

Bevezetés az R statisztikai programcsomag használatába

Ferenci Tamás, tamas.ferenci@medstat.hu

2024. október 27.

Tartalomjegyzék

1	Előszó	5
2	Az R szkriptek néhány alapvető jellemzője, az RStudio integrált fejlesztői környezet	7
3	Adattípusok, adatszerkezetek	11
3.1	Értékadás	11
3.2	Adattípusok	13
3.3	Adatszerkezetek és indexelés	16
4	Függvények	33
4.1	Függvényhívások	33
4.2	Saját függény definiálása	35
5	Az R programozása	37
5.1	Funkcionális programozás	37
6	Data table: egy továbbfejlesztett adatkeret	43
6.1	Sebesség és nagyméretű adatbázisok kezelése	45
6.2	Jobb kiíratás	46
6.3	Kényelmesebb sorindexelés (sor-szűrés és -rendezés)	47
6.4	Kibővített oszlopindexelés: oszlop-kiválasztás és oszlop-létrehozás műveletekkel	50
6.5	Csoportosítás (aggregáció)	57
6.6	Indexelések láncolása egymás után	65
6.7	Referencia szemantika	69

1. fejezet

Előszó

Az R egy ingyenes, nyílt forráskódú, rendkívüli tudású és folyamatosan fejlődő statisztikai programozási nyelv illetve környezet, mely kiválóan alkalmas a legkülönbözőbb statisztikai és adattudományi feladatok megoldására.

Az R egyik fontos jellemzője, hogy lényegében minden feladat elvégzéshez egy szkriptet kell írunk – szemben más statisztikai programokkal¹, ahol csak egy grafikus felületen kell kattintgatnunk. Ez elsőre ijesztőnek hangozhat, és csak-ugyan igaz, hogy más programokhoz képest a tanulási görbe meredekebben indul, hiszen a kattintgatással szemben itt már két szám átlagolásához is programot kell írni. 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. Kicsit is komolyabb elemzések, kutatások végzésekor az R megtanulásába befektetett munka hamar – és pláne: busásan – megtérül.

A fentiekből már érthető, hogy ahhoz, hogy el tudjunk kezdeni statisztikai elemzéseket végezni R-ben, először az R-rel mint programozási nyelvvel kell megismerkedni. Nagyon fontos hangsúlyozni, hogy ez a jegyzet kizárólag az R *nyelvi* kérdéseivel és programozásával foglalkozik, az R *statisztikai* célokra történő felhasználása egy másik jegyzetem (Ferenci Tamás: Bevezetés a biostatistikába) témája.

Az R talán legnagyobb erejét a hozzá megírt, megszámlálhatatlan sok² kiegészítő csomag adja, amikkel jószerével minden elképzelhető (és számos nehezen elképzelhető...) statisztikai feladat, adott esetben rendkívül bonyolultak is megoldhatóak, sokszor mindössze egy-egy függvényhívással. Számos kitűnő, jól

¹Megjegyzendő, hogy az R-hez is létezik ilyen grafikus felület, az R Commander, azonban használata komolyabb elemzési feladatok elvégzéséhez nem szükséges, illetve nem hasznos, kezdők számára azonban kitűnő bevezető eszköz lehet, mivel a jól ismert statisztikai programokhoz teljesen hasonló grafikus felülettel ruházta fel az R-et.

²2021 őszén már több mint 18 ezer!

dokumentált kiegészítő csomag érhető el (melyek maguk is ingyenesek és nyílt forráskódúak); nagyon tipikus, hogy a vadonatúj statisztikai módszereket is R-ben implementálják első közlésükkor. Az R csomagok központi repozitóriuma CRAN (Comprehensive R Archive Network).

Mindezek alapja az R mögött álló, rendkívül széles és erős nemzetközi közösség. Ingyenes programként bárki számára elérhető, nyílt forráskódú programként pedig jól bővíthető, illetve ez sokaknak a tudományra vonatkozó általános filozófiájával – “open science”, nyílt tudomány – is találkozik (így az enyémmel is). Számos statisztikus fejleszt R alá csomagokat, általában nagyon segítőkészek mind az esetleges hibák javításában, mind az új funkciók megvalósítására vonatkozóan. Több fórum érhető el (pl. a Stackoverflow), ahol a kezdőszintű egyszerű problémáktól a legspeciálisabb nehézségekig mindenben segítséget lehet kérni (és nem ritka, hogy a legnevesebb R fejlesztők válaszolnak!). Nagyon sok csomag jelen van a Github-on is, ami szintén kiváló platform az eszmecserére.

Az R különösen erős az eredmények kommunikálásban. Kiegészítő csomagokkal könnyedén lehetséges ún. dinamikus dokumentumok készítése, melyek együtt tartalmazzák a kódokat, és a kapcsolódó leírást.

A reprodukálható kutatás jegyében a cikkekkel együtt közzétett elemzések is nagyon gyakran R-ben íródtak, ezekből szintén sok ötlet meríthető.

2. fejezet

Az R szkriptek néhány alapvető jellemzője, az RStudio integrált fejlesztői környezet

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, utasításról utasításra.

Az RStudio fejlesztői környezet alapbeállításában a bal oldali rész alján látható a konzol, ahol közvetlenül beküldhetünk utasításokat az R-nek, illetve az – akár közvetlenül, akár a lent vázolt módon szkriptből – beküldött utasítások eredményei láthatóak. A konzol felett találjuk a megnyitott szkriptet, vagy szkripteket. 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). Amint volt róla szó, 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 egész szkript addig a sorig, amiben a kurzor áll, a **Ctrl+Alt+B** kombinációval, az egész szkript az aktuális sortól a végéig **Ctrl+Alt+E** kombinációval futtatható.

Az egyes utasításokat új sorban kell kezdeni (tehát enter-rel kell elválasztani egymástól). Elvileg egy sorba több utasítás is írható, ekkor az egyes utasításokat pontosvesszővel (;) kell elválasztani, de ezt minden körülmények között kerüljük.

Egy utasítás több sorba is átnyúlhat, ezt az R érzékeli, tehát, ha a sor végén még nem záródott be egy utasítás, akkor a következő sorban folytatja a feldolgozást. Azt, hogy új utasítást vár az R, onnan lehet látni, hogy a konzol elején a > jel látható. Ha az utasítás nem ér véget a sorban (ezt az R magától érzékeli, például onnan, hogy egy kinyitott zárójel nem lett bezárva a beküldött sorban), akkor automatikusan azt feltételezi, hogy ez azért van, mert a következő sorban folytatjuk az utasítást. Ilyenkor a konzol elején a + helyett a > jel látható. Ez jelzi, hogy a beküldött utasítást a következő folytatásának tekinti. Amint látja az R, hogy bezárult az utasítás, végrehajtja, és a konzol átugrik újra a > jelre: várja a következő utasítást. Ez a viselkedés egy gyakori hiba forrása: ha beküldünk egy utasítást, amiből véletlenül le hagyjuk a záró zárójelet, akkor az R várni fogja a folytatást. Ha azonban ezt nem vesszük észre, és beküldjük a következő utasítást, akkor nem azt fogja végre hajtani (ahogy várnánk), hanem az előző folytatásának tekinti, és úgy próbálja értelmezni. Az eredmény vagy hiba lesz, vagy az, hogy továbbra is + üzemmódban fogja várni az utasításokat, mi pedig nem kapunk eredményt. Ha ilyen történik, tehát küldjük be az utasításokat, amik teljesen helyesek, és mégsem kapunk eredményt, akkor érdemes megnézni, hogy nem + (folytatás) üzemmódban van-e az R. Ha igen, akkor küldjünk be záró zárójelet, ha ezzel sikerül lezárnunk az utasítást, akkor nyilván hibát kapunk, de legalább visszavehetjük az irányítást.

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. Fontos, hogy ezt betartsuk, ugyanis az RStudio funkcionalitása csak akkor fog működni, ha a fájlról tudja, hogy az egy R szkript, és ezt a kiterjesztés alapján azonosítja. 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 nyíl 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, illetve ha ki van kommentezve, akkor ezt megszünteti. Többsoros kommentre nincs külön jel R-ben, viszont RStudio-ban a **Ctrl+Shift+C** használható több sort kijelölve is, ekkor mindegyiket kommentezi (vagy eltünteti a kommentjelet, ha ki vannak kommentezve).

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 súgót 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. Mit jelent az, hogy információt tárolni? A változóban elmenthetünk információt (értékkadás), azt módosíthatjuk, majd kiolvashatjuk és felhasználhatjuk. Változóból tetszőleges számút létrehozhatunk. Elsőként meg kell ismerkednünk a változó fogalmával, a neki történő értékkaddással, és azzal, hogy milyen típusú adatokat tudunk változóban tárolni

3.1 Értékkadás

Változó értéket az értékkadás művelettel kap; ez kb. a „legyen egyenlő” módon olvasható ki. Az értékkadás jele az R-ben a `<-`. (A más programnyelveken megszokottabb `=`-t ne használjuk értékkadásra, 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 értékkadásra, ilyenkor értelemeszerűen fordul a helyzet, de ezt ritkán szokták alkalmazni.

Íme egy értékkadás:

```
a <- 1
```

Ami szembeötlő (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 értékkül, azt is meghatározta,

hogymilyen 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 változókkal 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, és ha `.` jellel kezdődik, akkor utána nem jöhet szám. (Bizonyos, úgynevezett foglalt szavakat, amiket az R nyelv használ, nem választhatunk változónévnek. Ezekből nagyon kevés van, így annyiban óvatosságnak kell lenni, hogy az R egy sor szokásos függvényét simán felüldefiniálhatjunk, ha létrehozunk olyan nevű változót.) 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 veszélyt, 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).

Az R-ben a tizedestörteket angol stílusban kell megadni, tehát a tizedesjelölő a pont, nem a vessző.

Így néz ki egy numerikus adattal történő értékadás:

```
szam <- 3.1
szam
```

```
## [1] 3.1
```

```
str(szam)
```

```
## num 3.1
```

Nézzük meg, hogy csakugyan case sensitive a nyelv:

```
SZAM
```

```
## Error: object 'SZAM' not found
```

```
Szam
```

```
## Error: object 'Szam' not found
```

```
szaM
```

```
## Error: 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] 2.4
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] 1.0 4.0 5.0 -2.0 3.5 10.0
```

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 nevezhetők; 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 indexeltm hiszen „egy szám” nincsen, az is vektor.)

