

Urejanje podatkov

Gregor Pirš, Matej Pičulin in Erik Štrumbelj

2021-06-14

Contents

Uvod	5
Struktura te knjige	6
Stil programske kode	6
1 Slovnica ravnanja s podatki	7
1.1 Priprava	8
1.2 Sodobna razpredelnica: tibble	12
1.3 Urejeno ovrednotenje	14
1.4 Izbira vrstic s filter()	15
1.5 Izbira stolpcev s select()	18
1.6 Urejanje vrstic z arrange()	20
1.7 Dodajanje novih spremenljivk z mutate()	22
1.8 Povzemanje vrednosti s summarise()	23
1.9 Pipe	26
1.10 filter() in mutate() na združenih podatkih	28
1.11 Izvajanje operacij nad večimi stolpci z across()	30
1.12 Povzemanje stolpcev	31
1.13 Povzemanje vrstic	32
1.14 Dodatek	35
1.15 Ali želite izvedeti več?	39
1.16 Domača naloga	39

2	Urejeni in relacijski podatki	47
2.1	Priprava	48
2.2	Urejeni podatki	52
2.3	<code>pivot_longer()</code> : pretvorba v daljšo obliko	56
2.4	<code>pivot_wider()</code> : pretvorba v širšo obliko	57
2.5	<code>separate()</code> in <code>unite()</code> : deljenje in združevanje stolpcev	59
2.6	Relacijske zbirke podatkov	61
2.7	Primer: Bančni podatki	61
2.8	Ključni	64
2.9	Združevanja	66
2.10	Operacije nad množicami	77
2.11	Ali želite izvedeti več?	78
2.12	Domača naloga	79
3	Nizi, kategorične spremenljivke in datumi	85
3.1	Priprava	85
3.2	Nizi	91
3.3	Kategorične spremenljivke	100
3.4	Datumi in ure	109
3.5	Shranjevanje in branje podatkov	119
3.6	Ali želite izvedeti več?	125
3.7	Domača naloga	125

Uvod

Pri delu s podatki se srečujemo z različnimi izzivi. Prvi izziv je zbiranje do podatkov. Takoj za tem pa se soočimo z drugim izzivom, ki običajno zahteva največ našega časa – čiščenje in urejanje podatkov. V večini primerov so podatki v izvorni obliki namreč **neurejeni** (ang. **messy** data).

Ta knjiga je namenjena spoznanju osnovnih konceptov čiščenja in urejanja podatkov, ki nam bodo olajšali nadaljnjo analizo in vizualizacijo. Vse koncepte bomo prikazali v programskem jeziku R. Podatkovne množice in izvirne datoteke te knjige so objavljene na Github repozitoriju. Cilj delavnice je spoznati:

- 1) Najbolj uporabne funkcije za urejanje podatkov;
- 2) Koncept t. i. **urejenih** (ang. **tidy**) podatkov;
- 3) Dobre prakse dela z datumi, nizi in kategoričnimi spremenljivkami.

Za sistematično delo s podatki v R-ju je bil razvit skupek paketov, ki se imenuje **tidyverse**. Sestavljen je iz 8 temeljnih paketov:

- **ggplot2**. Vizualizacija podatkov s **slovnico grafike** (ang. **grammar of graphics**).
- **dplyr**. Lažje urejanje podatkov, na primer izbiranje vrstic in stolpcev, dodajanje stolpcev, povzemanje in urejanje podatkov. Ta paket je glavna tema 1. predavanja.
- **tidyr**. Preoblikovanje podatkov med dolgo in široko obliko, oziroma preoblikovanje podatkov v urejeno obliko. Več o tem bomo povedali na 2. predavanju.
- **readr**. Učinkovito branje in shranjevanje podatkov.
- **purrr**. Funkcijsko programiranje v R.
- **tibble**. Moderna verzija **data.frame**. Tema 1. predavanja.
- **stringr**. Preprostejše delo z nizi. Tema 3. predavanja.
- **forcats**. Preprostejše delo s kategoričnimi spremenljivkami. Tema 3. predavanja.

Vseh 8 paketov lahko namestimo z enim ukazom:

```
install.packages("tidyverse")
```

Lahko pa namestimo tudi samo posamezne pakete:

```
install.packages("dplyr")
```

Struktura te knjige

Vsako poglavje ima 3 sklope:

- 1) **Priprava.** Ta sklop je namenjen temu, da se udeleženci pripravijo na predavanje. Ker bodo le-ta intenzivna in namenjena predstavitvi glavnih konceptov ter uporabi funkcij na praktičnih primerih, je dobro, da poznamo osnovne klice uporabljenih funkcij. V pripravi si bomo na preprostih podatkih pogledali, kako izvajati osnovne klice funkcij v tidyverse. Za vsako pripravo je na voljo video. Priprava traja največ 30 minut.
- 2) **Jedro.** V tem sklopu je zajeta vsebina posameznega predavanja in včasih dodatna snov, ki jo predelamo samostojno. Podrobneje opišemo posamezne koncepte in funkcije ter demonstriramo na praktičnih primerih.
- 3) **Domača naloga.** Na koncu vsakega predavanja so vaje za utrjevanje. Poskusimo jih rešiti sami. V tej knjigi bodo prikazani samo rezultati rešitev brez postopka oziroma programske kode. V kolikor se nam zatakne, lahko preverimo rešitev v izvornih datotekah Rmd, ki se nahajajo na repozitoriju. Nekatere naloge od nas zahtevaj, da kaj raziščemo sami, z uporabo vgrajene pomoči ali spleta, kot smo to navajeni pri vsakodnevnem programerskem delu. Domača naloga vsakega sklopa je sestavljena iz nekaj osnovnih nalog, ki ponovijo snov predavanj. Poleg teh pa so tudi težje naloge, pri kateri je potrebno koncepte uporabiti na realni podatkovni množici ali samostojno rešiti probleme, ki jih na predavanju ne bomo predelali.

Stil programske kode

V tej knjigi bomo predelali potrebne koncepte za urejanje podatkov, kar nam bo omogočilo bolj kvalitetno in učinkovito delo s podatki. Poleg poznavanja teh orodij in konceptov pa nam analizo olajša tudi konsistenten stil programiranja. Dober stil programiranja za naše delo ni nujen, je pa vsekakor dobrodošel, saj je programska koda bolj berljiva. Zbirka paketov tidyverse ima tudi svoj stilski vodnik. Vsak stilski vodnik vsebuje pravila, ki so določena dokaj arbitrarno, oziroma glede na preference avtorja. Najbolj pomembno je, da smo pri pisanju programske kode konsistentni in stilski vodnik nam to nudi.

Chapter 1

Slovnica ravnanja s podatki

Pri sistematičnem delu s podatki je zelo pomembna berljivost in preglednost programske kode. Pri delu velikokrat delimo svojo programsko kodo z drugimi strokovnjaki, zato je zaželeno, da je koda pregledna in razumljiva. Lahko se tudi zgodi, da imamo težave z razumevanjem svoje starejše kode. Zaradi tega je smiselno, da stremimo k čimvečji konsistentnosti in preglednosti. Na dolgi rok s tem prihranimo čas – s konsistentno uporabo enakih ukazov postanemo bolj učinkoviti, prav tako pa lažje prenesemo programsko kodo iz enega problema na drugega. Naša programska koda je s tem bolj robustna in ponovljiva, kar je pomembno iz vidika iskanja napak v analizah. Dober primer konsistentnosti in razumljivosti za sporazumevanje je naraven jezik – v kolikor dve osebi govorita enak jezik se bosta brez težav sporazumevali. Lahko si predstavljate, da ne boste imeli težav z razumevanjem teksta v slovenščini, če se bo avtor držal slovničnih in pravopisnih pravil. Tako kot se pri naravnem jeziku držimo določenih pravil, lahko podobno dosežemo tudi s programskim jezikom.

V tem predavanju se bomo osredotočili na temeljne operacije, ki jih izvajamo nad podatki. Te operacije so nepogrešljive pri vsaki analizi:

- izbira podmnožice vrstic,
- izbira podmnožice stolpcev,
- dodajanje stolpcev, ki so lahko izpeljani iz obstoječih stolpcev,
- urejanje razpredelnice glede na vrednosti stolpcev,
- povzemanje razpredelnic, na primer povprečja in vsote.

Paket **dplyr** vsebuje funkcije, ki nam v primerjavi z osnovno različico R-ja, te operacije olajšajo. Temelji na t. i. **slovnici ravnanja s podatki** (ang. **grammar of data manipulation**), ki programsko kodo pretvori v nekaj podobnega naravnemu jeziku.

Pri slovnici ravnanja s podatki poznamo 5 osnovnih glagolov, s katerimi preoblikujemo podatke. Vsak glagol ustreza eni izmed temeljnih operacij, ki smo

jih omenili zgoraj. Programska koda se potem bere podobno kot naravni jezik – glagoli programskemu jeziku povedo, kaj naj s podatki naredi. Ti glagoli so implementirani v obliki funkcij:

- `filter()` Izbira podmnožice vrstic glede na izbrane pogoje.
- `select()` Izbira podmnožice stolpcev, glede na imena stolpcev.
- `mutate()` Dodajanje stolpcev, ki so lahko izpeljani iz obstoječih stolpcev.
- `summarise()` Povzemanje podatkov v razpredelnici.
- `arrange()` Urejanje razpredelnice.

V tem predavanju bomo bolj podrobno spoznali vsakega izmed teh glagolov. Po tem pa si bomo ogledali še dva uporabna povzetka – povzemanje po vrsticah in povzemanje po stolpcih.

1.1 Priprava

V pripravi se bomo naučili osnovnih klicev petih glagolov iz slovnice ravnanja s podatki. Hkrati bomo primerjali osnovno različico R-ja z uporabo paketa `dplyr`. Pripravimo podatke:

```
library(tidyverse) # Nalozimo celotno zbirko paketov tidyverse.
df <- data.frame(
  ime = c("Maja", "Ales", "Tom", "Barbara", "Simon", "Tina"),
  spol = c("z", "m", "m", "z", "m", "z"),
  starost = c(23, 54, 21, 35, 53, 21),
  visina = c(170, 180, 192, 168, 177, 182)
)
```

S funkcijo `filter()` izberemo podmnožico vrstic v razpredelnici glede na izbrane pogoje. Izberimo ženske nižje od 180 centimetrov.

```
# Osnovni R:
df[df$spol == "z" & df$visina < 180, ]
```

```
##      ime spol starost visina
## 1  Maja   z      23    170
## 4 Barbara z      35    168
```

```
# dplyr:
filter(df, spol == "z", visina < 180)
```

```
##      ime spol starost visina
## 1  Maja   z      23    170
## 2 Barbara z      35    168
```


Opazimo, da z uporabo dplyr ni potrebno vsakič pisati `df$` pred imenom spremenljivke. Tukaj gre za t. i. **maskiranje podatkov** (ang. **data masking**). Več o tem v jedru poglavja.

S funkcijo `select()` izberemo podmnožico stolpcev. Izberimo stolpce `ime`, `spol` in `visina`:

```
# Osnovni R:
df[, c("ime", "spol", "visina")]
```

```
##      ime spol visina
## 1   Maja   z    170
## 2   Ales   m    180
## 3    Tom   m    192
## 4 Barbara z    168
## 5  Simon   m    177
## 6   Tina   z    182
```

```
# dplyr:
select(df, ime, spol, visina)
```

```
##      ime spol visina
## 1   Maja   z    170
## 2   Ales   m    180
## 3    Tom   m    192
## 4 Barbara z    168
## 5  Simon   m    177
## 6   Tina   z    182
```

Opazimo, da nam pri uporabi dplyr stolpcev ni potrebno pisati v narekovajih. Tukaj gre za t. i. **urejeno izbiranje** (ang. **tidy selection**). Več o tem v jedru poglavja.

S funkcijo `mutate()` dodajamo stolpce. Dodajmo višino v metrih:

```
# Osnovni R:
df2 <- df
df2$visina_v_metrih <- df2$visina / 100
df2
```

```
##      ime spol starost visina visina_v_metrih
## 1   Maja   z      23    170             1.70
## 2   Ales   m      54    180             1.80
## 3    Tom   m      21    192             1.92
## 4 Barbara z      35    168             1.68
## 5  Simon   m      53    177             1.77
## 6   Tina   z      21    182             1.82
```

```
# dplyr:
mutate(df, visina_v_metrih = visina / 100)
```

```
##      ime spol starost visina visina_v_metrih
## 1   Maja   z     23   170           1.70
## 2    Ales   m     54   180           1.80
## 3     Tom   m     21   192           1.92
## 4 Barbara z     35   168           1.68
## 5   Simon   m     53   177           1.77
## 6    Tina   z     21   182           1.82
```

S funkcijo `arrange()` urejamo razpredelnico. Uredimo osebe po starosti:

```
# Osnovni R:
df[order(df$starost), ]
```

```
##      ime spol starost visina
## 3     Tom   m     21   192
## 6    Tina   z     21   182
## 1   Maja   z     23   170
## 4 Barbara z     35   168
## 5   Simon   m     53   177
## 2    Ales   m     54   180
```

```
# dplyr:
arrange(df, starost)
```

```
##      ime spol starost visina
## 1     Tom   m     21   192
## 2    Tina   z     21   182
## 3   Maja   z     23   170
## 4 Barbara z     35   168
## 5   Simon   m     53   177
## 6    Ales   m     54   180
```

S funkcijo `summarise()` povzamemo podatke. Običajno se uporablja v kombinaciji z `group_by()`. Izračunajmo povprečno višino glede na spol:

```
# Osnovni R:
aggregate(visina ~ spol, data = df, FUN = mean)
```

```
##   spol   visina
## 1    m 183.0000
## 2    z 173.3333
```

```
# dplyr:
summarise(group_by(df, spol), povp_visina = mean(visina))
```

```
## # A tibble: 2 x 2
##   spol povp_visina
##   <chr>      <dbl>
## 1 m          183
## 2 z          173.
```

Naloga: Poglejmo si nov primer podatkov.

```
df <- data.frame(
  podjetje = c("A", "B", "C", "D", "E"),
  panoga = c("proizvodnja", "gostinstvo", "proizvodnja", "gostinstvo", "proizvodnja"),
  st_zaposlenih = c(100, 20, 110, 15, 20),
  dobiček = c(100000, 10000, 12000, 1000, 0)
)
```

Z uporabo dplyr:

- Izberite vrstice, ki imajo med (vključno) 10000 in 20000 dobička.

```
##   podjetje      panoga st_zaposlenih dobiček
## 1      B  gostinstvo         20    10000
## 2      C proizvodnja        110    12000
```

- Izberite drugi in četrti stolpec.

```
##           panoga dobiček
## 1 proizvodnja 100000
## 2 gostinstvo  10000
## 3 proizvodnja 12000
## 4 gostinstvo  1000
## 5 proizvodnja    0
```

- Dodajte stolpec, ki bo prikazal dobiček na zaposlenega.

```
##   podjetje      panoga st_zaposlenih dobiček dobiček_na_zaposlenega
## 1      A proizvodnja        100 100000      1000.00000
## 2      B  gostinstvo         20  10000       500.00000
## 3      C proizvodnja        110 12000      109.09091
## 4      D  gostinstvo         15  1000       66.66667
## 5      E proizvodnja         20     0        0.00000
```

- | ## | podjetje | panoga | st_zaposlenih | dobicek |
|------|----------|-------------|---------------|---------|
| ## 1 | D | gostinstvo | 15 | 1000 |
| ## 2 | B | gostinstvo | 20 | 10000 |
| ## 3 | E | proizvodnja | 20 | 0 |
| ## 4 | A | proizvodnja | 100 | 100000 |
| ## 5 | C | proizvodnja | 110 | 12000 |

- ```
A tibble: 2 x 2
panoga max_st_zaposlenih
<chr> <dbl>
1 gostinstvo 20
2 proizvodnja 110
```

Najprej si pogledjmo podatke, na katerih se bomo naučili osnovnih konceptov slovnice ravnanja s podatki. V mapi *data-raw* se nahajajo podatki *DS-jobs.csv*. Gre za rezultate ankete, ki so jo v povezavi z industrijo izvedli na spletni strani Kaggle (<https://www.kaggle.com/kaggle/kaggle-survey-2017>) leta 2017 z namenom raziskati trg dela na področju podatkovnih ved in strojnega učenja. Podatki so shranjeni v tekstovni datoteki, kjer so elementi ločeni s podpičjem. Preberimo podatke v našo sejo R:

| ##   | Gender                               | Country        | Age | EmploymentStatus     |                        |
|------|--------------------------------------|----------------|-----|----------------------|------------------------|
| ## 1 | Female                               | Australia      | 43  | Employed full-time   |                        |
| ## 2 | Male                                 | Russia         | 33  | Employed full-time   |                        |
| ## 3 | Male                                 | Taiwan         | 26  | Employed full-time   |                        |
| ## 4 | Male                                 | United States  | 25  | Employed part-time   |                        |
| ## 5 | Male                                 | United States  | 33  | Employed full-time   |                        |
| ## 6 | Male                                 | Czech Republic | 21  | Employed part-time   |                        |
| ##   |                                      |                |     | CurrentJobTitle      | LanguageRecommendation |
| ## 1 |                                      |                |     | Business Analyst     | Python                 |
| ## 2 | Software Developer/Software Engineer |                |     |                      | Python                 |
| ## 3 | Software Developer/Software Engineer |                |     |                      | Python                 |
| ## 4 |                                      |                |     | Researcher           | Python                 |
| ## 5 |                                      |                |     | Scientist/Researcher | Matlab                 |

```
6 Other Python
FormalEducation
1 Bachelor's degree
2 Bachelor's degree
3 Master's degree
4 Bachelor's degree
5 Doctoral degree
6 Some college/university study without earning a bachelor's degree
Major CompensationAmount CompensationCurrency
1 80000 AUD
2 Other 1200000 RUB
3 Computer Science 1100000 TWD
4 Physics 20000 USD
5 Electrical Engineering 100000 USD
6 Computer Science 20000 CZK
TimeGatheringData TimeModelBuilding TimeProduction TimeVisualizing
1 60 10 5 15
2 40 30 15 10
3 35 20 25 10
4 0 80 0 20
5 0 0 0 0
6 20 60 20 0
TimeFindingInsights TimeOtherSelect ExchangeRate
1 10 0 0.802310
2 5 0 0.017402
3 10 0 0.033304
4 0 0 1.000000
5 0 0 1.000000
6 0 0 0.045820
```

Spremenljivka `ds_jobs` je tipa `data.frame`. To je osnovna oblika, v kateri v R hranimo razpredelnice. V tidyverse obstaja paket **tibble**, ki je namenjen sodobni predstavitvi razpredelnice. Glavna funkcionalnost tega paketa je objekt **tibble**, ki predstavlja nadgradnjo data frame. V preostanku knjige bomo za objekte `data.frame` uporabljali izraz “data frame” in za objekte tipa **tibble** izraz “tibble”. Večina funkcij v tidyverse sicer lahko kot vhodni podatek prejme data frame, ampak ga nekatere potem samodejno pretvorijo v tibble. Kot dobro prakso predlagamo delo izključno s tibble. Poleg kompatibilnosti s funkcijami tidyverse je še nekaj drugih razlik v primerjavi z data frame, večino le-teh bomo spoznali v preostanku knjige.

Pretvorimo sedaj ta data frame v tibble s funkcijo `as_tibble()`.

```
library(tidyverse)
ds_jobs <- as_tibble(ds_jobs)
ds_jobs
```

```
A tibble: 4,523 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecomme~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Netherl~ 51 Employed full-time Engineer R
9 Male Colombia 34 Employed full-time Data Scientist Python
10 Male Germany 41 Independent contrac~ Data Scientist Python
... with 4,513 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```

Opazimo, da je oblika prikaza podatkov sedaj nekoliko drugačna. Najbolj očitna razlika je, da imamo na zaslonu prikazanih samo toliko stolpcev, kot jih je možno prikazati na zaslonu. Preostali stolpci so samo zapisani zaporedno z imeni, da lahko vidimo, katere stolpce še imamo v podatkih. S tem preprečimo, da bi izpis postal nepregleden. Še vedno lahko vidimo vse oziroma več stolpcev z uporabo `View()` ali pa če tibble izpišemo s pomočjo `print()` in ustrezno nastavitvijo širine, na primer `print(ds_jobs, width = 120)`. Izpis tibble pa nam nudi še nekaj dodatnih informacij v primerjavi z data frame. V prvi vrstici imamo izpisano dimenzijo podatkov – število vrstic in število stolpcev. Pod vsako spremenljivko je zapisan tudi njen tip. Tibble tudi dopušča imena stolpcev, ki niso standardna za R (na primer vsebujejo – in podobno), čeprav uporaba takih imen ni dobra praksa. Več o tem bomo povedali kasneje.

### 1.3 Urejeno ovrednotenje

Preden začnemo resneje delati z glagoli slovnice ravnanja s podatki, spoznajmo t. i. **urejeno ovrednotenje** (ang. **tidy evaluation**). To je posebnost tidy-verse in večina glagolov v dplyr ga uporablja. Kaj pa je urejeno ovrednotenje? To je nestandarden pristop k ovrednotenju izrazov v programskem jeziku R. V pripravi smo že srečali dva primera:

- Pri funkciji `filter()` ni bilo potrebno vsakič navesti `df$` za izbiro spremenljivk iz razpredelnice.
- Pri funkciji `select()` nismo potrebovali narekovajev.

Oba sta primera dveh vrst urejenega ovrednotenja:

- Pri nekaterih glagolih v dplyr lahko uporabimo spremenljivke (stolpce) tibbla (ali razpredelnice), kot da bi bile spremenljivke v globalnem okolju (torej lahko uporabimo `moja_spremenljivka` namesto `df$moja_spremenljivka`). Temu pravimo **maskiranje podatkov** (ang. **data masking**). Funkcije, ki podpirajo to strukturo in jih bomo spoznali v nadaljevanju so: `arrange()`, `count()`, `filter()`, `group_by()`, `mutate()` in `summarise()`.
- Pri nekaterih glagolih v dplyr lahko na lažji način izberemo spremenljivke (stolpce) glede na njihovo pozicijo, ime ali tip (na primer izbira stolpcev po imenu brez narekovajev, izbira stolpcev ki se začnejo na določen niz, izbira samo številskih stolpcev). Temu pravimo **urejeno izbiranje** (ang. **tidy selection**). Funkcije, ki podpirajo to strukturo so: `across()`, `count()`, `rename()`, `select()` in `pull()`.

Informacije o tem, ali funkcija vsebuje maskiranje podatkov ali urejeno izbiranje, lahko najdemo v datoteki s pomočjo pod razdelkom *Arguments*.

## 1.4 Izbira vrstic s `filter()`

S funkcijo `filter()` izbiramo podmnožico vrstic glede na izbrane pogoje. Sintaksa je:

```
filter(<tibble>, <pogoj1>, <pogoj2>, ...)
```

Kot prvi argument podamo tibble s podatki, potem pa z vejicami ločene pogoje. Izberimo vse osebe mlajše od 30 let:

```
library(dplyr)
filter(ds_jobs, Age < 30)
```

```
A tibble: 1,729 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Male Taiwan 26 Employed full-t~ Software Developer~ Python
2 Male United S~ 25 Employed part-t~ Researcher Python
3 Male Czech Re~ 21 Employed part-t~ Other Python
4 Male Russia 22 Employed full-t~ Data Analyst Python
5 Male Poland 29 Employed full-t~ Software Developer~ Python
6 Male Other 28 Employed full-t~ Data Scientist R
7 Male Mexico 26 Employed part-t~ Data Scientist Python
```

```
8 Male Singapore 24 Employed full-t~ Data Analyst Python
9 Male India 29 Employed full-t~ Data Scientist R
10 Male United S~ 25 Employed full-t~ Engineer Python
... with 1,719 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```

Več pogojev ločimo z vejico, kadar želimo, da veljajo vsi pogoji (enakovredno uporabi operatorja *in* – & pri naštevanju pogojev). Poglejmo si vse osebe mlajše od 30 let, ki prihajajo iz Nemčije:

```
filter(ds_jobs, Age < 30, Country == "Germany")
```

```
A tibble: 42 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Germany 24 Employed part-time Scientist/Resear~ R
2 Male Germany 28 Employed full-time Scientist/Resear~ Python
3 Male Germany 24 Independent contract~ Data Scientist Python
4 Female Germany 29 Employed full-time Business Analyst SQL
5 Male Germany 26 Employed part-time Researcher Python
6 Male Germany 27 Employed full-time Data Scientist Python
7 Female Germany 26 Employed part-time Statistician R
8 Male Germany 26 Independent contract~ Data Scientist Python
9 Male Germany 29 Employed full-time Machine Learning~ Python
10 Male Germany 25 Employed full-time Data Scientist Python
... with 32 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```

V kolikor želimo, da velja vsaj eden izmed pogojev, moramo uporabiti operator *ali* – |. Poglejmo si vse osebe mlajše od 30 ali starejše od 50 let:

```
filter(ds_jobs, Age < 30 | Age > 50)
```

```
A tibble: 2,006 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Male Taiwan 26 Employed full-t~ Software Developer~ Python
2 Male United S~ 25 Employed part-t~ Researcher Python
```



```
3 Male Czech Re~ 21 Employed part-t~ Other Python
4 Male Russia 22 Employed full-t~ Data Analyst Python
5 Male Netherla~ 51 Employed full-t~ Engineer R
6 Male Poland 29 Employed full-t~ Software Developer~ Python
7 Male Other 28 Employed full-t~ Data Scientist R
8 Male Mexico 26 Employed part-t~ Data Scientist Python
9 Male Singapore 24 Employed full-t~ Data Analyst Python
10 Male India 29 Employed full-t~ Data Scientist R
... with 1,996 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```

Če želimo nek kategorični stolpec pogojiti z večimi vrednostmi (na primer udeležence iz večih držav), lahko namesto večih | uporabimo operator `%in%`, ki preveri, če je element del množice:

```
filter(ds_jobs, Country %in% c("Germany", "Canada", "Ireland"))
```

```
A tibble: 306 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Male Germany 41 Independent contrac~ Data Scientist Python
2 Female Germany 49 Employed part-time Scientist/Resea~ Python
3 Male Germany 44 Employed full-time Other Python
4 A diffe~ Canada 23 Employed full-time Scientist/Resea~ Python
5 Female Germany 24 Employed part-time Scientist/Resea~ R
6 Male Canada 52 Employed full-time Software Develo~ Python
7 Male Ireland 27 Employed full-time Data Scientist Python
8 Male Canada 24 Employed full-time Business Analyst Python
9 Male Canada 46 Employed full-time Data Scientist Python
10 Male Canada 31 Employed full-time Data Analyst R
... with 296 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```

### 1.4.1 Manjkajoče vrednosti

Vrstice pogosto filtriramo na podlagi manjkajočih vrednosti. Včasih so te pomembne za samo analizo, saj nas lahko zanimajo razlogi za njihov pojav.

Včasih pa so to nepomembne vrstice, saj nam ne prinesejo dodatne informacije. V tem primeru jih običajno kar izločimo iz nadaljnje analize.

V nadaljevanju bomo spoznali, kako dodati nov stolpec, in to ilustrirali na izračunu plače v dolarjih. Za to bomo potrebovali stolpca `CompensationAmount` in `ExchangeRate`. V slednjem je kar nekaj manjkajočih vrednosti. Takšne vrstice bodo za analizo plač neuporabne, zato jih bomo sedaj izločili iz podatkov. Ali je vrednost enaka `NA` (objekt, ki predstavlja manjkajočo vrednost v R) preverimo s funkcijo `is.na()`. Izločimo sedaj te vrstice:

```
ds_jobs <- filter(ds_jobs, !is.na(ExchangeRate))
```

Podobno kot v zgornjem primeru lahko v pogoju nastopa poljubna funkcija.

## 1.5 Izbira stolpcev s `select()`

S funkcijo `select()` izbiramo podmnožico stolpcev. Osnovna sintaksa je:

```
filter(<tibble>, <stolpec1>, <stolpec2>, ...)
```

Izberimo sedaj stolpce `Country`, `Age` in `EmploymentStatus`.

```
select(ds_jobs, Country, Age, EmploymentStatus)
```

```
A tibble: 3,781 x 3
Country Age EmploymentStatus
<chr> <int> <chr>
1 Australia 43 Employed full-time
2 Russia 33 Employed full-time
3 Taiwan 26 Employed full-time
4 United States 25 Employed part-time
5 United States 33 Employed full-time
6 Czech Republic 21 Employed part-time
7 Russia 22 Employed full-time
8 Colombia 34 Employed full-time
9 Germany 41 Independent contractor, freelancer, or self-employed
10 Poland 29 Employed full-time
... with 3,771 more rows
```

Izberimo vse stolpce razen teh treh stolpcev. Za to enostavno dodamo minus pred imenom stolpca, ki ga želimo izločiti:

```
select(ds_jobs, -Country, -Age, -EmploymentStatus)
```

```
A tibble: 3,781 x 14
Gender CurrentJobTitle LanguageRecommen~ FormalEducation Major
<chr> <chr> <chr> <chr> <chr>
1 Female Business Analyst Python Bachelor's degree ""
2 Male Software Develop~ Python Bachelor's degree "Other"
3 Male Software Develop~ Python Master's degree "Computer ~
4 Male Researcher Python Bachelor's degree "Physics"
5 Male Scientist/Researc~ Matlab Doctoral degree "Electrica~
6 Male Other Python Some college/univers~ "Computer ~
7 Male Data Analyst Python Bachelor's degree "Informati~
8 Male Data Scientist Python Master's degree "Computer ~
9 Male Data Scientist Python I did not complete a~ ""
10 Male Software Develop~ Python Master's degree "Computer ~
... with 3,771 more rows, and 9 more variables: CompensationAmount <dbl>,
CompensationCurrency <chr>, TimeGatheringData <int>,
TimeModelBuilding <dbl>, TimeProduction <dbl>, TimeVisualizing <dbl>,
TimeFindingInsights <dbl>, TimeOtherSelect <int>, ExchangeRate <dbl>
```

Izberimo vse stolpce med Country in Major. Podobno kot v R 1:10 našteje vsa cela števila med 1 in 10, operator `:` v tidyverse izbere vse stolpce med Country in Major:

```
select(ds_jobs, Country:Major)
```

```
A tibble: 3,781 x 7
Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <int> <chr> <chr> <chr>
1 Australia 43 Employed full-time Business Analyst Python
2 Russia 33 Employed full-time Software Developer~ Python
3 Taiwan 26 Employed full-time Software Developer~ Python
4 United S~ 25 Employed part-time Researcher Python
5 United S~ 33 Employed full-time Scientist/Researc~ Matlab
6 Czech Re~ 21 Employed part-time Other Python
7 Russia 22 Employed full-time Data Analyst Python
8 Colombia 34 Employed full-time Data Scientist Python
9 Germany 41 Independent contractor~ Data Scientist Python
10 Poland 29 Employed full-time Software Developer~ Python
... with 3,771 more rows, and 2 more variables: FormalEducation <chr>,
Major <chr>
```

Izberimo vse stolpce, ki se začnejo z besedo Time. Za to bomo uporabili funkcijo `starts_with()`. Ta funkcija je t. i. *selection helper*, kar pomeni, da jo lahko

uporabimo le znotraj funkcij, ki omogočajo urejeno izbiro in nam omogoča lažjo izbiro na podlagi nekega pogoja. V tem primeru je ta pogoj, da se beseda začne na določen niz:

```
select(ds_jobs, starts_with("Time"))
```

```
A tibble: 3,781 x 6
TimeGatheringData TimeModelBuilding TimeProduction TimeVisualizing
<int> <dbl> <dbl> <dbl>
1 60 10 5 15
2 40 30 15 10
3 35 20 25 10
4 0 80 0 20
5 0 0 0 0
6 20 60 20 0
7 50 20 10 5
8 60 10 20 5
9 50 10 20 10
10 25 20 25 20
... with 3,771 more rows, and 2 more variables: TimeFindingInsights <dbl>,
TimeOtherSelect <int>
```

Poleg `starts_with()` dplyr vsebuje še več takšnih funkcij:

- `ends_with()` Ali se ime stolpca konča na določen niz?
- `contains()` Ali ime stolpca vsebuje niz?
- `matches()` Ali ime stolpca ustreza regularnemu izrazu? Več o regularnih izrazih bomo povedali v 3. predavanju.
- `num_range()` Ali ime stolpca vsebuje števila znotraj množice števil? Na primer, če imamo stolpce, ki v imenu vsebujejo števila – *stolpec1*, *stolpec2*, in tako naprej.

## 1.6 Urejanje vrstic z `arrange()`

Vrstice lahko tudi uredimo glede na vrednosti v posameznih stolpcih. Za to uporabimo funkcijo `arrange()`. Sintaksa te funkcije je:

```
arrange(<tibble>, <stolpec1>, <stolpec2>, ...)
```

kjer stolpci predstavljajo vrednosti, po katerih želimo urediti tibble.

Ustvarimo najprej nov tibble, v katerem bomo izbrali podmnožico stolpcev.

