

# Bevezetés az R statisztikai programcsomag használatába

Ferenci Tamás

2021-02-23



# Tartalomjegyzék

<b>1. Előszó</b>	<b>5</b>
<b>2. Általános megjegyzések</b>	<b>7</b>
<b>3. Adattípusok, adatszerkezetek</b>	<b>9</b>
3.1. Értékadás . . . . .	9
3.2. Adattípusok . . . . .	11
3.3. Adatszerkezetek és indexelés . . . . .	14
<b>4. Függvények</b>	<b>31</b>
4.1. Függvényhívások . . . . .	31
4.2. Saját függény definiálása . . . . .	33
<b>5. Az R programozása</b>	<b>35</b>
5.1. Funkcionális programozás . . . . .	35



# 1. fejezet

## Előszó

Az R ingyenes, nyílt forráskódú statisztikai környezet.

Az R egyik fontos jellemzője, hogy lényegében minden feladat elvégzéséhez programot kell írunk. Ez elsőre ijesztőnek hangozhat, és csakugyan igaz, hogy más programnyelvekhez képest a tanulási görbéje meredekebben indul: itt már két szám átlagolásához is programot kell írni, míg máshol csak kattintgatni kell. A dolog azonban kifizetődő: lehet, hogy egyszerű dolgokat más statisztikai környezetekben könnyebb végrehajtani, itt meg bonyolultabb, de cserében itt a bonyolultabbakat sem sokkal nehezebb, míg más statisztikai programokban az, vagy egyenesen lehetetlen.

A fentiekből már érthető, hogy ahhoz, hogy el tudjunk kezdeni statisztikai elemzéseket végezni R-ben, először az R-rel mint programozási nyelvvel kell megismerkedni.

Az R talán legnagyobb erejét a hozzá megírt, megszámlálhatatlan sok kiegészítő csomag adja, amikkel jószerével minden elképzelhető statisztikai feladat – adott esetben rendkívül bonyolult számítások jelentőek is – megoldhatóak, sokszor mindössze egy-egy függvényhívással. Nagyon tipikus, hogy a vadonatúj statisztikai módszereket is R-ben implementálják első közlésükkor. A ‘reprodukálható kutatás’ jegyében a cikkekkel együtt közzétett elemzések is nagyon gyakran R-ben íródtak, ezekből szintén sok ötlet meríthető. Mindezek alapja az R mögött álló, rendkívül széles nemzetközi közösség, mely erős támogatást jelent a hibák javítása, és az új funkciók megvalósításának irányában is.



## 2. fejezet

# Általános megjegyzések

Egy R-ben írt program, gyakrabban használt nevén szkript, R-beli utasítások sorozata. Lehet egyetlen sor mely két számot átlagol, vagy több ezer utasításból felépülő komplex elemzés. Az R interpretált nyelv, nem fordított, ami azt jelenti, hogy nem a szkript egészét, egyben fordítja le számítógép által végrehajtható kóddá az R, hanem az utasításokat egyesével hajtja végre. (Általában ez azt jelenti, hogy sorról-sorra, de bizonyos esetekben egy utasítás több sorba is átnyúlhat.)

Az RStudio fejlesztői környezet alapbeállításában a bal oldali rész alján látható a konzol, ahol közvetlenül beküldhetünk utasításokat az R-nek, felette pedig a szkript, vagy szkriptek. Új szkriptet megnyitni (vagy az elsőt megnyitni, ha még egy sincs nyitva - ez esetben a konzol az egész bal oldalt elfoglalja) a **Ctrl+Shift+N** billentyűkombinációval, vagy az ikonsor bal szélső ikonjára (fehér lap zöld plusz-jellel) kattintva, és ott az **R Script** pontot választva lehet.

A konzolba írt utasítások azonnal végrehajtódnak (amint **Enter**-t ütünk, és ezzel beküldjük az utasítást az R-nek), a szkriptbe írt parancsok pedig a **Ctrl+Enter** billentyűkombinációval futtathatóak. (Valójában ez sem mond ellent annak a szabálynak, hogy a konzolba írt dolgok futtatódnak, mert ha jobban megfigyeljük, akkor láthatjuk, hogy a **Ctrl+Enter** igazából csak átmásolja az utasítást a konzolba, majd beküldi.) Ha a szkriptben nincs kijelölve semmi, akkor a **Ctrl+Enter** azt a sort futtatja, amiben a kurzor áll, ha ki van jelölve valami, akkor a kijelölést. (Függetlenül attól, hogy az milyen, lehet több sor is, de egy sor részlete is). Egy utasítás több sorba is átnyúlhat, ez nem okoz problémát, ilyenkor az R megáll, és várja a további sorokat. Az RStudio ezeket szinte mindig felismeri, és okosan jár el: ilyenkor a **Ctrl+Enter** valójában nem egy sort fog beküldeni, hanem az egész utasítást, fontos azonban, hogy ehhez a legelső sorban kell állnunk. Az egész szkript **Ctrl+Alt+R** kombinációval futtatható le.

Az aktuálisan szerkesztett szkript **Ctrl+S** utasítással, vagy az ikonsorban a kék színű, egy darab floppy-lemezes ikonra kattintva menthető. A R-szkriptek

alapértelmezett kiterjesztése a `.R`. A `Ctrl+Alt+S` parancs, vagy a kék színű, több floppy-lemezes ikon az összes megnyitott szkriptet menti. Az RStudio képes megőrizni a nem mentett szkripteket is kilépésnél (a nevük `Untitled` majd utána egy sorszám), de erre a lehetőségre azért ne nagyon építsünk, mert egy összeomlásnál elveszhetnek; a biztos a névvel lementett szkript. Mentett szkriptet megnyitni a `Ctrl+O` billentyűparanccsal, vagy az ikonsorban a mappából kifelé mutató zöld nyilas ikonnal lehet.

Minden kicsit is komolyabb munkánkat érdemes szkriptben megírni, hiszen így lesz az elemzési munkafolyamat reprodukálható. A konzolt tipikusan csak gyors, ismétlődően nem igényelt egyszerű számításokhoz használjuk, aminek az eredményére később nem lesz szükségünk, vagy szkriptírás közben az apróbb bizonytalanságok eldöntéséhez (mi is lesz ennek a parancsnak az eredménye?) használjuk.

A kódunkat érdemes kommentelni, hogy később is világos legyen a működése. A komment olyan része a szkriptnek, melyet az R nem hajt végre, hiszen tudja, hogy nem R utasítás, hanem természetes nyelven írt megjegyzés. Ennek elkülönítésére a kommentjel szolgál, ez az R-ben a `#`: amennyiben az R egy ilyenhez ér, onnantól átugorja a leírtakat egészen a sor végéig. (Ez tehát ún. egysoros kommentjel.) A `#` az RStudio-ban a `Ctrl+Shift+C`-vel szűrhető be gyorsan (azon sort kommentezi, mégpedig az elejétől fogva, amelyikben a kurzor áll). Többsoros kommentre nincs külön jel R-ben, viszont RStudio-ban a `Ctrl+Shift+C` használható több sort kijelölve is (ha nincsenek kikommentezve, akkor kikommentezi, ha ki vannak, akkor eltünteti a kommentjeleket).

Az R kisbetű/nagybetű különbségre érzékeny (case sensitive) nyelv, tehát az `a` és az `A` nem ugyanaz, két különböző dolog.

Az RStudio nagyon sok eszközzel segíti a kódolást: színekkel jelöli a különböző tartalmú szintaktikai elemeket, elkezdve egy nevet beírni, `Tab`-bal kiegészíti azt (automatikusan, ha csak egy lehetőség van, egy listát ad, ha több is), rövidebb vagy hosszabb sűgöt jelenít meg közvetlenül a beírt kód mellett stb. Segíti a kód indentálását: a `Ctrl+I` kombináció szépen beindentálja a kijelölt részt. (Tipikus a `Ctrl+A` majd `Ctrl+I` kombináció: az előbbi kijelöli az egész szkriptet, így tehát ez mindent indentál.) Az R kódolási stílus kapcsán csak egyetlen megjegyzés előjáróban: vessző után rakjunk szóközt, de nyitó zárójel után, illetve záró zárójel előtt ne.



