Bevezetés az R-be

Oktatási segédlet - VÁZLAT

Abari Kálmán
DE, Pszichológiai Intézet
abari.kalman@gmail.com

Tartalomjegyzék

Előszó

E jegyzet célja az R nyelv és környezet elsajátításának segítése. A jegyzetet elsősorban a kezdő R felhasználóknak ajánljuk. A leírtak megértéséhez a középiskolai matematikán túl semmilyen előzetes ismeret nem szükséges.

A jegyzetben az R 2.6.2-es (2008-02-08) verziójának Windows platformra szánt változatát használjuk. Ez semmifajta megszorítást nem jelent, a használt parancsok a későbbi verziókban és az egyéb platformokon (Linux, MacOS) futó R példányokban is ugyanígy használhatók.

Örömmel fogadjuk az Olvasóink észrevételeit, amelyeket az `abari.kalman@gmail.com` címre várjuk.

1. Bevezetés

1.1. Mi az R?

Az R egy magas szintű programozási nyelv és környezet, melynek legfontosabb felhasználása az adatelemzés és az ahhoz kapcsolódó grafikus megjelenítés.

[…]

Az R nyelv egy interpretált szkript nyelv, azaz az utasításainkat nem fordíthatjuk le futtatható bináris állománnyá, hanem a végrehajtandó parancsokat az R egyesével értelmezi (ellenőrzi a parancs szabályosságát, ha megfelelőnek találja rögtön végrehajtja). Az R program legfontosabb része tehát az R-értelmező (interpreter), amely a parancsok végrehajtásáért felelős.

Az R-interpreterhez kétféle módon juttathatunk parancsot, vagy interaktívan, az egyes parancsok begépelésével, vagy „kötegelten" ún. szkript módban, előre összegyűjtött parancsok formájában.

A parancsok eredménye megjelenhet a terminálban, állományokban vagy adatbázisokban is, de grafikus megjelenítésre is van lehetőségünk.

A legegyszerűbb alapműveletektől kezdve a bonyolult statisztikai eljárásokig nagyon sokféle művelet hajtható végre az R-ben. Az egyes műveletek eredményeit ún. (adat)objektumokban tárolhatjuk. A műveletek pedig ún. operátorok és függvények formájában jelennek meg az R-ben.

Az R számos beépített függvényt bocsát a felhasználók rendelkezésére, de számos kiegészítés is készült az R-hez, melyeket ún. csomagok (package) formájában érhetünk el. Egy csomag függvényeket és adatobjektumokat tartalmazhat, amelyeket a csomag telepítésével és betöltésével tehetünk elérhetővé.

1.2. Az R telepítése

Első közelítésben gondoljunk úgy az R-re, mint egy többplatformos alkalmazásra, vagyis egy olyan programra, amely futtatható Windows, Linux és MacOS operációs rendszereken is. Az R szabad szoftver, tehát bátran letölthetjük a kedvenc operációs rendszerünknek megfelelő telepítő példányt az R hivatalos oldaláról (http://www.R-project.org), majd mint egy közönséges alkalmazást, a szokásos módon telepíthetjük a számítógépünkre.

A Windows operációs rendszerekre szánt verzió részletes telepítése megtalálható [1]-ben, a 80-84. oldalon.

1.3. Munkafolyamat (session) az R-ben

Az R program (rendszer) sikeres telepítése után az R futtatása, a megfelelő platform bináris állományának elindítását jelenti. A Windows operációs rendszerekben ez többnyire az Asztalon lévő R ikon segítségével lehetséges.

Az R indítása után a terminál vagy konzol ablakban megjelenik a program verziószáma és a kiadási dátuma. Jó gyakorlat lehet, ha rendszeresen ellátogatunk az R hivatalos oldalára, ahonnan az R aktuális példányának a letöltésével, mindig az R legfrissebb példányát használhatjuk. Érdemes az új példány telepítése előtt, a régi változatot eltávolítani a számítógépünkről. Ügyeljünk arra, hogy az R frissítése után a csomagokat újra kell telepíteni.

Fontos lehet az R-el végzett munkáink publikálásánál, hogy hogyan hivatkozunk erre a programra. A következő parancs begépelése után, a megjelenő fejrészből ez is kiolvasható:

```
> citation()
```

Az R indítása és leállítása közötti rész egy munkafolyamatot (session) határoz meg. A munkafolyamat során létrehozott objektumokat (lsd. később) az R név szerint tárolja. A létrehozott objektumok együttesét munkaterületnek (workspace) nevezzük. A munkafolyamathoz tartozik egy munkakönyvtár (working directory) is. Ez a könyvtár az alapértelmezett könyvtár abban az esetben, ha a parancsok megadásánál nem használunk könyvtárhivatkozást.

A Windows operációs rendszerekben az R ikonhoz tartozó 'Indítás helye' mező határozza meg az alapértelmezett munkakönyvtárát. A munkakönyvtár az R-ben lekérdezhető ill. beállítható a **getwd()** és a **setwd()** parancsok kiadásával.

```
> getwd()
[1] "C:/Documents and Settings/Abari Kálmán/Dokumentumok"

> setwd("C:/Documents and Settings/Abari Kálmán/Dokumentumok/peldak")
```

A munkakönyvtár jelentőségét tovább növeli, hogy az R indításakor ebben a könyvtárban 2 állomány létezését figyeli:

- .Rhistory (a visszahívható parancsokat tartalmazó szöveges állomány)
- .RData (a tárolt objektumokat tartalmazó bináris állomány).

A fenti állományok ugyanis betöltésre kerülnek az R indításakor, ha azokat az R megtalálja a munkakönyvtárban. Így ezek után, az .Rhistory állományból jövő parancsok között válogathatunk a parancssor használata során, ill. az .RData állományban tárolt objektumok azonnal elérhetőek, lesz egy induló munkaterületünk.

Ezeket az állományokat legtöbbször az R-ből való kilépés során, maga a rendszer hozza létre, ehhez csak igennel kell válaszolnunk a munkaterület mentésére vonatkozó kérdésre.

Azonban a fentiekhez hasonló szerkezetű állományokat mi is létrehozhatunk a munkánk során. A parancsok történetét tartalmazó szöveges állomány a **loadhistory()** és a **savehistory()** parancsokkal tölthető be és írható ki. A tárolt objektumokat a **load()** és a **save()** vagy **save.image()** parancsok töltik be és írják ki.

A munka(folyamat) befejezése a **quit()** vagy **q()** parancsok kiadásával lehetséges.

1.4. Az R parancssoros használata

Az R program indítása után a terminálban vagy a konzol ablakban megjelenő '>' (nagyobb jel) jelzi számunkra, hogy az R várja a parancsainkat. Ez az ún. prompt, amely után egy tetszőleges karaktersorozat begépelésére van lehetőségünk. A begépelt parancsot az ENTER billentyűvel küldjük el az R értelmezőnek, amely ha mindent „rendben" talál a parancsunkban, akkor végrehajtja azt.

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

A parancssort piros színezéssel emeljük ki a jegyzetben, az R válaszát a begépelt parancsra kék színnel jelöljük.

Nem minden parancs ad választ az R-ben:

```
> x<-0
```

Sőt, az is előfordulhat, hogy az R nem talált valamit „rendben” a parancsban. A begépelt sort ekkor természetesen nem hajtja/hajthatja végre, helyette hibát (error) jelez.

```
> Ez nem lesz jó.
Error: unexpected symbol in "Ez nem"
```

Hosszabb, bonyolult parancsok gépelésénél gyakran előfordul, hogy nem „teljes” a begépelt parancs, valami még hiányzik belőle (pl. egy záró kerek zárójel). Ezt az R észreveszi és az ENTER megnyomása után egy ’+’ prompt megjelenítésével jelzi ezt számunkra. A ’+’ prompt után van lehetőségünk a hiányzó részek pótlására, majd ha készen vagyunk az ENTER billentyűvel az összes eddig még végre nem hajtott sort elküldhetjük az értelmezőnek.

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

Ha nem találjuk a hiányzó részt a parancsunkban, akkor az R-nek ezen kényelmi funkciója oda vezethet, hogy örökké ’+’ promptot kapjuk az ENTER megnyomása után. Ezt a helyzetet hivatott megoldani az ESC billentyű, amellyel megszakíthatjuk az értelmező aktuális feldolgozási folyamatát, azaz ebben a szituációban egy új sort, immár egy ’>’ prompttal kezdődő (üres) sort kapunk a konzol ablakban.

Az R parancssoros használatát még további 2 dolog teszi kényelmesebbé. Egyrészt a korábban végrehajtott parancsainkat (history) visszahívhatjuk, lapozhatunk bennük előre, hátra. Erre a FEL/LE NYÍL billentyűkkel van lehetőségünk. Természetesen az így visszahívott parancsot tetszőleges módon átszerkeszthetjük: navigálhatunk a sorban előre hátra, beszúrhatunk/törölhetünk karaktereket vagy használhatjuk a vágóasztal billentyűparancsait. A visszahívott és módosított parancsot az ENTER segítségével újra végrehajthatjuk, és ehhez még a sor végére sem kell a szövegkurzort pozícionálni, az a sorban tetszőleges helyen állhat, az R mégis a teljes sort fogja értelmezni.

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

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

1.5. Szkriptek végrehajtása

Az R-et nemcsak interaktívan, parancsok egymás utáni begépelésével használhatjuk, hanem ún. szkript módban is. A végrehajtandó parancsokat összegyűjtjük és egy szöveges állományban eltároljuk (pl. ’parancsok.R’), majd az összes parancsot egyetlen sor begépelésével elküldjük az értelmezőnek.

```
> source("parancsok.R")
```

A fenti parancs feltételezi, hogy a szkript állomány (’parancsok.R’) az R környezet munkakönyvtárában van, ha esetleg más könyvtárban található, akkor az elérési utat is meg kell adnunk.

1.6 A csomagok használata

Az R indítása után néhány csomag automatikusan betöltésre kerül. Ezeket a csomagokat és az egyéb környezeteket listázhatjuk ki a **search()** függvénnyel.

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

A fenti eredményben a ’package’ karaktersorozattal kezdődő elemek mutatják, hogy melyek az alapértelmezés szerint betöltött csomagok. A korábban telepített csomagok betöltéséhez használjuk a **library()** függvényt.

```
> library(foreign)
> search()
 [1] ".GlobalEnv"        "package:foreign"   "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

A fenti példában a **foriegn** csomag betöltését és annak hatását követhetjük nyomon. Ezután a csomagban lévő függvényeket és objektumokat a parancsainkban felhasználhatjuk.

Egy adott csomagban (esetünkben a **foreign** csomagban) lévő függvények és objektumok a

```
> library(help=foreign)
```

paranccsal kérdezhetők le. Betöltött csomagok esetében használhatjuk az

```
> ls("package:foreign")
```

parancsot is.

Számítógépünkre az **install.packages()** függvénnyel telepíthetünk csomagokat. Grafikus felhasználói felületet tartalmazó csomagot telepíthetünk az

```
> install.packages("Rcmdr")
```

paranccsal.

1.7. Segítség az R használatához

Az R használatához számos segítséget találunk az Interneten és a telepített R példányunkban egyaránt. Az online segítségek közül elsősorban a http://cran.r-project.org címen olvasható R dokumentációkat emeljük ki, ahol több tucat, elsősorban angol nyelvű leírást találunk az R megismeréséhez. Erről az oldalról tölthető le, az Irodalomjegyzékben [1] alatt szereplő magyar nyelvű leírás is. Az R népszerűségének köszönhetően, nagyon sok további dokumentációt, tutoriált és példát találhatunk, ha az internetes keresőkhöz fordulunk.

Az R környezetben további lehetőségek állnak rendelkezésre. Az R megismerését kezdhetjük a

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

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

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

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

vagy a rövidebb

```
> ?t.test
```

parancsot, ha a **t.test()** függvényről szeretnénk részletes leírását kapni. Abban az esetben ha nem ismerjük teljesen a függvény nevét, használhatjuk a

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

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

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

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

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

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

```
> apropos("aov")
[1] ".__C__aov"           ".__C__maov"           "aov"
[4] "eff.aovlist"         "model.frame.aovlist" "summary.aov"
[7] "summary.aovlist"     "terms.aovlist"        "TukeyHSD.aov"
```

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

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

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

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

parancsokat.

2. Az R alapjai

2.1. Számolás az R-ben

Kezdjük az R nyelv megismerését egy egyszerű sor begépelésével:

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

Az ENTER megnyomása után a konzolon láthatjuk az összeadás eredményét a 4-et. Az eredmény előtt egy szögletes zárójelben lévő sorszámot is láthatunk ([1]), amely bonyolultabb eredményekben segít eligazodni. Később visszatérünk ennek az értelmezésére.

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

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

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

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

Természetesen a matematikában megszokott módon változtathatunk a műveletek végrehajtásának alapértelmezett sorrendjén, használhatunk kerek zárójeleket. Ezeket az R a megszokott módon értelmezi: a zárójelben szereplő műveletek végrehajtását előreveszi. A fenti példában tehát az összeadás művelet lesz az első, amelynek az eredménye 10. Ezt követi a szorzás, így kapjuk a kifejezés értékeként a 20-at.

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

- számok, amelyeket az R *numerikus konstans*nak nevez,
- műveleti jelek, amit az R-ben *aritmetikai operátor*nak nevezünk,
- kerek zárójelek.

Az R a parancssorba írt kifejezések értékét kiszámolja *kiértékeli a kifejezést* és a kapott értéket megjeleníti.

A numerikus konstansok többféle alakban is megjelenhetnek az R-ben.

Numerikus konstans formája az R-ben	Értéke	Leírás
`1, -1, 2, 100, 3.5, .4`	`1, -1, 2, 100, 3.5, 0.4`	pozitív és negatív egész és nem egész számok

`1L, -1L, 2L, 100L`	`1, -1, 2, 100`	pozitív és negatív egész számok az 'L' suffix használatával
`1.2e3, 3e+4, .6e-2, 4e1L`	`1200, 30000, 0.006, 40`	exponenciális alakú egész és nem egész számok
`0xef, 0Xf01, 0xEf03, 0xd1L`	`239, 3841, 61187, 209`	hexadecimális (16-os számrendszerben felírt számok)

Az R-ben használható aritmetikai operátorokat a precedencia csökkenő sorrendjében a következő táblázat tartalmazza.

Operátor formája	Művelet	Példa	Példa eredménye
`^ (vagy **)`	hatványozás	`2^3` `2**3`	`8` `8`
`+ - (unáris)`	előjelek	`+3.3` `-.5`	`3.3` `-.5`
`%% %/%`	maradékos osztás és egész osztás	`13%%4` `15%/%4`	`1` `4`
`* /`	szorzás és osztás	`2*3` `4 / 2`	`6` `2`
`+ - (binér)`	összeadás és kivonás	`2 + 3` `2 - 3`	`5` `-1`

A fentiek alapján összetettebb aritmetikai kifejezéseket is megfogalmazhatunk az R parancssorában.

```
> 4^2-3*2
[1] 10

> (56-14)/6-4*7*10/(5**2-5)
[1] -7
```

Az aritmetikai kifejezések használata során ne felejtkezzünk meg az operátorok precedenciájáról sem.

```
> -2^2
[1] -4

> (-2)^2
[1] 4
```

Itt a hatványozás és az előjel operátor közötti végrehajtási sorrend okozza az eltérő eredményeket.

Eddig láthattuk, hogy kifejezéseinket operátorok, számok és zárójelek segítségével építettük fel. Ezek a kifejezések két részletükben is általánosíthatók: (1) egyrészt a kifejezés adat részében, amelyet eddig a numerikus konstansok képviseltek, (2) másrészt a kifejezés műveleti részében, amelyet eddig az operátorok jelenítettek meg. Az adatrész általánosítása az *(adat)objektum* a műveleteké pedig a *függvény*. Ezeket tekintjük át a következőkben.

2.2. Objektumok

Ha egy kifejezés értéket nem egyszerűen a képernyőn szeretnénk megjeleníteni, hanem azt később is fel akarjuk használni, akkor *objektum*okat kell létrehoznunk. A memóriában tárolt adatok elérésére az R-ben objektumokon keresztül lehetséges. (Más nyelvekben ezt változónak vagy szimbólumnak nevezik, ha nem okoz félreértést, akkor mi is használhatjuk ezeket az elnevezéseket.)

```
> 117/11+2^3
[1] 18.63636
```

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

```
> x <- 117/11+2^3
```

Ha azonban létrehozunk egy **x** nevű objektumot a fenti módon, akkor ezt az értéket további kifejezésekben is szerepeltethetjük. Ahol eddig számok jelentek meg a kifejezésben, oda ez az **x** objektumnév is beírható.

Írhatjuk tehát a következőket:

```
> x+2
[1] 20.63636

> 2*x^3+5
[1] 12950.34
```

Minden objektumnak van neve és tartozik hozzá a memóriában egy terület, ahol a kérdéses érték tárolásra kerül. Esetünkben az objektum neve **x** a hozzá tartozó memóriaterületen pedig a `18.63636` értéket tárolja az R.

Az objektumok kezelése az R-ben alapvetően 3 dolgot jelent: (1) az objektum létrehozása, (2) az objektum értékének lekérdezése és (3) az objektum értékének megváltoztatása. (Később látjuk, mi mindent tehetünk még az objektumokkal.)

2.2.1. Értékadás

Objektumot *értékadással* hozhatunk létre. Az értékadás tartalmaz egy *értékadás operátort*, melynek alakja '<-', vagyis egy 'kisebb jel' és egy 'mínusz előjel' egymás után írva szóköz nélkül. (További értékadó operátorok a '->', '<<-', '->>' és a '='. Ezekről később lesz szó.)

Az értékadás általános alakja: `objektumnév <- kifejezés`. Ahol lehet a továbbiakban ezt a „balra nyíl" alakú értékadó operátort használjuk az értékadás során. Az egyszerűség kedvéért a „balra nyíl" előtt lévő objektumnevet az értékadás „bal oldalának", az utána lévő kifejezést az értékadás „jobb oldalának" nevezzük.

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

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

Fent, tehát a korábban nem létező **a** objektumot hoztuk létre. Az értékadás után **a** már létező objektum, a munkafolyamatunk munkaterületéhez tartozik.

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

```
> a <- 10/3
```

Most tehát a már létező **a** objektum értékét változtattuk meg.

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

```
> a
[1] 3.333333
```

Objektumot azonban kifejezésekben is felhasználhatjuk, akár értékadás jobb oldalán lévő kifejezésben is.

```
> a*3
[1] 10

> a <- 4+a*3
> a
[1] 14
```

2.2.2. Objektumok elnevezése

Az objektumok elnevezésére az angol ábécé kis és nagy betűit, a számjegyeket és a pont (.) illetve az aláhúzás (_) szimbólumokat használhatjuk. Az objektum neve csak betűvel vagy ponttal kezdődhet, de ez utóbbi esetben a következő karakternek betűnek kell lennie. Hagyományosan a pont karaktert használhatjuk az objektumnevek tagolására. Az R a magyar ékezetes karakterek használatát is megengedi az objektumnevekben, de csakúgy mint az állományok elnevezésében, érdemes ezek használatát mellőzni.

Az objektumoknak érdemes „beszédes” nevet választani, még ha ennek az ára némi extra gépelés is. (Tudjuk, a TAB karakter ezen segíthet.)

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

```
> pulzus.atlag <- 72.86
```

parancs után a

```
> Pulzus.atlag
Error: object "Pulzus.atlag" not found
```

sor hibát jelez, azaz a ’Pulzus.atlag’ objektumot nem találja az R. Minden olyan esetben, ha nem létező objektumra hivatkozunk, a fenti hibaüzenet jelenik meg a konzolban.

```
> Pulzus.atlag <- 69.37          # Új objektumot hozunk létre
> pulzus.atlag; Pulzus.atlag
[1] 72.86
[1] 69.37
```

A fenti példában már gondoskodtunk a nagy 'P'-vel kezdődő objektumról is, így lehetőségünk van hibaüzenet nélkül mindkét objektum érékének kiíratására.

Ez a példa két további apró újdonságot is tartalmaz.

Megjegyzést az R-ben a kettős kereszt (#) karakter használatával vezetünk be, az R értelmező a kettős kereszttől a sor végéig tartó részt figyelmen kívül hagyja. Itt helyezhetjük el a paranccsal kapcsolatos magyarázatainkat magunk vagy a kódot később olvasók számára.

Továbbá, ha a parancssorban több utasítást szeretnénk elhelyezni, akkor ezeket pontosvesszővel (;) kell elválasztanunk. Ezeket az utasításokat az R értelmező egymás után, balról jobbra haladva hajtja végre, mintha külön sorba írtuk volna őket.

2.3. Függvények

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

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

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

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

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

A R számos beépített függvény hívását teszi lehetővé. A függvényhívás általános alakja:

```
függvénynév(arg1, arg2, …, argN)
```

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

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

Először tekintsük át az R alapvető matematikai függvényeit.

Függvény	Leírás	Példa	Példa értéke
`abs(x)`	abszolútérték függvény	`abs(-1)`	`1`

`sign(x)`	előjel függvény	`sign(pi)`	`1`
`sqrt(x)`	négyzetgyök függvény	`sqrt(9+16)`	`5`
`exp(x)`	exponenciális függvény	`exp(1)`	`2.718282`
`log(x)`	természetes alapú logaritmus	`log(exp(3))`	`3`
`log(x,base)`	tetszőleges alapú logaritmus	`log(100, 10)`	`2`
`log10(x)` `log2(x)`	10-es és 2-es alapú logaritmus	`log10(1000)` `log2(256)`	`3` `8`
`cos(x)`	koszinusz függvény (x radiánban mért)	`cos(pi)`	`-1`
`sin(x)`	szinusz függvény (x radiánban mért)	`sin(pi/2)`	`1`
`tan(x)`	tangens függvény (x radiánban mért)	`tan(0)`	`0`

`acos(x)`
`asin(x)`
`atan(x)`
	inverz trigonometrikus függvények (x radiánban mért)	`2*asin(1)`
		`acos(-1)`
		`atan(pi)`
			`3.141593`
			`3.141593`
			`1.262627`

`cosh(x)`
`sinh(x)`
`tanh(x)`
	hiperbolikus trigonometrikus függvények	`cosh(0)`
		`sinh(0)`
		`tanh(0)`
			`1`
			`0`
			`0`

`acosh(x)`
`asinh(x)`
`atanh(x)`
	inverz hiperbolikus trigonometrikus függvények	`acosh(0)`
		`asinh(0)`
		`atanh(0)`
			`NaN`
			`0`
			`0`

`round(x, digits=0)`	kerekítés adott tizedesre	`round(1.5)` `round(-1.5)`	`2` `-2`
`floor(x)`	x-nél kisebb, legnagyobb egész	`floor(1.5)` `floor(-1.5)`	`1` `-2`
`ceiling(x)`	x-nél nagyobb, legkisebb egész	`ceiling(1.5)` `ceiling(-1.5)`	`2` `-1`
`trunc(x)`	az x-hez legközelebbi egész x és 0 között	`trunc(1.5)` `trunc(-1.5)`	`1` `-1`

Nézzük meg részletesebben a **log()** függvényt. Ha kikérjük a súgóját a

```
> ?log
```

parancs begépelésével, akkor megtudhatjuk, hogy ez a legáltalánosabb logaritmus függvény, tetszőleges alap esetén hívható. Számunkra most a legfontosabb a súgónak az sora, amely a logaritmus függvény használatát mutatja:

`log(x, base=exp(1))`.

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

```
> exp(1)
[1] 2.718282
```

parancsból. Ezt az irracionális számot a matematikában 'e'-vel jelöljük, és a természetes alapú logaritmus alapjának nevezzük.

Vagyis, ha nem határozzuk meg a második paramétert, akkor a **log()** függvény, természetes alap estén számítja az **x** logaritmusát.

Ezek alapján 2 természetes alapú logaritmusát a

```
> log(2)
[1] 0.6931472
```

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

```
> log(x=2);
[1] 0.6931472
```

sor természetesen ugyanazt az eredményt szolgáltatja, csak most explicit módon közöltük, hogy az aktuális paraméterben szereplő `2`-es értéket az **x** nevű formális paraméternek feleltetjük meg. Ez most felesleges gépelést jelentett és általában is elmondhatjuk, hogy matematikai függvények esetében az oly gyakori **x** argumentumnevet szokás szerint nem használjuk az aktuális paraméterek elnevezésére.

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

```
> log(100, 10)
[1] 2
```

parancsból olvashatjuk ki. A függvényhívásnál az **x** formális argumentum a 100, a **base** pedig a 10 értéket kapja. Természetesen ezt a hívásnál mi is rögzíthetjük a világosabb értelmezés kedvéért sajáz magunk számára a

```
> log(100, base=10)
[1] 2
```

vagy akár

```
> log(x=100, base=10)
[1] 2
```

formában is.

Arra is lehetőség van, hogy ha az nem okoz félreértést az R számára, megcseréljük az aktuális paraméterek sorrendjét. A legbiztonságosabb ekkor az összes paraméter nevesítése

```
> log(base=10, x=100)
[1] 2
```

de két argumentum esetén így is egyértelmű a hozzárendelés:

```
> log(base=10, 100); log(10, x=100)
[1] 2
[1] 2
```

Ha az argumentumok nevesítése nélkül cseréljük fel az aktuális paramétereket, akkor természetesen hibás eredményt kapunk, mert az a `10` szám `100`-as alapú logaritmusa lesz.

```
> log(10, 100)
[1] 0.5
```

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

```
> log(b=10, 100)
[1] 2
```

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

```
> log()
Error: 0 arguments passed to 'log' which requires 1 or 2
```

A fenti hibaüzenet láthatjuk, ha egy függvényt hibás paraméterszámmal hívunk.

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

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

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

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

Mostanra rendelkezünk elég ismerettel, hogy a kifejezés fogalmát pontosíthassuk. Egy konstans, egy objektum vagy egy függvényhívás önmagában kifejezés, de ezek operátorokkal és kerek zárójelekkel összefűzött sorozata is kifejezés.

2.4. Adattípusok az R-ben

Eddig numerikus értékekkel végeztünk műveleteket, de tekintsük át az R többi adattípusát is. A legalapvetőbb numerikus adat mellet beszélünk *karakteres* és *logikai* adatokról is.

2.4.1. Karakteres adatok

Az R-ben egy karakterkonstans (vagy más néven sztring) idézőjelekkel határolt, tetszőleges karaktereket tartalmazó sorozat.

