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
```

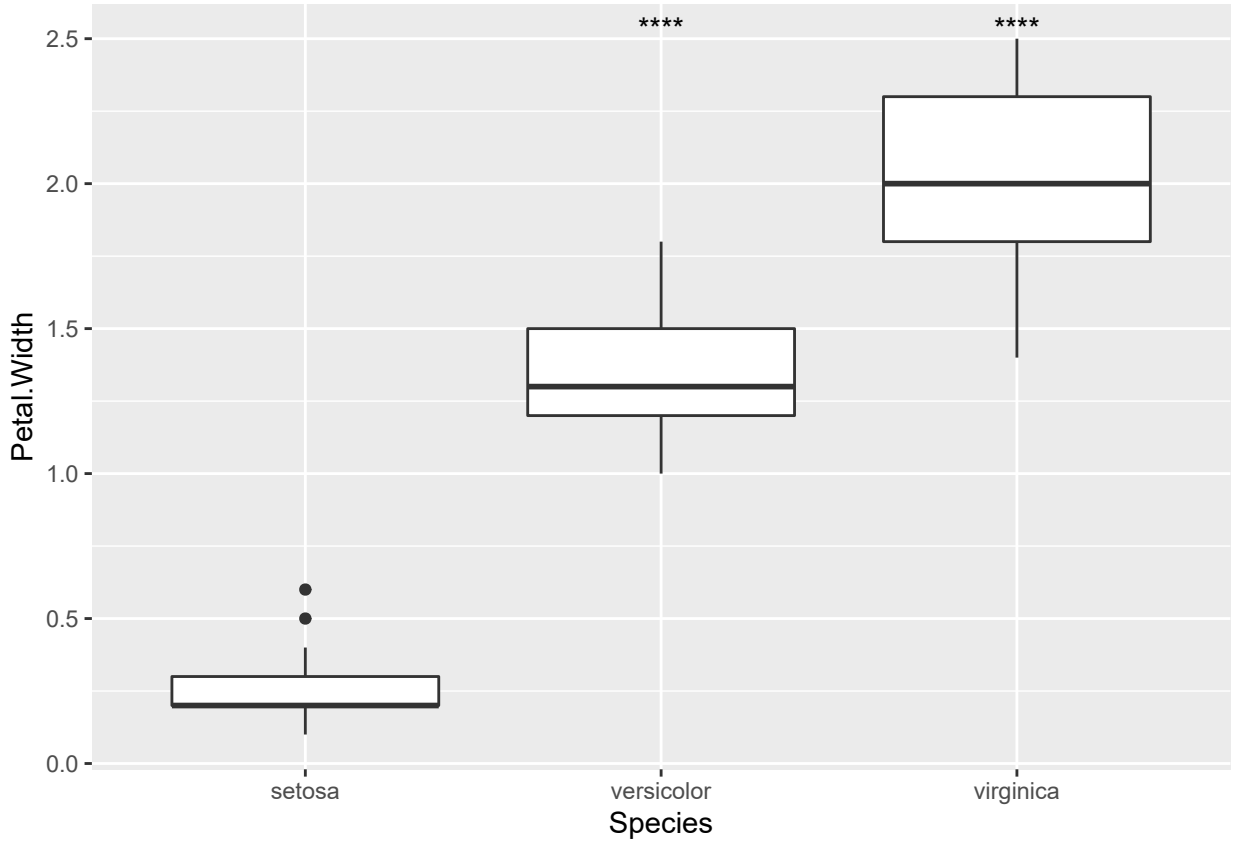
```
# ANOVO uporabimo za statistično primerjavo dolžine listov treh vrst perunike.
# Primerjamo, ali vrsta perunike vpliva na dolžino listov.
my_anova <- aov(Sepal.Length ~ Species, data = iris)
summary(my_anova)
```