## 3. fejezet

# Adattípusok, adatszerkezetek

Az R programozásának megértéséhez szükséges egyik alapelemünk a változó: változóban tudunk információt tárolni, legyen az egyetlen szám vagy egy egész adatbázis, vagy akár egy regressziós modell. Változóból tetszőleges számút létrehozhatunk. Elsőként meg kell ismerkednünk a változó fogalmával, a neki történő értékadással, és az információinkat tároló adatok R-beli jellemzőivel.

### 3.1. Értékadás

Változó értéket az értékadás művelettel kap; ez kb. a „legyen egyenlő” módon olvasható ki. Az értékadás jele az R-ben a `<-`. (A más programnyelveken megszokottabb `=`-t ne használjuk értékadásra, mert bár működne, de az R-es hagyományok szerint ezt egy másik helyzetre tartjuk fent, amit később látni is fogunk). A nyíl bal oldalára kerül a változó, a jobb oldalára az érték, amit adni akarunk neki. Elvileg használható a `->` is értékadásra, ilyenkor értelemszerűen fordul a helyzet, de ezt ritkán szokták alkalmazni.

Íme egy értékadás:

```
a <- 1
```

Ami szembeötlik (pláne, ha valakinek más, szigorúbb programnyelvből van háttere): ez az utasítás gond nélkül lefut, miközben sehol nem deklaráltuk, hogy az `a` legyen egy változó, pláne nem adtuk meg, hogy milyen típusú adatot akarunk benne tárolni! Az R „intelligensen” kitalált mindent: mivel látja, hogy korábban `a` nevű változó még nem létezett, ezért egyetlen szó nélkül, automatikusan létrehozza, illetve abból, hogy mit adtunk neki értékül, azt is meghatározta, hogy milyen legyen a típusa, jelen esetben szám. Majd természetesen az értékét

is beállítja arra, amit megadtunk. Már létező változónak történő értékadásnál az előző érték elveszlik, és felülíródik az aktuálisan megadottal.

Ez egy példa az R egy meglehetősen általános filozófiájára, amire később még sok további példát fogunk látni: hogy az R „megengedi trehánytságot” és igyekszik kitalálni, hogy mit akarhattunk. Bár ez első ránézésre rendkívül kényelmesnek hangzik, fontos hangsúlyozni, hogy ez egy kétélű fegyver! Egyfelől ugyanis valóban nagyon kényelmes, jelen esetben, hogy nem kell törődnünk a változók előzetes deklarálásával, típusuk megadásával, de másrészt így kiesik egy védővonal, ami megóvhatna minket a saját hibáinktól – hiszen a deklaráció rákényszerít (ett volna) minket arra, hogy jobban végiggondoljuk a változókval kapcsolatos kérdéseket. Így viszont könnyebben előfordulhat, hogy olyat csinálunk, amit igazából nem szeretnénk – ráadásul úgy, hogy észre sem vesszük! Elírjuk a változó nevét, és nem figyelmeztetést kapunk, hogy de hát ilyen változó nem létezik, hanem egyetlen hang nélkül létrejön egy új, hibás nevű (miközben az igazi értéke marad változatlan). Egy eredetileg szám típusú változónak értékül adunk egy szöveget, és ez egyetlen hang nélkül lefut, lecserélve a változó típusát.

R-ben a változónév karakterekből, számokból, a `.` és a `_` jelekből állhat, de nem kezdődhet számmal vagy `_` jellel. (Bizonyos, úgynevezett foglalt szavakat, amiket az R nyelv használ, nem választhatunk változónévnek.) Érdekes módon az, hogy az R mit ért karakter alatt, függhet az adott számítógép beállításaitól, de a legbiztosabb, ha a standard latin betűs (ASCII) karaktereket használjuk csak. (Azaz: lehetőleg ne használjunk ékezetes betűt változónévként. Elvileg el lehet vele boldogulni – adott esetben speciális szimbólummal jelölve, hogy az egy változónév – de nem éri meg a veszélyeséget, csomagokban kiszámíthatatlan gondokat okozhat.)

Egy fontos általános szabály, hogy ha egy utasításban értékadás van, akkor az eltárolás a „háttérben” történik meg, a konzolra nem íródik ki semmi. (Természetesen vannak kivételek, olyan számítások, amik mellékhatásként mindenképp kiírnak valamit a konzolra.) Értékadás nélküli utasítás futtatásánál viszont épp fordított a helyzet: az eredmény kiíratódik a konzolra, de nem tárolódik el sehol. Ha egy értékadást gömbölyű zárójelekbe ágyazunk (`(a <- 1)`), akkor el is tárolódik és ki is íratódik az eredmény; a gyakorlatban ritkán használjuk.

(Egy apró jótanács. Mi van akkor, ha lefuttatunk egy rendkívül hosszú utasítást, de véletlenül elfelejtjük benyilazni egy változóba... azaz az eredmény megjelenik a konzolon, viszont nem tárolódott le! Most futtathatjuk az egészet újra?! Szerencsére nem: az R valójában nyíl nélkül is eltárolja egy speciális változóban az eredményt, a neve `.Last.value`. Ha tehát ilyen történik, akkor ne essünk kétségbe, ezt speciális változót adjuk értékül a változónknak. De vigyázzunk, ilyen módon mindig csak a legutóbbi utasítás eredménye érhető el.)

## 3.2. Adattípusok

Elsőként meg kell ismerkednünk azzal, hogy a korábban említett típusok pontosan milyenek lehetnek – ez lényegében azt adja meg, hogy milyen jellegű adatot tárolunk az adott változóban. Az R-ben 4 fontos adattípus van: numerikus, amelybe a valós és az egész típusok tartoznak al csoportként, a szöveg és a logikai. (Elvileg még két további típus létezik, a complex és a raw, ezek nagyon ritkán használatosak.) Létezik még egy fogalom, a factor, ami adattípusnak tűnik, de mégsem az (egy másik típus speciális esete), erről később fogunk szót ejteni.

A változó típusát az R többféle módon is értelmezi, de a gyakorlatban inkább az `str` függvény ismerete a fontosabb, mellyel komplexebb adatszerkezetekről is jól áttekinthető információt tudunk nyerni.

### 3.2.1. Numerikus

Számok tárolására a numerikus típus (`numeric`, rövidítve `num`) szolgál.

Alapbeállításban ez a típus valós számokat tárol (precízen: double pontosságú lebegőpontos). A double pontossága jellemzően 53 bit (kb.  $2 \cdot 10^{-308}$ -tól  $2 \cdot 10^{308}$ -ig nagyjából  $2 \cdot 10^{-16}$  felbontással; az adott architektúra vonatkozó értéket a `.Machine` megmondja).

Így néz ki egy numerikus adattal történő értékadás:

```
szam <- 3.1
szam
```

```
## [1] 3
```

```
str(szam)
```

```
##  num 3.1
```

Nézzük meg, hogy csakugyan case sensitive a nyelv:

```
SZAM
```

```
## Error in eval(expr, envir, enclos): object 'SZAM' not found
```

```
Szam
```

```
## Error in eval(expr, envir, enclos): object 'Szam' not found
```

```
szaM
```

```
## Error in eval(expr, envir, enclos): object 'szaM' not found
```

Fontos megjegyezni, hogy attól mert valami történetesen egész, az R még nem fogja egész számként kezelni, ugyanúgy valósnak veszi:

```
szam <- 3
str(szam)
```

```
## num 3
```

Ha egészet (integer) akarunk, azt explicite jelölni kell a szám után fűzött L utótaggal:

```
egesz <- 3L  
egesz
```

```
## [1] 3
```

```
str(egesz)
```

```
## int 3
```

### 3.2.2. Szöveg

Szemben más programnyelvek, az R-ben nincs megkülönböztetve az egy karakter, és a több karakterből álló karakterfüzér (sztring). Számára mindkettő ugyanolyan típusú (`character`, rövidítve `chr`).

Szöveg:

```
szoveg <- "kiskutya"  
szoveg
```

```
## [1] "kiskutya"
```

```
str(szoveg)
```

```
## chr "kiskutya"
```

```
masikszoveg <- "a"  
masikszoveg
```

```
## [1] "a"
```

```
str(masikszoveg)
```

```
## chr "a"
```

Mint látható, a sztringkonstansokat idézőjellel kell jelölni. Az R megengedi a dupla (" ") és a szimpla (' ') idézőjel használatát is, de az előbbi a preferált (az R általi kiírás is mindenképp ilyennel történik), az utóbbit érdemes az egymásbaágyazott esetekre használni (tehát, ha egy sztringkonstans tartalmaz egy idézőjeles részt).