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 három <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 tördel, 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 elnevezhetőek, 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 keverhetőek 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 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 ...
## $ 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      999
```

```
lista$sz
```

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

```
##      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 harm
```

```
##      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 harm
```

```
##      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 sűgő 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 quantile.default(): 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%  
## 2387.96
```

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

```
##      23%      50%      60%
## 2387.96 2977.00 3169.40
```

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): az argumentum nem numerikus vagy logikai: NA-
## t
## eredményez
```

```
##          [,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] 3130.16
```

```
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] 3130.16
```

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

```
## [1] 323.7243
```

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

```
## [1] 3130.16
```

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

```
## [1] 323.7243
```

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] 23.2381
##
## [[2]]
## [1] 129.8148
##
## [[3]]
## [1] 2944.587

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

##      age      lwt      bwt
## 23.2381 129.8148 2944.5873

sapply( birthwt, mean )

## Warning in mean.default(X[[i]], ...): az argumentum nem numerikus vagy logikai:
## NA-t eredményez

##      low      age      lwt      race      smoke      ptl
## 3.121693e-01 2.323810e+01 1.298148e+02 1.846561e+00 3.915344e-
01 1.957672e-01
##      ht      ui      ftv      bwt      nev
## 6.349206e-02 1.481481e-01 7.936508e-01 2.944587e+03 NA

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

## Warning in mean.default(x): az argumentum nem numerikus vagy logikai: NA-
t
## eredményez

## $low
## [1] 0.3121693 0.0000000
##
## $age
## [1] 23.2381 23.0000
##
## $lwt
## [1] 129.8148 121.0000
##
## $race
## [1] 1.846561 1.000000
##
## $smoke
## [1] 0.3915344 0.0000000
##
## $ptl
## [1] 0.1957672 0.0000000
##
## $ht
```



```
##      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] 3102.719

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

##      1      2      3
## 3102.719 2719.692 2805.284
```


6. fejezet

Data table: egy továbbfejlesztett adatkeret

Amint volt már róla szó korábban, az adatkeret (data frame) az alapvető struktúra a feldolgozandó adatok, adatbázisok tárolására és kezelésére az R-ben. Noha ennek a célnak megfelel, számos téren kiegészíthető, továbbfejleszthető. Az évek alatt két nagy lehetőség kristályosodott ki és ment át széleskörű használatba, mely ilyen továbbfejlesztést jelent: a `dplyr` csomag és a `data.table` csomag. Jelen fejezet a `data.table` működését és jellemzőit fogja bemutatni, különös tekintettel a hagyományos data frame-mel való összevetésre.

A `data.table` csomag első verziója 2008-ban jelent meg, eredeti megalkotója Matt Dowle. Nagyon erőteljes, gyors, erősen optimalizált, némi gyakorlás után logikus, kompakt, konzisztens, könnyen lekódolható és jól olvasható szintaktikájú, jól támogatott csomag. A `data.table`-nek semmilyen függősége nincs az R-en kívül (ott is törekednek a nagyon régi változatok támogatására is), így kifejezetten problémamentesen beépíthető R kódokba, csomagokba.

Központi weboldala: <https://rdatatable.gitlab.io/data.table/>. Github-oldala: <https://github.com/Rdatatable/data.table>. CRAN-oldala: <https://cran.r-project.org/web/packages/data.table/index.html>.

A `data.table` mint csomag egy azonos nevű új adatstruktúrát definiál; ez lényegében egy “továbbfejlesztett data frame”. Ez az új adatstruktúra, a data table egyrészt olyan lehetőségeket biztosít, amik valamilyen módon megvalósíthatóak lennének szokásos data frame-mel is, de csak lassabban/nehézkesebben/több hibalehetőséggel, másrészt elérhetővé tesz olyan funkciókat is, amik data frame-mel egyáltalán nem megoldhatóak.

A következőkben át fogjuk tekinteni ezek legfontosabb példait.

A gyakorlati szemléltetésekhez töltsük be a könyvtárat (a `data.table` nem jön

446. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

az alap R installációval, így ha korábban nem tettük meg, elsőként telepíteni kell):

```
library(data.table)
```

Ebben a fejezetben a magyar Nemzeti Rákregiszter adatait fogjuk példa adatbázisnak használni.

Elsőként töltsük be a következő fájlt, ami eleve `data.table` formátumban tartalmazza az adatokat:

```
RawData <- readRDS(url(paste0(
  "https://github.com/tamas-ferenci/RakregiszterVizualizator/",
  "raw/refs/heads/master/RawDataLongWPop.rds")))
```

Néhány esetben össze fogjuk vetni a `data.frame`-et a `data.table`-el, ehhez “minősítsük vissza” az adatbázist data frame-mé, és ezt mentjük el egy új változóba:

```
RawDataDF <- data.frame(RawData)
```

Érdeemes ránézni egy data table felépítésére:

```
str(RawData)
```

```
## Classes 'data.table' and 'data.frame': 1313280 obs. of 7 variables:
## $ County : chr "Baranya megye" "Baranya megye" "Baranya megye" "Baranya megye"
## $ Sex : chr "Férfi" "Férfi" "Férfi" "Férfi" ...
## $ Age : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Year : num 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 ...
## $ ICDCode : chr "C00" "C01" "C02" "C03" ...
## $ N : int 0 0 0 0 0 0 0 0 0 0 ...
## $ Population: num 9876 9876 9876 9876 9876 ...
## - attr(*, ".internal.selfref")=<externalptr>
## - attr(*, "sorted")= chr [1:4] "County" "Sex" "Age" "Year"
```

Ami feltűnhet, hogy az objektumnak egyaránt van `data.table` és `data.frame` osztálya, egyebekben azonban a fenti információk megfelelnek egy data frame által mutatott felépítésnek. A két osztály jelenléte egyfajta visszafele kompatibilitást¹ jelent: egy data table olyan számítógépen is betölthető, ahol nincs `data.table` csomag, és működni fog (természetesen csak mint hagyományos data frame). Ezen túl az is igaz ennek következtében, hogy olyan függvénynek, ami `data.frame`-et vár mindig átadható `data.table` is.

¹Arra azért vigyázni kell, hogy van példa arra, hogy pontosan ugyanaz a hívás mást ad vissza a data frame-nél és data table-nél. Egyébként ez a válasz arra a gyakran felmerülő kérdésre, hogy ha olyan jó a `data.table`, akkor miért nem győzik meg egyszerűen a fejlesztői az R fejlesztőit, hogy építsék be a tulajdonságait az R-es alap data frame-be is. Egyébként volt példa ilyenre is, de az előbbi ok miatt ez nem lehet általános, hiszen ez azt jelentené, hogy meglevő kódok működése is megváltozna, ami végeláthatlan sok R kód működését ronthatná el. Ilyen módosítást ma már nem igen lehet megtenni a `data.frame`-mel.

Visszatérve a tábla felépítésére, a fentiek alapján már elmondható, hogy a tábla mit tartalmaz: az új rákos esetek előfordulását Magyarországon évenként (2000-től 2018-ig), megyénként, nemenként, életkoronként (ez 5 éves felbontású, tehát a 40 igazából azt jelenti, hogy “40-45 év”), és a rák típusa szerint. Ez utóbbi ún. BNO-kóddal van megadva: a Betegségek Nemzetközi Osztályozása (BNO, angol rövidítéssel ICD) egy nemzetközileg egységes rendszer, mely minden betegséghez egy kódot rendel. A kód első karaktere egy betű, ez a főcsoport; a rákos betegségek a C főcsoportban, illetve a D elején vannak, a második és harmadik karakter egy szám, ami konkrét betegséget vagy betegségcsoportt azonosít; például C00 az ajak rosszindulatú daganata, C01 a nyelvgyök rosszindulatú daganata és így tovább². Az esetek számát az N nevű változó tartalmazza, a háttérpopuláció lélekszámát³, tehát, hogy hány fő volt adott évben, adott megyében, adott nemben, adott életkorban – pedig a **Population**. (Ez tehát azonos lesz azokra a sorokra, amelyek csak a rák típusában térnek el, hiszen ezekre a háttérpopuláció lélekszáma természetesen ugyanaz.)

6.1 Sebesség és nagyméretű adatbázisok kezelése

Ez a probléma a legtöbb szokásos elemzési feladatnál nem jelentkezik, itt sem fogunk rá részletes példát nézni, de röviden érdemes arról megemlékezni, hogy a hagyományos adatkeret (data frame) adatstuktúra nem szerencsés, ha nagyméretű adatbázisokat kell kezelünk.

Az első probléma kapásból az adatok beolvasásánál fog jelentkezni: a **read.csv** (és társai) egész egyszerűen lassúak. Pár százezer sorig ennek semmilyen érzékelhető hatása nincsen, mert még így is elég gyors a beolvasás, így a legtöbb feladatnál ez a probléma nem jelentkezik, de millió soros, több millió soros adatbázisoknál, ha a tábla mérete több gigabájt vagy több tíz gigabájt, akkor a beolvasás a méret növekedtével gyorsan lassul, míg végül teljesen reménytelené válik. A **data.table** definiálja az **fread** függvényt mely ezzel szemben villámgyors, és még ilyen méretű adatok beolvasásánál is elfogadható sebességet produkál. (Az **fread**-nek ezen kívül van pár további előnye is az R beépített beolvasó függvényeihez képest, olyan, amik kis méretű adatbázisoknál is érdekesek lehetnek, például nagyon okosan detektálja az oszlopelválasztókat és az oszloptípusokat.) Hasonló a helyzet kiírásnál: a **write.csv** és társai nagyon nagy adatbázisoknál elfogadhatatlanul lassúak lesznek, de a **data.table** könyvtár **fwrite** függvénye ilyenkor is jól működik.

A második probléma, hogy még ha valahogy be is olvastuk az adatbázist a memóriába, akkor is bajban leszünk az adattranszformációkkal: a data frame nincs túl jól optimalizálva ilyen szempontból, egy sor művelet nagyon lassú. Ismét

²A kód folytatható, ami finomabb felbontást ad, például a C00.0 a felső ajak külső felszínének daganata, a C00.1 az alsó ajak külső felszínének daganata stb., de a táblázatunk a háromjegyű besorolást tartalmazza.

³Ez ún. évközepi lélekszám, tehát az év alatti – folyamatosan változó – lélekszámok átlaga. Ezért lehet az értéke törtszám is.

csak: ennek kis, közepes és a legtöbb terület mércéje szerinti nagy adatbázisoknál nincs jelentősége, mert még így is gyors, de a nagyon nagy adatbázisoknál bajban leszünk data frame-et használva. A data table ezzel szemben nagyon jól optimalizált, képest többmagú processzoroknál bizonyos műveletek párhuzamos végrehajtására is, így az adattranszformációs műveleteknél⁴ (aggregáció, táblaegyesítések, de akár új változó létrehozása) sokkal jobb sebességet tud produkálni.

A fentieket többféle benchmark vizsgálat is megerősíti.

6.2 Jobb kiíratás

A data frame kiíratásánál (tehát ha egyszerűen beírjuk, hogy `RawDataDF`, ami ekvivalens a `print(RawDataDF)` függvény meghívásával) az alapbeállítás az, hogy kiírja a konzolra az első jó sok sorát az adatbázisnak⁵. Ez nem túl praktikus: az 587. sor ismerete jellemzően nem sokat ad hozzá az első 586-hoz, cserében hosszasan kell görgetnünk a rengeteg sor miatt, hogy elérjünk a kiíratás tetejére, aminek viszont volna jelentősége, mert ott látjuk az oszlopok neveit. (Nem véletlenül gyakori, hogy sokan eleve a `head(RawDataDF)` típusú kéréssel íratják ki a data frame-eket!)

A data table alapértelmezett kiíratása okosabb, mert csak az első néhány és az utolsó néhány sort⁶ írja ki:

RawData

