Data management in R

Zoltan Kekecs, Marton Kovacs

February 18, 2020

Contents

1 Adatmenedzsment **2**
1.1 Absztrakt . . . 2
1.2 Ismétlés . . . 2

2 Alapvető parancsok az adatok megismeréséhez **3**
2.1 Beépített adattáblák . . . 3
2.2 Nyers adatok és meta-adatok megtekintése . . . 3
2.3 Alapvető parancsok az adattábla struktúrájának megértéséhez . . . 3
2.4 Hivatkozás objektumok rész-elemeire (Subsetting) . . . 4

3 Combine függvény **6**

4 Tidyverse **7**
4.1 A négy dplyr alapfunkció . . . 8
4.2 Egyéb hasznos dplyr funkciók . . . 10
4.3 Változók újrakódolása . . . 17
4.4 A mutate() függvény változatai (ajánlott anyag) . . . 20

1 Adatmenedzsment

1.1 Absztrakt

Ezen a gyakorlaton megtanuljuk az adatkezelés alapjait az R programban. A gyakorlat bemutatja hogyan lehet adattáblát létrehozni és külső fájlból beolvasni, hogyan lehet az adattábla egyes részeire (bizonyos soraira vagy oszlopaira) hivatkozni (subsetting), és azt, hogy hogyan tudjuk az adatokat formázni és módosítani.

1.2 Ismétlés

Az előző órán tanultuk hogy az R többféle adatosztályt és típust különböztet meg.

Vektorok:

- karakter vektor (character): "Ez egy karakter érték"
- faktor (factor): Egy olyan karater vektor ami csak rögzített értékeket vehet fel
- szám vektor (numeric): 2 vagy 13.5. A szám vektornak két típusa van: egész szám vektor (integer): 2L (Az L mondja meg az R-nek, hogy az - előtte lévő számot, mint egész számot kezelje), vagy racionális szám (double): pl.: 13.5
- logikai vektor (logical): csak TRUE vagy FALSE értékeket vehet fel (nagybetű is számít)
- complex szám vektor (complex): 1+5i

Komplexebb adatstruktúrák:

- mátrix (matrix): a vektorok egy táblázatot alkotnak. A mátrixban minden vektor/adat típusa csak ugyan az lehet.
- adattábla (data.frame): egy olyan mátrix, aminben megengedett hogy az oszlopokban egymáshoz képest más adat típus szerepeljen
- lista (list): egy olyan vektor, aminek az elemei lehetnek más adatstruktúrák.

Az adat osztályát a class() függvénnyel tudjuk ellenőrizni, az adott osztályon belüli típusát pedig a typeof() függvénnyel.

De nem csak általánosságban tudjuk megnézni hogy milyen típusú változó egy objektum. Ha szeretnénk tudni hogy egy változó egy adott típusba tartozik-e, as is.változótípus függvénnyel erre rákérdezhetünk. Így például az is.vector() kód rákérdez hogy az adott objektum vektor-e, és az is.integer() kód rákérdez hogy a változó integer típusú-e.

```
number <- c(3, 4)

class(number)
```

```
## [1] "numeric"
```

```
typeof(number)
```

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

```
is.numeric(number)
```

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

```
is.integer(number)
```

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

```
is.vector(number)
```

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

Gyakorlás

Ellenőrizd hogy mi az is.character(number) kód **eredménye** milyen osztályba tartozik.

2 Alapvető parancsok az adatok megismeréséhez

2.1 Beípített adattáblák

Az R-ben vannak előre beépített adattáblák. Ezekkel jól lehet gyakorolni az adatkezelést.

Most az egyik ilyen beépített adatbázissal fogunk majd dolgozni, a USArrests adatbázissal, ami különböző bűntényekkel kapcsolatos letartóztatások statisztikáit tartalmazza az Egyesült Államok különböző államaira lebontva.

A ?USArrests lefuttatásával további információkat kaphatsz az adatokról.

2.2 Nyers adatok és meta-adatok megtekintése

A következő parancsok futtatásával megnézhetjük a nyers adatokat táblázatos formában és meta-adatokat kaphatunk az adatbázisról.

```
View(USArrests)

USArrests

?USArrests
```

2.3 Alapvető parancsok az adattábla struktúrájának megértéséhez

Gyakran megesik hogy nem akarjuk az egész adattáblát megnézni, hanem csak annak valamelyik részéről szeretnénk információt kapni, vagy ezt az adatot valahogy felhasználni. Erre számos hasznos funkció áll rendelkezésünkre.

- **str()**: információt ad arról hogy milyen olytályba tartozik az adott objektum, hány sora és hány oszlopa van, és az egyes oszlopokban milyen típusú adatokat tartalmaznak, és egy kis mintát is kapunk az adatokból.
- **names()** : oszlopok neveit listázza ki (column names, headers)
- **row.names()**: sorok/megfigyelések neveit adja meg
- **nrow()**: sorok száma
- **ncol()**: oszlopok száma
- **head()**: kilistázza az adattábla első x sorát (alapértelmezett 5 sor, de változtathatjuk)
- **tail()**: kilistázza az adattábla utolsó x sorát (alapértelmezett 5 sor, de változtathatjuk)

```
str(USArrests)
```

```
## 'data.frame':    50 obs. of  4 variables:
##  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
##  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
##  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
##  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
```

```
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

```
row.names(USArrests)
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
```