Természetesen az "1" kifejezés nem az 1 számot, hanem az 1-et (mint karaktert) tartalmazó sztringet jelenti! A sztringet tehát mindig jelölni kell (különben a `kiskutya` beírásakor egy ilyen nevű változót kezdene keresni az R).

### 3.2.3. Logikai

Logikai (`logical`, rövidítve `logi`) típusú változóban igaz/hamis értékeket tárolhatunk:

```
logikai <- TRUE  
logikai
```

```
## [1] TRUE
```

```
str(logikai)
```

```
## logi TRUE
```

A `TRUE` rövidíthető `T`-nek, a `FALSE` pedig `F`-nek.

### 3.2.4. Az adattípusokhoz kapcsolódó néhány fontos művelet

Adott típus tesztelése az `is.<típus>` alakban lehet:

```
is.integer(szam)
```

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

```
is.integer(egesz)
```

```
## [1] TRUE
```

```
is.integer(szoveg)
```

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

```
is.integer(logikai)
```

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

Az `is.numeric` azt jelenti, hogy `is.integer` vagy `is.double`:

```
is.double(szam)
```

```
## [1] TRUE
```

```
is.double(egesz)
```

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

```
is.numeric(szam)
```

```
## [1] TRUE
```

```
is.numeric(egesz)
```

```
## [1] TRUE
```

Adott típusá alakítás `as.<típus>` alakban lehet:

```

as.character(szam)

## [1] "3"
as.numeric(szoveg)

## Warning: NAs introduced by coercion
## [1] NA
as.numeric("2.4")

## [1] 2
as.numeric(logikai)

## [1] 1

```

A sémát már a fentiek is mutatják: a konvertálásnál egy „erőssorrend”, jelesül `character < double = integer < logical`, amely irányban mindig lehet konvertálni (a T 1-re, a F 0-ra alakul, a többi értelemszerű). A sorrenddel ellentétesen is elképzelhető, hogy lehet konvertálni, de ez már nem biztos, azon múlik, hogy értelmesen végrehajtható-e (a "kiskutya" nem konvertálható számmá, az "1" igen). Sok függvény automatikusan konvertál, például ha egy logikai igaz értékhez hozzáadunk 1-et, akkor 2-t kapunk, mert a háttérben, szó nélkül, át fogja konvertálni számmá.

A sikertelen konverziók NA-t adnak, amely az R-ben lényegében a „hiányzó érték” jele.

Speciális szerepe van még a NULL-nak (ez inkább olyasmit jelöl, hogy „üres objektum”), illetve az NaN-nek (not-a-number, tipikusan olyan adja, mint például ha negatív szám logaritmusát vesszük).

### 3.3. Adatszerkezetek és indexelés

Most, hogy ismerjük az adattípusokat, azzal kell folytatnunk, hogy ezekből milyen komplexebb struktúrák rakhatóak össze.

#### 3.3.1. Vektor

A vektor homogén, egydimenziós adatszerkezet. Egydimenziós, mert egy „kiterjedése” van, egy indexszel hivatkozhatunk az elemeire, és homogén, mert minden benne lévő adat ugyanolyan típusú kell legyen. Szemben a „vektor” matematikai fogalmával, nem kötelező, hogy ezek számok legyenek, de mindenképp ugyanolyannak kell lennie a típusuknak.

Vektor legegyszerűbb módon az elemei felsorolásával hozható létre, ehhez a `c` függvény használható:

```
szamvektor <- c(1, 4, 5, -2, 3.5, 10)
szamvektor
```

```
## [1] 1 4 5 -2 3.5 10
```

Sok függvény vektort ad vissza eredményül, például a `seq`-val generálhatunk egy reguláris sorozatot. A függvényekről később lesz szó, úgyhogy most kommentár nélkül: a `seq(1, 101, 2)` hívás kidobja a számokat 1-től 101-ig 2-esével:

```
seq(1, 101, 2)
```

```
## [1] 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37
## [20] 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75
## [39] 77 79 81 83 85 87 89 91 93 95 97 99 101
```

Az eredmény egy vektor.

Arra a speciális esetre, hogy 1-esével lépkedünk, olyan sűrűn van szükség, hogy arra van egy külön, rövidebb jelölés, a `::`:

```
1:100
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
## [19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
## [55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
## [73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
## [91] 91 92 93 94 95 96 97 98 99 100
```

A sorok elején lévő, szögletes zárójelbe írt számok nem részei a vektornak, az az olvashatóságot segíti: ha nagyon hosszú vektorban kell egy adott elem pozícióját megtalálni, akkor nem a legelejétől kell számolni, elég a sor elejétől menni.

Feltűnhet, hogy a korábbi `szam` kiíratás esetén is megjelent egy `[1]` a sor elején. Ez nem véletlen: a valóságban „skalár” nincs az R-ben, igazából a `szam` is egy vektor (csak épp egy elemből áll).

Ahogy volt róla szó, nem csak numerikus adatokból képezhető vektor, hanem bármilyenből:

```
karaktervektor <- c("a", "b", "xyz")
karaktervektor
```

```
## [1] "a" "b" "xyz"
```

A vektor homogén, ezért az alábbi utasítások csak és kizárólag azért futnak le mégis, mert a háttérben ilyenkor az R a „leggyengébbre” konvertálja az összeset (hogy kikényszerítse a homogenitást):

```
c(1, "a")
```

```
## [1] "1" "a"
```

```
c(2, TRUE)
```

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

A vektor elemei el is nevezhetőek; a nevek később a `names`-zel lekérhetőek:

```
szamvektor <- c(első = 4, második = 1, harmadik = 7)
szamvektor
```

```
##      első  második harmadik
##      4      1      7
```

```
names(szamvektor)
```

```
## [1] "első"      "második"  "harmadik"
```

A `names` érdekesen viselkedik, mert nem csak megadja a neveket, de bele is nyilazhatunk értéket, ez esetben beállítja:

```
names(szamvektor) <- c("egy", "ketto", "harom")
szamvektor
```

```
##      egy ketto harom
##      4      1      7
```

Az adatszerkezetek esetén egy alapvető kérdés az indexelés, tehát, hogy hogyan hivatkozhatunk adott pozícióban lévő elemre vagy elemekre. Ennek az R-ben meglehetősen sok módja lehetséges, de általános, hogy az indexelést a szögletes zárójel jelöli. (Később fogunk még egy szintaktikai elemet látni indexelésre.)

A legegyszerűbb eset, ha egyetlen számmal indexelünk: ekkor az adott pozícióban lévő elemet kapjuk meg. Például:

```
szamvektor[3]
```

```
##      harom
##      7
```

Megtehetjük azt is, hogy nem egy számot, hanem egy vektort adunk át, ekkor a felsorolt pozícióban lévő elemeket kapjuk, a felsorolás sorrendjében:

```
szamvektor[c(1, 3)]
```

```
##      egy harom
##      4      7
```

(Ugye látjuk, hogy ez a kettő igazából ugyanaz? Az előbbi példa is vektorral indexelt.)

Egy elem kiválasztható többször is, illetve tetszőleges sorrendben:

```
szamvektor[c(2, 2, 1, 3, 2, 3, 1, 1)]
```

```
##      ketto ketto      egy harom ketto harom      egy      egy
```



```
##      1      1      4      7      1      7      4      4
```

Nemlétező elem indexelése NA-t ad:

```
szamvektor[10]
```

```
## <NA>
```

```
## NA
```

A második alapvető megoldás a logikai vektorral való indexelés: ekkor egy ugyanolyan hosszú vektort kell átadnunk, mint az indexelendő vektor, és azokat az elemeket választja ki, ahol logikai igaz érték van:

```
szamvektor[c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE)]
```

```
## egy harm <NA> <NA>
```

```
##      4      7      NA      NA
```

Valójában azonban ez is működik, hiába rövidebb az indexelő vektor:

```
szamvektor[c(TRUE, TRUE, FALSE)]
```

```
## egy ketto
```

```
##      4      1
```

Ez egy újabb példa a kétélű flexibilitásra: azért fog működni, mert ilyenkor az R „reciklálja” az indexelő vektort.