```
ds_jobs_tmp <- select(ds_jobs, CurrentJobTitle, Country,
 CompensationCurrency, Age, CompensationAmount)
```

Uredimo sedaj podatke glede na leta:

```
arrange(ds_jobs_tmp, Age)
```

```
A tibble: 3,781 x 5
CurrentJobTitle Country CompensationCurre~ Age CompensationAmo~
<chr> <chr> <chr> <int> <dbl>
1 Predictive Modeler Australia AUD 0 78000
2 Scientist/Researcher United St~ USD 1 100000
3 Programmer Other GBP 11 0
4 Data Scientist United St~ USD 16 50000
5 Software Developer/Soft~ Russia USD 18 40000
6 Programmer Other USD 18 1000
7 Machine Learning Engine~ Other USD 19 30000
8 Programmer Russia USD 19 40000
9 Scientist/Researcher Canada CAD 19 0
10 Computer Scientist Brazil BRL 19 400
... with 3,771 more rows
```

Opazimo, da imamo nekaj neveljavnih starosti, na primer 0 in 1, najverjetneje tudi 11. Prav tako imamo nekaj nesmiselnih vrednosti v stolpcu o plači. Pri celostni analizi bi seveda raziskali, zakaj je prišlo do takih vrednosti, oziroma bi jih izločili iz analize. Za namen spoznavanja ravnanja s podatki in dplyr to ni pomembno, tako da temu ne bomo posvečali pozornosti. Bralcem pa predlagamo, naj razmislijo, kako bi se tega lotili z že naučenimi koncepti.

Če želimo podatke urediti padajoče, potem uporabimo funkcijo `desc()`:

```
arrange(ds_jobs_tmp, desc(Age))
```

```
A tibble: 3,781 x 5
CurrentJobTitle Country CompensationCurre~ Age CompensationAmo~
<chr> <chr> <chr> <int> <dbl>
1 Statistician United Ki~ ILS 100 1000000000000
2 Other Other EUR 99 15000
3 Researcher Portugal EUR 78 63000
4 Data Scientist Canada USD 75 110
5 Software Developer/Soft~ Netherlan~ EUR 73 40000
6 Data Analyst Russia USD 70 14000
7 Business Analyst United St~ USD 70 130000
8 Machine Learning Engine~ United Ki~ GBP 70 40000
```

```
9 Scientist/Researcher United St~ USD 69 200000
10 Business Analyst United St~ USD 68 125000
... with 3,771 more rows
```

Uredimo lahko tudi glede na več stolpcev, kjer se najprej uredi po prvem zapisanem, nato drugem, itd.:

```
arrange(ds_jobs_tmp, Age, CompensationAmount)
```

```
A tibble: 3,781 x 5
CurrentJobTitle Country CompensationCurre~ Age CompensationAmo~
<chr> <chr> <chr> <int> <dbl>
1 Predictive Modeler Australia AUD 0 78000
2 Scientist/Researcher United St~ USD 1 100000
3 Programmer Other GBP 11 0
4 Data Scientist United St~ USD 16 50000
5 Programmer Other USD 18 1000
6 Software Developer/Soft~ Russia USD 18 40000
7 Scientist/Researcher Canada CAD 19 0
8 Computer Scientist Brazil BRL 19 400
9 Machine Learning Engine~ Other USD 19 30000
10 Programmer Russia USD 19 40000
... with 3,771 more rows
```

## 1.7 Dodajanje novih spremenljivk z `mutate()`

Pogosto želimo ustvariti nove stolpce, ki so izpeljani iz obstoječih stolpcev. Pri naših podatkih imamo stolpec `CompensationAmount`, ki predstavlja letno plačo in `ExchangeRate`, ki predstavlja menjalni tečaj lokalne valute v ameriški dolar. Če želimo imeti primerljive podatke, moramo izračunati vrednosti v dolarjih za vse podatke. Za to uporabimo funkcijo `mutate()`, ki doda stolpec (ali več stolpcev). Sintaksa funkcije je:

```
<tibble> <- mutate(<tibble>, <ime-novega-stolpca> = <funkcija-obstojećih-stolpcev>, ..
```

Dodajmo stolpec `CompensationUSD`, ki bo prikazal letno plačo v USD:

```
ds_jobs <- mutate(ds_jobs, CompensationUSD = CompensationAmount * ExchangeRate)
select(ds_jobs, CompensationAmount, ExchangeRate, CompensationUSD)
```

```
A tibble: 3,781 x 3
CompensationAmount ExchangeRate CompensationUSD
```

```
<dbl> <dbl> <dbl>
1 80000 0.802 64185.
2 1200000 0.0174 20882.
3 1100000 0.0333 36634.
4 20000 1 20000
5 100000 1 100000
6 20000 0.0458 916.
7 624000 0.0174 10859.
8 156000000 0.000342 53352
9 150000 1.20 179374.
10 126000 0.281 35419.
... with 3,771 more rows
```

Znotraj klica `mutate()` lahko uporabimo stolpce, ki smo jih ustvarili v istem klicu v preteklih vrsticah. Recimo, da želimo poleg plače v USD izračunati še mesečno plačo v USD:

```
ds_jobs <- mutate(ds_jobs,
 CompensationUSD = CompensationAmount * ExchangeRate,
 MonthlyCompUSD = CompensationUSD / 12)
select(ds_jobs, CompensationAmount, ExchangeRate, CompensationUSD, MonthlyCompUSD)
```

```
A tibble: 3,781 x 4
CompensationAmount ExchangeRate CompensationUSD MonthlyCompUSD
<dbl> <dbl> <dbl> <dbl>
1 80000 0.802 64185. 5349.
2 1200000 0.0174 20882. 1740.
3 1100000 0.0333 36634. 3053.
4 20000 1 20000 1667.
5 100000 1 100000 8333.
6 20000 0.0458 916. 76.4
7 624000 0.0174 10859. 905.
8 156000000 0.000342 53352 4446
9 150000 1.20 179374. 14948.
10 126000 0.281 35419. 2952.
... with 3,771 more rows
```

## 1.8 Povzemanje vrednosti s `summarise()`

Funkcija `summarise()` se uporablja za povzemanje vrednosti (na primer povprečja, vsote, števci, ...). Sintaksa funkcije je:

```
summarise(<tibble>, <ime-povzetka> = <funkcija-ki-povzame-stolpec>, ...)
```

Najprej pogledajmo delovanje te funkcije, tako da povzamemo povprečen čas priprave podatkov:

```
summarise(ds_jobs, MeanDataCleaning = mean(TimeGatheringData))
```

```
A tibble: 1 x 1
MeanDataCleaning
<dbl>
1 37.3
```

Funkcija enostavno vrne povprečje stolpca `TimeGatheringData`. Ta informacija je sicer uporabna, ampak to ni edina funkcionalnost te funkcije in je običajno ne uporabljamo v tej obliki. Njena moč se izrazi, ko jo uporabimo v kombinaciji z ukazom `group_by()`. Ta ukaz združi vrstice glede na vrednosti v podanih stolpcih. Združene vrednosti imajo posebno funkcijo v paketu `dplyr` in vplivajo na funkcionalnosti funkcij `summarise()`, `mutate()` in `filter()`. Vpliv grupiranja na `mutate()` in `filter()` si bomo ogledali nekoliko kasneje, pogledjmo sedaj vpliv na `summarise()`. Recimo, da nas zanima, v katerih službah je potrebnega največ čiščenja podatkov. Najprej podatke združimo po stolpcu `CurrentJobTitle`, potem pa uporabimo `summarise()`:

```
ds_jobs_grouped <- group_by(ds_jobs, CurrentJobTitle)
summarise(ds_jobs_grouped, MeanDataCleaning = mean(TimeGatheringData))
```

```
A tibble: 17 x 2
CurrentJobTitle MeanDataCleaning
<chr> <dbl>
1 "" 40
2 "Business Analyst" 37.9
3 "Computer Scientist" 33.3
4 "Data Analyst" 41.2
5 "Data Miner" 48.0
6 "Data Scientist" 39.4
7 "DBA/Database Engineer" 37.7
8 "Engineer" 36.4
9 "Machine Learning Engineer" 34.7
10 "Operations Research Practitioner" 37.8
11 "Other" 36.3
12 "Predictive Modeler" 37.1
13 "Programmer" 35.8
14 "Researcher" 31.3
```



```
15 "Scientist/Researcher" 33.5
16 "Software Developer/Software Engineer" 36.9
17 "Statistician" 34.7
```

Izgleda, da so povprečja kar blizu – čas čiščenja podatkov je relativno neodvisen od delovnega mesta.

Povzemamo lahko tudi preko večih stolpcev. Poglejmo si število ljudi z različnimi statusi zaposlitve v kombinaciji z izobrazbo. Da preštejemo število vrstic, ki ustrezajo grupiranju, uporabimo funkcijo `n()`:

```
ds_jobs_grouped <- group_by(ds_jobs, FormalEducation, EmploymentStatus)
summarise(ds_jobs_grouped, Count = n())
```

```
A tibble: 21 x 3
Groups: FormalEducation [8]
FormalEducation EmploymentStatus Count
<chr> <chr> <int>
1 "" Employed full-time 1
2 "Bachelor's degree" Employed full-time 857
3 "Bachelor's degree" Employed part-time 52
4 "Bachelor's degree" Independent contractor, freelanc~ 76
5 "Doctoral degree" Employed full-time 719
6 "Doctoral degree" Employed part-time 26
7 "Doctoral degree" Independent contractor, freelanc~ 50
8 "I did not complete any formal educa~ Employed full-time 13
9 "I did not complete any formal educa~ Employed part-time 2
10 "I did not complete any formal educa~ Independent contractor, freelanc~ 10
... with 11 more rows
```

Ker je štetje primerov zelo pogosta operacija, obstaja tudi funkcija `count()`, ki naredi enako kot kombinacija `group_by()` in `summarise()`:

```
count(ds_jobs, FormalEducation, EmploymentStatus)
```

```
A tibble: 21 x 3
FormalEducation EmploymentStatus n
<chr> <chr> <int>
1 "" Employed full-time 1
2 "Bachelor's degree" Employed full-time 857
3 "Bachelor's degree" Employed part-time 52
4 "Bachelor's degree" Independent contractor, freelanc~ 76
5 "Doctoral degree" Employed full-time 719
6 "Doctoral degree" Employed part-time 26
```

```
7 "Doctoral degree" Independent contractor, freelanc~ 50
8 "I did not complete any formal educa~ Employed full-time 13
9 "I did not complete any formal educa~ Employed part-time 2
10 "I did not complete any formal educa~ Independent contractor, freelanc~ 10
... with 11 more rows
```

## 1.9 Pipe

V praksi ravnanje s podatki zajame večino, če ne kar vseh funkcij, ki smo jih predstavili do sedaj. Če želimo sproti shranjevati naše spremembe, moramo po vsaki uporabi funkcije spremenjene podatke ponovno shraniti v spremenljivko. To lahko postane nekoliko nepregledno. Poglejmo si potek dela, kjer bomo nad osnovnimi podatki izvedli te operacije:

- Izbrali bomo vrstice, kjer so osebe starejše od 30 let in država ni `Other` ali prazen niz.
- Izločili vse stolpce, ki vsebujejo niz `Time`.
- Izračunali stolpec s plačo v ameriških dolarjih.
- Povzeli plačo glede na državo.

Z uporabo shranjevanja podatkov v spremenljivko, kot smo navajeni iz osnovne različice R, bi s funkcijami iz `dplyr` izgledalo takole:

```
ds_jobs2 <- read.csv2("./data-raw/DS-jobs.csv")
ds_jobs2 <- as_tibble(ds_jobs2)
ds_jobs2 <- filter(ds_jobs2, Age > 30, !(Country %in% c("Other", "")))
ds_jobs2 <- select(ds_jobs2, -contains("Time"))
ds_jobs2 <- mutate(ds_jobs2, CompensationUSD = CompensationAmount * ExchangeRate)
ds_jobs2 <- group_by(ds_jobs2, Country)
ds_jobs2_summarised <- summarise(ds_jobs2, MeanCompensation = mean(CompensationUSD, na
ds_jobs2_summarised
```

```
A tibble: 51 x 2
Country MeanCompensation
<chr> <dbl>
1 Argentina 39282.
2 Australia 112800.
3 Belarus 33500
4 Belgium 74141.
5 Brazil 47799.
6 Canada 85471.
7 Chile 44152.
8 Colombia 43303.
```

```
9 Czech Republic 50223.
10 Denmark 88136.
... with 41 more rows
```

Pri računanju povprečja smo uporabili argument `na.rm = T`, s katerim smo manjkajoče vrednosti ignorirali. Celoten postopek je vseboval kar nekaj prepisovanja. Predvsem spremenljivko `ds_jobs2` smo morali prepisati kar 6-krat. Dplyr pa vsebuje poseben operator, ki ga imenujemo *pipe* in ga označimo z `%>%`. S tem operatorjem funkcije povežemo v sosledje. Poglejmo si, kako deluje:

```
ds_jobs2 <- read.csv2("./data-raw/DS-jobs.csv")
ds_jobs2_summarised <- ds_jobs2 %>%
 filter(Age > 30, !(Country %in% c("Other", ""))) %>%
 select(-contains("Time")) %>%
 mutate(CompensationUSD = CompensationAmount * ExchangeRate) %>%
 group_by(Country) %>%
 summarise(MeanCompensation = mean(CompensationUSD, na.rm = T))
ds_jobs2_summarised
```

```
A tibble: 51 x 2
Country MeanCompensation
<chr> <dbl>
1 Argentina 39282.
2 Australia 112800.
3 Belarus 33500
4 Belgium 74141.
5 Brazil 47799.
6 Canada 85471.
7 Chile 44152.
8 Colombia 43303.
9 Czech Republic 50223.
10 Denmark 88136.
... with 41 more rows
```

Sedaj smo do povzetka prišli z zaporednim izvajanjem operacij nad spremenljivko `ds_jobs`. Ta način je bolj pregleden, saj bralec kode takoj opazi, da se je vse izvajalo nad istimi podatki. Opazimo tudi, zakaj gre za **slovnico** ravnanja s podatki. Programska koda zapisana zgoraj se bere skoraj kot naravni jezik. Na primer, **izberi vrstice**, kjer so leta večja od 30 in država ni v ustrezni množici. Zatem **izberi stolpce**, ki ne vsebujejo besede Time. **Dodaj** novo spremenljivko, **združi** podatke in jih **povzemi**.

## 1.10 filter() in mutate() na združenih podatkih

Spoznali smo, kako funkcija `group_by()` vpliva na povzemanje podatkov. Uporabimo pa jo lahko tudi v povezavi s `filter()` in `mutate()`. Kombinacija z izbiro vrstic pride prav, kadar želimo pogojno izbiro na nek drugi stolpec. Kot primer si pogledajmo, kako bi iz podatkov za vsako državo filtrirali top 3 anketirance, ki prejmejo najvišjo plačo. Najprej bomo podatke grupirali, nato pa uporabili filter:

```
ds_jobs %>%
 select(Country, Age, CurrentJobTitle, CompensationUSD) %>%
 group_by(Country) %>%
 filter(rank(desc(CompensationUSD)) <= 3) %>%
 arrange(Country)
```

```
A tibble: 154 x 4
Groups: Country [53]
Country Age CurrentJobTitle CompensationUSD
<chr> <int> <chr> <dbl>
1 "" NA Data Scientist 107624.
2 "" 63 Machine Learning Engineer 160000
3 "" NA Operations Research Practitioner 120000
4 "Argentina" 55 Data Scientist 100000
5 "Argentina" 26 Data Scientist 65000
6 "Argentina" 26 Data Scientist 80000
7 "Australia" 39 Data Scientist 280808.
8 "Australia" 50 Data Miner 248716.
9 "Australia" 37 Software Developer/Software Engineer 400000
10 "Belarus" 22 Data Scientist 10800
... with 144 more rows
```

Kombinacija `group_by()` in `mutate()` je uporabna, kadar želimo ustvariti novo spremenljivko, pri kateri bomo pri izračunu potrebovali kak povzetek vrednosti znotraj posamezne skupine. Primer takšne transformacije je na primer standardiziranje znotraj skupine. Standardna ocena je:

$$z_i = \frac{x_i - \bar{x}}{S_x},$$

kjer je  $\bar{x}$  povprečje vektorja  $x$  in  $S_x$  njegov vzorčni standardni odklon.

Poskusimo za vsako državo izračunati vzorčno povprečno vrednost in standardni odklon, ter s tema vrednostima ustrezno transformirati plačo v USD. Da pa bo funkcija `mutate()` vedela, katere vrednosti naj vzame za računanje teh dveh statistik moramo podatke najprej grupirati glede na državo:

```
ds_jobs %>%
 select(Country, Age, CurrentJobTitle, CompensationUSD) %>%
 group_by(Country) %>%
 mutate(CompensationStand = (CompensationUSD - mean(CompensationUSD)) /
 sd(CompensationUSD))
```

```
A tibble: 3,781 x 5
Groups: Country [53]
Country Age CurrentJobTitle CompensationUSD CompensationSta~
<chr> <int> <chr> <dbl> <dbl>
1 Australia 43 Business Analyst 64185. -0.617
2 Russia 33 Software Developer/Softwa~ 20882. -0.136
3 Taiwan 26 Software Developer/Softwa~ 36634. 0.0566
4 United Sta~ 25 Researcher 20000 -0.0468
5 United Sta~ 33 Scientist/Researcher 100000 -0.0347
6 Czech Repu~ 21 Other 916. -0.907
7 Russia 22 Data Analyst 10859. -0.417
8 Colombia 34 Data Scientist 53352 0.770
9 Germany 41 Data Scientist 179374. 2.35
10 Poland 29 Software Developer/Softwa~ 35419. 0.486
... with 3,771 more rows
```

Če ta tibble shranimo v novo spremenljivko, se bo informacija o združevanju ohranila.

```
ds_jobs_grouped <- ds_jobs %>%
 select(Country, Age, CurrentJobTitle, CompensationUSD) %>%
 group_by(Country, CurrentJobTitle)
ds_jobs_grouped
```

```
A tibble: 3,781 x 4
Groups: Country, CurrentJobTitle [543]
Country Age CurrentJobTitle CompensationUSD
<chr> <int> <chr> <dbl>
1 Australia 43 Business Analyst 64185.
2 Russia 33 Software Developer/Software Engineer 20882.
3 Taiwan 26 Software Developer/Software Engineer 36634.
4 United States 25 Researcher 20000
5 United States 33 Scientist/Researcher 100000
6 Czech Republic 21 Other 916.
7 Russia 22 Data Analyst 10859.
8 Colombia 34 Data Scientist 53352
9 Germany 41 Data Scientist 179374.
10 Poland 29 Software Developer/Software Engineer 35419.
... with 3,771 more rows
```

Opazimo, da ima ta tibble dodatno informacijo v drugi vrstici, ki nam sporoča, da je združen glede na spremenljivki `Country` in `CurrentJobTitle`. Poleg tega je v oglatih oklepajih zapisano število unikatnih skupin. Pri tem so vsi pari države in trenutne pozicije, za katere nimamo nobenega podatka, izpuščeni. Informacija o tem, da je ta tibble grupiran, je pomembna, saj se bodo vse nadaljnje operacije nad njim izvajale nad skupinami. Če tega ne želimo, lahko uporabimo funkcijo `ungroup()`.

```
ds_jobs_ungrouped <- ds_jobs_grouped %>%
 ungroup()
ds_jobs_ungrouped
```

```
A tibble: 3,781 x 4
Country Age CurrentJobTitle CompensationUSD
<chr> <int> <chr> <dbl>
1 Australia 43 Business Analyst 64185.
2 Russia 33 Software Developer/Software Engineer 20882.
3 Taiwan 26 Software Developer/Software Engineer 36634.
4 United States 25 Researcher 20000
5 United States 33 Scientist/Researcher 100000
6 Czech Republic 21 Other 916.
7 Russia 22 Data Analyst 10859.
8 Colombia 34 Data Scientist 53352
9 Germany 41 Data Scientist 179374.
10 Poland 29 Software Developer/Software Engineer 35419.
... with 3,771 more rows
```

## 1.11 Izvajanje operacij nad večimi stolpci z `across()`

S kombinacijo funkcij `mutate()` in `across()` lahko izvajamo isto operacijo hkrati na več stolpcih. Znotraj funkcije `across()` lahko uporabljamo iste funkcije za izbiro kot znotraj `select()`. Spremenimo vrednosti stolpcev, ki se začnejo s `Time`, v deleže tako, da jih pomnožimo z 0.01. Na tem mestu bomo uporabili dva nova operatorja: `.` in `~`. Operator `.` v dplyr igra vlogo podatkov, nad katerimi operiramo. Operator `~` je nekakšna bližnjica, ki ustvari funkcijo. Na primer `~ x^2` je bližnjica za zapis `function(x) {x^2}`. To je uporabno predvsem, ko funkcijo potrebujemo samo na enem mestu znotraj našega poteka dela in jo tako lahko na krajši način zapišemo. Poglejmo si sedaj spremembo stolpcev v deleže:

```
ds_jobs %>%
 mutate(across(starts_with("Time"), ~ . * 0.01)) %>%
 select(Country, CurrentJobTitle, starts_with("Time"))
```

```
A tibble: 3,781 x 8
Country CurrentJobTitle TimeGatheringDa~ TimeModelBuildi~ TimeProduction
<chr> <chr> <dbl> <dbl> <dbl>
1 Australia Business Analyst 0.6 0.1 0.05
2 Russia Software Develop~ 0.4 0.3 0.15
3 Taiwan Software Develop~ 0.35 0.2 0.25
4 United S~ Researcher 0 0.8 0
5 United S~ Scientist/Researc~ 0 0 0
6 Czech Re~ Other 0.2 0.6 0.2
7 Russia Data Analyst 0.5 0.2 0.1
8 Colombia Data Scientist 0.6 0.1 0.2
9 Germany Data Scientist 0.5 0.1 0.2
10 Poland Software Develop~ 0.25 0.2 0.25
... with 3,771 more rows, and 3 more variables: TimeVisualizing <dbl>,
TimeFindingInsights <dbl>, TimeOtherSelect <dbl>
```

Funkciji `across()` smo najprej podali stolpce, na katerih želimo izvajati izračune, nato pa funkcijo, ki jo želimo izvesti.

## 1.12 Povzemanje stolpcev

Pogosto želimo dobiti numerične povzetke glede na vrednosti v stolpcih. Z uporabo osnovne različice R to lahko naredimo s funkcijo `apply()`, ki ji podamo tibble numeričnih vrednosti (lahko tudi `data.frame` ali matriko), določimo dimenzijo 2, ki predstavlja stolpce, ter podamo kateri povzetek želimo (na primer povprečje, varianco, maksimalno vrednost, ...). Izračunajmo povprečja in standardne odklone stolpcev, ki se začnejo s `Time`:

```
ds_jobs_times <- ds_jobs %>%
 select(starts_with("Time"))
apply(ds_jobs_times, 2, mean, na.rm = T)
```

```
TimeGatheringData TimeModelBuilding TimeProduction TimeVisualizing
37.341973 21.085472 11.172853 14.190924
TimeFindingInsights TimeOtherSelect
13.375298 2.202176
```

```
apply(ds_jobs_times, 2, sd, na.rm = T)
```

```
TimeGatheringData TimeModelBuilding TimeProduction TimeVisualizing
20.96041 15.19101 12.03243 10.99431
TimeFindingInsights TimeOtherSelect
12.01139 11.18898
```

`apply()` nam v teh primerih vrne vektor, čeprav smo operacijo izvajali na tibble. Ideja paketa `tidyverse` je, da so izhodni podatki enakega tipa kot vhodni – v tem primeru tibble. Če želimo izračunati povzetke za vsak stolpec, lahko v paketu `dplyr` uporabimo kombinacijo funkcije `summarise()` in `across()`. Kot smo že spoznali, nam funkcija `across()` omogoča izvajanje operacij nad večimi stolpci.

```
ds_jobs %>%
 summarise(across(starts_with("Time"), mean, na.rm = T))

A tibble: 1 x 6
TimeGatheringData TimeModelBuilding TimeProduction TimeVisualizing
<dbl> <dbl> <dbl> <dbl>
1 37.3 21.1 11.2 14.2
... with 2 more variables: TimeFindingInsights <dbl>, TimeOtherSelect <dbl>
```

Povzetke lahko enostavno izračunamo tudi za različne skupine z uporabo funkcije `group_by()`:

```
ds_jobs %>%
 group_by(EmploymentStatus) %>%
 summarise(across(starts_with("Time"), mean, na.rm = T))

A tibble: 3 x 7
EmploymentStatus TimeGatheringDa~ TimeModelBuildi~ TimeProduction
<chr> <dbl> <dbl> <dbl>
1 Employed full-time 37.6 20.6 11.0
2 Employed part-time 37.4 26.1 10.3
3 Independent contractor, free~ 34.8 23.0 12.8
... with 3 more variables: TimeVisualizing <dbl>, TimeFindingInsights <dbl>,
TimeOtherSelect <dbl>
```

### 1.13 Povzemanje vrstic

Kadar analiziramo podatke, preverimo, ali so vnešeni podatki smiselni. Na primer, v stolpcih, ki se začnejo s `Time`, so odstotkovne vrednosti časa, ki ga anketiranci porabijo za posamezne naloge. Te bi se morale sešteti v 100 in v primeru, ko se ne, se lahko odločimo, da takšne vrstice izbrišemo. Na tem primeru si bomo sedaj pogledali še operacije nad stolpci. Naš cilj bo, da dodamo temu tibblu še en stolpec, v katerem bomo sešteli vse te stolpce.

Funkcija `apply` deluje tudi nad stolpci, če spremenimo drugi argument:



```
tmp <- ds_jobs %>%
 select(starts_with("Time"))
head(apply(tmp, 1, sum, na.rm = T))
```

```
[1] 100 100 100 100 0 100
```

Kako pa to naredimo z dplyr, tako da se bo naravno vključilo v potek dela? Prva ideja bi morda bila, da enostavno naštejemo vse stolpce.

