

Predavanje 08 – Odgovori na vprašanja

Statistični testi

Večina klasičnih statističnih testov in modelov je vgrajenih že v osnovni R. Poglejmo si uporabo treh izmed najbolj popularnih, t-testa, ANOVE in linearne regresije.

```
# Modelirajmo porabo goriva, pri čemer kot neodvisne spremenljivke uporabimo:  
# število cilindrov, konjsko moč in težo.  
lr <- lm(mpg ~ cyl + hp + wt, data = mtcars)  
summary(lr)
```

```
##  
## Call:  
## lm(formula = mpg ~ cyl + hp + wt, data = mtcars)  
##  
## Residuals:  
##      Min       1Q   Median       3Q      Max   
## -3.9290 -1.5598 -0.5311  1.1850  5.8986   
##  
## Coefficients:  
##              Estimate Std. Error t value Pr(>|t|)      
## (Intercept) 38.75179    1.78686   21.687 < 2e-16 ***  
## cyl         -0.94162    0.55092   -1.709 0.098480 .    
## hp          -0.01804    0.01188   -1.519 0.140015      
## wt          -3.16697    0.74058   -4.276 0.000199 ***  
## ---  
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
##  
## Residual standard error: 2.512 on 28 degrees of freedom  
## Multiple R-squared:  0.8431, Adjusted R-squared:  0.8263   
## F-statistic: 50.17 on 3 and 28 DF,  p-value: 2.184e-11
```

```
# t-test uporabimo za statistično primerjavo pričakovane širine listov  
# dveh vrst perunike.  
x_vir <- iris$Sepal.Width[iris$Species == "virginica"]  
x_ver <- iris$Sepal.Width[iris$Species == "versicolor"]  
  
t.test(x_vir, x_ver)
```

```
##  
## Welch Two Sample t-test  
##  
## data:  x_vir and x_ver  
## t = 3.2058, df = 97.927, p-value = 0.001819  
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
## 0.07771636 0.33028364
## sample estimates:
## mean of x mean of y
##      2.974      2.770
```

```
# ANOVA uporabimo za statistično primerjavo dolžine listov treh vrst perunike.
# Primerjamo, ali vrsta perunike vpliva na dolžino listov.
my_anova <- aov(Sepal.Length ~ Species, data = iris)
summary(my_anova)
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## Species      2  63.21  31.606   119.3 <2e-16 ***
## Residuals   147  38.96   0.265
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

ggplot2 – statistična signifikantnost

Včasih želimo rezultate statističnega testa prikazati kar na grafu. Poglejmo si sedaj primer t-testa v ggplot2. Za to bomo potrebovali še en paket **ggpubr** in funkcijo iz tega paketa **stat_compare_means**. Tej funkciji bomo poda

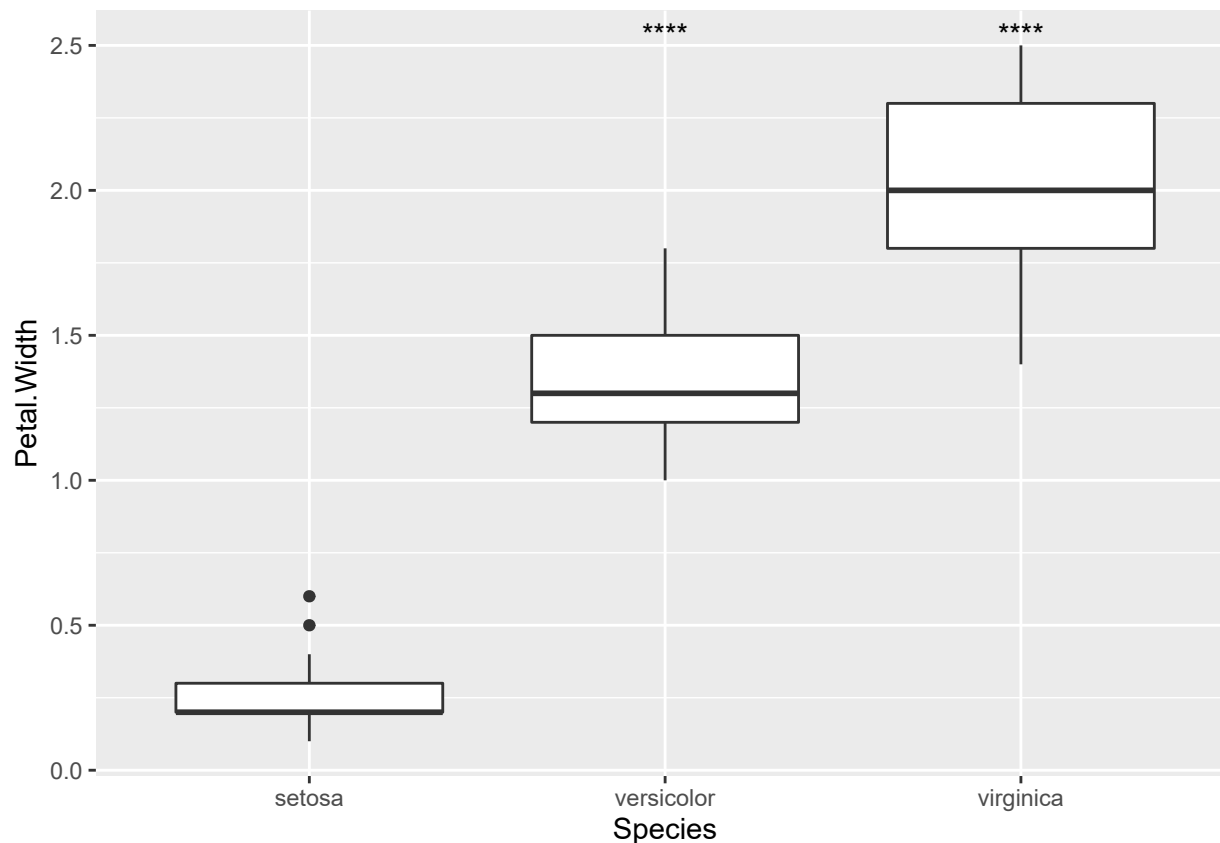
```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
library(ggpubr)
```

