentation</i>	
<i>Objects explorer</i>	A munkafolyamatban létrejött, illetve a betöltött csomagok részét képező objektumok listájából választhatunk.
<i>Session explorer</i>	A munkafolyamatban létrehozott fájlok (pl.: jegyzetek) között lehet tallózni.
<i>Files explorer</i>	Fájlkezelőt nyit meg.
<i>Scratchpad (common)</i>	Közös jegyzetfüzetet nyit meg, aminek a tartalma egy RTF-fájlba lesz kiírva.
<i>User scratchpad</i>	A felhasználó által jegyzet hozható létre, ami RTF-állományba íródik ki.
<i>Session scratchpad</i>	A munkafolyamatban során jegyzet hozható létre, ami RTF-állományba íródik ki.

SciViews R grafikai ablak

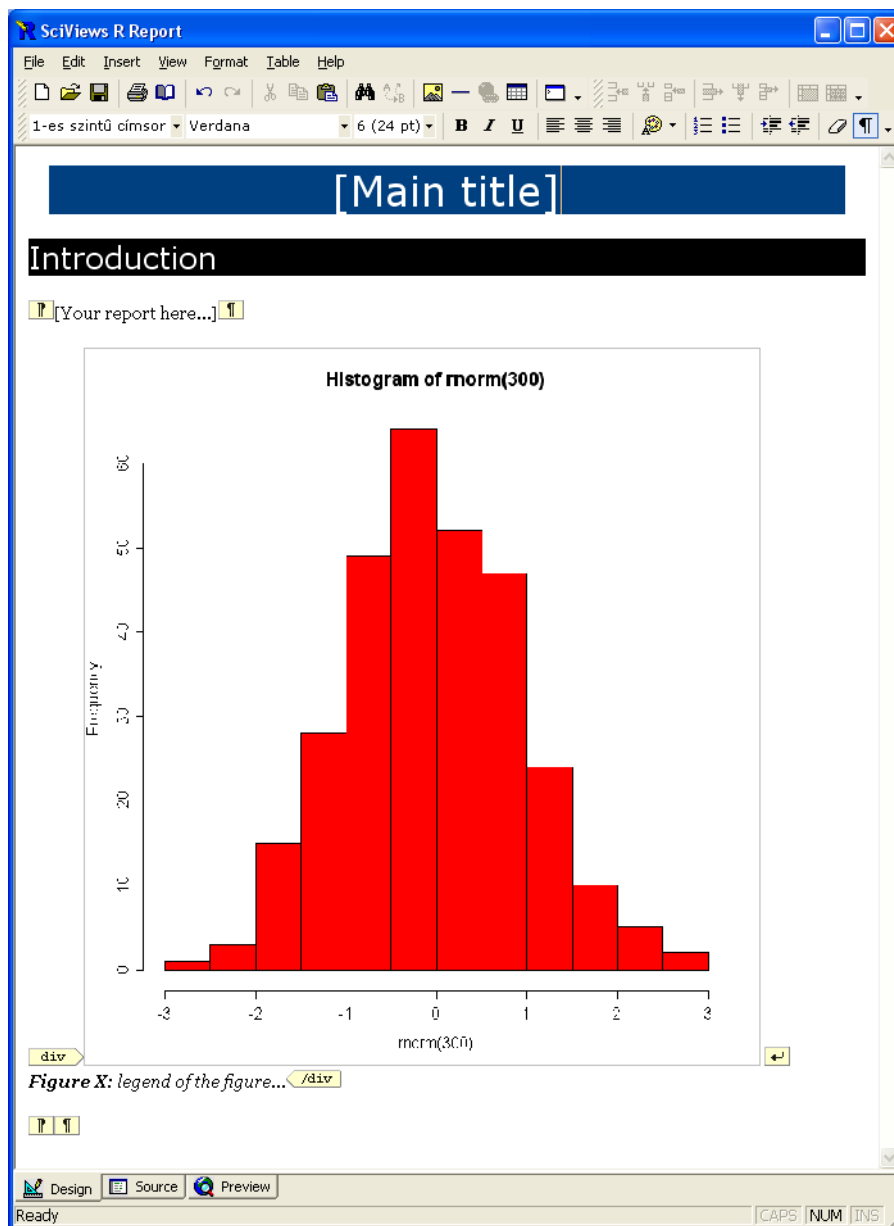
Ha a *SciViews R*-ben ábrát generálunk, akkor az a megszokott grafikai ablaktól némileg eltérő ablakban jelenik meg (52. ábra). A legfőbb különbség az, hogy a későbbiekben látható *SciViews R Report*-nak átadhatjuk a létrehozott ábrát. Erre szolgál az *Edit* menüben belüli *Report (png)* és *Report multiple formats* menüpontok, illetve az ezeknek megfelelő eszköztárban látható utolsó két ikon is. További lényeges különbség, hogy a grafikai ablakból nyithatunk újabb grafikai ablakot, illetve az egyes grafikai ablakok aktivitását az ablakokból irányíthatjuk. Új grafikus ablakot a *File* menü *New Graph CTRL+N* opciójával nyithatunk. Az adott grafikai ablakot pedig az *Edit* menü *Activate CTRL+A* menüpontjával aktivizálhatjuk, aminek természetesen az a következménye, hogy az addig aktív grafikai ablak inaktív lesz. Ugyanezt érthetjük el az eszköztár negyedik és ötödik gombjának lenyomásával is.



52. ábra. SciViews R grafikai ablak

SciViews R Report

A *SciViews R Report* egy HTML-szerkesztő, amibe közvetlenül tudunk beilleszteni a SciViews R grafikai ablakból, illetve a SciViews R-konzolból képeket, táblázatokat (53. ábra). Az R-környezetből származó ábrákhoz, számszerű eredményekhez magyarázatokat fűzhetünk, így teljes jelentések készíthetők.