```
> "Helló"
[1] "Helló"

> 'Itt vagyok'
[1] "Itt vagyok"
```

Az karakterkonstansokat határoló idézőjel lehet egyszeres és dupla is, de egy konstanson belül nem keverhetjük őket. Az R a dupla idézőjelet részesíti előnyben.

Egy karakterkonstans tetszőleges karaktert (betűt, számjegyet, írásjeleket, szóközt, stb) tartalmazhat, egyedül azt az idézőjelet kell elkerülnünk, amelyet a konstans létrehozásánál használtuk.

A karakterkonstansok tartalmazhatnak ún. escape szekvenciákat, olyan backslash jellel ('\', fordított perjel) kezdődő karaktersorozatokat, amelyeket speciálisan kell értelmezni. A legfontosabb escape szekvenciák a következők:

Escape szekvencia	Jelentése
\'	' (szimpla idézőjel)
\"	" (dupla idézőjel)
\n	új sor karakter
\r	kocsi vissza karakter
\t	tabulator
\\	\ (backslash) karakter

A fentiek alapján a következő idézőjel-használatok engedélyezettek a karakterkonstansokban:

```
> '\'alma\''; "'alma'"; "\"alma\""; '"alma"'
[1] "'alma'"
[1] "'alma'"
[1] "\"alma\""
[1] "\"alma\""
```

Karakteres objektumokat is létrehozhatunk.

```
> neve<-'Zsolt'; foglalkozasa<-"festő"
```

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

Függvény	Leírás	Példa	Példa értéke
`paste(…,sep)`	sztringek összefűzése	`paste('a', 'b', sep='=')`	"a=b"
`nchar(x)`	karakterszrting hossza	`nchar('alma')`	4
`substr(x,start,stop)`	sztring egy része	`substr('alma', 3, 5)`	"ma"
`tolower(x)`	kisbetűsre konvertál	`tolower("Kiss Géza")`	"kiss géza"
`toupper(x)`	nagybetűsre konvertál	`toupper("Kiss Géza")`	"KISS GÉZA"
`chartr(old,new,x)`	karakterek cseréje	`chartr("it","ál","titik")`	"lálák"
`cat(…, sep)`	kiíratás	`cat('alma','fa\n',sep='')`	almafa
`grep(),regexpr(), gregexpr()`	részsztringek keresése		
`sub(), gsub()`	részsztringek cseréje		

2.4.2. Logikai adatok

Az eddigiekben megismert numerikus és karakteres konstansok nagyon sokféle érték megjelenítésére képesek, de ugyanígy a numerikus és karakteres objektumokhoz nagyon sok lehetséges numerikus és karakteres érték rendelhető. A logikai adattípus ezektől lényegesen egyszerűbb tárolási szerkezet, mivel összesen két értéket tárolására van módunk. Ezek a logikai igaz és a logikai hamis érték, amelyek az R nyelvben a `TRUE` és a `FALSE` logikai konstansokat jelentik. Az R a logikai konstansok írását a `T` és `F` globális változók bevezetésével segíti, ezek induló értéke a `TRUE` és `FALSE` logikai konstans.

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

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

Logikai értékeket vagy objektumokat relációs operátorok segítségével is létrehozhatunk.

Operátor formája	Művelet	Példa	Példa eredménye
<	kisebb	1<2 'alma'<'körte'	TRUE TRUE
>	nagyobb	3<(1+2) 'abc'>'ab'	FALSE TRUE
<=	kisebb egyenlő	1<=-.3 'él'<='elő'	FALSE TRUE
>=	nagyobb egyenlő	3/4>=7/9 'aki'>='Ági'	FALSE TRUE
==	egyenlő	20==2e1 'Len'=='len'	TRUE FALSE
!=	nem egyenlő	exp(1)!=pi 'Len'!='len'	TRUE TRUE

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

Az ilyen, logikai értékkel visszatérő kifejezéseket (egyszerű) *logikai kifejezéseknek* nevezzük. Ezekből az egyszerű logikai kifejezésekből a *logikai operátorok* segítségével összetett logikai kifejezéseket hozhatunk létre.

Operátor formája	Művelet	Példa	Példa eredménye
!	logikai NEM	!(1<2) !TRUE !FALSE	FALSE FALSE TRUE
& és &&	logikai ÉS	TRUE & TRUE TRUE & FALSE FALSE & TRUE FALSE & FALSE	TRUE FALSE FALSE FALSE

`| és ||`	logikai VAGY	`TRUE | TRUE`
		`TRUE | FALSE`
		`FALSE | TRUE`
		`FALSE | FALSE`
			`TRUE`
			`TRUE`
			`TRUE`
			`FALSE`

3. Adatszerkezetek

Az előző fejezetben láttuk, hogy a konstansok és az objektumok alapvetően 3 típusba sorolható értéket vehetnek fel. Ezek a numerikus, karakteres és logikai típusok voltak. Az eddigi példákban az objektumokhoz hozzárendelt érték „elemi" volt. Egy numerikus változó a hozzárendelt memóriaterületen pl. a `12` értéket tárolhatja, egy másik karakteres objektum a `"január"` értéket és egy logikai objektum pl. a `TRUE` értéket.

Az R azonban támogatja az összetettebb adatszerkezeteket is, sőt a fenti „elemi" esetek az ún. vektor adatszerkezet legegyszerűbb, speciális változatának tekinthetők. Az összetett adatszerkezetre úgy gondolhatunk, hogy az objektum neve nem egyetlen számra, karaktersorozatra vagy logikai értékre vonatkozik, hanem ezekből többre: pl. két számra, vagy három karaktersorozatra, vagy tíz logikai értékre, esetleg ezekre mind együtt.

Az R a következő adatszerkezeteket tartalmazza: vektor (vector), faktor (factor), mátrix (matrix), tömb (array), lista (list), adattábla (data frame), idősor (ts).

3.1 Vektorok

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

3.1.1. Vektorok létrehozása

Vektort legegyszerűbben a **c()** függvénnyel hozhatunk létre, a paraméterben sorban felsoroljuk a vektort alkotó értékeket. Numerikus vektort hozhatunk létre, ha a paraméterben numerikus konstansokat sorolunk fel:

```
> v1<-c(2,4,6,8)
> v1
[1] 2 4 6 8
```

A fenti példában a **v1** objektum egy 4 elemű vektor. Az első eleme a `2` numerikus konstans, a második eleme a `4`, a harmadik a `6` és a negyedik egyben utolsó eleme a `8`. A vektor elemei kiíratáskor szóközökkel elválasztva jelennek meg a konzolban.

Karakteres vektort hasonlóan hozhatunk létre:

```
> v2<-c("erős", 'közepes', "gyenge")
> v2
[1] "erős"    "közepes" "gyenge"
```

A fenti **v2** vektor 3 elemű.

Egy logikai vektor csak logikai konstansokat tartalmazhat (`TRUE` vagy `FALSE`):

```
> v3<-c(TRUE, FALSE, TRUE)
> v3
[1]  TRUE FALSE  TRUE
```

A **v1**, **v2** , **v3** objektum egy-egy példa az R különböző típusú vektoraira. Egy vektor típusát a **typeof()** függvénnyel kérdezhetjük le:

```
> typeof(v1); typeof(v2); typeof(v3)
[1] "double"
[1] "character"
[1] "logical"
```

A **v1** objektum 'double' típusa a numerikus típusok egyik változata (nem egész értékeket is tárolhatunk benne), a **v2** karakteres és a **v3** logikai típusa pedig a fenti példából könnyen kiolvasható.

Az objektumok fontos jellemzője az objektum hossza, ami vektorok esetén a vektort alkotó elemek számát jelenti. Ezt a **length()** függvénnyel kérdezhetjük le.

```
> length(v1); length(v2); length(v3)
[1] 4
[1] 3
[1] 3
```

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

```
> szamok<-c(1, (2+3)*4, 1/4, .5^3)
> szamok
[1]  1.000 20.000  0.250  0.125

> nevek<-c("Péter", paste('Zso', "lt", sep=""))
> nevek
[1] "Péter" "Zsolt"

> iteletek<-c(T, 1<2, 2==3)
> iteletek
[1]  TRUE  TRUE FALSE
```

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

```
> eset1<-c(2,4,"6",8)
> eset1
[1] "2" "4" "6" "8"

> eset2<-c(T, FALSE,"6")
> eset2
[1] "TRUE"  "FALSE" "6"

> eset3<-c(T, FALSE, 3)
> eset3
[1] 1 0 3
```

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

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

```
> elol<-c(1,2,3)
> hatul<-c(7,8,9)
> c(0,elol,4,5,6,hatul,10)
 [1]  0  1  2  3  4  5  6  7  8  9 10
```

A fenti példában létrehozott (majd el is vesző) 11 elemű vektor összerakásához felhasználtunk két 3 elemű vektort is.

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

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

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

> 10:1
 [1] 10  9  8  7  6  5  4  3  2  1

> -1.5:5
[1] -1.5 -0.5  0.5  1.5  2.5  3.5  4.5
```

Látható, hogy az így létrejövő sorozatok lehetnek csökkenő vagy növekvő rendezettségűek valamint tört értékeket is használhatunk operandusként. A sorozat nem feltétlenül a kettőspont utáni értékig tart, annyi igaz, hogy a `stop` értéknél mindig kisebb egyenlő.

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

```
> seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

A **seq()** függvény használatához 4 nevesített paraméter jelentését kell megtanulnunk: a `from` a sorozat első elemét határozza meg, a `to` az utolsó elemet, a `by` a lépésközt és a `length.out` a létrehozandó vektor elemeinek a számát. A négy paraméterből 3 megadása már egyértelműen azonosítja a kívánt vektort:

```
> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9

> seq(from=1, to=10, length.out=5)
[1]  1.00  3.25  5.50  7.75 10.00

> seq(to=10, by=1.3, length.out=5)
[1]  4.8  6.1  7.4  8.7 10.0

> seq(from=1, by=1.3, length.out=5)
[1] 1.0 2.3 3.6 4.9 6.2
```

A **sequence()** függvény a paraméterében szereplő értékig, mint végpontig 1-től kezdődő 1 lépésközű sorozatot hoz létre. A függvényt pozitív értékeket tartalmazó vektorral is hívhatjuk, ekkor több, a fenti szabálynak eleget tevő szabályos vektor sorozata lesz az eredményvektor:

```
> sequence(4)
[1] 1 2 3 4
```

```
> sequence(c(4,5))
[1] 1 2 3 4 1 2 3 4 5

> sequence(c(4,5,3))
 [1] 1 2 3 4 1 2 3 4 5 1 2 3
```

Tetszőleges típusú vektor létrehozására használhatjuk a **rep()** függvényt, amely egy létező vektor értékeit ismétli meg.

```
> rep(2, times=3)
[1] 2 2 2

> rep(c(2, 0, -2), times=3)
[1]  2  0 -2  2  0 -2  2  0 -2

> rep("tó", times=3)
[1] "tó" "tó" "tó"

> rep(c(F,T,T), times=3)
[1] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
```

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

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

```
> rep(2, each=3)
[1] 2 2 2

> rep(c(2, 0, -2), each=3)
[1]  2  2  2  0  0  0 -2 -2 -2

> rep("tó", each=3)
[1] "tó" "tó" "tó"

> rep(c(F,T,T), each=3)
[1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

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

Utolsó esetként vegyük azt az esetet, amikor elemenként szeretnénk ismételni, de mindegyiket esetleg eltérő mértékben. Ekkor az each paraméterben a bemenő vektor elemszámával azonos hosszú vektort kell megadni. Ez a vektor tartalmazza az egyes elemek ismétlési számát.

```
> rep(c(2,3,4), c(1,2,3))
[1] 2 3 3 4 4 4

> rep(c("tó","part"), c(2,3))
[1] "tó"   "tó"   "part" "part" "part"

> rep(c(T,F,T), c(2,3,4))
[1]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

3.1.2. Műveletek vektorokkal

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

```
> c(1,2)+c(3,4)
[1] 4 6
```

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

Két vektor összeadásánál természetesen használhatunk objektumneveket is:

```
> x<-1:3; y<-2:4
> x+y
[1] 3 5 7
```

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

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

```
> c(1,2)-c(2,3)
[1] -1 -1

> x<-1:3; y<-2:4
> x-y; x*y; x/y; x^y
[1] -1 -1 -1
[1]  2  6 12
[1] 0.5000000 0.6666667 0.7500000
[1]  1  8 81
```

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

A komponensenkénti végrehajtás szabálya logikai operátorokra is érvényes:

```
> !c(T,T,F,F)
[1] FALSE FALSE  TRUE  TRUE

> c(T,T,F,F) & c(T,F,T,F)
[1]  TRUE FALSE FALSE FALSE

> c(T,T,F,F) | c(T,F,T,F)
[1]  TRUE  TRUE  TRUE FALSE
```

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

```
> sqrt(c(4,9,16))
[1] 2 3 4

> log(1:3)
[1] 0.0000000 0.6931472 1.0986123
```

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

```
> c(1,2)+5
[1] 6 7
```

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

```
> c(1,2)+c(3,4,5)
[1] 4 6 6
Warning message:
In c(1, 2) + c(3, 4, 5) :
  longer object length is not a multiple of shorter object length
```

Itt egy 2 elemű és egy 3 elemű vektort adunk össze. A rövidebbik vektort még egyszer megismételve nem használjuk fel minden elemét, így a figyelmezető üzenetet megkapjuk. Az eredményvektor az 1+3, 2+4 és a 1+5 összeadások eredménye. A következő példában már nincs figyelmeztetés:

```
> c(1,2)+c(3,4,5,6)
[1] 4 6 6 8
```

3.1.3. Függvények vektorokkal

Az R egyik legnagyobb erőssége, hogy vektorok elemeivel ciklusok magadása nélkül különböző műveleteket végezhetünk. A legfontosabb vektor alapú függvényeket a következő táblázat tartalmazza:

Függvény	Leírás	Példa	Példa értéke
`max(x)`	az x vektor legnagyobb eleme	`max(1:10)`	`10`
`min(x)`	az x vektor legkisebb eleme	`min(11:20)`	`11`
`sum(x)`	x elemeinek összege	`sum(1:5)`	`15`
`prod(x)`	x elemeinek szorzata	`prod(1:5)`	`120`
`mean(x)`	x számtani közepe (mintaátlag)	`mean(1:10)`	`5.5`
`median(x)`	x mediánja	`median(1:10)`	`5.5`

`range(x)`	x legkisebb és legnagyobb eleme	`range(1:10)`	`1 10`
`sd(x)`	az x tapasztalati szórása	`sd(1:10)`	`3.027650`
`var(x)`	az x tapasztalati varianciája	`var(1:10)`	`9.166667`
`cor(x,y)`	korreláció x és y között	`cor(1:10,11:20)`	`1`

3.1.4. Az NA hiányzó érték

Gyakorlati esetekben sokszor előfordul, hogy a vektor adott értékét nem ismerjük, de mégis jelentősége miatt jelölnünk kell. Az R-ben az `NA` értékkel jelölhetjük a hiányzó adatot, amely tetszőleges típusú vektor esetében használható:

```
> x<-c(2,NA,4); x
[1]  2 NA  4

> x<-c("erős", NA, "gyenge"); x
[1] "erős"   NA       "gyenge"

> x<-c(T, NA, FALSE); x
[1]  TRUE    NA FALSE
```

Az `NA` érték tesztelésére az **is.na()** függvényt használhatjuk, amelynek a visszatérési értékében lévő vektor ott tartalmaz `TRUE` értéket, ahol hiányzó adatot találunk.

```
> is.na(c(1, NA))
[1] FALSE  TRUE
```

Hiányzó értékeket is tartalmazó vektor esetén néhány függvény meglepő eredményt adhat, pl:

```
> mean(c(1:10,NA))
[1] NA
```

Ha kíváncsiak vagyunk az `NA` értéken kívüli elemek átlagára, akkor egy második paramétert is szerepeltetnünk kell a **mean()** függvényben:

```
> mean(c(1:10,NA), na.rm=T)
[1] 5.5
```

Az **na.rm** argumentum `TRUE` értéke biztosítja, hogy az átlag számítása során a hiányzó értékeket figyelmen kívül hagyjuk.

3.1.5. Az Inf és a NaN

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

```
> 1/0
[1] Inf

> log(0)
[1] -Inf
```

```
> exp(Inf)
[1] Inf

> mean(c(1,2,Inf))
[1] Inf
```

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

```
> 0/0
[1] NaN

> Inf-Inf
[1] NaN

> Inf/Inf
[1] NaN
```

Egy kifejezés véges vagy végtelen voltát az **is.finite()** vagy **is.infinite()** függvényekkel tesztelhetjük. A `NaN` értékre az **is.nan()** függvénnyel kérdezhetünk rá. Érdekes megfigyelni, hogy a `NaN` értékre mind az **is.nan()** mind az **is.na()** függvény igazat ad:

```
> is.na(NaN); is.nan(NaN)
[1] TRUE
[1] TRUE

> is.na(NA); is.nan(NA)
[1] TRUE
[1] FALSE
```

3.1.6. Objektumok attribútumai

Az R-ben használható objektumok, pl. az eddig vizsgált vektorok, két alapvető attribútummal rendelkeznek. Ez a *mód* (mode) és a *hossz* (length), melyek az objektum tárolási szerkezetéről és az elemeinek a számáról adnak tájékoztatást. Ezeket az attribútumokat a **mode()** és a **length()** függvények segítségével kérdezhetjük le, ill. állíthatjuk be:

```
> x<--1:1; x; mode(x); length(x)
[1] -1  0  1
[1] "numeric"
[1] 3

> mode(x)<-"logical"; length(x)<-5; x
[1]  TRUE FALSE  TRUE    NA    NA
```

Más attribútumokat is rendelhetünk objektumokhoz, melyek speciális jelentéssel bírnak. A **names** attribútummal például a vektor egyes értékeit nevezhetjük el. (Későbbiekben látjuk a **dim**, **dimnames**, **row.names**, **level**, **class**, **tsp** attribútumok jelentőségét is.)

A **names** attribútum lekérdezhető és beállítható a **names()** függvénnyel:

```
> x<-1:5
> names(x)<-c("elégtelen", "elégséges", "közepes", "jó", "jeles")
> x
elégtelen elégséges   közepes        jó     jeles
        1         2         3         4         5

> names(x)
```

```
[1] "elégtelen" "elégséges" "közepes"   "jó"        "jeles"
```

A **names** attribútum egy karakteres vektor, a hozzárendelés után pl. az R egy kiíratásban is felhasználja ezeket a címkéket. További lehetőségek az attribútumok meghatározására az **attributes()** és az **attr()** függvénnyek. Az

```
> attributes(x)
$names
[1] "elégtelen" "elégséges" "közepes"   "jó"        "jeles"
```

parancs az összes attribútumot (az elsődlegeseket nem) kiírja a képernyőre, de ezt a függvényt használva tudjuk az összes attribútumot törölni is (az elsődlegeseket nem törölhetjük):

```
> attributes(x)<-NULL
> attributes(x)
NULL
```

Az **attr()** függvényben meg kell adnunk az elérendő attribútum nevét is:

```
> attr(x,"names")<-c("elégtelen", "elégséges", "közepes", "jó", "jeles")
> attr(x,"names")
[1] "elégtelen" "elégséges" "közepes"   "jó"        "jeles"
```

3.1.7. Vektorok indexelése

A vektorok révén egyetlen változónév segítségével tetszőleges számú konstans értékre hivatkozhattunk, így például nagyon egyszerűen mindegyik elemhez hozzáadhattunk 1-et:

```
> x<-1:10
> x+1
 [1]  2  3  4  5  6  7  8  9 10 11
```

Sokszor van szükség azonban arra is, hogy a vektor egyes elemeit külön tudjuk elérni, lekérdezni vagy módosítani. A vektor egy tetszőleges részét, egy vagy több elemét az *indexelés* művelettel érhetjük el. Az index operátor a szögletes zárójel (`[]`), amit a vektor neve után kell írnunk. Az index operátorban numerikus, karakteres és logikai vektorok is szerepelhetnek. Nézzük ezeket sorban.

Ha létrehozunk egy 10 elemű **x** vektort a

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

paranccsal, akkor megfigyelhetjük, hogy az **x** vektor első eleme `11`, a második `12`, az utolsó, a tizedik pedig éppen `20`. Ebben a felsorolásban az elemek sorszámai (első, második, tizedik) pontosan a vektor indexeit jelentik. A vektor indexelése tehát 1-el kezdődik, ez az első elem indexe, a második elem indexe 2, az utolsó elemé pedig 10.

Ha az index operátorba egy ilyen egyszerű sorszámot írunk, akkor a vektor adott indexű elemét érhetjük el:

```
> x[1]
[1] 11

> x[2]
[1] 12
```

```
> x[10]
[1] 20
```

De nem csak lekérdezhetjük, hanem az értékadó operátor segítségével felül is írhatjuk valamelyik elem értékét:

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

Itt először a második elemet `100`-ra cseréljük, majd a harmadikat a második kétszeresére. A változást látjuk a képernyőn. Ha az **x** vektort az elemszámánál nagyobb indexszel próbáljuk elérni, akkor `NA` értéket kapunk:

```
> x[11]
[1] NA
```

Ha negatív skalár értékkel indexelünk, akkor a negatív előjellel megadott sorszámon kívül az összes többi elemet elérhetjük:

```
> x<-11:20; x[-3]
[1] 11 12 14 15 16 17 18 19 20

 > x[-5]<-0; x
 [1]  0  0  0  0 15  0  0  0  0  0
```

Vektorokat azonban nem csak numerikus skalárral, hanem két vagy több elemű numerikus vektorokkal is indexelhetünk. Ebben az esetben az indexben felsorolt sorszámoknak megfelelő indexű elemeket érhetjük el:

```
> x<-11:20
> x[c(1,3,5)]; x[3:6]
[1] 11 13 15
[1] 13 14 15 16

> y<-c(3,7); x[y]<-c(100,200); x
 [1]  11  12 100  14  15  16 200  18  19  20
```

Természetesen indexeléskor negatív értékeket tartalmazó numerikus vektorokat is használhatunk:

```
> x<-11:20; x[-seq(from=0, to=10, by=3)]; x[-c(1,10)]; x[-(1:5)]
[1] 11 12 14 15 17 18 20
[1] 12 13 14 15 16 17 18 19
[1] 16 17 18 19 20
```

Egy vektor indexe mindig egész szám, de az R megengedi, hogy tört értékeket szerepeltessünk az index operátorban, ekkor az egész részét veszi az indexeknek (csonkolja őket):

```
> x<-11:20; x[2.3]; x[2.8]; x[-2.3]; x[-2.8]
[1] 12
[1] 12
[1] 11 13 14 15 16 17 18 19 20
[1] 11 13 14 15 16 17 18 19 20
```

Egy **name** attribútummal is rendelkező vektort indexelhetünk karakteres vektorral is:

```
> x<-1:5
> names(x)<-c("elégtelen", "elégséges", "közepes", "jó", "jeles")
> x["közepes"]; x[c("közepes","jó")]
közepes
      3
közepes      jó
      3       4
```

Ha tekintünk egy másik példát,

```
> x<-c(18,12,20); names(x)<-0:2; x
 0  1  2
18 12 20
```

ahol a (0, 1, 2) értékek előfordulási gyakoriságait a (18, 12, 20) elemeket tartalmazó vektorban rögzítjük. Az elemek nevei most is karakteres konstansok, az automatikus konverzióról az R gondoskodik:

```
> names(x)
[1] "0" "1" "2"
```

Az **x** vektor indexelésénél fontos, hogy megkülönböztessük a numerikus és a karakteres indexeket, az utóbbiaknál mindig idézőjelet kell használnunk:

```
> x[1]; x["1"]
 0
18
 1
12

> x[c(1,3)]; x[c("0", "2")]
 0  2
18 20
 0  2
18 20
```

Az R-ben a vektorokat logikai vektorokkal is indexelhetjük, a `TRUE` logikai értékkel jelezzük, hogy az adott sorszámú elemet el akarjuk érni:

```
> x<-11:15; x[c(TRUE, FALSE, T, T, F)]
[1] 11 13 14
```

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

Az indexelésre használt logikai vektor elemszáma kisebb is lehet, mint az indexelt vektor hossza, ekkor az R az indexvektor ismétlését használja:

```
> x<-11:15; x[c(T,F)]; x[T]; x[F]
[1] 11 13 15
[1] 11 12 13 14 15
integer(0)
```

Ha a csupa `TRUE` értékű vektorral indexelünk, akkor az **x** vektor összes elemét megkapjuk, ha pedig a csupa `FALSE` értékkel, akkor az üres vektort kapjuk, az `integer(0)` az üres, egész értékeket „tartalmazó” vektort jelöli.

A logikai index-értékek lehetővé teszik a vektor szűrését is, bizonyos feltételeknek eleget tevő értékek leválogatását. Tekintsük ehhez az **x** vektort:

```
> x<-c(4,7,9,2,8)
> which(x<5)
[1] 1 4
```

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

```
> x<5
[1]  TRUE FALSE FALSE  TRUE FALSE
```

logikai vektor az 1. és 4. pozícióban tartalmaz logikai igaz értéket. Ha tehát egy adott feltételnek eleget tevő vektorelemek indexére vagyunk kíváncsiak, a **which()** függvényt használhatjuk. Ha a feltételnek eleget tevő vektor elemeit is el akarjuk érni, akkor vagy a **which()** által szolgáltatott numerikus értékekkel,

```
> x[which(x<5)]
[1] 4 2
```

vagy sokkal elegánsabb módon közvetlenül a logikai vektor értékeivel indexelünk:

```
> x[x<5]
[1] 4 2
```

Természetesen összetett logikai kifejezésekkel is indexelhetünk pl.:

```
> which(3<=x & x<=7)
[1] 1 2

> x[3<=x & x<=7]
[1] 4 7
```

A vektorok indexelése során az indexoperátor üresen is maradhat, ekkor a vektor összes elemét érhetjük el, továbbá az `NA`, `NaN` és `NULL` értékekkel is indexelhetünk:

```
> x<-11:15; x[]; x[NA]; x[NaN]; x[NULL]
[1] 11 12 13 14 15
[1] NA NA NA NA NA
[1] NA
integer(0)
```

3.1.8. Vektorok rendezése

Sokszor szükség van egy vektor elemeit növekvő vagy csökkenő sorrendben látni. Az R-ben a vektor elemeit a **sort()** függvénnyel rendezhetjük:

```
> x<-c(1:5,5:3); x
[1] 1 2 3 4 5 5 4 3

> sort(x); sort(x, decreasing=T); rev(sort(x))
[1] 1 2 3 3 4 4 5 5
```

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

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

Ha a **sort()** függvénnyel átrendezett vektort a továbbiakban fel szeretnénk használni, akkor azt egy újabb objektumban tároljuk. Rossz gyakorlat, ha felülírjuk a kiinduló vektorunkat.

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

```
> x<-c(1:5,5:3); order(x)
[1] 1 2 3 8 4 7 5 6

> x[order(x)]; x[order(x, decreasing=T)];
[1] 1 2 3 3 4 4 5 5
[1] 5 5 4 4 3 3 2 1
```

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

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

3.1.9. Előre definiált objektumok, nevesített konstansok

A R-ben a következő globális változókat használhatjuk, amelyek a **base** csomagban vektorokként kerültek megvalósításra:

```
> pi; T; F
[1] 3.141593
[1] TRUE
[1] FALSE

> LETTERS; letters
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
[14] "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
[14] "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

> month.abb; month.name
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug"
 [9] "Sep" "Oct" "Nov" "Dec"
 [1] "January"   "February"  "March"     "April"
 [5] "May"       "June"      "July"      "August"
 [9] "September" "October"   "November"  "December"
```

3.2. Faktorok

A faktor a vektorhoz nagyon hasonló, homogén, egydimenziós adatszerkezet, amelyet elsősorban kategorikus változók értékeinek tárolására használunk. Faktorok esetében csak numerikus és karakteres adattípusokat használhatunk.

Numerikus vagy karakteres vektorból a **factor()** függvény segítségével hozhatunk létre faktort. A faktor az alapértelmezett attribútumokon kívül egy **levels** attribútumot is tartalmaz, amely a faktor különböző értékeit (szintjeit) sorolja fel. A faktorok **class** attribútumának értéke pedig `factor`.