```
ds_jobs %>%
 select(Country, CurrentJobTitle, starts_with("Time")) %>%
 mutate(TotalTime = TimeGatheringData + TimeModelBuilding + TimeProduction +
 TimeVisualizing + TimeFindingInsights + TimeOtherSelect) %>%
 select(!starts_with("Time")) # Ta vrstica je samo za lepši izpis.
```

```
A tibble: 3,781 x 3
Country CurrentJobTitle TotalTime
<chr> <chr> <dbl>
1 Australia Business Analyst 100
2 Russia Software Developer/Software Engineer 100
3 Taiwan Software Developer/Software Engineer 100
4 United States Researcher 100
5 United States Scientist/Researcher 0
6 Czech Republic Other 100
7 Russia Data Analyst 100
8 Colombia Data Scientist 100
9 Germany Data Scientist 100
10 Poland Software Developer/Software Engineer 100
... with 3,771 more rows
```

Sicer je to v našem primeru bilo izvedljivo, saj smo imeli samo 6 stolpcev. Kako pa bi to naredili z večimi stolpci? Morda lahko uporabimo `starts_with()`:

```
ds_jobs %>%
 select(Country, CurrentJobTitle, starts_with("Time")) %>%
 mutate(TimeTotal = sum(starts_with("Time"), na.rm = T))
```

```
Error: Problem with `mutate()` column `TimeTotal`.
i `TimeTotal = sum(starts_with("Time"), na.rm = T)`.
x `starts_with()` must be used within a *selecting* function.
i See <https://tidyselect.r-lib.org/reference/faq-selection-context.html>.
```

R vrne napako in nas opozori, da se lahko `starts_with()` uporabi le znotraj izbire. Če želimo v tem primeru omogočiti urejeno izbiro stolpcev, uporabimo funkcijo `c_across()`. Ta funkcija je po funkcionalnosti bolj podobna funkciji `c()` ali `select()`, kot pa funkciji `across()`, tako da jih ne smemo zamenjati:

```
ds_jobs %>%
 select(Country, CurrentJobTitle, starts_with("Time")) %>%
 mutate(TotalTime = sum(c_across(starts_with("Time")), na.rm = T)) %>%
 select(!starts_with("Time"))
```

```
A tibble: 3,781 x 3
Country CurrentJobTitle TotalTime
<chr> <chr> <dbl>
1 Australia Business Analyst 375462
2 Russia Software Developer/Software Engineer 375462
3 Taiwan Software Developer/Software Engineer 375462
4 United States Researcher 375462
5 United States Scientist/Researcher 375462
6 Czech Republic Other 375462
7 Russia Data Analyst 375462
8 Colombia Data Scientist 375462
9 Germany Data Scientist 375462
10 Poland Software Developer/Software Engineer 375462
... with 3,771 more rows
```

Sedaj smo dobili nek rezultat, ki pa še vedno ni pravilen. V čem je težava? Če `sum()` uporabimo znotraj `mutate()`, ta vrne vsoto znotraj skupin, določenih z `group_by()`. Ker podatkov nismo grupirali, vrne vsoto kar čez celotne podatke. Rešitev se torej skriva v ustreznem združevanju vrstic. V `dplyr` obstaja funkcija, ki celoten tibble grupira po posameznih vrsticah in to je `rowwise()`:

```
ds_jobs %>%
 select(Country, CurrentJobTitle, starts_with("Time")) %>%
 rowwise() %>%
 mutate(TotalTime = sum(c_across(starts_with("Time")), na.rm = T)) %>%
 select(!starts_with("Time"))
```

```
A tibble: 3,781 x 3
Rowwise:
Country CurrentJobTitle TotalTime
<chr> <chr> <dbl>
1 Australia Business Analyst 100
2 Russia Software Developer/Software Engineer 100
3 Taiwan Software Developer/Software Engineer 100
```

```
4 United States Researcher 100
5 United States Scientist/Researcher 0
6 Czech Republic Other 100
7 Russia Data Analyst 100
8 Colombia Data Scientist 100
9 Germany Data Scientist 100
10 Poland Software Developer/Software Engineer 100
... with 3,771 more rows
```

## 1.14 Dodatek

### 1.14.1 Zamenjava vrstnega reda stolpcev

Vrstni red stolpcev zamenjamo s funkcijo `relocate()`. Ustvarimo najprej manjši tibble:

```
ds_jobs_select <- ds_jobs %>%
 select(Gender:Major)
ds_jobs_select
```

```
A tibble: 3,781 x 8
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecomme~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Colombia 34 Employed full-time Data Scientist Python
9 Male Germany 41 Independent contrac~ Data Scientist Python
10 Male Poland 29 Employed full-time Software Develop~ Python
... with 3,771 more rows, and 2 more variables: FormalEducation <chr>,
Major <chr>
```

Če želimo določene stolpce premakniti na začetek, jih enostavno podamo funkciji `relocate()`. Dajmo na prvo mesto stolpca `Major` in `Age`:

```
ds_jobs_select %>%
 relocate(Major, Age)
```

```
A tibble: 3,781 x 8
Major Age Gender Country EmploymentStatus CurrentJobTitle LanguageRecomme~
<chr> <int> <chr> <chr> <chr> <chr> <chr>
1 "" 43 Female Austr~ Employed full-t~ Business Analy~ Python
2 "Othe~ 33 Male Russia Employed full-t~ Software Devel~ Python
3 "Comp~ 26 Male Taiwan Employed full-t~ Software Devel~ Python
4 "Phys~ 25 Male United~ Employed part-t~ Researcher Python
5 "Elec~ 33 Male United~ Employed full-t~ Scientist/Rese~ Matlab
6 "Comp~ 21 Male Czech ~ Employed part-t~ Other Python
7 "Info~ 22 Male Russia Employed full-t~ Data Analyst Python
8 "Comp~ 34 Male Colomb~ Employed full-t~ Data Scientist Python
9 "" 41 Male Germany Independent con~ Data Scientist Python
10 "Comp~ 29 Male Poland Employed full-t~ Software Devel~ Python
... with 3,771 more rows, and 1 more variable: FormalEducation <chr>
```

Poljubno ureditev dobimo tako, da enostavno zapišemo vrstni red stolpcev, kot ga želimo. Funkcija `relocate()` omogoča še nekatere možnosti urejanja, kot na primer, glede na tip spremenljivke. Za več informacij o različnih načinih urejanja stolpcev bralcu predlagamo uporabo pomoči `?relocate`.

### 1.14.2 Preimenovanje stolpcev

Stolpce preimenujemo s funkcijo `rename()`.

```
ds_jobs_select %>%
 rename(employment_status = EmploymentStatus,
 current_job_title = CurrentJobTitle)
```

```
A tibble: 3,781 x 8
Gender Country Age employment_status current_job_title LanguageRecomme~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Colombia 34 Employed full-time Data Scientist Python
9 Male Germany 41 Independent contrac~ Data Scientist Python
10 Male Poland 29 Employed full-time Software Develop~ Python
... with 3,771 more rows, and 2 more variables: FormalEducation <chr>,
Major <chr>
```

Tibble lahko vsebuje tudi imena stolpcev, ki niso veljavna za spremenljivke v R. V tem primeru jih moramo zapisati znotraj ```. Na primer, spremenljivki v R ne moremo prirediti imena z minusom. Poizkusimo to narediti v tibliu:

```
ds_jobs_select %>%
 rename(`employment-status` = EmploymentStatus,
 `current-job-title` = CurrentJobTitle)
```

```
A tibble: 3,781 x 8
Gender Country Age `employment-status` `current-job-tit~ LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Colombia 34 Employed full-time Data Scientist Python
9 Male Germany 41 Independent contrac~ Data Scientist Python
10 Male Poland 29 Employed full-time Software Develop~ Python
... with 3,771 more rows, and 2 more variables: FormalEducation <chr>,
Major <chr>
```

### 1.14.3 Summarise in group unpeeling

Kot smo že spoznali je funkcija `summarise()` najbolj uporabna v kombinaciji z `group_by()`. Poglejmo si sedaj bolj podrobno, kakšen tibble je rezultat te kombinacije. Najprej samo grupirajmo `ds_jobs`:

```
ds_jobs_grouped <- ds_jobs %>%
 group_by(FormalEducation, EmploymentStatus)
ds_jobs_grouped
```

```
A tibble: 3,781 x 19
Groups: FormalEducation, EmploymentStatus [21]
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
```

```
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Colombia 34 Employed full-time Data Scientist Python
9 Male Germany 41 Independent contrac~ Data Scientist Python
10 Male Poland 29 Employed full-time Software Develop~ Python
... with 3,771 more rows, and 13 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>, CompensationUSD <dbl>, MonthlyCompUSD <dbl>
```

V drugi vrstici vidimo, da je ta tibble grupiran po spremenljivkah `FormalEducation` in `EmploymentStatus`. Poglejmo kaj se zgodi, ko uporabimo `summarise()`:

```
ds_jobs_summarised <- ds_jobs_grouped %>%
 summarise(Count = n())
ds_jobs_summarised
```

```
A tibble: 21 x 3
Groups: FormalEducation [8]
FormalEducation EmploymentStatus Count
<chr> <chr> <int>
1 "" Employed full-time 1
2 "Bachelor's degree" Employed full-time 857
3 "Bachelor's degree" Employed part-time 52
4 "Bachelor's degree" Independent contractor, freelanc~ 76
5 "Doctoral degree" Employed full-time 719
6 "Doctoral degree" Employed part-time 26
7 "Doctoral degree" Independent contractor, freelanc~ 50
8 "I did not complete any formal educa~ Employed full-time 13
9 "I did not complete any formal educa~ Employed part-time 2
10 "I did not complete any formal educa~ Independent contractor, freelanc~ 10
... with 11 more rows
```

Opazimo, da je ta novi tibble grupiran samo po spremenljivki `FormalEducation`. Privzeto `summarise()` vedno odstrani zadnje grupiranje. Če tega ne želimo, lahko uporabimo dodaten parameter `.groups = "keep"`.

```
ds_jobs_summarised <- ds_jobs_grouped %>%
 summarise(Count = n(), .groups = "keep")
ds_jobs_summarised
```

```
A tibble: 21 x 3
Groups: FormalEducation, EmploymentStatus [21]
```

```
FormalEducation EmploymentStatus Count
<chr> <chr> <int>
1 "" Employed full-time 1
2 "Bachelor's degree" Employed full-time 857
3 "Bachelor's degree" Employed part-time 52
4 "Bachelor's degree" Independent contractor, freelanc~ 76
5 "Doctoral degree" Employed full-time 719
6 "Doctoral degree" Employed part-time 26
7 "Doctoral degree" Independent contractor, freelanc~ 50
8 "I did not complete any formal educa~ Employed full-time 13
9 "I did not complete any formal educa~ Employed part-time 2
10 "I did not complete any formal educa~ Independent contractor, freelanc~ 10
... with 11 more rows
```

## 1.15 Ali želite izvedeti več?

V tem poglavju smo spoznali temeljne operacije nad podatki in njihovo implementacijo v R paketu dplyr. Opis vseh funkcij v dplyr najdemo tukaj: <https://dplyr.tidyverse.org/reference/index.html>. Jedrnat povzetek pa je na voljo tukaj, pod poglavjem Data Transformation Cheatsheet: <https://www.rstudio.com/resources/cheatsheets/>.

## 1.16 Domača naloga

- 1) Začeli bomo z relativno preprosto nalogo, kjer bomo ponovili osnovne ukaze iz slovnice ravnanja s podatki. Osnovna različica programskega jezika R že vsebuje nekatere podatkovne zbirke. Z ukazom `data()` dobimo opis vseh zbirk. V tej nalogi bomo uporabili podatkovno zbirko `mtcars`:

```
head(mtcars)
```

```
mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

Za podrobnejši opis podatkov uporabite pomoč `?mtcars`. Najprej ustvarite novo spremenljivko `mtcars_tib`, v katero shranite razpredelnico `mtcars` kot tibble. Nato vsako izmed spodnjih nalog izvedite posebej

(torej v vsaki točki izvedite ukaz na `mtcars_tib`, ampak tako spremenjenega tibbla ne shranite nazaj v to spremenljivko), razen če je v nalogi eksplicitno navedeno drugače. Vaše naloge so sledeče:

- Ustvarite novo spremenljivko `mtcars_tib`, v katero shranite razporednico `mtcars` kot tibble.
- Izberite vse vrstice avtomobilov z avtomatskim menjalnikom.

```
A tibble: 19 x 11
mpg cyl disp hp drat wt qsec vs am gear carb
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 21.4 6 258 110 3.08 3.22 19.4 1 0 3 1
2 18.7 8 360 175 3.15 3.44 17.0 0 0 3 2
3 18.1 6 225 105 2.76 3.46 20.2 1 0 3 1
4 14.3 8 360 245 3.21 3.57 15.8 0 0 3 4
5 24.4 4 147. 62 3.69 3.19 20 1 0 4 2
6 22.8 4 141. 95 3.92 3.15 22.9 1 0 4 2
7 19.2 6 168. 123 3.92 3.44 18.3 1 0 4 4
8 17.8 6 168. 123 3.92 3.44 18.9 1 0 4 4
9 16.4 8 276. 180 3.07 4.07 17.4 0 0 3 3
10 17.3 8 276. 180 3.07 3.73 17.6 0 0 3 3
11 15.2 8 276. 180 3.07 3.78 18 0 0 3 3
12 10.4 8 472 205 2.93 5.25 18.0 0 0 3 4
13 10.4 8 460 215 3 5.42 17.8 0 0 3 4
14 14.7 8 440 230 3.23 5.34 17.4 0 0 3 4
15 21.5 4 120. 97 3.7 2.46 20.0 1 0 3 1
16 15.5 8 318 150 2.76 3.52 16.9 0 0 3 2
17 15.2 8 304 150 3.15 3.44 17.3 0 0 3 2
18 13.3 8 350 245 3.73 3.84 15.4 0 0 3 4
19 19.2 8 400 175 3.08 3.84 17.0 0 0 3 2
```

- Izberite vse vrstice, kjer je poraba manjša od 15 galon na miljo ali večja od 20 galon na miljo in je motor oblike V.

```
A tibble: 8 x 11
mpg cyl disp hp drat wt qsec vs am gear carb
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 21 6 160 110 3.9 2.62 16.5 0 1 4 4
2 21 6 160 110 3.9 2.88 17.0 0 1 4 4
3 14.3 8 360 245 3.21 3.57 15.8 0 0 3 4
4 10.4 8 472 205 2.93 5.25 18.0 0 0 3 4
5 10.4 8 460 215 3 5.42 17.8 0 0 3 4
6 14.7 8 440 230 3.23 5.34 17.4 0 0 3 4
7 13.3 8 350 245 3.73 3.84 15.4 0 0 3 4
8 26 4 120. 91 4.43 2.14 16.7 0 1 5 2
```

- Izberite vse stolpce, kjer ime stolpca vsebuje črko a.



```
A tibble: 32 x 4
drat am gear carb
<dbl> <dbl> <dbl> <dbl>
1 3.9 1 4 4
2 3.9 1 4 4
3 3.85 1 4 1
4 3.08 0 3 1
5 3.15 0 3 2
6 2.76 0 3 1
7 3.21 0 3 4
8 3.69 0 4 2
9 3.92 0 4 2
10 3.92 0 4 4
... with 22 more rows
```

- Izberite zadnje 4 stolpce.

```
A tibble: 32 x 4
vs am gear carb
<dbl> <dbl> <dbl> <dbl>
1 0 1 4 4
2 0 1 4 4
3 1 1 4 1
4 1 0 3 1
5 0 0 3 2
6 1 0 3 1
7 0 0 3 4
8 1 0 4 2
9 1 0 4 2
10 1 0 4 4
... with 22 more rows
```

- V tibble `mtcars_tib` dodajte stolpca, kjer bosta izračunani število litrov na 100 kilometrov in teža v kilogramih (v tisočicah). 1 milja je približno 1.61 kilometra, 1 galona 3.79 litra in 1 funt 0.45 kilograma.

```
A tibble: 32 x 4
mpg wt lp100km wt_in_kg
<dbl> <dbl> <dbl> <dbl>
1 21 2.62 11.2 1.18
2 21 2.88 11.2 1.29
3 22.8 2.32 10.3 1.04
4 21.4 3.22 11.0 1.45
5 18.7 3.44 12.6 1.55
6 18.1 3.46 13.0 1.56
7 14.3 3.57 16.5 1.61
```

```
8 24.4 3.19 9.65 1.44
9 22.8 3.15 10.3 1.42
10 19.2 3.44 12.3 1.55
... with 22 more rows
```

- Izračunajte povprečno porabo avtomobilov v odvisnosti števila cilindrov.

```
A tibble: 3 x 2
cyl mean_mpg
<dbl> <dbl>
1 4 26.7
2 6 19.7
3 8 15.1
```

- Izračunajte povprečno konjsko moč v odvisnosti od oblike motorja in ali je avtomobil avtomatik ali ne.

```
A tibble: 4 x 3
Groups: vs [2]
vs am mean_hp
<dbl> <dbl> <dbl>
1 0 0 194.
2 0 1 181.
3 1 0 102.
4 1 1 80.6
```

- Normalizirajte vse stolpce, ki vsebujejo decimalna števila, na interval  $[0, 1]$ . To naredimo tako, da vrednostim odštejemo minimalno vrednost in delimo z razliko med maksimalno in minimalno vrednostjo:

$$x'_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}.$$

```
A tibble: 32 x 13
mpg cyl disp hp drat wt qsec vs am gear carb lp100km
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 0.451 6 0.222 110 0.525 0.283 0.233 0 1 4 4 0.272
2 0.451 6 0.222 110 0.525 0.348 0.3 0 1 4 4 0.272
3 0.528 4 0.0920 93 0.502 0.206 0.489 1 1 4 1 0.215
4 0.468 6 0.466 110 0.147 0.435 0.588 1 0 3 1 0.259
5 0.353 8 0.721 175 0.180 0.493 0.3 0 0 3 2 0.360
6 0.328 6 0.384 105 0 0.498 0.681 1 0 3 1 0.386
7 0.166 8 0.721 245 0.207 0.526 0.160 0 0 3 4 0.607
8 0.596 4 0.189 62 0.429 0.429 0.655 1 0 4 2 0.172
9 0.528 4 0.174 95 0.535 0.419 1 1 0 4 2 0.215
10 0.374 6 0.241 123 0.535 0.493 0.452 1 0 4 4 0.339
... with 22 more rows, and 1 more variable: wt_in_kg <dbl>
```

- Izračunajte povprečne vrednosti vseh stolpcev.

```
A tibble: 1 x 13
mpg cyl disp hp drat wt qsec vs am gear carb lp100km
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 20.1 6.19 231. 147. 3.60 3.22 17.8 0.438 0.406 3.69 2.81 12.8
... with 1 more variable: wt_in_kg <dbl>
```

- 2) **Težja naloga.** V mapi *data-raw* se nahajajo podatki o gozdnih požarih na Portugalskem. Podatki so bili uporabljeni v znanstvenem članku (Cortez and Morais, 2007), kjer so napovedovali velikost požganega območja v odvisnosti od meteoroloških in drugih podatkov. Vrednosti 0 za požgano območje predstavljajo požare, kjer je pogorelo manj kot 100 kvadratnih metrov.

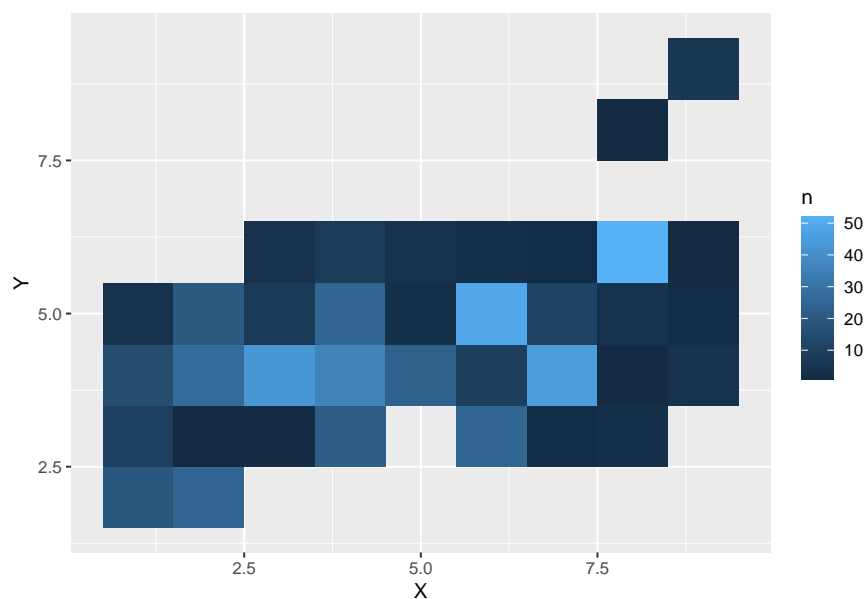
- Preberite podatke in jih shranite kot tibble.
- Preverite, v katerem mesecu je največ požarov in jih uredite padajoče od tistega z največ požari do tistega z najmanj.

```
A tibble: 12 x 2
month n
<chr> <int>
1 aug 184
2 sep 172
3 mar 54
4 jul 32
5 feb 20
6 jun 17
7 oct 15
8 apr 9
9 dec 9
10 jan 2
11 may 2
12 nov 1
```

- Preverite, ali obstajajo območja v parku, kjer se bolj pogosto pojavljajo požari. Za vsako kombinacijo koordinat bomo torej izračunali število požarov. Rezultat lahko predstavimo z razpredelnico. Glede na to, da imamo dvodimenzionalne podatke, bi jih morda bilo smiselno predstaviti vizualno. V kolikor poznate paket ggplot2, predlagamo da si pogledate funkcijo `geom_tile()`.

```
A tibble: 36 x 3
X Y n
<dbl> <dbl> <int>
1 1 2 19
```

```
2 1 3 10
3 1 4 15
4 1 5 4
5 2 2 25
6 2 3 1
7 2 4 27
8 2 5 20
9 3 3 1
10 3 4 43
... with 26 more rows
```



- Dodajte stolpec, ki bo za vsak požar izračunal delež požganega območja glede na vse požare na posameznih koordinatah. Za tem smiselno filtrirajte podatke (ali smo v novem stolpcu dobili kakšne nepričakovane, oziroma neveljavne vrednosti?).

```
A tibble: 509 x 5
X Y month day area_by_coord
<dbl> <dbl> <chr> <chr> <dbl>
1 7 5 mar fri 0
2 7 4 oct tue 0
3 7 4 oct sat 0
4 8 6 mar fri 0
5 8 6 mar sun 0
6 8 6 aug sun 0
7 8 6 aug mon 0
```

```
8 8 6 aug mon 0
9 8 6 sep tue 0
10 7 5 sep sat 0
... with 499 more rows
```

- Preverite, ali ob vročem vremenu in nizki vlažnosti pogori večji delež območja, ki smo ga izračunali v prejšnji točki, tako da izberete vrstice, kjer je temperatura višja od 0.8 kvantila temperature in vlažnost nižja od 0.2 kvantila vlažnosti ter izračunate povprečje.  $q$ -ti kvantil je ocena števila, za katerega velja, da je  $q$  vrednosti manjših od tega števila. Za računanje kvantilov uporabite funkcijo `quantile()`. Za primerjavo izračunajte še povprečje te spremenljivke za vse preostale vrstice. Ali se rezultati skladajo z vašo intuicijo?

```
A tibble: 1 x 1
mean_area_by_coord
<dbl>
1 0.153
```

```
A tibble: 1 x 1
mean_area_by_coord
<dbl>
1 0.0555
```

- Izračunajte povprečje standardiziranih indeksov in ga vstavite kot stolpec pred prvo spremenljivko, ki predstavlja indeks.

```
A tibble: 509 x 15
Rowwise:
X Y month day mean_indices FPMC_index DMC_index DC_index ISI_index
<dbl> <dbl> <chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
1 7 5 mar fri -1.21 -0.796 -1.32 -1.86 -0.859
2 7 4 oct tue -0.304 -0.00502 -1.18 0.480 -0.510
3 7 4 oct sat -0.254 -0.00502 -1.05 0.552 -0.510
4 8 6 mar fri -0.739 0.193 -1.21 -1.92 -0.00890
5 8 6 mar sun -0.719 -0.239 -0.934 -1.82 0.122
6 8 6 aug sun 0.218 0.301 -0.405 -0.256 1.23
7 8 6 aug mon -0.0979 0.301 -0.349 -0.225 -0.118
8 8 6 aug mon 0.320 0.157 0.530 0.232 0.361
9 8 6 sep tue 0.120 0.0669 0.283 0.575 -0.445
10 7 5 sep sat 0.0375 0.337 -0.363 0.599 -0.423
... with 499 more rows, and 6 more variables: temp <dbl>, RH <dbl>,
wind <dbl>, rain <dbl>, area <dbl>, area_by_coord <dbl>
```



## Chapter 2

# Urejeni in relacijski podatki

Pri kakršnemkoli delu smo običajno zelo ciljno naravnani. Dobimo nalogo in jo želimo čimprej in čimbolje opraviti. Z namenom učinkovitosti se običajno poslužimo znanih orodij in postopkov, ki jih prilagodimo samemu problemu. Pri delu s podatki se ciljna naravnost običajno izrazi tako, da želimo čimprej priti do analize in zaključkov, samemu urejanju podatkov pa ne posvetimo pretirane pozornosti, oziroma le toliko, kot je nujno potrebno (kar je še vedno lahko dolgotrajno). Na kratek rok to deluje v redu in celo prihranimo nekaj časa. Na dolgi rok pa običajno zahteva veliko več časa, saj se moramo pri vsaki novi nalogi na novo prilagajati podatkom. Boljši pristop bi bil, da bi problem prilagodili ustaljenemu postopku. Običajno lahko večino podatkov uredimo do te mere, da so si po obliki podobni. V kolikor se naučimo narediti to za neko splošno podatkovno zbirko, lahko potem vsakič pristopimo do nadaljnjega dela na podoben način. V praksi s tem na dolgi rok prihranimo veliko časa.

Koncept **urejenih podatkov** (ang. **tidy data**), ki ga bomo spoznali v tem poglavju, je formalizacija intuitivne predstave kaj podatki so. Podatke, ki so **urejeni**, lahko veliko lažje transformiramo in pripravimo na nadaljnjo analizo. Tudi funkcije v tidyverse so implementirane tako, da na vhod prejmejo urejene podatke in takšne tudi vrnejo. Z drugimi besedami ohranjajo urejenost.

Relacijski podatki pa so podatki o različnih entitetah (na primer podjetje, delavec, službeno vozilo, klient), ki so shranjeni v različnih razpredelnicah. Kadar želimo analizirati relacijske podatke moramo razumeti povezave med njimi in kako delati z njimi. Spoznali bomo koncept relacijskih podatkovnih zbirk in kako uporabiti tidyverse za delo z njimi.

## 2.1 Priprava

V tem poglavju bomo spoznali, kako podatke pretvorimo iz daljše v krajšo obliko (in obratno) ter kako delamo z relacijskimi podatki. Kaj vsi ti koncepti pomenijo in kako so povezani z urejenimi podatki, bomo predelali v jedru poglavja.

Pri pretvorbi podatkov v daljšo obliko gre za pretvorbo, kjer vrednosti večih stolpcev združimo v en stolpec. Poglejmo si razpredelnico, kjer imamo shranjene podatke za več let:

```
df <- tibble(
 ime = c("Mojca", "Miha", "Mateja"),
 `2018` = c(5.5, 4.6, 8.7),
 `2019` = c(5.8, 4.2, 9)
)
df
```

```
A tibble: 3 x 3
ime `2018` `2019`
<chr> <dbl> <dbl>
1 Mojca 5.5 5.8
2 Miha 4.6 4.2
3 Mateja 8.7 9
```

Recimo, da želimo stolpca z leti spraviti v en stolpec. Uporabimo funkcijo `pivot_longer()`.

```
df_longer <- pivot_longer(df, c(`2018`, `2019`), names_to = "leto", values_to = "rezultat")
df_longer
```

```
A tibble: 6 x 3
ime leto rezultat
<chr> <chr> <dbl>
1 Mojca 2018 5.5
2 Mojca 2019 5.8
3 Miha 2018 4.6
4 Miha 2019 4.2
5 Mateja 2018 8.7
6 Mateja 2019 9
```

Lahko naredimo tudi obratno transformacijo, torej da vrednosti enega stolpca razširimo v več stolpcev. Na primer, razširimo stolpec `ime`:



```
pivot_wider(df_longer, names_from = ime, values_from = rezultat)
```

```
A tibble: 2 x 4
leto Mojca Miha Mateja
<chr> <dbl> <dbl> <dbl>
1 2018 5.5 4.6 8.7
2 2019 5.8 4.2 9
```

**Naloga:** Spodnjo razpredelnico transformirajte v daljšo obliko, tako da informacije o številu oddelkov shranite v 1 stolpec.

```
df <- tibble(
 podjetje = c("Podjetje A", "Podjetje A", "Podjetje B"),
 kraj_tovarne = c("Koper", "Kranj", "Koper"),
 prihodek = c(100000, 120000, 60000),
 razvojni_oddelki = c(2, 3, 1),
 prodajni_oddelki = c(3, 3, 2)
)
df
```

```
A tibble: 3 x 5
podjetje kraj_tovarne prihodek razvojni_oddelki prodajni_oddelki
<chr> <chr> <dbl> <dbl> <dbl>
1 Podjetje A Koper 100000 2 3
2 Podjetje A Kranj 120000 3 3
3 Podjetje B Koper 60000 1 2
```

Rešitev:

```
A tibble: 6 x 5
podjetje kraj_tovarne prihodek oddelek stevilo_oddelkov
<chr> <chr> <dbl> <chr> <dbl>
1 Podjetje A Koper 100000 razvojni_oddelki 2
2 Podjetje A Koper 100000 prodajni_oddelki 3
3 Podjetje A Kranj 120000 razvojni_oddelki 3
4 Podjetje A Kranj 120000 prodajni_oddelki 3
5 Podjetje B Koper 60000 razvojni_oddelki 1
6 Podjetje B Koper 60000 prodajni_oddelki 2
```

Spoznali bomo tudi relacijske podatke, pri katerih so podatki razdeljeni med več razpredelnic. Zato bomo potrebovali več funkcij, ki nam omogočajo združevanje teh razpredelnic. Poglejmo si dve razpredelnici:

```

ekipe <- tibble(
 id_ekipe = c(1, 2, 3, 4),
 ekipa = c("Liverpool", "Manchester United", "Arsenal", "Rokova ekipa")
)
igralci <- tibble(
 id_igralca = c(1, 2, 3, 4, 5, 6, 7),
 ime = c("Henderson", "Fernandes", "Alisson", "Rashford", "Novak", "Aubameyang", "Vega"),
 id_ekipe = c(1, 2, 1, 2, 8, 3, 8)
)
ekipe

```

```

A tibble: 4 x 2
id_ekipe ekipa
<dbl> <chr>
1 1 Liverpool
2 2 Manchester United
3 3 Arsenal
4 4 Rokova ekipa

```

```
igralci
```

```

A tibble: 7 x 3
id_igralca ime id_ekipe
<dbl> <chr> <dbl>
1 1 Henderson 1
2 2 Fernandes 2
3 3 Alisson 1
4 4 Rashford 2
5 5 Novak 8
6 6 Aubameyang 3
7 7 Vega 8

```

Za združevanje razpredelnic obstaja več funkcij, vse imajo končnico `_join`. Poglejmo si, kako jih kličemo in kaj vsaka izmed njih naredi. Več bomo o njih povedali na predavanju.

`left_join()` združi razpredelnici tako, da obdrži vse primere iz prve razpredelnice:

```
left_join(igralci, ekipe, by = "id_ekipe")
```

```

A tibble: 7 x 4
id_igralca ime id_ekipe ekipa
<dbl> <chr> <dbl> <chr>

```

```
1 1 Henderson 1 Liverpool
2 2 Fernandes 2 Manchester United
3 3 Alisson 1 Liverpool
4 4 Rashford 2 Manchester United
5 5 Novak 8 <NA>
6 6 Aubameyang 3 Arsenal
7 7 Vega 8 <NA>
```

`right_join()` združi razpredelnici tako, da obdrži vse primere iz druge razpredelnice:

```
right_join(igralci, ekipe, by = "id_ekipe")
```

```
A tibble: 6 x 4
id_igralca ime id_ekipe ekipa
<dbl> <chr> <dbl> <chr>
1 1 Henderson 1 Liverpool
2 2 Fernandes 2 Manchester United
3 3 Alisson 1 Liverpool
4 4 Rashford 2 Manchester United
5 6 Aubameyang 3 Arsenal
6 NA <NA> 4 Rokova ekipa
```

`inner_join()` združi razpredelnici tako, da obdrži samo primere, ki se pojavijo v obeh razpredelnicah:

```
inner_join(igralci, ekipe, by = "id_ekipe")
```

```
A tibble: 5 x 4
id_igralca ime id_ekipe ekipa
<dbl> <chr> <dbl> <chr>
1 1 Henderson 1 Liverpool
2 2 Fernandes 2 Manchester United
3 3 Alisson 1 Liverpool
4 4 Rashford 2 Manchester United
5 6 Aubameyang 3 Arsenal
```

`full_join()` združi razpredelnici tako, da obdrži vse primere iz obeh razpredelnic:

```
full_join(igralci, ekipe, by = "id_ekipe")
```

```
A tibble: 8 x 4
id_igralca ime id_ekipe ekipa
<dbl> <chr> <dbl> <chr>
1 1 Henderson 1 Liverpool
2 2 Fernandes 2 Manchester United
3 3 Alisson 1 Liverpool
4 4 Rashford 2 Manchester United
5 5 Novak 8 <NA>
6 6 Aubameyang 3 Arsenal
7 7 Vega 8 <NA>
8 NA <NA> 4 Rokova ekipa
```

**Naloga:** Obstajata še dve operaciji združevanja, ki pa ne delujeta popolnoma enako kot zgornje funkcije. Pokličite funkciji `semi_join()` in `anti_join()` in poizkusite ugotoviti, kaj sta ti funkciji naredili. Sintaksa je enaka kot pri ostalih funkcijah `join`.