53. ábra. SciViews R jelentésszerkesztő

SciViews R Bundle függvények

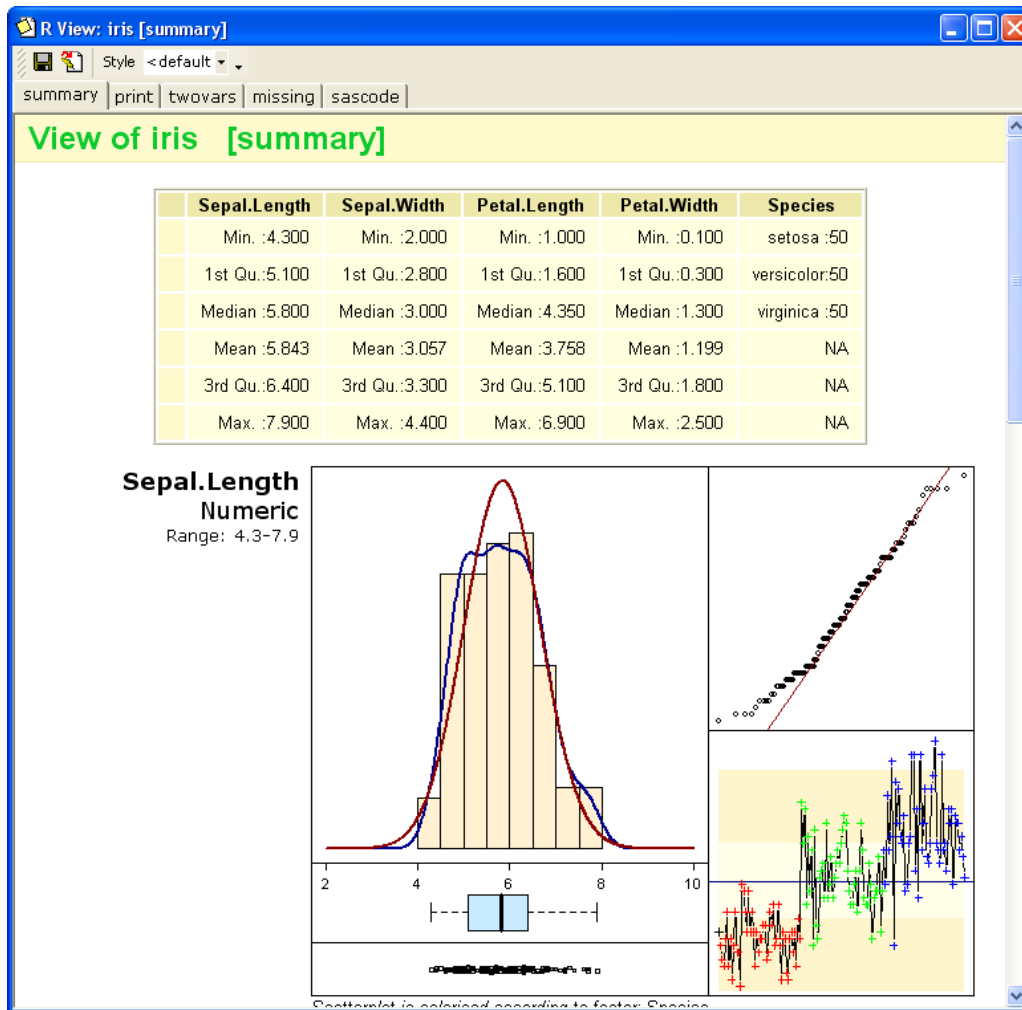
Az alábbi függvények a *SciViews R Bundle* részei. Parancssorból hívhatjuk meg őket.

export

Segítségével egyes R-objektumokat tudunk exportálni, különböző kimeneti formákban.

```
export(x, type = "raw", file, append = FALSE,
       objname = deparse(substitute(x)), ...)
```

Az *x* az exportálandó objektum, *type* argumentumnak pedig megadhatjuk a *raw*, az *ascii*, a *html*, a *latex* vagy a *sascode* valamelyikét.



54. ábra. view(iris)

copy

A `copy` tulajdonképpen az `export` függvénynek speciális esete, amikor is a `file` argumentumnak "clipboard" értéket adunk meg.

```
copy(x, type = "raw", objname = deparse(substitute(x)), ...)
```

Az argumentumok az `export`-nál láthatóan adhatók meg.

view

A `view` függvény lehetőséget teremt arra, hogy az adott R-objektum(ok)-ból könnyen, gyorsan készíthessünk előzetes feltáró jellegű ábrákat, összegző táblázatokat.

```
view(x, type = "summary", objname = deparse(substitute(x)), ...)
```

Az `x` az adott R-objektum, a `type` pedig lehet "summary", "print", "twovars", "missing", "methods" vagy "sascode". Az, hogy milyen „típusoknak megfelelő elemzések” jelennek meg a `view`-ban, a forrásként szolgáló R-objektumtól függ. A `view(iris)` utasítás eredményét mutatja az 54. ábra. Az `R view` ablak megjelenésének stílusát megváltoztathatjuk (<default>, Pastel, R2HTML, Report, SciViews), tartalmát elmenthetjük, illetve átadhatjuk a `SciViews R Report`-nak.

clippaste

A `copy` utasítással a vágólapra helyezett objektumot egy másik objektumba illeszti be.

```
clippaste(name = "newobj", type = "ascii", objclass = "data.frame", pos = 1, ...)
```

report

A `report` függvényt jelentésgenerálásra használhatjuk. Például a `report(iris)` generál egy táblázatot az `iris` adatokból, amit a *SciViews R Report*-ban jelenít meg.