```
## Warning: package 'ggpubr' was built under R version 4.0.5
```

```
ggplot(iris, aes(x = Species, y = Petal.Width)) +
  geom_boxplot() +
  stat_compare_means(label = "p.signif", method = "t.test",
    ref.group = "setosa")
```



Prikaz točk in povprečja na grafu

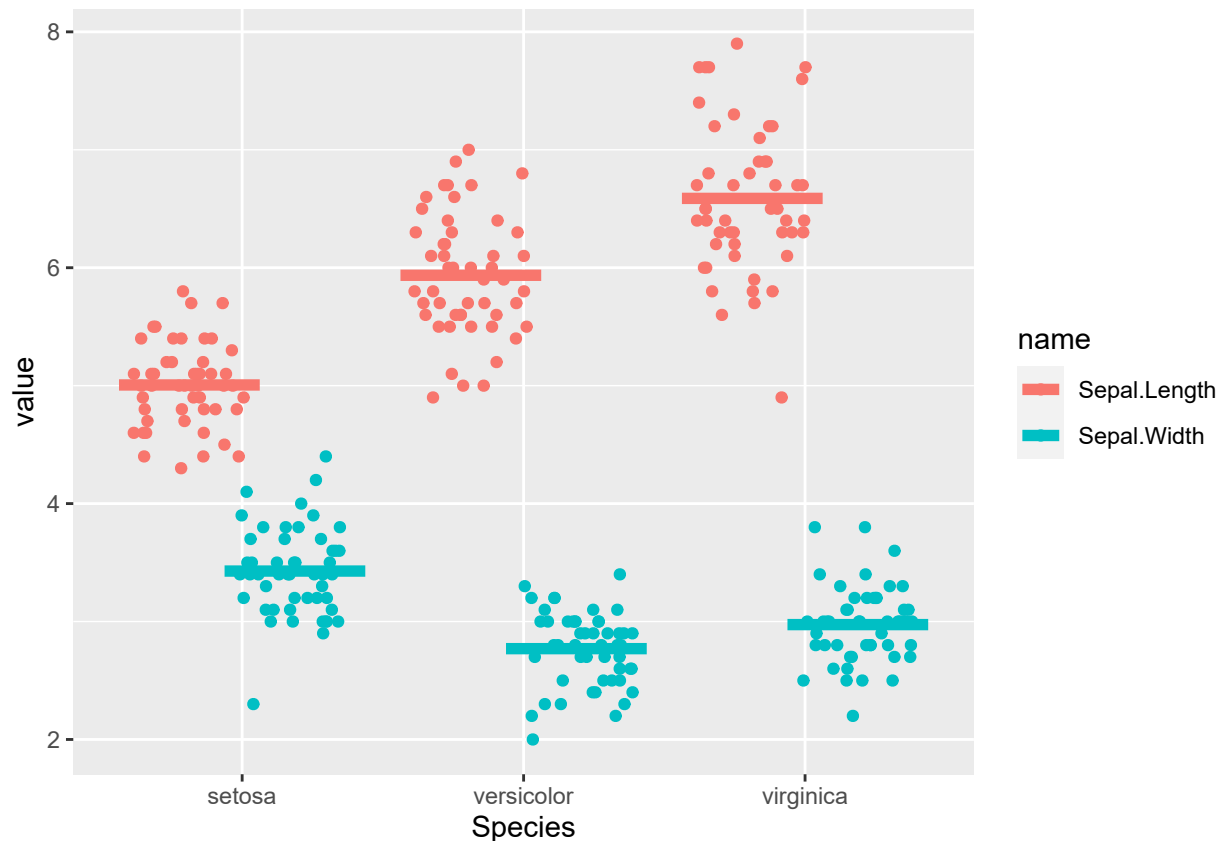
Poglejmo si še en zanimiv graf, kjer bomo prikazali točke in povprečja na istem grafu. Pogledali si bomo porazdelitve dolžin in širin listov različnih perunik. Najprej si pripravimo `data.frame`.

```
library(tidyr)
iris_longer <- iris[ , c("Sepal.Length", "Sepal.Width", "Species")]
iris_longer <- pivot_longer(iris_longer, Sepal.Length:Sepal.Width)
head(iris_longer)
```

```
## # A tibble: 6 x 3
##   Species name      value
##   <fct>   <chr>      <dbl>
## 1 setosa Sepal.Length  5.1
## 2 setosa Sepal.Width   3.5
## 3 setosa Sepal.Length  4.9
## 4 setosa Sepal.Width   3
## 5 setosa Sepal.Length  4.7
## 6 setosa Sepal.Width   3.2
```

Za izris povprečij s črto bomo potrebovali `geom_hline` iz paketa **ungeviz** (<https://wilkelab.org/ungeviz/index.html>). Za izris točk uporabimo pri `geom_point` argument `position = position_jitterdodge()`. To najprej loči dolžine in širine listov (`dodge`) in potem še nekoliko raztrosi točke (`jitter`), da je bolj pregledno, kje imamo več točk. Če ne bi uporabili tega, bi enostavno dobili prikazane vse točke v isti liniji.

```
library(ungeviz)
ggplot(iris_longer, aes(x = Species, y = value, color = name)) +
  geom_point(position = position_jitterdodge()) +
  stat_summary(
    fun = "mean",
    position = position_dodge(width = 0.75),
    geom = "hpline"
  )
)
```



Urejanje data.frame, izpis števila vrstic, ki ustreza pogoju

Zgoraj smo naložili podatke o avtih. Pa si pogledjmo, kako urediti `data.frame` glede na neko spremenljivko. Recimo, da nas zanima 10 avtomobilov z najmanj konjskimi močmi.

```
cars_decreasing <- mtcars[order(mtcars$hp), ]
cars_decreasing[1:10, ]
```

```
##      mpg  cyl  disp  hp drat    wt  qsec vs am gear carb
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52 1  1   4    2
## Merc 240D     24.4   4 146.7  62 3.69 3.190 20.00 1  0   4    2
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90 1  1   4    1
## Fiat 128      32.4   4  78.7  66 4.08 2.200 19.47 1  1   4    1
## Fiat X1-9     27.3   4  79.0  66 4.08 1.935 18.90 1  1   4    1
```

```
## Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2
## Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
## Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
## Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1
## Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
```

Kar smo naredili je, da smo najprej uredili `mtcars` glede na stolpec `hp`, ki predstavlja konjske moči. To smo naredili tako, da smo uporabili funkcijo `order`, ki nam vrne urejeno permutacijo indeksov. Na primer

```
order(c(5, 2, 3, 1))
```

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

Najmanjše število (1) je na četrtem mestu, torej je prva vrednost, ki jo vrne `order` 4. Drugo najmanjše število (2) je na drugem mestu, torej je druga vrednost, ki jo vrne `order` 2. Če želimo naraščajočo permutacijo, uporabimo argument `decreasing = TRUE`. Poglejmo si sedaj 10 avtomobilov z največ konjskimi močmi.