Za branje podatkov iz tekstovnih datotek velikokrat uporabljamo funkcijo `read.csv()` ali katero od preostalih izpeljank funkcije `read.table()`. Tidyverse ima svojo različico teh funkcij, ki pa imajo nekaj dodatne funkcionalnosti. Najbolj pomembna je ta, da se podatki samodejno shranijo kot tibble. To omogoča relativno enostavno branje datotek, kjer stolpci niso poimenovani v skladu s pravili programskega jezika R (na primer, lahko se začnejo s številom, lahko imajo minuse, presledke in podobno). Kot smo omenili shranjevanje podatkov, kjer imena stolpcev niso standardne oblike, ni dobra praksa. Vsekakor pa se pri realnih podatkih velikokrat zgodi, da imamo takšna imena. V tem primeru je bolje, da jih prebermo takšna kot so in jih programsko spremenimo, saj s tem ne posegamo v izvirne podatke. Je pa potrebno pri teh funkcijah dodatno nastaviti kodiranje, da znajo prebrati šumnike. Poglejmo si uporabo funkcije `read_csv2()` paketa `readr`, kjer bomo ustrezno nastavili kodiranje.

```
df <- read_csv2("./data-raw/SLO-gradbena-dovoljenja-messy1.csv",
 locale = readr::locale(encoding = "cp1250"))
```

Več o kodiranjih bomo povedali na zadnjem predavanju.

## 2.2 Urejeni podatki

Omenili smo že, da se v praksi srečamo z najrazličnejšimi oblikami zapisov podatkov. Skupek paketov tidyverse je namenjen delu s tako imenovanimi **urejenimi podatki** (ang. **tidy data**). Ideja je, da se ustvari enoten standard za obliko podatkov, s katero je lažje delati. V kolikor se držimo tega standarda pri vseh naših analizah, nam to omogoča, da vedno uporabljamo ista orodja (na primer, `ggplot2`) in se nam ni potrebno učiti novih orodij za vsako analizo. Standard lahko povzamemo s 3 lastnostmi:

- 1) vsak stolpec je spremenljivka,
- 2) vsaka vrstica je primer podatka,
- 3) vsaka vrednost ima svojo celico.

Morda se na tej točki to sliši nekoliko abstraktno. Poglejmo si praktičen primer. Nabrali smo podatke o številu izdanih gradbenih dovoljenj v Sloveniji, razdeljeno glede na občine. Podatke smo prenesli s spletne strani statističnega urada Slovenije <https://pxweb.stat.si/SiStat/slshranili> in jih shranili na več načinov. Najprej si pogledjmo podatke v takšni obliki, kot smo jih dobili naravnost iz vira.

```
A tibble: 424 x 16
OBČINE TIP.STAVBE `2007` `2008` `2009` `2010` `2011` `2012` `2013` `2014`
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Ajdovšč~ Stanovanjsk~ 52 55 45 33 52 40 29 30
2 Ajdovšč~ Nestanovanj~ 19 9 22 15 27 11 23 11
3 Ankaran~ Stanovanjsk~ NA NA NA NA NA NA NA NA
4 Ankaran~ Nestanovanj~ NA NA NA NA NA NA NA NA
5 Apače Stanovanjsk~ 10 11 22 12 7 5 9 10
6 Apače Nestanovanj~ 3 3 8 3 4 6 2 3
7 Beltinci Stanovanjsk~ 16 19 11 15 19 14 5 13
8 Beltinci Nestanovanj~ 4 6 1 3 8 4 4 5
9 Benedikt Stanovanjsk~ 11 12 6 9 7 3 16 10
##10 Benedikt Nestanovanj~ 3 2 1 3 5 3 4 3
... with 414 more rows, and 6 more variables: 2015 <dbl>, 2016 <dbl>,
2017 <dbl>, 2018 <dbl>, 2019 <dbl>, 2020 <dbl>
```

Najprej imamo na voljo spremenljivki `OBČINE` in `TIP.STAVBE`. Potem pa imamo za vsako leto našete vrednosti, oziroma števila gradbenih dovoljenj. Podatki so velikokrat shranjeni v takšnem formatu, saj ima določene prednosti. Tak format je bolj prijazen za prikaz človeku, saj lahko samo s pogledom na razpredelnico hitro oceni, ali obstaja trend v posamezni vrstici. Format pa ni najboljši za delo s podatki. Govorili smo že o čistih podatkih in da vse funkcije v `tidyverse` podpirajo operacije nad takšnimi podatki. Kot vhod bo večina teh funkcij prejela čiste podatke in takšne potem tudi vrnila.

Kaj je razlog, da ti podatki niso čisti? Ne drži, da imamo v vsakem stolpcu spremenljivko, saj imamo eno spremenljivko razvlečeno čez več stolpcev – leto. Ta podatek vsekakor predstavlja spremenljivko, torej bi moral imeti enoten stolpec. Poglejmo si te podatke še v dveh nečistih formatih.

```
A tibble: 28 x 214
TIP.STAVBE Leto Ajdovščina `Ankaran/Ancaran~ Apače Beltinci Benedikt
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Stanovanjske stav~ 2007 52 NA 10 16 11
2 Stanovanjske stav~ 2008 55 NA 11 19 12
```

```
3 Stanovanjske stav~ 2009 45 NA 22 11 6
4 Stanovanjske stav~ 2010 33 NA 12 15 9
5 Stanovanjske stav~ 2011 52 NA 7 19 7
6 Stanovanjske stav~ 2012 40 NA 5 14 3
7 Stanovanjske stav~ 2013 29 NA 9 5 16
8 Stanovanjske stav~ 2014 30 NA 10 13 10
9 Stanovanjske stav~ 2015 38 3 12 23 13
10 Stanovanjske stav~ 2016 31 1 10 22 15
... with 18 more rows, and 207 more variables: Bistrica ob Sotli <dbl>,
Bled <dbl>, Bloke <dbl>, Bohinj <dbl>, Borovnica <dbl>, Bovec <dbl>,
Braslovče <dbl>, Brda <dbl>, Brezovica <dbl>, Brežice <dbl>, Cankova <dbl>,
Celje <dbl>, Cerklje na Gorenjskem <dbl>, Cerknica <dbl>, Cerkno <dbl>,
Cerkvenjak <dbl>, Cirkulane <dbl>, Črenšovci <dbl>, Črna na Koroškem <dbl>,
Črnomelj <dbl>, Destrnik <dbl>, Divača <dbl>, Dobje <dbl>,
Dobropolje <dbl>, Dobrna <dbl>, Dobrova - Polhov Gradec <dbl>,
Dobrovnik/Dobronak <dbl>, Dol pri Ljubljani <dbl>, Dolenjske Toplice <dbl>,
Domžale <dbl>, Dornava <dbl>, Dravograd <dbl>, Duplek <dbl>,
Gorenja vas - Poljane <dbl>, Gorišnica <dbl>, Gorje <dbl>,
Gornja Radgona <dbl>, Gornji Grad <dbl>, Gornji Petrovci <dbl>, Grad <dbl>,
Grosuplje <dbl>, Hajdina <dbl>, Hoče - Slivnica <dbl>, Hodoš/Hodos <dbl>,
Horjul <dbl>, Hrastnik <dbl>, Hrpelje - Kozina <dbl>, Idrija <dbl>,
Ig <dbl>, Ilirska Bistrica <dbl>, Ivančna Gorica <dbl>, Izola/Isola <dbl>,
Jesenice <dbl>, Jezersko <dbl>, Juršinci <dbl>, Kamnik <dbl>, Kanal <dbl>,
Kidričevo <dbl>, Kobarid <dbl>, Kobilje <dbl>, Kočevje <dbl>, Komen <dbl>,
Komenda <dbl>, Koper/Capodistria <dbl>, Kostanjevica na Krki <dbl>,
Kostel <dbl>, Kozje <dbl>, Kranj <dbl>, Kranjska Gora <dbl>,
Križevci <dbl>, Krško <dbl>, Kungota <dbl>, Kuzma <dbl>, Laško <dbl>,
Lenart <dbl>, Lendava/Lendva <dbl>, Litija <dbl>, Ljubljana <dbl>,
Ljubno <dbl>, Ljutomer <dbl>, Log - Dragomer <dbl>, Logatec <dbl>,
Loška dolina <dbl>, Loški Potok <dbl>, Lovrenc na Pohorju <dbl>,
Luče <dbl>, Lukovica <dbl>, Majšperk <dbl>, Makole <dbl>, Maribor <dbl>,
Markovci <dbl>, Medvode <dbl>, Mengeš <dbl>, Metlika <dbl>, Mežica <dbl>,
Miklavž na Dravskem polju <dbl>, Miren - Kostanjevica <dbl>, Mirna <dbl>,
Mirna Peč <dbl>, Mislinja <dbl>, ...
```

Sedaj imamo podobno situacijo kot prej – ena spremenljivka je razvlečena preko več stolpcev – v tem primeru je to občina. Kot smo že omenili, so vsaki nečisti podatki nečisti na svoj način. Podatki so popolnoma enaki kot v prejšnjem prikazu, ampak razporednica izgleda popolnoma drugače. Čisti podatki pa imajo samo eno pravilno obliko, torej ne more priti do takšnih dvoumnih prikazov.

Poglejmo si še tretji format:

```
A tibble: 5,936 x 3
OBČINA_TIP Leto Število.gradbenih.dovoljenj
```

```
<chr> <dbl> <dbl>
1 Ajdovščina_Stanovanjske stavbe 2007 52
2 Ajdovščina_Stanovanjske stavbe 2008 55
3 Ajdovščina_Stanovanjske stavbe 2009 45
4 Ajdovščina_Stanovanjske stavbe 2010 33
5 Ajdovščina_Stanovanjske stavbe 2011 52
6 Ajdovščina_Stanovanjske stavbe 2012 40
7 Ajdovščina_Stanovanjske stavbe 2013 29
8 Ajdovščina_Stanovanjske stavbe 2014 30
9 Ajdovščina_Stanovanjske stavbe 2015 38
10 Ajdovščina_Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

Ta je morda nekoliko bližje čistim podatkom, kot prejšnja dva, ampak še vedno ni v popolnoma pravilni obliki. V čem je težava? Dve spremenljivki imamo podani v enem stolpcu – občino in tip. Ker gre za različni spremenljivki, bi bilo dobro, da se pojavita v različnih stolpcih.

Poglejmo si sedaj še čiste podatke:

```
A tibble: 5,936 x 4
OBČINE TIP.STAVBE Leto Število.gradbenih.dovoljenj
<chr> <chr> <dbl> <dbl>
1 Ajdovščina Stanovanjske stavbe 2007 52
2 Ajdovščina Stanovanjske stavbe 2008 55
3 Ajdovščina Stanovanjske stavbe 2009 45
4 Ajdovščina Stanovanjske stavbe 2010 33
5 Ajdovščina Stanovanjske stavbe 2011 52
6 Ajdovščina Stanovanjske stavbe 2012 40
7 Ajdovščina Stanovanjske stavbe 2013 29
8 Ajdovščina Stanovanjske stavbe 2014 30
9 Ajdovščina Stanovanjske stavbe 2015 38
10 Ajdovščina Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

Sedaj ima vsaka spremenljivka (občina, tip in leto) svoj stolpec, zadnji stolpec pa je namenjen vrednostim. V tem poglavju se bomo naučili nečiste podatke spremeniti v čiste.

Čisti podatki imajo običajno več vrstic kot nečisti in jim zato pravimo da so **daljši** (ang. **longer**). Nečisti pa so običajno **širši** (ang. **wider**). Izogibamo se besedam dolgi in široki, saj je ta definicija relativna, se pravi lahko uporabimo transformacijo, ki naredi podatke daljše, ne pa nujno dolge, saj morda obstaja še kakšna operacija, ki jih bo naredila še daljše.

## 2.3 pivot\_longer(): pretvorba v daljšo obliko

Funkcija `pivot_longer()` podatke spremeni v daljšo obliko. Ta transformacija je pri delu s podatki bolj pogosta kot sprememba v širšo. Običajno uporabljamo to transformacijo, ko preurejamo podatke v čiste.

Poglejmo si ponovno nečiste podatke, ki smo jih dobili naravnost iz vira:

```
A tibble: 424 x 16
OBČINE TIP.STAVBE `2007` `2008` `2009` `2010` `2011` `2012` `2013` `2014`
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Ajdovšč~ Stanovanjsk~ 52 55 45 33 52 40 29 30
2 Ajdovšč~ Nestanovanj~ 19 9 22 15 27 11 23 11
3 Ankaran~ Stanovanjsk~ NA NA NA NA NA NA NA NA
4 Ankaran~ Nestanovanj~ NA NA NA NA NA NA NA NA
5 Apače Stanovanjsk~ 10 11 22 12 7 5 9 10
6 Apače Nestanovanj~ 3 3 8 3 4 6 2 3
7 Beltinci Stanovanjsk~ 16 19 11 15 19 14 5 13
8 Beltinci Nestanovanj~ 4 6 1 3 8 4 4 5
9 Benedikt Stanovanjsk~ 11 12 6 9 7 3 16 10
10 Benedikt Nestanovanj~ 3 2 1 3 5 3 4 3
... with 414 more rows, and 6 more variables: 2015 <dbl>, 2016 <dbl>,
2017 <dbl>, 2018 <dbl>, 2019 <dbl>, 2020 <dbl>
```

Sedaj želimo te podatke spremeniti v čisto obliko. Vse stolpce, ki prikazujejo različne vrednosti spremenljivke leto, moramo zapisati v 1 stolpec. Uporabimo funkcijo `pivot_longer()`, ki prejme argumente:

- **data.** Katere podatke želimo spremeniti.
- **cols.** V katerih stolpcih imamo vrednosti spremenljivke, ki jo želimo shraniti v 1 stolpec.

```
df %>% pivot_longer(cols = `2007`:`2020`)
```

```
A tibble: 5,936 x 4
OBČINE TIP.STAVBE name value
<chr> <chr> <chr> <dbl>
1 Ajdovščina Stanovanjske stavbe 2007 52
2 Ajdovščina Stanovanjske stavbe 2008 55
3 Ajdovščina Stanovanjske stavbe 2009 45
4 Ajdovščina Stanovanjske stavbe 2010 33
5 Ajdovščina Stanovanjske stavbe 2011 52
6 Ajdovščina Stanovanjske stavbe 2012 40
7 Ajdovščina Stanovanjske stavbe 2013 29
```



```
8 Ajdovščina Stanovanjske stavbe 2014 30
9 Ajdovščina Stanovanjske stavbe 2015 38
10 Ajdovščina Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

Sedaj imamo leta shranjena v stolpcu, prav tako pa vrednosti. Stolpca sta dobila privzeti meni `name` in `value`. Funkcija `pivot_longer()` pa lahko prejme še več opcijskih argumentov, za nas bosta najbolj pomembna 2:

- `names_to`. Ime stolpca, v katerega bomo shranili spremenljivko.
- `value_to`. Ime stolpca, v katerega bomo shranili vrednosti.

Uporabimo sedaj še ta 2 parametra:

```
df_longer <- df %>% pivot_longer(cols = `2007`:`2020`,
 names_to = "Leto",
 values_to = "Število")
df_longer
```

```
A tibble: 5,936 x 4
OBČINE TIP.STAVBE Leto Število
<chr> <chr> <chr> <dbl>
1 Ajdovščina Stanovanjske stavbe 2007 52
2 Ajdovščina Stanovanjske stavbe 2008 55
3 Ajdovščina Stanovanjske stavbe 2009 45
4 Ajdovščina Stanovanjske stavbe 2010 33
5 Ajdovščina Stanovanjske stavbe 2011 52
6 Ajdovščina Stanovanjske stavbe 2012 40
7 Ajdovščina Stanovanjske stavbe 2013 29
8 Ajdovščina Stanovanjske stavbe 2014 30
9 Ajdovščina Stanovanjske stavbe 2015 38
10 Ajdovščina Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

## 2.4 `pivot_wider()`: pretvorba v širšo obliko

Običajno bo ta transformacija naredila podatke nečiste, vendar s tem ni nič narobe, saj imajo tudi takšni podatki svoje prednosti:

- Podatki v širši obliki so človeku lažje berljivi.
- Nekatera podjetja in področja imajo razvite standarde, v katerih potrebujemo podatke v širši obliki.

- Nekatere metode, predvsem gre tukaj za metode strojnega učenja, delujejo bolje ali izključno s podatki v širši obliki.
- Če želimo podatke pretvoriti v matriko.

Za pretvorbo podatkov v širšo obliko uporabimo funkcijo `pivot_wider()`, ki prejme dva argumenta:

- `names_from`. Ime stolpca, katerega želimo raztegniti v širšo obliko.
- `values_from`. Ime stolpca, v katerem so shranjene vrednosti.

Pretvorimo sedaj `df_longer` v širšo obliko glede na stolpec `TIP.STAVBE`.

```
df_wider <- df_longer %>%
 pivot_wider(names_from = TIP.STAVBE, values_from = Številco)
df_wider[1:14,]
```

```
A tibble: 14 x 4
OBČINE Leto `Stanovanjske stavbe` `Nestanovanjske stavbe`
<chr> <chr> <dbl> <dbl>
1 Ajdovščina 2007 52 19
2 Ajdovščina 2008 55 9
3 Ajdovščina 2009 45 22
4 Ajdovščina 2010 33 15
5 Ajdovščina 2011 52 27
6 Ajdovščina 2012 40 11
7 Ajdovščina 2013 29 23
8 Ajdovščina 2014 30 11
9 Ajdovščina 2015 38 49
10 Ajdovščina 2016 31 66
11 Ajdovščina 2017 33 60
12 Ajdovščina 2018 42 36
13 Ajdovščina 2019 38 39
14 Ajdovščina 2020 42 46
```

S takšnim prikazom lahko relativno hitro opazimo določene trende. Na primer v Ajdovščini se je gradilo veliko več stanovanjskih stavb med leti 2007 in 2014, leta 2015 pa se je očitno začelo graditi več nestanovanjskih stavb, kar bi lahko nakazovalo na gospodarsko rast tega mesta. Za človeka je torej tak prikaz boljši. Vsekakor pa bi v tem primeru raje uporabili vizualizacijo.

## 2.5 separate() in unite(): deljenje in združevanje stolpcev

V uvodu tega poglavja smo prikazali podatke, kjer sta bili dve spremenljivki shranjeni v enem stolpcu. Poglejmo si te podatke še enkrat:

```
df <- read_csv2("./data-raw/SL0-gradbena-dovoljenja-messy2.csv",
 locale = readr::locale(encoding = "cp1250"))
df
```

```
A tibble: 5,936 x 3
OBČINA_TIP Leto Število.gradbenih.dovoljenj
<chr> <dbl> <dbl>
1 Ajdovščina_Stanovanjske stavbe 2007 52
2 Ajdovščina_Stanovanjske stavbe 2008 55
3 Ajdovščina_Stanovanjske stavbe 2009 45
4 Ajdovščina_Stanovanjske stavbe 2010 33
5 Ajdovščina_Stanovanjske stavbe 2011 52
6 Ajdovščina_Stanovanjske stavbe 2012 40
7 Ajdovščina_Stanovanjske stavbe 2013 29
8 Ajdovščina_Stanovanjske stavbe 2014 30
9 Ajdovščina_Stanovanjske stavbe 2015 38
10 Ajdovščina_Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

Včasih se srečamo celo z dvema vrednostima v istem stolpcu. Da ločimo ti spremenljivki na dva stolpca uporabimo funkcijo `separate()`:

```
separate(<podatki>, col = <ime-stolpca>, into = <ime-novih-stolpcev>, sep = <znak-ki-locuje>)
```

Uporabimo sedaj to funkcijo da pretvorimo df v čisto obliko:

```
df_tidy <- df %>%
 separate(col = "OBČINA_TIP", into = c("OBČINA", "TIP"), sep = "_")
df_tidy
```

```
A tibble: 5,936 x 4
OBČINA TIP Leto Število.gradbenih.dovoljenj
<chr> <chr> <dbl> <dbl>
1 Ajdovščina Stanovanjske stavbe 2007 52
2 Ajdovščina Stanovanjske stavbe 2008 55
3 Ajdovščina Stanovanjske stavbe 2009 45
4 Ajdovščina Stanovanjske stavbe 2010 33
```

```
5 Ajdovščina Stanovanjske stavbe 2011 52
6 Ajdovščina Stanovanjske stavbe 2012 40
7 Ajdovščina Stanovanjske stavbe 2013 29
8 Ajdovščina Stanovanjske stavbe 2014 30
9 Ajdovščina Stanovanjske stavbe 2015 38
10 Ajdovščina Stanovanjske stavbe 2016 31
... with 5,926 more rows
```

Obstaja pa tudi obratna operacija `unite()`, ki združi dva stolpca:

```
unite(<podatki>, <stolpec1>, <stolpec2>, ..., sep = <znak-ki-locuje>)
```

Pri tem tri pikice predstavljajo morebitne preostale stolpce, saj jih lahko združimo več.

Za primer si pogledjmo, kako bi v eno spremenljivko shranili podatke o številu stanovanjskih in nestanovanjskih gradbenih dovoljenj. Najprej pretvorimo podatke v širšo obliko glede na tip, potem pa ta nova stolpca združimo s funkcijo `unite()`.

```
df_wider <- df_tidy %>%
 pivot_wider(names_from = TIP, values_from = Število.gradbenih.dovoljenj) %>%
 unite("Stanovanjske/Nestanovanjske",
 "Stanovanjske stavbe",
 "Nestanovanjske stavbe",
 sep = "/")
df_wider
```

```
A tibble: 2,968 x 3
OBČINA Leto `Stanovanjske/Nestanovanjske`
<chr> <dbl> <chr>
1 Ajdovščina 2007 52/19
2 Ajdovščina 2008 55/9
3 Ajdovščina 2009 45/22
4 Ajdovščina 2010 33/15
5 Ajdovščina 2011 52/27
6 Ajdovščina 2012 40/11
7 Ajdovščina 2013 29/23
8 Ajdovščina 2014 30/11
9 Ajdovščina 2015 38/49
10 Ajdovščina 2016 31/66
... with 2,958 more rows
```

## 2.6 Relacijske zbirke podatkov

Pogosto se pri analizi podatkov srečamo z razpredelniciami, ki so med seboj logično povezane. Nekaj primerov:

- V spletni trgovini lahko hranimo podatke v 3 razpredelnicah o produktih, kupcih in nakupih. Razpredelnice so med seboj povezane, na primer razpredelnica o nakupih vsebuje ID kupca in produkta.
- Baze podatkov o filmih, kot je IMDB, imajo na primer podatke o filmih, ocenjevalcih, igralcih in ocenah. Filmi povezujejo vse preostale razpredelnice.
- Biološke podatkovne zbirke lahko imajo razpredelnice atomov, molekul in vezi.
- Pri železniškem omrežju imamo razpredelnice z vlaki, vagoni, železniškimi postajami, prihodi in odhodi.
- Pri nogometu imamo razpredelnice z igralci, klubi in odigranimi tekmami.

Takšnim podatkovnim zbirkam pravimo **relacijske zbirke podatkov**, saj so poleg podatkov v razpredelnicah pomembne tudi relacije oziroma povezave med razpredelniciami. Zaenkrat smo se naučili, kako urejati podatke v eni razpredelnici. Če želimo analizirati relacijske podatke, moramo znati upoštevati tudi povezave med njimi in jih ustrezno združevati. V tem poglavju bomo predelali operacije, ki nam to omogočajo. Morda ste se že srečali z jezikom **SQL**, ki se običajno uporablja za urejanje podatkov v sistemih za **upravljanje relacijskih podatkovnih baz** (ang. relational database management systems, RDBMS). Paket dplyr ima podobno sintakso kot SQL, vendar pa ni popolnoma enaka. Je tudi enostavnejši za uporabo pri analizi podatkov, saj je ustvarjen prav s tem namenom.

## 2.7 Primer: Bančni podatki

V tem poglavju bomo delali s podatki češke banke (<https://data.world/lpetrocelli/czech-financial-dataset-real-anonymized-transactions>, <https://relational.fit.cvut.cz/dataset/Financial>). Gre za realno anonimizirano podatkovno zbirko, ki je bila uporabljena v izzivu PKDD'99 Discovery Challenge (<https://sorry.vse.cz/~berka/challenge/pkdd1999/berka.htm>). Cilj izziva je bil odkriti dobre in slabe kliente z namenom izboljšanja ponudbe. Mi bomo te podatke uporabili za ilustracijo operacij na relacijski zbirki podatkov.

V mapi `data_raw/financial` se nahaja 5 razpredelnic v csv formatu: `account.csv`, `client.csv`, `disp.csv`, `loan.csv` in `transaction-smaller.csv`. Izvirni podatki vsebujejo še nekaj razpredelnic, vendar jih bomo z namenom učinkovitega prikaza izpustili. Prav tako smo pri razpredelnici `transaction.csv` naključno izbrali 20000 vrstic, saj originalna datoteka vsebuje preko milijon vrstic, kar bi upočasnilo

izvajanje ukazov in zasedlo veliko prostora na repozitoriju. V kolikor želite raziskati celotno zbirko, predlagamo, da si podatke prenesete iz vira. Poglejmo si sedaj vsako izmed razpredelnic.

Razpredelnica `account` vsebuje podatke o računih na banki.

```
account <- read_csv2("./data-raw/financial/account.csv")
account
```

```
A tibble: 4,500 x 4
account_id district_id frequency date
<dbl> <dbl> <chr> <date>
1 1 18 monthly payment 1995-03-24
2 2 1 monthly payment 1993-02-26
3 3 5 monthly payment 1997-07-07
4 4 12 monthly payment 1996-02-21
5 5 15 monthly payment 1997-05-30
6 6 51 monthly payment 1994-09-27
7 7 60 monthly payment 1996-11-24
8 8 57 monthly payment 1995-09-21
9 9 70 monthly payment 1993-01-27
10 10 54 monthly payment 1996-08-28
... with 4,490 more rows
```

Razpredelnica `client` vsebuje podatke o strankah.

```
client <- read_csv2("./data-raw/financial/client.csv")
client
```

```
A tibble: 5,369 x 4
client_id gender birth_date district_id
<dbl> <chr> <date> <dbl>
1 1 F 1970-12-13 18
2 2 M 1945-02-04 1
3 3 F 1940-10-09 1
4 4 M 1956-12-01 5
5 5 F 1960-07-03 5
6 6 M 1919-09-22 12
7 7 M 1929-01-25 15
8 8 F 1938-02-21 51
9 9 M 1935-10-16 60
10 10 M 1943-05-01 57
... with 5,359 more rows
```

Razpredelnica `disp` poveže podatke o osebah in računih, torej, katere osebe imajo pravico opravljati s katerimi računi.

```
disp <- read_csv2("./data-raw/financial/disp.csv")
disp
```

```
A tibble: 5,369 x 4
disp_id client_id account_id type
<dbl> <dbl> <dbl> <chr>
1 1 1 1 1 OWNER
2 2 2 2 2 OWNER
3 3 3 3 2 DISPONENT
4 4 4 4 3 OWNER
5 5 5 5 3 DISPONENT
6 6 6 6 4 OWNER
7 7 7 7 5 OWNER
8 8 8 8 6 OWNER
9 9 9 9 7 OWNER
10 10 10 10 8 OWNER
... with 5,359 more rows
```

Razpredelnica `loan` vsebuje podatke o posojilih.

```
loan <- read_csv2("./data-raw/financial/loan.csv")
loan
```

```
A tibble: 682 x 7
loan_id account_id date amount duration payments status
<dbl> <dbl> <date> <dbl> <dbl> <dbl> <chr>
1 4959 2 1994-01-05 80952 24 3373 A
2 4961 19 1996-04-29 30276 12 2523 B
3 4962 25 1997-12-08 30276 12 2523 A
4 4967 37 1998-10-14 318480 60 5308 D
5 4968 38 1998-04-19 110736 48 2307 C
6 4973 67 1996-05-02 165960 24 6915 A
7 4986 97 1997-08-10 102876 12 8573 A
8 4988 103 1997-12-06 265320 36 7370 D
9 4989 105 1998-12-05 352704 48 7348 C
10 4990 110 1997-09-08 162576 36 4516 C
... with 672 more rows
```

Razpredelnica `trans` vsebuje podatke o transakcijah.