Lehetséges negatív indexelés is, ez kiválaszt mindent, *kivéve* amit indexeltünk:

```
szamvektor[-3]
```

```
## egy ketto
```

```
##      4      1
```

```
szamvektor[-c(1, 3)]
```

```
## ketto
```

```
##      1
```

Ha vannak elnevezések, akkor azok használhatóak indexelésre is:

```
szamvektor["masodik"]
```

```
## <NA>
```

```
## NA
```

```
szamvektor[c("masodik", "utolso")]
```

```
## <NA> <NA>
```

```
## NA NA
```

Az indexelés és az értékadás kombinálható is:

```
szamvektor[3] <- 99
szamvektor
```

```
## egy ketto harom
## 4 1 99
```

```
szamvektor[10]
```

```
## <NA>
## NA
```

Ha nemlétezőnek adunk értéket, automatikusan kiterjeszti a vektort, a többi helyre pedig NA kerül (megint újabb példa a kétélű flexibilitásra):

```
szamvektor[10] <- 999
szamvektor
```

```
## egy ketto harom
## 4 1 99 NA NA NA NA NA NA 999
```

### 3.3.2. Mátrix

A mátrix homogén, kétdimenziós adatszerkezet.

Legegyszerűbben úgy tölthető fel, ha egy vektort áttördelünk, a `matrix` függvény használatával (az `nc` argumentummal az oszlopok, az `nr` argumentummal a sorok számát állíthatjuk be, értelemszerűen elég a kettőből egyet megadni):

```
szammatrix <- matrix(1:6, nc = 2)
szammatrix
```

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

Alapból oszlopok szerint tölt, de a `byrow` argumentummal ezt átállíthatjuk:

```
matrix(1:6, nc = 2, byrow = TRUE)
```

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

A dimenzió, illetve külön a sorok és oszlopok száma könnyen lekérhető:

```
dim(szammatrix)
```

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

```
nrow(szammatrix)
```

```
## [1] 3
```

```
ncol(szammatrix)
```

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

A mátrix oszlopai és sorai is elnevezhetők, emiatt itt nem egy `names` van, hanem egy `row.names` és egy `names`, ez utóbbi az oszlopnév, de egyebekben teljesen hasonlóan viselkednek.

Indexelés ugyanúgy végezhető, csak épp mindkét dimenzióra mondanunk kell valamit; a kettő vesszővel választandó el:

```
szammatrix[c(2, 3), 2]
```

```
## [1] 5 6
```

Mindkét dimenzió tetszőleges korábban látott módon indexelhető, tehát a különböző módok keverhetők is:

```
szammatrix[c(1, 2), c(T, F)]
```

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

Ha egy dimenziót nem indexelünk, akkor az R úgy érti, hogy onnan minden elem (de a vessző ekkor sem hagyható el!):

```
szammatrix[2, ]
```

```
## [1] 2 5
```

### 3.3.3. Tömb (array)

A tömb (array) homogén,  $n$ -dimenziós adatszerkezet (nem foglalkozunk vele részletesebben, ritkán használatos).

### 3.3.4. Data frame

A data frame (adatkeret) heterogén, kétdimenziós, rektanguláris adatszerkezet. Pontosabban szólva félig heterogén: az oszlopok homogének, de a különböző oszlopok típusai eltérhetnek egymástól. Lényegében tehát - nem feltétlenül ugyanolyan típusú - vektorok összefogva; a rektanguláris azt jelenti, hogy minden vektor ugyanolyan hosszú kell legyen.

Ez a legtipikusabb adatszerkezet orvosi adatok tárolására: sorokban a megfigyelési egységek, oszlopokban a változók.

A `data` paranccsal egy kiegészítő csomagban található kész adat tölthető be:

```
data(birthwt, package = "MASS")
birthwt
```

##	low	age	lwt	race	smoke	ptl	ht	ui	ftv	bwt
## 85	0	19	182	2	0	0	0	1	0	2523
## 86	0	33	155	3	0	0	0	0	3	2551
## 87	0	20	105	1	1	0	0	0	1	2557
## 88	0	21	108	1	1	0	0	1	2	2594
## 89	0	18	107	1	1	0	0	1	0	2600
## 91	0	21	124	3	0	0	0	0	0	2622
## 92	0	22	118	1	0	0	0	0	1	2637
## 93	0	17	103	3	0	0	0	0	1	2637
## 94	0	29	123	1	1	0	0	0	1	2663
## 95	0	26	113	1	1	0	0	0	0	2665
## 96	0	19	95	3	0	0	0	0	0	2722
## 97	0	19	150	3	0	0	0	0	1	2733
## 98	0	22	95	3	0	0	1	0	0	2751
## 99	0	30	107	3	0	1	0	1	2	2750
## 100	0	18	100	1	1	0	0	0	0	2769
## 101	0	18	100	1	1	0	0	0	0	2769
## 102	0	15	98	2	0	0	0	0	0	2778
## 103	0	25	118	1	1	0	0	0	3	2782
## 104	0	20	120	3	0	0	0	1	0	2807
## 105	0	28	120	1	1	0	0	0	1	2821
## 106	0	32	121	3	0	0	0	0	2	2835
## 107	0	31	100	1	0	0	0	1	3	2835
## 108	0	36	202	1	0	0	0	0	1	2836
## 109	0	28	120	3	0	0	0	0	0	2863
## 111	0	25	120	3	0	0	0	1	2	2877
## 112	0	28	167	1	0	0	0	0	0	2877
## 113	0	17	122	1	1	0	0	0	0	2906
## 114	0	29	150	1	0	0	0	0	2	2920
## 115	0	26	168	2	1	0	0	0	0	2920
## 116	0	17	113	2	0	0	0	0	1	2920
## 117	0	17	113	2	0	0	0	0	1	2920
## 118	0	24	90	1	1	1	0	0	1	2948
## 119	0	35	121	2	1	1	0	0	1	2948
## 120	0	25	155	1	0	0	0	0	1	2977
## 121	0	25	125	2	0	0	0	0	0	2977
## 123	0	29	140	1	1	0	0	0	2	2977
## 124	0	19	138	1	1	0	0	0	2	2977
## 125	0	27	124	1	1	0	0	0	0	2922
## 126	0	31	215	1	1	0	0	0	2	3005
## 127	0	33	109	1	1	0	0	0	1	3033
## 128	0	21	185	2	1	0	0	0	2	3042

## 129	0	19	189	1	0	0	0	0	2	3062
## 130	0	23	130	2	0	0	0	0	1	3062
## 131	0	21	160	1	0	0	0	0	0	3062
## 132	0	18	90	1	1	0	0	1	0	3062
## 133	0	18	90	1	1	0	0	1	0	3062
## 134	0	32	132	1	0	0	0	0	4	3080
## 135	0	19	132	3	0	0	0	0	0	3090
## 136	0	24	115	1	0	0	0	0	2	3090
## 137	0	22	85	3	1	0	0	0	0	3090
## 138	0	22	120	1	0	0	1	0	1	3100
## 139	0	23	128	3	0	0	0	0	0	3104
## 140	0	22	130	1	1	0	0	0	0	3132
## 141	0	30	95	1	1	0	0	0	2	3147
## 142	0	19	115	3	0	0	0	0	0	3175
## 143	0	16	110	3	0	0	0	0	0	3175
## 144	0	21	110	3	1	0	0	1	0	3203
## 145	0	30	153	3	0	0	0	0	0	3203
## 146	0	20	103	3	0	0	0	0	0	3203
## 147	0	17	119	3	0	0	0	0	0	3225
## 148	0	17	119	3	0	0	0	0	0	3225
## 149	0	23	119	3	0	0	0	0	2	3232
## 150	0	24	110	3	0	0	0	0	0	3232
## 151	0	28	140	1	0	0	0	0	0	3234
## 154	0	26	133	3	1	2	0	0	0	3260
## 155	0	20	169	3	0	1	0	1	1	3274
## 156	0	24	115	3	0	0	0	0	2	3274
## 159	0	28	250	3	1	0	0	0	6	3303
## 160	0	20	141	1	0	2	0	1	1	3317
## 161	0	22	158	2	0	1	0	0	2	3317
## 162	0	22	112	1	1	2	0	0	0	3317
## 163	0	31	150	3	1	0	0	0	2	3321
## 164	0	23	115	3	1	0	0	0	1	3331
## 166	0	16	112	2	0	0	0	0	0	3374
## 167	0	16	135	1	1	0	0	0	0	3374
## 168	0	18	229	2	0	0	0	0	0	3402
## 169	0	25	140	1	0	0	0	0	1	3416
## 170	0	32	134	1	1	1	0	0	4	3430
## 172	0	20	121	2	1	0	0	0	0	3444
## 173	0	23	190	1	0	0	0	0	0	3459
## 174	0	22	131	1	0	0	0	0	1	3460
## 175	0	32	170	1	0	0	0	0	0	3473
## 176	0	30	110	3	0	0	0	0	0	3544
## 177	0	20	127	3	0	0	0	0	0	3487
## 179	0	23	123	3	0	0	0	0	0	3544
## 180	0	17	120	3	1	0	0	0	0	3572
## 181	0	19	105	3	0	0	0	0	0	3572