```
## [13] "Illinois"       "Indiana"        "Iowa"           "Kansas"
## [17] "Kentucky"       "Louisiana"      "Maine"          "Maryland"
## [21] "Massachusetts"  "Michigan"       "Minnesota"      "Mississippi"
## [25] "Missouri"       "Montana"        "Nebraska"       "Nevada"
## [29] "New Hampshire"  "New Jersey"     "New Mexico"     "New York"
## [33] "North Carolina" "North Dakota"   "Ohio"           "Oklahoma"
## [37] "Oregon"         "Pennsylvania"   "Rhode Island"   "South Carolina"
## [41] "South Dakota"   "Tennessee"      "Texas"          "Utah"
## [45] "Vermont"        "Virginia"       "Washington"     "West Virginia"
## [49] "Wisconsin"      "Wyoming"
```

```
nrow(USArrests)
```

```
## [1] 50
```

```
ncol(USArrests)
```

```
## [1] 4
```

```
head(USArrests)

tail(USArrests, 10)
```

2.4 Hivatkozás objektumok rész-elemeire (Subsetting)

Gyakran előfordul hogy az objektumoknak csak egy részét szeretnénk használni. Mondjuk tegyük fel hogy szeretnénk a USArrests adatbázisban azt megvizsgálni hogy az USÁban átlagosan hány embert tartóztattak le gyilkosságért 1973-ban. A mean() funkcióval tudjuk az átlagot meghatározni, de ennek csak numerikus vektor lehet a bemenete, adattábla objektumot nem adhatunk meg ebben a függvényben. Vagyis itt csak az adatbázisnak a "Murder" oszlopára van szükségünk.

Ennek a megoldására két lehetőségünk is van a base R-ben. Az egyik a $ jel használata. Ez viszonylag tiszta kódot eredményez, de kevésbé felxibilis mint a paraméterezés (lásd alább).

```
USArrests$Murder
```

```
##  [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2  2.2
## [16]  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2  2.1  7.4
## [31] 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2 12.7  3.2  2.2
## [46]  8.5  4.0  5.7  2.6  6.8
```

```
class(USArrests$Murder)
```

```
## [1] "numeric"
```

```
is.vector(USArrests$Murder)
```

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

```
mean(USArrests$Murder)
```

```
## [1] 7.788
```

A másik pedig lehetőség a paraméterezés. Ebben a megoldásban az objektum neve után egy szögletes zárójelet rakunk, és azon belül határozzuk meg, az objektum melyik részét szeretnénk megtartani, vagy éppen elvetni.

```
USArrests[,"Murder"]
```

```
##  [1] 13.2 10.0  8.1  8.8  9.0  7.9  3.3  5.9 15.4 17.4  5.3  2.6 10.4  7.2  2.2
## [16]  6.0  9.7 15.4  2.1 11.3  4.4 12.1  2.7 16.1  9.0  6.0  4.3 12.2  2.1  7.4
## [31] 11.4 11.1 13.0  0.8  7.3  6.6  4.9  6.3  3.4 14.4  3.8 13.2 12.7  3.2  2.2
```

```
## [46]  8.5  4.0  5.7  2.6  6.8
```

```
class(USArrests[,"Murder"])
```

```
## [1] "numeric"
```

```
is.vector(USArrests[,"Murder"])
```

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

```
mean(USArrests[,"Murder"])
```

```
## [1] 7.788
```

Itt fontos, hogy ha ha többdimenziós objektumról van szó (mint például egy adattábla, data.frame), pontosan meg tudjuk jelölni, melyik dimenzió mentén szeretnénk az adott hivatkozást használni. Az általános szabály az, hogy a szögletes zárójelben egy vesszővel elválasztva először a sorokra vonatkozó kiválasztási szabályt adjuk meg, majd a vessző után az oszlopokra vonatkozó kiválasztási szabályt.

Így például a USArrests[2, "Murder"] azt jelenti, hogy a USArrests adattábla 2. sorában a "Murder" nevű oszlopban szereplő adatot szeretnénk használni.

Néhány további példa segíthet a megértésben:

```
USArrests[1:3, "Assault"]
```

```
## [1] 236 263 294
```

```
USArrests[c("Illinois", "Arkansas"), "UrbanPop"]
```

```
## [1] 83 50
```

```
USArrests[c("Illinois", "Arkansas"), 2:4]
```

```
##          Assault UrbanPop Rape
## Illinois     249       83 24.0
## Arkansas     190       50 19.5
```

A paraméteres subsetting arra is lehetőséget ad, hogy kizárjunk bizonyos elemeket, amikre nincs szükségünk. Ezt általában a minusz jellel tehetjük meg. Sajnos itt a nevek használata nem olyan egyszerű mint a kiválasztásnál, így ilyenkor általában a sorok vagy oszlopok számaival dolgozunk.

Ez a függvény például kihagyja a negyediktől az ötvenedik sorig a sorokat (vagyis csak az első három sort hagyja meg),

```
USArrests[-c(4:50),]
```

```
##         Murder Assault UrbanPop Rape
## Alabama   13.2     236       58 21.2
## Alaska    10.0     263       48 44.5
## Arizona    8.1     294       80 31.0
```

Ez a parancs pedig ezen felül még kihagyja a második oszlopot is (az Assault oszlopot). Az alábbi két parancs ugyan azt éri el:

```
USArrests[-c(4:50),-2]
```

```
##         Murder UrbanPop Rape
## Alabama   13.2       58 21.2
## Alaska    10.0       48 44.5
## Arizona    8.1       80 31.0
```

```
USArrests[-c(4:50),-which(names(USArrests) == "Assault")]
```

```
##          Murder UrbanPop Rape
## Alabama    13.2       58 21.2
## Alaska     10.0       48 44.5
## Arizona     8.1       80 31.0
```

3 Combine függvény

A fenti példákban erősen hagyatkozunk a c() függvényre. Ez azt jelenti, "combine", és azok az elemek amit a c()-n belül vannak, egy vektorba rendezüdnek.