ODBC-kapcsolat létrehozása

A következőkben (kezdő felhasználók részére) néhány ODBC-kapcsolat létrehozását mutatom be Microsoft Windows környezetben. A műveletek egy része a különböző adatforrásokhoz megegyezik:

- A *Start* menüből kiválasztjuk a *Beállítások* menü belüli *Vezérlőpult* almenüt.
- A megjelenő fájlkezelőből kiválasztjuk a *Felügyeleti eszközök* parancsikont
- A megjelenő indítóikonok közül kiválasztjuk az *ODBC adatforrások* elnevezésűt
- Az 55. ábrán látható párbeszéd-ablakon a *Felhasználói DSN* fülecskével ellátott felületen a *Hozzáadás...* gombra kattintva megjelenik az *Új adatforrás létrehozása* elnevezésű űrlap, amelyen az egyes meghajtóknak megfelelően folytatjuk a kapcsolat felépítését.
- Az adott adatforrásnak megfelelő illesztőprogramot kiválasztjuk az űrlap listájából (56. ábra)

Microsoft Excel

Az Excel esetében az adott *munkafüzet*hez az alábbi illesztőprogramok közül választhatunk :

- Driver do Microsoft Excel (*.xls)
- Microsoft Excel Driver (*.xls)
- Microsoft Excel-Treiber (*.xls)

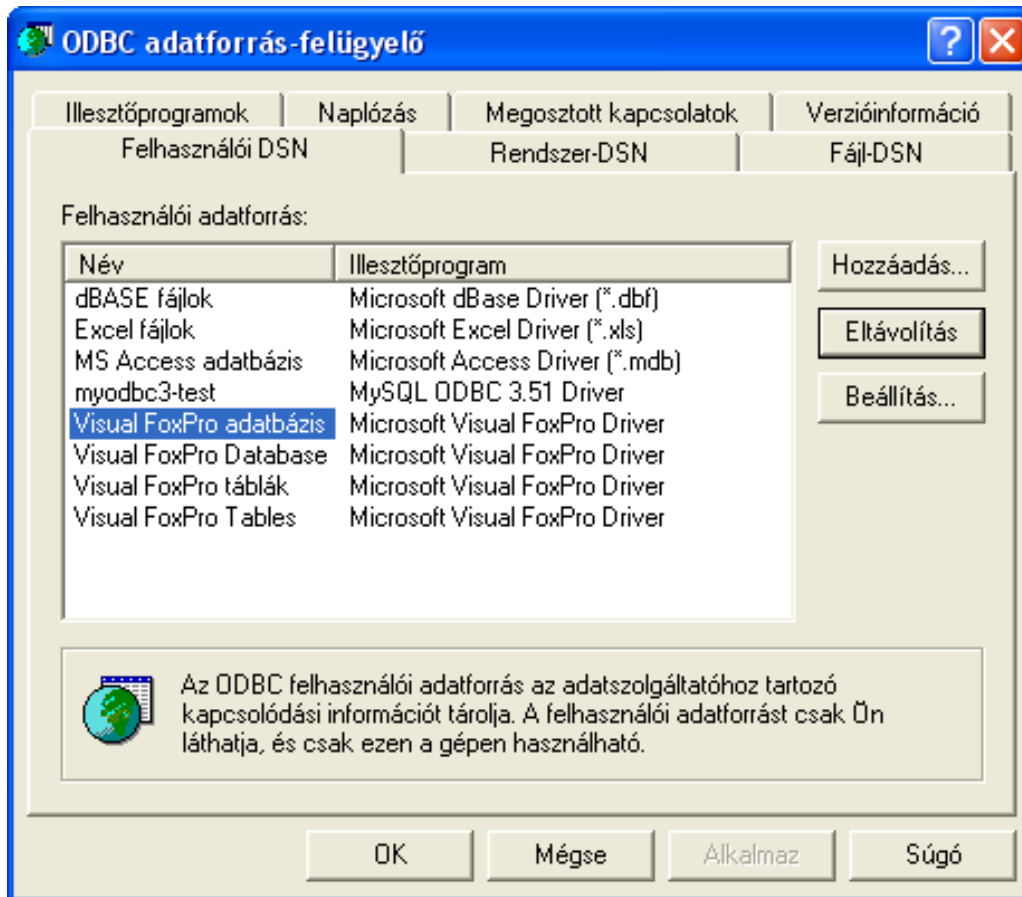
A fentiek közül valamelyiket kiválasztva és a *Befejezés* gombra kattintva megjelenik az 57. ábrán látható űrlap. Ezen első lépésben nevezzük el a létrehozandó kapcsolatot (a példában: *ODBCexcel*). Ezután tanácsos kiválasztani a *Verzió* legördülő menüből az adott Excel munkafüzet verzióját. Következő lépésként a *Munkafüzet választása...* gomb megnyomása következtében megjelenő fájlkezelő segítségével kiválasztjuk azt az Excel munkafüzetet, amelyhez a kapcsolatot építjük. Végül nincs más dolgunk, mint az *OK* gombra kattintani, így az 55. ábrán látható *Felhasználói adatforrás* listájában megjelenik az új ODBC-kapcsolat neve.