```
trans <- read_csv2("./data-raw/financial/transaction-smaller.csv")
trans
```

```
A tibble: 20,000 x 10
trans_id account_id date type operation amount balance k_symbol bank
<dbl> <dbl> <date> <chr> <chr> <dbl> <dbl> <chr> <chr>
1 736882 2517 1997-07-17 CHOICE CHOICE 21992 22279 <NA> <NA>
2 201830 686 1997-05-08 INCOME DEPOSIT 10533 18473 <NA> <NA>
3 3158278 10478 1998-01-29 EXPEN~ CHOICE 2100 8821 <NA> <NA>
4 41116 135 1994-05-09 EXPEN~ CHOICE 2900 21513 <NA> <NA>
5 1046207 3578 1996-09-08 EXPEN~ TRANSFER~ 4051 51755 SIPO KL
6 875501 2982 1997-04-30 EXPEN~ CHOICE 12100 41859 <NA> <NA>
7 893918 3047 1996-11-30 EXPEN~ CHOICE 15 24788 SERVICES <NA>
8 3442751 1543 1998-07-31 INCOME <NA> 71 17153 UROK <NA>
9 462371 1571 1998-08-25 EXPEN~ CHOICE 2760 25770 <NA> <NA>
10 1028586 3513 1993-10-12 EXPEN~ TRANSFER~ 4507 31227 SIPO KL
... with 19,990 more rows, and 1 more variable: account <dbl>
```

Imamo 5 razpredelnice, vse pa so med seboj povezane. Razpredelnici `account` in `client` sta povezani preko razpredelnice `disp`. Razpredelnici `loan` in `trans` sta povezani z razpredelnico `account` preko spremenljivke `account_id`. To strukturo najboljše prikažemo z **relacijskim diagramom**.

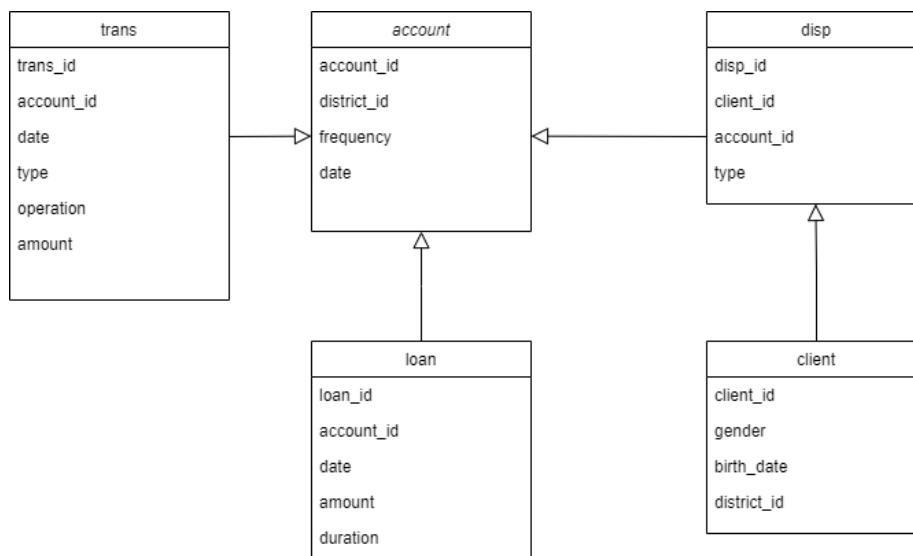


Figure 2.1: Relacijski diagram

## 2.8 Ključi

Spremenljivkam, ki povezujejo razpredelnice, pravimo **ključi**. Te spremenljivke (ali zbirke spremenljivk) edinstveno definirajo podatke. Lahko gre za eno spre-



menljivko, kot je na primer `account_id` v razpredelnici `account`. Lahko pa obstaja več spremenljivk, ki definirajo en podatek. Na primer, če imamo razpredelnico s temperaturami za vsak dan in uro. Potem ni nujno, da ima vsaka vrstica svoj ID, lahko pa jih edinstveno ločimo na podlagi dveh spremenljivk – dneva in ure. V tem primeru gre torej za ključ, ki je sestavljen iz dveh spremenljivk.

Poznamo dva glavna tipa ključev:

- **Primarni ključ.** Ta ključ edinstveno definira podatek v razpredelnici. Na primer, `trans_id` v razpredelnici `trans`. V urejenih podatkih ima vsaka tabela svoj primarni ključ.
- **Tuj ključ.** To je ključ v razpredelnici, ki je primarni ključ v eni od preostalih razpredelnic. Na primer, `account_id` v razpredelnici `trans`. Vrednosti tujih ključev se lahko podvajajo. Na primer, več transakcij lahko ima isto vrednost tujega ključa za `account_id`, saj se na enem bančnem računu izvede več transakcij.

V kolikor razpredelnica nima primarnega ključa, lahko ustvarimo t. i. nadomestni ključ, ki igra vlogo primarnega ključa. To lahko naredimo na primer tako, da vsaki vrstici priredimo njeno zaporedno vrednost v razpredelnici. Na primer `mutate(row_number())`.

Primarni in tuj ključ skupaj tvorita relacijo med razpredelnicama. Na primer `account_id` predstavlja relacijo med razpredelnicama `trans` in `account`. Relacije so lahko ena-proti-ena (ena država ima enega predsednika in ena oseba je lahko predsednik samo ene države), ena-proti-mnogo (en igralec lahko igra za en klub, ampak en klub ima več igralcev) ali mnogo-proti-mnogo (en avtor lahko napiše več knjig in ena knjiga je lahko napisana s strani večih avtorjev).

Kadar imamo opravka z relacijskimi podatki, je smiselno preveriti, ali je primarni ključ res edinstven za vsako razpredelnico.

```
df_list <- list(account, client, disp, trans, loan)
id_vec <- c("account_id", "client_id", "disp_id", "trans_id", "loan_id")
for (i in 1:length(df_list)) {
 tmp <- df_list[[i]] %>%
 group_by(.data[[id_vec[i]]]) %>%
 summarise(n = n()) %>%
 filter(n > 1)
 print(tmp)
}
```

```
A tibble: 0 x 2
... with 2 variables: account_id <dbl>, n <int>
A tibble: 0 x 2
... with 2 variables: client_id <dbl>, n <int>
```

```
A tibble: 0 x 2
... with 2 variables: disp_id <dbl>, n <int>
A tibble: 0 x 2
... with 2 variables: trans_id <dbl>, n <int>
A tibble: 0 x 2
... with 2 variables: loan_id <dbl>, n <int>
```

V prejšnjem klicu kode se pojavi nova sintaksa, in sicer `.data[[id_vec[i]]]`. Funkcija `group_by()` uporablja t. i. *tidyselect*, s katerim izbiramo stolpce brez da bi jih dali v narekovaje. To pa predstavlja težavo, kadar so imena stolpcev shranjena v neki spremenljivki, kot v tem primeru. Tidyverse je ustvarjen na načelu, da olajša bolj pogoste operacije (na primer, enostavno uporaba `group_by()` pri urejanju posamezne razpredelnice), za ceno težje izvedbe manj pogostih operacij (na primer, uporaba `group_by()` v zanki `for`). Veliko večino urejanja podatkov bomo lahko z uporabo tidyverse naredili brez uporabe zank ali naprednih lastnih funkcij. V kolikor se boste lotili bolj programerskega pristopa, pa predlagamo, da si preberete navodila za programiranje z `dplyr`, ki jih dobite tako, da v konzoli kličete `vignette('programming')`. Na tej delavnici ne bomo predstavili podrobnosti teh pristopov. Zaenkrat je dovolj, da poznamo samo zgornji klic. Torej, če imamo imena stolpcev shranjena v neki spremenljivki, potem moramo znotraj `tidyselecta` uporabiti `.data[[<spremenljivka-z-imeni-stolpcev>]]`.

## 2.9 Združevanja

Kadar imamo opravka z večimi razpredelnicami potrebujemo orodja, s katerimi lahko posamezne pare razpredelnic združimo. Vešči uporabniki R morda že poznajo funkcijo `merge`, ki je del osnovne različice R in je namenjena splošnemu združevanju razpredelnic. Seveda pa tidyverse premore svoje različice podobnih funkcij, ki premorejo enake lastnosti kot preostale funkcije v tej zbirki – prejmejo in vrnejo podatke v enakem formatu in sicer tibblu. Poleg tega so funkcije iz paketa `dplyr` tudi hitrejšje od `merge`, kar ima pomembno vlogo, kadar imamo opravka z nekoliko večjimi podatkovnimi množicami.

Združevanja podatkovnih razpredelnic lahko ločimo na 3 sklope:

- **Mutirajoča združevanja** (ang. **mutating joins**). Dodajo nove stolpce k razpredelnici iz ujemajočih vrstic druge razpredelnice.
- **Filtrirajoča združevanja** (ang. **filtering joins**). Izberejo vrstice ene razpredelnice glede na to, ali se te ujemajo z vrsticami v drugi razpredelnici.
- **Operacije nad množicami**. Operirajo nad vrsticami, kot da so ti deli množice.

### 2.9.1 Mutirajoča združevanja

Mutirajoča združevanja so pogosta operacija pri delu z relacijskimi podatki. Te operacije združijo dve (ali več) razpredelnici glede na vrednosti stolpcev. Obstajajo 4 takšne operacije:

- `left_join()`. Ohrani vse vrstice prve (leve) razpredelnice in poveže ustrezne vrstice iz druge razpredelnice s temi vrsticami.
- `right_join()`. Ohrani vse vrstice druge (desne) razpredelnice in poveže ustrezne vrstice iz prve razpredelnice s temi vrsticami.
- `full_join()`. Ohrani vse vrstice obeh razpredelnic.
- `inner_join()`. Ohrani samo tiste vrstice, ki se pojavijo v obeh razpredelnicih.

Prvi trije so tako imenovani zunanji stiki (*outer join*), saj uporabijo vrstice, ki se pojavijo vsaj v eni razpredelnici. Za lažje razumevanje bomo najprej prikazali uporabo stikov na podatkih, ki jih bomo ustvarili sami. Sintaksa pri vseh združevanjih je:

```
left_join(<razpredelnica1>, <razpredelnica2>)
```

Ustvarimo dva tibbla:

```
df1 <- tibble(
 id = c("id1", "id2", "id3", "id4"),
 x = c(4, 6, 1, 2)
)

df2 <- tibble(
 id = c("id1", "id3", "id4", "id5"),
 y = c(20, 52, 11, 21)
)

df1
```

```
A tibble: 4 x 2
id x
<chr> <dbl>
1 id1 4
2 id2 6
3 id3 1
4 id4 2
```

```
df2
```

```
A tibble: 4 x 2
id y
<chr> <dbl>
1 id1 20
2 id3 52
3 id4 11
4 id5 21
```

- `left_join()` obdrži tibble `df1` takšen kot je in mu pripne stolpec `y` iz tibbla `df2`, kjer so vrednosti spremenljivke `id` enake. Za tiste vrstice `df1`, ki nimajo ustreznega `id` v `df2` se vrednosti v spremenljivki `y` nastavijo na `NA`.

```
left_join(df1, df2)
```

```
A tibble: 4 x 3
id x y
<chr> <dbl> <dbl>
1 id1 4 20
2 id2 6 NA
3 id3 1 52
4 id4 2 11
```

- `right_join()` obdrži tibble `df2` takšen kot je in mu pripne stolpec `x` iz tibbla `df1`, kjer so vrednosti spremenljivke `id` enake. Za tiste vrstice `df2`, ki nimajo ustreznega `id` v `df1`, se vrednosti v spremenljivki `y` nastavijo na `NA`.

```
right_join(df1, df2)
```

```
A tibble: 4 x 3
id x y
<chr> <dbl> <dbl>
1 id1 4 20
2 id3 1 52
3 id4 2 11
4 id5 NA 21
```

- `inner_join()` obdrži samo tiste podatke, kjer se `id` nahaja v obeh razporednicah (torej 1, 3 in 4). Vse preostale vrstice zavrže.

```
inner_join(df1, df2)
```

```
A tibble: 3 x 3
id x y
<chr> <dbl> <dbl>
1 id1 4 20
2 id3 1 52
3 id4 2 11
```

- `full_join()` obdrži vse podatke iz `df1` in `df2`. Kjer ni ustreznega `id` v nasprotni razpredelnici se vrednosti nastavijo na `NA`.

```
full_join(df1, df2)
```

```
A tibble: 5 x 3
id x y
<chr> <dbl> <dbl>
1 id1 4 20
2 id2 6 NA
3 id3 1 52
4 id4 2 11
5 id5 NA 21
```

Najbolj pogosto bomo uporabljali `left_join()`, kadar bo cilj obdržati originalno razpredelnico, kot je, ali `inner_join()`, kadar bomo želeli podatke brez manjkajočih vrednosti. Stik `right_join()` je samo drugače usmerjen `left_join()`.

Do sedaj smo prikazovali, kako združimo razpredelnice glede na primarni ključ, za katerega predpostavljamo, da je unikaten. Torej vsaka vrstica ima svoj ključ, ki se v razpredelnici ne ponovi. Včasih pa razpredelnice združujemo glede na sekundarni ključ. V tem primeru se lahko zgodi, da imamo relacijo ena-proti-mnogo. Če vzamemo bančne podatke od zgoraj ima lahko en račun več skrbnikov. Kaj se zgodi v tem primeru? Kaj pa če združimo transakcije in osebe glede na račun? En račun ima lahko več transakcij in več skrbnikov. Ker pri obeh razpredelnicah uporabimo sekundarni ključ, bomo najverjetneje dobili podvojene vrednosti pri obeh. Poglejmo si sedaj na primeru podatkov, ki jih generiramo sami.

```
df3 <- tibble(
 id1 = c("id1", "id2", "id3", "id4"),
 id2 = c("id1", "id1", "id3", "id4"),
 x = c(5, 6, 1, 2)
)
```

```
df4 <- tibble(
 id2 = c("id1", "id2", "id3"),
 y = c(20, 52, 11)
)
df5 <- tibble(
 id3 = c("id1", "id2", "id3", "id4"),
 id2 = c("id1", "id1", "id4", "id5"),
 z = c(5, 1, 23, 5)
)
df3
```

```
A tibble: 4 x 3
id1 id2 x
<chr> <chr> <dbl>
1 id1 id1 5
2 id2 id1 6
3 id3 id3 1
4 id4 id4 2
```

```
df4
```

```
A tibble: 3 x 2
id2 y
<chr> <dbl>
1 id1 20
2 id2 52
3 id3 11
```

```
df5
```

```
A tibble: 4 x 3
id3 id2 z
<chr> <chr> <dbl>
1 id1 id1 5
2 id2 id1 1
3 id3 id4 23
4 id4 id5 5
```

df3 in df5 imata podvojen sekundarni ključ. Združimo sedaj df3 in df4 z uporabo `inner_join()`.

```
inner_join(df3, df4)
```

```
A tibble: 3 x 4
id1 id2 x y
<chr> <chr> <dbl> <dbl>
1 id1 id1 5 20
2 id2 id1 6 20
3 id3 id3 1 11
```

Ključ torej ostane podvojen. Kaj pa se zgodi, če imata obe razpredelnici podvojene ključee? V tem primeru dobimo kartezični produkt vseh podvojenih vrednosti:

```
inner_join(df3, df5)
```

```
A tibble: 5 x 5
id1 id2 x id3 z
<chr> <chr> <dbl> <chr> <dbl>
1 id1 id1 5 id1 5
2 id1 id1 5 id2 1
3 id2 id1 6 id1 5
4 id2 id1 6 id2 1
5 id4 id4 2 id3 23
```

df3 in df5 imata podvojeno vrednost ključa id1. Torej dobimo vse kombinacije preostalih vrednosti (5, 5), (5, 1), (6, 5) in (6, 1).

### 2.9.2 Argument by

Združevanja, ki smo jih spoznali, privzeto združijo razpredelnici glede na vrednosti v vseh stolpcih, ki imajo enaka imena – tamu pravimo tudi **naravno združevanje** (ang. **natural join**). Lahko pa tudi sami določimo, po katerih stolpcih želimo združiti podatke. To naredimo tako, da pri združevanjih uporabimo argument `by`. Sintaksa združevanj je potem:

```
inner_join(<razpredelnica1>, <razpredelnica2>, by = <vektor-imen-stolpcev>)
```

`inner_join()` dveh razpredelnic bi potem zapisali kot:

```
inner_join(df3, df4, by = "id2")
```

```
A tibble: 3 x 4
id1 id2 x y
<chr> <chr> <dbl> <dbl>
1 id1 id1 5 20
2 id2 id1 6 20
3 id3 id3 1 11
```

Ta primer služi samo kot ilustracija in je uporaba `by` nepotrebna. Seveda pa se pri realnih podatkih velikokrat srečamo s stanjem, kjer ta argument potrebujemo. Prav tako je koda s parametrom `by` bolj robustna, saj sami definiramo, glede na katere stolpce naj se razpredelnice združujejo in ne more priti do kakšnih napak pri ponovljivosti.

Razpredelnici lahko združimo tudi po stolpcih, ki nimajo istega imena. Ni nenavadno, da imamo dve razpredelnici z enakimi spremenljivkami, ki pa so poimenovane drugače. Če bi želeli taki razpredelnici združiti glede na to spremenljivko, potem bi jo morali načeloma v eni razpredelnici preimenovali. S paketom `dplyr` pa lahko to naredimo tudi drugače. Združimo sedaj `df3` in `df5` glede na stolpca `x` in `z` ter skupni stolpec `id2`.

```
inner_join(df3, df5, by = c("x" = "z", "id2"))
```

```
A tibble: 1 x 4
id1 id2 x id3
<chr> <chr> <dbl> <chr>
1 id1 id1 5 id1
```

### 2.9.3 Filtrirajoča združevanja

Pri filtrirajočih združevanjih ne gre toliko za združevanja, kolikor gre za izbiro posameznih vrstic, glede na ujemanje vrednotsti stolpcev v neki drugi razpredelnici. Poznamo 2 takšni združevanja:

- `semi_join()`. Obdrži vse vrstice v prvi razpredelnici, ki ustrezajo vrsticam v drugi razpredelnici.
- `anti_join()`. Izloči vse vrstice v prvi razpredelnici, ki ustrezajo vrsticam v drugi razpredelnici.

Poglejmo si uporabo teh združevanj na naših generiranih razpredelnicah.

```
df1
```

```
A tibble: 4 x 2
id x
<chr> <dbl>
1 id1 4
2 id2 6
3 id3 1
4 id4 2
```



```
df2
```

```
A tibble: 4 x 2
id y
<chr> <dbl>
1 id1 20
2 id3 52
3 id4 11
4 id5 21
```

```
semi_join(df1, df2)
```

```
A tibble: 3 x 2
id x
<chr> <dbl>
1 id1 4
2 id3 1
3 id4 2
```

`semi_join()` je torej izločil vrstico z `id2`, saj se ta ne pojavi v `df2`.

```
anti_join(df1, df2)
```

```
A tibble: 1 x 2
id x
<chr> <dbl>
1 id2 6
```

`semi_join()` je torej izločil vse vrstice, ki se pojavijo tudi v `df2`. Ostane torej samo vrstica z `id2`.

### 2.9.4 Združevanja na realnih podatkih

Sedaj, ko smo spoznali glavne lastnosti različnih združevanj na primerih, ki so nam omogočali lažjo predstavo, pa se posvetimo realnim podatkom, ki smo jih predstavili v začetku tega poglavja. Imamo torej podatke o bančnih računih, transakcijah, posojilih, skrbnikih računov in povezovalno razpredelnico med računi in skrbniki. Lotimo se sedaj relativno enostavne analize, kjer bomo naredili sledeče:

- 1) Ustvarili novo razpredelnico, kjer bomo imeli podatke o vseh bančnih računih in o lastnikih teh računov. Lastnik računa je lahko samo en, medtem ko je skrbnikov lahko več.

- 2) Filtrirali razpredelnico iz točke 1), v kateri bodo samo tisti, ki imajo posojila nad 100000 kron.
- 3) Ustvarili novo razpredelnico klientov, kjer bomo imeli podatke o klientih in posojilih in bodo zajeti samo klienti s posojili.
- 4) Izračunali kateri klienti, ki imajo posojilo, imajo tudi največ transakcij.

Za vsakega izmed teh korakov bomo morali uporabiti eno od združevanj, ki smo jih spoznali. Na primer, v samih razpredelnicah nimamo direktne povezave med komitenti in transakcijami, tako da bomo morali zadeve nekako združiti. Podobno velja za ostale alineje.

Najprej želimo združiti razpredelnici `account` in `client`. Za to bomo potrebovali povezovalno razpredelnico `disp` v kateri se tudi nahaja informacija o tem, ali je klient lastnik ali samo skrbnik računa. Najprej povežemo razpredelnico `account` z razpredelnico `disp` in filtriramo glede na tip klienta:

```
account_disp <- left_join(account, disp) %>%
 filter(type == "OWNER")
account_disp
```

```
A tibble: 4,500 x 7
account_id district_id frequency date disp_id client_id type
<dbl> <dbl> <chr> <date> <dbl> <dbl> <chr>
1 1 18 monthly payment 1995-03-24 1 1 OWNER
2 2 1 monthly payment 1993-02-26 2 2 OWNER
3 3 5 monthly payment 1997-07-07 4 4 OWNER
4 4 12 monthly payment 1996-02-21 6 6 OWNER
5 5 15 monthly payment 1997-05-30 7 7 OWNER
6 6 51 monthly payment 1994-09-27 8 8 OWNER
7 7 60 monthly payment 1996-11-24 9 9 OWNER
8 8 57 monthly payment 1995-09-21 10 10 OWNER
9 9 70 monthly payment 1993-01-27 12 12 OWNER
10 10 54 monthly payment 1996-08-28 13 13 OWNER
... with 4,490 more rows
```

Sedaj lahko to novo razpredelnico povežemo z razpredelnico `client`.

```
account_client <- left_join(account_disp, client)
```

S tem smo dobili željeno razpredelnico, v kateri imamo za vsak račun tudi informacijo o lastniku. Kot drugi korak želimo ustvariti razpredelnico, v kateri bodo samo podatki o klientih, ki imajo posojila v vrednosti več kot 100000 kron. Najprej ustvarimo razpredelnico, v kateri so samo takšna posojila, nato pa uporabimo `semi_join()`, ki bo iz razpredelnice `account_client` izbral samo vrstice, ki se bodo ujemale z vrsticami v tej novi razpredelnici posojil.

```
loan_100k <- loan %>%
 filter(amount > 100000)
account_100k <- semi_join(account_client, loan_100k)
account_100k
```

```
A tibble: 0 x 9
... with 9 variables: account_id <dbl>, district_id <dbl>, frequency <chr>,
date <date>, disp_id <dbl>, client_id <dbl>, type <chr>, gender <chr>,
birth_date <date>
```

Hm...dobili smo prazen tibble, čeprav obstajajo tako velika posojila. Zakaj je do tega prišlo? V obeh razpredelnicih se nahajata spremenljivki `account_id` in `date`. Ampak spremenljivka `date` ni ista spremenljivka, pri razpredelnici `account_client` predstavlja, kdaj je bil račun odprt, pri `loan_100k` pa predstavlja kdaj je bilo posojilo odobreno. Torej po tej spremenljivki ne smemo združevati. Uporabimo `by`:

```
account_100k <- semi_join(account_client, loan_100k, by = "account_id")
account_100k
```

```
A tibble: 377 x 9
account_id district_id frequency date disp_id client_id type gender
<dbl> <dbl> <chr> <date> <dbl> <dbl> <chr> <chr>
1 37 20 monthly pay~ 1997-08-18 45 45 OWNER M
2 38 19 weekly paym~ 1997-08-08 46 46 OWNER F
3 67 16 monthly pay~ 1994-10-19 78 78 OWNER F
4 97 74 monthly pay~ 1996-05-05 116 116 OWNER M
5 103 44 monthly pay~ 1996-03-10 124 124 OWNER M
6 105 21 monthly pay~ 1997-07-10 127 127 OWNER F
7 110 36 monthly pay~ 1996-07-17 132 132 OWNER M
8 173 66 monthly pay~ 1993-11-26 210 210 OWNER M
9 226 70 monthly pay~ 1997-02-23 272 272 OWNER F
10 276 38 monthly pay~ 1997-12-08 333 333 OWNER M
... with 367 more rows, and 1 more variable: birth_date <date>
```

V naslednjem koraku želimo imeti podatke o klientih in posojilih. Najprej bomo morali združiti razpredelnici `client` in `disp`, nato pa dodati še razpredelnico `loan`. Ustvarimo to novo razpredelnico kar z uporabo operatorja `%>%`:

```
client_loan <- client %>%
 left_join(disp) %>%
 inner_join(loan, by = "account_id")
client_loan
```

```
A tibble: 827 x 13
client_id gender birth_date district_id disp_id account_id type loan_id
<dbl> <chr> <date> <dbl> <dbl> <dbl> <chr> <dbl>
1 2 M 1945-02-04 1 2 2 OWNER 4959
2 3 F 1940-10-09 1 3 2 DISPONENT 4959
3 25 F 1939-04-23 21 25 19 OWNER 4961
4 31 M 1962-02-09 68 31 25 OWNER 4962
5 45 M 1952-08-26 20 45 37 OWNER 4967
6 46 F 1940-01-30 19 46 38 OWNER 4968
7 78 F 1944-06-13 16 78 67 OWNER 4973
8 116 M 1942-01-28 74 116 97 OWNER 4986
9 117 F 1936-09-20 74 117 97 DISPONENT 4986
10 124 M 1967-09-21 44 124 103 OWNER 4988
... with 817 more rows, and 5 more variables: date <date>, amount <dbl>,
duration <dbl>, payments <dbl>, status <chr>
```

Na koncu preverimo še, kateri klienti, ki imajo posojilo, imajo največ transakcij. Za to bomo morali najprej izračunati število transakcij na vsakem bančnem računu. Uporabimo znanje, ki smo ga pridobili na prvem predavanju:

```
trans_count <- trans %>%
 group_by(account_id) %>%
 summarise(n_trans = n())
trans_count
```

```
A tibble: 4,205 x 2
account_id n_trans
<dbl> <int>
1 1 6
2 2 10
3 3 3
4 4 6
5 5 3
6 6 8
7 7 3
8 8 7
9 9 3
10 10 1
... with 4,195 more rows
```

```
left_join(client_loan, trans_count) %>%
 arrange(desc(n_trans))
```

```
A tibble: 827 x 14
```

```
client_id gender birth_date district_id disp_id account_id type loan_id
<dbl> <chr> <date> <dbl> <dbl> <dbl> <chr> <dbl>
1 11126 F 1965-01-22 1 10818 9034 OWNER 6820
2 6367 M 1970-04-28 44 6367 5270 OWNER 6077
3 7291 F 1940-12-02 77 7291 6034 OWNER 6228
4 7195 M 1962-09-11 50 7195 5952 OWNER 6216
5 4620 F 1940-11-01 54 4620 3834 OWNER 5754
6 4621 M 1946-02-10 54 4621 3834 DISPONENT 5754
7 11461 M 1974-07-08 70 11153 9307 OWNER 6895
8 11866 M 1937-09-02 1 11558 9640 OWNER 6960
9 11867 F 1934-11-19 1 11559 9640 DISPONENT 6960
10 13657 F 1963-05-12 59 13349 11111 OWNER 7259
... with 817 more rows, and 6 more variables: date <date>, amount <dbl>,
duration <dbl>, payments <dbl>, status <chr>, n_trans <int>
```

## 2.10 Operacije nad množicami

V tem poglavju si bomo ogledali operacije nad množicami. Te delujejo nad vektorji, prav tako pa nad `data.frame` oziroma nad `tibble`. Poznamo 3 glavne operacije:

- **Unija.** Vrne vse elemente, ki se pojavijo v eni ali drugi množici.
- **Presek.** Vrne vse elemente, ki se pojavijo v obeh množicah.
- **Razlika.** Vrne vse elemente prve množice, ki se ne pojavijo v drugi množici.

Poglejmo si uporabo teh operacij nad `tibble`.

```
df1 <- tibble(
 id = c("id1", "id2"),
 x = c(4, 6)
)

df2 <- tibble(
 id = c("id1", "id3"),
 x = c(4, 52)
)

df1
```

```
A tibble: 2 x 2
id x
<chr> <dbl>
1 id1 4
2 id2 6
```

```
df2
```

```
A tibble: 2 x 2
id x
<chr> <dbl>
1 id1 4
2 id3 52
```

```
union(df1, df2)
```

```
A tibble: 3 x 2
id x
<chr> <dbl>
1 id1 4
2 id2 6
3 id3 52
```

```
intersect(df1, df2)
```

```
A tibble: 1 x 2
id x
<chr> <dbl>
1 id1 4
```

```
setdiff(df1, df2)
```

```
A tibble: 1 x 2
id x
<chr> <dbl>
1 id2 6
```

```
setdiff(df2, df1)
```

```
A tibble: 1 x 2
id x
<chr> <dbl>
1 id3 52
```

## 2.11 Ali želite izvedeti več?

Hadley Wickham je objavil znanstveni članek na temo urejenih podatkov: <https://vita.had.co.nz/papers/tidy-data.pdf>, ki je vsekakor vreden branja. Za več informacij o neurejenih podatkih in v katerih primerih so lahko celo bolj zaželeni, predlagamo ta blog: <https://simplystatistics.org/2016/02/17/non-tidy-data/>.

## 2.12 Domača naloga

- 1) Spodaj imamo podatke o stroških za 4 osebe. Razpredelnica je v neurejeni obliki. Vaša naloga je, da jo pretvorite v urejeno obliko.

```
podatki_o_stroskih <- tibble(
 ime = c("Miha", "Ana", "Andrej", "Maja"),
 april_2019 = c(400, 200, 300, 350),
 maj_2019 = c(390, 250, 280, 400),
 april_2020 = c(410, 150, 500, 400),
 maj_2020 = c(300, 320, 550, 320)
)
```

Rešitev:

```
A tibble: 16 x 4
ime mesec leto strosek
<chr> <chr> <chr> <dbl>
1 Miha april 2019 400
2 Miha maj 2019 390
3 Miha april 2020 410
4 Miha maj 2020 300
5 Ana april 2019 200
6 Ana maj 2019 250
7 Ana april 2020 150
8 Ana maj 2020 320
9 Andrej april 2019 300
10 Andrej maj 2019 280
11 Andrej april 2020 500
12 Andrej maj 2020 550
13 Maja april 2019 350
14 Maja maj 2019 400
15 Maja april 2020 400
16 Maja maj 2020 320
```

- 2) V mapi *data-raw* se nahajajo podatki o predsedniških volitvah v ZDA. Najprej izberite samo podmnožico vrstic, kjer sta kandidata Joe Biden ali Donald Trump, in izločite stolpec `party`. Nato pretvorite podatke v širšo obliko, tako da bo vsak izmed kandidatov imel svoj stolpec.

```
A tibble: 4,633 x 4
state county `Joe Biden` `Donald Trump`
<chr> <chr> <dbl> <dbl>
1 Delaware Kent County 44518 40976
2 Delaware New Castle County 194238 87685
```

```
3 Delaware Sussex County 56657 71196
4 District of Columbia District of Columbia 29509 1149
5 District of Columbia Ward 2 24247 2365
6 District of Columbia Ward 3 33584 2972
7 District of Columbia Ward 4 35117 1467
8 District of Columbia Ward 5 36585 1416
9 District of Columbia Ward 6 44699 3360
10 District of Columbia Ward 7 30253 885
... with 4,623 more rows
```

- 3) Pri bančnih podatkih smo zaenkrat delali samo s petimi razpredelnici. Celotna zbirka je nekoliko večja, saj vsebuje še 3 razpredelnice. V mapi *data-raw/financial-hw/* se nahajajo še preostale razpredelnice. Pri nalogi bomo uporabili razpredelnico *district*, ki vsebuje podatke o okrajih. Vsekakor pa se lahko za lastno vajo poigrate še s preostalima dvema. Vaše naloge so:

- Preberite podatke o okrajih v R.
- Ugotovite, kaj je primarni ključ te razpredelnice.
- Ustrezno dopolnite entitetni diagram. To lahko naredite ročno, v kolikor pa se želite naučiti narediti bolj profesionalne diagrame pa predlagamo spletno orodje <https://app.diagrams.net/>. V mapi *support-files* se nahaja naš diagram. Tega lahko enostavno naložite v to orodje in ga dopolnite.
- Poiščite 3 pokrajine (A3) z največjo povprečno vrednostjo posojil.