```
my_vector <- c(1, 3, 2, -3, 3)
my_vector
```

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

Emlékezz vissza hogy a vektorokkal kapcsolatban az a megkötés, hogy csak ugyan olyan típusú adatok lehetnek a vektor elemei! Amikor megpróbálunk olyan elemeket kombinálni amik más típusúak, akkor az R kiválasztja az egyik típust, és olyan típusúként használja az összes elemet az új vektorban. Szám típusú adatok és karakter típusú adatok összekombinálásával karakter típusú elemeket kapunk.

A típusváltás akár az adott elem átalakításához is vezethet. Például logikai adatokat számadatokkal kombinálva egy számvektort kapunk, és ilyenkor a TRUE-ból 1, a FALSE-ból 0 lesz.

```
my_other_vector <- c(1, 3, "two", "three")
my_other_vector
```

```
## [1] "1"     "3"     "two"   "three"
```

```
class(my_other_vector)
```

```
## [1] "character"
```

```
my_new_vector <- c(1, 3, TRUE)
my_new_vector
```

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

```
class(my_new_vector)
```

```
## [1] "numeric"
```

Természetesen vektor objektumokat is kombinálhatunk a c() függvénnyel (a fenti megkötés figyelembevételéve).

```
my_final_vector <- c(my_vector, my_new_vector)
my_final_vector
```

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

Gyakorlás

- Mentsd el a **USArrests** adattábla sorainak neveit egy új objektumként aminek az a neve hogy **row_names**
- egy fügvénnyel nézd meg hány elemből áll ez a **row_names** objektum
- egy fügvénnyel nézd meg milyen ennek az objektumnak az osztálya
- Csinálj egy új objektumot, ami egy olyan adattáblát tartalmaz, ami a **USArrests** objektumnak csak az "UrbanPop" és "Rape" oszlopait tartalmazza (legyen az objektum neve **USArrests_UrbanPop_Rape**).
- Listázd ki az **USArrests_UrbanPop_Rape** adattáblának az utolsó 8 sorát
- Nézd meg hogy populáció hány százaléka lakik városokban "Colorado" és "Mississippi" államokban?
- csinálj egy számvektort aminek elemei: 1.3, 4, 25, 2.13

4 Tidyverse

Az adatmenedzsmentet az R közösség gyakran "Data wrangling"-nek nevezi. Ezt a base R funciókkal is meg lehet csinálni, de általában egy egy nagyobb adatelemzési projekt végére ez nehezen átlátható kódot eredményez. A tiszta és átlátható kódolás elősegítésére a **Tidyverse** package gyűjtemény használhatjuk. Ez olyan R package-ek gyűjteménye, melyek mind egy standardizált és átlátható kód-írási rendszert támogatnak. Az egésznek az alapja a tiszta adatmenedzsmentet elősegítő *dplyr pakcage*.

Először töltsük be a Tidyverse package-et a library() funkcióval. (Ha a package még nincs felinstallálva, akkor ekőször fel kell installálni az install.packages() funkcióval, lásd az előző óra anyagát.)

```
library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------- tidyverse 1
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ---------------------------------------------------------------------- tidyverse_conflic
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Most hozzunk létre egy vektort néhány számmal.

```
x <- c(55:120, 984, 552, 17, 650)
```

Az egyik legfontosabb eleme a Tidy kódolásnak a %>% (pipe operátor) használata. A pipe operátort arra találták ki hogy függvények sorát egymás után egyszerűen le lehessen futtatni egy rövid átlátható kód segítségével.

Például ha a fenti vektor átlagának 10-es alapú logaritmusát akarjuk kiszámolni, majd az eredményt egy tizedesjegy pontossággal megadni, használhatjuk a round(log(mean(x)), digits = 1) függvény-sorozatot, de a sok egymásba foglalt zárójel miatt a végeredmény nehezen átlátható. Ehelyett alább látható hogy a %>% segítségével ugyanezt hogyan lehet megoldani.

A pipe-ot talán elképzelhetjük úgy mint egy csővezetéket vagy futószalagot, ami egy gyár különböző munkaállomásain vezeti végig a terméket. Az adat a termék, a függvények pedig a munkaállomások, és a pipe azt jelenti, hogy a pipe utáni függvény bemenete a pipe előtti eredmény legyen. Így a kódon tisztán látható, hogy az x vektor volt a kiindulópontunk, és hogy azon sorrendben milyen funkciókat hajtottunk végre hogy megkapjuk a végeredményt.

Ezt függvények láncba kapcsolásának, vagy chaining-nek is hívják.

```
round(log(mean(x)), digits = 1)
```

```
## [1] 4.7
```

```
x %>%
  mean() %>%
    log() %>%
      round(digits = 1)
```

```
## [1] 4.7
```

Nem minden funkció alkalmazható Tidyverse-ben, de a legtöbb funkciónak van egy Tidyverse kompatibilis változata valamelyik pacage-ben.