Microsoft Access

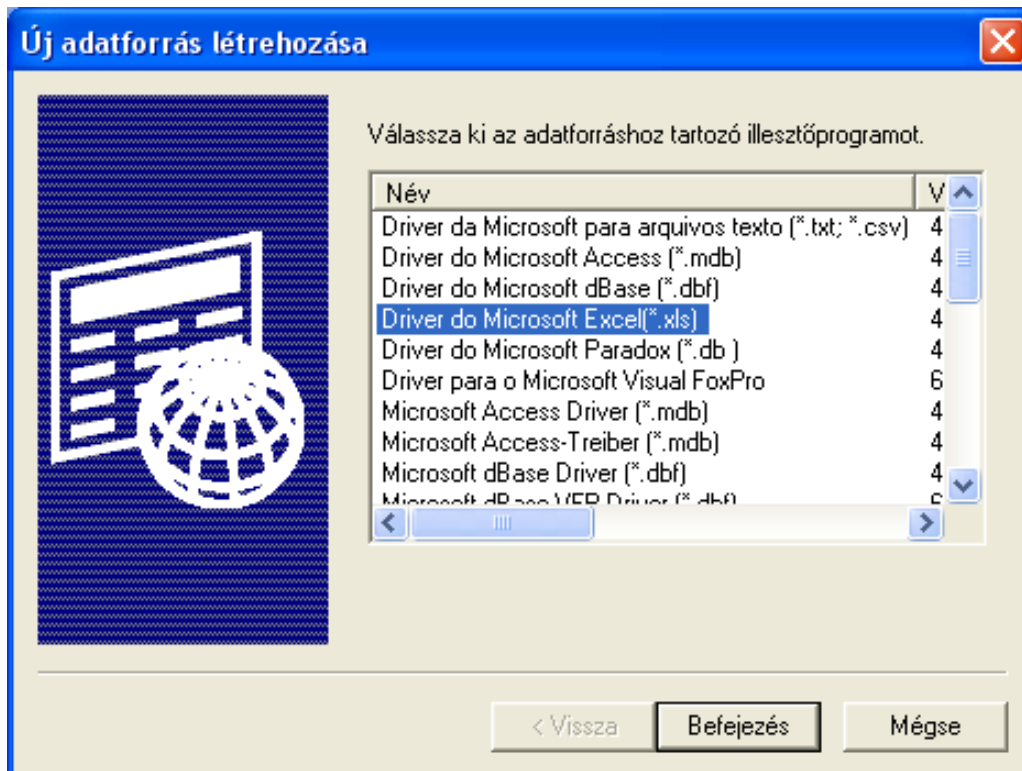
Az Access esetében az adott *munkafüzet*hez az alábbi illesztőprogramok közül választhatunk :

- Driver do Microsoft Access (*.mdb)
- Microsoft Access Driver (*.mdb)
- Microsoft Access-Treiber (*.mdb)

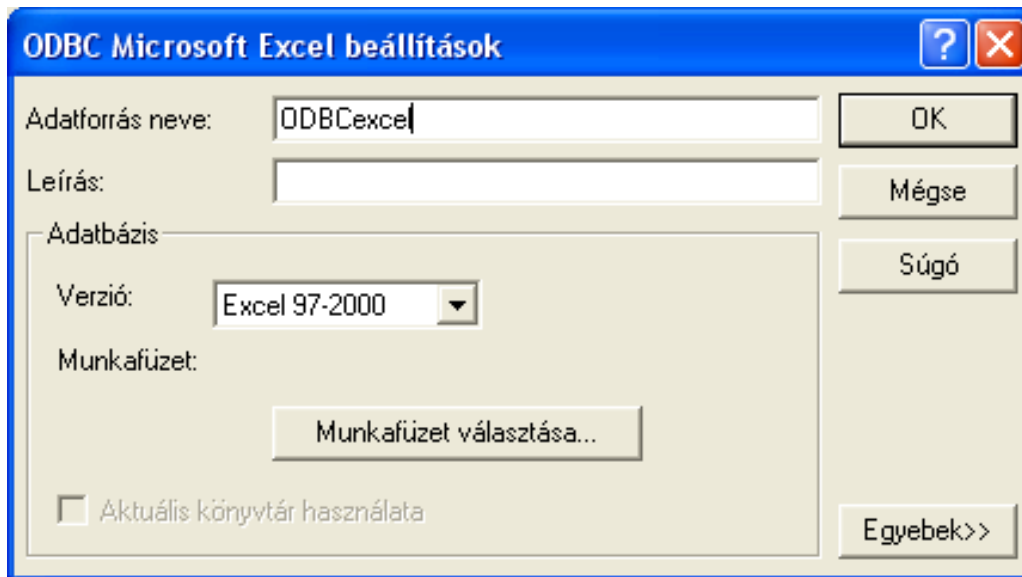
Az 58. ábrán látható űrlapon az Excelhez hasonlóan el kell neveznünk a kapcsolatot (a példában: *ODBCaccess*). A *Kiválasztás...* gomb megnyomása után megjelenő fájlkezelővel megkeressük a Microsoft Access (.mdb) adatbázist, amihez kapcsolatot szeretnénk kiépíteni. Az *OK* gombra kattintva a az 55. ábrán látható *Felhasználói adatforrás* listájában máris megjelenik az új ODBC-kapcsolat neve.