## 182	0	23	130	1	0	0	0	0	0	3586
## 183	0	36	175	1	0	0	0	0	0	3600
## 184	0	22	125	1	0	0	0	0	1	3614
## 185	0	24	133	1	0	0	0	0	0	3614
## 186	0	21	134	3	0	0	0	0	2	3629
## 187	0	19	235	1	1	0	1	0	0	3629
## 188	0	25	95	1	1	3	0	1	0	3637
## 189	0	16	135	1	1	0	0	0	0	3643
## 190	0	29	135	1	0	0	0	0	1	3651
## 191	0	29	154	1	0	0	0	0	1	3651
## 192	0	19	147	1	1	0	0	0	0	3651
## 193	0	19	147	1	1	0	0	0	0	3651
## 195	0	30	137	1	0	0	0	0	1	3699
## 196	0	24	110	1	0	0	0	0	1	3728
## 197	0	19	184	1	1	0	1	0	0	3756
## 199	0	24	110	3	0	1	0	0	0	3770
## 200	0	23	110	1	0	0	0	0	1	3770
## 201	0	20	120	3	0	0	0	0	0	3770
## 202	0	25	241	2	0	0	1	0	0	3790
## 203	0	30	112	1	0	0	0	0	1	3799
## 204	0	22	169	1	0	0	0	0	0	3827
## 205	0	18	120	1	1	0	0	0	2	3856
## 206	0	16	170	2	0	0	0	0	4	3860
## 207	0	32	186	1	0	0	0	0	2	3860
## 208	0	18	120	3	0	0	0	0	1	3884
## 209	0	29	130	1	1	0	0	0	2	3884
## 210	0	33	117	1	0	0	0	1	1	3912
## 211	0	20	170	1	1	0	0	0	0	3940
## 212	0	28	134	3	0	0	0	0	1	3941
## 213	0	14	135	1	0	0	0	0	0	3941
## 214	0	28	130	3	0	0	0	0	0	3969
## 215	0	25	120	1	0	0	0	0	2	3983
## 216	0	16	95	3	0	0	0	0	1	3997
## 217	0	20	158	1	0	0	0	0	1	3997
## 218	0	26	160	3	0	0	0	0	0	4054
## 219	0	21	115	1	0	0	0	0	1	4054
## 220	0	22	129	1	0	0	0	0	0	4111
## 221	0	25	130	1	0	0	0	0	2	4153
## 222	0	31	120	1	0	0	0	0	2	4167
## 223	0	35	170	1	0	1	0	0	1	4174
## 224	0	19	120	1	1	0	0	0	0	4238
## 225	0	24	116	1	0	0	0	0	1	4593
## 226	0	45	123	1	0	0	0	0	1	4990
## 4	1	28	120	3	1	1	0	1	0	709
## 10	1	29	130	1	0	0	0	1	2	1021
## 11	1	34	187	2	1	0	1	0	0	1135

## 13	1	25	105	3	0	1	1	0	0	1330
## 15	1	25	85	3	0	0	0	1	0	1474
## 16	1	27	150	3	0	0	0	0	0	1588
## 17	1	23	97	3	0	0	0	1	1	1588
## 18	1	24	128	2	0	1	0	0	1	1701
## 19	1	24	132	3	0	0	1	0	0	1729
## 20	1	21	165	1	1	0	1	0	1	1790
## 22	1	32	105	1	1	0	0	0	0	1818
## 23	1	19	91	1	1	2	0	1	0	1885
## 24	1	25	115	3	0	0	0	0	0	1893
## 25	1	16	130	3	0	0	0	0	1	1899
## 26	1	25	92	1	1	0	0	0	0	1928
## 27	1	20	150	1	1	0	0	0	2	1928
## 28	1	21	200	2	0	0	0	1	2	1928
## 29	1	24	155	1	1	1	0	0	0	1936
## 30	1	21	103	3	0	0	0	0	0	1970
## 31	1	20	125	3	0	0	0	1	0	2055
## 32	1	25	89	3	0	2	0	0	1	2055
## 33	1	19	102	1	0	0	0	0	2	2082
## 34	1	19	112	1	1	0	0	1	0	2084
## 35	1	26	117	1	1	1	0	0	0	2084
## 36	1	24	138	1	0	0	0	0	0	2100
## 37	1	17	130	3	1	1	0	1	0	2125
## 40	1	20	120	2	1	0	0	0	3	2126
## 42	1	22	130	1	1	1	0	1	1	2187
## 43	1	27	130	2	0	0	0	1	0	2187
## 44	1	20	80	3	1	0	0	1	0	2211
## 45	1	17	110	1	1	0	0	0	0	2225
## 46	1	25	105	3	0	1	0	0	1	2240
## 47	1	20	109	3	0	0	0	0	0	2240
## 49	1	18	148	3	0	0	0	0	0	2282
## 50	1	18	110	2	1	1	0	0	0	2296
## 51	1	20	121	1	1	1	0	1	0	2296
## 52	1	21	100	3	0	1	0	0	4	2301
## 54	1	26	96	3	0	0	0	0	0	2325
## 56	1	31	102	1	1	1	0	0	1	2353
## 57	1	15	110	1	0	0	0	0	0	2353
## 59	1	23	187	2	1	0	0	0	1	2367
## 60	1	20	122	2	1	0	0	0	0	2381
## 61	1	24	105	2	1	0	0	0	0	2381
## 62	1	15	115	3	0	0	0	1	0	2381
## 63	1	23	120	3	0	0	0	0	0	2410
## 65	1	30	142	1	1	1	0	0	0	2410
## 67	1	22	130	1	1	0	0	0	1	2410
## 68	1	17	120	1	1	0	0	0	3	2414
## 69	1	23	110	1	1	1	0	0	0	2424

```
## 71    1  17 120    2    0  0  0  0  2 2438
## 75    1  26 154    3    0  1  1  0  1 2442
## 76    1  20 105    3    0  0  0  0  3 2450
## 77    1  26 190    1    1  0  0  0  0 2466
## 78    1  14 101    3    1  1  0  0  0 2466
## 79    1  28  95    1    1  0  0  0  2 2466
## 81    1  14 100    3    0  0  0  0  2 2495
## 82    1  23  94    3    1  0  0  0  0 2495
## 83    1  17 142    2    0  0  1  0  0 2495
## 84    1  21 130    1    1  0  1  0  3 2495
```

Csak a felső néhány sor a `head` paranccsal kérhető le (az alsó néhány sor pedig a `tail`-lel):

```
head(birthwt)
```

```
##      low age lwt race smoke ptl ht ui ftv  bwt
## 85    0  19 182    2    0  0  0  1  0 2523
## 86    0  33 155    3    0  0  0  0  3 2551
## 87    0  20 105    1    1  0  0  0  1 2557
## 88    0  21 108    1    1  0  0  1  2 2594
## 89    0  18 107    1    1  0  0  1  0 2600
## 91    0  21 124    3    0  0  0  0  0 2622
```