```
## Key: <County, Sex, Age, Year>
##           County   Sex   Age  Year ICDCode      N Population
##           <char> <char> <num> <num> <char> <int>      <num>
##      1: Baranya megye  Férfi    0  2000    C00        0    9876.0
##      2: Baranya megye  Férfi    0  2000    C01        0    9876.0
##      3: Baranya megye  Férfi    0  2000    C02        0    9876.0
##      4: Baranya megye  Férfi    0  2000    C03        0    9876.0
##      5: Baranya megye  Férfi    0  2000    C04        0    9876.0
##      ---
## 1313276:   Zala megye    Nő    85  2018    D06        0    4483.5
## 1313277:   Zala megye    Nő    85  2018    D07        0    4483.5
## 1313278:   Zala megye    Nő    85  2018    D09        0    4483.5
```

⁴Ezek egy részénél nem kell külön függvényt hívni, csak “maga a data table” gyorsabb lesz mint a data frame. Más részénél szükség van egy külön függvényre, például a táblaegyesítésnél a `merge`-re. De ez is gyorsabb lesz, aminek a hátterében az van, hogy a `data.table`-nek van saját, ugyanilyen nevű függvénye (`data.table::merge`), és ez fog a data frame-hez tartozó alapváltozat, tehát a `base::merge` helyett futni.

⁵Egész pontosan annyit, amennyi a `max.print` opció értéke; ez a `getOption("max.print")` paranccsal kérdezhető le. Az alapbeállítása tipikusan 1000.

⁶A precizitás kedvéért: ezt csak akkor teszi, ha a sorok száma nagyobb mint a `datatable.print.rows` opció értéke, ami alapbeállítás szerint 100. De ez is logikus: kis adatbázisnál érdemes az egészet kiíratni, hiszen úgy is áttekinthető, nagyoknál lesz fontos csak az első néhány és az utolsó néhány sor kiíratása.

```
head(RawDataDF[RawDataDF$Year == 2010 & RawDataDF$Age >= 40 &
  RawDataDF$County %in% c("Budapest",
    "Pest megye") &
```

486. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

```
RawDataDF$Sex == "Férfi" &
RawDataDF$ICDCode == "C18",])
```

```
##           County Sex Age Year ICDCode    N Population
## 146899 Budapest Férfi  40 2010     C18     3    57445.5
## 148723 Budapest Férfi  45 2010     C18    10    42410.0
## 150547 Budapest Férfi  50 2010     C18    17    45329.0
## 152371 Budapest Férfi  55 2010     C18    44    55633.5
## 154195 Budapest Férfi  60 2010     C18    59    45170.0
## 156019 Budapest Férfi  65 2010     C18   120    39588.0
```

A dolog tökéletesen működik, ámde nem túl kényelmes: folyton be kell írni a `RawDataDF`-t a feltételek közé. A kód hosszú, lassabb megírni, és az olvashatóság is romlik. Fontos hangsúlyozni, hogy ez nem hagyható el, és teljesen igaza is van az R-nek, hogy nem hagyható el: `Year` nevű változó nem létezik, tehát teljes joggal ad hibát, ha előle – vagy bármelyik másik elől – elhagyjuk a data frame nevét.

Mégis: a gyakorlatban az esetek 99,99%-ában, ha egy változó nevére hivatkozunk miközben egy adatkeret sorindexelését végezzük, akkor azt természetesen úgy értjük, hogy *annak az adatkeretnek* az adott nevű oszlopa (és nem egy külső változó). Éppen emiatt a data table megengedi ezt a szintaktikát: ha pusztán egy változó nevére hivatkozunk, akkor ő megnézi, hogy nincs-e olyan nevű oszlopa az indexelt adattáblának, és ha van, akkor úgy veszi, hogy arra szerettünk volna hivatkozni. Éppen ezért az alábbi kód `data.frame`-mel nem, de `data.table`-lel működik:

```
RawData[Year == 2010 & Age >= 40 &
        County %in% c("Budapest", "Pest megye") &
        Sex == "Férfi" & ICDCode == "C18",]
```

```
## Key: <County, Sex, Age, Year>
##           County Sex Age Year ICDCode    N Population
##           <char> <char> <num> <num> <char> <int>      <num>
## 1: Budapest Férfi  40 2010     C18     3    57445.5
## 2: Budapest Férfi  45 2010     C18    10    42410.0
## 3: Budapest Férfi  50 2010     C18    17    45329.0
## 4: Budapest Férfi  55 2010     C18    44    55633.5
## 5: Budapest Férfi  60 2010     C18    59    45170.0
## 6: Budapest Férfi  65 2010     C18   120    39588.0
## 7: Budapest Férfi  70 2010     C18    80    27335.5
## 8: Budapest Férfi  75 2010     C18    75    22253.5
## 9: Budapest Férfi  80 2010     C18    72    15775.5
## 10: Budapest Férfi  85 2010     C18    35    10922.0
## 11: Pest megye Férfi  40 2010     C18     8    48086.0
## 12: Pest megye Férfi  45 2010     C18     6    36113.5
## 13: Pest megye Férfi  50 2010     C18    14    37167.0
```


6.3. KÉNYELMESEBB SORINDEXELÉS (SOR-SZŰRÉS ÉS -RENDEZÉS)49

```
## 14: Pest megye Férfi 55 2010 C18 32 41154.0
## 15: Pest megye Férfi 60 2010 C18 43 32527.0
## 16: Pest megye Férfi 65 2010 C18 56 26071.0
## 17: Pest megye Férfi 70 2010 C18 62 16926.5
## 18: Pest megye Férfi 75 2010 C18 45 11964.5
## 19: Pest megye Férfi 80 2010 C18 23 7015.0
## 20: Pest megye Férfi 85 2010 C18 11 4250.5
##          County Sex Age Year ICDCode N Population
```

A kapott kód világosabb, gyorsabban beírható és jobban olvasható!

A data table azt is megengedi, hogy a vesszőt elhagyjuk (a data frame nem, ott hibát adna ha nem írnánk vesszőt!):

```
RawData[Year == 2010 & Age >= 40 &
        County %in% c("Budapest", "Pest megye") &
        Sex == "Férfi" & ICDCode == "C18"]
```

```
## Key: <County, Sex, Age, Year>
##          County Sex Age Year ICDCode N Population
##          <char> <char> <num> <num> <char> <int> <num>
## 1: Budapest Férfi 40 2010 C18 3 57445.5
## 2: Budapest Férfi 45 2010 C18 10 42410.0
## 3: Budapest Férfi 50 2010 C18 17 45329.0
## 4: Budapest Férfi 55 2010 C18 44 55633.5
## 5: Budapest Férfi 60 2010 C18 59 45170.0
## 6: Budapest Férfi 65 2010 C18 120 39588.0
## 7: Budapest Férfi 70 2010 C18 80 27335.5
## 8: Budapest Férfi 75 2010 C18 75 22253.5
## 9: Budapest Férfi 80 2010 C18 72 15775.5
## 10: Budapest Férfi 85 2010 C18 35 10922.0
## 11: Pest megye Férfi 40 2010 C18 8 48086.0
## 12: Pest megye Férfi 45 2010 C18 6 36113.5
## 13: Pest megye Férfi 50 2010 C18 14 37167.0
## 14: Pest megye Férfi 55 2010 C18 32 41154.0
## 15: Pest megye Férfi 60 2010 C18 43 32527.0
## 16: Pest megye Férfi 65 2010 C18 56 26071.0
## 17: Pest megye Férfi 70 2010 C18 62 16926.5
## 18: Pest megye Férfi 75 2010 C18 45 11964.5
## 19: Pest megye Férfi 80 2010 C18 23 7015.0
## 20: Pest megye Férfi 85 2010 C18 11 4250.5
##          County Sex Age Year ICDCode N Population
```

Fontos azonban, hogy ez csak ebben az esetben, tehát sorindexelésnél használható: ha nincs vessző, akkor automatikusan úgy veszi, hogy amit beírtunk, az sorindex (e megállapodás nélkül nem tudhatná, hogy mit akartunk indexelni).

A tény, hogy nem kell hivatkozni az adatkeret nevére, nem csak szűrésnél igaz,

hanem rendezésnél is. Ezt ugyanis az `order` függvény valósítja meg, ami elérhető volt a data frame-hez is, csak ott ilyen módon kellett használnunk:

```
head(RawDataDF[order(RawDataDF$N),])
```

```
##           County  Sex Age Year ICDCCode N Population
## 1 Baranya megye Férfi  0 2000      C00 0         9876
## 2 Baranya megye Férfi  0 2000      C01 0         9876
## 3 Baranya megye Férfi  0 2000      C02 0         9876
## 4 Baranya megye Férfi  0 2000      C03 0         9876
## 5 Baranya megye Férfi  0 2000      C04 0         9876
## 6 Baranya megye Férfi  0 2000      C05 0         9876
```

A data table azonban itt is megengedi⁷ a fenti – nagyon logikus – egyszerűsítést:

```
RawData[order(N),]
```

```
##           County  Sex Age Year ICDCCode      N Population
##           <char> <char> <num> <num> <char> <int>      <num>
##      1: Baranya megye Férfi  0 2000      C00  0      9876.0
##      2: Baranya megye Férfi  0 2000      C01  0      9876.0
##      3: Baranya megye Férfi  0 2000      C02  0      9876.0
##      4: Baranya megye Férfi  0 2000      C03  0      9876.0
##      5: Baranya megye Férfi  0 2000      C04  0      9876.0
##      ---
## 1313276: Budapest      Nő  60 2008      C50 307      64674.0
## 1313277: Budapest      Nő  55 2000      C50 308      67218.5
## 1313278: Budapest      Nő  55 2002      C50 308      69118.0
## 1313279: Budapest      Nő  70 2018      C44 331      56508.0
## 1313280: Budapest      Nő  55 2003      C50 350      69396.5
```

Természetesen a “szűrés” és “rendezés” csak felhasználói szempontból két külön művelet. Az R számára a kettő ugyanaz: sorindexelés, csak annyi eltéréssel, hogy az előbbi esetben logikai vektort kap, az utóbbiban pedig számvektort (hiszen az `order` egyszerűen megadja sorban minden elemre, hogy az adott elem hányadik a nagyság szerinti sorrendben).

6.4 Kibővített oszlopindexelés: oszlop-kiválasztás és oszlop-létrehozás műveletekkel

Hagyományos data frame esetén a vessző után jön az oszlopindexelés, ami egy dolgot jelenthet: oszlopok kiválasztását. Tehát, dönthetünk, hogy mely

⁷Egyébként ez utóbbi esetben nem ugyanaz az `order` fut le: a `data.table` definiál egy saját `order`-t, tehát az előbbi esetben a `base::order`, az utóbbinál a `data.table::order` fut. A `data.table` csomag `order`-je egyébként is okosabb, például sokkal kényelmesebb ha több változó szerint és változó irányban kell rendeznünk: egyszerűen fel kell sorolnunk az `order`-en belül a változókat, és amelyek szerint csökkenő sorrendben akarunk rendezni, ott ki kell tennünk a változó neve elé egy `-` jelet.

6.4. KIBŐVÍTETT OSZLOPINDEXELÉS: OSZLOP-KIVÁLASZTÁS ÉS OSZLOP-LÉTREHOZÁS MŰVELETEK

oszlopokat kérjük (és melyeket nem), de más lehetőségünk nincs. Oszlopok kiválasztását célszerű mindig névvel és nem számmal végeznünk (hogy a kód az adatbázis esetleges későbbi módosításaira robusztusabb legyen, ne romoljon el új oszlop beszúrásától vagy törlésétől, valamint, hogy önállóan is jobban olvasható legyen a kód). Ekkor lényegében egy sztring-vektort kell átadnunk. A példa kedvéért itt – az előzőekkel szemben – a 40-45 éves budapesti férfiak vastagbélrákos eseteire szorítsuk meg magunkat, viszont tartsuk meg az összes évet. Ez esetben logikus csak az évet – és persze az N-et és a Population-t – kiíratni, hiszen a többi konstans:

```
head(RawDataDF[RawDataDF$Age == 40 &
  RawDataDF$County == "Budapest" &
  RawDataDF$Sex == "Férfi" &
  RawDataDF$ICDCode == "C18",
  c("Year", "N", "Population")])
```

```
##      Year N Population
## 145939 2000 8    51602.0
## 146035 2001 4    47836.0
## 146131 2002 4    45296.5
## 146227 2003 4    43632.5
## 146323 2004 4    43085.0
## 146419 2005 5    43442.5
```

Ez a szintaktika a `data.table`-el is működik⁸:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
  ICDCode == "C18",
  c("Year", "N", "Population")]
```

```
##      Year      N Population
##      <num> <int>      <num>
## 1: 2000      8    51602.0
## 2: 2001      4    47836.0
## 3: 2002      4    45296.5
## 4: 2003      4    43632.5
## 5: 2004      4    43085.0
```

⁸Valójában van egy különbség, ami akkor jelentkezik, ha a kiválasztandó oszlopok neveit eltávolítjuk egy változóban, és az indexelésnél ezt a változót szeretnénk felhasználni ahelyett, hogy kézzel beírjuk a neveket. Legyen például `colsel <- c("Year", "N", "Population")`. Ekkor a `data.frame`-nél mindegy, hogy a `RawDataDF[, c("Year", "N", "Population")]` vagy a `RawDataDF[, colsel]` formát használjuk, az eredmény ugyanaz lesz. Ami logikus is, hiszen látszólag ugyanazt írtuk be kétszer. Nagyon meglepő módon azonban a `data.table`-nél nem mindegy: a `RawData[, c("Year", "N", "Population")]` működni fog, de a `RawData[, colsel]` nem! Ennek az az oka, hogy `RawData[, colsel]` összeakad egy szintaktikával, amit később fogunk látni, és amelyben ez azt jelentené, hogy “válaszd ki a `colsel` nevű oszlopot és add vissza vektorként”. Ami természetesen nem fog sikerülni, hiszen ilyen nevű oszlop nincs. Van azonban megoldás: ha erre volna szükségünk akkor vagy a `RawData[, ..colsel]` vagy a `RawData[, colsel, with = FALSE]` alakot kell használnunk.

526. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

```
## 6: 2005      5      43442.5
## 7: 2006      5      44511.5
## 8: 2007      6      46903.5
## 9: 2008      4      50505.5
## 10: 2009     6      54015.0
## 11: 2010     3      57445.5
## 12: 2011     8      60721.0
## 13: 2012     7      62471.5
## 14: 2013     5      63746.5
## 15: 2014     8      66250.5
## 16: 2015    13      70511.5
## 17: 2016    11      74622.0
## 18: 2017     6      77902.0
## 19: 2018    10      80555.0
```

A `data.table`-nek van azonban egy saját, külön szintaktikája erre, és célszerű is azt megszokni és használni mindig, mert a későbbi funkciókat az teszi elérhetővé:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCode == "C18",
        .(Year, N, Population)]
```

```
##      Year      N Population
##      <num> <int>      <num>
## 1: 2000      8    51602.0
## 2: 2001      4    47836.0
## 3: 2002      4    45296.5
## 4: 2003      4    43632.5
## 5: 2004      4    43085.0
## 6: 2005      5    43442.5
## 7: 2006      5    44511.5
## 8: 2007      6    46903.5
## 9: 2008      4    50505.5
## 10: 2009     6    54015.0
## 11: 2010     3    57445.5
## 12: 2011     8    60721.0
## 13: 2012     7    62471.5
## 14: 2013     5    63746.5
## 15: 2014     8    66250.5
## 16: 2015    13    70511.5
## 17: 2016    11    74622.0
## 18: 2017     6    77902.0
## 19: 2018    10    80555.0
```

Megjegyzendő, hogy a `.` egyszerűen egy rövidítés, amit a `data.table` csomag bevezet arra, hogy `list`, magyarul itt az történik, hogy egy listát kell átadnunk⁹,

⁹Ez elsőre meglepő lehet, de valójában teljesen logikus: ha visszaemlékszünk, akkor már

6.4. KIBŐVÍTETT OSZLOPINDEXELÉS: OSZLOP-KIVÁLASZTÁS ÉS OSZLOP-LÉTREHOZÁS MŰVELETEK

benne az – idézőjelek nélküli – oszlopnevekkel. A listás megoldás előnye, hogy valójában nem kötelező explicite kiírni, hogy `.` majd felsorolni a változóneveket zárójelben, bármilyen függvényt is használhatunk a vessző után ami listát ad eredményül. Később látunk majd erre példát.

Az is érthető a listás megoldás fényében, hogy `data table`-lel átnevezhetünk változót úgymond “menet közben” (`data frame`-mel már ezt sem lehetett!):

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCode == "C18",
        .(Year, Esetszam = N, Lelekszam = Population)]
```

##	Year	Esetszam	Lelekszam
##	<num>	<int>	<num>
## 1:	2000	8	51602.0
## 2:	2001	4	47836.0
## 3:	2002	4	45296.5
## 4:	2003	4	43632.5
## 5:	2004	4	43085.0
## 6:	2005	5	43442.5
## 7:	2006	5	44511.5
## 8:	2007	6	46903.5
## 9:	2008	4	50505.5
## 10:	2009	6	54015.0
## 11:	2010	3	57445.5
## 12:	2011	8	60721.0
## 13:	2012	7	62471.5
## 14:	2013	5	63746.5
## 15:	2014	8	66250.5
## 16:	2015	13	70511.5
## 17:	2016	11	74622.0
## 18:	2017	6	77902.0
## 19:	2018	10	80555.0

Ez már utat mutat a következő, igazi újdonsághoz.

Előtte még említsük meg, hogy a `data table` egyik jellegzetessége, hogy a `RawData[, .(Year)]` típusú hívások mindig `data table`-t adnak vissza¹⁰. Ha egyetlen változót választunk ki, de azt vektorként szeretnénk visszakapni (ez a kérdés nyilván csak egyetlen változó kiválasztásakor merül fel), akkor használjuk

a `data.frame`-nél is láttuk, hogy az igazából az oszlopokból, mint vektorokból alkotott lista. Innen nézve teljesen érthető, hogy az oszlopokat egy lista elemeiként kell felsorolni!

¹⁰Ez nem nyilvánvaló: a `RawDataDF[, "Year"]` egy vektor lesz! Természetesen a `RawDataDF[, c("Year", "County")]` megint csak `data frame`; vagyis lényegében az történik, hogy a `data frame` automatikusan egyszerűsít: ha lehet – azaz egyetlen változót (oszlopot) választottunk ki – akkor egyszerűsíti vektorrá, ha nem, mert többet, akkor marad a `data frame`. Ez kényelmes is lehet, de közben mégis csak egy inkonzisztencia, hogy ugyanolyan típusú hívások eredménye teljesen eltérő adatstruktúra is lehet. Ezzel szemben a `data.table`-nél a `RawData[, .(...)]` típusú hívások *mindig* `data table`-t adnak vissza.

a `RawData$Year` vagy a `RawData[["Year"]]` formát¹¹.

Ez eddig nem nagy változás, még csak azt sem igazán lehet mondani, hogy az előzőhöz hasonló kényelmi továbbfejlesztés, hiszen ez a szintaktika nem sokkal tér el a korábbtól. Az igazán érdekes rész azonban most jön, a `data.table` ugyanis lehetővé tesz valamit, ami a `data.frame`-nél fel sem merült: nem csak passzívan kiválaszthatunk oszlopokat, hanem *műveleteket is végezhetünk* velük, így új oszlopokat hozva létre! Lényegében “on the fly”, azaz menet közben végezhetünk műveleteket és hozhatunk létre új oszlopokat, anélkül, hogy azokat fizikailag le kellene tárolnunk az adatbázisba. A data table vessző utáni pozíciójában tehát

Például a rákos megbetegedéseknél fontos az incidencia, tehát a lélekszámhoz viszonyított előfordulás. (Értelemszerűen nem mindegy, hogy 10 vagy 10 ezer ember közül került ki 1 rákos adott évben.) Ezt tipikusan 100 ezer lakosra vonatkoztatva szokták megadni. Nézzük meg a következő data table-t használó megoldást:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCCode == "C18",
        .(Year, N, Population, Inc = N / Population * 1e5)]
```

##		Year	N	Population	Inc
##		<num>	<int>	<num>	<num>
##	1:	2000	8	51602.0	15.503275
##	2:	2001	4	47836.0	8.361903
##	3:	2002	4	45296.5	8.830704
##	4:	2003	4	43632.5	9.167478
##	5:	2004	4	43085.0	9.283974
##	6:	2005	5	43442.5	11.509467
##	7:	2006	5	44511.5	11.233052
##	8:	2007	6	46903.5	12.792222
##	9:	2008	4	50505.5	7.919930
##	10:	2009	6	54015.0	11.108026
##	11:	2010	3	57445.5	5.222341
##	12:	2011	8	60721.0	13.175014
##	13:	2012	7	62471.5	11.205110
##	14:	2013	5	63746.5	7.843568
##	15:	2014	8	66250.5	12.075381
##	16:	2015	13	70511.5	18.436709
##	17:	2016	11	74622.0	14.740961
##	18:	2017	6	77902.0	7.701985
##	19:	2018	10	80555.0	12.413879

Azaz az `Inc` oszlopot létrehoztuk a nélkül, hogy előzetesen azt le kellett volna tárolnunk magába az adatbázisba! Menet közben számoltuk ki, és még nevet is adtunk neki. Az oszlopok tehát itt, a vessző utáni pozícióban úgy viselkednek

¹¹Elvileg a `RawData[, Year]` is használható, de ezt talán jobb kerülni, ritkán fordul elő.

6.4. KIBŐVÍTETT OSZLOPINDEXELÉS: OSZLOP-KIVÁLASZTÁS ÉS OSZLOP-LÉTREHOZÁS MŰVELETEK

egy data table-nél mintha szokásos változók lennének!

Ebből is adódik, hogy a lehetőségeink még ennél is bővebbek: nem csak egyszerű aritmetikai műveleteket végezhetünk egy oszloppal (vagy épp több oszloppal! – mint arra ez előbbi kód is példát mutat), hanem *bármilyen* R függvényt rájuk eresztgetünk! Tekintsünk példának a következő kódot, mely megadja, hogy a 40-45 éves budapesti férfiak körében összesen hány vastagbélrákos eset volt az adatbázis által lefedett 19 év alatt:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCode == "C18",
        .(N = sum(N))]
```

```
##          N
##      <int>
## 1:    121
```