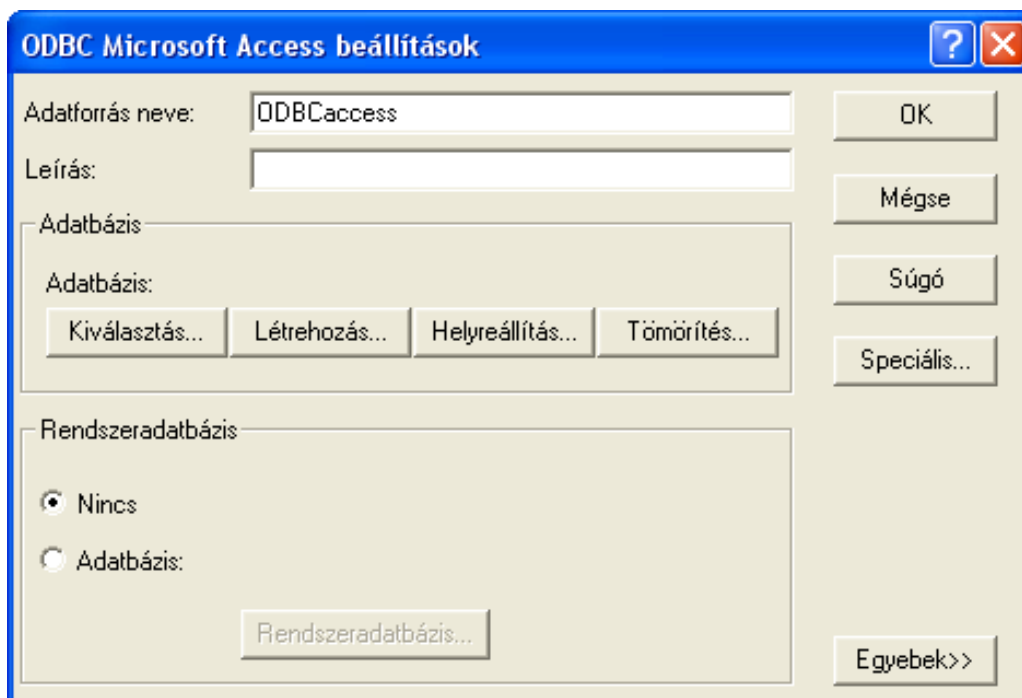
55. ábra. ODBC meghajtó kiválasztása



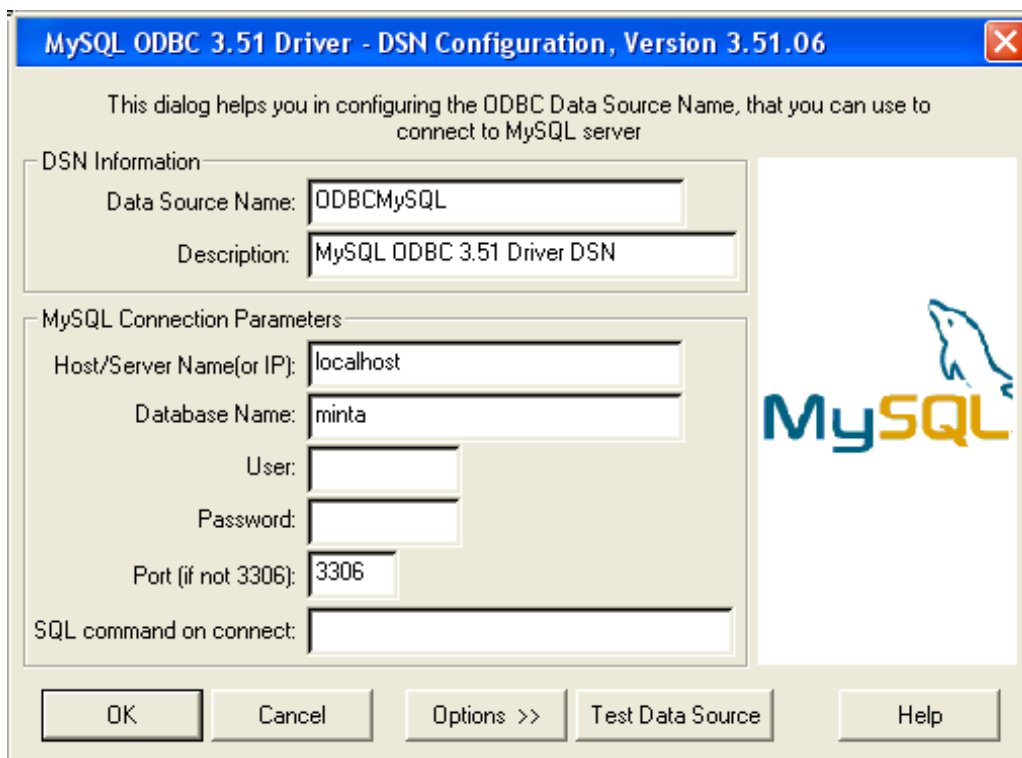
56. ábra. ODBC illesztőprogram kiválasztása



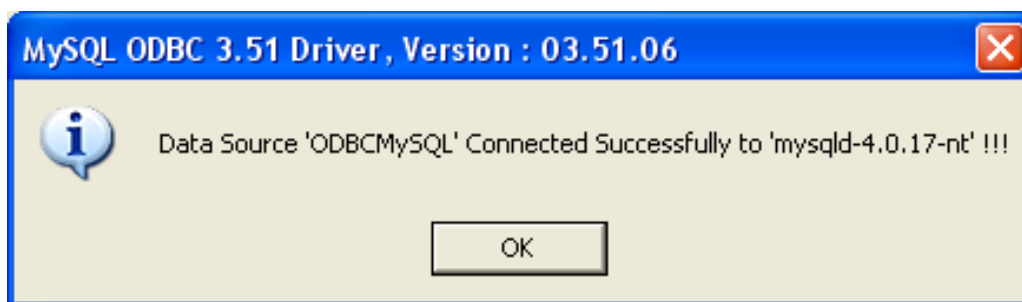
57. ábra. Az illesztő beállítása Excel munkafüzethez



58. ábra. Az illesztő beállítása Access adatbázishoz



59. ábra. Az illesztő beállítása MySQL adatbázishoz I.



60. ábra. Az illesztő beállítása MySQL adatbázishoz II.