Például a sima összeadás és kivonás függvények nem használhatóak önállóan a %>% után. (Ez általában akkor jelent problémát amikor a matematikai művelettel kezdődik a sor. Amint a matematikai művelet valamilyen funció után következik akkor már általában lefut.)

```
x %>%
  mean() %>%
    log() %>%
      round(digits = 1) %>%
        -3 %>%
          +5 %>%
            /2
```

A magrittr package-ben található subtract(), add(), divide_by() stb. függvények lehetőséget adnak az ilyen alapvető matematikai műveletek Tidy kódban való megírására:

```
library(magrittr)
```

```
##
## Attaching package: 'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
x %>%
  mean() %>%
    log() %>%
      round(digits = 1) %>%
        subtract(3) %>%
          add(5) %>%
            divide_by(2)
```

```
## [1] 3.35
```

Most a ToothGrowth nevű beépített adattáblát fogjuk használni, hogy megtanuljuk a függvények használatát.

Gyakorlás

- Használd a korábban tanult függvényeket hogy megismerd a ToothGrowth adatbázist (Egyelőre nem szükséges a %>% használata, de használhatod ha akarod).
- Hány sor és hány oszlop van az adattáblában?
- Mi az objektum osztálya?
- Milyen típusú adatok szerepelnek az egyes oszlopokban?
- Mi az átlaga a "len" változónak?

TIPP: A %>% operátort a Ctrl + Shift + M gombok megnyomásával gyorsan beírhatod, ha nem akarod a karaktereket egyenként begépelni.

4.1 A négy dplyr alapfunkció

A dplyr pakcage-ben 4 alapvető funció van, amit mindenképpen ismerni kell:

- **filter()**: ezzel választjuk ki melyik megfigyelésekkel (sorokkal) szeretnénk dolgozni
- **mutate()**: módosíthatunk meglévő adatokat, vagy létrehozhatunk új adatokat az adattáblában

- **group_by()**: valamilyen szempont szerint tudjuk csoportosítani az adataink. *FONTOS: ezután az R a csoportokonként külön végzi el a láncban később jövő függvényeket.*
- **summarise()**: összesíti az adatokat valamilyen másik függvény szerint

Példák az alkalmazásra:

filter(): Válasszunk ki azokat az eseteket, ahol narancslével adták be a C-vitamint:

```
ToothGrowth %>%
  filter(supp == "OJ")
```

```
##     len supp dose
## 1  15.2   OJ  0.5
## 2  21.5   OJ  0.5
## 3  17.6   OJ  0.5
## 4   9.7   OJ  0.5
## 5  14.5   OJ  0.5
## 6  10.0   OJ  0.5
## 7   8.2   OJ  0.5
## 8   9.4   OJ  0.5
## 9  16.5   OJ  0.5
## 10  9.7   OJ  0.5
## 11 19.7   OJ  1.0
## 12 23.3   OJ  1.0
## 13 23.6   OJ  1.0
## 14 26.4   OJ  1.0
## 15 20.0   OJ  1.0
## 16 25.2   OJ  1.0
## 17 25.8   OJ  1.0
## 18 21.2   OJ  1.0
## 19 14.5   OJ  1.0
## 20 27.3   OJ  1.0
## 21 25.5   OJ  2.0
## 22 26.4   OJ  2.0
## 23 22.4   OJ  2.0
## 24 24.5   OJ  2.0
## 25 24.8   OJ  2.0
## 26 30.9   OJ  2.0
## 27 26.4   OJ  2.0
## 28 27.3   OJ  2.0
## 29 29.4   OJ  2.0
## 30 23.0   OJ  2.0
```

mutate(): Hozzunk létre egy új oszlopot, ami nem mm-ben, hanem cm-ben tárolja a fogak hosszát

FONTOS: a módosítások csak akkor mentődnek ha az eredményt egy új objektumhoz rendeljük!

```
my_ToothGrowth <- ToothGrowth %>%
  mutate(len_cm = len / 10)
```

summarise(): Most nézzük meg, hogy mennyi a fogak átlag hossza cm-ben.

```
my_ToothGrowth %>%
  summarise(mean_len_cm = mean(len_cm))
```

```
##   mean_len_cm
## 1    1.881333
```

A summarise() függvényben az **n()** függvényt használva meg tudjuk számolni azt is, hogy hány eset van.

Mivel ez még mindig egy összefoglaló statisztika, a betehetjük ezt is a summarise() függvénybe egy vesszővel elválasztva a mean()-től. Az eredmény egy táblázat lesz

```
my_ToothGrowth %>%
  summarise(mean_len_cm = mean(len_cm),
            n_cases = n())
```

```
##   mean_len_cm n_cases
## 1    1.881333      60
```

A **group_by()** függvényt használva az R a csoportokon belül végzi el a summarise()-ban előírt függvényeket.

```
ToothGrowth %>%
  mutate(len_cm = len / 10) %>%
    group_by(supp) %>%
      summarise(mean_len_cm = mean(len_cm),
                cases = n())
```

```
## # A tibble: 2 x 3
##   supp  mean_len_cm cases
##   <fct>       <dbl> <int>
## 1 OJ           2.07    30
## 2 VC           1.70    30
```