Az oszlopok és a sorok is elnevezhetők:

```
str(birthwt)
```

```
## 'data.frame':    189 obs. of  10 variables:
## $ low   : int  0 0 0 0 0 0 0 0 0 0 ...
## $ age   : int  19 33 20 21 18 21 22 17 29 26 ...
## $ lwt   : int  182 155 105 108 107 124 118 103 123 113 ...
## $ race  : int  2 3 1 1 1 3 1 3 1 1 ...
## $ smoke: int  0 0 1 1 1 0 0 0 1 1 ...
## $ ptl   : int  0 0 0 0 0 0 0 0 0 0 ...
## $ ht    : int  0 0 0 0 0 0 0 0 0 0 ...
## $ ui     : int  1 0 0 1 1 0 0 0 0 0 ...
## $ ftv   : int  0 3 1 2 0 0 1 1 1 0 ...
## $ bwt   : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

```
names(birthwt)
```

```
## [1] "low"  "age"  "lwt"  "race" "smoke" "ptl"  "ht"   "ui"   "ftv"
## [10] "bwt"
```

```
colnames(birthwt)
```

```
## [1] "low"  "age"  "lwt"  "race" "smoke" "ptl"  "ht"   "ui"   "ftv"
## [10] "bwt"
```



Az adatkeret a mátrixhoz hasonlóan indexelhető:

```
birthwt[3, ]
```

```
##      low age lwt race smoke ptl ht ui ftv  bwt
## 87    0  20 105    1     1  0  0  0   1 2557
```

```
birthwt[3, 4]
```

```
## [1] 1
```

```
birthwt[3, c(5, 6)]
```

```
##      smoke ptl
## 87      1   0
```

Sőt, ha vannak elnevezéseink, az is használható. A következő 4 mind egyenértékű:

```
birthwt[, 10]
```

```
##      [1] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 2722 2733 2751 2750 2769
##      [16] 2769 2778 2782 2807 2821 2835 2835 2836 2863 2877 2877 2906 2920 2920 2920
##      [31] 2920 2948 2948 2977 2977 2977 2977 2922 3005 3033 3042 3062 3062 3062 3062
##      [46] 3062 3080 3090 3090 3090 3100 3104 3132 3147 3175 3175 3203 3203 3203 3225
##      [61] 3225 3232 3232 3234 3260 3274 3274 3303 3317 3317 3317 3321 3331 3374 3374
##      [76] 3402 3416 3430 3444 3459 3460 3473 3544 3487 3544 3572 3572 3586 3600 3614
##      [91] 3614 3629 3629 3637 3643 3651 3651 3651 3651 3699 3728 3756 3770 3770 3770
##     [106] 3790 3799 3827 3856 3860 3860 3884 3884 3912 3940 3941 3941 3969 3983 3997
##     [121] 3997 4054 4054 4111 4153 4167 4174 4238 4593 4990  709 1021 1135 1330 1474
##     [136] 1588 1588 1701 1729 1790 1818 1885 1893 1899 1928 1928 1928 1936 1970 2055
##     [151] 2055 2082 2084 2084 2100 2125 2126 2187 2187 2211 2225 2240 2240 2282 2296
##     [166] 2296 2301 2325 2353 2353 2367 2381 2381 2381 2410 2410 2410 2414 2424 2438
##     [181] 2442 2450 2466 2466 2466 2495 2495 2495 2495
```

```
birthwt$bwt
```

```
##      [1] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 2722 2733 2751 2750 2769
##      [16] 2769 2778 2782 2807 2821 2835 2835 2836 2863 2877 2877 2906 2920 2920 2920
##      [31] 2920 2948 2948 2977 2977 2977 2977 2922 3005 3033 3042 3062 3062 3062 3062
##      [46] 3062 3080 3090 3090 3090 3100 3104 3132 3147 3175 3175 3203 3203 3203 3225
##      [61] 3225 3232 3232 3234 3260 3274 3274 3303 3317 3317 3317 3321 3331 3374 3374
##      [76] 3402 3416 3430 3444 3459 3460 3473 3544 3487 3544 3572 3572 3586 3600 3614
##      [91] 3614 3629 3629 3637 3643 3651 3651 3651 3651 3699 3728 3756 3770 3770 3770
##     [106] 3790 3799 3827 3856 3860 3860 3884 3884 3912 3940 3941 3941 3969 3983 3997
##     [121] 3997 4054 4054 4111 4153 4167 4174 4238 4593 4990  709 1021 1135 1330 1474
##     [136] 1588 1588 1701 1729 1790 1818 1885 1893 1899 1928 1928 1928 1936 1970 2055
##     [151] 2055 2082 2084 2084 2100 2125 2126 2187 2187 2211 2225 2240 2240 2282 2296
##     [166] 2296 2301 2325 2353 2353 2367 2381 2381 2381 2410 2410 2410 2414 2424 2438
##     [181] 2442 2450 2466 2466 2466 2495 2495 2495 2495
```

```
birthwt[, "bwt"]
```

```
##      [1] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 2722 2733 2751 2750 2769
##     [16] 2769 2778 2782 2807 2821 2835 2835 2836 2863 2877 2877 2906 2920 2920 2920
##     [31] 2920 2948 2948 2977 2977 2977 2977 2922 3005 3033 3042 3062 3062 3062 3062
##     [46] 3062 3080 3090 3090 3090 3100 3104 3132 3147 3175 3175 3203 3203 3203 3225
##     [61] 3225 3232 3232 3234 3260 3274 3274 3303 3317 3317 3317 3321 3331 3374 3374
##     [76] 3402 3416 3430 3444 3459 3460 3473 3544 3487 3544 3572 3572 3586 3600 3614
##     [91] 3614 3629 3629 3637 3643 3651 3651 3651 3651 3699 3728 3756 3770 3770 3770
##    [106] 3790 3799 3827 3856 3860 3860 3884 3884 3912 3940 3941 3941 3969 3983 3997
##    [121] 3997 4054 4054 4111 4153 4167 4174 4238 4593 4990  709 1021 1135 1330 1474
##    [136] 1588 1588 1701 1729 1790 1818 1885 1893 1899 1928 1928 1928 1936 1970 2055
##    [151] 2055 2082 2084 2084 2100 2125 2126 2187 2187 2211 2225 2240 2240 2282 2296
##    [166] 2296 2301 2325 2353 2353 2367 2381 2381 2381 2410 2410 2410 2414 2424 2438
##    [181] 2442 2450 2466 2466 2466 2495 2495 2495 2495
```

```
birthwt[["bwt"]]
```

```
##      [1] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 2722 2733 2751 2750 2769
##     [16] 2769 2778 2782 2807 2821 2835 2835 2836 2863 2877 2877 2906 2920 2920 2920
##     [31] 2920 2948 2948 2977 2977 2977 2977 2922 3005 3033 3042 3062 3062 3062 3062
##     [46] 3062 3080 3090 3090 3090 3100 3104 3132 3147 3175 3175 3203 3203 3203 3225
##     [61] 3225 3232 3232 3234 3260 3274 3274 3303 3317 3317 3317 3321 3331 3374 3374
##     [76] 3402 3416 3430 3444 3459 3460 3473 3544 3487 3544 3572 3572 3586 3600 3614
##     [91] 3614 3629 3629 3637 3643 3651 3651 3651 3651 3699 3728 3756 3770 3770 3770
##    [106] 3790 3799 3827 3856 3860 3860 3884 3884 3912 3940 3941 3941 3969 3983 3997
##    [121] 3997 4054 4054 4111 4153 4167 4174 4238 4593 4990  709 1021 1135 1330 1474
##    [136] 1588 1588 1701 1729 1790 1818 1885 1893 1899 1928 1928 1928 1936 1970 2055
##    [151] 2055 2082 2084 2084 2100 2125 2126 2187 2187 2211 2225 2240 2240 2282 2296
##    [166] 2296 2301 2325 2353 2353 2367 2381 2381 2381 2410 2410 2410 2414 2424 2438
##    [181] 2442 2450 2466 2466 2466 2495 2495 2495 2495
```

A nem dupla szögletes zárójellel történő indexelés eltérése, hogy nem a kiválasztott vektort, hanem egy csak a kiválasztott vektorból álló data frame-et ad vissza:

```
birthwt[["bwt"]]
```

```
##      [1] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 2722 2733 2751 2750 2769
##     [16] 2769 2778 2782 2807 2821 2835 2835 2836 2863 2877 2877 2906 2920 2920 2920
##     [31] 2920 2948 2948 2977 2977 2977 2977 2922 3005 3033 3042 3062 3062 3062 3062
##     [46] 3062 3080 3090 3090 3090 3100 3104 3132 3147 3175 3175 3203 3203 3203 3225
##     [61] 3225 3232 3232 3234 3260 3274 3274 3303 3317 3317 3317 3321 3331 3374 3374
##     [76] 3402 3416 3430 3444 3459 3460 3473 3544 3487 3544 3572 3572 3586 3600 3614
##     [91] 3614 3629 3629 3637 3643 3651 3651 3651 3651 3699 3728 3756 3770 3770 3770
##    [106] 3790 3799 3827 3856 3860 3860 3884 3884 3912 3940 3941 3941 3969 3983 3997
##    [121] 3997 4054 4054 4111 4153 4167 4174 4238 4593 4990  709 1021 1135 1330 1474
##    [136] 1588 1588 1701 1729 1790 1818 1885 1893 1899 1928 1928 1928 1936 1970 2055
```

```
## [151] 2055 2082 2084 2084 2100 2125 2126 2187 2187 2211 2225 2240 2240 2282 2296
## [166] 2296 2301 2325 2353 2353 2367 2381 2381 2410 2410 2410 2414 2424 2438
## [181] 2442 2450 2466 2466 2495 2495 2495 2495
```

```
str(birthwt[["bwt"]])
```

```
## int [1:189] 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

```
head(birthwt["bwt"])
```

```
##      bwt
## 85 2523
## 86 2551
## 87 2557
## 88 2594
## 89 2600
## 91 2622
```

```
str(birthwt["bwt"])
```

```
## 'data.frame': 189 obs. of 1 variable:
## $ bwt: int 2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```

Használhatunk különféle módszereket (az alábbiak közül a második a logikai indexelés miatt fog működni):

```
head(birthwt[, c("lwt", "smoke")])
```

```
##      lwt smoke
## 85 182      0
## 86 155      0
## 87 105      1
## 88 108      1
## 89 107      1
## 91 124      0
```

```
head(birthwt[birthwt$smoke == 1, ])
```

```
##      low age lwt race smoke ptl ht ui ftv  bwt
## 87    0  20 105    1     1  0  0  0   1 2557
## 88    0  21 108    1     1  0  0  1   2 2594
## 89    0  18 107    1     1  0  0  1   0 2600
## 94    0  29 123    1     1  0  0  0   1 2663
## 95    0  26 113    1     1  0  0  0   0 2665
## 100   0  18 100    1     1  0  0  0   0 2769
```

```
head(birthwt[birthwt$smoke == 1 & birthwt$race == 1, ])
```

```
##      low age lwt race smoke ptl ht ui ftv  bwt
## 87    0  20 105    1     1  0  0  0   1 2557
## 88    0  21 108    1     1  0  0  1   2 2594
```

```
## 89    0  18 107    1    1  0  0  1    0 2600
## 94    0  29 123    1    1  0  0  0    1 2663
## 95    0  26 113    1    1  0  0  0    0 2665
## 100   0  18 100    1    1  0  0  0    0 2769
```

Az adatkeret heterogén:

```
birthwt$nev <- "a"
head(birthwt)
```

```
##      low age lwt race smoke ptl ht ui ftv  bwt nev
## 85    0  19 182    2    0  0  0  1    0 2523  a
## 86    0  33 155    3    0  0  0  0    3 2551  a
## 87    0  20 105    1    1  0  0  0    1 2557  a
## 88    0  21 108    1    1  0  0  1    2 2594  a
## 89    0  18 107    1    1  0  0  1    0 2600  a
## 91    0  21 124    3    0  0  0  0    0 2622  a
```

```
str(birthwt)
```

```
## 'data.frame':    189 obs. of  11 variables:
## $ low   : int  0 0 0 0 0 0 0 0 0 0 0 ...
## $ age   : int  19 33 20 21 18 21 22 17 29 26 ...
## $ lwt   : int  182 155 105 108 107 124 118 103 123 113 ...
## $ race  : int  2 3 1 1 1 3 1 3 1 1 ...
## $ smoke: int  0 0 1 1 1 0 0 0 1 1 ...
## $ ptl   : int  0 0 0 0 0 0 0 0 0 0 ...
## $ ht    : int  0 0 0 0 0 0 0 0 0 0 ...
## $ ui    : int  1 0 0 1 1 0 0 0 0 0 ...
## $ ftv   : int  0 3 1 2 0 0 1 1 1 0 ...
## $ bwt   : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
## $ nev   : chr  "a" "a" "a" "a" ...
```

### 3.3.5. Lista

A lista heterogén, egydimenziós adatszerkezet.

Legegyszerűbben elemei felsorolásával hozható létre, a `list` függvényt használva:

```
lista <- list(sz = szamvektor, k = karaktervektor, m = szammatrix, df = birthwt[1:5,
])
lista
```

```
## $sz
## egy ketto harom
##      4      1      99      NA      NA      NA      NA      NA      NA      999
##
## $k
## [1] "a"    "b"    "xyz"
```

```
##
## $m
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
##
## $df
##      low age lwt race smoke ptl ht ui ftv  bwt nev
## 85    0  19 182    2     0  0  0  1  0 2523  a
## 86    0  33 155    3     0  0  0  0  3 2551  a
## 87    0  20 105    1     1  0  0  0  1 2557  a
## 88    0  21 108    1     1  0  0  1  2 2594  a
## 89    0  18 107    1     1  0  0  1  0 2600  a
```

```
str(lista)
```

```
## List of 4
## $ sz: Named num [1:10] 4 1 99 NA NA NA NA NA NA 999
## ..- attr(*, "names")= chr [1:10] "egy" "ketto" "harom" "" ...
## $ k : chr [1:3] "a" "b" "xyz"
## $ m : int [1:3, 1:2] 1 2 3 4 5 6
## $ df:'data.frame': 5 obs. of 11 variables:
## ..$ low : int [1:5] 0 0 0 0 0
## ..$ age : int [1:5] 19 33 20 21 18
## ..$ lwt : int [1:5] 182 155 105 108 107
## ..$ race : int [1:5] 2 3 1 1 1
## ..$ smoke: int [1:5] 0 0 1 1 1
## ..$ ptl : int [1:5] 0 0 0 0 0
## ..$ ht : int [1:5] 0 0 0 0 0
## ..$ ui : int [1:5] 1 0 0 1 1
## ..$ ftv : int [1:5] 0 3 1 2 0
## ..$ bwt : int [1:5] 2523 2551 2557 2594 2600
## ..$ nev : chr [1:5] "a" "a" "a" "a" ...
```

Számmal és – ha van neki – névvel is indexelhető:

```
lista[[1]]
```

```
##      egy ketto harom
##      4      1    99    NA    NA    NA    NA    NA    NA    999
```

```
lista$sz
```

```
##      egy ketto harom
##      4      1    99    NA    NA    NA    NA    NA    NA    999
```

```
lista[["sz"]]
```

```
## egy ketto harom
##      4      1      99      NA      NA      NA      NA      NA      NA      999
```

Az egy zárójellel történő indexelés látszólag ugyanaz, de csak látszólag:

```
lista[1]
```

```
## $sz
## egy ketto harom
##      4      1      99      NA      NA      NA      NA      NA      NA      999
```

```
typeof(lista[[1]])
```

```
## [1] "double"
```

```
typeof(lista[1])
```

```
## [1] "list"
```

Tartomány is indexelhető:

```
lista[1:2]
```

```
## $sz
## egy ketto harom
##      4      1      99      NA      NA      NA      NA      NA      NA      999
##
## $k
## [1] "a"    "b"    "xyz"
```

```
lista[[1:2]]
```

```
## [1] 1
```

Az előbbi dolgok természetesen kombinálhatóak is:

```
idx <- "sz"
lista[[idx]]
```

```
## egy ketto harom
##      4      1      99      NA      NA      NA      NA      NA      NA      999
```

Az adatkeret igazából egy, az oszlopokból - mint vektorokból - összerakott lista (tehát két szűkítés van: az elemek csak vektorok lehetnek és ugyanolyan hosszúaknak kell lenniük).

## 4. fejezet

# Függvények

A függvényekről

### 4.1. Függvényhívások

Függvény úgy hívható, hogy megadjuk a nevét, majd utána zárójelben az argumentumát, vagy argumentumait (lehet, hogy egy sincs, de a zárójelet ekkor is ki kell írni):