```
##              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`. Poleg statističnega testa bomo izrisali tudi boxplot (Diagram s *škatlami in brčicami*).

```
library(ggplot2)
library(ggpubr)
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

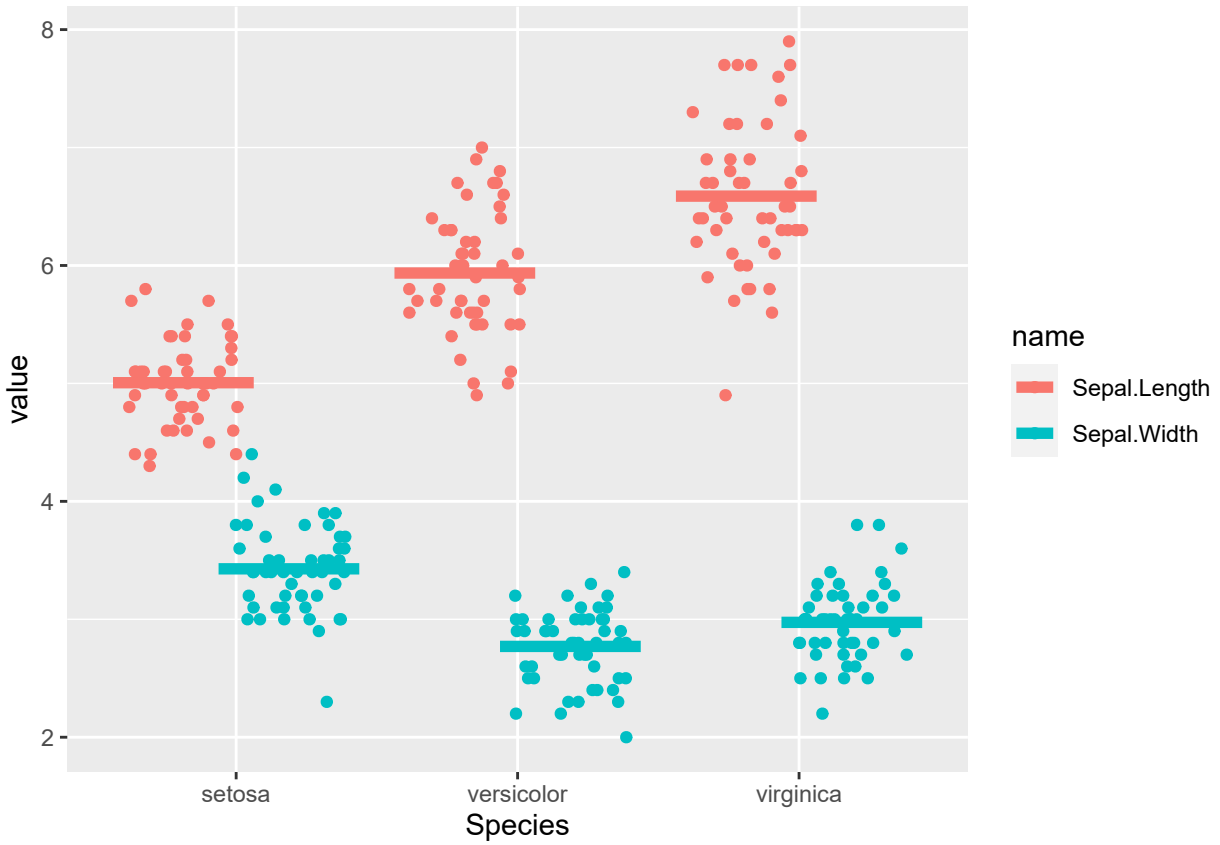
Poglejmo si še en zanimiv graf, kjer bomo prikazali točke in povprečja na istem grafu. Pogledali si bomo porazdelitve dolžin in širin čašnih listov različnih perunik. Najprej si pripravimo `data.frame`.

```
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 `hpline` 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"
  )
```



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. Poglejmo si odvisnost milj na galono (`mpg`) od števila cilindrov.

```
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
```

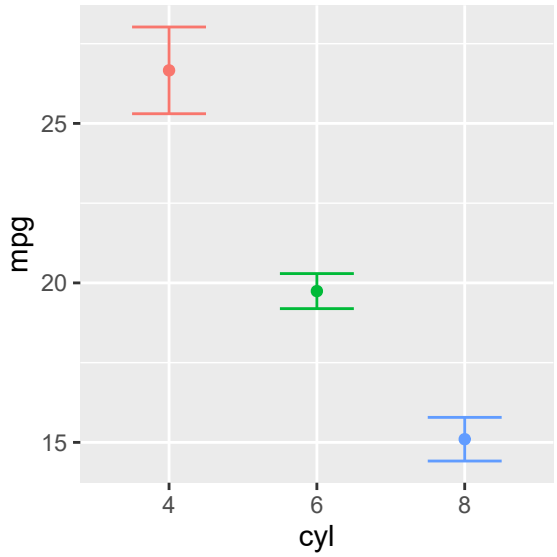
```
mus <- aggregate(mpg ~ cyl, mtcars, FUN = mean)
sds <- aggregate(mpg ~ cyl, mtcars, FUN = function(x) {sd(x) / sqrt(length(x))})
df  <- cbind(mus, SE = sds$mpg)
df$cyl <- as.character(df$cyl)