```
cars_increasing <- mtcars[order(mtcars$hp, decreasing = TRUE), ]
cars_increasing[1:10, ]
```

```
##           mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Maserati Bora 15.0  8 301.0 335 3.54 3.570 14.60 0 1  5   8
## Ford Pantera L 15.8  8 351.0 264 4.22 3.170 14.50 0 1  5   4
## Duster 360 14.3  8 360.0 245 3.21 3.570 15.84 0 0  3   4
## Camaro Z28 13.3  8 350.0 245 3.73 3.840 15.41 0 0  3   4
## Chrysler Imperial 14.7  8 440.0 230 3.23 5.345 17.42 0 0  3   4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82 0 0  3   4
## Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98 0 0  3   4
## Merc 450SE 16.4  8 275.8 180 3.07 4.070 17.40 0 0  3   3
## Merc 450SL 17.3  8 275.8 180 3.07 3.730 17.60 0 0  3   3
## Merc 450SLC 15.2  8 275.8 180 3.07 3.780 18.00 0 0  3   3
```

Velikokrat želimo neko podmnožico podatkov, ki ustreza nekemu pogoju. Na primer, želimo vse avtomobile, ki imajo več manj kot 100 konjskih moči.

```
cars_subset <- mtcars[mtcars$hp < 100, ]
```

Če nas zanima samo število takih vrstic, to dobimo tako, da najprej ustvarimo ustrezno podmnožico in nato preštejemo vrstice:

```
nrow(mtcars[mtcars$hp < 100, ])
```

```
## [1] 9
```

Manjkajoče vrednosti

Velikokrat se pri delu z realnimi podatki srečamo z manjkajočimi vrednostmi. V R so manjkajoče vrednosti označene z `NA` (not available). Poglejmo si vektor, ki vsebuje manjkajoče vrednosti.

```
x <- c(4, 6, 1, NA, 5, NA, 6)
```

Ali vektor (enako za stolpce v `data.frame` na primer) vsebuje manjkajoče vrednosti lahko preverimo s funkcijo `anyNA`.

```
anyNA(x)
```

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

Za posamezno vrednost preverimo ali je enaka NA z `is.na`.

```
is.na(x[1])
```

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

```
is.na(x[4])
```

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

Kaj se zgodi, če poizkusimo izračunati povprečje `x`?

```
mean(x)
```

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

Vrne NA. Če želimo, da nam R vseeno vrne povprečje vseh vrednosti, ki niso enake NA uporabimo argument `na.rm = TRUE`. Večina funkcij ki povzemajo številske vrednosti ima možnost podati ta argument. Alternativno bi lahko ročno izbrali podmnožico `x` kjer vrednosti niso NA in izračunali povprečje.

```
mean(x, na.rm = TRUE)
```

```
## [1] 4.4
```

```
mean(x[!is.na(x)])
```

```
## [1] 4.4
```

Nekonsistentni podatki

Poleg manjkajočih vrednosti se pogosto v podatkih pojavijo tudi nekonsistentnosti zaradi ročnega vnašanja. Na primer v numeričnem stolpcu se pojavijo števila ki imajo decimalno piko ali vejico, ali pa se pojavijo celo besede. V takem primeru je potrebnega nekaj ročnega dela s takšnimi stolpci. Poglejmo si datoteko **nekonsistentni_podatki.csv**, ki je v mapi `data_raw`.

```
podatki <- read.table("./data_raw/nekonsistentni_podatki.csv", dec = ",", sep = ";",  
                      quote = "\"", header = TRUE)  
head(podatki)
```

```
##      ime vrednost
## 1   Miha      4,6
## 2  Mojca      3.8
## 3  Matej       b
## 4 Matjaz       6
## 5    Tom       7
## 6   Anja       2
```

Z ukazom `str` lahko preverimo tipe stolpcev.

```
str(podatki)
```

```
## 'data.frame':  8 obs. of  2 variables:
## $ ime      : chr  "Miha" "Mojca" "Matej" "Matjaz" ...
## $ vrednost: chr  "4,6" "3.8" "b" "6" ...
```

Opazimo, da je R prebral oba stolpca kot besede (character). Če želimo stolpec `vrednost` spremeniti v numeričen, bomo morali narediti 2 stvari:

- 1) Ustrezno popraviti decimalne vejice v decimalne pike (saj R uporablja decimalno piko).
- 2) Pretvoriti stolpec v numeričnega.

Decimalne vejice bomo spremenili v decimalne pike z ukazom `gsub`. Ta funkcija se uporablja za zamenjavo dela besede z neko drugo besedo. Na primer:

```
beseda <- "Ne maram R!"
gsub(pattern = "Ne maram", replacement = "Obozujem", x = beseda)
```

```
## [1] "Obozujem R!"
```

S tem bomo sedaj zamenjali vejice s pikami v stolpcu `vrednost`:

```
podatki$vrednost <- gsub(pattern = ",", replacement = ".", x = podatki$vrednost)
```

Sedaj moramo samo še pretvoriti podatke v numerične s funkcijo `as.numeric`.

```
podatki$vrednost <- as.numeric(podatki$vrednost)
```

```
## Warning: NAs introduced by coercion
```

```
head(podatki)
```

```
##      ime vrednost
## 1   Miha      4.6
## 2  Mojca      3.8
## 3  Matej      NA
## 4 Matjaz      6.0
## 5    Tom      7.0
## 6   Anja      2.0
```

Opazimo, da je R vrstice, ki jih ne zna pretvoriti v številke (na primer tretjo vrstico, kjer imamo besedo v tem stolpcu), avtomatsko pretvoril v NA (manjkajoče vrednosti).

Branje podatkov v excelu za določene vrstice in stolpce

Velikokrat se srečamo z večimi tabelami na enem Excelovem listu. Kako preberemo točno določeno tabelo? V datoteki **podatki_premaknjeni.xlsx** imamo takšen list. Recimo, da želimo prebrati samo drugo tabelo, ki se nahaja v 14. vrstici in stolpcu E. To naredimo tako, da uporabimo argument `startRow = 14`. S tem bo `read.xlsx` začel brati podatke v 14. vrstici. Prebral bo vse podatke, do konca podatkov.