```
A tibble: 3 x 2
A3 mean_loan
<chr> <dbl>
1 east Bohemia 165997.
2 south Bohemia 156236.
3 central Bohemia 155392.
```

- 4) **Težka naloga.** Namestite paket *nycflights13*. Gre za relacijsko podatkovno zbirko o letih iz New Yorka v letu 2013. Naložite podatke z `library(nycflights13)`. Uporabljali bomo štiri razpredelnice: *flights*, *weather*, *airlines* in *planes*. Vaša naloga je:

- Poizkusite najti primarni ključ za razpredelnico *flights*. Ali gre za primarni ključ lahko preverite tako, da preverite ali ta ključ unikatno določa vrstico v podatkih, torej da preštejete podatke, grupirane glede na ta ključ. Ključ je lahko sestavljen tudi iz večih spremenljivk. Na prvi pogled bi rekli, da je primarni ključ številka leta, ampak temu ni



tako (preverimo s štetjem). Ali je morda kakšna druga kombinacija spremenljivk? Lahko da razpredelnica nima primarnega ključa. V tem primeru določite nadomestni ključ tako, da dodate stolpec ID z `mutate(ID = row_number())`.

- Ugotovite, kaj so primarni in kaj tuji ključi preostalih razpredelnic. Pri nekaterih razpredelnicah v tej zbirki bomo imeli sestavljene ključe, torej bodo ključi sestavljeni iz večih stolpcev. Namig: Pri vremenu je manjša napaka v podatkih in se tudi primarni ključ ponovi v zanemarljivem številu primerov. Vsekakor so napake v realnih podatkih pričakovane in moramo na to biti pozorni!
- Narišite relacijski diagram.
- Ustvarite novo razpredelnico tako da razpredelnici `flights` dodate podrobnosti o lastnostih letal za vsak let.

```
A tibble: 336,776 x 28
ID year.x month day dep_time sched_dep_time dep_delay arr_time
<int> <int> <int> <int> <int> <int> <dbl> <int>
1 1 2013 1 1 517 515 2 830
2 2 2013 1 1 533 529 4 850
3 3 2013 1 1 542 540 2 923
4 4 2013 1 1 544 545 -1 1004
5 5 2013 1 1 554 600 -6 812
6 6 2013 1 1 554 558 -4 740
7 7 2013 1 1 555 600 -5 913
8 8 2013 1 1 557 600 -3 709
9 9 2013 1 1 557 600 -3 838
10 10 2013 1 1 558 600 -2 753
... with 336,766 more rows, and 20 more variables: sched_arr_time <int>,
arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
time_hour <dtm>, year.y <int>, type <chr>, manufacturer <chr>,
model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>
```

- Ustvarite novo razpredelnico tako da razpredelnici `flights` dodate podrobnosti o vremenu na letališču vsak let.

```
A tibble: 336,776 x 30
ID year month day dep_time sched_dep_time dep_delay arr_time
<int> <int> <int> <int> <int> <int> <dbl> <int>
1 1 2013 1 1 517 515 2 830
2 2 2013 1 1 533 529 4 850
3 3 2013 1 1 542 540 2 923
4 4 2013 1 1 544 545 -1 1004
```

```
5 5 2013 1 1 554 600 -6 812
6 6 2013 1 1 554 558 -4 740
7 7 2013 1 1 555 600 -5 913
8 8 2013 1 1 557 600 -3 709
9 9 2013 1 1 557 600 -3 838
10 10 2013 1 1 558 600 -2 753
... with 336,766 more rows, and 22 more variables: sched_arr_time <int>,
arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
time_hour.x <dtm>, temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>,
wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
visib <dbl>, time_hour.y <dtm>
```

- Poiščite vsa letala z 2 motorjema, ki so v New York priletela 5. aprila iz letališča Chicago Ohare Intl.

```
A tibble: 35 x 35
ID year.x month day dep_time sched_dep_time dep_delay arr_time
<int> <int> <int> <int> <int> <int> <dbl> <int>
1 169022 2013 4 5 545 600 -15 710
2 169051 2013 4 5 604 610 -6 739
3 169062 2013 4 5 624 630 -6 755
4 169086 2013 4 5 636 630 6 814
5 169100 2013 4 5 653 700 -7 827
6 169114 2013 4 5 659 700 -1 823
7 169125 2013 4 5 720 725 -5 851
8 169159 2013 4 5 752 759 -7 920
9 169183 2013 4 5 810 815 -5 933
10 169201 2013 4 5 823 830 -7 957
... with 25 more rows, and 27 more variables: sched_arr_time <int>,
arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
time_hour <dtm>, year.y <int>, type <chr>, manufacturer <chr>,
model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>,
name <chr>, lat <dbl>, lon <dbl>, alt <dbl>, tz <dbl>, dst <chr>,
tzone <chr>
```

- 5) **Zelo težka naloga.** V mapi *data-raw* se nahajajo podatki o kreditnih karticah v Tajvanu *default of credit card clients.xlsx*. Pridobili smo jih iz UCI Machine Learning repozitorija (Dua and Graff, 2017). Podatki so bili uporabljeni v znanstveni raziskavi (Yeh and Lien, 2009), kjer so napovedovali verjetnosti neplačil v odvisnosti od preteklih transakcij na kartici in podatkov o lastnikih. Podatki so v *xlsx* datoteki. Bodite pozorni, da je prva vrstica datoteke nepomembna in se glava začne komaj v drugi vrstici. Trenutno so podatki v obliki, v kateri so zelo primerni kot vhodni

podatek za kak model, na primer linearno regresijo. Vsekakor pa niso v primerni obliki za učinkotivo urejanje in hranjenje. Vaša naloga je, da podatke preberete v R in razpredelnico pretvorite v urejeno obliko.

Predlagamo, da nalogo poizkusite rešiti sami. Naloga zahteva precej razmisleka in tudi nekaj samostojne raziskave (na primer, kaj posamezni stolpci pomenijo – pomagajte si lahko s spletno stranjo, iz katere smo prenesli podatke). V kolikor se vam zatakne, smo vam spodaj pripravili nekaj namigov:

- Najprej je potrebno razmisliti, kaj so spremenljivke. Na primer, ali sta `PAY_1` in `PAY_2` 2 spremenljivki, ali predstavljata 1 spremenljivko, ki pa je razdeljena glede na neko drugo spremenljivko?
- Predlagamo da začnete ukaze tako, da razpredelnico spremenite v daljšo obliko, kjer vse spremenljivke, ki se pojavijo v večih stolpcih, shranite v 1 stolpec.
- V novem stolpcu so celice sestavljene iz 2 spremenljivk. Ena od teh je ID meseca. Torej moramo ta stolpec ločiti na 2 stolpca. Katero funkcijo uporabimo za to? Pri tem bo prav prišel tudi argument te funkcije `sep = -1`, ki bo stolpec ločil na zadnji znak v besedi in vse preostalo (na primer, “beseda3” bo razdelil na “beseda” in “3”). -1 predstavlja pri koliko znakih od konca proti začetku naredimo ločitev besede.
- V enem od teh dveh preostalih stolpcev imamo še vedno shranjene 3 spremenljivke, za katere bi bilo bolje, če so v 3 stolpcih. Ustrezno pretvorite tabelo. Na tej točki smo že skoraj pri koncu.
- ID mesecev žal ne sovпада z zaporednimi števili mesecev v letu. Predlagamo, da si ustvarite novo razpredelnico, ki bo mapirala ID mesecev v njihova zaporedna števila. Potem pa to razpredelnico povežete z razpredelnico, kjer hranimo podatke. Kako naredimo to? Kadar združujemo razpredelnice moramo tudi biti pozorni na to, da so stolpci, ki jih združujemo, istega tipa.

```
A tibble: 180,000 x 10
ID LIMIT_BAL SEX EDUCATION MARRIAGE AGE MONTH PAY PAY_AMT BILL_AMT
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1 1 20000 2 2 1 24 9 2 0 3913
2 1 20000 2 2 1 24 8 2 689 3102
3 1 20000 2 2 1 24 7 -1 0 689
4 1 20000 2 2 1 24 6 -1 0 0
5 1 20000 2 2 1 24 5 -2 0 0
6 1 20000 2 2 1 24 4 -2 0 0
7 2 120000 2 2 2 26 9 -1 0 2682
8 2 120000 2 2 2 26 8 2 1000 1725
```

```
9 2 120000 2 2 2 26 7 0 1000 2682
10 2 120000 2 2 2 26 6 0 1000 3272
... with 179,990 more rows
```

## Chapter 3

# Nizi, kategorične spremenljivke in datumi

Pri delu s podatki se pogosto srečamo s posebnimi podatkovnimi tipi, kot so nizi, kategorične spremenljivke in datumi. Z nizi smo že delali v prvih dveh poglavjih. Delali smo tudi s kategoričnimi spremenljivkami, čeprav tega nismo poudarili. V tem poglavju bomo spoznali naprednejše operacije nad temi podatkovnimi tipi v okviru zbirke tidyverse ter predstavili praktične primere, dobre prakse in pasti pri delu z njimi.

Konec poglavja pa bomo posvetili branju in pisanju podatkov v različnih oblikah.

### 3.1 Priprava

#### Nizi

Najprej si pogledjmo najbolj uporabne funkcije za delo z nizi.

Nize lahko združujemo na več načinov:

```
crke <- c("A", "B", "C", "D")
stevilke <- c("1", "2", "3", "4")
str_c(crke, stevilke)
```

```
[1] "A1" "B2" "C3" "D4"
```

```
str_c(crke, stevilke, sep = "-")
```

```
[1] "A-1" "B-2" "C-3" "D-4"
```

```
str_c(crke, stevilke, collapse = "-")
```

```
[1] "A1-B2-C3-D4"
```

Obratno jih lahko razčlenimo na več manjših nizov:

```
str_split("A1-B2-C3-D4", pattern = "-")
```

```
[[1]]
```

```
[1] "A1" "B2" "C3" "D4"
```

Zelo uporabna je tudi funkcija za zamenjavo podnizov.

```
str_replace("A1-B1-A2-B2", pattern = "1", "(ena)")
```

```
[1] "A(ena)-B1-A2-B2"
```

```
str_replace_all("A1-B1-A2-B2", pattern = "1", "(ena)")
```

```
[1] "A(ena)-B(ena)-A2-B2"
```

```
str_replace_all("A1-B1-A2-B2", c("1" = "(ena)", "2" = "(dva)"))
```

```
[1] "A(ena)-B(ena)-A(dva)-B(dva)"
```

Poglejmo, kako bi iz zbirke sadežev v že vgrajenem nizu `fruit` poiskali vse nize, ki vsebujejo `berry`.

```
length(fruit)
```

```
[1] 80
```

```
str_subset(fruit, "berry")
```

```
[1] "bilberry" "blackberry" "blueberry" "boysenberry" "cloudberry"
[6] "cranberry" "elderberry" "goji berry" "gooseberry" "huckleberry"
[11] "mulberry" "raspberry" "salal berry" "strawberry"
```

Podobno lahko vrnemo logični vektor nizov, ki vsebujejo `berry`.

```
str_detect(fruit, "berry")
```

```
[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE FALSE TRUE TRUE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE TRUE FALSE FALSE TRUE TRUE FALSE FALSE FALSE
[37] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[49] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
[73] TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE
```

Funkcija `str_detect()` je zelo uporabna pri izbiranju vrstic glede na nize. Običajno jo uporabljamo v kombinaciji z regularnimi izrazi, ki jih bomo spoznali v jedru tega poglavja.

### Kategorične spremenljivke

Pri pripravi na kategorične spremenljivke bomo uporabili podatke o uspehu študentov na izpitu:

```
tib <- tibble(
 ime = c("Maja", "Ales", "Tom", "Barbara", "Simon", "Tina"),
 spol = c("z", "m", "m", "z", "m", "z"),
 ocena = c(10, 10, 6, 8, 8, 7),
 dan_izpita = c(8, 8, 16, 16, 8, 23),
 mesec_izpita = rep(6, 6),
 leto_izpita = rep(2021, 6),
 opravljene_vaje = c(1, 1, 1, 1, 0, 1),
 procenti_vaje = c(70, 65, 80, 90, 30, 75)
)
tib
```

```
A tibble: 6 x 8
ime spol ocena dan_izpita mesec_izpita leto_izpita opravljene_vaje
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Maja z 10 8 6 2021 1
2 Ales m 10 8 6 2021 1
3 Tom m 6 16 6 2021 1
4 Barbara z 8 16 6 2021 1
5 Simon m 8 8 6 2021 0
6 Tina z 7 23 6 2021 1
... with 1 more variable: procenti_vaje <dbl>
```

Spremenimo **spol** v nominalno spremenljivko (brez ureditve) in **oceno** v ordinalno spremenljivko (obstaja vrsti red vrednosti):

```
tib <- tib %>% mutate(spol = factor(spol),
 ocena = factor(ocena, ordered = TRUE))
tib$spol
```

```
[1] z m m z m z
Levels: m z
```

```
tib$ocena
```

```
[1] 10 10 6 8 8 7
Levels: 6 < 7 < 8 < 10
```

Nivoje faktorja (levels) lahko uredimo tudi glede na vrednosti v nekem drugem stolpcu. To običajno pride prav pri vizualizaciji.

```
tib <- tib %>%
 mutate(spol = fct_reorder(spol, procenti_vaje, .desc = TRUE))
tib$spol
```

```
[1] z m m z m z
Levels: z m
```

## Datumi

Za delo z datumi bomo uporabljali paket **lubridate**. Datume lahko z lubridate shranimo na več načinov. Preprosto lahko sestavimo datum s posameznimi komponentami:

```
library(lubridate)
make_date(year = 2021, month = 6, day = 11)
```

```
[1] "2021-06-11"
```

Ali iz niza:

```
ymd("2021-06-11")
```

```
[1] "2021-06-11"
```

Lahko dodamo tudi čas:



```
make_datetime(year = 2021, month = 6, day = 11, hour = 11, min = 30, tz = "CET")
```

```
[1] "2021-06-11 11:30:00 CEST"
```

Z datumi lahko tudi računamo:

```
my_date <- make_date(year = 2021, month = 6, day = 11)
my_date
```

```
[1] "2021-06-11"
```

```
my_date + days(2)
```

```
[1] "2021-06-13"
```

```
my_date + months(3)
```

```
[1] "2021-09-11"
```

### Shranjevanje podatkov v csv

V pripravi v 2. poglavju smo že spoznali funkcijo `rear_csv2()` za branje podatkov iz tekstovnih datotek. Podatke lahko tudi shranimo v tekstovno datoteko s funkcijo `write_csv2()`:

```
write_csv2(tib, "./data-raw/studenti.csv")
```

**Naloga 1:** Datoteko *ocene.csv* v mapi *data-raw* preberite s funkcijami paketa `readr` (`read_csv2()` in `write_csv2()`). Shranite jo kot *ocene2.csv* in to datoteko preberite nazaj. Nato naredite enako z baznimi funkcijami v R (`read.csv2()` in `write.csv2()`). Kaj opazite?

Z `readr`:

```
A tibble: 6 x 7
ime spol ocena dan_izpita mesec_izpita leto_izpita opravljene_vaje
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Maja z 7 8 6 2021 1
2 Ales m 10 8 6 2021 1
3 Tom m 6 16 6 2021 1
4 Barbara z 8 16 6 2021 1
5 Simon m 8 8 6 2021 0
6 Tina z 7 23 6 2021 1
```

Z R:

```
X ime spol ocena dan_izpita mesec_izpita leto_izpita opravljene_vaje
1 1 Maja z 7 8 6 2021 1
2 2 Ales m 10 8 6 2021 1
3 3 Tom m 6 16 6 2021 1
4 4 Barbara z 8 16 6 2021 1
5 5 Simon m 8 8 6 2021 0
6 6 Tina z 7 23 6 2021 1
```

**Naloga 2:** Ponovno preberite *ocene.csv* ter:

- spremenite stolpec *opravljene\_vaje* v kategorične vrednosti,
- zamenjajte stolpce *leto\_izpita*, *mesec\_izpita*, *dan\_izpita* z *datum\_izpita*, ki vsebuje vse te podatke,
- izberite samo imena, ki se začnejo na 'T' in podatke shranite v datoteko *oceneT.csv*.

```
ime spol ocena opravljene_vaje datum_izpita
1 Tom m 6 1 2021-06-16
2 Tina z 7 1 2021-06-23
```

## Delo s šumniki

Pri delu z nizi RStudio uporablja privzeto kodiranje vašega operacijskega sistema. Preprost način, da preverite, ali R uporablja pravilno kodiranje, je, da v konzolo napišete "čebula". Nazaj morate dobiti enak niz in ne "cebula" ali celo "bula".

```
"čebula"
```

```
[1] "čebula"
```

Če R ne vrne istega niza, je potrebno nastaviti privzeto kodiranje na svojem operacijskem sistemu. Da vidimo, katero kodiranje je trenutno uporabljeno, lahko napišemo:

```
Sys.getlocale()
```

```
[1] "LC_COLLATE=Slovenian_Slovenia.1250;LC_CTYPE=Slovenian_Slovenia.1250;LC_MONETAR"
```

Pri tem izpisu vidimo, da uporabljamo sloveski nabor znakov CP1250. Če imate težave, lahko to začasno spremenite z ukazom:

```
Sys.setlocale(category = "LC_ALL", locale = "Slovenian_Slovenia.1250")
```

```
[1] "LC_COLLATE=Slovenian_Slovenia.1250;LC_CTYPE=Slovenian_Slovenia.1250;LC_MONETARY=Slovenian_Slovenia.1250"
```

Oziroma lahko na primer v OS Windows nastavite privzeto kodiranje pod “Settings -> Time & Language -> Region -> Regional Format”. Tukaj izberite “Slovenian (Slovenia)”.

Za slovenski OS Windows: “Nastavitve -> Ura in Jezik -> Regija -> Območne nastavitve” in izberite “Slovenščina (Slovenija)”.

## 3.2 Nizi

Zbirka paketov tidyverse vsebuje paket **stringr**, ki je namenjen delu z nizi. Vsi ukazi v paketu se začnejo s **str\_**, kar v kombinaciji s funkcionalnostjo avtomatskega dopolnjevanja omogoča hitro izbiro in pregled vseh funkcij.

V jeziku R lahko za definiranje niza uporabimo enojne (') ali dvojne navednice ("). R bo rezultat sicer vedno vrnil v dvojnih navednicah.

```
niz_dvojne_avednice <- "urejanje podatkov"
niz_enojne_avednice <- 'urejanje podatkov'
niz_dvojne_avednice
```

```
[1] "urejanje podatkov"
```

```
niz_enojne_avednice
```

```
[1] "urejanje podatkov"
```

Če želimo znotraj niza uporabiti dvojne navednice, lahko za definicijo uporabljamo enojne in obratno:

```
cit1 <- "Jure je rekel: 'Citiram Janeza'."
cit2 <- 'In Janez je rekel: "Citiram Jureta".'
cit1
```

```
[1] "Jure je rekel: 'Citiram Janeza'."
```

```
cit2
```

```
[1] "In Janez je rekel: \"Citiram Jureta\"."
```

Izpis v drugem primeru uporablja tako imenovan **escape character** (`\`), ki omogoča tudi vpis tabulaterjev (`\t`), znakov za novo vrstico (`\n`) in podobno. Vse znake, ki jih je tako možno zapisati, si lahko ogledate z ukazom `?""`.

Niz lahko vključuje tudi vse posebne znake, ki jih podpira Unicode. To je standard za konsistentno kodiranje teksta, ki vsebuje preko 100000 znakov. Za vnos znakov unicode uporabite `\uxxxx`, kjer je `xxxx` koda znaka, ki ga lahko poiščete na strani <https://unicode.org/charts/>.

```
normalna_porazdelitev <- "Normal(\u03BC, 1)"
normalna_porazdelitev
```

```
[1] "Normal(μ, 1)"
```

Dolžino nizov poiščemo s funkcijo `str_length()`:

```
str_length(c("tri", "štiri", "", citat1, normalna_porazdelitev))
```

```
[1] 3 5 0 32 12
```

Za združevanje nizov se uporablja `str_c()`, ki mu lahko dodamo tudi niz `sep`, ki ga vrine med vse podane nize:

```
str_c("paradižnik", "bučke", "jabolka", "paprika", "jagode", sep = ", ")
```

```
[1] "paradižnik, bučke, jabolka, paprika, jagode"
```

Če funkciji podamo vektorje lahko združimo tudi več nizov naenkrat. Pri tem bo število dobljenih nizov enako največjemu vektorju, manjši pa se bodo reciklirali, kar pomeni, da se bodo njihove vrednosti ponavljale, dokler ne dosežejo dolžine najdaljšega vektorja.

```
str_c("Podpoglavje ", c(1, 2, 3, 4), ": ",
 c("Nizi", "Faktorji", "Datumi", "Branje podatkov"))
```

```
[1] "Podpoglavje 1: Nizi" "Podpoglavje 2: Faktorji"
[3] "Podpoglavje 3: Datumi" "Podpoglavje 4: Branje podatkov"
```

Če želimo nize združiti zaporedno uporabimo `collapse`. Za izpis nizov brez escape znakov uporabite funkcijo `cat()`:

```
poglavja <- str_c("Podpoglavje ", c(1, 2, 3, 4), ": ",
 c("Nizi", "Faktorji", "Datumi", "Branje podatkov"), collapse = "\n")
poglavja
```

```
[1] "Podpoglavje 1: Nizi\nPodpoglavje 2: Faktorji\nPodpoglavje 3: Datumi\nPodpoglavje 4: Branje podatkov"
```

```
cat(poglavja)
```

```
Podpoglavje 1: Nizi
Podpoglavje 2: Faktorji
Podpoglavje 3: Datumi
Podpoglavje 4: Branje podatkov
```

Manjkajoče vrednosti lahko zamenjamo z `str_replace_na()`.

```
str_replace_na(c("ena", "dva", NA, "štiri"), replacement = "tri!")
```

```
[1] "ena" "dva" "tri!" "štiri"
```

### 3.2.1 Podnizi

Velikokrat pri delu z nizi potrebujemo le del niza ali pa želimo niz razdeliti na več manjših nizov. Poglejmo si na primeru niza " KUPI: paradižnik, bučke, jabolka, paprika, jagode. ", iz katerega želimo dobiti vektor sadja in zelenjave:

```
listek <- " KUPI: paradižnik, bučke, jabolka, paprika, jagode. "
```

Najprej odstranimo odvečne začetne in končne presledke. To storimo s `str_trim()`:

```
listek <- str_trim(listek)
listek
```

```
[1] "KUPI: paradižnik, bučke, jabolka, paprika, jagode."
```

Funkcija `str_sub()` vrne podniz glede na indekse znakov:

```
str_sub(listek, 7, 16) # Vrne znake med indeksom 7 do 16.
```

```
[1] "paradižnik"
```

```
str_sub(listek, 7) # Vrne znake od sedmega naprej.
```

```
[1] "paradižnik, bučke, jabolka, paprika, jagode."
```

```
str_sub(listek, -7) # Vrne zadnjih sedem znakov.
```

```
[1] "jagode."
```

```
str_sub(listek, -7, -2) # Vrne predzadnjih šest znakov.
```

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

S tem znanjem lahko odstranimo prvih šest znakov in zadnjega:

```
listek <- str_sub(listek, 7, -2)
listek
```

```
[1] "paradižnik, bučke, jabolka, paprika, jagode"
```

Sedaj, ko imamo elemente samo naštete in ločene z vejico, jih lahko preprosto razdelimo s funkcijo `str_split()`, ki ji podamo vzorec za ločevanje:

```
str_split(listek, pattern = ",", " ")
```

```
[[1]]
```

```
[1] "paradižnik" "bučke" "jabolka" "paprika" "jagode"
```

### 3.2.2 Iskanje vzorcev z regularnimi izrazi

#### 3.2.3 Uporaba regularnih izrazov

Uporabo regularnih izrazov si najprej pogljemo na preprostih podatkih, nato pa še na podatkih iz ankete.

Poiščimo sadje in zelenjavo, ki vsebuje šumnike:

```
seznam <- c("paradižnik", "bučke", "jabolka", "paprika", "jagode");
str_detect(seznam, "[čšž]")
```

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

Če želimo namesto logičnega vektorja dobiti dejanska imena elementov, uporabimo `str_subset()`:

```
str_subset(seznam, "[čšž]")
```

```
[1] "paradižnik" "bučke"
```

Poglejmo, koliko samoglasnikov vsebuje vsaka beseda v seznamu. Pozor, funkcija `str_count()` ne zazna ponovitev, če se le-te prekrivajo:

```
str_count(seznam, "[aeiou]")
```

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

```
str_count("ababa", "aba") # Ne zazna dveh ponovitev, ker se prekrivata.
```

```
[1] 1
```

S funkcijo `str_locate()` vrnemo indekse lokacij najdenih vzorcev, s `str_extract()` pa dobimo ta vzorec oziroma podniz. S `str_match()` dobimo podskupine.

Poiščimo podnize do prvega šumnika.

```
str_locate(seznam, "^.*[čšž]")
```

```
start end
[1,] 1 7
[2,] 1 3
[3,] NA NA
[4,] NA NA
[5,] NA NA
```

```
str_extract(seznam, "^.*[čšž]")
```

```
[1] "paradiž" "buč" NA NA NA
```

```
str_match(seznam, "^.*([čšž])")
```

```
[,1] [,2] [,3]
[1,] "paradiž" "paradi" "ž"
[2,] "buč" "bu" "č"
[3,] NA NA NA
[4,] NA NA NA
[5,] NA NA NA
```

V zadnjem primeru funkcija `str_match()` v prvem stolpcu vrne enako kot `str_extract()`, nato pa v naslednjih stolpcih vrne še vsako skupino posebej.

Preprosta in zelo uporabna pa je funkcija `str_replace()` oziroma `str_replace_all()`, s katero lahko zamenjamo dele nizov.

Prekrijmo šumnike z znaki "x":

```
str_replace(seznam, "[čšž]", "x")
```

```
[1] "paradixnik" "buxke" "jabolka" "paprika" "jagode"
```

Podamo lahko tudi zamenjavo za več ujemanj:

```
str_replace_all(seznam, c("č" = "c", "š" = "s", "ž" = "z"))
```

```
[1] "paradiznik" "bucke" "jabolka" "paprika" "jagode"
```

Poglejmo si sedaj še uporabo na realnih podatkih. Najprej naložimo podatke:

```
ds_jobs <- read.csv2("./data-raw/DS-jobs.csv")
ds_jobs <- tibble(ds_jobs)
ds_jobs
```

```
A tibble: 4,523 x 17
Gender Country Age EmploymentStatus CurrentJobTitle LanguageRecommen~
<chr> <chr> <int> <chr> <chr> <chr>
1 Female Austral~ 43 Employed full-time Business Analyst Python
2 Male Russia 33 Employed full-time Software Develop~ Python
3 Male Taiwan 26 Employed full-time Software Develop~ Python
4 Male United ~ 25 Employed part-time Researcher Python
5 Male United ~ 33 Employed full-time Scientist/Resear~ Matlab
6 Male Czech R~ 21 Employed part-time Other Python
7 Male Russia 22 Employed full-time Data Analyst Python
8 Male Netherl~ 51 Employed full-time Engineer R
9 Male Colombia 34 Employed full-time Data Scientist Python
10 Male Germany 41 Independent contrac~ Data Scientist Python
... with 4,513 more rows, and 11 more variables: FormalEducation <chr>,
Major <chr>, CompensationAmount <dbl>, CompensationCurrency <chr>,
TimeGatheringData <int>, TimeModelBuilding <dbl>, TimeProduction <dbl>,
TimeVisualizing <dbl>, TimeFindingInsights <dbl>, TimeOtherSelect <int>,
ExchangeRate <dbl>
```



Najbolj uporabna funkcija pri delu z nizi v podatkih je `str_detect()`, ki nam omogoča izbiro podmnožice vrstic. Pogosto jo kombiniramo z regularnimi izrazi. Poiščimo tiste vrstice, kjer imajo anketiranci več službenih nazivov. Ti so ločeni z znakom `/`:

```
ds_jobs %>%
 select(CurrentJobTitle) %>%
 filter(str_detect(CurrentJobTitle, "/"))
```

```
A tibble: 946 x 1
CurrentJobTitle
<chr>
1 Software Developer/Software Engineer
2 Software Developer/Software Engineer
3 Scientist/Researcher
4 Software Developer/Software Engineer
5 Scientist/Researcher
6 Scientist/Researcher
7 Scientist/Researcher
8 Scientist/Researcher
9 Software Developer/Software Engineer
10 Scientist/Researcher
... with 936 more rows
```

Izberimo samo tiste vrstice, kjer se `CurrentJobTitle` začne z “Data” in se `EmploymentStatus` konča s “full-time”:

```
ds_jobs %>%
 select(EmploymentStatus, CurrentJobTitle) %>%
 filter(str_detect(CurrentJobTitle, "^Data"),
 str_detect(EmploymentStatus, "full-time$"))
```

```
A tibble: 1,679 x 2
EmploymentStatus CurrentJobTitle
<chr> <chr>
1 Employed full-time Data Analyst
2 Employed full-time Data Scientist
3 Employed full-time Data Scientist
4 Employed full-time Data Scientist
5 Employed full-time Data Analyst
6 Employed full-time Data Scientist
7 Employed full-time Data Scientist
8 Employed full-time Data Analyst
9 Employed full-time Data Analyst
```

```
10 Employed full-time Data Scientist
... with 1,669 more rows
```

Včasih so nizi v podatkih predolgi, oziroma lahko brez škode za pomen izbrisemo del niza. Najlažje to naredimo tako, da ga zamenjamo s praznim nizom. Izberimo podnize “Employed”:

```
ds_jobs <- ds_jobs %>%
 mutate(EmploymentStatus = str_replace(EmploymentStatus, "Employed ", ""))
ds_jobs %>%
 select(EmploymentStatus, CurrentJobTitle)
```

```
A tibble: 4,523 x 2
EmploymentStatus CurrentJobTitle
<chr> <chr>
1 full-time Business Analyst
2 full-time Software Developer/Software En~
3 full-time Software Developer/Software En~
4 part-time Researcher
5 full-time Scientist/Researcher
6 part-time Other
7 full-time Data Analyst
8 full-time Engineer
9 full-time Data Scientist
10 Independent contractor, freelancer, or self~~ Data Scientist
... with 4,513 more rows
```

Sedaj imamo eno manjšo nekonsistentnost v podatkih. Nekateri nizi se začnejo z veliko začetnico, nekateri pa z malo. Spremenimo vse na malo začetnico:

```
ds_jobs <- ds_jobs %>%
 mutate(EmploymentStatus = str_to_lower(EmploymentStatus))
ds_jobs %>%
 select(EmploymentStatus, CurrentJobTitle)
```

```
A tibble: 4,523 x 2
EmploymentStatus CurrentJobTitle
<chr> <chr>
1 full-time Business Analyst
2 full-time Software Developer/Software En~
3 full-time Software Developer/Software En~
4 part-time Researcher
5 full-time Scientist/Researcher
6 part-time Other
```

```
7 full-time Data Analyst
8 full-time Engineer
9 full-time Data Scientist
10 independent contractor, freelancer, or self-- Data Scientist
... with 4,513 more rows
```

S funkcijo `str_count()` lahko uporabimo, da anketirancem dodamo stolpec s številom službenih nazivov:

```
ds_jobs %>%
 select(CurrentJobTitle) %>%
 mutate(NumberOfJobTitles = str_count(CurrentJobTitle, "/") + 1)
```

```
A tibble: 4,523 x 2
CurrentJobTitle NumberOfJobTitles
<chr> <dbl>
1 Business Analyst 1
2 Software Developer/Software Engineer 2
3 Software Developer/Software Engineer 2
4 Researcher 1
5 Scientist/Researcher 2
6 Other 1
7 Data Analyst 1
8 Engineer 1
9 Data Scientist 1
10 Data Scientist 1
... with 4,513 more rows
```

Poskušajmo razdeliti `CurrentJobTitle` na `FirstTitle` in `SecondTitle`, pri tem naj se naziva ponovita, če ima anketiranec samo enega.