```
quantile(birthwt$bwt)
```

```
##    0%   25%   50%   75%  100%  
##  709 2414 2977 3487 4990
```

Függvényről súgó a kérdőjellel kapható (két kérdőjel az összes ismert függvényt végigkeresi, akár névtöredékre is): `?quantile`.

Aminél egyenlőségjellel adva van érték a specifikációban, ott az default-ként viselkedik, nem kötelező megadni, viszont a default-tal nem rendelkezőket muszáj:

```
quantile()
```

```
## Error in is.factor(x): argument "x" is missing, with no default
```

Ha több argumentumot adunk meg, akkor azok a felsorolás sorrendjében osztódnak ki:

```
quantile(birthwt$bwt, 0.23)
```

```
## 23%  
## 2388
```

```
quantile(birthwt$bwt, c(0.23, 0.5, 0.6))
```

```
## 23% 50% 60%
## 2388 2977 3169
```

Argumentumra hivatkozhatunk névvel is, ez esetben nem kell a felsorolás sorrendjével törődnünk:

```
quantile(birthwt$bwt, c(0.23, 0.5, 0.6), type = 6)
```

```
## 23% 50% 60%
## 2381 2977 3175
```

```
quantile(probs = c(0.23, 0.5, 0.6), type = 6, x = birthwt$bwt)
```

```
## 23% 50% 60%
## 2381 2977 3175
```

Az általános gyakorlat az, hogy az első két-három argumentumot adhatjuk meg név nélkül (ezeknél elvárható, hogy fejből is tudja az ember, hogy mit jelent), de a többinél elegánsabb, ha mindenképp adunk nevet (tehát akkor is, ha sorrendben írjuk).

Egy függvény hívásánál az argumentumai elkülöníthetők egy listába, majd ugyanaz a hatása a `do.call` használatával elérhető (első argumentum a függvény, második az átadandó argumentumok listája):

```
quantile(probs = c(0.23, 0.5, 0.6), type = 6, x = birthwt$bwt)
```

```
## 23% 50% 60%
## 2381 2977 3175
```

```
do.call(quantile, list(probs = c(0.23, 0.5, 0.6), type = 6, x = birthwt$bwt))
```

```
## 23% 50% 60%
## 2381 2977 3175
```

Ez akkor jön jól, ha nem tudjuk előre, hogy mik az argumentumok (akár azt sem, hogy hány darab van belőlük!), pl. mert egy `lapply`-jal gyártottuk le, lásd később:

```
rbind(c(1, 2), c(3, 4), c(5, 6))
```

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

```
do.call(rbind, lapply(birthwt, function(x) c(mean(x), median(x))))
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning NA
```

```
##      [,1]      [,2]
## low  "0.312169312169312" "0"
```



```
## age    "23.2380952380952"    "23"  
## lwt    "129.814814814815"    "121"  
## race   "1.84656084656085"    "1"  
## smoke  "0.391534391534392"   "0"  
## ptl    "0.195767195767196"   "0"  
## ht     "0.0634920634920635"   "0"  
## ui     "0.148148148148148"   "0"  
## ftv    "0.793650793650794"   "0"  
## bwt    "2944.5873015873"     "2977"  
## nev    NA                    "a"
```

## 4.2. Saját függény definiálása

Ilyet is lehet.



## 5. fejezet

# Az R programozása

Programozás.

### 5.1. Funkcionális programozás

Az R, bár többféle paradigmában is tud dolgozni, érezhető funkcionális nyelv. Ezt elegáns is, célszerű is kihasználni!

Egy példa:

```
mean(birthwt$bwt[1:100])
```

```
## [1] 3130
```

```
elsoszazatlag <- function(data) {  
  result <- mean(data[1:100])  
  return(result)  
}
```

```
elsoszazatlag <- function(data) {  
  result <- mean(data[1:100])  
  result  
}
```

```
elsoszazatlag <- function(data) {  
  mean(data[1:100])  
}
```

```
elsoszazatlag(birthwt$bwt)
```

```
## [1] 3130
```

```
sd(birthwt$bwt[1:100])
```

```
## [1] 324
```

```
elsoszazf <- function(data, f = mean) {
  f(data[1:100])
}
elsoszazf(birthwt$bwt)
```

```
## [1] 3130
```

```
elsoszazf(birthwt$bwt, f = sd)
```

```
## [1] 324
```

A `lapply` az első argumentumban megadott lista minden elemére ráereszti a második argumentumban megadott függvényt, és az eredményt összefűzi egy listává (a `sapply` csak annyiban tér el, hogy lista helyett vektort ad vissza, ha lehetséges a listát vektorra konvertálni):

```
lapply(c("age", "lwt", "bwt"), nchar)
```

```
## [[1]]
```

```
## [1] 3
```

```
##
```

```
## [[2]]
```

```
## [1] 3
```

```
##
```

```
## [[3]]
```

```
## [1] 3
```

```
sapply(c("age", "lwt", "bwt"), nchar)
```

```
## age lwt bwt
```

```
## 3 3 3
```

```
lapply(c("age", "lwt", "bwt"), function(x) nchar(x))
```

```
## [[1]]
```

```
## [1] 3
```

```
##
```

```
## [[2]]
```

```
## [1] 3
```

```
##
```

```
## [[3]]
```

```
## [1] 3
```

```
lapply(c("age", "lwt", "bwt"), function(x) mean(birthwt[[x]]))
```

```
## [[1]]
```

```
## [1] 23
##
## [[2]]
## [1] 130
##
## [[3]]
## [1] 2945

sapply(c("age", "lwt", "bwt"), function(x) mean(birthwt[[x]]))

## age lwt bwt
## 23 130 2945

sapply(birthwt, mean)

## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA

## low age lwt race smoke ptl ht ui ftv bwt nev
## 3e-01 2e+01 1e+02 2e+00 4e-01 2e-01 6e-02 1e-01 8e-01 3e+03 NA

lapply(birthwt, function(x) c(mean(x), median(x)))

## Warning in mean.default(x): argument is not numeric or logical: returning NA

## $low
## [1] 0.3 0.0
##
## $age
## [1] 23 23
##
## $lwt
## [1] 130 121
##
## $race
## [1] 2 1
##
## $smoke
## [1] 0.4 0.0
##
## $ptl
## [1] 0.2 0.0
##
## $ht
## [1] 0.06 0.00
##
## $ui
## [1] 0.1 0.0
##
```

```
## $ftv
## [1] 0.8 0.0
##
## $bwt
## [1] 2945 2977
##
## $nev
## [1] NA "a"
```

A harmadik sor példát mutat arra, hogy anonim függvény is használható, az utolsó előtti pedig arra, hogy a `data.frame` igazából lista, aminek az elemei az oszlopai.

Az `apply` az első argumentumban megadott mátrix vagy adatkeret minden sorára vagy oszlopára (ezt a második argumentum dönti el) ráereszti a harmadik argumentumban megadott függvényt:

```
apply(birthwt, 2, mean)
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
```

```
## returning NA

## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA

##   low   age   lwt   race smoke   ptl   ht   ui   ftv   bwt   nev
##   NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA

apply(birthwt, 1, function(x) x[1])

## 85 86 87 88 89 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 106 107 108 109 111 112 113 114 115 116 117 118 119 120 121 123 124 125 126 127
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 148 149 150 151 154 155 156 159 160 161 162 163 164 166 167 168 169 170 172 173
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 174 175 176 177 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 195
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 196 197 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "0"
## 217 218 219 220 221 222 223 224 225 226 4 10 11 13 15 16 17 18 19 20
## "0" "0" "0" "0" "0" "0" "0" "0" "0" "0" "1" "1" "1" "1" "1" "1" "1" "1" "1"
## 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 40 42 43 44
## "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1"
## 45 46 47 49 50 51 52 54 56 57 59 60 61 62 63 65 67 68 69 71
## "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1"
## 75 76 77 78 79 81 82 83 84
## "1" "1" "1" "1" "1" "1" "1" "1" "1"
```

A `tapply` az első argumentumban megadott változó második argumentum szerint képezett csoportjaira ráereszti a harmadik argumentumban megadott függvényt:

```
mean(birthwt$bwt[birthwt$race == 1])

## [1] 3103

tapply(birthwt$bwt, birthwt$race, mean)

##      1      2      3
## 3103 2720 2805
```