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")
```



Tortni diagram

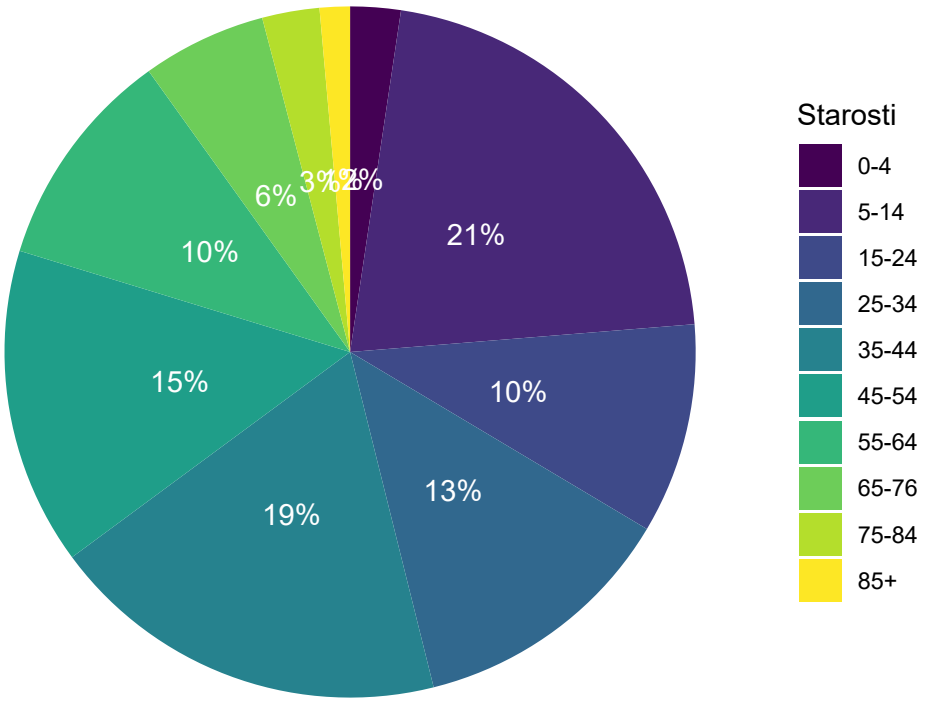
Kako narišemo tortni diagram s pomočjo paketa `ggplot2`? Poglejmo si kako izrišemo delež covid okužb po starostnih skupinah na dan 23.11.

```
covid <- data.frame(Starosti = c("0-4","5-14", "15-24", "25-34", "35-44", "45-54", "55-64", "65-76", "75
                    Stevilo = c(80, 725, 334, 426, 637, 504, 352, 197, 91, 48))
#Starost mora biti faktor, da ohranimo zaporedje
covid$Starosti <- factor(covid$Starosti, levels = covid$Starosti, ordered = T)

#Izračunamo procente
#Za prikaz procentov števila zaokrožimo na cela števila
covid$Procenti <- round(100*covid$Stevilo/sum(covid$Stevilo),0)

#Pripnemo znak za procent
covid$Procenti <- paste(covid$Procenti, '%', sep = '')


ggplot(covid, aes(x="", y = Stevilo, fill=Starosti)) +
  geom_bar(stat="identity", width=1,  position = position_stack(reverse = TRUE)) +
  coord_polar("y", start=0) + theme_void() +
  geom_text(aes(label = Procenti),
            position = position_stack(vjust = 0.5, reverse=TRUE),
                                                    color = "white")
```



V R je tortni diagram kar `geom_bar()` z vrednostjo Starost na eni osi in Procenti na drugi osi. Ker si želimo okrogli prikaz, uporabimo polarni koordinatski sistem namesto kartezijskega. Z ukazom `position_stack(reverse = TRUE)` zagotovim, da so skupine urejene v smeri urinega kazalca (privzeto so v nasprotni smeri). Starosti moramo definirati kot faktorje, če ne, nam R uredi deleže po velikosti.

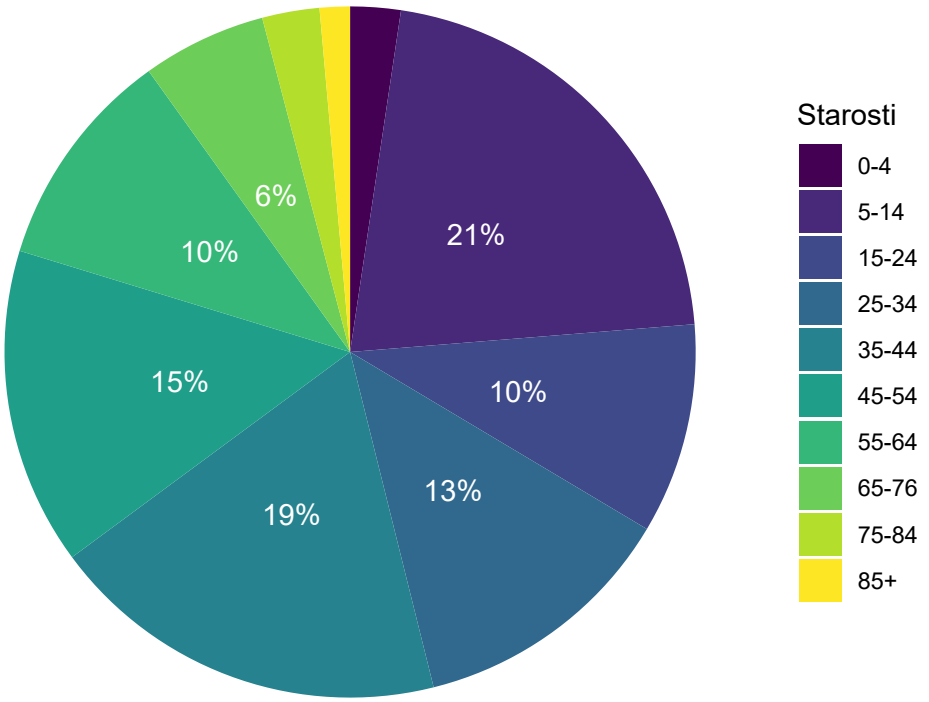
Z uporabo funcije `theme_void` odstranimo koordinatni sistem. V tem primeru vidimo, da je tortni diagram slab prikaz,ko opazujemo večje đtevilo razredov.

Prikažemo samo deleže večje od 5%.