Az N oszlop egy vektor, tehát azon túl, hogy oszthatjuk – elemenként – egy másik vektorral, mint ahogy az előbbi esetben tettük, nyugodtan összegezzhetjük is példának okáért. Ebből mellesleg az is látszik, hogy még az sem jelent problémát, ha a művelet által visszaadott eredménynek a hossza is eltér a bemenő változóétól! Hiszen a `sum(N)` 1 hosszú, míg a `Year` 19. (Az azonban fontos, hogy itt már a `Year` nem szerepel a kiválasztott oszlopok között: megtarthattuk volna a `Year`-t is, de mivel az 19 hosszú, így a mellette lévő oszlopban ugyanaz az összeg 19-szer meg lett volna ismételve.)

A fenti példákban egyszerre szűrtünk sorokat és számoltunk oszlopokat. (Ez természetesen nem kötelező, lehet csak az egyiket csinálni a másik nélkül.) Egyetlen példa a `data.table` optimalizálására: ilyenkor nem azt csinálja, hogy leszűri az egész adatbázist, és aztán végzi az oszlopműveleteket, hanem először megnézi, hogy mely oszlopokra van egyáltalán szükség – például csak a `Year`-re, `N`-re és `Population`-re – és ilyenkor *csak* azokat szűri le, így kerülve el, hogy olyan oszlopok szűrését is el kelljen végeznie, amik később nem is jelennek meg az eredményben. Ez azért lehetséges, mert a `data.table` “egyben látja” az egész feladatot, és így tud ilyen optimalizálásokat tenni.

Visszatérve, a dolog még jobban kombinálható: legyen a példa kedvéért a feladatunk az, hogy számoljuk ki az egész 19 éves periódusra az incidenciát. (Egy pillanatra érdemes itt megállni, és végiggondolni, hogy mi egyáltalán az ehhez szükséges művelet!) Íme a megvalósítás `data.table` használatával:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCode == "C18",
        .(Inc = sum(N) / sum(Population) * 1e5)]
```

```
##          Inc
##      <num>
## 1: 11.1515
```

Amint láthatjuk, tetszőleges komplexitású műveletet, számítást elvégezhetünk a

vessző után! És ezt szó szerint kell érteni: *bármilyen* R függvényt használhatunk az oszlopindexelés pozíciójában, a vessző után, bármilyen műveletet vagy számítást végezhetünk (tehát még csak olyan megkötés sincs, hogy csak bizonyos függvényeket, műveleteket tesz csak elérhetővé a `data.table`). Íme egy példa; lognormális eloszlást illesztünk az esetszámok különböző években mért értékeiből kapott eloszlásra:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
  ICDCode == "C18",
  .(fitdistrplus::fitdist(N, "lnorm")$estimate["meanlog"])]

##          V1
##          <num>
## 1: 1.772807
```

Szépen látszik itt is, hogy nyugodtan használhatjuk az `N`-et csak így, minden további nélkül – ugyanúgy viselkedik, mint egy szokásos változó, ugyanúgy használhatjuk egy számítás során.

Ráadásul, ha visszaemlékszünk, akkor szerepelt, hogy a vessző utáni pozícióban egy listának kell szerepelnie – de ezt előállíthatja egy függvény is! Például a `fitdistrplus::fitdist` eredményének `estimate` nevű komponense egy vektor. De ha ez `as.list`-tel átalakítjuk, akkor egy listát kapunk, így közvetlenül átadható a vessző utáni pozícióban (természetesen ilyenkor `.` nem kell, hiszen az `as.list` *elvé* egy listát ad vissza!):

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
  ICDCode == "C18",
  as.list(fitdistrplus::fitdist(N, "lnorm")$estimate)]

##      meanlog      sdlog
##      <num>      <num>
## 1: 1.772807 0.3902478
```

Ez tehát már messze-messze nem csak egyszerűen oszlopkiválasztás, amire itt módunk van, ha `data.table`-t használunk.

Egyetlen megjegyzés a végére: mi van akkor, ha kíváncsiak vagyunk arra, hogy hány sor van egy adattáblában (esetleg szűkítés után)? A `RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" & ICDCode == "C18", length(N)]` kézenfekvő megoldás, de nem túl elegáns (miért pont az `N` hosszát néztük meg? bármi más is ugyanezt az eredményt adná!). Erre a célra a `data.table` bevezet egy speciális szimbólumot, a `.N`-et, ami egyszerűen visszaadja¹² a sorok számát:

¹²Észrevehető, hogy az eredmény egy szám lesz, nem egy data table. Ennek az oka, hogy a `.N` – hiába van a nevében egy `.` – nem egy lista. Ha data frame-et szeretnénk visszakapni, akkor a korábbiakkal összhangban azt kell írunk, hogy `.(.N)`.


```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCCode == "C18", .N]
```

```
## [1] 19
```

6.5 Csoportosítás (aggregáció)

A `data.table` második, rendkívül erőteljes bővítése a hagyományos data frame funkcionalitásához képest a csoportosítás (aggregáció) lehetősége. A `data.table` bevezet egy *harmadik* pozíciót a szögletes zárójelen belül: megtehetjük, hogy *két* vesszőt teszünk ki a szögletes zárójelen belül, ez esetben az első vessző előtt van a sorindexelés (ahogy eddig is), az első és a második vessző között az oszlopkiválasztás és -számítás (ahogy eddig is), viszont a második vessző *után* megadhatunk egy listát egy vagy több változóból. (A `.` ugyanúgy használható a `list` helyett. Megadhatunk sztring-vektort is, benne a változók neveivel; ez különösen jól jön akkor, ha gépi úton állítjuk elő, hogy mik ezek a változók.) Mi fog ilyenkor történni? A `data.table` elsőként végrehajtja a sorok szűrését, ha kértünk ilyet, ezután pedig az új, harmadik pozícióban megadott változó vagy változók szerint csoportokat képez. Mit jelent az, hogy “csoport”? Azok a sorai a táblának, amelyekben a csoportosító változó egy adott értéket vesz fel: ahány lehetséges értéke van a csoportosító változónak a táblában, annyi csoport képződik, úgy, hogy csoporton belül a csoportosító változó homogén lesz. Ezt követően a `data.table` végrehajtja a megadott oszlopkiválasztásokat és/vagy oszlopműveleteket csoportonként *külön-külön*, végül pedig a kapott eredményeket újra összerakja egy táblába, úgy, hogy mindegyik csoport eredménye mellé beteszi oszlopként azt, hogy ott mi volt a csoportosító változó értéke. Az egyes csoportok abban a sorrendben fognak szerepelni az eredményben, ahogy egymás után jöttek a kiinduló táblában.

A jobb megértés kedvéért nézzünk egy gyakorlati példát! Kíváncsiak vagyunk az egész időintervallumra vonatkozó incidenciára, de az *összes* rák-típus esetén külön-külön megadva. Mit tudunk tenni? Fent láttuk a kódot, mely egy adott típusra ezt kiszámolja. Az remélhetőleg senkinek nem jut az eszébe, hogy kézzel lefuttassa először C00-val, aztán C01-gyel, aztán C02-vel... Működőképebb megoldás ennek valamilyen R paranccsal történő automatizálása. Rosszabb esetben a `for` jut az eszünkbe, jobb esetben az `apply` család valamely tagja. (A `for`-ciklus rosszabb eset, mert az R-ben a legtöbb esetben illendő kerülni, és jelen esetben tényleg meg is oldható a probléma megfelelő `apply` használatával, így ez is a célszerű választás.) Ha azonban a `data.table`-t használjuk, akkor még csak erre sincs szükség!

Nézzük ugyanis meg a következő hívást:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi",
        .(Inc = sum(N) / sum(Population) * 1e5),
        .(ICDCCode)]
```

586. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

##	ICDCode	Inc
##	<char>	<num>
## 1:	C00	0.36864474
## 2:	C01	2.21186843
## 3:	C02	2.58051316
## 4:	C03	1.01377303
## 5:	C04	2.48835198
## 6:	C05	0.73728948
## 7:	C06	0.55296711
## 8:	C07	0.92161184
## 9:	C08	0.73728948
## 10:	C09	1.65890132
## 11:	C10	3.13348027
## 12:	C11	1.19809540
## 13:	C12	0.46080592
## 14:	C13	4.42373685
## 15:	C14	0.82945066
## 16:	C15	4.14725330
## 17:	C16	6.72776646
## 18:	C17	1.75106250
## 19:	C18	11.15150331
## 20:	C19	2.58051316
## 21:	C20	7.09641120
## 22:	C21	0.55296711
## 23:	C22	3.87076974
## 24:	C23	1.01377303
## 25:	C24	1.47457895
## 26:	C25	7.37289475
## 27:	C26	0.46080592
## 28:	C30	0.64512829
## 29:	C31	1.10593421
## 30:	C32	7.46505593
## 31:	C33	0.09216118
## 32:	C34	30.78183558
## 33:	C37	0.46080592
## 34:	C38	1.10593421
## 35:	C39	0.36864474
## 36:	C40	1.01377303
## 37:	C41	2.94915790
## 38:	C43	17.41846385
## 39:	C44	43.13143429
## 40:	C45	0.64512829
## 41:	C46	0.46080592
## 42:	C47	0.00000000
## 43:	C48	2.39619079
## 44:	C49	10.78285857