```
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.0.4
```

```
podatki <- read.xlsx("./data_raw/podatki_premaknjeni.xlsx", startRow = 14)
head(podatki)
```

```
##   artikel kolicina cena
## 1     zoga        1   10
## 2     kolo        5  100
## 3     rolka        2   30
```

Poglejmo si še datoteko **podatki_premaknjeni2.xlsx**, kjer imamo 2 tabeli v 14. vrstici. Recimo, da želimo prebrati samo drugo. Žal v tem primeru funkcija `read.xlsx` ne dopušča samo nastavitve prvega stolpca, v katerem začeti branje, ampak moramo funkciji ročno podati vse indekse stolpcev, katere želimo prebrati. Te podamo v argument `cols`. Enako lahko naredimo tudi za vrstice z argumentom `rows`.

```
podatki <- read.xlsx("./data_raw/podatki_premaknjeni2.xlsx", startRow = 14, cols = 9:10)
head(podatki)
```

```
##   delavec delovna_doba
## 1      Tim           10
## 2     Nina            5
## 3   Mateja            2
## 4   Zlatko           10
```

Avtomatsko generiranje poročil in shranjevanje tabel v Word

V tem poglavju bomo potrebovali paketa **rmarkdown** in **knitr**. Predlagamo, da tabelo v Word prenesemo s pomočjo Rmd datotek. Rmd datoteke, so datoteke, v katerih lahko združujemo tekst in R, kot izhod pa dobimo dokumente v pdf, docx ali html oblikah. S tem so tudi zelo primerni za avtomatsko generiranje poročil. Da v poročilu izpišemo tabelo, ki jo imamo shranjeno v R, lahko uporabimo ukaz `kable(<ime-tabele-v-R>)` (glej *Predavanje_08 - Porocilo.Rmd*). Da poženemo Rmd datoteko znotraj R skripte, lahko uporabimo klic

```
rmarkdown::render("<ime-Rmd-datoteke>",
  output_file = "<ime-izhodne-datoteke-s-koncnico>",
  output_format = "<format-izhodne-datoteke>")
```

Za uporabo tega klica glej *Predavanje_08 - Porocilo.R*. Kadar generiramo datoteko s tem klicem lahko Rmd datoteka, ki jo generiramo, dostopa do vseh spremenljivk, ki jih imamo v trenutni R seji. S tem lahko potem generiramo več poročil znotraj zanke, kot lahko vidimo v prej omenjeni skripti.

Dvojna glava (header)

Včasih dobimo podatke v kakšnih posebnih oblikah, na primer z dvema headerjema. Takšni podatki se na primer nahajajo v datoteki *dvojni_header.csv*. Kako te podatke pretvorimo v dolgo obliko? Najprej bomo prebrali vsak header posebej in jih združili v enoten header. Nato bomo prebrali podatke in jim priredili skupen header. Takšne podatke potem znamo pretvorit v dolgo obliko. Na koncu moramo samo še razdružiti oba headerja. Potrebovali bomo tudi funkcijo **na.locf** iz paketa **zoo**, ki vse NA vrednosti v vektorju nadomesti z zadnjo vrednostjo, ki ni bila enaka NA.

```
library(zoo)
# Preberemo samo prvo vrstico (nrow = 1), ki ima prvi header. Funkcija unlist
# data.frame spremeni v vektor.
head2 <- unlist(read.table("./data_raw/dvojni_header.csv", sep = ";",
                           quote = "", nrow = 1))

# Preberemo samo drugo vrstico (izpustimo prvo -- skip = 1).
head1 <- unlist(read.table("./data_raw/dvojni_header.csv", sep = ";",
                           quote = "", nrow = 1, skip = 1))

# Nadomestimo NA vrednosti v head2.
tmp <- na.locf(unlist(head2), na.rm = FALSE)

# Združimo oba headerja.
my_header <- paste(head1, tmp, sep = "_")

# Preberemo vrednosti v podatkih, jim pripravimo nova imena in pretvorimo v
# dolgo obliko.
podatki <- read.table("./data_raw/dvojni_header.csv", sep = ";", quote = "", skip = 2,
                      header = FALSE)
colnames(podatki) <- my_header
podatki_long <- pivot_longer(podatki, m_2018:f_2019)
head(podatki_long)

# Dodamo nova stolpca za spol in leto, tako da razdružimo stolpec name. To
# naredim os funkcijo gsub, ki v besedi nadomesti nek vzorec, v našem primeru
# vse znake za "_" (spol) ali pred "_" (leto). Na koncu izbrišemo še stolpec
# ime.
podatki_long$spol <- gsub("\\\\_.*", "", podatki_long$name)
podatki_long$leto <- gsub(".*\\\\_", "", podatki_long$name)
podatki_long$name <- NULL
```

Več data.frame kot vhodni podatek za ggplot

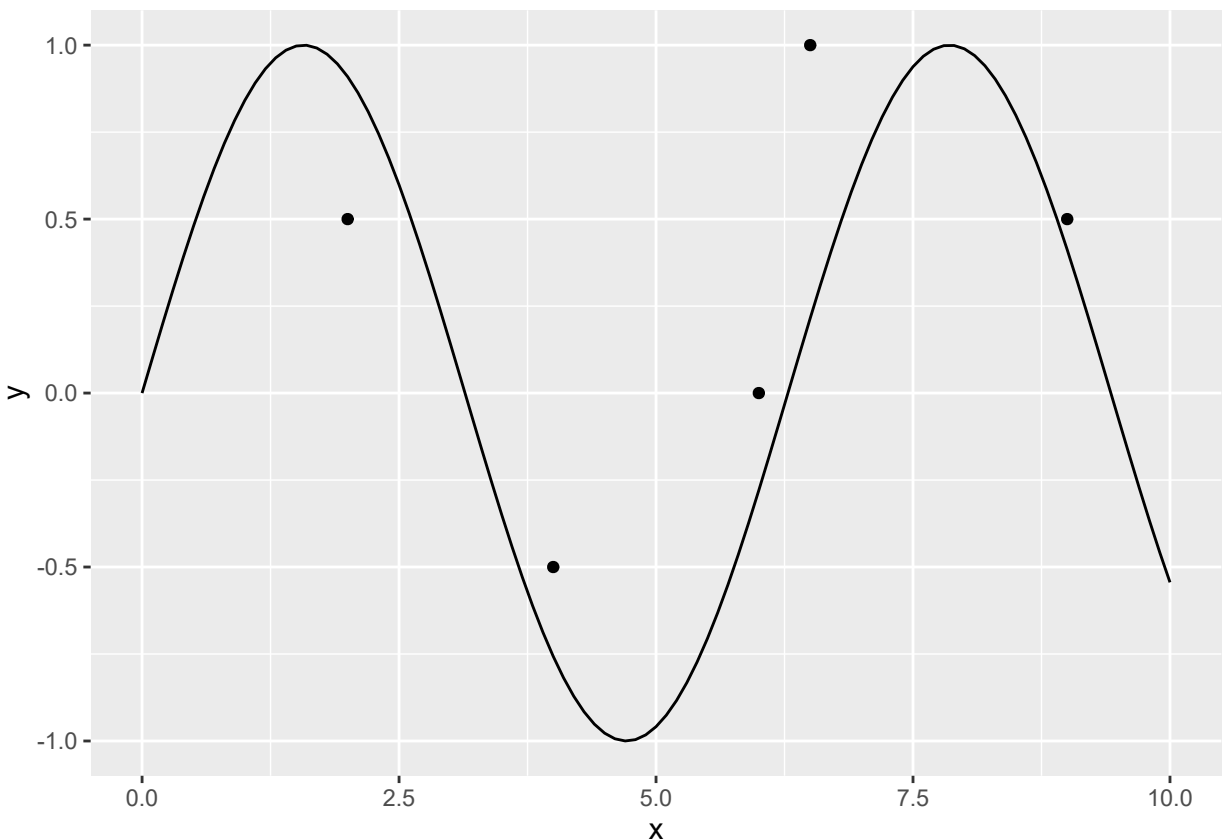
Včasih pridemo do situacije, kjer bi radi uporabili več različnih **data.frame** na enem grafu. Recimo, da imamo podatke o nekem pojavu v času, ki bi ga radi prikazali kot krivuljo, in podatke o nekih vzorcih iz tega pojava, ki bi jih radi predstavili kot točke. Generirajmo podatke, kjer bo pojav sinus vrednosti na osi x.