```
> x<-c(rep("A",3), rep("B",4), rep("C",3)); x
 [1] "A" "A" "A" "B" "B" "B" "B" "C" "C" "C"

> xf<-factor(x); xf
 [1] A A A B B B B C C C
Levels: A B C

> attributes(xf)
$levels
[1] "A" "B" "C"

$class
[1] "factor"
```

A fenti példában három lehetséges értéket tartalmazó faktort hoztunk létre, amelyek a **levels()** függvénnyel lekérdezhetők és átírhatók:

```
> levels(xf)
[1] "A" "B" "C"

> levels(xf)<-c(0:2); xf
 [1] 0 0 0 1 1 1 1 2 2 2
Levels: 0 1 2

> levels(xf)
[1] "0" "1" "2"
```

A szintek meghatározásakor a faktorban eddig nem szereplő értékeket is megadhatjuk:

```
> levels(xf)<-c("A","B","C","D"); xf
 [1] A A A B B B B C C C
Levels: A B C D
```

A szintek számáról és azok címkéjéről a **factor()** függvényben is gondoskodhatunk:

```
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels: 1 2 3 4 5

> factor(1:3, levels=1:5, labels="L")
[1] L1 L2 L3
Levels: L1 L2 L3 L4 L5

> factor(1:3, levels=1:5, labels=c("A","B","C","D","E"))
[1] A B C
Levels: A B C D E
```

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

```
> factor(c(1:5, NA, 3:6))
 [1] 1    2    3    4    5    <NA> 3    4    5    6
Levels: 1 2 3 4 5 6
```

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

```
> factor(c(1:5, NA, 3:6), exclude=NULL)
 [1] 1    2    3    4    5    <NA> 3    4    5    6
Levels: 1 2 3 4 5 6 <NA>

> factor(c(1:5, NA, 3:6), exclude=c(4, NA))
 [1] 1    2    3    <NA> 5    <NA> 3    <NA> 5    6
Levels: 1 2 3 5 6
```

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

Faktorokat a **gl()** függvénnyel is létrehozhatunk (’generate levels’), ahol a szintek számát és az ismétlések számát kell megadnunk.

```
> gl(3,2)
[1] 1 1 2 2 3 3
Levels: 1 2 3
```

További paraméterként a faktor hosszát és címkéit is meghatározhatjuk:

```
> gl(3,2,8)
[1] 1 1 2 2 3 3 1 1
Levels: 1 2 3

> gl(3,2,8,labels="F")
[1] F1 F1 F2 F2 F3 F3 F1 F1
Levels: F1 F2 F3

> gl(3,2,8,labels=c("gyenge", "közepes", "erős"))
[1] gyenge  gyenge  közepes közepes erős   erős   gyenge  gyenge
Levels: gyenge közepes erős
```

3.3. Mátrixok és tömbök

Az egydimenziós vektor többdimenziós megfelelője a *tömb* (array). A tömb a vektorhoz hasonlóan homogén adatszerkezet, amely az alapvető attribútumokon túl a **dim** attribútummal is rendelkezik. Egy vektort könnyen átalakíthatunk pl. egy 3 dimenziós tömbbé a **dim()** függvény segítségével:

```
> x<-1:8; is.vector(x)
[1] TRUE

> dim(x)<-c(2,2,2); is.vector(x); is.array(x)
[1] FALSE
[1] TRUE

> x
, , 1

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

, , 2

     [,1] [,2]
```

```
[1,]    5    7
[2,]    6    8
```

Az **x** vektorból egy háromdimenziós tömböt hoztunk létre. Az **is.vector()** és az **is.array()** függvények eligazítanak az objektum adatszerkezetével kapcsolatban, vektor ill. tömb paraméter esetén logikai igaz értéket adnak. A tömb kiíratása során az indexoperátorokban szereplő sorszámok segítségével igazodhatunk el az elemek között. A háromdimenziós **x** tömb dimenziói a sorok, oszlopok és a lapok. A 8 elemet két lapon a ’ , , 1’ és a ’ , , 2’ nevű lapokon két-két sorba ’[1,]’, ’[2,]’ és két-két oszlopba ’[,1]’, ’[,2]’ rendezve sorolja fel az R. A második lapon a 2. sor 1. eleméhez meg kell találnunk a ’ , , 2’ lapot, a ’[2,]’ sort és a ’[,1]’ oszlopot, ami esetünkben a 6.

Tömböket az **array()** függvénnyel is létrehozhatunk:

```
> x<-array(data=1:20, dim=c(4,5)); x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

A fenti **x** tömb dimenzióinak száma 2, az ilyen tömböket *mátrix*nak is nevezhetjük. Mátrixok létrehozására használhatjuk a **matrix()** függvényt is:

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

Itt az **nrow** paraméter segítségével irányítjuk az R-et, hogy a sorok és az oszlopok számát meghatározhassa. A **matrix()** függvényben az **ncol** paraméter is használható. Láthatjuk, hogy a 20 elemű vektorból az oszlopok mentén hoztuk létre a mátrixot. Ha sorfolytonosan szeretnénk a bemenő vektor elemeiből mátrixot képezni, akkor a **byrow** paramétert igazra kell állítanunk:

```
> matrix(1:12,ncol=4,byrow=T)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Mátrixok létrehozásához tehát egy adott elemszámú vektor szükséges, de előfordul, hogy a vektor elemeit ismételni kell:

```
> matrix(3:5, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    3    5    4
[2,]    4    3    5

> matrix(0, nrow=2, ncol=3)
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

Mátrixot és tömböt karakteres vagy logikai értékekből is építhetünk:

```
> matrix(c("a","b"), nrow=2, ncol=3, byrow=T)
     [,1] [,2] [,3]
[1,] "a"  "b"  "a"
[2,] "b"  "a"  "b"

> matrix(c(T,F,T), nrow=2, ncol=3, byrow=T)
     [,1]  [,2] [,3]
[1,] TRUE FALSE TRUE
[2,] TRUE FALSE TRUE
```

A tömbök indexelése nagyon hasonló a vektorok indexelésére, itt is a szögletes zárójel (`[]`) operátort kell használnunk a tömb egyes elemeinek elérésére. Az egyetlen különbség, hogy mivel itt a dimenziók száma nagyobb mint egy, az egyes dimenzióknak megfelelően, több indexeket kell megadnunk és ezeket vesszővel választjuk el az indexoperátoron belül. Tehát ha **x** például 3 dimenziós, akkor az **x[1,3,2]** egy lehetséges példa indexelésére, ahol az első sor, harmadik oszlopában lévő elemre gondolunk, a második lapról. A kétdimenziós mátrixok esetén csak a sor és oszlop azonosító indexre van szükségünk (pl. **x[2,3]**), 4 vagy afeletti dimenziószámok esetén természetesen 4 vagy több vesszővel elválasztott indexre.

Az egyes dimenziópozíciókban szereplő indexekre ugyanazok a szabályok érvényesek, mint a vektor esetén. Használhatunk pozitív vagy negatív numerikus skalárokat és vektorokat, de a karakteres és logikai vektorokkal való indexelés is megengedett. Ha egy dimenziópozíciót üresen hagyunk, az továbbra is az összes elemet jelenti abból a dimenzióból:

```
> x<-matrix(1:10,nrow=2, ncol=5,byrow=T); x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10

> x[2,3]
[1] 8

> x[2,c(1,4)]
[1] 6 9

> x[,c(1,4)]
     [,1] [,2]
[1,]    1    4
[2,]    6    9

> x[,-c(1,4)]
     [,1] [,2] [,3]
[1,]    2    3    5
[2,]    7    8   10
```

A mátrix indexelés során a kapott elemek elveszthetik a 2 dimenziójukat és egyszerű vektor lehet az eredmény. Ha ezt el akarjuk kerülni, használjuk a **drop** paramétert hamis értékkel az indexben:

```
> x[2,3,drop=F]
     [,1]
[1,]    8

>  x[2,c(1,4), drop=F]
     [,1] [,2]
[1,]    6    9

>  x[2,, drop=F]
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    6    7    8    9   10

>  x[,3, drop=F]
     [,1]
[1,]    3
[2,]    8
```

A mátrix sorait és oszlopait elnevezhetjük a **rownames()** és a **colnames()** függvényekkel, amelyek mint az látható, a **dimnames** attribútumot módosítják:

```
> rownames(x)<-c("eset1", "eset2")
> colnames(x)<-paste("sz.", 1:5, sep="")
> x
      sz.1 sz.2 sz.3 sz.4 sz.5
eset1    1    2    3    4    5
eset2    6    7    8    9   10

> attributes(x)
$dim
[1] 2 5

$dimnames
$dimnames[[1]]
[1] "eset1" "eset2"

$dimnames[[2]]
[1] "sz.1" "sz.2" "sz.3" "sz.4" "sz.5"
```

Amennyiben egy mátrixnak (vagy egy tetszőleges tömbnek) az egyes dimenzióit értékeit elnevezzük, akkor az R megjelenítéskor is használja őket, sőt az indexelés során is felhasználhatjuk:

```
> x["eset1","sz.2"]
[1] 2

> x["eset1",2]
[1] 2

> x["eset2",]
 sz.1 sz.2 sz.3 sz.4 sz.5
   6    7    8    9   10

> x["eset1", c(T,F)]
sz.1 sz.3 sz.5
   1    3    5
```

Mátrixok (és tömbök) esetén megengedett a mátrixszal történő indexelés is:

```
> x<-matrix(1:9, nrow=3); x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> index.m<-matrix(c(1:3,3:1), nrow=3); index.m
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
```

```
> x[index.m]<-0
> x
     [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    0    8
[3,]    0    6    9
```

3.3.1 Számítások a mátrix soraiban és oszlopaiban

Ha üresen hagyjuk a mátrix sor vagy oszlop pozícióját az indexelés során, akkor a mátrix teljes oszlopára vagy sorára tudunk hivatkozni, majd ezekkel, mint vektorokkal műveleteket hajthatunk végre:

```
> x<-matrix(1:10,nrow=2, ncol=5,byrow=T); x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10

> mean(x[1,]); var(x[,4])
[1] 3
[1] 12.5
```

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

```
> rowSums(x); rowMeans(x)
[1] 15 40
[1] 3 8

> colSums(x); colMeans(x)
[1]  7  9 11 13 15
[1] 3.5 4.5 5.5 6.5 7.5
```

Általánosabb megoldás, ha az **apply()** függvényt használjuk, amelyben a mátrix soraira vagy oszlopaira vonatkozó függvényt mi határozzuk meg. Az **apply()** első paramétere a mátrix, a második helyen pedig `1` vagy `2` áll, attól függően, hogy a mátrix soraira vagy oszlopaira akarjuk a harmadik paraméterben szereplő függvényt alkalmazni.

```
> apply(x, 1, mean); apply(x, 1, var); apply(x, 1, min)
[1] 3 8
[1] 2.5 2.5
[1] 1 6

> apply(x, 2, mean); apply(x, 2, var); apply(x, 2, min)
[1] 3.5 4.5 5.5 6.5 7.5
[1] 12.5 12.5 12.5 12.5 12.5
[1] 1 2 3 4 5
```

Lehetőség van a mátrix oszlopait úgy összegezni, hogy az egyes sorokat csoportokba soroljuk és az összegzést a csoportokon belül hajtjuk végre. Ha például az

```
> x<-matrix(1:12,nrow=4, ncol=3); x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

mátrix első két sora ill. a második két sora képez egy-egy csoportot, akkor ezt a sorok számával megegyező elemszámú vektor létrehozásával jelezhetjük a következő módon:

```
> csoportok<-c("A","A","B","B")
```

Magát az oszlopok összegzését az egyes csoportokon a **rowsum()** függvény végzi:

```
> rowsum(x, csoportok)
  [,1] [,2] [,3]
A    3   11   19
B    7   15   23
```

Amennyiben nem összegzés, hanem más művelet végrehajtása a cél kijelölt csoportokon, akkor a **tapply()** függvényt használhatjuk. A csoportok meghatározása itt a bemenő **x** mátrixszal egyező elemszámú faktorok listáján alapul, amit a második paraméterben kell meghatároznunk. A csoportok meghatározásánál a **row()** és **col()** függvényeket is használjuk, ezek az **x** mátrix szerkezetét megtartva minden értékben a sor ill. oszlopsorszámot tartalmazzák:

```
> row(x)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2
[3,]    3    3    3
[4,]    4    4    4

> col(x)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
[3,]    1    2    3
[4,]    1    2    3
```

Ezek alapján rögtön alternatív lehetőséget látunk sorokra és oszlopokra történő műveletvégzésre, hisz a

```
> tapply(x, row(x), max)
 1  2  3  4
 9 10 11 12
```

minden sor maximumát, a

```
> tapply(x, col(x), max)
 1  2  3
 4  8 12
```

pedig minden oszlop maximumát szolgáltatja. Ahhoz azonban, hogy az eredeti célunkat elérjük, nevezetesen, az oszlopok összegét vegyük, úgy, hogy az első két sor és a második két sor külön csoportba tartozik, még a **tapply()** második paraméterében újabb csoportosító vektort kell megadnunk. Ezt a vektort a **list()** függvénnyel fűzzük a **col(x)** csoportosító vektor elé, hisz ez a sorokra fog vonatkozni. A **tapply()** második paramétere tehát csoportosító vektorokat tartalmazó lista, amelynek az elemeit faktorokra konvertálja az R, mielőtt felhasználja őket.

```
> cs.matrix<-matrix(c("A","A","B","B"), nrow=4, ncol=3); cs.matrix
     [,1] [,2] [,3]
```

```
[1,] "A"  "A"  "A"
[2,] "A"  "A"  "A"
[3,] "B"  "B"  "B"
[4,] "B"  "B"  "B"

> tapply(x, list(cs.matrix,col(x)), sum)
  1  2  3
A 3 11 19
B 7 15 23
```

A fenti példában a sorok csoportosítása miatt hoztuk létre a **cs.matrix** mátrixot. Egyszerűbb azonban ha helyette a **csoportok** vektort használjuk fel, méghozzá egy mátrixszal indexelt alakját:

```
> csoportok[row(x)]
 [1] "A" "A" "B" "B" "A" "A" "B" "B" "A" "A" "B" "B"
```

Ekkor a **row(x)** mátrixot az alapértelmezett oszlopfolytonos módon véve, éppen a kívánt csoportosító vektort kapjuk, így írhatjuk:

```
> tapply(x, list(csoportok[row(x)],col(x)), sum)
  1  2  3
A 3 11 19
B 7 15 23
```

A **sum()** függvény helyett most már tetszőlegeset választhatunk ebben az általános alakban:

```
> tapply(x, list(csoportok[row(x)],col(x)), mean)
    1   2    3
A 1.5 5.5  9.5
B 3.5 7.5 11.5
```

Hasonló eredményt kapunk az **aggregate()** függvény használatával is:

```
> aggregate(x,list(csoportok),sum)
  Group.1 V1 V2 V3
1       A  3 11 19
2       B  7 15 23
```

Érdekes lehetőség az oszlopokban lévő elemek véletlenszerű átrendezése a **sample()** függvény segítségével. A **sample()** függvény a bemenő vektor elemeiből egy véletlen mintát állít elő, alapesetben a bemenet egy permutációját adja:

```
> sample(1:5)
[1] 1 2 5 3 4
```

Beállíthatjuk az eredményvektor elemszámát is a második paraméterben:

```
> sample(1:100,10)
 [1] 77 13 68 47 36 99 90 29 16 38
```

Ha tehát az oszlopok értékeit egymástól függetlenül fel akarjuk cserélni, írhatjuk a következőt:

```
> apply(x,2,sample)
     [,1] [,2] [,3]
[1,]    2    6   11
[2,]    1    5   12
```

```
[3,]    3    8    9
[4,]    4    7   10
```

3.3.2 Sorok és oszlopok kezelése

Létező mátrixot újabb sorokkal és oszlopokkal egészíthetünk ki az **rbind()** és a **cbind()** függvényekkel, de vektorokból is építhetünk segítségükkel mátrixot.

```
> cbind(1,1:2,1:4)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    2    2
[3,]    1    1    3
[4,]    1    2    4

> rbind(1,1:2,1:4)
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    1    2    1    2
[3,]    1    2    3    4
```

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

Új oszloppal vagy új sorral is kiegészíthetjük a mátrixunkat:

```
>  x<-matrix(1:12,nrow=4, ncol=3); x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> cbind(-3:0,x,13:16)
     [,1] [,2] [,3] [,4] [,5]
[1,]   -3    1    5    9   13
[2,]   -2    2    6   10   14
[3,]   -1    3    7   11   15
[4,]    0    4    8   12   16

> rbind(-1,x,1)
     [,1] [,2] [,3]
[1,]   -1   -1   -1
[2,]    1    5    9
[3,]    2    6   10
[4,]    3    7   11
[5,]    4    8   12
[6,]    1    1    1
```

Tetszőleges pozícióba beszúrhatunk egy oszlopot vagy egy sort:

```
> cbind(x,13:16)[,c(1,2,4,3)]
     [,1] [,2] [,3] [,4]
[1,]    1    5   13    9
[2,]    2    6   14   10
[3,]    3    7   15   11
[4,]    4    8   16   12

> rbind(x,-1)[c(1,2,3,5,4),]
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]   -1   -1   -1
[5,]    4    8   12
```

Hasznos lehetőség összesítő sorok vagy oszlopok mátrixhoz fűzése és elnevezése:

```
> x<-rbind(x,apply(x,2,mean))
> rownames(x)<-c(1:4,"átlag")
> x
      [,1] [,2] [,3]
1      1.0  5.0  9.0
2      2.0  6.0 10.0
3      3.0  7.0 11.0
4      4.0  8.0 12.0
átlag  2.5  6.5 10.5
```

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

```
> x<-matrix(1:12,nrow=4, ncol=3); x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

> cbind(x[,c(2,3,1)])     # oszlopcsere
     [,1] [,2] [,3]
[1,]    5    9    1
[2,]    6   10    2
[3,]    7   11    3
[4,]    8   12    4

> rbind(x[c(3,2,4,1),])   # sorcsere
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    2    6   10
[3,]    4    8   12
[4,]    1    5    9

> cbind(x[,c(1,3)])       # a 2. oszlop törlése
     [,1] [,2]
[1,]    1    9
[2,]    2   10
[3,]    3   11
[4,]    4   12

> rbind(x[c(1,3),])       # az 2. és a 4. sor törlése
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    3    7   11
```

3.5 Listák

Az eddig megismert vektor, faktor, mátrix és tömb adatszerkezet mindegyike homogén, csak azonos típusú értékeket tárolhatunk el bennük. A *lista* adatszerkezetben egymás után többfajta

adatot is felsorolhatunk, sem azok típusára sem azok méretére nincs megszorítás. A **list()** függvénnyel hozhatunk létre legegyszerűbben listákat, vesszővel elválasztva kell megadnunk a lista elemeit:

```
> x<-list(1:10, c("A","B"), c=T); x
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "A" "B"

$c
[1] TRUE
```

A fenti példában **x** egy 3 elemű lista, az első eleme egy 10 elemű numerikus vektor, a második eleme egy 2 elemű karakteres vektor, a harmadik eleme pedig egy 1 elemű logikai vektor. A harmadik elemnek 'c' nevet adtunk, ezt mindegyik listaelem esetén megtehettük volna. Ha a lista értékét megjelenítjük a képernyőn, akkor a listaelemek egymás alatt jelennek meg. Az első két esetben a kettős szögletes zárójelben ('[[]]') lévő sorszám azonosítja a lista elemeit, a harmadik esetben pedig a listaelem általunk megadott neve a '$' után.

A listaelemek nevét az **x** lista **names** attribútuma tartalmazza, segítségével a többi elemnek is adhatunk értéket:

```
> names(x)
[1] ""  ""  "c"

> names(x)[c(1,2)]<-c("a","b")
> names(x)
[1] "a" "b" "c"

> x
$a
 [1]  1  2  3  4  5  6  7  8  9 10

$b
[1] "A" "B"

$c
[1] TRUE
```

A lista elemeire a vektoroknál megszokott '[' operátor segítségével hivatkozhatunk, ahol numerikus, karakteres és logikai értékeket is megadhatunk:

```
> x[1]
$a
 [1]  1  2  3  4  5  6  7  8  9 10

> x[c(2,3)]
$b
[1] "A" "B"

$c
[1] TRUE

> x["a"]
$a
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> x[c(T,F,T)]
$a
 [1]  1  2  3  4  5  6  7  8  9 10

$c
[1] TRUE
```

A '[' operátorral kapott eredmény minden esetben lista, még akkor is, ha egyetlen elemét érjük el az **x** listának. Nagyon fontos ettől megkülönböztetni a '[[' operátor eredményét, amely a lista valamelyik elemével, annak az értékével tér vissza. Itt nincs mód több listaelem elérésére sem, szokás szerint numerikus vagy karakteres értékkel indexelhetünk.

```
> x[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

> x[["b"]]
[1] "A" "B"

> x[[3]]
[1] TRUE
```

A '[[' operátor alkalmazása helyett a rövidebb '$' operátort használhatjuk azoknak a listaelemeknek az elérésére, amelyeket korábban elneveztünk. A lista nevét és az elem nevét fűzzük össze a '$' operátorral:

```
> x$a
 [1]  1  2  3  4  5  6  7  8  9 10

> x$b
[1] "A" "B"

> x$c
[1] TRUE
```

Ha a lista elemét valamelyik módszer segítségével elértük, akkor további indexelés segítségével az elem összetevőit is lekérdezhetjük:

```
> x[["a"]][3:4]; x$a[4:5]<-0; x$c<-F; x
[1] 3 4
$a
 [1]  1  2  3  0  0  6  7  8  9 10

$b
[1] "A" "B"

$c
[1] FALSE
```

A lista mindem elemével a **lapply()** vagy az **sapply()** függvény segítségével hajthatunk végre műveletet:

```
> lapply(x,length)
$a
[1] 10

$b
```

```
[1] 2

$c
[1] 1

> sapply(x,length)
 a  b  c
10  2  1
```

Az **lapply()** a bemenő lista elemszámával egyező méretű listával tér vissza, melynek értékei az második paraméterben szereplő függvény visszatérési értékei. Az **sapply()** hasonlóan jár el, de a visszatérési értéke egy vektor.

3.6 Adattáblák (dataframes)

Az *adattábla* (*dataframe*) statisztikai feldolgozás szempontjából az R legfontosabb adatszerkezete. Inhomogén, kétdimenziós szerkezet, amely a lista és a mátrix adatszerkezetek előnyeit hordozza. Sorok és oszlopok alkotják, alapvetően azonos hosszúságú (oszlop)vektorok listájának tekinthető.

Adattáblát legegyszerűbben a **data.frame()** függvénnyel hozhatunk létre, a paraméterben az őt alkotó elemeket kell felsorolni. Ezek lehetnek vektorok, faktorok, mátrixok, listák vagy adattáblák is. Ha a paraméterek hossza nem azonos, akkor a függvény ismétli a rövidebb elemeket, de ez csak egész számszor lehetséges:

```
> x<-c('A','B'); y<-6:9; z<-1:8
> d<-data.frame(x,y,z); d
  x y z
1 A 6 1
2 B 7 2
3 A 8 3
4 B 9 4
5 A 6 5
6 B 7 6
7 A 8 7
8 B 9 8
```

A példában egy 8 sorból és 3 oszlopból álló adattáblát készítettünk. A **data.frame()** függvényben nem azonos hosszú vektorokat használtunk az adattábla létrehozására, az automatikus ismétléssel mégis eredményt értünk el. Nézzük, hogyan tekint az R az adattáblára:

```
> typeof(d); mode(d); length(d)
[1] "list"
[1] "list"
[1] 3

> is.list(d); is.data.frame(d)
[1] TRUE
[1] TRUE
```

Az adattáblák típusa és módja is '`list`', a hossza pedig az alkotó (oszlop)vektorok száma. Az adattáblára tehát tekinthetünk úgy, mint egy listára, melynek elemei az adattábla oszlopai lesznek.

Az adattábla sorai és oszlopai névvel is rendelkeznek, ezek attribútumokban foglalnak helyet:

```
> attributes(d)
```

```
$names
[1] "x" "y" "z"

$row.names
[1] 1 2 3 4 5 6 7 8

$class
[1] "data.frame"
```

Az **str()** függvény segítségével az adattábla szerkezetéről kapunk felvilágosítást:

```
> str(d)
'data.frame':   8 obs. of  3 variables:
 $ x: Factor w/ 2 levels "A","B": 1 2 1 2 1 2 1 2
 $ y: int  6 7 8 9 6 7 8 9
 $ z: int  1 2 3 4 5 6 7 8
```

Láthajuk, hogy a **d** adattáblánk 8 sort (megfigyelést) és 3 változót (oszlopot) tartalmaz, valamint leolvashatjuk az egyes oszlopok adattípusát is. Megfigyelhetjük, hogy a **d$x** oszlopot karakteres vektorból faktor típusú változóvá konvertálta a **data.frame()** függvény. Ezt az alapértelmezett és a statisztikában nagyon hasznos viselkedést az **I()** függvénnyel tudjuk megakadályozni:

```
> str(data.frame(I(x),y,z))
'data.frame':   8 obs. of  3 variables:
 $ x:Class 'AsIs'  chr [1:8] "A" "B" "A" "B" ...
 $ y: int  6 7 8 9 6 7 8 9
 $ z: int  1 2 3 4 5 6 7 8
```

Az **names** attribútum az adattábla oszlopainak, a **row.names** pedig a sorainak a nevét határozza meg. Ezeket az attribútumokat számos függvénnyel átírhatjuk: a **rownames()** a sorok nevét, a **colnames()** vagy a **names()** az oszlopok nevét írja át, de használhatjuk az általános **attr()** vagy **attributes()** függvényeket is. A sorok nevének meghatározásánál ügyeljünk arra, hogy azoknak egyedieknek kell lenniük, két azonos sornév nem fordulhat elő.