## 45:	C50	2.76483553
## 46:	C51	0.00000000
## 47:	C52	0.00000000
## 48:	C53	0.00000000
## 49:	C54	0.00000000
## 50:	C55	0.00000000
## 51:	C56	0.00000000
## 52:	C57	0.00000000
## 53:	C58	0.00000000
## 54:	C60	0.92161184
## 55:	C61	2.94915790
## 56:	C62	22.11868425
## 57:	C63	0.64512829
## 58:	C64	12.25743752
## 59:	C65	0.27648355
## 60:	C66	0.00000000
## 61:	C67	8.38666778
## 62:	C68	0.18432237
## 63:	C69	1.10593421
## 64:	C70	0.55296711
## 65:	C71	8.75531252
## 66:	C72	1.10593421
## 67:	C73	5.06886514
## 68:	C74	0.73728948
## 69:	C75	0.27648355
## 70:	C76	2.67267435
## 71:	C80	2.39619079
## 72:	C81	3.40996382
## 73:	C82	2.58051316
## 74:	C83	3.96293093
## 75:	C84	0.92161184
## 76:	C85	4.42373685
## 77:	C88	0.00000000
## 78:	C90	2.21186843
## 79:	C91	3.31780264
## 80:	C92	4.51589803
## 81:	C93	0.00000000
## 82:	C94	0.18432237
## 83:	C95	0.09216118
## 84:	C96	1.56674013
## 85:	C97	0.00000000
## 86:	D00	0.46080592
## 87:	D01	0.18432237
## 88:	D02	0.09216118
## 89:	D03	4.60805922
## 90:	D04	1.75106250

606. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

```
## 91:      D05  0.00000000
## 92:      D06  0.00000000
## 93:      D07  0.36864474
## 94:      D09  0.27648355
## 95:      D30  1.38241777
## 96:      D33  6.26696054
##      ICDCode      Inc
```

Mi történt itt? Először is, a sor-szűrések közül kivettük a konkrét rák-típust – ez értelemszerű, hiszen az összes ráktípusra vonatkozó adatot szeretnénk kapni, épp ez volt a feladat, tehát ebben nyilván nem szűrhetjük le előzetesen az adatbázist. Másodsor, bekerült a harmadik pozíciója, csoportosító változóként a rák típusa. Mit jelent ez? Azt, hogy a szűrés után a `data.table` a leszűrt adatbázisból rák-típus szerint csoportokat képez, tehát szétszedi az adatbázist kis táblákra úgy, hogy mindegyikben egy adott rák-típus adatai legyenek, mindegyikre elvégzi a második pozícióban, az oszlopindexelésnél megadott műveleteket (jelen esetben: kiszámítja az incidenciákat), majd ezeket az eredményeket, ami itt most egyetlen sor lesz, újra összerakja egy nagy táblába, jelezve, hogy az adott eredmény melyik kódhoz tartozik.

Nagyon szájbarágós, de talán egyszer érdemes a dolgot megnézni lépésről-lépésre. A `data.table` elsőként leszűri a táblát a sorindex szerint:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi"]
```

```
## Key: <County, Sex, Age, Year>
##      County      Sex   Age   Year ICDCode      N Population
##      <char> <char> <num> <num> <char> <int>      <num>
##  1: Budapest Férfi   40   2000     C00        1      51602
##  2: Budapest Férfi   40   2000     C01        6      51602
##  3: Budapest Férfi   40   2000     C02        2      51602
##  4: Budapest Férfi   40   2000     C03        1      51602
##  5: Budapest Férfi   40   2000     C04        2      51602
##    ---
## 1820: Budapest Férfi   40   2018     D06        0      80555
## 1821: Budapest Férfi   40   2018     D07        0      80555
## 1822: Budapest Férfi   40   2018     D09        2      80555
## 1823: Budapest Férfi   40   2018     D30        0      80555
## 1824: Budapest Férfi   40   2018     D33        4      80555
```

Ezt követően megnézi, hogy a csoportosító változó milyen értékeket vesz fel:

```
unique(RawData[Age == 40 & County == "Budapest" &
  Sex == "Férfi"]$ICDCode)
```

```
## [1] "C00" "C01" "C02" "C03" "C04" "C05" "C06" "C07" "C08" "C09" "C10" "C11"
## [13] "C12" "C13" "C14" "C15" "C16" "C17" "C18" "C19" "C20" "C21" "C22" "C23"
## [25] "C24" "C25" "C26" "C30" "C31" "C32" "C33" "C34" "C37" "C38" "C39" "C40"
## [37] "C41" "C43" "C44" "C45" "C46" "C47" "C48" "C49" "C50" "C51" "C52" "C53"
```

```
## [49] "C54" "C55" "C56" "C57" "C58" "C60" "C61" "C62" "C63" "C64" "C65" "C66"
## [61] "C67" "C68" "C69" "C70" "C71" "C72" "C73" "C74" "C75" "C76" "C80" "C81"
## [73] "C82" "C83" "C84" "C85" "C88" "C90" "C91" "C92" "C93" "C94" "C95" "C96"
## [85] "C97" "D00" "D01" "D02" "D03" "D04" "D05" "D06" "D07" "D09" "D30" "D33"
```

Majd ezek mindegyikére leszűkíti a (szűrt) táblát, lényegében kis táblákat készítve.
Így néz ki a C00-hoz tartozó:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCCode == "C00"]
```

```
## Key: <County, Sex, Age, Year>
##      County    Sex   Age   Year ICDCCode      N Population
##      <char> <char> <num> <num> <char> <int>      <num>
##  1: Budapest Férfi   40  2000     C00        1   51602.0
##  2: Budapest Férfi   40  2001     C00        1   47836.0
##  3: Budapest Férfi   40  2002     C00        0   45296.5
##  4: Budapest Férfi   40  2003     C00        0   43632.5
##  5: Budapest Férfi   40  2004     C00        0   43085.0
##  6: Budapest Férfi   40  2005     C00        0   43442.5
##  7: Budapest Férfi   40  2006     C00        0   44511.5
##  8: Budapest Férfi   40  2007     C00        0   46903.5
##  9: Budapest Férfi   40  2008     C00        0   50505.5
## 10: Budapest Férfi   40  2009     C00        0   54015.0
## 11: Budapest Férfi   40  2010     C00        0   57445.5
## 12: Budapest Férfi   40  2011     C00        1   60721.0
## 13: Budapest Férfi   40  2012     C00        1   62471.5
## 14: Budapest Férfi   40  2013     C00        0   63746.5
## 15: Budapest Férfi   40  2014     C00        0   66250.5
## 16: Budapest Férfi   40  2015     C00        0   70511.5
## 17: Budapest Férfi   40  2016     C00        0   74622.0
## 18: Budapest Férfi   40  2017     C00        0   77902.0
## 19: Budapest Férfi   40  2018     C00        0   80555.0
```

Így a C01-hez:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
        ICDCCode == "C01"]
```

```
## Key: <County, Sex, Age, Year>
##      County    Sex   Age   Year ICDCCode      N Population
##      <char> <char> <num> <num> <char> <int>      <num>
##  1: Budapest Férfi   40  2000     C01        6   51602.0
##  2: Budapest Férfi   40  2001     C01        1   47836.0
##  3: Budapest Férfi   40  2002     C01        0   45296.5
##  4: Budapest Férfi   40  2003     C01        3   43632.5
##  5: Budapest Férfi   40  2004     C01        2   43085.0
##  6: Budapest Férfi   40  2005     C01        0   43442.5
```

626. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

```
## 7: Budapest Férfi 40 2006 C01 3 44511.5
## 8: Budapest Férfi 40 2007 C01 2 46903.5
## 9: Budapest Férfi 40 2008 C01 0 50505.5
## 10: Budapest Férfi 40 2009 C01 2 54015.0
## 11: Budapest Férfi 40 2010 C01 0 57445.5
## 12: Budapest Férfi 40 2011 C01 3 60721.0
## 13: Budapest Férfi 40 2012 C01 1 62471.5
## 14: Budapest Férfi 40 2013 C01 1 63746.5
## 15: Budapest Férfi 40 2014 C01 0 66250.5
## 16: Budapest Férfi 40 2015 C01 0 70511.5
## 17: Budapest Férfi 40 2016 C01 0 74622.0
## 18: Budapest Férfi 40 2017 C01 0 77902.0
## 19: Budapest Férfi 40 2018 C01 0 80555.0
```

És így tovább.

Ezt követően minden kis táblára elvégzi az oszlopindexelésnél kijelölt műveletet. Így fog kinézni az eredmény a C00-s kis táblára:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
  ICDCode == "C00",
  .(Inc = sum(N) / sum(Population) * 1e5)]
```

```
##          Inc
##          <num>
## 1: 0.3686447
```

Így a C01-es kis táblára:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi" &
  ICDCode == "C01",
  .(Inc = sum(N) / sum(Population) * 1e5)]
```

```
##          Inc
##          <num>
## 1: 2.211868
```

Majd ezeket a kis táblákat egymás alá rendezi, abban a sorrendben, ahogy az eredeti táblában előfordultak a csoportosító változó értékei, és úgy, hogy mindegyikhez melléírja, hogy az adottnál mi volt a csoportosító változó értéke, tehát jelen esetben, hogy melyik ráktípushoz tartozik.

Így kaptuk a fent látható táblát (menjünk vissza és ellenőrizzük)...!

Nézzünk meg – most már nagyon részletes levezetés nélkül – még egy példát csoportosításra. Kíváncsiak vagyunk egy adott ráktípus korszpecifikus incidenciájára, tehát, hogy mennyi az incidencia adott életkorban. Ha mindezt rögzített évre, nemre és megyére kérdezzük, akkor célt érhetünk így:

```
RawData[Year == 2010 & County == "Budapest" &
  Sex == "Férfi" & ICDCode == "C18",
  .(Age, Inc = N / Population * 1e5)]
```

```
##      Age      Inc
##      <num>    <num>
## 1:      0  0.000000
## 2:      5  0.000000
## 3:     10  0.000000
## 4:     15  0.000000
## 5:     20  1.960054
## 6:     25  3.072716
## 7:     30  2.321088
## 8:     35 11.088472
## 9:     40  5.222341
##10:     45 23.579344
##11:     50 37.503585
##12:     55 79.089038
##13:     60 130.617667
##14:     65 303.122158
##15:     70 292.659728
##16:     75 337.025636
##17:     80 456.403917
##18:     85 320.454129
```

A dolog azonban nagyon nem szerencsés: kizárólag azért fog működni, mert a leszűkítés után egy adott életkorhoz már csak egyetlen sor tartozik. De ha ez nem így lenne, például kitörlünk valamit a feltételek közül, akkor teljesen rossz eredményt fog adni, hiszen ilyenkor ugyanaz az életkor többször fog megjelenni az eredményben, míg nekünk össze kellene adnunk az adott életkorhoz tartozó különböző megfigyeléseket.

A megoldás a csoportosítás az életkor szerint, és az összeadás adott életkoron belül:

```
RawData[Year == 2010 & County == "Budapest" &
  Sex == "Férfi" & ICDCode == "C18",
  .(Inc = sum(N) / sum(Population) * 1e5), .(Age)]
```

```
##      Age      Inc
##      <num>    <num>
## 1:      0  0.000000
## 2:      5  0.000000
## 3:     10  0.000000
## 4:     15  0.000000
## 5:     20  1.960054
## 6:     25  3.072716
```

646. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