```
ds_jobs %>%
 select(CurrentJobTitle) %>%
 mutate(FirstTitle = str_extract(CurrentJobTitle, "[^/]*"),
 SecondTitle = str_extract(CurrentJobTitle, "[^/]*$"))
```

```
A tibble: 4,523 x 3
CurrentJobTitle FirstTitle SecondTitle
<chr> <chr> <chr>
1 Business Analyst Business Analyst Business Analyst
2 Software Developer/Software Engineer Software Developer Software Engineer
3 Software Developer/Software Engineer Software Developer Software Engineer
4 Researcher Researcher Researcher
5 Scientist/Researcher Scientist Researcher
```

```
6 Other Other Other
7 Data Analyst Data Analyst Data Analyst
8 Engineer Engineer Engineer
9 Data Scientist Data Scientist Data Scientist
10 Data Scientist Data Scientist Data Scientist
... with 4,513 more rows
```

### 3.3 Kategorične spremenljivke

Kategorične spremenljivke so spremenljivke, ki lahko zavzamejo samo končno mnogo vnaprej določenih vrednosti. Najbolj pogoste so:

- **Nominalne spremenljivke.** To so spremenljivke brez ureditve: tip izdelka ali znamka avtomobila.
- **Ordinalne spremenljivke.** To so spremenljivke, ki imajo smiselno ureditev: stopnja izobrazbe ali šolski uspeh.

V R uporabljamo za delo s kategoričnimi spremenljivkami t. i. **faktorje** (ang. **factor**). Ti se od spremenljivk tipa niz razlikujejo v tem, da se v spremenljivki hrani informacija o vseh možnih vrednostih. Faktorju ni mogoče dodati vrednosti, ki je ni v množici možnih vrednosti, kar služi kot varovalka pred napakami pri vnosu podatkov.

Poglejmo si uporabo faktorja na dveh preprostih primerih, kjer bomo sami ustvarili spremenljivki. Kasneje si bomo pogledali še delo s faktorji na primeru realnih podatkov, kjer bomo ponovno uporabili podatke o zaposlitvah na področju podatkovnih ved.

Kot primer nominalne spremenljivke si oglejmo krvne skupine. Obstajajo 4 možne vrednosti. Ustvarimo vektor krvnih skupin za pet različnih oseb:

```
krvne_skupine <- c("B", "B", "O", "AB", "BA")
```

Sedaj je ta vektor shranjen kot niz. Kaj so slabosti takšnega shranjevanja kategoričnih podatkov? Prvič, nimamo nobenega varovala pred tipkarskimi napakami – R je zadnji vnos prebral kot *BA* in ga tako tudi shranil, čeprav ta krvna skupina ne obstaja:

```
krvne_skupine
```

```
[1] "B" "B" "O" "AB" "BA"
```

Pri ročnem vnosu podatkov hitro pride do tipkarskih napak in načeloma ne vemo, ali je avtor podatkov v tem primeru želel vnesti A, B ali AB.

Drugič, če želimo urediti to spremenljivko, se bodo vrednosti razvrstile po abecedi:

```
sort(krvne_skupine)

[1] "AB" "B" "B" "BA" "O"
```

Morda pa bi bilo bolj smiselno urediti po standardni ureditvi, torej A, B, AB, O.

Da se izognemo tem težavam, je bolje, če spremenljivko, za katero vemo, da bo zasedla eno od vnaprej določenih vrednosti, shranimo kot faktor. V R za to uporabimo funkcijo `factor()`. Naredimo faktor iz spremenljivke `dan_v_tednu`:

```
krvne_skupine_fac <- factor(krvne_skupine)
krvne_skupine_fac
```

```
[1] B B O AB BA
Levels: AB B BA O
```

Opazimo, da je sedaj spremenljivka drugačnega tipa, saj hrani tudi informacijo o možnih vrednostih oziroma ravneh (ang. levels). Ampak v tem primeru so te ravni napačne (ne zajame vseh 4 krvnih skupin, poleg tega pa vsebuje tudi eno napačno vrednost). Funkcija `factor()` privzeto kot ravni nastavi vse vrednosti v podani spremenljivki. Če želimo, ji lahko podamo dodaten argument `levels`, kjer ročno določimo, katere ravni bodo v spremenljivki. V kolikor to vemo vnaprej, je dobra praksa, da podamo tudi ta argument:

```
krvne_skupine_fac <- factor(krvne_skupine, levels = c("A", "B", "AB", "O"))
krvne_skupine_fac
```

```
[1] B B O AB <NA>
Levels: A B AB O
```

```
sort(krvne_skupine_fac)
```

```
[1] B B AB O
Levels: A B AB O
```

Opazimo dvojce: sedaj lahko spremenljivko uredimo glede na standardno notacijo in nesmiselne vrednosti se spremenijo v NA. Faktorju torej ne moremo prirediti vrednosti, ki ni enaka eni izmed vrednosti v ravneh. Da dostopamo do vseh ravni faktorja, uporabimo funkcijo `levels()`:

```
levels(krvne_skupine_fac)
```

```
[1] "A" "B" "AB" "O"
```

Včasih imajo kategorične spremenljivke tudi smiselno razvrstitev po velikosti, ki pa se običajno ne da numerično izmeriti. Kot primer si pogledjmo šolski uspeh, ki lahko zavzame 5 vrednosti. V kolikor želimo, da faktor hrani tudi informacijo o tem, da obstaja smiselna razvrstitev po velikosti, dodamo argument `ordered = TRUE`.

```
uspeh <- factor(c("odlično", "dobro", "dobro", "prav dobro"),
 levels = c("nezadostno", "zadostno", "dobro", "prav dobro", "odlično"),
 ordered = TRUE)
uspeh
```

```
[1] odlično dobro dobro prav dobro
Levels: nezadostno < zadostno < dobro < prav dobro < odlično
```

Opazimo, da imamo sedaj pri izpisu nivojev dodatno informacijo o razvrstitvi uspeha. V praksi nam to omogoča primerjavo, medtem ko tega pri faktorjih, ki nimajo razvrstitve po velikosti, ne moremo narediti.

```
uspeh[2] > uspeh[1]
```

```
[1] FALSE
```

```
krvne_skupine_fac[2] > krvne_skupine_fac[1]
```

```
Warning in Ops.factor(krvne_skupine_fac[2], krvne_skupine_fac[1]): '>' not
meaningful for factors
```

```
[1] NA
```

Poleg prednosti, ki smo jih že omenili, imajo faktorji posebno vlogo pri raznih statističnih modelih in modelih strojnega učenja. Nekatere metode eksplicitno zahtevajo faktorje. Prav tako razlikujejo med nominalnimi in ordinalnimi faktorji, kar se pozna na rezultatih. Ravno tako so faktorji pomembni v primeru, ko poznamo nabor možnih vrednosti a nekaterih pri meritvah še nismo opazili. Pri izrisu želimo take vrednosti izpisati z vrednostjo 0 ali NA.

Poglejmo si uporabo faktorjev na realni podatkovni množici. Ponovno bomo delali s podatki o zaposlitvah na področju podatkovnih ved. Preberimo podatke in ponovimo nekaj operacij, ki smo jih spoznali na prvem predavanju. Prav tako bomo izbrali samo podmnožico stolpcev za bolj jasen prikaz.

```
library(tidyverse)
ds_jobs <- read_csv2("./data-raw/DS-jobs.csv") %>%
 select(Country, Age, EmploymentStatus,
 FormalEducation, CompensationAmount, ExchangeRate) %>%
 filter(!is.na(ExchangeRate)) %>%
 mutate(CompensationUSD = CompensationAmount * ExchangeRate) %>%
 filter(CompensationUSD <= 2500000, CompensationUSD >= 10000)
ds_jobs
```

```
A tibble: 3,186 x 7
Country Age EmploymentStatus FormalEducation CompensationAmo~ ExchangeRate
<chr> <dbl> <chr> <chr> <dbl> <dbl>
1 Austral~ 43 Employed full-t~ Bachelor's deg~ 80000 0.802
2 Russia 33 Employed full-t~ Bachelor's deg~ 1200000 0.0174
3 Taiwan 26 Employed full-t~ Master's degree 1100000 0.0333
4 United ~ 25 Employed part-t~ Bachelor's deg~ 20000 1
5 United ~ 33 Employed full-t~ Doctoral degree 100000 1
6 Russia 22 Employed full-t~ Bachelor's deg~ 624000 0.0174
7 Colombia 34 Employed full-t~ Master's degree 156000000 0.000342
8 Germany 41 Independent con~ I did not comp~ 150000 1.20
9 Poland 29 Employed full-t~ Master's degree 126000 0.281
10 United ~ 35 Employed full-t~ Doctoral degree 133000 1
... with 3,176 more rows, and 1 more variable: CompensationUSD <dbl>
```

Imamo 3 spremenljivke, ki bi jih bilo smiselno shraniti kot faktorje – Country, EmploymentStatus in FormalEducation. Pretvorimo jih v faktorje. Pri tem pustimo kar privzeto nastavitve, da se kot nivoji uporabijo vse vrednosti v stolpcih.

```
library(tidyverse)
ds_jobs <- ds_jobs %>%
 mutate(Country = factor(Country),
 EmploymentStatus = factor(EmploymentStatus),
 FormalEducation = factor(FormalEducation))
ds_jobs
```

```
A tibble: 3,186 x 7
Country Age EmploymentStatus FormalEducation CompensationAmo~ ExchangeRate
<fct> <dbl> <fct> <fct> <dbl> <dbl>
1 Austral~ 43 Employed full-t~ Bachelor's deg~ 80000 0.802
2 Russia 33 Employed full-t~ Bachelor's deg~ 1200000 0.0174
3 Taiwan 26 Employed full-t~ Master's degree 1100000 0.0333
4 United ~ 25 Employed part-t~ Bachelor's deg~ 20000 1
5 United ~ 33 Employed full-t~ Doctoral degree 100000 1
```

```
6 Russia 22 Employed full-t~ Bachelor's deg~ 624000 0.0174
7 Colombia 34 Employed full-t~ Master's degree 156000000 0.000342
8 Germany 41 Independent con~ I did not comp~ 150000 1.20
9 Poland 29 Employed full-t~ Master's degree 126000 0.281
10 United ~ 35 Employed full-t~ Doctoral degree 133000 1
... with 3,176 more rows, and 1 more variable: CompensationUSD <dbl>
```

```
levels(ds_jobs$Country)
```

```
[1] "Argentina" "Australia"
[3] "Belarus" "Belgium"
[5] "Brazil" "Canada"
[7] "Chile" "Colombia"
[9] "Czech Republic" "Denmark"
[11] "Egypt" "Finland"
[13] "France" "Germany"
[15] "Greece" "Hong Kong"
[17] "Hungary" "India"
[19] "Indonesia" "Iran"
[21] "Ireland" "Israel"
[23] "Italy" "Japan"
[25] "Kenya" "Malaysia"
[27] "Mexico" "Netherlands"
[29] "New Zealand" "Nigeria"
[31] "Norway" "Other"
[33] "Pakistan" "People 's Republic of China"
[35] "Philippines" "Poland"
[37] "Portugal" "Republic of China"
[39] "Romania" "Russia"
[41] "Singapore" "South Africa"
[43] "South Korea" "Spain"
[45] "Sweden" "Switzerland"
[47] "Taiwan" "Turkey"
[49] "Ukraine" "United Kingdom"
[51] "United States" "Vietnam"
```

```
levels(ds_jobs$EmploymentStatus)
```

```
[1] "Employed full-time"
[2] "Employed part-time"
[3] "Independent contractor, freelancer, or self-employed"
```



```
levels(ds_jobs$FormalEducation)
```

```
[1] "Bachelor's degree"
[2] "Doctoral degree"
[3] "I did not complete any formal education past high school"
[4] "I prefer not to answer"
[5] "Master's degree"
[6] "Professional degree"
[7] "Some college/university study without earning a bachelor's degree"
```

Kaj se zgodi, če želimo v tibblu spremeniti vrednost faktorja v neko vrednost, ki je ni v tem faktorju? Recimo, da želimo v 2. vrstici spremeniti državo v Narnia:

```
ds_jobs2 <- ds_jobs
ds_jobs2$Country[2] <- "Narnia"
```

```
Warning in `[<-.factor`(`*tmp`, 2, value = structure(c(2L, NA, 47L, 51L, :
invalid factor level, NA generated
```

```
ds_jobs2
```

```
A tibble: 3,186 x 7
Country Age EmploymentStatus FormalEducation CompensationAmo~ ExchangeRate
<fct> <dbl> <fct> <fct> <dbl> <dbl>
1 Austral~ 43 Employed full-t~ Bachelor's deg~ 80000 0.802
2 <NA> 33 Employed full-t~ Bachelor's deg~ 1200000 0.0174
3 Taiwan 26 Employed full-t~ Master's degree 1100000 0.0333
4 United ~ 25 Employed part-t~ Bachelor's deg~ 20000 1
5 United ~ 33 Employed full-t~ Doctoral degree 100000 1
6 Russia 22 Employed full-t~ Bachelor's deg~ 624000 0.0174
7 Colombia 34 Employed full-t~ Master's degree 156000000 0.000342
8 Germany 41 Independent con~ I did not comp~ 150000 1.20
9 Poland 29 Employed full-t~ Master's degree 126000 0.281
10 United ~ 35 Employed full-t~ Doctoral degree 133000 1
... with 3,176 more rows, and 1 more variable: CompensationUSD <dbl>
```

Ker Narnia ni nivo v faktorju, se nadomesti z NA. Če želimo dodati nov nivo temu faktorju, uporabimo `fct_expand()`:

```
ds_jobs2 <- ds_jobs %>%
 mutate(Country = fct_expand(Country, "Narnia"))
ds_jobs2$Country[2] <- "Narnia"
ds_jobs2
```

```
A tibble: 3,186 x 7
Country Age EmploymentStatus FormalEducation CompensationAmo~ ExchangeRate
<fct> <dbl> <fct> <fct> <dbl> <dbl>
1 Austral~ 43 Employed full-t~ Bachelor's deg~ 80000 0.802
2 Narnia 33 Employed full-t~ Bachelor's deg~ 1200000 0.0174
3 Taiwan 26 Employed full-t~ Master's degree 1100000 0.0333
4 United ~ 25 Employed part-t~ Bachelor's deg~ 20000 1
5 United ~ 33 Employed full-t~ Doctoral degree 100000 1
6 Russia 22 Employed full-t~ Bachelor's deg~ 624000 0.0174
7 Colombia 34 Employed full-t~ Master's degree 156000000 0.000342
8 Germany 41 Independent con~ I did not comp~ 150000 1.20
9 Poland 29 Employed full-t~ Master's degree 126000 0.281
10 United ~ 35 Employed full-t~ Doctoral degree 133000 1
... with 3,176 more rows, and 1 more variable: CompensationUSD <dbl>
```

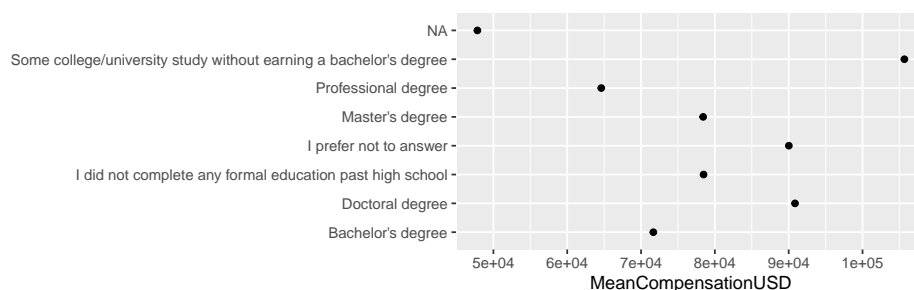
### 3.3.1 Sprememba razvrstitve faktorja

Kot smo omenili že pri krvnih skupinah, imajo velikokrat tudi faktorji, ki niso razvrščeni po velikosti, neko ustaljeno ureditev. Ureditev pa lahko tudi kasneje spremenimo. Ta operacija je običajno uporabna pri vizualizaciji. Poglejmo si, na primer, kako so plače povezane z izobrazbo. Za vizualizacijo rezultatov bomo uporabili razsevni diagram:

```
ds_jobs_agg <- ds_jobs %>%
 group_by(FormalEducation) %>%
 summarise(MeanCompensationUSD = mean(CompensationUSD))
ds_jobs_agg
```

```
A tibble: 8 x 2
FormalEducation MeanCompensationU~
<fct> <dbl>
1 Bachelor's degree 71665.
2 Doctoral degree 90856.
3 I did not complete any formal education past high school 78470.
4 I prefer not to answer 90023.
5 Master's degree 78411.
6 Professional degree 64614.
7 Some college/university study without earning a bachelor's~ 105675.
8 <NA> 47833.
```

```
ggplot(ds_jobs_agg, aes(x = FormalEducation, y = MeanCompensationUSD)) +
 geom_point() +
 coord_flip() +
 xlab("")
```



Ta graf je sicer zelo informativen, ampak bi s težavo hitro ugotovili, kako so nivoji faktorja razvrščeni glede na plačo. ggplot razvrsti vrednosti glede na to, kako so razvrščene v faktorju:

```
levels(ds_jobs$FormalEducation)
```

```
[1] "Bachelor's degree"
[2] "Doctoral degree"
[3] "I did not complete any formal education past high school"
[4] "I prefer not to answer"
[5] "Master's degree"
[6] "Professional degree"
[7] "Some college/university study without earning a bachelor's degree"
```

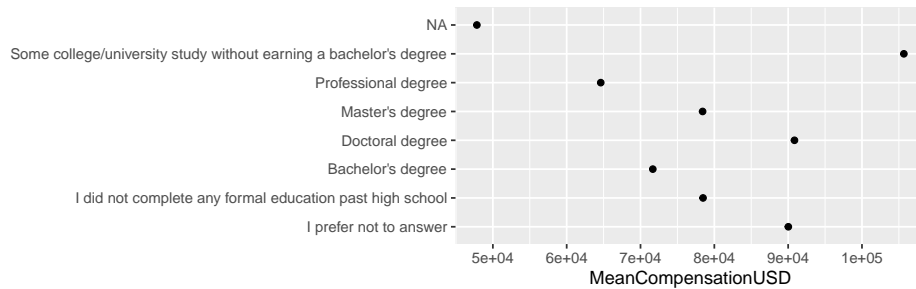
Morda bi bilo bolje tak graf urediti glede na vrednosti spremenljivke MeanCompensationUSD, za kar moramo določiti novo razvrstitev te spremenljivke. Za to obstaja v paketu **forcats**, ki je del tidyverse, funkcija `fct_reorder()`.

```
ggplot(ds_jobs_agg, aes(x = fct_reorder(FormalEducation, MeanCompensationUSD), y = MeanCompensationUSD)) +
 geom_point() +
 coord_flip() +
 xlab("")
```



Razvrstitev lahko uredimo tudi ročno s funkcijo `fct_relevel()`, ki ohrani privzeto razvrstitev tako, da podane nivoje premakne na začetek, oziroma lahko podamo argument `after`.

```
ggplot(ds_jobs_agg, aes(x = fct_relevel(FormalEducation, "I prefer not to answer", "I did not complete any formal education past high school", "Bachelor's degree", "Doctoral degree", "Master's degree", "Professional degree", "Some college/university study without earning a bachelor's degree", "NA")))
 geom_point() +
 coord_flip() +
 xlab("")
```



### 3.3.2 Preimenivanje obstoječih in določanje novih nivojev

Nivoje faktorjev lahko preimenujemo s funkcijo `fct_recode()`.

```
ds_jobs <- ds_jobs %>%
 mutate(EmploymentStatus = fct_recode(EmploymentStatus,
 "full-time" = "Employed full-time",
 "part-time" = "Employed part-time",
 "other" = "Independent contractor, freelancer, part-time",
))
head(ds_jobs$EmploymentStatus)
```

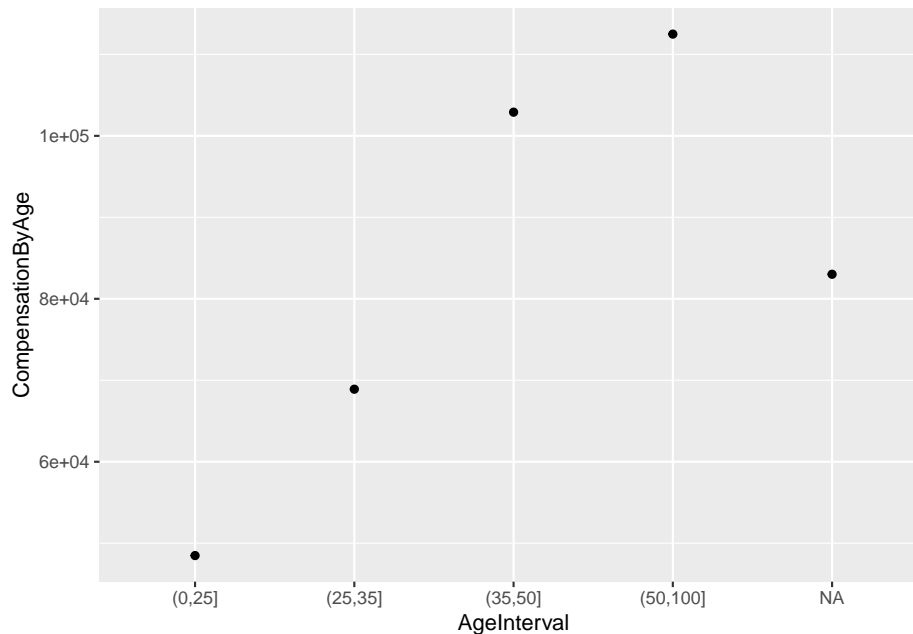
```
[1] full-time full-time full-time part-time full-time full-time
Levels: full-time part-time other
```

Sorodno lahko zgornjo funkcijo zamenjamo z `fct_collapse()`, ki lahko združi več nivojev v enega.

### 3.3.3 Razbitje numerične spremenljivke na intervale

Pogosto želimo numerično spremenljivko segmentirati na določene intervale. Na primer, pri določanju avtomobilskih zavarovalnih premij lahko zavarovance segmentiramo glede na starost. V R za to uporabimo funkcijo `cut()`. Razdelimo spremenljivko `Age` na intervale, kjer bodo osebe razdeljene do 25 let, nad 25 in do 35 let, nad 35 do 50 let, in nad 50 let.

```
ds_jobs <- ds_jobs %>%
 mutate(AgeInterval = cut(Age, breaks = c(0, 25, 35, 50, 100)))
ds_jobs_agg <- ds_jobs %>%
 group_by(AgeInterval) %>%
 summarise(CompensationByAge = mean(CompensationUSD))
ggplot(ds_jobs_agg, aes(x = AgeInterval, y = CompensationByAge)) + geom_point()
```



## 3.4 Datumi in ure

Delo z datumi in urami na prvi pogled morda deluje enostavno. Vendar pa zaradi različnih fizikalnih zakonitosti ali človeških konstruktov lahko pride do težav. Na primer, vsako leto nima 365 dni. Prav tako v nekaterih časovnih conah 3. ura zjutraj ne sledi vedno 2. uri, saj pride do premika ure.

Za delo z datumi bomo uporabljali paket **lubridate**. Glavni komponenti v tem paketu sta **datum** (date) in **čas** (time) ter združena komponenta **datum in čas** (datetime). S tem paketom lahko datume ustvarimo na 2 načina:

- 1) Z nizom:

```
library(lubridate)
ymd("2021-04-02")
```

```
[1] "2021-04-02"
```

```
ymd("2021/04/02")
```

```
[1] "2021-04-02"
```

```
ymd(20210402)
```

```
[1] "2021-04-02"
```

```
dmy("02.04.2021")
```

```
[1] "2021-04-02"
```

```
ymd_hms("2021-04-02 12:01:00") # Tipa datetime.
```

```
[1] "2021-04-02 12:01:00 UTC"
```

```
ymd(20210402, 20210403)
```

```
[1] "2021-04-02" "2021-04-03"
```

```
mdy("April 2nd, 2021") # Deluje za angleška imena mesecev.
```

```
[1] "2021-04-02"
```

2) S posameznimi komponentami:

```
make_date(2021, 4, 2)
```

```
[1] "2021-04-02"
```

```
make_datetime(2021, 4, 2, 12, 1, 0)
```

```
[1] "2021-04-02 12:01:00 UTC"
```

Opazimo, da pri datumu in času spremenljivka hrani tudi informacijo o časovnem pasu. Privzeto lubridate dela s časovnim pasom UTC (Coordinated Universal Time), ki je naslednik GMT (Greenwich Mean Time). Prednost tega časovnega pasu je, da se ne prilagaja spremembi ure v pomladnih in jesenskih mesecih. Te spremembe lahko privedejo do napak pri računanju z datumi in časi, tako da je računanje v UTC bolj varno. Seveda pa lahko ročno nastavimo drugi časovni pas z argumentom `tz`. Paket lubridate uporablja IANA časovne pasove (<https://www.iana.org/time-zones>), ki so definirani s kombinacijo celine in države. Za Ljubljano bi časovni pas nastavili tako:

```
ymd_hms("2021-04-02 12:01:00", tz = "Europe/Ljubljana")
```

```
[1] "2021-04-02 12:01:00 CEST"
```

Pomembno je, da vemo, v katerem časovnem pasu so bile opravljene meritve v naših podatkih, da lahko potem ustrezno pretvorimo spremenljivko v časovno. Seveda pa lahko tudi pretvarjamo časovne spremenljivke med časovnimi pasovi. Za to uporabimo funkcijo `with_tz()`. Vsakemu času v določenem časovnem pasu lahko priredimo nek čas v drugem časovnem pasu. V kolikor želimo bolj robustno računati z datumi in urami, lahko vedno datume pretvorimo v UTC čas, naredimo izračune in potem pretvorimo nazaj v lokalni časovni pas.

```
my_datetime <- ymd_hms("2021-04-02 12:01:00", tz = "Europe/Ljubljana")
my_datetime
```

```
[1] "2021-04-02 12:01:00 CEST"
```

```
my_datetime_UTC <- with_tz(my_datetime, tz = "UTC")
my_datetime_UTC
```

```
[1] "2021-04-02 10:01:00 UTC"
```

V ozdaju pa spremenljivki hranita isti čas. To lahko preverimo z:

```
my_datetime == my_datetime_UTC
```

```
[1] TRUE
```

V R je časovni pas namenjen samo izpisu datumov in časov. Vrednost spremenljivke ostane nespremenjena. To lahko preverimo tako, da odštejemo en datum od drugega, kar nam vrne razliko v času:

```
my_datetime - my_datetime_UTC
```

```
Time difference of 0 secs
```

V kolikor smo narobe prebrali datum v začetku (na primer, v podatkih je bil datum v UTC, prebrali pa smo v lokalnem času) zgornja pretvorba med časovnimi pasovi ni ustrezna, saj bomo s tem zajeli napačen čas. V tem primeru moramo uporabiti funkcijo `force_tz()`. Poizkusite kaj naredi ta funkcija, tako da z njo

pretvorijo `my_datetime` v UTC in potem izračunajte razliko, podobno kot smo to naredili zgoraj.

Kadar delamo sekvence datumov in časov, te upoštevajo premik ure in prehodov v naslednje dni. Poglejmo si prehod na poletni čas v letu 2021:

```
datetime_dst <- seq(ymd_hms("2021-03-28 00:00:00", tz = "Europe/Ljubljana"),
 ymd_hms("2021-03-28 04:00:00", tz = "Europe/Ljubljana"),
 by = "30 min")
datetime_dst
```

```
[1] "2021-03-28 00:00:00 CET" "2021-03-28 00:30:00 CET"
[3] "2021-03-28 01:00:00 CET" "2021-03-28 01:30:00 CET"
[5] "2021-03-28 03:00:00 CEST" "2021-03-28 03:30:00 CEST"
[7] "2021-03-28 04:00:00 CEST"
```

```
with_tz(datetime_dst, tz = "UTC")
```

```
[1] "2021-03-27 23:00:00 UTC" "2021-03-27 23:30:00 UTC"
[3] "2021-03-28 00:00:00 UTC" "2021-03-28 00:30:00 UTC"
[5] "2021-03-28 01:00:00 UTC" "2021-03-28 01:30:00 UTC"
[7] "2021-03-28 02:00:00 UTC"
```

Pozorni moramo biti tudi na kombiniranje datumov. V kolikor uporabimo funkcijo `c()`, vedno preverimo, v katerem časovnem pasu je rezultat.

### 3.4.1 Računanje z datumi in časi

Vsaka časovna spremenljivka, ki vsebuje datum in čas, je sestavljena iz komponent. Te so leto, mesec, dan, ura, minuta in sekunda. Za dostop do posameznih komponent imamo na voljo več funkcij:

- `year()`
- `month()`
- `mday()` Dan v mesecu.
- `wday()` Dan v tednu. Privzeto se začne z nedeljo. To lahko spremenimo z argumentom `week_start`.
- `hour()`
- `minute()`
- `second()`

Poglejmo sedaj kaj vračajo te funkcije:



```
x <- now()
x
```

```
[1] "2021-06-14 08:09:42 CEST"
```

```
year(x)
```

```
[1] 2021
```

```
month(x)
```

```
[1] 6
```

```
mday(x)
```

```
[1] 14
```

```
wday(x)
```

```
[1] 2
```

```
wday(x, week_start = 1)
```

```
[1] 1
```

```
hour(x)
```

```
[1] 8
```

```
minute(x)
```

```
[1] 9
```

```
second(x)
```

```
[1] 42.49473
```

S komponentami lahko tudi spreminjamo dele časovne spremenljivke:

```
mday(x) <- 5
x
```

```
[1] "2021-06-05 08:09:42 CEST"
```

Pri računanju s časovnimi enotami v lubridate poznamo tri razrede:

- **trajanja** (ang. duration). Čas v sekundah. Funkcije `dseconds()`, `dminutes()`, `ddays()`, `dweeks()` in `dyears()`. Pri trajanjih se vedno uporabi pretvorba, da ima vsak dan 24 ur in vsako leto 365.25 dni. Slednje predstavlja povprečno število dni v letu. Tako da bo funkcija `dyears(4)` vedno vrnila število sekund, ki ustreza 4x365.25 dnem, ki imajo vsak po 24 ur.
- **obdobja** (ang. period). Čas v človeških enotah kot je na primer teden. Funkcije `seconds()`, `minutes()`, `days()`, `weeks()`, `months()` in `years()`.
- **intervali** (ang. interval). Časovni interval med dvema točkama.

Pozoren bralec je opazil, da pri trajanjih nismo navedli funkcije za mesece. To pa zato, ker imajo meseci lahko 28, 29, 30 ali 31 dni. Vsekakor bi pri izbiri osnovne enote za trajanja prišlo do neke arbitrarne odločitve, koliko dni vzamemo privzeto. 30 ali 31? V vsakem primeru bo vsaj polovica mesecev imela napačno trajanje. Pri dnevih in letih si lažje privoščimo posplošitev.