```
> rownames(d)<-paste(1:8, ".szemely", sep="")
> names(d)<-c("X","Y","Z")
> d
          X Y Z
1.szemely A 6 1
2.szemely B 7 2
3.szemely A 8 3
4.szemely B 9 4
5.szemely A 6 5
6.szemely B 7 6
7.szemely A 8 7
8.szemely B 9 8
```

Az adattáblák indexelése a mátrixoknál és a listáknál látott módokon is történhet. Ha az adattáblára, mint egy mátrixra tekintünk, használhatjuk a következő hivatkozásokat:

```
> d[2,3]
[1] 2

> d[c(2,3),3]
[1] 2 3

> d[c(2,3),]
```

```
          X Y Z
2.szemely B 7 2
3.szemely A 8 3

> d[,3]
[1] 1 2 3 4 5 6 7 8

> d[,3, drop=F]
          Z
1.szemely 1
2.szemely 2
3.szemely 3
4.szemely 4
5.szemely 5
6.szemely 6
7.szemely 7
8.szemely 8
```

Ha az adattáblát listaként indexeljük, akkor érvényesek a következők:

```
> d$X
[1] A B A B A B A B
Levels: A B

> d$Y
[1] 6 7 8 9 6 7 8 9

> d$Y[3:5]
[1] 8 9 6
```

Az adattábla indexelésénél logikai vektorokat is használhatunk, melyek az adattábla tartalmára vonatkozó relációs kifejezések is lehetnek. Ezzel a módszerrel érhetjük el, hogy az adattábla sorait valamilyen szempont szerint megszűrjük:

```
> d[d$Y < 8,]
          X Y Z
1.szemely A 6 1
2.szemely B 7 2
5.szemely A 6 5
6.szemely B 7 6

> d[d$Y < 8 & d$Z > 2,]
          X Y Z
5.szemely A 6 5
6.szemely B 7 6
```

3.7 Idősorok

[…]

4. Adatok olvasása és írása

Az R-ben adatokkal dolgozunk, amelyek beolvasására és kiírására az R számos eljárást kínál. Adatokat beolvashatunk a billentyűzetről, a vágóasztalról és külső adatforrásból: állományból vagy adatbázisból is. Az R-ben feldolgozott adatokat a vágóasztalra vagy állományba írhatjuk ki.

4.1. Adatok beolvasása

4.1.1.A c() és a scan() függvények

Leggyorsabban a **c()** függvény használatával hozhatunk létre adatvektort, de nagyobb elemszám, tipikusan 10 feletti esetekben, nem ez a legkényelmesebb megoldás:

```
> x<-c(4,8,3,6,8,2)
```

Szintén a parancssort használhatjuk adatbevitelre a **scan()** függvény használata során:

```
> x<-scan()

1:
```

A függvényhívás hatására megjelenő '1:' után az első vektorelemet gépelhetjük be, a bevitel végét az ENTER billentyű lenyomásával jelezzük. A megjelenő '2:' után a második elem begépelésére van lehetőségünk, és így tovább. Ha 5 elemű numerikus vektort akarunk létrehozni, akkor a '6:' megjelenése után egy ENTER segítségével fejezhetjük be a vektor létrehozását. Ekkor kilépünk a **scan()** függvényből és ezután az **x** vektor a begépelt elemeket fogja tartalmazni.

```
> x<-scan()
1: 2
2: 14
3: 3.5
4: 4.9
5: 3
6:
Read 5 items

> x
[1]  2.0 14.0  3.5  4.9  3.0
```

A **scan()** függvény használata során az adatokat a vágóasztalról is beilleszthetjük (CTRL-V), akár rögtön az '1:' megjelenése után. A tipikusan szövegszerkesztőből vagy táblázatkezelőből származó adatokat érdemes úgy előkészíteni, hogy egymás alatt, egy oszlopban soroljuk fel a számokat, majd ezt másoljuk fel a vágóasztalra. Az így beillesztett adatokat a fenti **scan()** függvényhívás helyesen fogja értelmezni.

A számok begépelése során lehetőségünk van szóközökkel, vagy esetleg más karakterekkel elválasztani az egy sorban megadott numerikus konstansokat. Az alapértelmezett szóköz elválasztó karaktert a **sep** argumentum segítségével változtathatjuk meg. Az előző vektor beolvasását így is elvégezhetjük:

```
> x<-scan()
```

```
1: 2 14 3.5
4: 4.9 3
6:
Read 5 items
```

Karakteres adatokat is beolvashatunk a **what** paraméter beállításával. Az alapértelmezett értéke a `double(0)`, amely, mint korábban láttuk, numerikus vektor létrehozását tette lehetővé. Ha az üres karaktersorozatot állítjuk be, akkor karakteres konstansokat is begépelhetünk:

```
> x<-scan(what="", sep="\n")
1: Első sor
2: Ez a második sor
3: Utolsó...
4:
Read 3 items

> x
[1] "Első sor"          "Ez a második sor" "Utolsó..."
```

A fenti példában a **sep** paraméter értéke (”\n”) biztosítja, hogy a vektor elemeit a sor vége karakter és ne a szóköz válassza el.

A numerikus és karakteres adatokon kívül logikai és bináris adatok beolvasása is lehetséges a **scan()** függvénnyel, sőt, egy lista elemeit is beolvashatjuk:

```
> x<-scan(what=list(nev="",kor=0,suly=0))
1: a 33 72
2: b 42 81
3: c 39 78
4:
Read 3 records

> x
$nev
[1] "a" "b" "c"

$kor
[1] 33 42 39

$suly
[1] 72 81 78


> as.data.frame(x)
  nev kor suly
1   a  33   72
2   b  42   81
3   c  39   78
```

Az elemi adattípusok mellett a fenti példában használt lista is lehet a **what** paraméter értéke, ekkor minden listaelem típusát a szokásos elemi adattípus jelzésével kell megadnunk. Látjuk, hogy adattáblákat is megadhatunk ezzel a módszerrel.

Vágóasztalon lévő információt közvetlenül a **readClipboard()** függvény segítségével is objektumhoz rendelhetünk. Tipikusan karakteres konstansok létrehozására használjuk, Windows környezetben:

```
x<-readClipboard()
```

A **dget()** függvény segítségével a **dput()** függvénnyel kiírt szöveges állományból olvashatunk vissza objektumokat.

```
x<- dget("adat.txt")
```

4.1.2. A read.table() család

A leggyakoribb módszer külső állomány beolvasására a **read.table()** függvény használata. Az adatokat táblázatszerűen tartalmazó, tagolt (adott karakterrel elválasztott) szöveges állományok olvasására használhatjuk. A paraméterekben többek között gondoskodhatunk az első sorban lévő oszlopnevekről (**header**), az elválasztó karakterről (**sep**), a tizedesvessző alakjáról (**dec**) és a hiányzó értékek jelöléséről (**na.strings**).

```
> d<-read.table("c:/temp/adat.txt",header=T,sep=";",dec=",")
```

A fenti sor egy első sorában oszlopneveket tartalmazó szöveges állomány tartalmát helyezi el a **d** adattáblában. Az állományban az adatokat (és az oszlopneveket is) a pontosvessző (;) választja el, a numerikus értékekben pedig vesszőt használunk a tizedesvessző jelölésére.

A paraméterekben leírt feltételeknek megfelelő szöveges állományt egy egyszerű szövegszerkesztővel is létrehozhatjuk, de sokszor egyszerűbb táblázatkezelőt használni, és az abban elkészült, táblázatos formában lévő adatokat megfelelő formátumban exportálni. (Pl. magyar Excel esetén választhatjuk a 'CSV (pontosvesszővel tagolt)' formátumot).

A **read.table()** függvény helyett használhatjuk a **read.csv()** és **read.csv2()** függvényeket is, amelyek csak a paraméterek alapértelmezett értékeiben térnek el az alapfüggvénytől. Ezekben a függvényekben a **header** alapértelmezetten `TRUE`, az elválasztó karakter pedig a vessző (csv) ill. a pontosvessző (csv2), valamint a tizedesvessző alakja a pont (csv) ill. a vessző (csv2). Ha tabulátorral tagolt állományt szeretnénk beolvasni, akkor a **read.delim()** ill. a **read.delim2()** függvényeket érdemes használni, mert az elválasztó karakter itt alapértelmezés szerint a tabulátor karaktert (`"\t"`).

4.1.3. A read.fwf() függvény

A legtöbb beolvasandó szöveges állomány tabulátorral vagy pontosvesszővel tagolt. Ritkábban szükség lehet fix széles mezőket tartalmazó állományok beolvasására is. A **read.fwf()** függvény **width** paraméterében kell megadnunk az egyes mezők hosszát. A függvény a megadott mezőhossz értékek alapján egy ideiglenes, tabulátorral elválasztott szöveges állományt hoz létre, amely a **read.table()** függvénnyel kerül ténylegesen feldolgozásra.

```
> allomany.nev<-tempfile()
> cat(file=allomany.nev,"A;B;C","123456","987654",sep="\n")
> read.fwf(allomany.nev, widths=c(1,2,3),header=T,sep=";")
  A  B   C
1 1 23 456
2 9 87 654
```

A fenti példában a **tempfile()** függvényt használjuk egy a rendszerünkben érvényes ideiglenes állomány nevének meghatározására. A **cat()** függvénnyel egy 3 soros szöveges állományt hozunk létre. Az első sor pontosvesszővel elválasztott oszlopneveket tartalmaz, a következő két sor pedig 3 fix széles adatmezőt tartalmaz. Ezek hossza rendre 1, 2 és 3 karakternyi. A **read.fwf()** függvényben pontosan ezeket a mezőhosszakat adjuk meg a **width** paraméterben. A **header** paraméterrel jelezzük, hogy az első sor oszlopneveket

tartalmaz, a **sep** paraméter pedig az első sorban használt elválasztó karaktert jelöli. A **sep** paraméterre csak akkor van szükség, ha oszlopneveket tartalmazó sort is be akarunk olvasni. Láthatjuk, hogy a függvény által visszaadott adattábla 2 sort és 3 oszlopot tartalmaz.

4.1.4. Bináris állományok olvasása

Az R-ben számos más statisztikai programcsomag adatállományát is beolvashatjuk. Ezek a **foreign** csomagban található függvények a

```
> library(foreign)
```

függvényhívás után lesznek elérhetőek, és segítségükkel többek között SPSS, SAS, Minitab, S-PLUS, Stata és Systat állományokat is beolvashatunk. Az összes elérhető függvény listáját a

```
> ls("package:foreign")
 [1] "data.restore"  "lookup.xport"  "read.arff"     "read.dbf"
 [5] "read.dta"      "read.epiinfo"  "read.mtp"      "read.octave"
 [9] "read.S"        "read.spss"     "read.ssd"      "read.systat"
[13] "read.xport"    "write.arff"    "write.dbf"     "write.dta"
[17] "write.foreign"
```

paranccsal kérdezhetjük le.

4.1.5. Adatbázisok elérése

A kis és közepes nagyságú adattáblák megnyitása és kezelése az R-ben nem jelent nehézséget. Ugyan a **read.table()** függvény a megfelelő oszloptípusok kitalálása miatt, sokszor lassúnak bizonyul, kb. 100 Mb alatti szöveges állományok kezelése nem okoz problémát. Mivel az R az objektumokat a memóriában tárolja, az ennél nagyobb méretű állományok esetén adatbázis-kezelők használata javasolt. Az adatokat tehát nem az R-ben, hanem egy külső adatforrásban tároljuk, amit az R-ből elérhetünk, lekérdezhetünk.

Az R az **RODBC** csomag segítségével számos adatbázis-kezelőhöz tud kapcsolódni (pl. Microsoft SQL Server, Access, MySQL, Oracle), de nagyméretű állományokban tárolt információkhoz is hozzáférhetünk (pl. Excel, DBase, szöveges állományok).

Az RODBC csomagban lévő függvények segítségével kapcsolódhatunk egy ODBC kapcsolattal rendelkező adatbázishoz. Ez lehet akár egy Excel állomány is:

```
> kapcsolat<-odbcConnect("Excel adatok")
> sqlTables(kapcsolat)
> minta<-sqlFetch(kapcsolat, "Munka1")

> odbcCloseAll()
```

Adatbázishoz az **odbcConnect()** függvény segítségével kapcsolódhatunk, majd az **sqlTables()** hívással megkajuk az elérhető táblákat. Az **sqlFetch()** teljes tábla tartalmát adja vissza.

4.2. Adatok kiírása

Az objektumok értékeinek kiírása könnyebb feladat, mint az adatállományok beolvasása. Egyszerűbb objektumokat a **cat()** függvénnyel, adattáblákat és mátrixokat a **write.table()** függvénnyel írhatjuk ki.

4.2.1. A cat() függvény

A **cat()** függvénnyel egyszerű objektumokat írhatunk a képernyőre vagy állományba. A **file** paraméterben gondoskodhatunk a kimeneti állomány nevéről:

```
> cat("Helló világ\n")
Helló világ

> cat("Helló világ\n", file="kiir.txt")
```

A **sep** paraméterben az elemi adatokat elválasztó karaktert határozzuk meg, az alapértelmezés a szóköz.

```
> x<-1:5
> cat(x, "\n", sep="\t")
1       2       3       4       5
```

A példában az **x** vektor értékei jelennek meg a képernyőn, majd egy sortörés karakter. Az adatokat a tabulátor jel választja el.

4.2.2. A write.table() család

Adattáblák és mátrixok kiírására a **write.table()** függvényt használhatjuk. Az első paraméter a kiírandó objektum neve, a második pedig a kimenti állomány elérési útja. A **row.names** és a **col.names** logikai paraméterek szabályozzák, hogy a sor és oszlopnevek szerepeljenek-e a kimeneti állományban. Ezek alapértelmezett értéke `TRUE`.

```
> xmat<-matrix(1:12,nrow=3)
> write.table(xmat,"table.txt",col.names=F,row.names=F)
```

Mátrixok kiírásánál sokszor a sor- és oszlopnevek kiírásától eltekintünk, adattáblák esetében azonban ez fontos lehet. Arra is van lehetőségünk, hogy az állomány helyett a vágóasztalt válasszuk kimenetnek. Ekkor a

```
> write.table(women,"clipboard", sep="\t", col.names=NA)
```

sorral a vágóasztalra helyezhetjük az adattáblát, amit egy táblázatkezelőbe azonnal beilleszthetünk (`CTRL+V`). A fenti példa a sorneveket is kiírja, így a helyes beillesztéshez az első sorban, a sor elején egy új oszlopnévre is szükség van. A **col.names** argumentum `NA` értéke üres oszlopnév mezőt helyez el az első sorban.

A **write.table()** paramétereinek alapértelmezett értékén változtat a **write.csv()** és **write.csv2()** függvény.

5. Adattáblák kezelése

Az adatkezelés szempontjából legfontosabb R objektum az adattábla (dataframe). Mint korábban láttuk, a mátrixhoz hasonlóan sorokat és oszlopokat tartalmaz, illetve a listához hasonlóan elemekből, méghozzá azonos hosszúságú oszlopvektorokból, épül fel. Az adattábla kettős eredete jelentősen megkönnyíti az ilyen adatok kezelését.

Az adattábla sorai egyedekre (személyek, tárgyak, dolgok, stb.) vonatkozó megfigyelések, az oszlopok pedig a megfigyelt tulajdonságok. A statisztikában úgy mondanánk, hogy az adattáblában az adatmátrixunkat/többdimenziós mintánkat rögzíthetjük, a sorok a mintaelemek, az oszlopok a megfigyelt változók.

Az adattábla inhomogén adatszerkezet, oszlopai különböző típusú adatokat is tartalmazhatnak. Jellemzően kvalitatív (nominális és ordinális skálán mért) adatok tárolására a faktort használjuk, kvantitatív (intervallum és arányskálán mért) adatok tárolására a numerikus vektort. Természetesen adattáblában karakteres és logikai vektorok is szerepelhetnek, sőt dátumokat és időpontokat is kezelhetünk az adattáblában.

Az adatok adattáblába szervezésénél vezérlő elv, hogy az azonos változóhoz tartozó adatértékek kerüljenek azonos oszlopba. Tekintsünk egy egyszerű kísérletet, ahol arra keressük a választ, hogy a táplálkozás módja befolyásolja-e a vér alvadási idejét. Véletlenszerűen kiválasztunk 24 állatot az egyes diétákhoz (A, B, C, D) és adatainkat papíron a következőképpen rendezzük:

A	B	C	D
62	63	68	56
60	67	66	62
63	71	71	60
59	64	67	61
	65	68	63
	66	68	64
			63
			59

Az egyes numerikus értékek másodpercben mért véralvadási időket jelentenek. Ahhoz, hogy helyesen hozhassuk létre az adattáblánkat, a következőképpen kell átalakítani a fenti táblázatot.

A	62
A	60
A	63
A	59
B	63
B	67

B	71
B	64
B	65
B	66
C	68
C	66
C	71
C	67
C	68
C	68
D	56
D	62
D	60
D	61
D	63
D	64
D	63
D	59

Ezt a táblázatot már rögzíthetjük az R egy adattáblájába.

5.1. Adattáblák létrehozása

Korábban már láttuk, hogy a **data.frame()** függvénnyel hogyan hozhatunk létre adattáblát (3.5. fejezet): a függvény argumentumában az adattáblát alkotó oszlopvektorokat kell felsorolnunk. Állományban rögzített adattábla beolvasására is láttunk példát a 4.1.2. fejezetben: pl. a **read.table()** függvénnyel adattáblába olvashatjuk adatainkat.

További lehetőség a **fix()** és az **edit()** függvények használata, melyekkel az Excel munkalapjához hasonló, kényelmes felületen hozhatjuk létre új, vagy módosíthatjuk meglévő adattábláinkat. A függvények a paraméterükben várják a módosítandó adattábla nevét, de az **edit()** – a **fix()**-szel ellentétben – nem változtatja meg a paraméter értékét, hanem a módosított adattábla lesz a visszatérési értéke.

```
> fix(d)            # módosítja d-t
> d.uj<-edit(d)     # d változatlan, a módosítások d.uj-ba kerülnek
```

Az R számos beépített *dataset*-tel (többnyire adattábla, mátrix, idősor) rendelkezik. A **data()** függvény segítségével fedezhetjük fel a parancssorból már is elérhető adattáblák nevét és rövid leírását. További információt is kérhetünk egy kiválasztott adattábláról (pl. women):

```
> ?women
```

Az alapértelmezetten betöltött csomagok közül a **datasets** tartalmaz adattáblákat, de számos más csomagban is találhatók dataset-ek, pl: **MASS**, **survival**, **nlme**. A telepített csomagban lévő összes dataset is lekérdezhető következő sorral:

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

5.2. Adattáblák indexelése

Az adattáblák indexelésére a mátrixok és listák indexelési eljárásait is használhatjuk. A 3.5. fejezetben már áttekintettük ezeket a lehetőségeket.

A következő példákban a **datasets** csomag **mtcars** adattábláját használjuk, amely a parancssorból azonnal elérhető. Az 32 db régi autó fogyasztását és 10 további jellemzőjét tartalmazó adattábla első 10 sorát láthatjuk itt:

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

Az adattábla változóinak részletes leírását a következő parancs szolgáltatja:

```
> ?mtcars
```

Indexeléssel elérhetjük adattábla tetszőleges értékét:

```
> mtcars[6,3]
[1] 225

> mtcars$disp[6]
[1] 225
```

vagy tetszőleges sorát:

```
> mtcars[6,]
        mpg cyl disp  hp drat   wt  qsec vs am gear carb
Valiant 18.1   6  225 105 2.76 3.46 20.22  1  0    3    1
```

vagy oszlopát:

```
> mtcars[,8]
 [1] 0 0 1 1 0 1 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0
[28] 1 0 0 0 1

> mtcars$am
 [1] 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1
[28] 1 1 1 1 1
```

Az adattábla értékeinek egy tetszőleges tartománya is elérhető, ha vektorokkal indexelünk:

```
> mtcars[11:15,c(1,3,8)]
                   mpg  disp vs
Merc 280C         17.8 167.6  1
Merc 450SE        16.4 275.8  0
Merc 450SL        17.3 275.8  0
Merc 450SLC       15.2 275.8  0
Cadillac Fleetwood 10.4 472.0  0
```

Ha az indexvektorban használt numerikus értékek sorrendjét felcseréljük, akkor az adattábla oszlopait, esetleg sorait cserélhetjük fel:

```
> mtcars[15:11,c(3,8,1)]
                   disp vs  mpg
Cadillac Fleetwood 472.0  0 10.4
Merc 450SLC       275.8  0 15.2
Merc 450SL        275.8  0 17.3
Merc 450SE        275.8  0 16.4
Merc 280C         167.6  1 17.8
```

Amikor az adattábla egyetlen oszlopából válogatunk le adatot, akkor az eredmény nem adattábla, hanem egy vektor lesz. Ezt elkerülhetjük, ha a **drop**=FALSE argumentumot használjuk:

```
> mtcars[c(2,4,6),8]
[1] 0 1 1

> mtcars[c(2,4,6),8,drop=F]
                vs
Mazda RX4 Wag    0
Hornet 4 Drive   1
Valiant          1
```

Bizonyos esetekben szükség lehet az adattábla sorainak véletlen kiválasztására. Ekkor a **sample()** függvényt használjuk:

```
> mtcars[sample(1:32,10),]
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Porsche 914-2     26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

A **sample()** függvény alapértelmezés szerint visszatevés nélküli választ véletlen értékeket az első paraméterből (esetünkben 32 számból 10-et), de ha a **replace**=TRUE argumentumot használjuk, akkor visszatevéssel fog választani:

```
> d<-mtcars[sample(11:13,4,replace=T),]; d
            mpg cyl  disp  hp drat   wt qsec vs am gear carb
Merc 280C  17.8   6 167.6 123 3.92 3.44 18.9  1  0    4    4
Merc 450SL 17.3   8 275.8 180 3.07 3.73 17.6  0  0    3    3
Merc 280C  17.8   6 167.6 123 3.92 3.44 18.9  1  0    4    4
Merc 450SE 16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
```

A **sample()** függvény fenti paraméterezése mellett biztosan előfordul sorismétlés az új **d** adattáblában. Most az 1. és a 3. sor azonos, de mivel a sorok nevének az adattáblában különbözőnek kell lennie a "Merc 280C" egy ".1" résszel egészült ki a megismételt 3. sorban.

Az sorismétléseket tartalmazó adattáblák kezelésére a **duplicated()** és a **unique()** függvényeket használhatjuk. A sorismétlések felderítésére a **duplicated()** függvényt használjuk:

```
> duplicated(d)
[1] FALSE FALSE  TRUE FALSE

> d[duplicated(d),]
             mpg cyl  disp  hp drat   wt qsec vs am gear carb
Merc 280C.1 17.8   6 167.6 123 3.92 3.44 18.9  1  0    4    4
```

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

```
> unique(d)
            mpg cyl  disp  hp drat   wt qsec vs am gear carb
Merc 280C  17.8   6 167.6 123 3.92 3.44 18.9  1  0    4    4
Merc 450SL 17.3   8 275.8 180 3.07 3.73 17.6  0  0    3    3
Merc 450SE 16.4   8 275.8 180 3.07 4.07 17.4  0  0    3    3
```

5.3. A with() és az attach()

Az adattáblák változóinak indexelés nélküli elérésére több módszert is nyújt az R. Egyrészt számos statisztikai függvény rendelkezik **data** argumentummal, amelynek ha egy adattábla nevét adjuk értékül, akkor illető adattábla oszlopneveit mindenfajta indexelés nélkül használhatjuk a statisztikai függvény többi argumentumában.

```
> names(women)
[1] "height" "weight"
> lm(height~weight,data=women)
```

További lehetőségünk a **with()** függvény használata, amelyben ha az első argumentumban az adattábla nevét szerepeltetjük, a második argumentumban szereplő kifejezésben nem kell indexelnünk az adattábla változóit:

```
> with(women, lm(height~weight))
```

Ha több utasítást szeretnénk végrehajtani a **with()** függvény használata során, akkor a második argumentumban használjunk kapcsos zárójelet ({}), az utasításokat pedig írjuk külön sorba, vagy válasszuk el pontosvesszővel őket.

```
> with(women, {plot(height~weight); abline(lm(height~weight))})
```

Az **attach()** függvény használatával is megtakaríthatjuk a változók indexelését. A függvény argumentumában szereplő adattáblát az elérési útba helyezzük, így a névfeloldás során az adattáblánkat is végignézi az R.

```
> names(women)
[1] "height" "weight"
> height
Error: object "height" not found
> weight
Error: object "weight" not found
```

```
> attach(women)
> height
 [1] 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
> weight
 [1] 115 117 120 123 126 129 132 135 139 142 146 150 154 159 164
```

Az elérési útból a **detach()** függvénnyel törölhetünk adattáblát (vagy csomagot).

```
> detach(women)
> height
Error: object "height" not found
> weight
Error: object "weight" not found
```

Az **attach()** használatával azonban óvatosnak kell lennünk, mert a munkaterület egyéb változóival könnyen ütközhetnek az adattábla változónevei:

```
> height<-1:10
> attach(women)

        The following object(s) are masked _by_ .GlobalEnv :

         height

> height
 [1]  1  2  3  4  5  6  7  8  9 10
> weight
 [1] 115 117 120 123 126 129 132 135 139 142 146 150 154 159 164
```

5.4. Sorok és oszlopok nevei

Az adattábla sorait a **row.names**, az oszlopait a **names** attribútum használatával nevezhetjük el. A sornevek egymástól különböző, karakteres vagy numerikus egész értékek lehetnek, míg az oszlopnevek csak karakteres adatok. A sor- és oszlopnevek lekérdezésére és beállítására korábban már láttunk példát (3.5. fejezet).