```
pojav_x <- seq(0, 10, by = 0.1)
pojav_y <- sin(pojav_x)
vzorci_x <- c(2, 4, 6, 6.5, 9)
vzorci_y <- c(0.5, -0.5, 0, 1, 0.5)
```

```
df1 <- data.frame(x = pojav_x, y = pojav_y)
df2 <- data.frame(x = vzorci_x, y = vzorci_y)
```

Sedaj imamo 2 `data.frame` enega za pojav in enega za vzorce. Namesto, da podamo podatke in estetike v funkcijo `ggplot`, jih v tem primeru podamo naravnost v **geom**. Torej bomo podali *df1* v geom `geom_line` in *df2* v geom `geom_point`.

```
ggplot() +
  geom_line(data = df1, mapping = aes(x = x, y = y)) +
  geom_point(data = df2, mapping = aes(x = x, y = y))
```



Pri tem moramo biti pozorni, da podamo argumente z imeni, ali pa v pravilnem vrstnem redu. Vrstni red je obraten kot pri argumentih, ki jih prejme `ggplot`, torej najprej estetike in potem podatki. Če uporabimo imena argumentov, potem to ne predstavlja težave.

Frekvenčno vzorčenje na isti časovni osi

Včasih imamo na voljo podatke, pri katerih vzorčimo vrednosti v določenih intervalih. Na primer 4Hz vzorčenje naredi 4 vzorce v eni sekundi. Podatki v tem primeru bi bili

- 1) Časovni zapis začetka vzorčenja.
- 2) Frekvenca vzorčenja (kolikokrat v sekundi vzorčimo).
- 3) Vektor vzorcev.

Lahko se zgodi, da različne vrednosti vzorčimo na različnih frekvencah. Če jih vse želimo pokazati na istem grafu z isto časovno osjo, moramo najprej ustvariti `data.frame` v katerem so vzorci in njihovi časovni zamiki od začetka vzorčenja. Na primer pri 4Hz vzorčenju bi ti bili 0.25, 0.5, 0.75, 1, 1.25 sekunde in tako naprej.

Poglejmo si to na konkretnem primeru, kjer imamo 2 vrednosti, recimo A in B. Obe smo začeli vzorčiti ob istem času, pri vrednosti A smo uporabili 64Hz vzorčenje, pri vrednosti B pa 4Hz vzorčenje. Podatki so v obliki csv (A.csv in B.csv). Prva vrstica predstavlja zakodiran čas začetka vzorčenja, druga vrstica frekvenco vzorčenja. Preostale vrstice predstavljajo vzorce. V takem primeru, bomo potrebovali 3 spremenljivke: čas začetka, frekvenco in vektor vzorcev.

Najprej preberemo samo prvo vrstico in shranimo v spremenljivko, ki predstavlja začetek vzorčenja. Da preberemo samo 1 vrstico csv datoteke uporabimo argument `nrows = 1`.

```
# Najprej preberemo samo prvo vrstico in shranimo v spremenljivko, ki
# predstavlja začetek vzorčenja. Da preberemo samo 1 vrstico csv datoteke
# uporabimo argument nrows = 1.
timestamp_A <- read.table(file = './data_raw/A.csv', header = F, nrows = 1)

# Izpustimo prvo vrstico (argument skip = 1), preberemo 1 vrstico in shranimo
# v spremenljivko, ki predstavlja frekvenco.
frequency_A <- read.table(file = './data_raw/A.csv', header = F, nrows = 1, skip = 1)

# Preberemo preostale podatke (vzorce) od 3. vrstice naprej (skip = 2).
y_A <- read.table(file = './data_raw/A.csv', header = F, skip = 2)
```

Pretvorimo frekvenco v časovni interval med vzorcema (`step_size_A`) in ustvarimo vektor časovnih zamikov vzorčenj od začetka vzorčenja (`t_A`). 1 sekundo torej delimo s frekvenco (dobimo razdaljo med vzorcema v deležih sekunde), nato pa naredimo časovno sekvenco od prvega vzorca do zadnjega vzorca, pri čemer je razlika med zaporednima številoma enaka časovnemu intervalu med vzorcema. Tukaj predpostavimo, da je prvi vzorec oddaljen od začetka za 1 frekvenco. V kolikor prvi vzorec dobimo na času 0, bi morali to ustrezno upoštevati pri kreiranju spremenljivke `t_A`:

```
step_size_A <- as.numeric(1 / frequency_A)
n_y_A <- nrow(y_A)
t_A <- seq(step_size_A, n_y_A * step_size_A, by = step_size_A)
```

S tem sedaj ustvarimo `data.frame`:

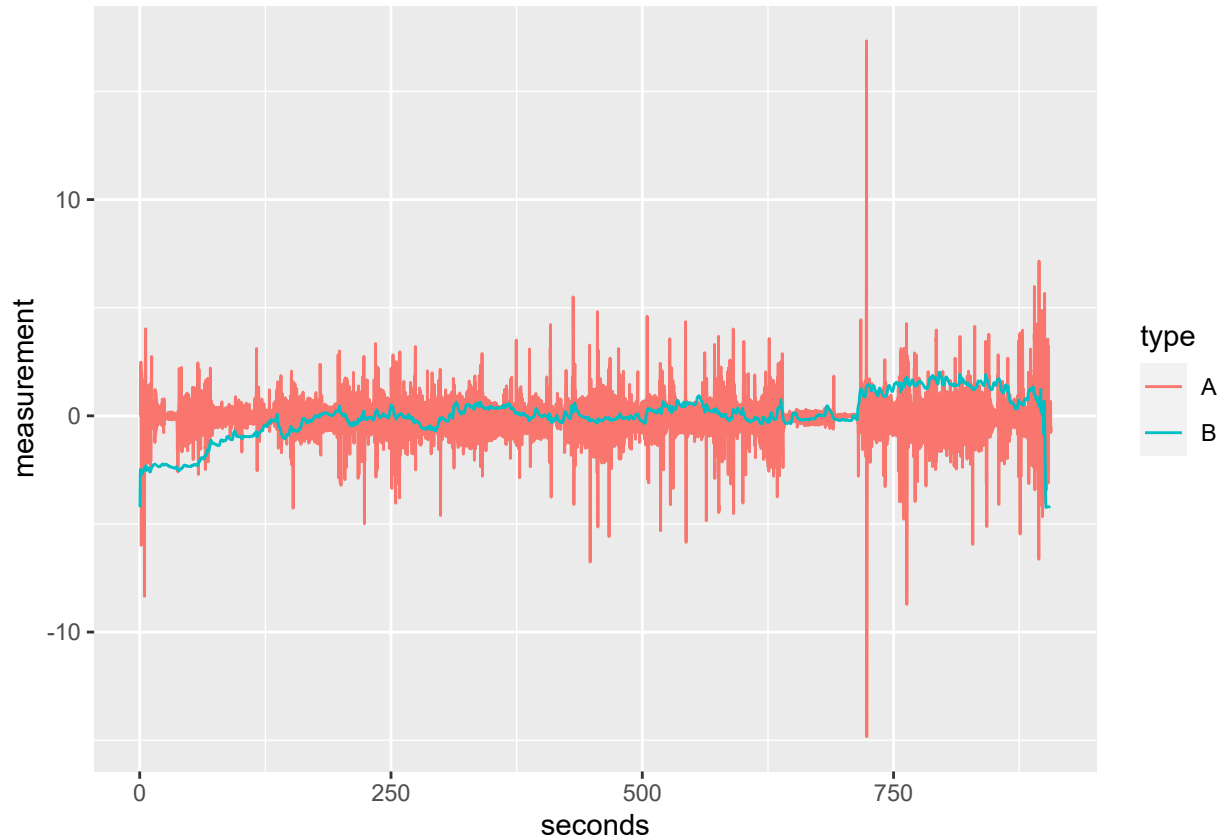
```
df1 <- data.frame(x = t_A, y = y_A, type = "A")
colnames(df1) <- c("seconds", "measurement", "type")
# Standardiziramo, da so vrednosti A in B na isti skali.
df1$measurement <- scale(df1$measurement)
```

Enako ponovimo še za vrednost B:

```
timestamp_B <- read.table(file = './data_raw/B.csv', header = F, nrows = 1)
frequency_B <- read.table(file = './data_raw/B.csv', header = F, nrows = 1, skip = 1)
y_B <- read.table(file = './data_raw/B.csv', header = F, skip = 2)
step_size_B <- as.numeric(1 / frequency_B)
n_y_B <- nrow(y_B)
t_B <- seq(step_size_B, n_y_B * step_size_B, by = step_size_B)
df2 <- data.frame(x = t_B, y = y_B, type = "B")
colnames(df2) <- c("seconds", "measurement", "type")
# Standardiziramo, da so vrednosti A in B na isti skali.
df2$measurement <- scale(df2$measurement)
```

Združimo `data.frame` in izrišemo vrednosti:

```
df <- rbind(df1, df2)
ggplot(df, aes(x = seconds, y = measurement, color = type)) + geom_line()
```



Paket dplyr

Na tej delavnici smo se naučili kako izbrati podmnožico vrstic in stolpcev `data.frame`, kako dodati stolpce in kako povzeti rezultate glede na kategorije. Vendar pa je osnovna različica R relativno toga pri izvajanju teh manipulacij. Za lažje in bolj pregledno delo z `data.frame` predlagamo uporabo paketa **dplyr**. Pogledali si bomo 4 najbolj pogoste funkcije iz tega paketa:

- 1) **filter**. Izberemo podmnožico vrstic.
- 2) **select**. Izberemo podmnožico stolpcev.
- 3) **mutate**. Dodamo stolpec.
- 4) **summarize**. Numerični povzetek preko kategorij.

Najbolje, da si ogledamo te funkcije v uporabi. Hkrati bomo prikazali uporabo z **dplyr** in osnovno različico R.

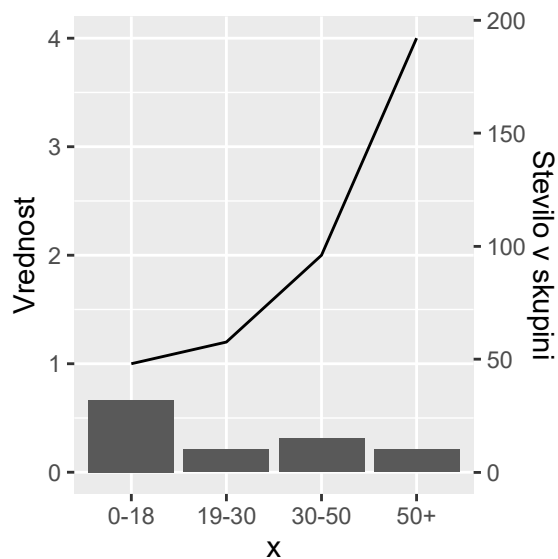
Kako v ggplot2 narediti graf z dvema y-osema?

Predlagamo, da ne uporabljate grafov z dvema y-osema: (<https://blog.datawrapper.de/dualaxis/>).

V kolikor zadeve ne morete rešiti drugače, pa se lahko poslužite sledečega trika:

```
x <- c("0-18", "19-30", "30-50", "50+")
y1 <- c(1, 1.2, 2, 4)
y2 <- c(32, 10, 15, 10)
df <- data.frame(x = x, y1 = y1, y2 = y2)

ggplot(df) +
  geom_line(aes(x = x, y = y1, group = 1)) +
  geom_bar(aes(x = x, y = y2 / 8 / 6), stat = "identity") +
  scale_y_continuous(
    name = "Vrednost",
    sec.axis = sec_axis(trans=~. * 8 * 6, name="Stevilo v skupini")
  )
```



Kar smo naredili je, da smo drugo spremenljivko `y2` transformirali, da je bila na enakem razponu, kot prva spremenljivka `y` (deljenje z 8). Potem smo jo še nekoliko zmanjšali, da smo dobili škatle pod črto, za lepši izgled (deljenje s 6, ta korak bi lahko preskočili). Potem moramo samo še dodati drugo os, kjer definiramo obratno transformacijo z `sec_axis(trans=~. * 8 * 6)`. Torej, če želite imeti dve osi, je najprej potrebno drugo spremenljivko ustrezno transformirati in nato dodati obratno transformacijo v argument `sec.axis` funkcije `scale_y_continuous`.

Kako združiti različne grafe?

Pogledali si bomo primer, kako združiti dva različna grafa. V bioinformatiki se pojavijo grafi, kjer imamo podatke o koordinatah pokritosti genoma pri sekvenciranju za več skupin, ki jih želimo prikazati kot območje pod krivuljo (`geom_area`). Poleg tega imamo za enak razpon na x osi tudi podatke o anotaciji genomske sekvence (torej za vsako anotacijo imamo začetno in končno točko na x osi). Poglejmo si sedaj, kako prikazati anotacijo in posamezne pokritosti vse na enem grafu. Za združevanje grafov bomo uporabili funkcijo `ggarrange` iz paketa `egg`.

```
# Najprej generiramo sintetične podatke, ki ilustrirajo pokritost genoma.
x <- seq(0, 7000, by = 1000)
```

```

y1 <- c(1000, 2000, 1000, 1000, 500, 1500, 2500, 2200)
y2 <- c(1000, 1800, 1200, 1200, 700, 1000, 1500, 1700)
df <- data.frame(x = x, y1 = y1, y2 = y2)
df_longer <- pivot_longer(df, y1:y2)

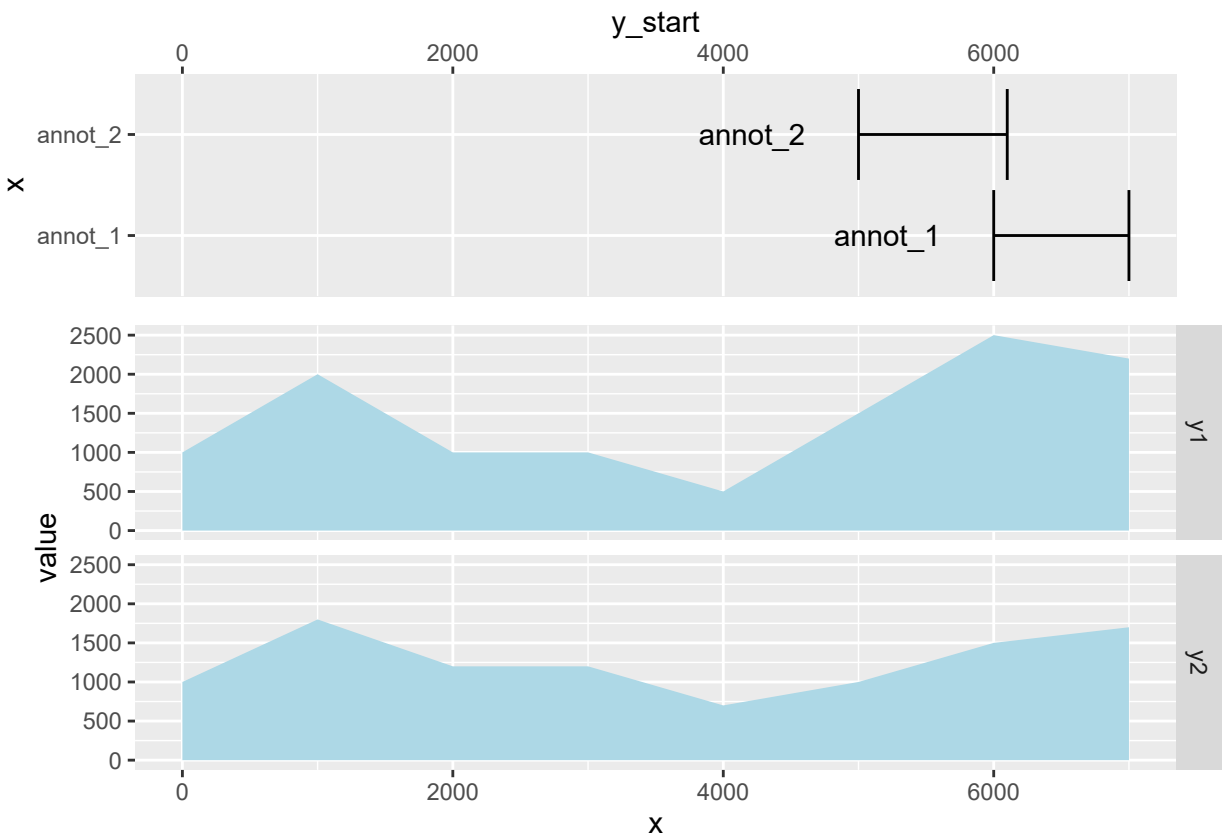
# Graf pokritosti genoma.
g1 <- ggplot(df_longer, aes(x = x, y = value)) +
  geom_area(fill = "lightblue") +
  facet_grid(name ~ .)