```
## 7: 30 2.321088
## 8: 35 11.088472
## 9: 40 5.222341
## 10: 45 23.579344
## 11: 50 37.503585
## 12: 55 79.089038
## 13: 60 130.617667
## 14: 65 303.122158
## 15: 70 292.659728
## 16: 75 337.025636
## 17: 80 456.403917
## 18: 85 320.454129
```

Ez immár működik másféle szűréssel is, például ha Budapest helyett az egész országra vagyunk kíváncsiak:

```
RawData[Year == 2010 & Sex == "Férfi" & ICDCode == "C18",
        .(Inc = sum(N) / sum(Population) * 1e5), .(Age)]
```

```
##      Age      Inc
##      <num>      <num>
## 1: 0 0.4013437
## 2: 5 0.0000000
## 3: 10 0.3909763
## 4: 15 0.3278087
## 5: 20 1.2121488
## 6: 25 2.2652654
## 7: 30 3.2732344
## 8: 35 7.1454000
## 9: 40 11.2810011
## 10: 45 21.7613797
## 11: 50 46.2794518
## 12: 55 94.8719665
## 13: 60 130.8683356
## 14: 65 246.5976935
## 15: 70 293.1432072
## 16: 75 367.2073711
## 17: 80 347.1158754
## 18: 85 323.4103513
```

Érdemes végiggondolni (ez általában is hasznos): ilyenkor az életkor szerinti kis táblákban 20 sor lesz – az egyes megyékkel – és ezek fölött fogunk összegezni.

Megjegyzendő, hogy a csoportosító változónak nevet is adhatunk:

```
RawData[Year == 2010 & Sex == "Férfi" & ICDCode == "C18",
        .(Inc = sum(N) / sum(Population) * 1e5),
        .(Eletkor = Age)]
```



```
##      Eletkor      Inc
##      <num>      <num>
## 1:      0    0.4013437
## 2:      5    0.0000000
## 3:     10    0.3909763
## 4:     15    0.3278087
## 5:     20    1.2121488
## 6:     25    2.2652654
## 7:     30    3.2732344
## 8:     35    7.1454000
## 9:     40   11.2810011
## 10:    45   21.7613797
## 11:    50   46.2794518
## 12:    55   94.8719665
## 13:    60  130.8683356
## 14:    65  246.5976935
## 15:    70  293.1432072
## 16:    75  367.2073711
## 17:    80  347.1158754
## 18:    85 323.4103513
```

Ami azonban sokkal izgalmasabb, hogy műveletet is végezhetünk! Itt is igaz, hogy nem kell a változót külön letárolni, hanem menet közben kiszámolhatjuk, majd építhetünk is rá rögtön (jelen esetben egy csoportosítást). Például, ha ki akarjuk számolni az incidenciát külön a 70 év alattiak és felettiek körében:

```
RawData[Year == 2010 & Sex == "Férfi" & ICDCCode == "C18",
        .(Inc = sum(N) / sum(Population) * 1e5),
        .(Idos = Age > 70)]
```

```
##      Idos      Inc
##      <lgcl>      <num>
## 1: FALSE  43.83737
## 2:  TRUE 352.54419
```

6.6 Indexelések láncolása egymás után

A `data.table` következő újítása, hogy megengedi egy már indexelt tábla (`RawData[...]`) *újabb* indexelését. Tehát használhatjuk a `RawData[...][...]` alakot, ahol a második indexelés pontosan ugyanúgy fog viselkedni, mint az első (ugyanúgy használhatunk sorindexelést, szűrést és rendezést, oszlop kiválasztást és -transzformációt, csoportosítást), *de* úgy, hogy az az első, már indexelt táblára vonatkozik! Lényegében mintha elmentettük volna a `RawData[...]`-t egy változóba, és utána a változót indexelnénk szokásos módon – csak itt nem kell semmit külön elmenteni. Az, hogy a második indexelés már az első indexelésben átalakított táblára vonatkozik, egy kritikusan fontos előny, amint

az rögtön világossá is fog válni.

Ha pontosak akarunk lenni, akkor ezt az egymás utáni többszöri indexelést igazából a hagyományos data frame is megengedi, tehát például a `RawDataDF[101:200,][5:15,]` egy teljesen szabályos hívás (és természetesen egyenértékű lesz azzal, hogy `RawDataDF[105:115,]`). A probléma az, hogy a használhatósága nagyon korlátozott, mert a második indexben, ha változóra hivatkozunk, az az *eredeti* adatkeret változója tud csak lenni, nem az első indexelésben már átttranszformálté! (Értelemszerűen, hiszen az nincs is elmentve, nincs is semmilyen külön neve, ahogy hivatkozhatnánk rá.) Ha csak a legegyszerűbb transzformációt, a sorok szűrését vesszük: a `RawDataDF[RawDataDF$Sex == "Férfi",][RawDataDF$Year == 2010,]` nem fog működni, ez onnan is kapásból látszik, hogy a `RawDataDF$Year == 2010` ugyanolyan hosszú, mint a `RawDataDF`, viszont a `RawDataDF[RawDataDF$Sex == "Férfi",]` már rövidebb, tehát ez így biztosan nem lehet jó, mert az adattáblát hosszabb vektorral próbáljuk indexelni, mint ahány sora van. Data frame használatával erre a problémára nincs megoldás, hiszen a `RawDataDF[RawDataDF$Sex == "Férfi",]` táblázat `Year` változójára nem tudunk sehogy sem hivatkozni a második indexelésben, hiszen az nincs elmentve, nincs is külön neve, amivel hivatkozhatnánk.

A data table esetében azonban, kihasználva, hogy a változóra hivatkozhatunk csak a nevével, a táblázat neve nélkül, erre nagyon egyszerű a megoldás: annyi a feladat, hogy a második indexben szereplő `Year` alatt a `data.table` azt értse, hogy az első indexelés *után* kapott táblázat `Year` nevű változója (ne azt, hogy az eredetié). És így is van megírva a `data.table`, ezért szerepelt korábban az a megfogalmazás, hogy a második index az első indexeléssel már transzformált táblára vonatkozik. Így aztán a következő hívás tökéletesen működik data table-lel:

```
RawData[Sex == "Férfi"][Year == 2010]
```

```
## Key: <County, Sex, Age, Year>
##           County    Sex   Age  Year ICDCCode      N Population
##           <char> <char> <num> <num> <char> <int>      <num>
##    1: Baranya megye Férfi    0  2010      C00        0      9430.0
##    2: Baranya megye Férfi    0  2010      C01        0      9430.0
##    3: Baranya megye Férfi    0  2010      C02        0      9430.0
##    4: Baranya megye Férfi    0  2010      C03        0      9430.0
##    5: Baranya megye Férfi    0  2010      C04        0      9430.0
##    ---
## 34556:  Zala megye Férfi   85  2010      D06        0      1447.5
## 34557:  Zala megye Férfi   85  2010      D07        0      1447.5
## 34558:  Zala megye Férfi   85  2010      D09        0      1447.5
## 34559:  Zala megye Férfi   85  2010      D30        0      1447.5
## 34560:  Zala megye Férfi   85  2010      D33        4      1447.5
```

Ez még nem a legátütőbb példa – bár sokszor az ilyenek is nagyon jól jönnek

– hiszen használhattunk volna egyszerűen `&` jelet és egyetlen indexelést. A dolog igazi erejét az adja, hogy – ismét csak abból fakadóan, hogy a második index már az elsőnek indexelt táblát látja, neki nem is számít, hogy az nem egy lementett tábla, hanem egy már átalakított – módunk van menet közben létrehozott változókra is hivatkozni! Például miután kiszámoltuk rák-típusonként az incidenciát, szeretnénk a táblázatot az incidenciák szerint növekvő sorba rakni. Íme a megoldás:

```
RawData[Age == 40 & County == "Budapest" & Sex == "Férfi",
        .(Inc = sum(N) / sum(Population) * 1e5),
        .(ICDCode)][order(Inc)]
```

```
##      ICDCode      Inc
##      <char>    <num>
##  1:      C47  0.00000000
##  2:      C51  0.00000000
##  3:      C52  0.00000000
##  4:      C53  0.00000000
##  5:      C54  0.00000000
##  6:      C55  0.00000000
##  7:      C56  0.00000000
##  8:      C57  0.00000000
##  9:      C58  0.00000000
## 10:      C66  0.00000000
## 11:      C88  0.00000000
## 12:      C93  0.00000000
## 13:      C97  0.00000000
## 14:      D05  0.00000000
## 15:      D06  0.00000000
## 16:      C33  0.09216118
## 17:      C95  0.09216118
## 18:      D02  0.09216118
## 19:      C68  0.18432237
## 20:      C94  0.18432237
## 21:      D01  0.18432237
## 22:      C65  0.27648355
## 23:      C75  0.27648355
## 24:      D09  0.27648355
## 25:      C00  0.36864474
## 26:      C39  0.36864474
## 27:      D07  0.36864474
## 28:      C12  0.46080592
## 29:      C26  0.46080592
## 30:      C37  0.46080592
## 31:      C46  0.46080592
## 32:      D00  0.46080592
```

686. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

## 33:	C06	0.55296711
## 34:	C21	0.55296711
## 35:	C70	0.55296711
## 36:	C30	0.64512829
## 37:	C45	0.64512829
## 38:	C63	0.64512829
## 39:	C05	0.73728948
## 40:	C08	0.73728948
## 41:	C74	0.73728948
## 42:	C14	0.82945066
## 43:	C07	0.92161184
## 44:	C60	0.92161184
## 45:	C84	0.92161184
## 46:	C03	1.01377303
## 47:	C23	1.01377303
## 48:	C40	1.01377303
## 49:	C31	1.10593421
## 50:	C38	1.10593421
## 51:	C69	1.10593421
## 52:	C72	1.10593421
## 53:	C11	1.19809540
## 54:	D30	1.38241777
## 55:	C24	1.47457895
## 56:	C96	1.56674013
## 57:	C09	1.65890132
## 58:	C17	1.75106250
## 59:	D04	1.75106250
## 60:	C01	2.21186843
## 61:	C90	2.21186843
## 62:	C48	2.39619079
## 63:	C80	2.39619079
## 64:	C04	2.48835198
## 65:	C02	2.58051316
## 66:	C19	2.58051316
## 67:	C82	2.58051316
## 68:	C76	2.67267435
## 69:	C50	2.76483553
## 70:	C41	2.94915790
## 71:	C61	2.94915790
## 72:	C10	3.13348027
## 73:	C91	3.31780264
## 74:	C81	3.40996382
## 75:	C22	3.87076974
## 76:	C83	3.96293093
## 77:	C15	4.14725330
## 78:	C13	4.42373685