Sokszor előfordul, hogy egy adattábla valamely változójának értékeivel szeretnénk a sorokat elnevezni, ill. fordítva, az adattábla sorneveit oszlopvektorban szeretnénk látni. Az adattábla állományból történő beolvasása során a **read.table()** függvényben a `"row.names=n"` argumentum megadásával a szöveges állomány `n`. oszlopából nyerjük a sorok neveit.

Az **mtcars** adattábla sorneveit a következő parancs segítségével vihetjük be változóba:

```
> mtcars2<-data.frame(name=rownames(mtcars),mtcars,row.names=1:32)
> mtcars2[1:10,]
                name  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
1          Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
2      Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
3         Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
4     Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
5  Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
6            Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
7         Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
8          Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
9           Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
10          Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

A fordított irányhoz a következő parancsot kell használnunk:

```
> mtcars3<-mtcars2                    # új adattábla

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

5.5. Rendezés

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

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

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

```
> mtcars[order(mtcars$mpg)[1:10],]
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
```

Rendezési szempontnak a sorneveket is használhatjuk:

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

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

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

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

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

5.6. Adattábla szűrése

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

```
> mtcars$mpg<15
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[11] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE
[21] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[31] FALSE FALSE
```

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

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

```
Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
```

Több változón alapuló feltétel megadásához összetett logikai kifejezést kell írnunk:

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

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

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

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

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

5.7. Hiányzó értékeket tatalmazó sorok eltávolítása

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

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

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

```
Merc 450SE     16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL     17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
```

5.8. Adattábla oszlopainak transzformálása

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

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

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

> women$suly<-round(women$weight*0.45)
> women
   height weight suly
1      58    115   52
2      59    117   53
3      60    120   54
4      61    123   55
5      62    126   57
6      63    129   58
7      64    132   59
8      65    135   61
9      66    139   63
10     67    142   64
11     68    146   66
12     69    150   68
13     70    154   69
14     71    159   72
15     72    164   74
```

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

```
> transform(women,magassag=round(height*2.45))
   height weight suly magassag
1      58    115   52      142
2      59    117   53      145
3      60    120   54      147
4      61    123   55      149
5      62    126   57      152
```

```
6       63    129    58       154
7       64    132    59       157
8       65    135    61       159
9       66    139    63       162
10      67    142    64       164
11      68    146    66       167
12      69    150    68       169
13      70    154    69       172
14      71    159    72       174
15      72    164    74       176
```

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

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

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

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

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

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

A létrejövő faktor szintjei az intervallumok leírásaiból állnak, természetesen ezeket megváltoztathatjuk, csak a **labels** paramétert kell használnunk:

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

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

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

6. Grafika az R-ben

Az R számos eljárást ad a grafikus ábrák létrehozására, sőt saját ábratípusokat is létrehozhatunk. A parancssorba gépelt **demo(graphics)** vagy **demo(persp)** függvényhívásokkal képet kaphatunk az R grafikus lehetőségeiről.

Az R segítségével *grafikus eszköz*re (*graphical device*) rajzolhatunk, amely alapértelmezetten a képernyő, de egy adott típusú állomány is lehet. A grafikus (rajz)függvényeink két alapvető csoportba sorolhatók, az ún. *magas-szintű* (*high-level*) rajzfüggvények és az *alacsony-szintű* (*low-level*) függvények.

6.1. Grafikus eszközök

Az R-ben megnyithatunk egy vagy több grafikus eszközt, ezekből mindig az egyik az aktuális (aktív), amelyre a grafikus függvényeinkkel rajzolhatunk. Az alapértelmezett grafikus eszközünk az R-ben a képernyő egy ablaka (*windows*), de több állománytípus (pl. *jpeg*, *pdf*, *postscript*) áll rendelkezésre, amelyekben létrehozhatjuk ábráinkat. A grafikus eszközök leírását a

```
> ?device
```

paranccsal érhetjük el.

Grafikus eszközt a típusuknak megfelelő függvényhívással hozhatunk létre. A következő sorban két *windows* és egy *jpeg* típusú grafikus eszközt hozunk létre:

```
> windows(); windows(); jpeg("abra.jpg")
> dev.list()
        windows         windows jpeg:75:abra.jpg
              2               3                 4
```

Az **dev.list()** függvénnyel a megnyitott grafikus eszközeink listáját kapjuk meg: az eszköz típusát és azonosítóját látjuk a képernyőn. Az első megnyitott eszközünk 2-es azonosítót kapja.

A megnyitott eszközeink egyike az aktuális grafikus eszköz, az utoljára megnyitott eszköz lesz az aktuális. Az aktuális eszközt a **dev.cur()** függvénnyel kérdezhetjük le, beállítása a **dev.set()** függvénnyel lehetséges:

```
> dev.cur()
jpeg:75:abra.jpg
               4

> dev.set(3)
windows
      3

> dev.cur()
windows
      3
```

A grafikus eszközök bezárását a **dev.off()** függvény végzi. Paraméter nélkül az aktuális eszközt zárhatjuk be, numerikus paraméterrel pedig az adott sorszámú eszközt. A visszatérési értéke az aktuális grafikus eszköz. A **graphics.off()** az összes nyitott eszközt bezárja:

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

```
jpeg:75:abra.jpg
                4

> dev.off(2)
jpeg:75:abra.jpg
                4

> graphics.off()
```

6.2. Az eszközfelület felosztása

Nagyon hasznos lehetőség az R-ben, hogy a rajzterület felosztható különálló részekre (ablakokra), melyek mindegyikére a többitől függetlenül rajzolhatunk. A felosztásra több lehetőségünk is van, ezek nem minden esetben kompatibilisek egymással.

A legegyszerűbben a **mfrow** vagy **mfcol** grafikus paramétereket használhatjuk a felosztásra, egy kételemű numerikus vektor lehet az értékük, amelyek a felosztott ablakok sorainak és oszlopainak a számát határozzák meg.

```
> par(mfrow=c(3,2))
```

A fenti sor hatására egy 3 sorból és 2 oszlopból álló, 6 elkülönült ablakot tartalmazó rajzterületet kapunk. Az **mfrow** esetén a felosztás soronként, az **mfcol** esetén pedig oszloponként történik.

Az aktuális felosztásról a **layout.show()** függvény ad tájékoztatást:

```
> par(mfrow=c(3,2)); layout.show(6)
```

A függvény sorszámokkal azonosítja a különböző rajzterületeket, így azok beazonosíthatók, nagyságuk és elhelyezkedésük az ábráról leolvasható:

1	2
3	4
5	6

A fenti felosztás az **mfcol** paraméter beállításával is elérhető, de ekkor a rajzterületek kiosztása oszloponként történik:

```
> par(mfcol=c(3,2)); layout.show(6)
```

1	4
2	5
3	6

A felosztott rajzterületekre az ábrák rajzolása az azonosítók sorrendjében történik, először az 1-es sorszámú rajzterületre rajzolunk, majd a következő magas-szintű függvényhívás a 2-es sorszámú rajzterületre rajzol, és így tovább.

Ritkán előfordul, hogy a fenti sorrendet felül szeretnénk bírálni és azt szeretnénk, hogy a következő magas-szintű függvényünk egy adott (nem a sorrendben következő) rajzterületre vonatkozzon. Ekkor az **mfg** paramétert használhatjuk az aktuális rajzterület beállítására:

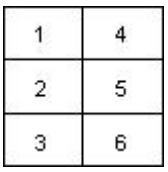
```
> par(mfg=c(2,1,3,2))
```

A fenti példában arról rendelkezünk, hogy a 2. sor 1. oszlopába kerüljön a következő ábra. A 4 elemű numerikus vektor utolsó két értéke csak megismétli a felosztás tényét, miszerint a rajzterület 3 sor és 2 oszlop mentén lett felosztva.

A rajzterület rugalmasabb felosztását teszi lehetővé **layout()** függvény, amely a paraméterében szereplő mátrix értékei mentén végzi a felosztást. A rajzterület részei eltérő méretűek is lehetnek, valamint gondoskodhatunk az egymás melletti részek egyesítéséről is.

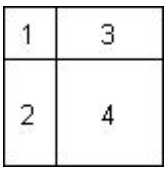
A **mfcol** paraméter használatánál szereplő felosztást a következő **layout()** függvényhívás hajtja végre:

```
> layout(matrix(1:6,ncol=2)); layout.show(6)
```



Ha eltérő nagyságú területekre szeretnénk felosztani a rajzunkat, akkor a **widths** és a **heights** paraméterekkel állíthatjuk be az oszlopok és a sorok egymáshoz viszonyított arányát.

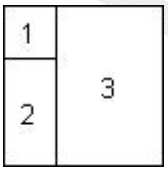
```
> layout(matrix(1:4,ncol=2), widths=c(1,2), heights=c(1,2))
> layout.show(4)
```



A felosztás itt egy 2x2-es mátrix alapján történik, ahol a **widths** paraméter a két oszlop szélességét állítja be: a második oszlop az első oszlop szélességének kétszerese. A **heights** paraméter hasonlóan határozza meg a sorok magasságát.

A **layout()** függvény első paraméterében szereplő mátrix azonos értékeket is tartalmazhat, ekkor egymás melletti rajzterületek összevonására van lehetőségünk:

```
> layout(matrix(c(1:3,3),ncol=2), widths=c(1,2), heights=c(1,2))
> layout.show(3)
```



A fenti két felosztási módszertől alapjaiban eltérő a **split.screen()** függvénnyel történő felosztás.

```
> split.screen(c(2,1))
[1] 1 2
```

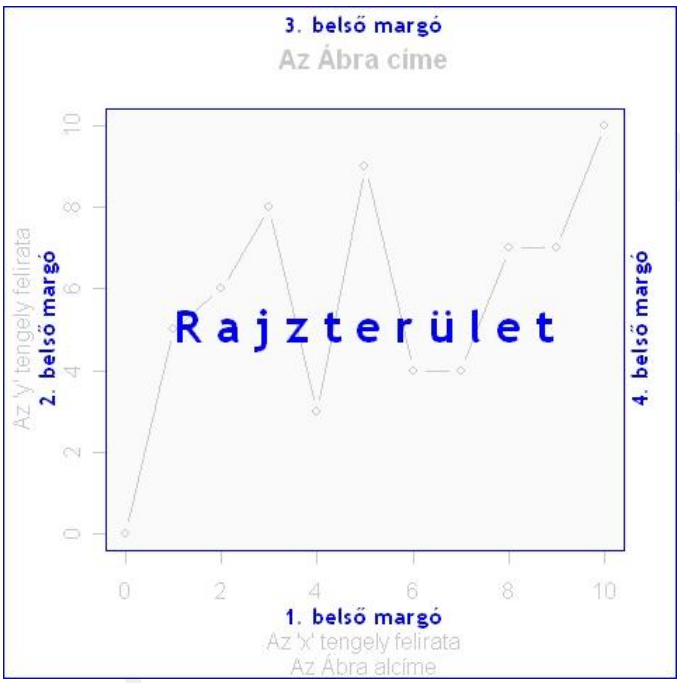
A példában 2 sor és 1 oszlop mentén a rajzterületet egy felső és alsó részre osztottuk. Ezek azonosítója a visszatérési értéknek megfelelően 1 és 2. Az azonosítókat felhasználhatjuk az adott rajzterület továbbosztásához:

```
> split.screen(c(1,3), screen=2)
[1] 3 4 5
```

A 2-es rajzterületet osztjuk 1 sor és 3 oszlop mentén három egyenlő részre, ezek új azonosítóival tér vissza a függvény. A rajzterület kiválasztását a **screen()** függvénnyel végezzük, paraméterben egy rajzterület azonosítóját kell megadnunk. Rajzterület tartalmát az **erase.screen()** függvénnyel törölhetjük, a rajzterület felosztásából a **close.screen()** függvénnyel léphetünk ki.

6.3. Az eszközfelület részei

Az eszközfelületen egyetlen ábra létrehozásakor a következő részeket különböztethetjük meg: külső margó, ábraterület, belső margó, rajzterület. A 6.1. ábrán belülről kifelé haladva a rajzterület és a belső margó látható, melyek együtt egy olyan ábraterületet alkotnak, amely teljesen kitölti az eszközfelületet. Külső margó alapértelmezés szerint nincs, ezt magunknak kell beállítani.



6.1. ábra

A rajzterületre kerülnek az ábrázolandó pontok, vonalak, görbék, 2 és 3 dimenziós alakzatok. A belső margón foglal helyet az ábra címe és alcíme, a tengelyek feliratai, a beosztások címkéi és maguk a beosztások is. A rajzterület és a belső margó határán helyezkedik el a két tengely, valamint a rajzterületet körbevevő szegély is.

A belső margók a **mai** (inch-ben mért) és **mar** (szövegsorokban mért) paraméterek segítségével kérdezhetők le és állíthatók be.

```
> par(c("mai", "mar"))
$mai
[1] 0.95625 0.76875 0.76875 0.39375

$mar
[1] 5.1 4.1 4.1 2.1
```

A kapott értékek rendre az alsó, bal oldali, felső és jobb oldali margókat jelentik.

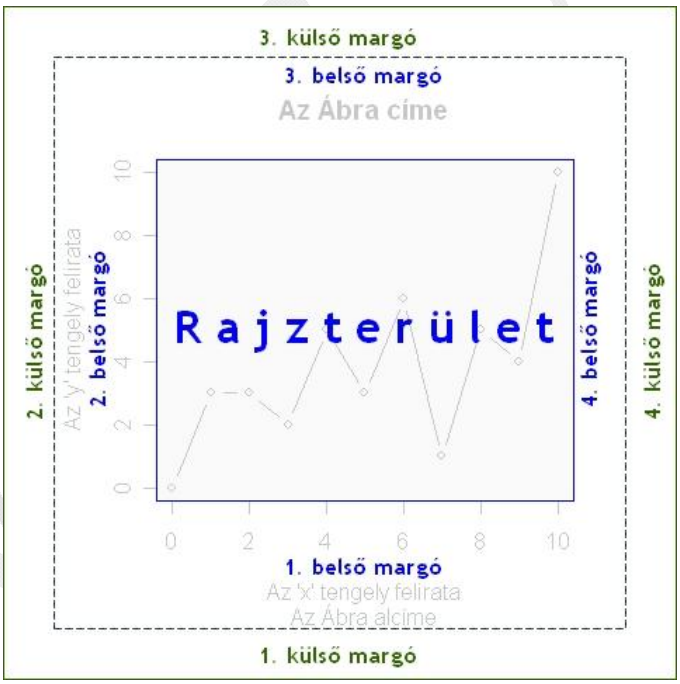
A külső margók az **omi** és az **oma** paraméterek segítségével kezelhetők és mint láttuk alapértelmezetten nem kerülnek beállításra:

```
> par(c("omi", "oma"))
$omi
[1] 0 0 0 0

$oma
[1] 0 0 0 0
```

Az **omi** az inch-ben mért, az **oma** pedig a szövegsorban mért margónagyságot határozza meg. Ha 2 szövegsornyira állítjuk mindegyik külső margót

```
> par(oma=c(2,2,2,2))
> par(c("omi", "oma"))
$omi
[1] 0.375 0.375 0.375 0.375

$oma
[1] 2 2 2 2
```

akkor az eszközfelület a 6.2. ábrának megfelelően épül fel.



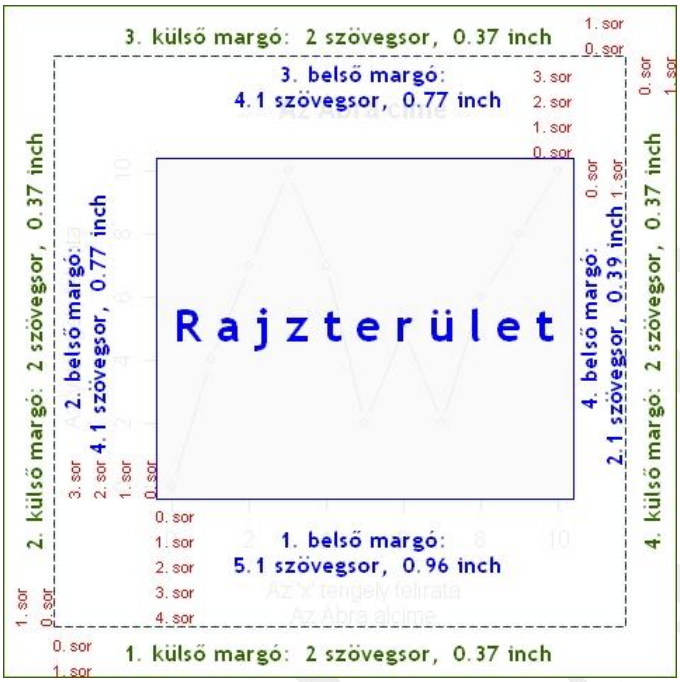
6.2. ábra

A margókban (külső vagy belső) szöveges információt az **mtext()** függvénnyel helyezhetünk el. Az **side** paraméter a margó kiválasztását jelenti (1=alsó, 2=bal, 3=felső, 4=jobb), a **line** paraméterrel a margó szövegsorát adjuk meg (0-val kezdődik), az **outer** paraméter pedig a külső vagy belső margó közötti választásról gondoskodik (`TRUE`=külső margó, `FALSE`=belső margó). A külső és belső margóba írhatunk a következő parancsokkal:

```
> mtext("Külső margó", outer=T, line=2)
```

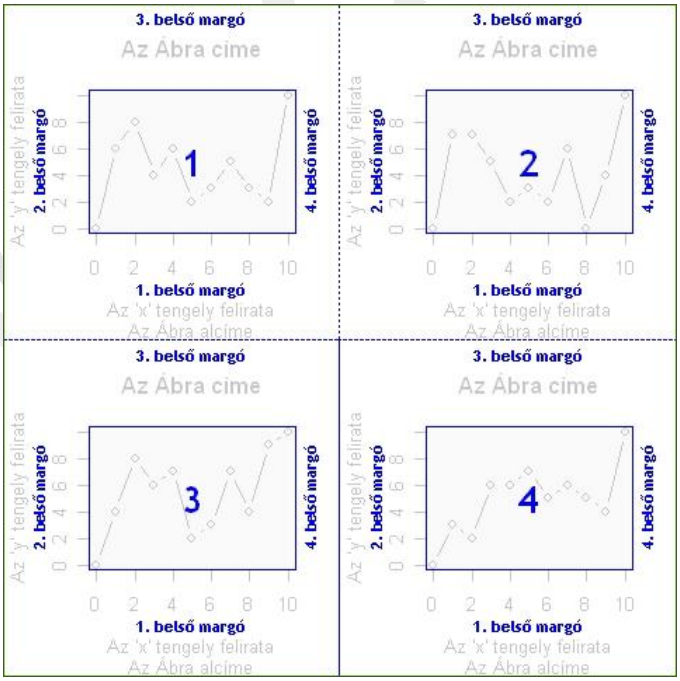
```
> mtext("Belső margó", outer=F, line=1)
```

A belső és külső margóba írható szövegsorokról a 6.3. ábra tájékoztatást. Az **mtext()** függvény **line** paramétere határozza meg a szöveg helyét, az **adj** paraméter a szöveg igazítását (a tengelyekkel párhuzamos irányban az adj=0 balra igazít, adj=1 jobbra igazít).



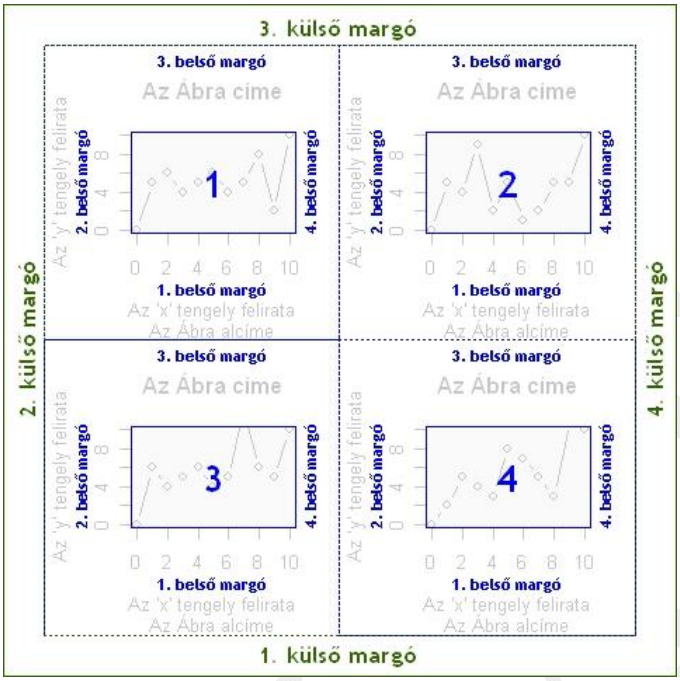
6.3. ábra

Amennyiben az eszközfelületünket több ablakra osztjuk az alapértelmezett felosztásban az ablakokban egy-egy ábraterület található, amelyek belső margóval és rajzterülettel rendelkeznek (6.4. ábra).



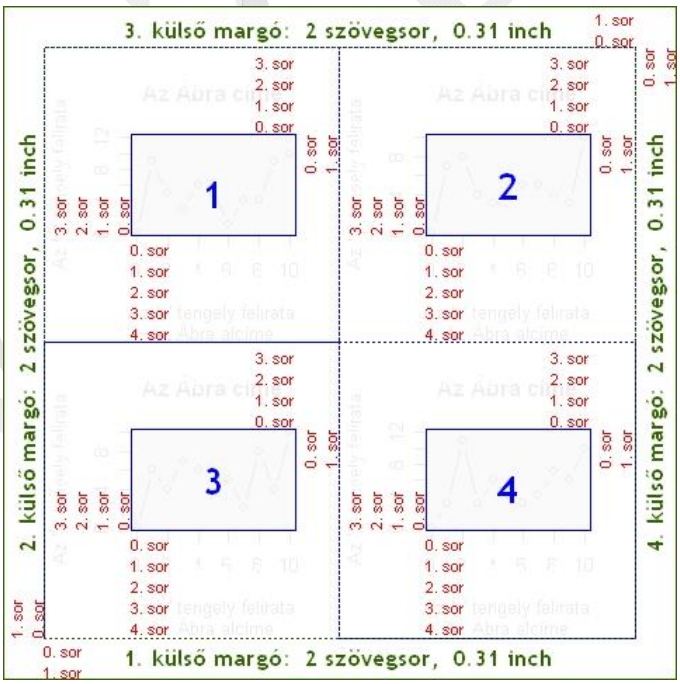
6.4. ábra

A külső margó most is hiányzik, de a szokásos grafikus paraméterek segítségével beállíthatjuk, ekkor a 6.5. ábrának megfelelő felosztást kapjuk.



6.5. ábra

Természetesen a margókba most is írhatunk tetszőleges szöveget az **mtext()** függvény segítségével a 6.6. ábrának megfelelően.



6.6. ábra

6.4. Magas-szintű rajzfüggvények

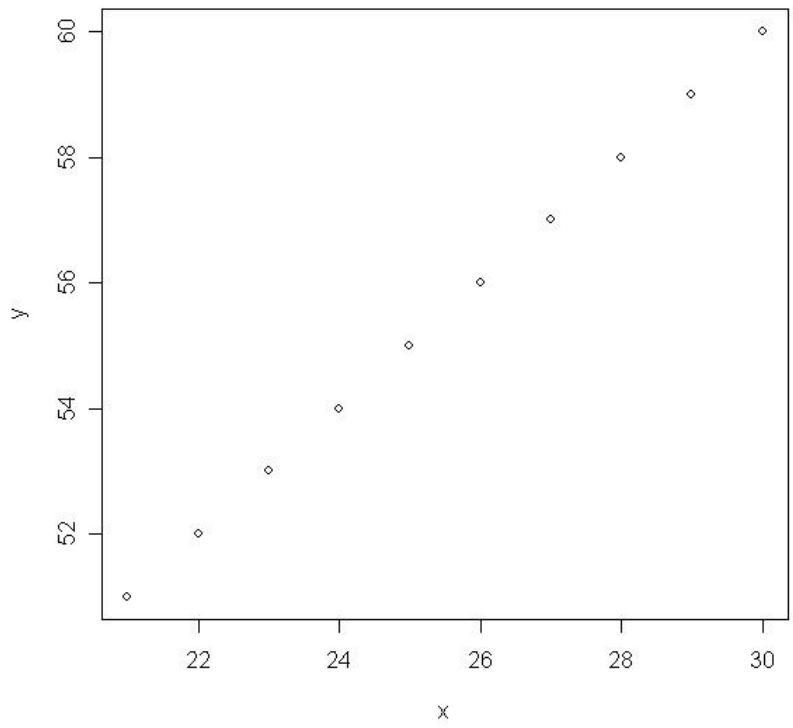
A magas-szintű rajzfüggvények az aktív grafikai eszközön hoznak létre új ábrát, ami a legtöbbször a régi tartalom törlésével jár. Az ábrához többek között tengelyek, címkék és feliratok tartoznak, amelyek megjelenítéséről a grafikai paraméterekkel gondoskodhatunk.

A legtöbbször használt magas-szintű függvények a **plot()**, **hist()**, **barplot()**, **boxplot()**, **pie()** és a **curve()**.

6.4.1. A plot() függvény

A **plot()** az R legalapvetőbb függvénye ábrák létrehozására. Egyik legegyszerűbb felhasználása, hogy az *x* és *y* tengelyek mentén a függvény bemeneti paramétereiben meghatározott értékpárokhoz (mint x és y értékekhez) egy-egy pontot jelenítsen meg az ábrán (6.7. ábra):