```
ddays(1)
```

```
[1] "86400s (~1 days)"
```

```
days(1)
```

```
[1] "1d 0H 0M 0S"
```

Poglejmo si preprost primer, kako dodati določena obdobja:

```
my_datetime <- ymd_hms("2021/06/08 11:05:30", tz = "Europe/Ljubljana")
my_datetime + ddays(1)
```

```
[1] "2021-06-09 11:05:30 CEST"
```

```
my_datetime + days(1)
```

```
[1] "2021-06-09 11:05:30 CEST"
```

```
my_datetime + dminutes(120)
```

```
[1] "2021-06-08 13:05:30 CEST"
```

```
my_datetime + minutes(120)
```

```
[1] "2021-06-08 13:05:30 CEST"
```

```
my_datetime + months(2)
```

```
[1] "2021-08-08 11:05:30 CEST"
```

Trajanja in obdobja so si očitno zelo podobni, ampak imajo eno veliko razliko, kadar računamo z dnevi, tedni in leti. Prvič, kadar bomo uporabljali `dyears()` lahko hitro pride do težave, saj bomo prišteli 0.25 dneva. Poglejmo si to na primeru:

```
my_datetime + years(1)
```

```
[1] "2022-06-08 11:05:30 CEST"
```

```
my_datetime + dyears(1)
```

```
[1] "2022-06-08 17:05:30 CEST"
```

Opazimo, da smo prišteli 6 dodatnih ur. Drugič, kaj se zgodi, kadar prištejemo teden ali dan v času, ko pride do premika ure. Premik ure se je po lokalnem času zgodil 28. 3. 2021 ob 2 zjutraj.

```
my_datetime <- ymd_hms("2021/03/27 11:05:30", tz = "Europe/Ljubljana")
my_datetime + ddays(1)
```

```
[1] "2021-03-28 12:05:30 CEST"
```

```
my_datetime + days(1)
```

```
[1] "2021-03-28 11:05:30 CEST"
```

```
my_datetime + dweeks(1)
```

```
[1] "2021-04-03 12:05:30 CEST"
```

```
my_datetime + weeks(1)
```

```
[1] "2021-04-03 11:05:30 CEST"
```

Funkcija `years()` deluje, kot bi pričakovali tudi na prestopnem letu:

```
my_datetime <- ymd_hms("2020/06/08 11:05:30", tz = "Europe/Ljubljana")
my_datetime + years(1)
```

```
[1] "2021-06-08 11:05:30 CEST"
```

S funkcijami trajanja in obdobji lahko tudi računamo, na primer:

```
dyears(2) + ddays(4) + dseconds(20)
```

```
[1] "63460820s (~2.01 years)"
```

```
days(2) + minutes(20) + seconds(120)
```

```
[1] "2d 0H 20M 120S"
```

```
5 * dminutes(20)
```

```
[1] "6000s (~1.67 hours)"
```

```
5 * minutes(20)
```

```
[1] "100M 0S"
```

Najbolje, da jo prikažemo na dveh primerih – premik ure in prestopno leto. Obdobja so bolj naraven prikaz za človeka.

```
my_datetime <- ymd_hms("2021/06/08 11:05:30", tz = "Europe/Ljubljana")
my_datetime + ddays(1)
```

```
[1] "2021-06-09 11:05:30 CEST"
```

```
my_datetime + days(1)

[1] "2021-06-09 11:05:30 CEST"

my_datetime + dminutes(120)

[1] "2021-06-08 13:05:30 CEST"

my_datetime + minutes(120)

[1] "2021-06-08 13:05:30 CEST"

my_datetime + dyears(1)

[1] "2022-06-08 17:05:30 CEST"

my_datetime + years(1)

[1] "2022-06-08 11:05:30 CEST"

my_datetime + months(2)

[1] "2021-08-08 11:05:30 CEST"
```

V bazi letalskih letov posodobimo podatke tako, da izračunamo niz, ki predstavlja odhod letala.

```
library(nycflights13)
head(flights)

A tibble: 6 x 20
ID year month day dep_time sched_dep_time dep_delay arr_time
<int> <int> <int> <int> <int> <int> <dbl> <int>
1 1 2013 1 1 517 515 2 830
2 2 2013 1 1 533 529 4 850
3 3 2013 1 1 542 540 2 923
4 4 2013 1 1 544 545 -1 1004
5 5 2013 1 1 554 600 -6 812
6 6 2013 1 1 554 558 -4 740
... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

```
flights_datetime <- flights %>% select(year, month, day, hour, minute) %>%
 mutate(DepartureTime = make_datetime(year, month, day, hour, minute))
flights_datetime
```

```
A tibble: 336,776 x 6
year month day hour minute DepartureTime
<int> <int> <int> <dbl> <dbl> <dtm>
1 2013 1 1 5 15 2013-01-01 05:15:00
2 2013 1 1 5 29 2013-01-01 05:29:00
3 2013 1 1 5 40 2013-01-01 05:40:00
4 2013 1 1 5 45 2013-01-01 05:45:00
5 2013 1 1 6 0 2013-01-01 06:00:00
6 2013 1 1 5 58 2013-01-01 05:58:00
7 2013 1 1 6 0 2013-01-01 06:00:00
8 2013 1 1 6 0 2013-01-01 06:00:00
9 2013 1 1 6 0 2013-01-01 06:00:00
10 2013 1 1 6 0 2013-01-01 06:00:00
... with 336,766 more rows
```

```
head(flights_datetime$DepartureTime)
```

```
[1] "2013-01-01 05:15:00 UTC" "2013-01-01 05:29:00 UTC"
[3] "2013-01-01 05:40:00 UTC" "2013-01-01 05:45:00 UTC"
[5] "2013-01-01 06:00:00 UTC" "2013-01-01 05:58:00 UTC"
```

Ali opazimo kakšno težavo? Datume smo prebrali v časovni coni UTC, podani pa so v lokalni časovni coni. Podatkov torej nismo pravilno pretvorili! Poizkusimo še enkrat:

```
flights_datetime <- flights %>% select(year, month, day, hour, minute) %>%
 mutate(DepartureTime = make_datetime(year, month, day, hour, minute,
 tz = "America/New_York"))
flights_datetime
```

```
A tibble: 336,776 x 6
year month day hour minute DepartureTime
<int> <int> <int> <dbl> <dbl> <dtm>
1 2013 1 1 5 15 2013-01-01 05:15:00
2 2013 1 1 5 29 2013-01-01 05:29:00
3 2013 1 1 5 40 2013-01-01 05:40:00
4 2013 1 1 5 45 2013-01-01 05:45:00
5 2013 1 1 6 0 2013-01-01 06:00:00
6 2013 1 1 5 58 2013-01-01 05:58:00
```

```
7 2013 1 1 6 0 2013-01-01 06:00:00
8 2013 1 1 6 0 2013-01-01 06:00:00
9 2013 1 1 6 0 2013-01-01 06:00:00
10 2013 1 1 6 0 2013-01-01 06:00:00
... with 336,766 more rows
```

```
head(flights_datetime$DepartureTime)
```

```
[1] "2013-01-01 05:15:00 EST" "2013-01-01 05:29:00 EST"
[3] "2013-01-01 05:40:00 EST" "2013-01-01 05:45:00 EST"
[5] "2013-01-01 06:00:00 EST" "2013-01-01 05:58:00 EST"
```

## 3.5 Shranjevanje in branje podatkov

### 3.5.1 Delo z binarnimi datotekami

V programskem jeziku R lahko spremenljivke shranjujemo in beremo (v trenutno sejo R) kot binarne objekte. Prednost tega shranjevanja pred shranjevanjem v csv ali excelove razpredelnice je, da se ohranijo tudi meta informacije, kot so recimo nivoji pri faktorjih. Shranjujemo na dva prevladujoča načina:

- 1) S kombinacijo funkcij `save()` in `load()`.
- 2) S kombinacijo funkcij `saveRDS()` in `readRDS()`.

Pomembna razlika med prvim in drugim pristopom je, da lahko s prvim shranimo več spremenljivk naenkrat, z drugim pa samo eno. Na prvi pogled bi torej pričakovali, da je prvi pristop boljši, oziroma bolj zaželen. Ampak ima eno pomembno slabost, zaradi katere predlagamo uporabo drugega pristopa.

Funkcija `save()` shrani spremenljivke v trenutni seji R v datoteko s končnico *rda* ali *RData*. To naredi tako, da shrani tako vrednost spremenljivke kot tudi ime spremenljivke. To pomeni, da ko bomo takšno datoteko prebrali v novo sejo R, bomo ustvarili spremenljivke z enakimi imeni, kot smo jih shranili. Pri tem pa lahko pride do težav. Recimo, da imamo v trenutni seji R že nek nabor spremenljivk nato pa želimo vanjo prenesti še neke druge spremenljivke, ki smo jih pred časom shranili s funkcijo `save()` v datoteko `saved-data.rda`. Kaj se bo zgodilo, če bo katera od spremenljivk v naši trenutni seji imela enako ime kot ena od spremenljivk shranjenih v `saved-data.rda`? R bo enostavno to spremenljivko prepisal s spremenljivko, ki se je nahajala v tej rda datoteki. Takšen postopek dela je lahko torej nevaren, saj lahko *nevede* izbrišemo obstoječe spremenljivke.

Predlagamo torej uporabo kombinacije funkcij `saveRDS()` in `readRDS()`. Funkcija `saveRDS()` shrani samo vrednost spremenljivke, ne pa tudi njenega imena, tako da ne pride do podobnih težav kot pri prvem pristopu. Končnica tako shranjenih datotek je *rds*. Poglejmo si uporabo teh funkcij:

```
x <- c(3, 6, 3, 7)
x
```

```
[1] 3 6 3 7
```

```
saveRDS(x, "./my-saved-files/my-x.rds")

x2 <- readRDS("./my-saved-files/my-x.rds")
x2
```

```
[1] 3 6 3 7
```

Vedno ko preberemo podatke v sejo R s funkcijo `readRDS()`, ji moramo prirediti ime, saj je v rds datoteki shranjena samo njena vrednost. S tem se tudi izognemo podobnim težavam kot pri funkcijah `save()` in `load()`.

Pomanjkljivost shranjevanja datotek rds je v tem, da lahko naenkrat shranimo samo 1 spremenljivko. Ampak to pomanjkljivost lahko zaobidemo tako, da več spremenljivk enostavno shranimo v seznam (`list()`). Poglejmo si sedaj na primer, kako bi shranili več spremenljivk.

```
tmp_list <- list(
 "x" = x,
 "some_datetime" = my_datetime,
 "ds_jobs" = ds_jobs
)

saveRDS(tmp_list, "./my-saved-files/my-list.rds")
read_list <- readRDS("./my-saved-files/my-list.rds")
names(read_list)
```

```
[1] "x" "some_datetime" "ds_jobs"
```

```
x2 <- read_list[["x"]]
x2
```

```
[1] 3 6 3 7
```

```
my_datetime2 <- read_list[["some_datetime"]]
my_datetime2
```

```
[1] "2021-06-08 11:05:30 CEST"
```



```
ds_jobs2 <- read_list[["ds_jobs"]]
ds_jobs2
```

```
A tibble: 3,186 x 8
Country Age EmploymentStatus FormalEducation CompensationAmo~ ExchangeRate
<fct> <dbl> <fct> <fct> <dbl> <dbl>
1 Austral~ 43 full-time Bachelor's deg~ 80000 0.802
2 Russia 33 full-time Bachelor's deg~ 1200000 0.0174
3 Taiwan 26 full-time Master's degree 1100000 0.0333
4 United ~ 25 part-time Bachelor's deg~ 20000 1
5 United ~ 33 full-time Doctoral degree 100000 1
6 Russia 22 full-time Bachelor's deg~ 624000 0.0174
7 Colombia 34 full-time Master's degree 156000000 0.000342
8 Germany 41 other I did not comp~ 150000 1.20
9 Poland 29 full-time Master's degree 126000 0.281
10 United ~ 35 full-time Doctoral degree 133000 1
... with 3,176 more rows, and 2 more variables: CompensationUSD <dbl>,
AgeInterval <fct>
```

### 3.5.2 Branje in shranjevanje z ostalimi datotekami

#### 3.5.2.1 Excel

Paket **openxlsx** omogoča delo z exelovimi razpredelnici. Poglejmo si preproste a uporabne ukaze. Za branje podatkov s tem paketom uprabljamo `read.xlsx()`, za shranjevanje datotek pa `write.xlsx()`.

Preberimo podatke ocen, ki smo jih imeli v testni skripti, in jo pretvorimo v tibble:

```
library(openxlsx)
ocene <- tibble(openxlsx::read.xlsx("./test-script/data-raw/student-performance.xlsx"))
ocene
```

```
A tibble: 649 x 12
school sex famsize Pstatus traveltime studytime internet absences G1
<chr> <chr> <chr> <chr> <dbl> <dbl> <chr> <dbl> <dbl>
1 GP F GT3 A 2 2 no 4 0
2 GP F GT3 T 1 2 yes 2 9
3 GP F LE3 T 1 2 yes 6 12
4 GP F GT3 T 1 3 yes 0 14
5 GP F GT3 T 1 2 no 0 11
6 GP M LE3 T 1 2 yes 6 12
7 GP M LE3 T 1 2 yes 0 13
```

```
8 GP F GT3 A 2 2 no 2 10
9 GP M LE3 A 1 2 yes 0 15
10 GP M GT3 T 1 2 yes 0 12
... with 639 more rows, and 3 more variables: G2 <dbl>, G3 <dbl>,
averageScore <dbl>
```

Privzeto funkcija `read.xlsx()` odpre prvi list v datoteki in poišče glavo razpredelnice. Če želimo druge liste moramo podati ime lista ali številko lista s parametrom `sheet`:

```
library(openxlsx)
ocene_portugalscina <- tibble(read.xlsx("./test-script/data-raw/student-performance.xlsx",
ocene_matematika <- tibble(read.xlsx("./test-script/data-raw/student-performance.xlsx",
ocene_portugalscina %>%
 select(school, G1, G2, G3, averageScore)
```

```
A tibble: 649 x 5
school G1 G2 G3 averageScore
<chr> <dbl> <dbl> <dbl> <dbl>
1 GP 0 11 11 7.33
2 GP 9 11 11 10.3
3 GP 12 13 12 12.3
4 GP 14 14 14 14
5 GP 11 13 13 12.3
6 GP 12 12 13 12.3
7 GP 13 12 13 12.7
8 GP 10 13 13 12
9 GP 15 16 17 16
10 GP 12 12 13 12.3
... with 639 more rows
```

```
ocene_matematika %>%
 select(school, G1, G2, G3, averageScore)
```

```
A tibble: 395 x 5
school G1 G2 G3 averageScore
<chr> <dbl> <dbl> <dbl> <dbl>
1 GP 5 6 6 5.67
2 GP 5 5 6 5.33
3 GP 7 8 10 8.33
4 GP 15 14 15 14.7
5 GP 6 10 10 8.67
6 GP 15 15 15 15
7 GP 12 12 11 11.7
```

```
8 GP 6 5 6 5.67
9 GP 16 18 19 17.7
10 GP 14 15 15 14.7
... with 385 more rows
```

Če ima datoteka veliko število listov, jih lahko shranimo v R-jev list:

```
library(openxlsx)
path <- "./test-script/data-raw/student-performance.xlsx"
imenaListov <- getSheetNames(path)
ocene_s <- list()
for(i in 1:length(imenaListov)) {
 ocene_s[[i]] <- tibble(read.xlsx(path))
}
ocene_s
```

```
[[1]]
A tibble: 649 x 12
school sex famsize Pstatus traveltime studytime internet absences G1
<chr> <chr> <chr> <chr> <dbl> <dbl> <chr> <dbl> <dbl>
1 GP F GT3 A 2 2 no 4 0
2 GP F GT3 T 1 2 yes 2 9
3 GP F LE3 T 1 2 yes 6 12
4 GP F GT3 T 1 3 yes 0 14
5 GP F GT3 T 1 2 no 0 11
6 GP M LE3 T 1 2 yes 6 12
7 GP M LE3 T 1 2 yes 0 13
8 GP F GT3 A 2 2 no 2 10
9 GP M LE3 A 1 2 yes 0 15
10 GP M GT3 T 1 2 yes 0 12
... with 639 more rows, and 3 more variables: G2 <dbl>, G3 <dbl>,
averageScore <dbl>
##
[[2]]
A tibble: 649 x 12
school sex famsize Pstatus traveltime studytime internet absences G1
<chr> <chr> <chr> <chr> <dbl> <dbl> <chr> <dbl> <dbl>
1 GP F GT3 A 2 2 no 4 0
2 GP F GT3 T 1 2 yes 2 9
3 GP F LE3 T 1 2 yes 6 12
4 GP F GT3 T 1 3 yes 0 14
5 GP F GT3 T 1 2 no 0 11
6 GP M LE3 T 1 2 yes 6 12
7 GP M LE3 T 1 2 yes 0 13
8 GP F GT3 A 2 2 no 2 10
```

```
9 GP M LE3 A 1 2 yes 0 15
10 GP M GT3 T 1 2 yes 0 12
... with 639 more rows, and 3 more variables: G2 <dbl>, G3 <dbl>,
averageScore <dbl>
```

V primeru, da je razpredelnica v excelovem listu premaknjena ali pa želimo prebrati le del podatkov, lahko uporabimo parametra `startRow` in `cols`. Preberimo samo ocene za matematiko:

```
path <- "./test-script/data-raw/student-performance.xlsx"
ocene_math <- tibble(read.xlsx(path, sheet = "Math scores", startRow = 1,
 cols = 9:12))
ocene_math
```

```
A tibble: 395 x 4
G1 G2 G3 averageScore
<dbl> <dbl> <dbl> <dbl>
1 5 6 6 5.67
2 5 5 6 5.33
3 7 8 10 8.33
4 15 14 15 14.7
5 6 10 10 8.67
6 15 15 15 15
7 12 12 11 11.7
8 6 5 6 5.67
9 16 18 19 17.7
10 14 15 15 14.7
... with 385 more rows
```

Da podatke shranimo, uporabimo `write.xlsx()`.

```
path <- "./test-script/data-raw/samo-ocene-math.xlsx"
write.xlsx(ocene_math, path)
```

### 3.5.2.2 SPSS

SPSS je program za statistično analizo. Datoteke povezane z SPSS imajo običajno končnico `.sav`. Za branje iz in uvažanje v SPSS lahko uporabimo paket **haven**. V mapi `data_raw` imamo podatke o osebah `osebe.sav`. Za uvoz teh podatkov v R uporabimo funkcijo `read_sav`.

```
library(haven)
podatki <- read_sav("./data-raw/osebe.sav")
podatki
```

```
A tibble: 5 x 5
Ime Visina Teza Spol Starost
<chr> <dbl> <dbl> <chr> <dbl>
1 Alen 171 70 m 41
2 Bojan 185 78 m 35
3 Cvetka 165 64 z 28
4 Dejan 190 95 m 52
5 Eva 152 67 z 22
```

Da shranimo data frame v sav datoteko, uporabimo funkcijo `write_sav()`. Shranimo sedaj data frame `iris` v sav datoteko:

```
write_sav(iris, "./data-clean/iris.sav")
```

Paket `haven` ima tudi funkcijo `read_por()` ki podpira starejše verzije datotek iz SPSS.

## 3.6 Ali želite izvedeti več?

V tem poglavju smo si ogledali operacije treh paketov `tidyversa`. Če želite hiter in dokaj celovit pregled nad vsemi ukazi priporočamo, da si za vsak paket ogledate tako imenovan **cheat sheet**:

- **stringr**: [https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R\\_strings.pdf](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf)
- **forcats**: <http://www.flutterbys.com.au/stats/downloads/slides/figure/factors.pdf>
- **lubridate**: [https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R\\_lubridate.pdf](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_lubridate.pdf)

Paket **stringr** uporablja paket **stringi**, ki ima na voljo več operacij, ki pa niso tako pogoste. Če se boste pri svojem delu srečali z zahtevnejšimi nalogami, lahko uporabite ta paket.

Za razumevanje regularnih izrazov in tudi avtomatsko razlago priporočamo stran <https://regex101.com/>. Lahko si tudi ogledate knjigo *Mastering Regular Expressions* (Friedl, 2006), ki obsežno obravnava snov regularnih izrazov.

## 3.7 Domača naloga

- 1) Odprite prvi list ocen v `student-performance.xlsx` v mapi `test-script/data-raw/`.

- Odprite le stolpce `school`, `absences`, `G1`, `G2`, `G3` in `averageScore`.
- Dodajte stolpec *točke*, ki ima vrednosti med 0 in 100, tako da ima najvišja pripadajoča vrednost v `averageScore` vrednost 100.
- Dodajte stolpec *ocena*, kjer so vrednosti za *točke* med 0 in 10 enake 1, 10 in 20 enake 2 itd.
- Dodajte stolpec *besedna\_ocena* tipa faktor, ki bo imel vrednosti glede na *oceno* in sicer ocene  $<6$  = “nezadostno”, 6 = “zadostno”, 7 in 8 = “dobro”, 9 = “zelo dobro” in 10 = “odlično”.
- Shranite podatke v datoteko `students-performance-math-normalized.xlsx` in v binarno datoteko `students-performance-math-normalized.rds`.
- Ponovno preberite obe datoteki, prvo v spremenljivko `podatki_excel` in drugo v `podatki_binarni`. Katero razliko opazite?

```
A tibble: 649 x 9
school absences G1 G2 G3 averageScore točke ocena besedna_ocena
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
1 GP 4 0 11 11 7.33 39.3 4 nezadostno
2 GP 2 9 11 11 10.3 55.4 6 zadostno
3 GP 6 12 13 12 12.3 66.1 7 dobro
4 GP 0 14 14 14 14 75 8 dobro
5 GP 0 11 13 13 12.3 66.1 7 dobro
6 GP 6 12 12 13 12.3 66.1 7 dobro
7 GP 0 13 12 13 12.7 67.9 7 dobro
8 GP 2 10 13 13 12 64.3 7 dobro
9 GP 0 15 16 17 16 85.7 9 zelo dobro
10 GP 0 12 12 13 12.3 66.1 7 dobro
... with 639 more rows
```

```
school absences G1 G2 G3 averageScore točke ocena besedna_ocena
1 GP 4 0 11 11 7.333333 39.28571 4 nezadostno
2 GP 2 9 11 11 10.333333 55.35714 6 zadostno
3 GP 6 12 13 12 12.333333 66.07143 7 dobro
4 GP 0 14 14 14 14.000000 75.00000 8 dobro
5 GP 0 11 13 13 12.333333 66.07143 7 dobro
6 GP 6 12 12 13 12.333333 66.07143 7 dobro
```

```
A tibble: 649 x 9
school absences G1 G2 G3 averageScore točke ocena besedna_ocena
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fct>
1 GP 4 0 11 11 7.33 39.3 4 nezadostno
2 GP 2 9 11 11 10.3 55.4 6 zadostno
3 GP 6 12 13 12 12.3 66.1 7 dobro
4 GP 0 14 14 14 14 75 8 dobro
5 GP 0 11 13 13 12.3 66.1 7 dobro
6 GP 6 12 12 13 12.3 66.1 7 dobro
```

```
7 GP 0 13 12 13 12.7 67.9 7 dobro
8 GP 2 10 13 13 12 64.3 7 dobro
9 GP 0 15 16 17 16 85.7 9 zelo dobro
10 GP 0 12 12 13 12.3 66.1 7 dobro
... with 639 more rows
```

- 2) Naložite paket `nycflights13` s podatki letov. V tej bazi je več razpredelnic in sicer **flights**, **airlines**, **planes**, **airports** in **weather**. Za razumevanje si pomagajte z `?flights`.

- a) Izpišite vsa letališča, ki imajo 3 ali več besed v svojem imenu.

```
A tibble: 837 x 8
faa name lat lon alt tz dst tzone
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 06A Moton Field Municipal Ai~ 32.5 -85.7 264 -6 A America/Chica~
2 09J Jekyll Island Airport 31.1 -81.4 11 -5 A America/New_Y~
3 0A9 Elizabethton Municipal A~ 36.4 -82.2 1593 -5 A America/New_Y~
4 0G6 Williams County Airport 41.5 -84.5 730 -5 A America/New_Y~
5 0G7 Finger Lakes Regional Ai~ 42.9 -76.8 492 -5 A America/New_Y~
6 0P2 Shoestring Aviation Airf~ 39.8 -76.6 1000 -5 U America/New_Y~
7 0S9 Jefferson County Intl 48.1 -123. 108 -8 A America/Los_A~
8 0W3 Harford County Airport 39.6 -76.2 409 -5 A America/New_Y~
9 10C Galt Field Airport 42.4 -88.4 875 -6 U America/Chica~
10 17G Port Bucyrus-Crawford Co~ 40.8 -83.0 1003 -5 A America/New_Y~
... with 827 more rows
```

- b) **Težja** Spremenite imena letališč tako, da odstranite besedi "Airport" ali "Airfield" iz konca imena.

```
A tibble: 1,458 x 8
faa name lat lon alt tz dst tzone
<chr> <chr> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 04G "Lansdowne " 41.1 -80.6 1044 -5 A America/New_York
2 06A "Moton Field Municipal~ 32.5 -85.7 264 -6 A America/Chicago
3 06C "Schaumburg Regional" 42.0 -88.1 801 -6 A America/Chicago
4 06N "Randall " 41.4 -74.4 523 -5 A America/New_York
5 09J "Jekyll Island " 31.1 -81.4 11 -5 A America/New_York
6 0A9 "Elizabethton Municipa~ 36.4 -82.2 1593 -5 A America/New_York
7 0G6 "Williams County " 41.5 -84.5 730 -5 A America/New_York
8 0G7 "Finger Lakes Regional~ 42.9 -76.8 492 -5 A America/New_York
9 0P2 "Shoestring Aviation " 39.8 -76.6 1000 -5 U America/New_York
10 0S9 "Jefferson County Intl" 48.1 -123. 108 -8 A America/Los_Ang~
... with 1,448 more rows
```

- c) Za vsak let sestavite besedno poročilo spodnje oblike, ki ga shranite v stolpec *Poročilo leta*.

Besedilo naj se glasi: “Let iz letalisca Newark Liberty Intl dne 22.01.13 ob 11:00 v letalisce George Bush Intercontinental je imel 2 minut zamude v odhodu in 11 v prihodu.”

Minute so lahko tudi negativne.

```
A tibble: 336,776 x 2
`Poročilo leta` dep_string
<chr> <dtm>
1 Let iz letalisca Newark Liberty Intl, dne 01.01.13, ob 0~ 2013-01-01 05:15:0
2 Let iz letalisca La Guardia, dne 01.01.13, ob 05:29 v le~ 2013-01-01 05:29:0
3 Let iz letalisca John F Kennedy Intl, dne 01.01.13, ob 0~ 2013-01-01 05:40:0
4 <NA> 2013-01-01 05:45:0
5 Let iz letalisca La Guardia, dne 01.01.13, ob 06:00 v le~ 2013-01-01 06:00:0
6 Let iz letalisca Newark Liberty Intl, dne 01.01.13, ob 0~ 2013-01-01 05:58:0
7 Let iz letalisca Newark Liberty Intl, dne 01.01.13, ob 0~ 2013-01-01 06:00:0
8 Let iz letalisca La Guardia, dne 01.01.13, ob 06:00 v le~ 2013-01-01 06:00:0
9 Let iz letalisca John F Kennedy Intl, dne 01.01.13, ob 0~ 2013-01-01 06:00:0
10 Let iz letalisca La Guardia, dne 01.01.13, ob 06:00 v le~ 2013-01-01 06:00:0
... with 336,766 more rows
```

- d) Časi odhodov in prihodov so v razpredelnici `flights` v lokalnem času. Dodajte spremenljivko `dep_time_utc`, ki bo vsebovala vrednost odhoda v časovnem pasu UTC. Zapis naj bo oblike integer, tako kot `dep_time`.

```
~~~
## # A tibble: 336,776 x 8
##   year month   day dep_time dep_time_utc origin dest tzone
##   <int> <int> <int>   <int> <chr>      <chr> <chr> <chr>
## 1  2013     1     1     517 1017      EWR   IAH   America/New_York
## 2  2013     1     1     533 1033      LGA   IAH   America/New_York
## 3  2013     1     1     542 1042      JFK   MIA   America/New_York
## 4  2013     1     1     544 1044      JFK   BQN   America/New_York
## 5  2013     1     1     554 1054      LGA   ATL   America/New_York
## 6  2013     1     1     554 1054      EWR   ORD   America/New_York
## 7  2013     1     1     555 1055      EWR   FLL   America/New_York
## 8  2013     1     1     557 1057      LGA   IAD   America/New_York
## 9  2013     1     1     557 1057      JFK   MCO   America/New_York
## 10 2013     1     1     558 1058      LGA   ORD   America/New_York
## # ... with 336,766 more rows
~~~
```

- e) Simuliramo. Vsi leti bodo zaradi nepričakovane prenove letališča po 23. 10. 2013 zamaknjeni za dva tedna. Izpišite posodobljene datume v originalni obliki (`year`, `month`, `day`).



```
A tibble: 63,797 x 20
ID year month day dep_time sched_dep_time dep_delay arr_time
<int> <int> <int> <int> <int> <int> <dbl> <int>
1 47500 2013 11 6 454 500 -6 632
2 47501 2013 11 6 512 517 -5 749
3 47502 2013 11 6 541 545 -4 837
4 47503 2013 11 6 544 545 -1 935
5 47504 2013 11 6 545 550 -5 930
6 47505 2013 11 6 546 545 1 856
7 47506 2013 11 6 549 600 -11 711
8 47507 2013 11 6 549 600 -11 900
9 47508 2013 11 6 550 600 -10 716
##10 47509 2013 11 6 552 600 -8 735
... with 63,787 more rows, and 12 more variables: sched_arr_time <int>,
arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
time_hour <dtm>
```



# Bibliography

- Cortez, P. and Morais, A. d. J. R. (2007). A data mining approach to predict forest fires using meteorological data.
- Dua, D. and Graff, C. (2017). UCI machine learning repository.
- Friedl, J. E. (2006). *Mastering regular expressions*. " O'Reilly Media, Inc."
- Yeh, I.-C. and Lien, C.-h. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480.