4.2 Egyéb hasznos dplyr funkciók

select(): Változók kiválasztása

Kiválaszhatunk bizonyos változókat, ha csak azokat szeretnénk megtartani. A mínusz jellel pedig törölhetünk egy adott változót. Választhatunk pozíció alapján is változót, de ez kevésbé ajánlott mert nehezebben érthető kódot eredményez, és később a bemeneti adat változásaival hibákhoz is vezethet. A select() függvénynek vannak segítő függvényei is, amivel szöveg részletek alapján tudunk választani több változót.

```
ToothGrowth %>%
  select(supp, len) %>%
  summary()
```

```
##  supp         len
##  OJ:30   Min.   : 4.20
##  VC:30   1st Qu.:13.07
##          Median :19.25
##          Mean   :18.81
##          3rd Qu.:25.27
##          Max.   :33.90
```

```
ToothGrowth %>%
  select(-dose)
```

```
##     len supp
## 1   4.2   VC
## 2  11.5   VC
## 3   7.3   VC
## 4   5.8   VC
## 5   6.4   VC
## 6  10.0   VC
## 7  11.2   VC
## 8  11.2   VC
## 9   5.2   VC
```

```
## 10  7.0   VC
## 11 16.5   VC
## 12 16.5   VC
## 13 15.2   VC
## 14 17.3   VC
## 15 22.5   VC
## 16 17.3   VC
## 17 13.6   VC
## 18 14.5   VC
## 19 18.8   VC
## 20 15.5   VC
## 21 23.6   VC
## 22 18.5   VC
## 23 33.9   VC
## 24 25.5   VC
## 25 26.4   VC
## 26 32.5   VC
## 27 26.7   VC
## 28 21.5   VC
## 29 23.3   VC
## 30 29.5   VC
## 31 15.2   OJ
## 32 21.5   OJ
## 33 17.6   OJ
## 34  9.7   OJ
## 35 14.5   OJ
## 36 10.0   OJ
## 37  8.2   OJ
## 38  9.4   OJ
## 39 16.5   OJ
## 40  9.7   OJ
## 41 19.7   OJ
## 42 23.3   OJ
## 43 23.6   OJ
## 44 26.4   OJ
## 45 20.0   OJ
## 46 25.2   OJ
## 47 25.8   OJ
## 48 21.2   OJ
## 49 14.5   OJ
## 50 27.3   OJ
## 51 25.5   OJ
## 52 26.4   OJ
## 53 22.4   OJ
## 54 24.5   OJ
## 55 24.8   OJ
## 56 30.9   OJ
## 57 26.4   OJ
## 58 27.3   OJ
## 59 29.4   OJ
## 60 23.0   OJ
```

```
ToothGrowth %>%
  select(1, 2)
```

```
##     len supp
## 1   4.2   VC
## 2  11.5   VC
## 3   7.3   VC
## 4   5.8   VC
## 5   6.4   VC
## 6  10.0   VC
## 7  11.2   VC
## 8  11.2   VC
## 9   5.2   VC
## 10  7.0   VC
## 11 16.5   VC
## 12 16.5   VC
## 13 15.2   VC
## 14 17.3   VC
## 15 22.5   VC
## 16 17.3   VC
## 17 13.6   VC
## 18 14.5   VC
## 19 18.8   VC
## 20 15.5   VC
## 21 23.6   VC
## 22 18.5   VC
## 23 33.9   VC
## 24 25.5   VC
## 25 26.4   VC
## 26 32.5   VC
## 27 26.7   VC
## 28 21.5   VC
## 29 23.3   VC
## 30 29.5   VC
## 31 15.2   OJ
## 32 21.5   OJ
## 33 17.6   OJ
## 34  9.7   OJ
## 35 14.5   OJ
## 36 10.0   OJ
## 37  8.2   OJ
## 38  9.4   OJ
## 39 16.5   OJ
## 40  9.7   OJ
## 41 19.7   OJ
## 42 23.3   OJ
## 43 23.6   OJ
## 44 26.4   OJ
## 45 20.0   OJ
## 46 25.2   OJ
## 47 25.8   OJ
## 48 21.2   OJ
## 49 14.5   OJ
## 50 27.3   OJ
## 51 25.5   OJ
## 52 26.4   OJ
## 53 22.4   OJ
```

```
## 54 24.5   OJ
## 55 24.8   OJ
## 56 30.9   OJ
## 57 26.4   OJ
## 58 27.3   OJ
## 59 29.4   OJ
## 60 23.0   OJ
```

```
ToothGrowth %>%
  select(2:3)
```

```
##    supp dose
## 1    VC  0.5
## 2    VC  0.5
## 3    VC  0.5
## 4    VC  0.5
## 5    VC  0.5
## 6    VC  0.5
## 7    VC  0.5
## 8    VC  0.5
## 9    VC  0.5
## 10   VC  0.5
## 11   VC  1.0
## 12   VC  1.0
## 13   VC  1.0
## 14   VC  1.0
## 15   VC  1.0
## 16   VC  1.0
## 17   VC  1.0
## 18   VC  1.0
## 19   VC  1.0
## 20   VC  1.0
## 21   VC  2.0
## 22   VC  2.0
## 23   VC  2.0
## 24   VC  2.0
## 25   VC  2.0
## 26   VC  2.0
## 27   VC  2.0
## 28   VC  2.0
## 29   VC  2.0
## 30   VC  2.0
## 31   OJ  0.5
## 32   OJ  0.5
## 33   OJ  0.5
## 34   OJ  0.5
## 35   OJ  0.5
## 36   OJ  0.5
## 37   OJ  0.5
## 38   OJ  0.5
## 39   OJ  0.5
## 40   OJ  0.5
## 41   OJ  1.0
## 42   OJ  1.0
## 43   OJ  1.0
```

```
## 44   OJ  1.0
## 45   OJ  1.0
## 46   OJ  1.0
## 47   OJ  1.0
## 48   OJ  1.0
## 49   OJ  1.0
## 50   OJ  1.0
## 51   OJ  2.0
## 52   OJ  2.0
## 53   OJ  2.0
## 54   OJ  2.0
## 55   OJ  2.0
## 56   OJ  2.0
## 57   OJ  2.0
## 58   OJ  2.0
## 59   OJ  2.0
## 60   OJ  2.0
```

```
ToothGrowth %>%
  select(starts_with("d", ignore.case = TRUE))
```

```
##    dose
## 1   0.5
## 2   0.5
## 3   0.5
## 4   0.5
## 5   0.5
## 6   0.5
## 7   0.5
## 8   0.5
## 9   0.5
## 10  0.5
## 11  1.0
## 12  1.0
## 13  1.0
## 14  1.0
## 15  1.0
## 16  1.0
## 17  1.0
## 18  1.0
## 19  1.0
## 20  1.0
## 21  2.0
## 22  2.0
## 23  2.0
## 24  2.0
## 25  2.0
## 26  2.0
## 27  2.0
## 28  2.0
## 29  2.0
## 30  2.0
## 31  0.5
## 32  0.5
## 33  0.5
```

```
## 34  0.5
## 35  0.5
## 36  0.5
## 37  0.5
## 38  0.5
## 39  0.5
## 40  0.5
## 41  1.0
## 42  1.0
## 43  1.0
## 44  1.0
## 45  1.0
## 46  1.0
## 47  1.0
## 48  1.0
## 49  1.0
## 50  1.0
## 51  2.0
## 52  2.0
## 53  2.0
## 54  2.0
## 55  2.0
## 56  2.0
## 57  2.0
## 58  2.0
## 59  2.0
## 60  2.0
```