```
> x<-21:30
> y<-51:60
> plot(x,y)
```



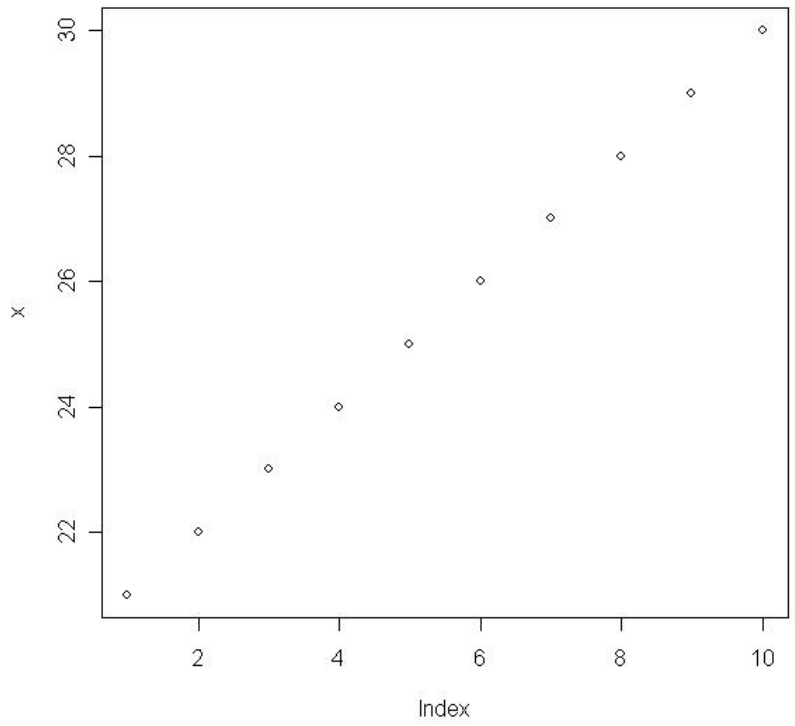
6.7. ábra

Az **x** és **y** numerikus vektorok 10-10 eleműek, a 6.7. ábrán is 10 pontot látunk. Az első pontnak megfelelő karika a (21, 51) koordinátájú pontban rajzolódik ki, ezek a koordináták az **x** és **y** vektor első elemei. A második kis karika rajzolásához a numerikus vektorok 2. elemeit használja a **plot()**, ez a (22, 52) koordinátájú pont lesz. A többi 8 pont is hasonlóan, az azonos pozíción lévő vektorelemeknek megfelelő értékpárok alapján jelenik meg az ábrán.

A **plot()** függvény egyetlen vektorral is hívható:

```
> x<-21:30
```

```
> plot(x)
```

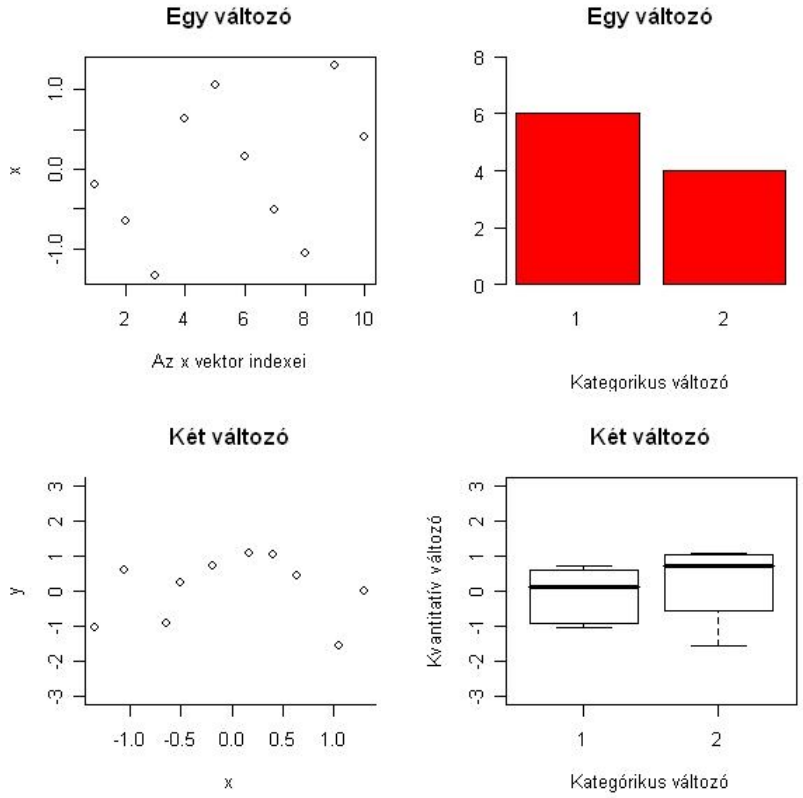


6.8 ábra

A 6.8. ábrán a **plot()** függvény a bemeneti **x** vektor alapján jelenít meg 10 pontot. A pontok megjelenítéséhez szükséges két koordináta közül most csak az egyik az *y* koordináta áll rendelkezésre, ezt éppen az **x** vektor elemei alkotják. Az *x* koordináták az **x** vektor indexeiből állnak, amely egy 10 elemű vektor esetén az 1,2, …,10 elemeket jelenti. Az első ábrázolt pont koordinátái ennek megfelelően az (1, 21), a második (2, 22) és így tovább, az utolsó (10, 30).

A **plot()** függvény paraméterében a numerikus vektorok helyett faktorok is szerepelhetnek. Ha egyetlen faktorral hívjuk, akkor oszlopdiagramot kapunk, ha faktorral és numerikus vektorral, akkor dobozdiagramot kapunk. Ezeket az eseteket foglalja össze a 6.9. ábra.

```
> layout(matrix(1:4,ncol=2, byrow=T))
> x<-rnorm(10)
> y<-rnorm(10)
> f<-gl(2,3,10)
> plot(x, main="Egy változó", xlab="Az x vektor indexei")
> plot(f, main="Egy változó", sub="Kategorikus változó",
+         las=1, col="red", pch=16, , ylim=c(0, 8))
> plot(cbind(x,y), main="Két változó",  ylim=c(-3,3), bty="l")
> plot(y~f, main="Két változó", xlab="Kategórikus változó",
+           ylab="Kvantitatív változó", ylim=c(-3,3))
```



6.9. ábra

A 6.9. ábra megjelenítéséhez a **plot()** függvényekben különböző paramétereket használhatunk. Nézzük a legfontosabbakat sorban:

main Az ábra címe, felül, középen, kielmelten fog megjelenni.

sub Az ábra alcíme, az ábra alján jelenik meg.

xlab, ylab Az *x* és *y* tengely felirata, ha elhagyjuk, akkor az ábrázolt objektum nevét olvashatjuk.

axes A tengelyek és a rajzterületet szegélyeinek megjelenítését szabályozzuk. Alapértelmezett értéke a TRUE. Ha FALSE értéket adunk, akkor az **axis()** és a **box()** függvényekkel a tengelyeket és a szegélyt később is megrajzolhatjuk.

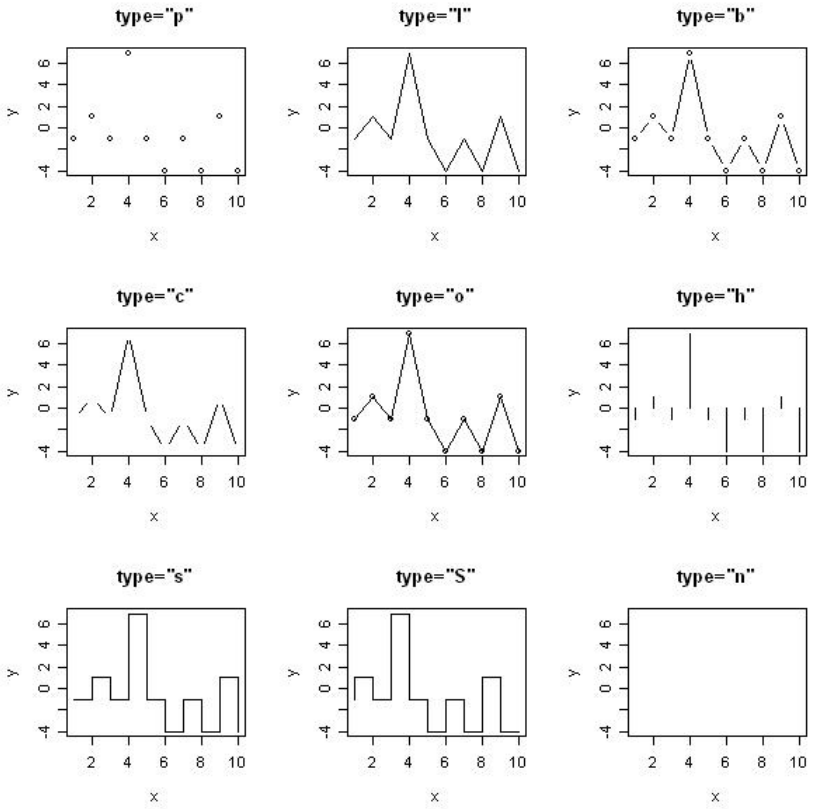
bty A rajzterület szegélyezését állíthatjuk be. Értéke egy karakter lehet, melyek jelentése a következő:

”o”	teljes keretet rajzol
”l”	bal oldalt és lent lesz csak szegély
”7”	fent és jobboldalt rajzol keretet
"c",	alul, baloldalt és felül lesz rajzol szegélyt
"u",	baloldalt, alul és jobboldalt kapunk keretet
"]"	felül, jobboldalt és alul jelenik meg szegély
”n”	nem jelenít meg szegélyt

xlim, ylim	Az x és y tengelyek ábrázolási tartományát határozzuk meg. Értékeik egy-egy kételemű numerikus vektor, melyek az ábrázolt intervallum alsó és felső határait adják.
fg, bg,	Az ábra színeinek megadására használt paraméterek.
col, **col.axis,** **col.lab,** **col.main,** **col.sub**	
las	Az *x* és *y* tengelyek címkéi lehetnek párhuzamosak és merőlegesek a tengelyeikre nézve. A lehetséges eseteknek megfelelően a paraméter értéke lehet: 0 párhuzamosak a tengelyükkel (alapértelmezés) 1 *x*-re párhuzamos, *y*-ra merőleges 2 merőlegesek a tengelyükre 3 *x*-re merőleges, *y*-ra párhuzamos
pch	A kirajzolt pont határozza meg. Értéke vagy egy karakter és ekkor az lesz a megjelenített pont alakja, vagy tipikusan egy skalár 0 és 25 között.
lty	
cex, **cex.axis,** **cex.lab,** **cex.main,** **cex.sub**	
family, **font, font.axis,** **font.lab,** **font.main,** **font.sub**	

A **plot()** függvény fontos argumentuma a **type**, amelyikkel jelentősen módosíthatjuk azonos adatok esetén is a megjelenítést. A lehetséges értékeket és a hozzájuk tartozó megjelenítést a 6.10. ábra tartalmazza.

```
> layout(matrix(1:9,ncol=3, byrow=T))
> x<-1:10; y<-rpois(10,lambda=5); m<-cbind(x=x,y=y-5)
> plot(m,type="p",main='type="p"')
> plot(m,type="l",main='type="l"')
> plot(m,type="b",main='type="b"')
> plot(m,type="c",main='type="c"')
> plot(m,type="o",main='type="o"')
> plot(m,type="h",main='type="h"')
> plot(m,type="s",main='type="s"')
> plot(m,type="S",main='type="S"')
> plot(m,type="n",main='type="n"')
```

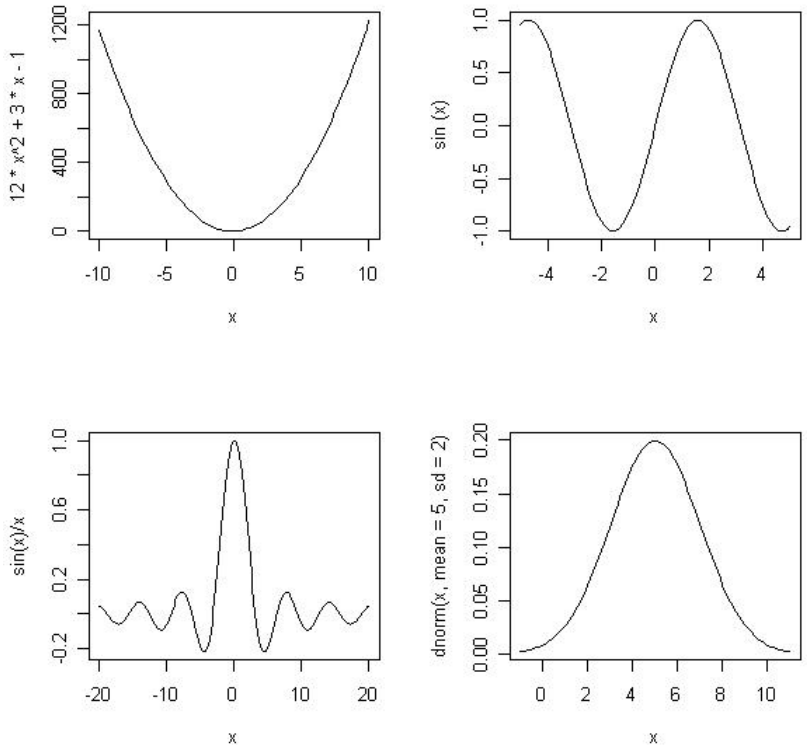


6.10. ábra

6.4.2. A curve() függvény

A **curve()** függvény segítségével egy *x*-ben változó függvény grafikonját rajzolhatjuk meg. Az első argumentum kifejezése írja le az ábrázolandó függvényt, amely R-beli matematikai függvények nevét is tartalmazhatja. A **from** és **to** paraméterek határozzák meg a megjelenítés értelmezési tartományát, az **n** paraméter pedig, hogy összesen hány behelyettesítés történjen a függvénybe. A 6.11. ábra tartalmaz néhány példát a **curve()** függvény használatára.

```
> par(mfrow=c(2,2))
> curve(12*x**2+3*x-1, from=-10, to=10)
> curve(sin, from=-5, to=5)
> curve(sin(x)/x, from=-20, to=20, n=200)
> curve(dnorm(x,mean=5,sd=2),from=-1, to=11)
```



6.11. ábra

6.4.3. A hist() függvény

A **hist()** függvény segítségével hisztogramot rajzolhatunk, azaz bizonyos osztályintervallumokhoz tartozó téglalapokat láthatunk egymás mellet, melyek területükkel jelzik az osztályintervallum (relatív)gyakoriságát. Az osztályintervallumok létrehozását a bemenő adatvektor alapján a **hist()** függvény is végezheti, de tipikusabb a **breaks** paraméterrel vezérelni az intervallumok létrehozását. A **hist()** függvény legfontosabb argumentumai a következők:

breaks Az értéke lehet: (1) skalár (ekkor az osztályintervallumok számát jelenti), (2) numerikus vektor (az osztályintervallumok határai), (3) karaktersorozat (az osztályhatárok megállapítására használt algoritmus neve (`"Sturges"`, `"Scott"`, `"FD"` / `"Freedman-Diaconis"`) vagy (4) egy saját függvény neve, amely az osztályokat létrehozza.

freq Gyakoriság (`TRUE`) vagy relatív gyakoriság (`FALSE`) megjelenítése között választhatunk a paraméter segítségével. Alapértelmezetten `TRUE`, de ha nem azonos nagyságúk az osztályintervallumok, akkor `FALSE` lesz az értéke.

right Az osztályintervallumok bal vagy jobb oldali zártságát szabályozhatjuk vele. Alapértelmezett értéke a `TRUE`, ami jobb oldali zártságot jelent, azaz ezek az értékek még az osztályhoz

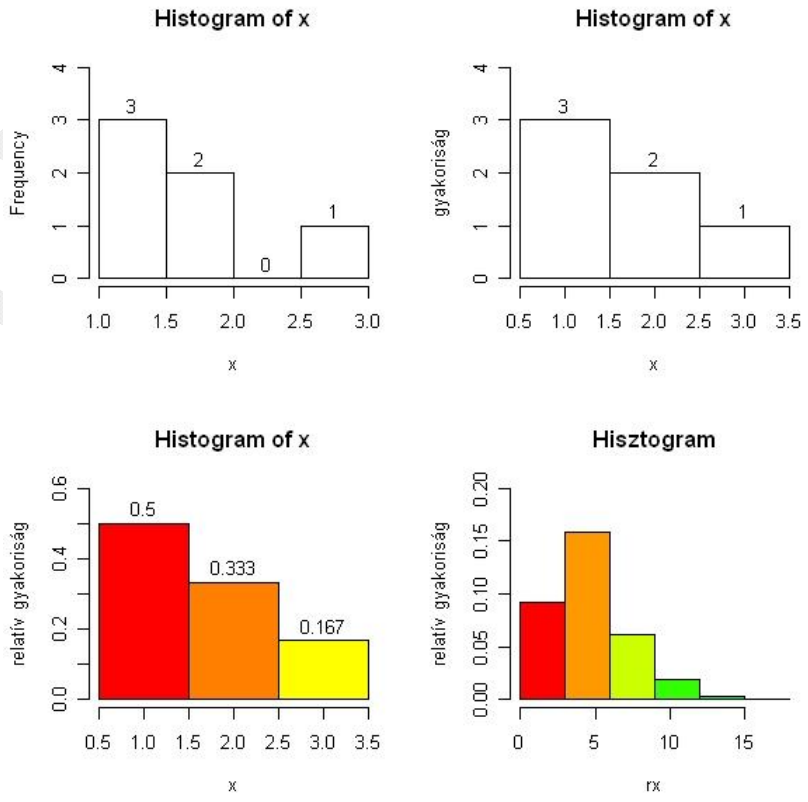
tartoznak.

include.lowest Amennyiben a **right** értéke `TRUE` és a legkisebb érték az első osztályintervallum bal szélső értéke, akkor az argumentum `TRUE` értéke esetén (ez az alapértelmezett) az osztályintervallumhoz sorolja a függvény ezt az elemet. Ha a **right** értéke `FALSE` (jobb oldali nyitottság), akkor mindez a legnagyobb értékre és az utolsó osztály jobb szélső értékére értendő.

plot Ha megváltoztatjuk az alapértelmezett `TRUE` értéket `FALSE`-ra, akkor nem történik meg a hisztogram kirajzolása, hanem a visszatérési értékkel (ami egy lista) dolgozhatunk tovább.

labels Az oszlopok tetején (relatív)gyakoriságot jelenít meg.

```
> par(mfrow=c(2,2))
> x<-c(1,1,1,2,2,3)
> rx<-rpois(120, lambda=5)
> hist(x,ylim=c(0,4),labels=T)
> hist(x,breaks=c(.5,1.5,2.5,3.5),right=TRUE,ylim=c(0,4),
+                labels=T,ylab="gyakoriság")
> hist(x,freq=F,breaks=c(.5,1.5,2.5,3.5),right=TRUE,ylim=c(0,.6),
+                labels=T,ylab="relatív gyakoriság",col=rainbow(12))
> hist(rx,freq=F,breaks=seq(0,20,3),ylab="relatív gyakoriság",
+         col=rainbow(10),main="Hisztogram",ylim=c(0,.2))
```



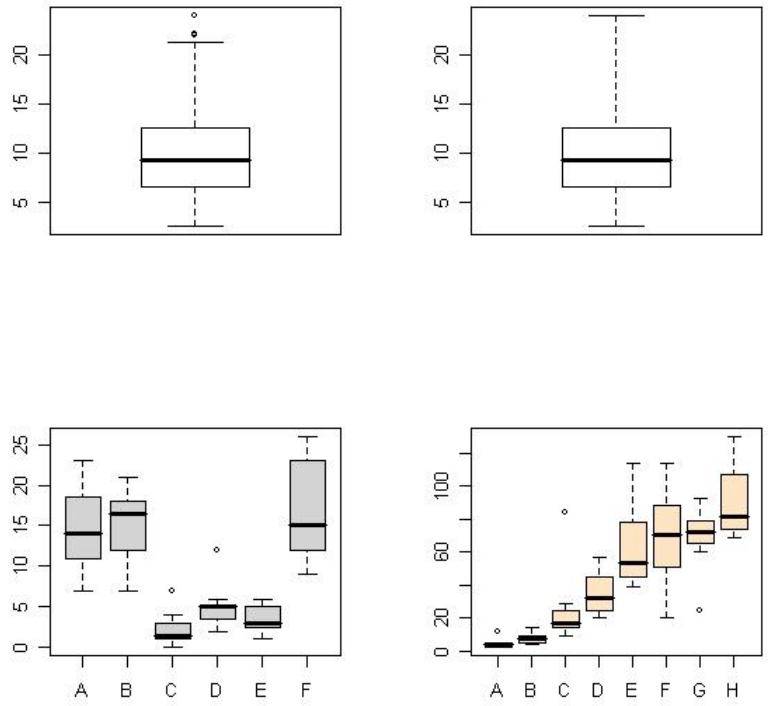
6.12. ábra

6.4.4. A boxplot() függvény

A **boxplot()** függvény a hisztogramhoz hasonlóan az adatmegtekintés vizualizálását segíti ún. dobozdiagramok megjelenítésével. A dobozdiagramról a minta terjedelme (legkisebb és legnagyobb értéke), az alsó- és felsőkvartilis, illetve a medián olvasható le. Általában több csoport adatait hasonlítjuk össze a dobozdiagram segítségével.

A **boxplot()** függvény **range** paraméterével a kiugró értékek kezelését befolyásolhatjuk.

```
> par(mfrow=c(2,2))
> rx<-rchisq(100,10)
> boxplot(rx)
> boxplot(rx,range=0)
> boxplot(count~spray, data=InsectSprays, col="lightgray")
> boxplot(decrease~treatment, data=OrchardSprays, col="bisque")
```



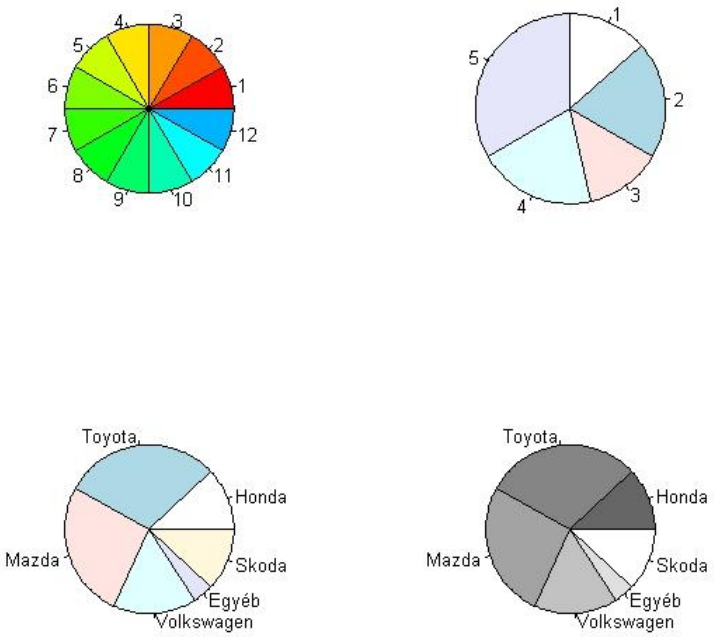
6.13. ábra

6.4.5. A pie() függvény

A kördiagramot a **pie()** függvénnyel készíthetünk. A **radius** argumentum a kör sugarát adja meg, a **clockwise** a körcikkek körbejárási irányát adja meg.

```
> par(mfrow=c(2,2))
```

```
> pie(rep(1,12), col = rainbow(20))
> pie(c(2,3,2,3,5), radius=0.9, clockwise = T)
> pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
> names(pie.sales) <- c("Honda", "Toyota", "Mazda", "Volkswagen", "Egyéb",
"Skoda")
> pie(pie.sales)
> pie(pie.sales, col = gray(seq(0.4,1.0,length=6)))
```



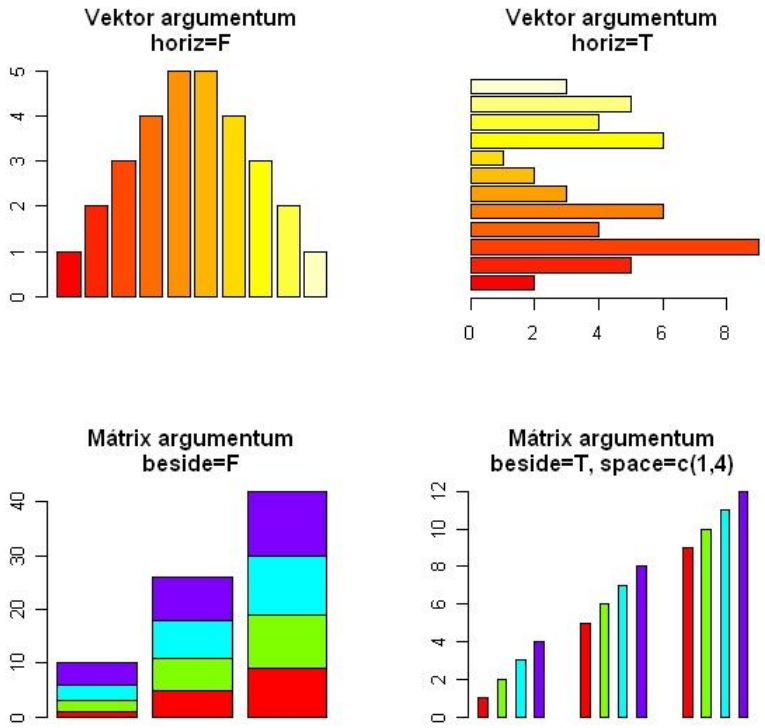
6.14. ábra

6.4.6. A barplot() függvény

A **barplot()** függvény oszlopdiagram létrehozását teszi lehetővé. Numerikus vektorból vagy mátrixból hozhatunk létre oszlopdiagramot. Használhatjuk a **horiz** argumentumot, amellyel fekvő vagy álló oszlopdiagramot hozhatunk létre vagy a **space**-t, amellyel az oszlopok egymás közötti távolságát adhatjuk meg. Mátrix bemenő paraméter esetén az egy oszlopban lévő elemeket egy csoportpa sorolja a **box()** függvény. A **beside** argumentum `FALSE` értéke mellett a csoportok egymás mellett, a csoportok elemei pedig egymás felett jelennek meg. A **beside** TRUE értéke mellett a csoportok elemei egymás mellett szerepelnek, de a csoportok között nagyobb lesz a távolság. A fentieket szemlélteti a 6.15. ábra.