```
## 79:      C85  4.42373685
## 80:      C92  4.51589803
## 81:      D03  4.60805922
## 82:      C73  5.06886514
## 83:      D33  6.26696054
## 84:      C16  6.72776646
## 85:      C20  7.09641120
## 86:      C25  7.37289475
## 87:      C32  7.46505593
## 88:      C67  8.38666778
## 89:      C71  8.75531252
## 90:      C49 10.78285857
## 91:      C18 11.15150331
## 92:      C64 12.25743752
## 93:      C43 17.41846385
## 94:      C62 22.11868425
## 95:      C34 30.78183558
## 96:      C44 43.13143429
##      ICDCode      Inc
```

Hiába nem is létezik `Inc` nevű változó az eredeti adattáblában, ez a hívás mégis tökéletesen fog működni! Megint csak: azért, mert a második index már az első indexeléssel átalakított táblát kapja meg, és azt látja, pontosan ugyanúgy, mintha az egy lementett tábla lenne.

6.7 Referencia szemantika

A `data.table` bevezet egy új megközelítést arra, hogy új változót definiáljunk egy táblában – ám hamar ki fog derülni, hogy itt jóval többről van szó, mint egyszerűen egy alternatív jelölésről.

Például számoljuk ki, és ezúttal a táblázatban is tároljuk el az incidenciákat¹³:

```
RawData$Inc <- RawData$N / RawData$Population * 1e5
```

A `data.table` által bevezett új megoldás esetén az értékadás jele a `:=`, de ami talán még fontosabb, hogy ezt, elsőre elég meglepő módon, úgy kell megadni, mintha indexelnénk, tehát szögletes zárójelek között! A második pozícióba, az oszlopindex helyébe kell kerülnön:

```
RawData[, Inc2 := N / Population * 1e5]
```

A kettő valóban ugyanazt eredményezi:

¹³Az összes fenti esetben ezt el tudtuk kerülni, és jobb is elkerülni: gondoljunk arra, hogy ha csoportosítást is csinálunk, akkor ezekkel az előre kiszámolt rétegenkénti incidenciákkal nem megyünk semmire. (Általában is igaz, hogy a kiszámítható dolgok közül csak azokat érdemes fizikailag letárolni az adatbázisban, amik kiszámítása sok időt venne igénybe.) Tehát ez most szigorúan csak illusztratív példa új változó létrehozására.

```
identical(RawData$Inc, RawData$Inc2)
```

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

Ebben van egy újdonság: az összes eddigi példában *új* táblát hoztunk létre (még ha csak ki is írtuk, és nem mentettük el változóba), ez az első eset, ahol *meglevő* táblát módosítunk. Ez nagyon fontos: mint láthatjuk is, nem kell az eredményt belementenünk egy változóba, azért nem, mert az utasítás lefuttatásakor *maga az eredeti tábla módosult!* Ezt szokták az informatikában referencia szerinti módosításnak¹⁴ hívni. (És igen, ezt az indexelés szintaktikájával éri el a `data.table`, még ha elég meglepő is első látásra.)

Kiírás ilyenkor ugyanúgy nincs, mint általában az értékadásos utasításoknál R-ben. Ha szeretnénk az értékadás után rögtön ki is írni a táblát akkor egy `[]` jelet kell tennünk a parancs után, pl. `RawData[, Inc2 := N / Population * 1e5][]`.

Használhatjuk ezt a megoldást meglevő változó felülírására, nem csak új létrehozására. Például, ha meggondoljuk magunkat, és az incidenciát per millió fő mértékegységben szeretnénk megadni:

```
RawData[, Inc2 := Inc2 * 10]
```

Egyszerre több változót is definiálhatunk (lehet vegyesen új definiálása és régi felülírása, ennek nincs jelentősége), ennek módszere:

```
RawData[, c("logPop", "sqrtPop") := list(log(Population),
                                           sqrt(Population))]
```

Mivel az értékadás bal oldalán sztring-vektor áll, így könnyen előállítható gépi úton is. A jobb oldalon pedig lista szerepel, így itt is igaz, hogy nem muszáj kézzel felsorolni, bármilyen olyan függvény szerepelhet ott, ami listát ad vissza.

Változó törölhető is ilyen módon:

```
RawData[, Inc2 := NULL]
```

Ha több változót törölnénk:

```
RawData[, c("logPop", "sqrtPop") := NULL]
```

Mi értelme van mindennek? Az első válasz az, hogy bizonyos esetekben gyorsabb¹⁵. A második, hogy mindez kombinálható a `data.table` többi elemével,

¹⁴Ez problémát jelenthet akkor, ha egy függvényen belül csinálunk ilyet, hiszen ez azt fogja maga után vonni, hogy a bemenetként átadott adattábla át fog alakulni. Ez esetben a `copy` függvény segíthet: ezzel készíthetünk első lépésben egy másolatot a tábláról, és ha utána azon dolgozunk, akkor az eredeti, bemenetként megkapott tábla nem fog átalakulni.

¹⁵Az R a 3.1.0-s verzió előtt minden ilyen változó-értékdadási műveletnél `deep copy`-t csinált az adatbázisról, ami azt jelenti, hogy nem csak a memóriamutatókat frissítette (ez lenne a `shallow copy`), hanem az egész adatbázist fizikailag átmásolta egy másik memóriaterületre. Ez nagyon gazdaságtalan, pláne, mert értelmetlen is, hiszen egy új változó definiálásától a meglevő tartalom maradhatna ugyanott. Ezt a 3.1.0-s verzióban orvosolták, de az továbbra is

tehát a sorindexeléssel és a csoportosítással.

Például szeretnénk a “Budapest” kifejezést lecserélni arra, hogy “Főváros” a megye változóban. Ezt megoldhatjuk így:

```
RawData[County == "Budapest", County := "Főváros"]
```

Tehát: ha az értékadást szűréssel kombináljuk, akkor a nem kiválasztott soroknál nem változik az érték. (Ha pedig nem meglevő változót módosítunk, hanem újat hozunk létre, akkor a nem kiválasztott soroknál NA kerül az új változóba.)

Ezt könnyen megoldhattuk volna másképp is, de nézzük egy izgalmasabb példát. Szeretnénk minden nemre, életkorra, megyére és ráktípusra eltárolni, hogy az adott nemből, életkorból, megyéből és ráktípusból mi volt a legkisebb feljegyzett incidencia (a különböző évek közül, tehát). Ezt `data.table` nélkül csak macerásabban tudnánk megtenni, de a `data.table` használatával nagyon egyszerű (és nagyon logikus) a megoldás:

```
RawData[, MinInc := min(Inc), .(County, Sex, Age, ICDCode)]
```

A csoportosító változót kell használnunk, ami teljesen logikus is: képezi a csoportokat nem, életkor, megye és ráktípus szerint (tehát az egyes csoportokban a különböző évek fognak szerepelni), veszi azok körében az `Inc` minimumát, és azt menti el `MinInc` néven – az adott csoport különböző soraihoz mindig ugyanazt az értéket. Íme:

```
RawData[ICDCode == "C18" & Age == 70 & County == "Főváros"]
```

##	County	Sex	Age	Year	ICDCode	N	Population	Inc	MinInc
##	<char>	<char>	<num>	<num>	<char>	<int>	<num>	<num>	<num>
##	1: Főváros	Férfi	70	2000	C18	97	30697.5	315.9866	217.0223
##	2: Főváros	Férfi	70	2001	C18	111	32326.5	343.3715	217.0223
##	3: Főváros	Férfi	70	2002	C18	108	31711.5	340.5705	217.0223
##	4: Főváros	Férfi	70	2003	C18	99	30984.0	319.5198	217.0223
##	5: Főváros	Férfi	70	2004	C18	100	30205.5	331.0655	217.0223
##	6: Főváros	Férfi	70	2005	C18	97	29194.5	332.2544	217.0223
##	7: Főváros	Férfi	70	2006	C18	90	28123.5	320.0171	217.0223
##	8: Főváros	Férfi	70	2007	C18	96	27422.5	350.0775	217.0223
##	9: Főváros	Férfi	70	2008	C18	95	27080.5	350.8059	217.0223
##	10: Főváros	Férfi	70	2009	C18	102	26957.0	378.3804	217.0223
##	11: Főváros	Férfi	70	2010	C18	80	27335.5	292.6597	217.0223
##	12: Főváros	Férfi	70	2011	C18	97	28288.0	342.9016	217.0223
##	13: Főváros	Férfi	70	2012	C18	89	30601.5	290.8354	217.0223
##	14: Főváros	Férfi	70	2013	C18	118	32451.0	363.6252	217.0223
##	15: Főváros	Férfi	70	2014	C18	108	34269.5	315.1490	217.0223
##	16: Főváros	Férfi	70	2015	C18	103	35428.0	290.7305	217.0223

megmaradt, hogy nem az egész oszlop kap értéket, csak egy része, akkor deep copy készül. Ezzel szemben a `data.table` minden esetben és minden verzióban shallow copy-t csinál értékadásnál.

726. FEJEZET. DATA TABLE: EGY TOVÁBBFEJLESZTETT ADATKERET

##	17: Főváros	Férfi	70	2016	C18	107	35831.5	298.6199	217.0223
##	18: Főváros	Férfi	70	2017	C18	101	36012.0	280.4621	217.0223
##	19: Főváros	Férfi	70	2018	C18	78	35941.0	217.0223	217.0223
##	20: Főváros	Nő	70	2000	C18	115	52434.0	219.3233	161.1007
##	21: Főváros	Nő	70	2001	C18	99	53138.5	186.3056	161.1007
##	22: Főváros	Nő	70	2002	C18	95	51751.0	183.5713	161.1007
##	23: Főváros	Nő	70	2003	C18	110	50498.5	217.8283	161.1007
##	24: Főváros	Nő	70	2004	C18	92	49073.0	187.4758	161.1007
##	25: Főváros	Nő	70	2005	C18	102	47308.5	215.6061	161.1007
##	26: Főváros	Nő	70	2006	C18	74	45934.0	161.1007	161.1007
##	27: Főváros	Nő	70	2007	C18	95	45165.0	210.3399	161.1007
##	28: Főváros	Nő	70	2008	C18	94	44594.5	210.7883	161.1007
##	29: Főváros	Nő	70	2009	C18	84	44558.5	188.5162	161.1007
##	30: Főváros	Nő	70	2010	C18	81	45258.5	178.9719	161.1007
##	31: Főváros	Nő	70	2011	C18	101	46479.0	217.3024	161.1007
##	32: Főváros	Nő	70	2012	C18	104	48132.0	216.0725	161.1007
##	33: Főváros	Nő	70	2013	C18	134	50451.0	265.6042	161.1007
##	34: Főváros	Nő	70	2014	C18	124	52854.0	234.6085	161.1007
##	35: Főváros	Nő	70	2015	C18	94	54534.5	172.3680	161.1007
##	36: Főváros	Nő	70	2016	C18	120	55317.5	216.9295	161.1007
##	37: Főváros	Nő	70	2017	C18	95	55986.5	169.6838	161.1007
##	38: Főváros	Nő	70	2018	C18	113	56508.0	199.9717	161.1007
##	County	Sex	Age	Year	ICDCode	N	Population	Inc	MinInc