MySQL

Ahhoz, hogy ODBC-kapcsolatot tudjunk létrehozni egy *MySQL*-adatbázishoz, előtte telepíteni kell a *MySQL Connector/ODBC*³⁶ meghajtót. Az 56. ábra illesztőprogram listájából kiválasztjuk a *MySQL ODBC 3.51*³⁷ *Driver*-t és a *Befejezés* gombra kattintva az 59. ábrán látható űrlap jelenik meg. Az űrlapon legalább két mezőt ki kell töltenünk: a *Data Source Name* lesz az adatforrás neve (ODBCMySQL), a *Database Name* pedig az az adatbázis (*minta*), amihez az interfészt akarjuk kiépíteni. További mezők is kitöltendők, ha szükséges (a *User* és a *Password*), de ez általában nem kell. A kapcsolat beállításainak ellenőrzésére érdemes a *Test Data Source* gombra kattintanunk. Ha minden rendben van, akkor a 60. ábrához hasonló üzenetet kapunk. A korábbi két kapcsolattípushoz hasonlóan, az 59. ábra *OK* gombjára kattintva az 55. ábrán látható *Felhasználói adatforrás* listájában megjelenik az új ODBC kapcsolat neve.

PostgreSQL

A *PostgreSQL* szerver telepítését nem kell kiegészítenünk külön ODBC-meghajtó installálásával. Az 56. ábra által jelzett űrlapról kiválasztjuk a *PostgreSQL* listaelemet. A *Befejezés* gombra kattintva a 61. ábrán látható űrlap jelenik meg. Ezen a felületen mindenképpen ki kell tölteni a következő mezőket: *Data Source*, *Database*, *Server*, *User Name* és *Password*. Az első elem lesz a *Felhasználói adatforrás* listájába bejegyzett

³⁶<http://www.mysql.com/products/connector/odbc/>

³⁷A verziószám – természetesen – a telepített verziótól függ.



61. ábra. Az illesztő beállítása PostgreSQL adatbázishoz

név (ODBCpgSQL). A második pedig az adatbázis neve (*minta*), amivel a kapcsolatot szeretnénk létrehozni. A harmadik elem a szerver neve, ami ha nem hálózati, akkor valószínűleg *localhost* lesz. Az adott adatbázishoz jogosultsággal rendelkező felhasználónevet is meg kell adni, illetve annak jelszavát is. Ha mindent jól adtunk meg, akkor a *Save* gombra kattintva kattintva az 55. ábrán látható *Felhasználói adatforrás* listájában megjelenik az új ODBC kapcsolat neve.

Szoftverintegráció

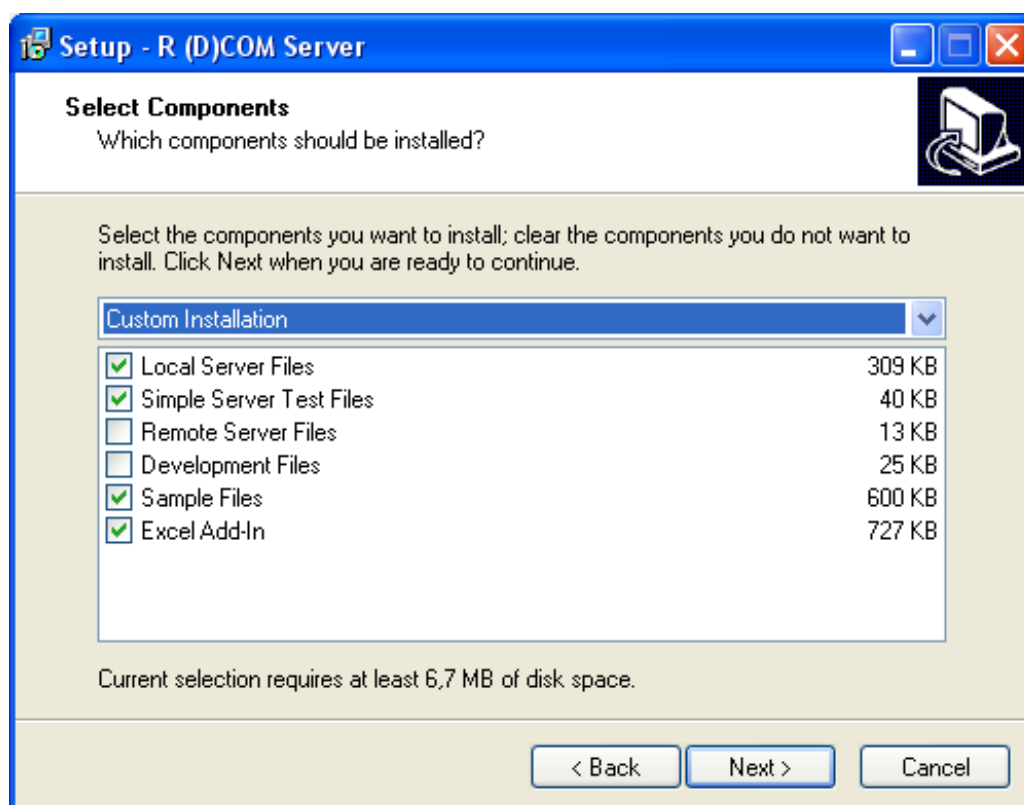
Néhány egyszerű példán keresztül mutatom be annak lehetőségeit, hogy egyéb szoftverekkel, programozási nyelvekkel hogyan lehet összekapcsolni az R-környezet képességeit.

R (D)COM Server

Microsoft Windows operációs rendszerre elérhető egy COM szerver (*R (D)COM Server*³⁸), ami lehetővé teszi, hogy kliens-szoftverek kapcsolatot teremthessenek az R-környezettel.