```
> par(mfrow=c(2,2))
> barplot(c(1:5,5:1),main="Vektor argumentum\nhoriz=F",col=heat.colors(10))
> barplot(rpois(12, lambda=5),main="Vektor argumentum\nhoriz=T",
+                             col=heat.colors(12),horiz=T)
> m<-matrix(1:12,ncol=3); m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
```

```
[4,]    4    8   12
> barplot(m,main="Mátrix argumentum\nbeside=F",col=rainbow(4))
> barplot(m,main="Mátrix argumentum\nbeside=T, space=c(1,4)",
+           beside=T, space=c(1,4), col=rainbow(4))
```



6.15. ábra

6.5. Alacsony-szintű rajzfüggvények

Az alacsony-szintű függvények segítségével meglévő ábráinkhoz adhatunk hozzá elemeket. Szöveget írhatunk a külső vagy belső margóra az **mtext()**, az ábraterületre pedig a **text()** függvénnyel. Az ábraterültre pontokat helyezhetünk el **points()**, vonalakat a **lines()** és az **abline()** függvénnyel. Továbbá rajzolhatunk téglalapot a **rect()**, sokszöget a **polygon()** függvénnyel. Az ábránkra tengelyeket rajzolhatunk az **axis()**, szegélyeket a **box()**, jelmagyarázatot a **legend()**, címet és alcímet a **title()** függvénnyel.

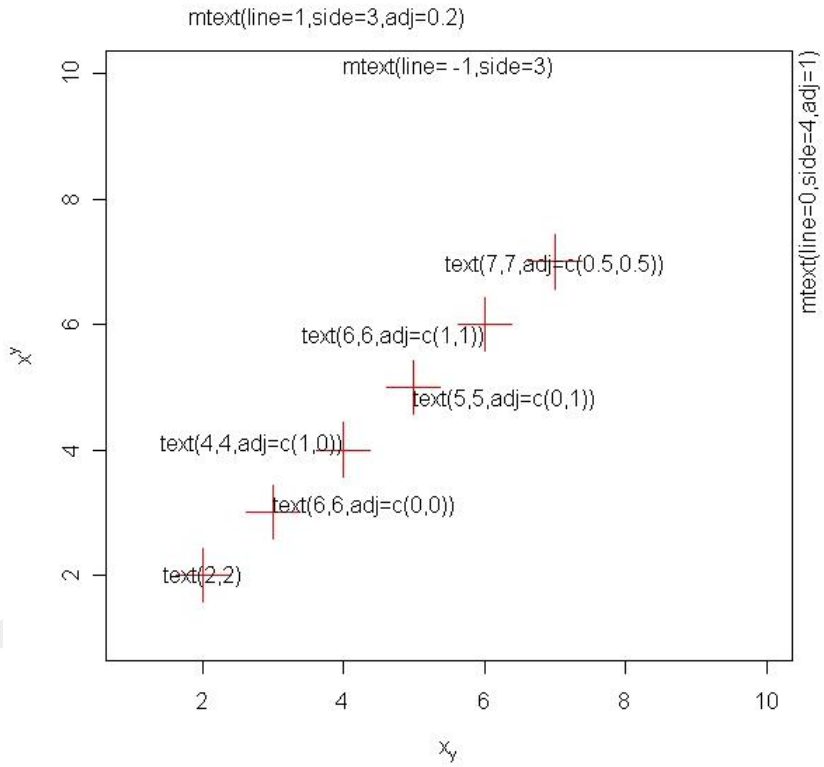
6.5.1. Szöveg elhelyezése

Az **mtext()** függvényben a szöveg helyét a **line** argumentum határozza meg: az alapértelmezett `0` segítségével a margóra tudunk írni , de kintebb vagy beljebb is tudunk írni a margóhoz képest, pozitív vagy negatív érték megadása esetén. A **text()** függvényben a szöveg helyét a felhasználó koordinátarendszerében értendő **x** és **y** argumentum értékei határozzák meg.

Mindkét függvény esetében az **adj** argumentum határozza meg a szöveg igazítását, ami tipikusan 0 és 1 közötti numerikus érték, de a legtöbb grafikus eszközön kibővíthetjük ezt a tartományt. A **text()** függvény esetében ez a paraméter kételemű vektor is lehet, az első érték

az *x* a második az *y* irányában határozza meg az igazítást. Az **mtext()** függvény esetében egyetlen érték is elegendő igazításra, a másik pozíciót a **line** argumentum értékéből következik. Mindezt a 6.16. ábráról is leolvashatjuk.

```
> plot(1:10,xlab=expression(x[y]),ylab=expression(x^y), type="n")
> mtext("mtext(line= -1,side=3)",line=-1,side=3)
> mtext("mtext(line=1,side=3,adj=0.2)",line=1,side=3,adj=0.2)
> mtext("mtext(line=0,side=4,adj=1)",line=0,side=4,adj=1)
> points(2:7,2:7,pch=3, cex=4,col="red")
> text(2,2,"text(2,2)")
> text(3,3,"text(6,6,adj=c(0,0))",adj=c(0,0))
> text(4,4,"text(4,4,adj=c(1,0))",adj=c(1,0))
> text(5,5,"text(5,5,adj=c(0,1))",adj=c(0,1))
> text(6,6,"text(6,6,adj=c(1,1))",adj=c(1,1))
> text(7,7,"text(7,7,adj=c(0.5,0.5))",adj=c(.5,.5))
```



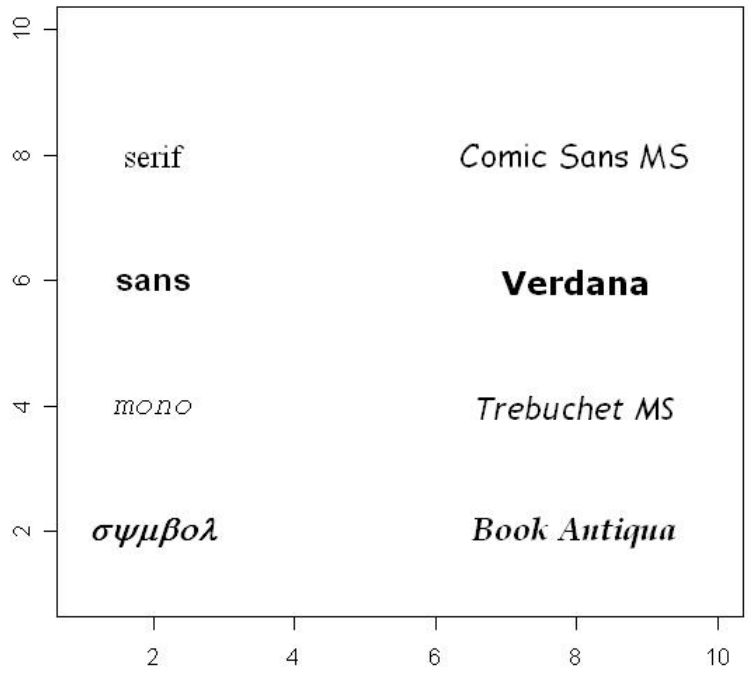
6.16. ábra

Az *x* és *y* tengely feliratában alsó és felső indexet is használhatunk. Ehhez az **expression()** függvényt használjuk fel, amellyel kifejezéseket hozhatunk létre az R-ben. Alsó indexek írására a szögletes zárójelet (pl. `x[y]`), felső indexre a hatványozás jelét (pl. `x^y`) használhatjuk.

A megjelenítendő szöveg betűtípusát is megváltoztathatjuk a **font** és a **family** argumentumok segítségével. A **font** értékei: 1=normál, 2=félkövér,3=dőlt és 4=félkövér/dőlt. A **family** argumentum egy betűcsalád nevét tartalmazhatja, alapértelmezett értékei a ”serif”, ”sans”, ”mono” és ”symbol” lehetnek. A számítógépen egyéb betűcsaládok a **windowsFonts()** és a **windowFont()** függvények használata után lesznek elérhetőek az R-ben.

```
> windowsFonts( font.comic = windowsFont("Comic Sans MS"),
```

```
+                font.verdana = windowsFont("Verdana"),
+                font.trebuchet = windowsFont("Trebuchet MS"),
+                font.book.antiqua = windowsFont("Book Antiqua"))
> plot(1:10,type="n",xlab="",ylab="")
> par(cex=1.4)
> text(2,8,"serif",family="serif",font=1)
> text(2,6,"sans",family="sans",font=2)
> text(2,4,"mono",family="mono",font=3)
> text(2,2,"symbol",family="symbol",font=4)
> text(8,8,"Comic Sans MS",family="font.comic",font=1)
> text(8,6,"Verdana",family="font.verdana",font=2)
> text(8,4,"Trebuchet MS",family="font.trebuchet",font=3)
> text(8,2,"Book Antiqua",family="font.book.antiqua",font=4)
```



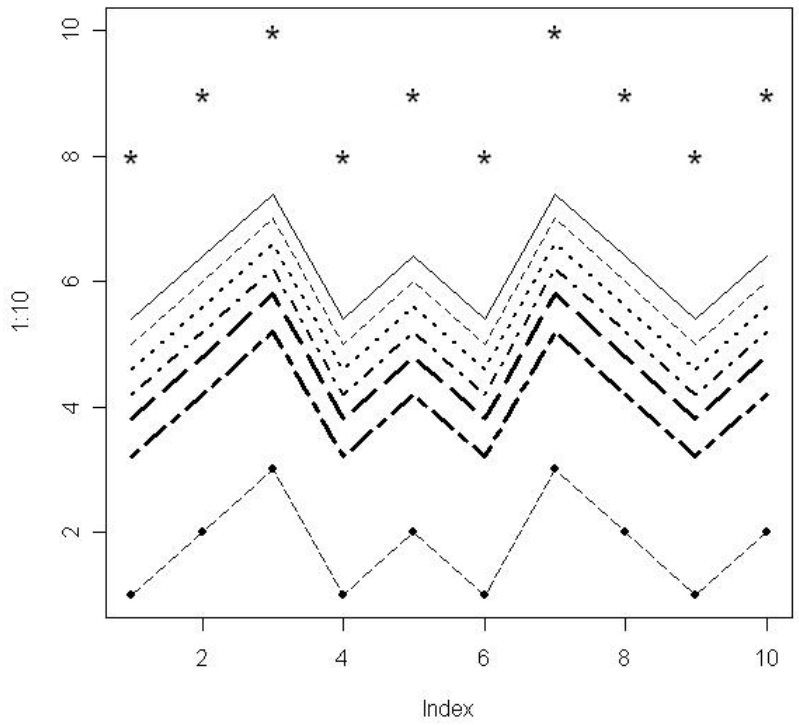
6.17. ábra

6.5.2. Pontok, vonalak

A **points()** függvény **x** és **y** paramétere a megjelenítendő pontok koordinátáit tartalmazzák, míg a **lines()** függvény ugyanezen paraméterek mellett egyenes szakaszokkal köti össze pontokat. A megjelenítendő pontok alakját a **pch**, a vonalak típusát pedig a **lty** paraméter határozza meg.

```
> plot(1:10, type="n")
> x<-c(1,2,3,1,2,1,3,2,1,2)
> points(1:10,x+7, pch="*",cex=2)
> lines(1:10,x+4.4,lty=1)
> lines(1:10,x+4,  lty=2, lwd=1.5)
> lines(1:10,x+3.6,lty=3, lwd=2)
> lines(1:10,x+3.2,lty=4, lwd=2.5)
> lines(1:10,x+2.8,lty=5, lwd=3)
```

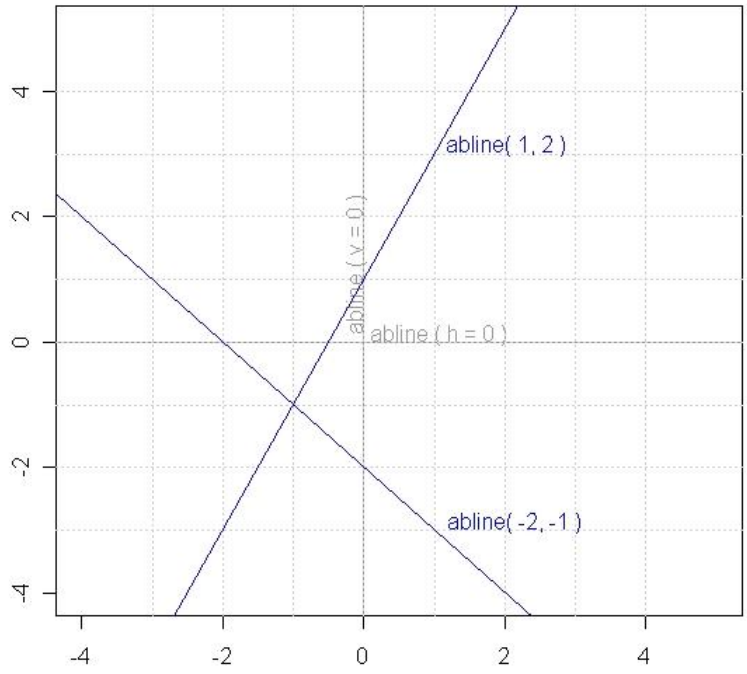
```
> lines(1:10,x+2.2,lty=6, lwd=3.5)
> points(1:10,x,   pch=16)
> lines(1:10,x,    lty=5)
```



6.18. ábra

Egyeneseket az **abline()** segítségével is létrehozhatunk. Legfontosabb argumentumok az **a** és a **b**, amelyek az y tengellyel való metszéspontot és az egyenes meredekséget adják meg. Függőleges és vízszintes egyeneseket is rajzolhatunk a **v** és **h** paraméterek megadásával.

```
> plot(-4:5,-4:5, type="n",xlab="",ylab="")
> abline(h=0, v=0, col = "gray60")
> text(0.1,0, "abline ( h = 0 )",col="gray60",adj=c(0,-0.1))
> text(0,0.1, "abline ( v = 0 )",col="gray60",adj=c(0,-0.1),srt=90)
> abline(h = -3:4, v = -3:4, col = "lightgray", lty=3)
> abline(a=1, b=2, col="blue")
> text(1,3,"abline( 1, 2 )", col="blue",adj=c(-.1,-.1))
> abline(a=-2, b=-1, col="blue")
> text(1,-3,"abline( -2, -1 )", col="blue",adj=c(-.1,-.1))
```

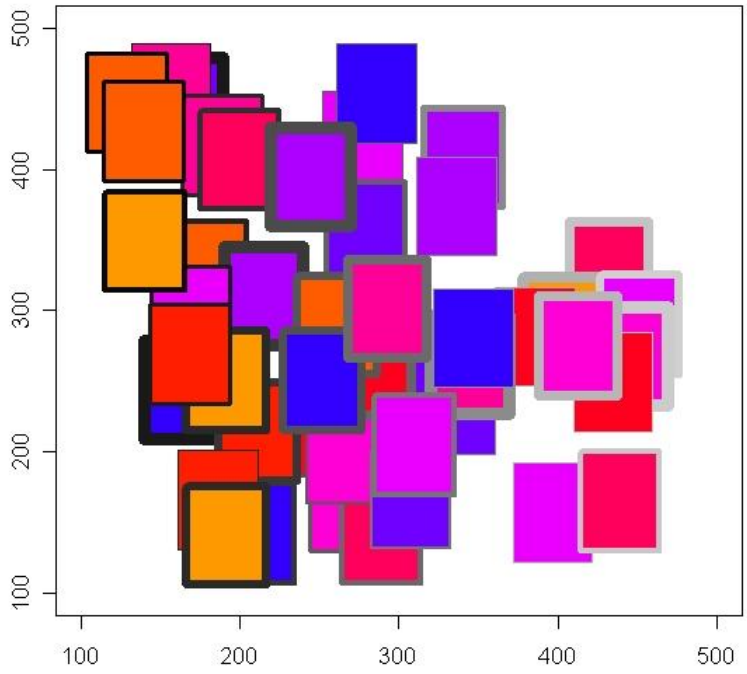


6.19. ábra

6.5.3. Téglalapok, poligonok, nyilak

A **rect()** függvénnyel téglalapot rajzolhatunk az ábraterületre. A téglalapok bal alsó és jobb felső sarkának *x* és *y* koordinátáját kell megadnunk, ezek rendre: **xleft**, **ybottom**, **xright**, **ytop**.

```
> plot(100:500,100:500, type="n",xlab="",ylab="")
> x<-runif(50)*350
> y<-runif(50)*330
> rect(100+x, 100+y, 150+x, 170+y, col=rainbow(11,start=.7,end=.1),
+                                  border=gray(x/400),lwd=x%%10)
```

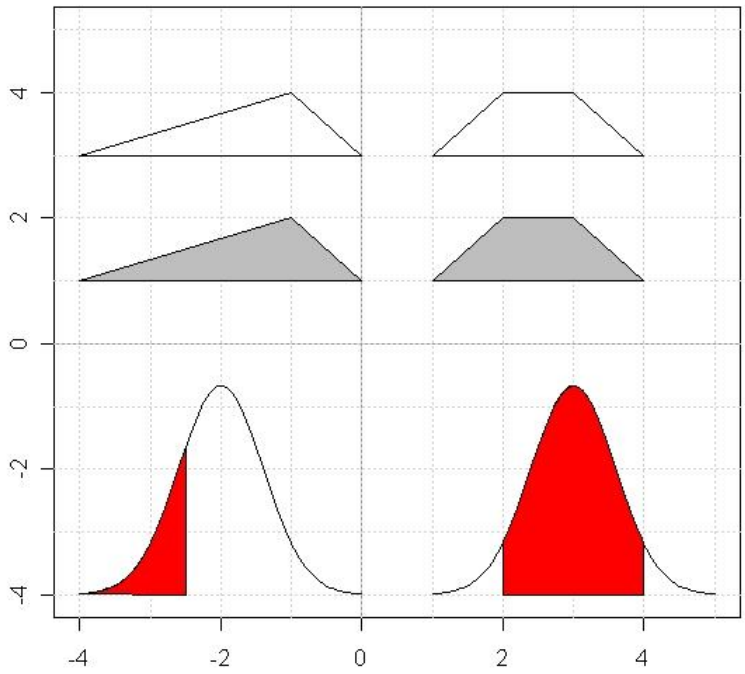


6.20. ábra

A **polygon()** függvény tetszőleges egyenesekkel határolt síkidomok létrehozásáért felelős. Az **x** és **y** paraméterében várja a csúcspontok koordinátáit. A lenti példában a vonalak rajzolására használatos **segments()** függvény is bemutatásra kerül, azt láthatjuk, hogy a **polygon()** függvény argumentuma, hogyan feleltethető meg a **segments()** függvény bemenő paraméterének.

```
my_segments<-function(x,y) {
  i<-1:(length(x)-1)
  segments(x[i],y[i],x[i+1],y[i+1])
  segments(x[length(x)],y[length(x)],x[1],y[1])
}

plot(-4:5,-4:5, type="n",xlab="",ylab="")
abline(h=0, v=0, col = "gray60")
abline(h = -4:5, v = -4:5, col = "lightgray", lty=3)
x<-c(-4,-1,0)
y<-c(1,2,1)
polygon(x,y,col="gray")
y<-y+2
my_segments(x,y)

x<-c(1,2,3,4)
y<-c(1,2,2,1)
polygon(x,y,col="gray")
y<-y+2
my_segments(x,y)

my_func<-function(x) {
```

```
  return (5*dnorm(x,mean=-2,sd=.6)-4)
}
curve(my_func,from=-4,to=0, add=T)
xp<-seq(-4,-2.5,.01)
polygon(c(xp,-2.5),c(my_func(xp),-4),col="red")
curve(my_func(x-5),from=1,to=5, add=T)
xp<-seq(2,4,.01)
polygon(c(2,xp,4),c(-4,my_func(xp-5),-4),col="red")
```



6.21. ábra

6.5.4. Egyéb kiegészítők

Ha ábránk létrehozása során korábban nem gondoskodtunk cím/alcím, tengelyek, szegély vagy jelmagyarázat megrajzolásáról, akkor alacsony-szintű rajzfüggvényekkel utólag is hozzáadhatjuk ezeket a grafikánkhoz.

A **tilte()** függvény fontosabb argumentumai a következők:

main, sub	Az ábra címének és alcímének meghatározására használjuk.
xlab, ylab	Tengelyfeliratok hozzáadása.
line	Az alapértelmezett szövegpozíciót írhatjuk felül, ha meghatározzuk a szövegsor sorszámát.
outer	Az ábra címét a külső margóra írhatjuk ha `TRUE` értéket adunk meg.

A **box()** függvény legfontosabb argumentuma:

which A bekeretezendő terület meghatározása. Értékei lehetnek:

`"plot"`	rajzterület szegélyezése
`"figure"`	ábraterület szegélyezése
`"inner"`	több ábra esetén az ábraterületeket fogja össze
`"outer"`	az eszközfelület szegélyezése

A tengelyek megjelenítését szabályzó grafikai paraméterek, az **axis()** függvény használata során:

xaxt, yaxt Az *x* és *y* tengely kirajzolását tilthatjuk meg, ha `"n"` értékkel látjuk el. Alapértelmezett az `"s"`, ekkor megrajzolja az illető tengelyt. (Hasonló a szerepe, mint az **axes** argumentumnak.)

mgp=`c(3,1,0)` A tengely egyes részeinek a rajzterület szélétől mért távolságát meghatározó 3 elemű vektor. Az első érték a tengelyfelirat, a második a tengelycímkék, a harmadik magának a tengely vonalának a margóját adja meg.

lab Háromelemű numerikus vektor, amelynek az első két elemével az *x* és az *y* tengely beosztásainak a számát határozhatjuk meg. A tényleges osztásszám az általunk megadott értéktől eltérhet. A harmadik elemet nem veszi figyelembe az R.

xaxp, yaxp Leginkább lekérdezésre szánt 3 elemű numerikus vektor. Az első és második elem a két szélső beosztás értéket adja, a harmadik elem pedig az osztályok számát.

xaxs, yaxs Az alapértelmezett `"r"` érték mellett a tengelyek által átfogott intervallum az ábrázolandó adatokból kiszámolt vagy az **xlim**, **ylim** paraméterekből kapott intervallumnál 4%-al nagyobb lesz. Pontos egyezéshez az `"i"` értéket kell megadnunk.

tck, tcl, las, xlim, ylim, log, xlog, ylog Korábban már szerepletek.

Az **axis()** függvény fontos argumentumai:

side A rajzolandó tengely helye: 1=lennt, 2=balra, 3=fennt, 4=jobbra..

at=`NULL` A beosztások helyét határozza meg. Ha az értéke `NULL`, akkor automatikusan számolja az R, egyébként a megadott numerikus vektor elemeinek megfelelő helyre kerül beosztás.

labels A beosztások címkéit határozza meg.

Jelmagyarázat többnyire a rajzterületen belül foglal helyet, a pozícióját is a felhasználói koordinátákban kell megadni. A jelmagyarázat létrehozását számos paraméter segíti, de az

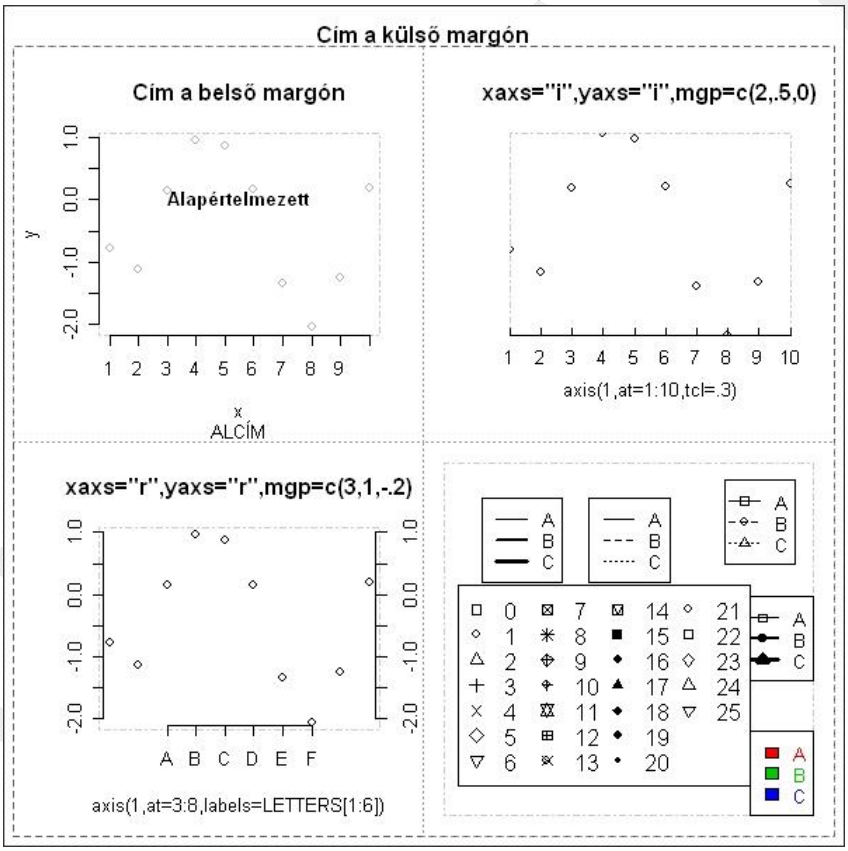
ábrán szereplő és a jelmagyarázatban használt jelek közötti összhangot nekünk kell megteremteni, az R semmilyen ellenőrzést nem végez ezzel kapcsolatban.

A **legend()** függvény argumentumai:

x, y	A jelmagyarázat bal felső sarkának a koordinátái a rajzterület koordintáiban mérve. Az **x** argumentum karakteres értékével az igazítást adhatjuk meg: `"bottomright"`, `"bottom"`, `"bottomleft"`, `"left"`, `"topleft"`, `"top"`, `"topright"`, `"right"` and `"center"`
legend	A jelmagyarázat szövegét tartalmazó karakteres vektor.
fill, lty, lwd, pch	A jelmagyarázat szövegét és az ábrán használt jeleket összekötő szimbólumok létrehozására használható argumentumok.
inset=`0`	Ha kulcsszavakat használunk az igazításra, akkor a margótól mért távolságot itt adhatjuk meg.
merge	Ha pontokat és vonalakat is használunk a jelmagyarázatban, akkor `TRUE` érték mellett ezeket kombinálja a megjelenítéskor.
horiz=`FALSE`	A jelmagyarázat tájolását befolyásoló logikai paraméter.
ncol=`1`	A jelmagyarázat oszlopainak a számát határozza meg.
text.col, bg	A szöveg színe és a háttérszín jelmagyarázatban.