arrange: Értékek sorba rendezése bizonyos változók alapján

Egy bizonyos változó értékei mentén sorba is rendezhetjük az adatokat.

```
ToothGrowth %>%
  mutate(len_cm = len / 10) %>%
    group_by(supp) %>%
      summarise(mean_len_cm = mean(len_cm),
                cases = n()) %>%
        arrange(mean_len_cm)
```

```
## # A tibble: 2 x 3
##   supp  mean_len_cm cases
##   <fct>       <dbl> <int>
## 1 VC           1.70    30
## 2 OJ           2.07    30
```

Ha a mínusz jelet a változó elé rakjuk, akkor csökkenő sorrendbe rakja az értékeket, növekvő helyet.

```
ToothGrowth %>%
  mutate(len_cm = len / 10) %>%
    group_by(supp) %>%
      summarise(mean_len_cm = mean(len_cm),
                cases = n()) %>%
        arrange(-mean_len_cm)
```

```
## # A tibble: 2 x 3
##   supp  mean_len_cm cases
##   <fct>       <dbl> <int>
```

```
## 1 OJ           2.07    30
## 2 VC           1.70    30
```

rename(): A átnevezhetünk változókat

```
ToothGrowth %>%
  rename(new_name = dose)
```

```
##     len supp new_name
## 1   4.2   VC      0.5
## 2  11.5   VC      0.5
## 3   7.3   VC      0.5
## 4   5.8   VC      0.5
## 5   6.4   VC      0.5
## 6  10.0   VC      0.5
## 7  11.2   VC      0.5
## 8  11.2   VC      0.5
## 9   5.2   VC      0.5
## 10  7.0   VC      0.5
## 11 16.5   VC      1.0
## 12 16.5   VC      1.0
## 13 15.2   VC      1.0
## 14 17.3   VC      1.0
## 15 22.5   VC      1.0
## 16 17.3   VC      1.0
## 17 13.6   VC      1.0
## 18 14.5   VC      1.0
## 19 18.8   VC      1.0
## 20 15.5   VC      1.0
## 21 23.6   VC      2.0
## 22 18.5   VC      2.0
## 23 33.9   VC      2.0
## 24 25.5   VC      2.0
## 25 26.4   VC      2.0
## 26 32.5   VC      2.0
## 27 26.7   VC      2.0
## 28 21.5   VC      2.0
## 29 23.3   VC      2.0
## 30 29.5   VC      2.0
## 31 15.2   OJ      0.5
## 32 21.5   OJ      0.5
## 33 17.6   OJ      0.5
## 34  9.7   OJ      0.5
## 35 14.5   OJ      0.5
## 36 10.0   OJ      0.5
## 37  8.2   OJ      0.5
## 38  9.4   OJ      0.5
## 39 16.5   OJ      0.5
## 40  9.7   OJ      0.5
## 41 19.7   OJ      1.0
## 42 23.3   OJ      1.0
## 43 23.6   OJ      1.0
## 44 26.4   OJ      1.0
## 45 20.0   OJ      1.0
## 46 25.2   OJ      1.0
```

```
## 47 25.8   OJ      1.0
## 48 21.2   OJ      1.0
## 49 14.5   OJ      1.0
## 50 27.3   OJ      1.0
## 51 25.5   OJ      2.0
## 52 26.4   OJ      2.0
## 53 22.4   OJ      2.0
## 54 24.5   OJ      2.0
## 55 24.8   OJ      2.0
## 56 30.9   OJ      2.0
## 57 26.4   OJ      2.0
## 58 27.3   OJ      2.0
## 59 29.4   OJ      2.0
## 60 23.0   OJ      2.0
```