Microsoft Excel

Ha az *R (D)COM Server* telepítése során az Excel Add-In jelölőnégyzetét bejelöljük, akkor a Microsoft Excel menürendszerében az *Ablak* és a *Súgó* menüpontok között megjelenik egy *RExcel* menüpont. Az *RExcel*



62. ábra. R (D)COM Server telepítő képernyő

almenüpontjai között található az *R Start* opció, amivel az Excel–R kapcsolatot megnyitjuk. Ennek következtében az Excelen belül használhatunk R-függvényeket, -objektumokat. Három különböző módon dolgozhatunk az R-rel az Excelen belül: *jegyzetfüzet-mód*, *makró-mód* és *munkafüzet-mód*.

A *jegyzetfüzet-módban* az R-kódokat közvetlenül használhatjuk az Excel munkafüzetben belül, illetve adatobjektumokat tudunk a két környezet között átadni. Az *RExcel* menüből elérhető parancsokat használhatjuk: *R Start*, *Close R*, *Run Code*, *Get*, *Put*, *Copy Code*, *Debug R*, *Options*, *RExcel Help*, *Demo worksheets*, *R Help*.

³⁸<http://cran.r-project.org/contrib/extra/dcom>

A *makró-módban* VBA-nyelven írhatunk makrókat, a következő eljárások és függvények felhasználásával:

```
RInterface.StartRServer(),
RInterface.StopRServer(),
RInterface.RRun(commandstring),
RInterface.PutArray(varname,range),
RInterface.GetArray(varname,range).
```

A *munkafüzet-módban* közvetlenül hívhatunk meg R-függvényeket az Excel-cellákban. Ehhez a következő függvényeket használhatjuk:

```
RVarSet(var,expression,...),
RPut(var,range,...),
RStrPut(var,range,...),
REval(range,...),
REvalC(range,component,...),
RApply(function,args),
RApplyC(function,component,args),
RApplyA(function,args,...),
RApplyAC(function,component,args,...),
RExec(range,...),
RCall(function,args),
RCallA(function,args,...),
MakeArgs(arange,transpose=FALSE).
```

Microsoft Visual Basic

A Microsoft Visual Basic felhasználásával fejleszthetünk olyan alkalmazásokat, amelyek az R-környezet adatait is magukban foglalják. Egy példa:

```
Private Sub Command1_Click()
    Dim sconn As StatConnector
    Dim gfxdev As ISGFX

    Set sconn = New StatConnector

    sconn.Init "R"

    Set gfxdev = Gfx1.GetGFX
    sconn.AddGraphicsDevice "dev1", gfxdev

    sconn.EvaluateNoReturn "plot(sin(1:10))"
    sconn.EvaluateNoReturn "a<-0:100"
    sconn.EvaluateNoReturn "b<-sin(a)"
    sconn.EvaluateNoReturn "x<-sin(1:10)"
    ' sconn.EvaluateNoReturn "plot(x)"
    ' sconn.EvaluateNoReturn "demo(""graphics"")"
    sconn.Close
End Sub

Private Sub Form_Resize()
    Gfx1.Width = Width - 800
    Gfx1.Height = Height - 400
    Command1.Left = Width - 750
End Sub
```

Python

Az R-környezet használható Python-nyelv alkalmazásával is, ezáltal a Python-szoftverek kihasználhatják az R-nyelv statisztikai, grafikai képességeit. Ahhoz, hogy az R (D)COM-ot elérhessük a Pythonnal, telepítenünk kell a *win32com* interfészt³⁹. Ezek után már meghívhatjuk az R-függvényeket a Python-értelmezőn is, például a következő módon:

³⁹<http://www.python.org/windows/win32com/>

```
from win32com.client import Dispatch
sc=Dispatch("StatConnectorSrv.StatConnector")
sc.Init("R")
m=sc.Evaluate("b<-matrix(rnorm(20),5,4)")
print m
```

RPy

Az *RPy*-csomag telepítésével⁴⁰ létrehozunk egy interfészt a Python és az R-környezet között. Ez az interfész lehetővé teszi, hogy platformfüggetlen módon írassunk olyan programokat, amelyek mind Python, mind pedig R-kódokat tartalmaznak. A telepítőcsomag kiválasztásánál nagyon figyeljünk, hogy mind a rendszerünkön telepített Python-nak, mind az R-nek megfelelő verziót szerezzünk be. Egy nagyon egyszerű példa látható az alábbiakban:

```
from rpy import *
r.sum(r.rnorm(300))
```

Az `r.` előtaggal jelezzük azt, hogy az R-környezet függvényét hívjuk meg.

⁴⁰<http://rpy.sourceforge.net/>

Irodalomjegyzék

Táblázatok jegyzéke

1.	Fontosabb <code>typeof</code> visszatérési értékek	19
2.	A <i>típus</i> , <i>mód</i> és <i>tárolási mód</i> kombinációk	20
3.	Típus-konverziók	20
4.	Véletlen sorozatok	24
5.	Aritmetikai operátorok	38
6.	Mátrix-függvények	40
7.	Foreign csomag függvények	43
8.	A <code>read.table</code> függvénycsoport különbségei	43
9.	A <code>formatC</code> értékformáló kódjai	51
10.	Grafikai meghajtók	53
11.	Összehasonlító operátorok	65
12.	Logikai operátorok	66
13.	String-függvények	69
14.	Speciális karakterek	70
15.	Általános függvények	70

Tárgymutató

- .RData, 10
- .Rhistory, 10
- <-, 9

- abind, 99
- abs, 70
- acos, 70
- adatbázis, 46
- add, 57
- adj, 59
- all.screens, 57
- ann, 59
- apply, 65
- apropos, 15, 92, 94
- array, 26
- as.matrix, 47
- asin, 70
- ask, 59
- assign, 9
- atan, 70
- attach, 34
- attr, 19
- attributes, 19
- axes, 57