```
> op<-par()
> par(oma=c(.5,.5,2,.5))
> par(mfrow=c(2,2))
> x<-rnorm(10)

> plot(x,main="",axes=F,xlab="",ylab="",col=gray(0.7))
> text(5.5,0,"Alapértelmezett",font=2)
> title(main="Cím a külső margón",outer=T,line=0.2)
> title(main="Cím a belső margón")
> title(sub="ALCÍM")
> title(xlab="x",ylab="y")
> box("outer",lty=1,col=grey(.2))
> box("inner",lty=2,col=grey(.4))
> box("figure",lty=3,col=grey(.6))
> box("plot",lty=4,col=grey(.8))
> axis(1, at=1:10);axis(2)

> par(xaxs="i")
> par(yaxs="i")
> par(mgp=c(2,.5,0))
> plot(x,main='xaxs="i",yaxs="i",mgp=c(2,.5,0)',axes=F,
 +       xlab="axis(1,at=1:10,tcl=.3)",ylab="")
> box("plot",lty=4,col=grey(.8))
> axis(1,at=1:10,tcl=.3)
>
> par(xaxs="r")
> par(yaxs="r")
>
> par(mgp=c(3,1,-.2))
> plot(x,main='xaxs="r",yaxs="r",mgp=c(3,1,-
.2)',axes=F,xlab="axis(1,at=3:8,labels=LETTERS[1:6])",ylab="")
```

```
> box("plot",lty=4,col=grey(.8))
> axis(1,at=3:8,labels=LETTERS[1:6])
> axis(2)
> axis(4)
>
> par(mgp=c(3,1,0))
>
> par(mar=c(1,1,1,1))
> plot(1:10,main="",axes=F,xlab="",ylab="",type="n")
> box("figure",lty=3,col=grey(.6))
> box("plot",lty=4,col=grey(.8))
> legend("topleft",LETTERS[1:3],lw=1:3,inset=.1)
> legend("top",LETTERS[1:3],lty=1:3,inset=.1)
> legend("topright",LETTERS[1:3],lty=1:3,pch=0:3,inset=.05)
> legend("right",LETTERS[1:3],lw=1:3,pch=0:3)
> legend("bottomright",LETTERS[1:3],fill=2:4,text.col=2:4)
> legend(1,7, as.character(0:25), pch = 0:25,ncol=4,cex=1.2)
>
>
> par(op)
```



6.22. ábra

6.6. Interaktív grafikus függvények

Az R hagyományos grafikus rendszere elsősorban statikus ábrák létrehozását támogatja, a kész grafikán további interaktív műveletekre csak korlátozottan van lehetőség. Ezek közül most a **locator()** és az **identify()** függvényeket tekintjük át.

A **locator()** függvény lehetővé teszi, hogy a felhasználó a rajzterületen az egér bal gombjával kijelöljön egy vagy több pontot. A függvény visszatérési értéke egy lista, amely a megjelölt pontok *x* és *y* koordinátáit sorolja fel.

```
> legend(locator(1), as.character(0:25), pch = 0:25)
```

A **locator()** függvény fontosabb argumentumai:

n	A meghatározandó pontok maximális számát adhatjuk meg. Alapértelmezetten 512 pont helyét kaphatjuk meg.
type	Értéke az "n", "p", "l" vagy "o" karakter valamelyike lehet. Az alapértelmezett "n" esetében az egérkattintás után nem történik megjelenítés, míg "p" és "o" hatására egy pont fog megjelenni, "l" vagy "o" estén pedig vonal fogja összekötni a megjelölt helyeket.

Az **identify()** függvény a rajzterületen megjelenő adatpontokhoz rendel címkét. A **locator()** függvényhez hasonlóan az egér bal gombjával itt is ki kell jelölni egy pontot a rajzterületen, majd a függvény a megjelölt ponthoz legközelebb lévő adatponthoz rendel egy címkét. Az **indentify()** függvény visszatérési értéke az azonosított adatpontok indexe.

```
> library(MASS)
> attach(mammals)
> plot(body,brain,log="xy")
> identify(body,brain,row.names(mammals),n=10)
[1]  2  7 32 35 36 44 46 48 52 58
```

Az **identify()** függvény fontosabb argumentumai:

x, y	Numerikus vektorok, amelyek az ábra adatpontjainak *x* és *y* koordinátáit határozzák meg.
labels	Karakteres vektor, amely egyes adatpontok címkéit tartalmazza. Az **x** és **y** argumentumokkal azonos hosszú vektor, alapértelmezetten sorszámokat tartalmaz.
pos=FALSE	Ha FALSE az értéke, akkor a függvény az azonosított adatpontok indexével fog visszatérni. Ha TRUE, akkor egy listával, amely az **ind** elemében az indexeket, **pos** elemében pedig a címke adatponthoz viszonyított helyét tartalmazza (1=lennt, 2=balra, 3=fennt, 4=jobbra, ill. 0=ekkor az **atpen**=TRUE volt)
n	A megjelelölendő pontok száma.
plot=TRUE	Logikai érték, amely ha TRUE (ez az alapértelmezett), akkor megjelenít címkéket, egyébként nem.
atpen=FALSE	A címke pozíciója TRUE érték esetén az egérkattintás helye lesz, FALSE esetén automatikusan kerül meghatározásra.
offset=0.5	A címke és az adatpont távolsága. A numerikus érték a karakterszélességhez mért. Az **atpen**=FALSE esetén nem használja az R.

tolarence=`0.25` Inch-ben mért numerikus érték, amely az egérkattintás és a „közeli” adatpont közötti maximális távolságot jelenti.

7. Matematika az R-ben

7.1. Eloszlások

Előre definiált eloszlások széles körét biztosítja számunkra az R. Mindegyik eloszláshoz négy függvény tartozik, amelyek csak az egy betűs előtagban (prefix) különböznek. A prefixként használható *d*, *p*, *q* és *r* jelentése a következő:

- *d* – az eloszlás értékeire kérdezhetünk rá: diszkrét esetben az adott érték előfordulási valószínűségét kapjuk, folytonos esetben a sűrűségfüggvény adott pontbeli értékét,
- *p* – a kommulatív eloszlás, azaz az eloszlásfüggvény adott pontbeli értékét kapjuk,
- *q* – az eloszlás kvantilisei, azaz egy adott valószínűséghez tartozó értéket kapunk,
- *r* – adott számú véletlen számot generálhatunk az adott eloszlásból.

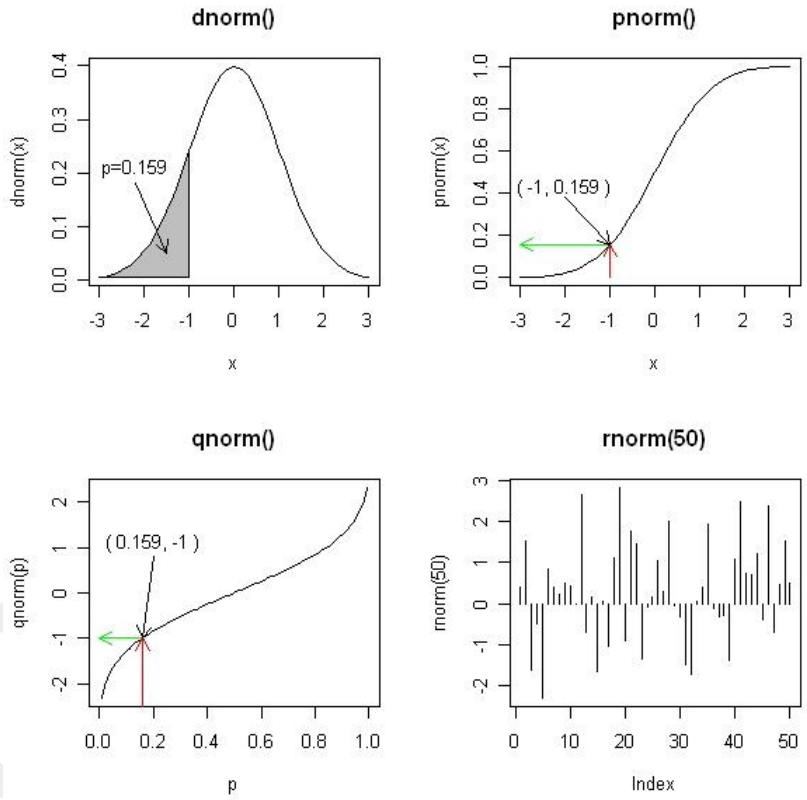
A következő táblázat foglalja össze az R-ben használható legfontosabb eloszlásokat. Ha kiválasztjuk például a normális eloszlás **...norm()** függvényvéget, akkor az előtagok használatával megkaphatjuk a már valós **dnorm()**, **pnorm()**, **qnorm()** és **rnorm()** függvényeket.

Eloszlás (*-diszkrét)	Függvény vége	Eloszlás paraméterei
béta	**...beta()**	**shape1**, **shape2**
binomiális*	**...binom()**	**size**, **prob**
Cauchy	**...cauchy()**	**location**, **scale**
χ^2	**...chisq()**	**df**
exponenciális	**...exp()**	**rate**
Fisher F	**...f()**	**df1**, **df2**
gamma	**...gamma()**	**shape**
geometriai*	**...geom()**	**prob**
hipergeometrikus*	**...hyper()**	**m**, **n**, **k**
logisztikus	**...logis()**	**location**, **scale**
lognormális	**...lnorm()**	**meanlog**, **sdlog**
negatív binomiális*	**...nbinom()**	**size**, **prob**
normális	**...norm()**	**mean**, **sd**
Poisson*	**...pois()**	**lambda**
Wilcoxon*	**...signrank()**	**n**
Student t	**...t()**	**df**
egyenletes	**...unif()**	**min**, **max**
Weilbull	**...weibull()**	**shape**, **scale**
Wilcoxon*	**...wilcox()**	**m**, **n**

A folytonos normális eloszlás és a diszkrét binomiális eloszlás segítségével bemutatjuk a fenti függvények használatát. A normális eloszláshoz tartozó négy függvény használatára látunk példát a 7.1. ábrán.

```
> # Előkészítés
> par(mfrow=c(2,2)); x<-seq(-3,3,0.1); p<-seq(0,1,0.01)
> # 1. ábra: dnorm()
> plot(x,dnorm(x),type="l",main="dnorm()")
> polygon(c(x[x<=-1],-1),c(dnorm(x[x<=-1]),dnorm(-3)),col="grey")
> text(-2.2,.2,paste("p=",round(pnorm(-1),3),sep=""),adj=c(.5,0))
```

```
> arrows(-2.2,.18,-1.5,.05,length=0.1)
> # 2. ábra: pnorm()
> plot(x,pnorm(x),type="l",main="pnorm()")
> arrows(-1,0,-1,pnorm(-1),col="red",length=0.1)
> arrows(-1,pnorm(-1),-3,pnorm(-1),col="green",length=0.1)
> text(-2,.4,paste("( -1,", round(pnorm(-1),3),")"),adj=c(.5,0))
> arrows(-2,.38,-1,pnorm(-1),length=0.1)
> # 3. ábra: qnorm()
> plot(p,qnorm(p),type="l",main="qnorm()")
> arrows(pnorm(-1),-3,pnorm(-1),-1,col="red",length=0.1)
> arrows(pnorm(-1),-1,0,-1,col="green",length=0.1)
> text(.2,1,paste("( ", round(pnorm(-1),3),", -1 )",sep=""),adj=c(.5,0))
> arrows(.2,0.8,pnorm(-1),-1,length=0.1)
> # 4. ábra: rnorm()
> plot(rnorm(50),type="h",main="rnorm(50)")
```



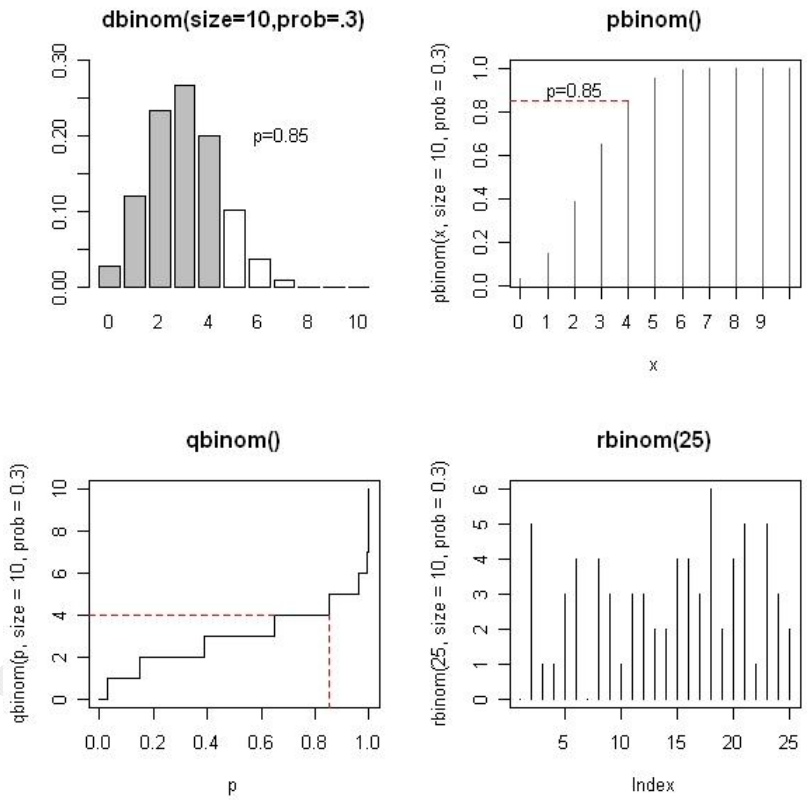
7.1. ábra

A **dnorm()** függvény alapértelmezés szerint a standard normális eloszlás sűrűségfüggvényének értékeit szolgáltatja. A **mean** és az **sd** argumentumokkal tetszőleges paraméterű normális eloszlást definiálhatunk.

A binomiális eloszláshoz tartozó négy függvényre a 7.2. ábrán látunk példát.

```
> # Előkészítés
> par(mfrow=c(2,2)); x<-0:10; p<-seq(0,1,0.01)
> # 1. ábra: dbinom()
> barplot(dbinom(x,size=10,prob=.3),main="dbinom(size=10,prob=.3)",
+ names.arg=0:10,col=c(rep("gray",5),rep("white",5)),ylim=c(0,.3))
> text(9,.2,paste("p=",round(pbinom(4,size=10,prob=.3),3),sep=""))
```

```
> # 2. ábra: pbinom()
> plot(x,pbinom(x,size=10,prob=.3),type="h",main="pbinom()",
+        axes=F,col=gray(.4))
> box(); axis(1,at=0:10); axis(2)
> segments(4, pbinom(4,size=10,prob=.3),
+          -1,pbinom(4,size=10,prob=.3),lty=2,col="red")
> text(2,pbinom(4,size=10,prob=.3),
+        paste("p=",round(pbinom(4,size=10,prob=.3),3),sep=""),
+        adj=c(.5,-.3))
> # 3. ábra: qbinom()
> plot(p,qbinom(p,size=10,prob=.3),type="s",main="qbinom()")
> segments(.85,-1,.85,4,lty=2,col="red")
> segments(pbinom(3,size=10,prob=.3),4,-1,4,lty=2,col="red")
> # 4. ábra: rbinom()
> plot(rbinom(25,size=10,prob=.3),type="h",main="rbinom(25)")
```



7.2. ábra

8. Statisztika az R-ben

Az R és a statisztika elválaszthatatlan egymástól, így néhány R példán keresztül mi is áttekintjük a legfontosabb statisztikai eljárásokat. Ennek a fejezetnek a feldolgozása statisztikai alapismereteket igényel.

8.1 Középértékek

[…]

9. Az R programozása

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

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

Az R a többi programozási nyelvhez hasonlóan a bonyolultabb problémák megoldására speciális szerkezeteket is biztosít. Alapvetően két feladatot kell megoldani, a feltételes programvégrehajtást és az utasítások többszöri végrehajtását.

9.1. Blokk utasítás

Az utasítások csoportosíthatók (blokkba szervezhetők) a kapcsos zárójelek (”{” és ”}”) segítségével következő módon:

```
{ utasítás1; utasítás2; …; utasításn }
```

Az így kapott szerkezet a továbbiakban egyetlen utasításnak tekinthető, és tetszőleges helyen, ahol utasítás egyáltalán megjelenhet, a fenti blokk utasítás is szerepelhet. A blokk utasításban az elválasztásra használhatjuk az újsort is. A blokk utasítás értéke a benne szereplő utolsó utasítás értéke lesz.

```
> y<- { x<-10
+ x+1
+ }
> y
[1] 11
```

9.2. Feltételes utasítások

Utasítások feltételtől függő végrehajtására a feltételes utasítások használhatók. Ha egy feltétel teljesül, akkor végrehajtjuk az `utasítás1` programrészt, ha nem teljesül, akkor valami más `utasítás2` programrészt hajtunk végre.

A feltételes utasítás általános alakja a következő:

```
if (feltétel)
      utasítás1
else
      utasítás2
```

A `feltétel` egy logikai értéket szolgáltató kifejezés, amely ha `TRUE` értéket vesz fel, akkor a teljes *if* utasítás értéke az `utasítás1`, ha pedig `FALSE-t`, akkor az `utasítás2` értéke lesz. Vagyis, igaz `feltétel` esetén az `utasítás1` programrészt, hamis esetben pedig az `utasítás2` programrészt hajtjuk végre.

A `feltétel` lehet numerikus érték is, ekkor a szokásos módon konvertálja az R logikai értékre (0=`FALSE`, minden más érték=`TRUE`).

Összetett logikai kifejezések is szerepelhetnek a `feltétel` -ben, ezeket többnyire a `&&` és `||` logikai operátorokkal építjük fel. A korábban megismert elemenkénti művelet-végrehajtást támogató `&` és `|` logikai operátorokhoz képest a `&&` és `||` operátorok mindig egyetlen logikai értékkel térnek vissza:

```
> c(T,F) & c(F,T)
[1] FALSE FALSE

> c(T,F) && c(F,T)
[1] FALSE
```

A `&&` és `||` operátorok támogatják a *rövidzár* végrehajtást, vagyis csak akkor értékelik ki a második argumentumot ha az szükséges:

```
> x<-0
> 1 || (x<-1)
[1] TRUE
> x
[1] 0

> 1 && (x<-1)
[1] TRUE
> x
[1] 1
```

A feltételes utasítás rövidebb formájában a teljes *else* részt elhagyhatjuk:

```
if (feltétel)
      utasítás1
```

A feltételes utasításokat egymásba is ágyazhatjuk a következő formában:

```
if (feltétel1)
      utasítás1
else if (feltétel2)
      utasítás2
else if (feltétel3)
      utasítás3
      .
      .
      .
else if (feltételn-1)
      utasításn-1
else
      utasításn
```

Ezen túl, a feltételes utasításnak egy **ifelse()** nevű függvényváltozata is létezik. A függvény három argumentuma (**test**, **yes**, **no**) megfelel az *if* utasítás `feltétel`, `utasítás1`, és `utasítás2` részeinek, azzal az engedmény, hogy a *vektorizált* végrehajtás is megengedett:

```
> x <- 1:10
> ifelse(x<=5,"yes","no")
 [1] "yes" "yes" "yes" "yes" "yes" "no"  "no"  "no"  "no"  "no"
```

Többszörös feltételvizsgálatra a **switch()** függvényt is használhatjuk.

```
> switch(2,elso=10,masodik=20,harmadik=30)
[1] 20

> switch("elso",elso=10,masodik=20,harmadik=30)
[1] 10
```

A **switch()** első paramétere egy numerikus vagy karakteres skalár, amely segítségével a többi paraméterből választhatunk visszatérési értéket.

9.3. Ciklus utasítások

Utasítások többszöri végrehajtására három vezérlőutasítást biztosít az R: *for*, *while* és *repeat*. A ciklusutasításokban felhasználhatjuk a *next* és a *break* kulcsszavakat.

A *for* utasítás szintaxisa a következő:

```
for (változó in vektor)
     utasítás
```

A `vektor` lehet vektor vagy lista, a `változó` pedig rendre felveszi ennek az elemeit. Minden lépésben az `utasítás` is végrehajtásra kerül. Annyiszor hajtjuk végre az `utasítás` részt, ahány eleme van a `vektor` objektumnak.

```
> for(x in 1:5)
+ print(x)
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5

> x
[1] 5
```

A *for* mellékhatása, hogy a `változó` objektuma az utasítás végrehajtása után is elérhető, értéke pedig a `vektor` utolsó elemének az értéke lesz.

A *while* utasítás szintaxisa a következő:

```
while (feltétel)
     utasítás
```

A *while* utasítás végrehajtása a `feltétel` kiértékelésével kezdődik, amely ha igaz, az `utasítás` rész is kiértékelésre kerül. Ezt a `feltétel` újbóli kiértékelése követi, és az `utasítás` mindaddig kiértékelésre kerül, míg egyszer már a `feltétel` hamissá válik. A *while* utasítás értéke az utolsó ciklusban kiértékelt `utasítás` értéke lesz. Ha a `feltétel` rögtön az első esetben hamis, vagyis az `utasítás` egyszer sem hajtódik végre, akkor a *while* utasítás `NULL` értéket szolgáltat.

```
> x<-1
> while (x<=5) {
+ print(x); x<-x+1 }
```

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

> x
[1] 6
```

A *repeat* utasítás szintaxisa a következő:

```
repeat utasítás
```

A *repeat* az `utasítás` folyamatos ismétlését végzi. Ha el akarjuk kerülni a végtelen ciklust, akkor gondoskodnunk kell a kilépésről. Ezt az `utasítás`ban szereplő feltételvizsgálat és *break* utasítás oldhatja meg, így a *repeat* `utasítás` része mindig egy *blokk* utasítás.

```
> x<-1
> repeat {
+ print(x)
+ x<-x+1
+ if(x>5) break
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

A *break* utasítás mindhárom ciklusban azonos jelentésű: a ciklusból lépünk ki. A *next* utasítás a ciklus újbóli végrehajtását kezdeményezhetjük.

9.4. Függvény létrehozása

Az R-ben saját függvényt a következő szintaktikával hozhatunk létre:

```
function (argumentumlista) függvénytörzs
```

Az `argumentumlista` *formális argumentumok* vesszővel elválasztott felsorolása, de akár el is hagyhatjuk. Egy formális argumentum állhat önmagában egy változónévből vagy egyenlőségjellel kapcsolhatunk hozzá kifejezést is, ekkor a kifejezés értéke lesz a formális argumentum alapértelmezett értéke.

```
> sajat.fuggveny <- function(x,y)
+ x+y
> sajat.fuggveny(1,2)
[1] 3
```

A fenti példában az **x** és **y** formális argumentum nem rendelkezik alapértelmezett értékkel, a függvényhívás során mindkét argumentumról gondoskodnunk kell.

```
> sajat.fuggveny <- function(x=0,y=0)
+ x+y
> sajat.fuggveny()
[1] 0
```

```
> sajat.fuggveny(2)
[1] 2

> sajat.fuggveny(2,3)
[1] 5
```

Most gondoskodtunk alapértelmezett értékről minkét argumentum számára, így 0, 1 vagy 2 aktuális paraméter megadásával is hívható a függvény.

Az R-ben a '…' (három pont) speciális formális argumentumot is használhatjuk, amely tetszőleges számú aktuális argumentum megadását lehetővé teszi a függvényhívás során.

```
> sajat.fuggveny <- function(x, ...)
+ sum(x, ...)
> sajat.fuggveny()
Error in sajat.fuggveny() : argument "x" is missing, with no default

> sajat.fuggveny(1)
[1] 1

> sajat.fuggveny(1,2)
[1] 3

> sajat.fuggveny(1,2,3)
[1] 6

> sajat.fuggveny(1,2,3,4)
[1] 10
```

A fenti függvény két formális argumentuma az **x** és a **...** (három pont), amely egyedül a paraméter nélküli hívást nem teszi lehetővé, egyébként tetszőleges számú aktuális argumentummal hívhatjuk.

A fenti példákban nagyon egyszerű `függvénytörzs`eket láttunk, lényegében egyetlen utasításból álltak. A tipikus azonban az, hogy a függvények törzse egy blokkutasítás, mellyel több utasítást foghatunk össze.

[…]

Irodalomjegyzék

[1] <-...erre, erre...! (Bevezetés az R-nyelv és környezet használatába) (szerző: Solymosi Norbert)

[2] Crawley, Michael J. *The R Book*. John Wiley & Sons Ltd, England. 2007.

R feladatok

Alapok

1. Határozzuk meg az első száz természetes szám összegét!
2. Határozzuk meg a száznál kisebb páratlan természetes számok összegét!
3. Határozzuk meg az első száz páros természetes szám összegét!
4. Határozzuk meg az első száz, 3-mal osztható természetes szám összegét!
5. Határozzuk meg a száznál kisebb, 3-mal osztható természetes számok összegét!
6. Határozzuk meg az első tíz négyzetszám összegét!
7. Készítsünk egy olyan 10 elemű vektort, amelynek elemei egy számtani sorozat egymást követő elemei! Legyen az utolsó elem 120, a szomszédos elemek közötti távolság pedig 3!
8. Készítsünk egy olyan 10 elemű vektort, amelynek elemei egy számtani sorozat egymást követő elemei! Legyen az első elem a hexadecimális E01, a szomszédos elemek közötti távolság pedig 12!
9. Készítsünk egy 10 elemű vektort, mely a $\sin\left(\frac{\pi}{2}i\right)$, i=1,…,10 értékeket tartalmazza! (Ügyeljünk a kerekítésre, használjuk a *round()* függvényt!)
10. Készítsünk egy olyan 10 elemű vektort, amelynek elemei egy mértani sorozat egymást követő elemei! Legyen az első elem 20, a szomszédos elemek közötti hányados pedig 3!
11. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeit 2-szer egymás után felsorolva tartalmazza!
12. Hozzunk létre egy *y* vektort, melynek első és utolsó eleme legyen 0, valamint a közbülső értékek egy tetszőleges *x* vektor elemeit 2-szer egymás után felsorolva tartalmazzák!
13. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeit 12-szer egymás után felsorolva tartalmazza!
14. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeinek mindegyikét 12-szer rendre megismétli!
15. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor első elemét 12-szer megismétli, az összes többi *x* elemet pedig 1-szer felsorolva tartalmazza!
16. Töltsük fel 0-val a 10 elemű *x* vektor páros indexű elemeit!
17. Töltsük fel 0-val a 10 elemű *x* vektor páratlan indexű elemeit!
18. Töltsük fel 0-val a tetszőleges elemszámú *x* vektor páros indexű elemeit!
19. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeit fordított sorrendben tartalmazza!
20. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeinek alsó felét tartalmazza! (Páratlan elemszámú *x* esetén az alsó „kisebbik felét”. Használjuk a *floor()* függvényt!)
21. Hozzunk létre egy *y* vektort, mely egy tetszőleges *x* vektor elemeinek felső felét tartalmazza! (Páratlan elemszámú *x* esetén a felső „nagyobbik felét”. Használjuk a *ceiling()* függvényt!)

Egyéb adatszerkezetek

1. Készítsünk egy csupa 0 elemeket tartalmazó, 3 sorból és 4 oszlopból álló mátrixot!

2. Készítsünk egy 3 sorból és 5 oszlopból álló mátrixot, amelynek elemei a sor- és oszlopindexek szorzatait tartalmazza!

[…]