4.3 Változók újrakódolása

recode(): diszkrét változókat tudunk vele újra kódolni.

```
ToothGrowth %>%
  mutate(dose_recode = recode(dose,
                              "0.5" = "small",
                              "1.0" = "medium",
                              "2.0" = "large"))
```

```
##     len supp dose dose_recode
## 1   4.2   VC  0.5       small
## 2  11.5   VC  0.5       small
## 3   7.3   VC  0.5       small
## 4   5.8   VC  0.5       small
## 5   6.4   VC  0.5       small
## 6  10.0   VC  0.5       small
## 7  11.2   VC  0.5       small
## 8  11.2   VC  0.5       small
## 9   5.2   VC  0.5       small
## 10  7.0   VC  0.5       small
## 11 16.5   VC  1.0      medium
## 12 16.5   VC  1.0      medium
## 13 15.2   VC  1.0      medium
## 14 17.3   VC  1.0      medium
## 15 22.5   VC  1.0      medium
## 16 17.3   VC  1.0      medium
## 17 13.6   VC  1.0      medium
## 18 14.5   VC  1.0      medium
## 19 18.8   VC  1.0      medium
## 20 15.5   VC  1.0      medium
## 21 23.6   VC  2.0       large
## 22 18.5   VC  2.0       large
## 23 33.9   VC  2.0       large
## 24 25.5   VC  2.0       large
## 25 26.4   VC  2.0       large
## 26 32.5   VC  2.0       large
## 27 26.7   VC  2.0       large
## 28 21.5   VC  2.0       large
## 29 23.3   VC  2.0       large
```

```
## 30 29.5   VC  2.0       large
## 31 15.2   OJ  0.5       small
## 32 21.5   OJ  0.5       small
## 33 17.6   OJ  0.5       small
## 34  9.7   OJ  0.5       small
## 35 14.5   OJ  0.5       small
## 36 10.0   OJ  0.5       small
## 37  8.2   OJ  0.5       small
## 38  9.4   OJ  0.5       small
## 39 16.5   OJ  0.5       small
## 40  9.7   OJ  0.5       small
## 41 19.7   OJ  1.0      medium
## 42 23.3   OJ  1.0      medium
## 43 23.6   OJ  1.0      medium
## 44 26.4   OJ  1.0      medium
## 45 20.0   OJ  1.0      medium
## 46 25.2   OJ  1.0      medium
## 47 25.8   OJ  1.0      medium
## 48 21.2   OJ  1.0      medium
## 49 14.5   OJ  1.0      medium
## 50 27.3   OJ  1.0      medium
## 51 25.5   OJ  2.0       large
## 52 26.4   OJ  2.0       large
## 53 22.4   OJ  2.0       large
## 54 24.5   OJ  2.0       large
## 55 24.8   OJ  2.0       large
## 56 30.9   OJ  2.0       large
## 57 26.4   OJ  2.0       large
## 58 27.3   OJ  2.0       large
## 59 29.4   OJ  2.0       large
## 60 23.0   OJ  2.0       large
```

case_when(): diszkrét változókat tudunk vele generlálni akár folytonos akár diszkrét változókból.

```
ToothGrowth %>%
  mutate(dose_descriptive = case_when(dose == 0.5 ~ "small",
                                      dose > 0.5 ~ "medium_to_large"))
```

```
##     len supp dose dose_descriptive
## 1   4.2   VC  0.5            small
## 2  11.5   VC  0.5            small
## 3   7.3   VC  0.5            small
## 4   5.8   VC  0.5            small
## 5   6.4   VC  0.5            small
## 6  10.0   VC  0.5            small
## 7  11.2   VC  0.5            small
## 8  11.2   VC  0.5            small
## 9   5.2   VC  0.5            small
## 10  7.0   VC  0.5            small
## 11 16.5   VC  1.0  medium_to_large
## 12 16.5   VC  1.0  medium_to_large
## 13 15.2   VC  1.0  medium_to_large
## 14 17.3   VC  1.0  medium_to_large
## 15 22.5   VC  1.0  medium_to_large
## 16 17.3   VC  1.0  medium_to_large
```

```
## 17 13.6   VC  1.0  medium_to_large
## 18 14.5   VC  1.0  medium_to_large
## 19 18.8   VC  1.0  medium_to_large
## 20 15.5   VC  1.0  medium_to_large
## 21 23.6   VC  2.0  medium_to_large
## 22 18.5   VC  2.0  medium_to_large
## 23 33.9   VC  2.0  medium_to_large
## 24 25.5   VC  2.0  medium_to_large
## 25 26.4   VC  2.0  medium_to_large
## 26 32.5   VC  2.0  medium_to_large
## 27 26.7   VC  2.0  medium_to_large
## 28 21.5   VC  2.0  medium_to_large
## 29 23.3   VC  2.0  medium_to_large
## 30 29.5   VC  2.0  medium_to_large
## 31 15.2   OJ  0.5            small
## 32 21.5   OJ  0.5            small
## 33 17.6   OJ  0.5            small
## 34  9.7   OJ  0.5            small
## 35 14.5   OJ  0.5            small
## 36 10.0   OJ  0.5            small
## 37  8.2   OJ  0.5            small
## 38  9.4   OJ  0.5            small
## 39 16.5   OJ  0.5            small
## 40  9.7   OJ  0.5            small
## 41 19.7   OJ  1.0  medium_to_large
## 42 23.3   OJ  1.0  medium_to_large
## 43 23.6   OJ  1.0  medium_to_large
## 44 26.4   OJ  1.0  medium_to_large
## 45 20.0   OJ  1.0  medium_to_large
## 46 25.2   OJ  1.0  medium_to_large
## 47 25.8   OJ  1.0  medium_to_large
## 48 21.2   OJ  1.0  medium_to_large
## 49 14.5   OJ  1.0  medium_to_large
## 50 27.3   OJ  1.0  medium_to_large
## 51 25.5   OJ  2.0  medium_to_large
## 52 26.4   OJ  2.0  medium_to_large
## 53 22.4   OJ  2.0  medium_to_large
## 54 24.5   OJ  2.0  medium_to_large
## 55 24.8   OJ  2.0  medium_to_large
## 56 30.9   OJ  2.0  medium_to_large
## 57 26.4   OJ  2.0  medium_to_large
## 58 27.3   OJ  2.0  medium_to_large
## 59 29.4   OJ  2.0  medium_to_large
## 60 23.0   OJ  2.0  medium_to_large
```