# Generiramo sintetične podatke, ki ilustrirajo anotacijo genoma.
tmp2 <- data.frame(x = c("annot_1", "annot_2"), y_start = c(6000, 5000),
  y_end = c(7000, 6100))

# Graf anotacije genoma. Potrebno je paziti, da nastavimo isto skalo s
# scale_y_continuous. Tukaj uporabimo y, namesto x, saj koordinate obrnemo s
# coord_flip.
g2 <- ggplot(tmp2, aes(x = x)) +
  geom_errorbar(aes(ymin = y_start, ymax = y_end)) +
  geom_text(aes(label = x, y = y_start), hjust = 1.5) +
  coord_flip() +
  scale_y_continuous(position = "right", limits = c(0, 7000))

# Združimo grafa.
egg::ggarrange(g2, g1, ncol = 1, heights = c(1, 2))

```



```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##   filter, lag

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
# filter izbere podmožico vrstic.
head(filter(mtcars, hp > 90)) # dplyr
```

```
##           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
```

```
# head(mtcars[mtcars$hp > 90, ]) # base R
```

```
# select izbere podmožico stolpcev.
head(select(mtcars, mpg, cyl, hp)) # dplyr
```

```
##           mpg cyl  hp
## Mazda RX4      21.0   6 110
## Mazda RX4 Wag  21.0   6 110
## Datsun 710     22.8   4  93
## Hornet 4 Drive  21.4   6 110
## Hornet Sportabout 18.7   8 175
## Valiant        18.1   6 105
```

```
# head(mtcars[, c("mpg", "cyl", "hp")]) # base R
```

```
# mutate doda stolpec.
head(mutate(mtcars, nov_stolpec = mpg + cyl * hp)) # dplyr
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb nov_stolpec
## 1  21.0     6  160 110 3.90 2.620 16.46 0  1   4    4      681.0
## 2  21.0     6  160 110 3.90 2.875 17.02 0  1   4    4      681.0
## 3  22.8     4  108  93 3.85 2.320 18.61 1  1   4    1      394.8
## 4  21.4     6  258 110 3.08 3.215 19.44 1  0   3    1      681.4
## 5  18.7     8  360 175 3.15 3.440 17.02 0  0   3    2     1418.7
## 6  18.1     6  225 105 2.76 3.460 20.22 1  0   3    1      648.1
```

```
# tmp <- mtcars # base R -- 3 vrstice
# tmp$nov_stolpec <- tmp$mpg + tmp$cyl * tmp$hp
# head(tmp)

# summarize deluje podobno kot aggregate torej povzame numerične vrednosti, glede na neke kategorije.
summarize(group_by(mtcars, cyl), mean_hp = mean(hp))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 3 x 2
##   cyl mean_hp
##   <dbl> <dbl>
## 1     4   82.6
## 2     6  122.
## 3     8  209.
```

```
# aggregate(mtcars$hp, by = list(mtcars$cyl), FUN = mean) # base R
```

ggplot2 – errorbar

Na statističnih grafih, ki vsebujejo opisne statistike, kot je npr. povprečje, pogosto prikažemo še negotovost v obliki standardnih odklonov ali standardnih napak. S knjižnico ggplot2 to storimo z uporabo `geom-a errorbar`. Pred tem moramo ustrezno pripraviti podatke tako, da dodamo še stolpec s spodnjo in zgornjo mejo napake. Če je napaka simetrična, potrebujemo le en stolpec.

```
data("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
```

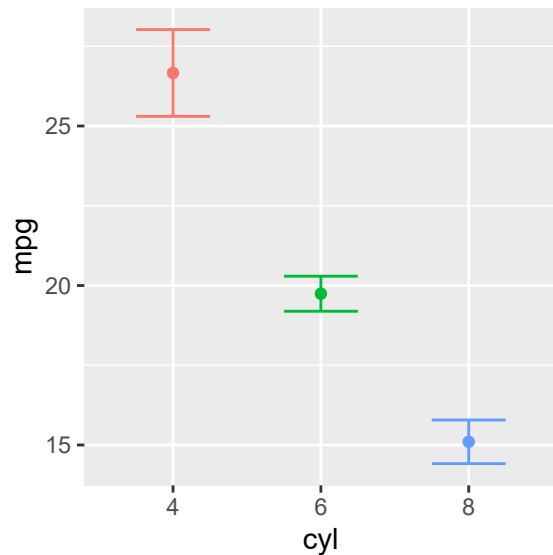
```
mus <- aggregate(mpg ~ cyl, mtcars, FUN = mean)
sds <- aggregate(mpg ~ cyl, mtcars, FUN = function(x) {sd(x) / sqrt(length(x))})
df <- cbind(mus, SE = sds$mpg)
df$cyl <- as.character(df$cyl)

head(df)
```

```
##   cyl      mpg      SE
## 1   4 26.66364 1.3597642
## 2   6 19.74286 0.5493967
## 3   8 15.10000 0.6842016
```



```
library(ggplot2)
ggplot(df, aes(x = cyl, y = mpg, colour = cyl)) +
  geom_point() +
  geom_errorbar(aes(ymin = mpg - SE, ymax = mpg + SE), width = 0.5) +
  theme(legend.position = "none")
```



Pripenjanje data.frame z različnimi stolpci

Včasih želimo združiti več data.frame, ki pa imajo enake samo nekatere stolpce. V tem primeru lahko uporabimo funkcijo `bind_rows` iz paketa `dplyr`.

```
library(dplyr)
height1 <- c(171, 185, 165)
weight1 <- c(70, 78, 64)
name1 <- c("Alen", "Bojan", "Cvetka")
height2 <- c(190, 152)
name2 <- c("Dejan", "Eva")
df1 <- data.frame(height = height1, weight = weight1, name = name1)
df2 <- data.frame(name = name2, height = height2)
df <- bind_rows(df1, df2)
```

Razhroščevanje (debugging) znotraj zanke

Omenili smo že, da lahko opazujemo dogajanje v zanki tako, da si z ukazom `print` sproti izpisujemo vrednosti spremenljivk v telesu zanke. Obstaja pa še en ukaz, ki nam omogoča zaustaviti delovanje zanke in pogledati vrednosti v zanki v določeni ponovitvi. To je ukaz `browser`, ki ga pokličemo znotraj zanke. Na primer:

```
for (i in 1:10) {
  x <- 2 * i
  y <- x + 5
```

```
if (y > 10) {  
    browser()  
}  
}
```

Ko tako zaustavimo zanko, se znajdemo v okolju browserja, in lahko dostopamo do spremenljivk, ki so znotraj zanke. V kolikor browserju v konzoli podamo vrednost **n** se browser pomakne na naslednjo ponovitev zanke. Z ukazom **c** mu povemo naj izvede celotno zanko do konca. Iz browserja lahko prekinemo izvajanje zanke in se vrnemo v osnovno konzolo s tipko **escape**.