- bg, 59
- bitmap(), 53
- bmp(), 53
- break, 65

- c, 20, 21
- car, 99
- ceiling, 70
- cex, 59
- cex.axis, 59
- cex.main, 59
- cex.sub, 59
- character, 25
- chron, 99
- cin, 59
- class, 20
- clippaste, 109
- col, 59
- col.axis, 59
- col.lab, 59
- col.main, 59
- col.sub, 59
- copy, 109
- cor, 70
- cos, 70
- cov, 70
- cra, 59

- crt, 59
- csi, 59
- csv, 42
- cxy, 59

- D, 35
- DAAG, 99
- data.entry, 36
- data.frame, 28, 31, 33, 47
- data.restore, 43
- dataentry, 36
- DBMS, 46
- de, 36
- demo, 16
- dev.cur(), 53
- dev.list(), 52
- dev.next(), 53
- dev.off(), 53
- dev.prev(), 53
- dev.set, 53
- dget, 45
- dim, 19, 26, 28
- dimnames, 20, 28
- din, 59
- djmrgl, 63
- dos, 42
- dput, 49
- dump, 49

- effects, 99
- Emacs, 10
- erase, 57
- err, 59
- eval, 34
- example, 15
- exp, 70
- export, 108
- expression, 34

- factor, 25
- family, 59
- fBasis, 99
- fg, 59
- fig, 59
- figs, 57
- fin, 59
- fix, 26, 36, 91
- floor, 70
- font, 59
- font.axis, 59
- font.lab, 59
- font.main, 59

- font.sub, 59
- for, 65
- formatC, 50
- ftable, 27

- gamma, 59
- getwd, 41
- gl, 23
- GNU, 6
- gregmisc, 42

- heights, 55
- help, 11, 92, 94
- help.search, 14, 92, 94
- help.start, 12, 92, 94
- history, 50
- Hmisc, 97
- HTML, 50

- identify, 58
- indexelés, 31
- iplots, 61
- is.data.frame, 34
- is.list, 34
- is.matrix, 34
- its, 99

- jpeg(), 53

- Kate, 10
- KLIMT, 61

- lab, 60
- lapply, 65
- las, 60
- LaTeX, 50
- leaps, 99
- lend, 60
- length, 22, 35, 70
- lheight, 60
- list, 30, 33
- ljoin, 60
- lmitre, 60
- lmtest, 99
- load, 91
- loadhistory, 50, 91
- log, 57, 70
- log10, 70
- logical, 25
- lookup.xport, 43
- looping, 65
- ls, 10, 19, 49, 91
- lty, 60
- lwd, 60

- magyarítás, 45
- main, 57
- man, 11
- mapply, 65
- mar, 60
- mat, 55
- matrix, 27, 31

- max, 70
- MDI, 90
- mean, 11, 70
- median, 70
- mex, 60
- mfcol, 60
- mfg, 60
- mfrow, 60
- mgp, 60
- Microsoft Access, 111
- Microsoft Excel, 41, 111
- min, 70
- mkh, 60
- mode, 19
- multcomp, 99
- mvtnorm, 99
- MySQL, 114

- names, 33
- ncol, 50
- new, 57
- next, 65
- numeric, 25

- objects, 10, 19
- ODBC, 46, 114
- oma, 60
- omd, 60
- omi, 60
- oz, 99

- paste, 69
- pch, 61
- pdf(), 53
- Perl, 42
- pictex(), 53
- pin, 61
- plt, 61
- png(), 53
- PostgreSQL, 114
- postscript(), 53
- próba, 6
- prod, 70
- ps, 61
- pty, 61

- quadprog, 97
- quantreg, 99
- quartz(), 53

- range, 70
- RDBMS, 46
- read.csv, 43
- read.csv2, 43
- read.dbf, 43
- read.delim, 43
- read.delim2, 43
- read.dta, 43
- read.epiinfo, 43
- read.mtp, 43
- read.octave, 43
- read.S, 43

read.spss, 43
read.ssd, 43
read.systat, 43
read.table, 43
read.xls, 42
read.xport, 43
readLines, 42
relimp, 99
rep, 22
repeat, 65
report, 110
respect, 55
Rggobi, 63
rgl, 63, 99
rJava, 61
rm, 10, 91
round, 70

S, 6
sandwich, 99
save, 48
save.image, 49, 91
savehistory, 50, 91
scan, 21
screen, 57
SDI, 90
search, 91
seq, 21
sequence, 23
setwd, 41, 91
sin, 70
sink, 49
smo, 61
source, 10, 49, 90
speciális karakterek, 69
sqrt, 70
srt, 61
storage.mode, 19
str, 35
strucchange, 99
sub, 57
sum, 70
summary, 35
svGUI, 103
Sys.getlocale, 46
Sys.setlocale, 46

t, 47
tan, 70
tapply, 65
tck, 61
tcl, 61
tmag, 61
try.all.packages, 14
ts, 30
tseries, 99
tsp, 20
type, 57, 61
typeof, 19

unlink, 50
update.packages, 92, 94
usr, 61

var, 70
vector, 25, 31
view, 109

which.max, 70
which.min, 70
while, 65
widths, 55
win.metafile(), 53
win.print(), 53
windows(), 53
write, 47
write.dbf, 43
write.dta, 43
write.foreign, 43
write.table, 48

X11(), 53
xaxp, 61
xaxs, 61
xaxt, 61
Xemacs, 10
xfig(), 53
xgobi, 61
xlab, 57
xlog, 61
xls, 41
xls2csv, 42
xpd, 61
xtable, 50

yaxp, 61
yaxs, 61
yaxt, 61
ylab, 57
ylog, 61

zoo, 99