Gyakorlás

Most a titanic adattáblán fogunk gyakorolni. (Installáld fel a titanic package-et ha még nincs meg.)

```
library(titanic)

titanic_data <- titanic_train %>%
  drop_na()
?titanic_train
```

```
## starting httpd help server ... done
```

- Mennyi volt a hajón utazó utasok közül a férfiak és nők átlag életkora?
- Hozz létre egy age_group nevű változót, amiben a következő csoportokba kerülnek életkoruk szerint az utasok: 0-14, 15-21, 22-35, 36-50, 50-63, 64+
- Nézd meg, hogy hányan élték túl a különböző osztályokon utazó utasok közül?
- Osztályonként melyik korcsoportban élték túl a legtöbben?

4.4 A mutate() függvény változatai (ajánlott anyag)

A mutate függvény különböző változataival több változót is megváltozhtathatunk egyszerre! A mutate_all() az összes változót egyszerre megváltoztathatjuk

```
ToothGrowth %>%
  mutate_all(.funs = list(~ as.character(.)))
```

A mutate_at bizonyos változókra használ egy függvényt

```
ToothGrowth_factor <-
  ToothGrowth %>%
    mutate_at(list(~ as.factor(.)), .vars = vars(supp, dose))
```

Most leelenőrizhetjük, hogy tényleg faktor típusra változtattuk-e a supp és dose változókat.

```
is.factor(ToothGrowth_factor$dose)
```

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

A mutate_if() függvény csak azokra a változókra használ egy függvényt, amelyek eleget tesznek egy feltételnek.

```
ToothGrowth %>%
  mutate_if(.predicate = is.factor, .funs = list( ~ stringr::str_to_lower(.)))
```

```
##     len supp dose
## 1   4.2   vc  0.5
## 2  11.5   vc  0.5
## 3   7.3   vc  0.5
## 4   5.8   vc  0.5
## 5   6.4   vc  0.5
## 6  10.0   vc  0.5
## 7  11.2   vc  0.5
## 8  11.2   vc  0.5
## 9   5.2   vc  0.5
## 10  7.0   vc  0.5
## 11 16.5   vc  1.0
## 12 16.5   vc  1.0
## 13 15.2   vc  1.0
## 14 17.3   vc  1.0
## 15 22.5   vc  1.0
## 16 17.3   vc  1.0
## 17 13.6   vc  1.0
## 18 14.5   vc  1.0
## 19 18.8   vc  1.0
## 20 15.5   vc  1.0
## 21 23.6   vc  2.0
## 22 18.5   vc  2.0
## 23 33.9   vc  2.0
## 24 25.5   vc  2.0
```

```
## 25 26.4   vc  2.0
## 26 32.5   vc  2.0
## 27 26.7   vc  2.0
## 28 21.5   vc  2.0
## 29 23.3   vc  2.0
## 30 29.5   vc  2.0
## 31 15.2   oj  0.5
## 32 21.5   oj  0.5
## 33 17.6   oj  0.5
## 34  9.7   oj  0.5
## 35 14.5   oj  0.5
## 36 10.0   oj  0.5
## 37  8.2   oj  0.5
## 38  9.4   oj  0.5
## 39 16.5   oj  0.5
## 40  9.7   oj  0.5
## 41 19.7   oj  1.0
## 42 23.3   oj  1.0
## 43 23.6   oj  1.0
## 44 26.4   oj  1.0
## 45 20.0   oj  1.0
## 46 25.2   oj  1.0
## 47 25.8   oj  1.0
## 48 21.2   oj  1.0
## 49 14.5   oj  1.0
## 50 27.3   oj  1.0
## 51 25.5   oj  2.0
## 52 26.4   oj  2.0
## 53 22.4   oj  2.0
## 54 24.5   oj  2.0
## 55 24.8   oj  2.0
## 56 30.9   oj  2.0
## 57 26.4   oj  2.0
## 58 27.3   oj  2.0
## 59 29.4   oj  2.0
## 60 23.0   oj  2.0
```