```
covid$Procenti <- round(100*covid$Stevilo/sum(covid$Stevilo),0)
covid$Procenti_label <- covid$Procenti
#Oblikujemo izpis
covid$Procenti_label <- paste(covid$Procenti, '%', sep = '')


#Izberemo podatke, ki ustrezalo več kot 5% deležu, ostale pobrišemo
covid$Procenti_label[covid$Procenti < 5] <- ""

ggplot(covid, aes(x="", y = Stevilo, fill=Starosti)) +
  geom_bar(stat="identity", width=1,  position = position_stack(reverse = TRUE)) +
  coord_polar("y", start=0) + theme_void() +
  geom_text(aes(label = Procenti_label),
            position = position_stack(vjust = 0.5, reverse=TRUE),
            color = "white")
```



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
```

Velikokrat, ko imamo manjkajoče podatke želimo te vrstice obravnavati drugače. Poglejmo si naprimer že vgrajene podatke *airquality*, ki imajo manjkajoče vrednosti.

```
head(airquality)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Vrstice z manjkajočimi vrednostimi lahko preprosto odstranimo.

```
head(na.omit(airquality))
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 7    23     299  8.6   65     5   7
## 8    19      99 13.8   59     5   8
```

Lahko pa recimo manjkajoče vrednosti zamenjamo z povprečnimi.

```
df <- airquality
for(i in 1:ncol(df)){
  df[is.na(df[,i]), i] <- mean(df[,i], na.rm = TRUE)
}
head(df)
```

```
##      Ozone  Solar.R Wind Temp Month Day
## 1 41.00000 190.0000  7.4   67     5   1
## 2 36.00000 118.0000  8.0   72     5   2
## 3 12.00000 149.0000 12.6   74     5   3
## 4 18.00000 313.0000 11.5   62     5   4
## 5 42.12931 185.9315 14.3   56     5   5
## 6 28.00000 185.9315 14.9   66     5   6
```

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 niza znakov (beseda, stavek, ...) z nekim drugim nizom. Na primer:

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

```
## [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).

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. Seveda pa se lahko poslužimo tudi bolj preprostih, ampak manj sistematičnih pristopov, kot je na primer shranjevanje tabele v Excel in potem ročno kopiranje v Word. 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)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
# Preberemo samo prvo vrstico (nrow = 1), ki ima prvi header. Funkcija unlist
# data.frame spremeni v vektor.
head1 <- unlist(read.table("./data_raw/dvojni_header.csv", sep = ";",
                           quote = "", nrow = 1))
head1
```

```
##   V1   V2   V3   V4   V5
##   NA 2018   NA 2019   NA
```

```
# Preberemo samo drugo vrstico (izpustimo prvo -- skip = 1).
head2 <- unlist(read.table("./data_raw/dvojni_header.csv", sep = ";",
                           quote = "", nrow = 1, skip = 1))
head2
```

```
##     V1     V2     V3     V4     V5
## "kraj"    "m"    "f"    "m"    "f"
```

```
# Nadomestimo NA vrednosti v head2.
tmp <- na.locf(unlist(head1), na.rm = FALSE)
tmp
```

```
##   V1   V2   V3   V4   V5
```

```
##   NA 2018 2018 2019 2019
```

```
# Združimo oba headerja.
my_header <- paste(head2, tmp, sep = "_")
my_header
```

```
## [1] "kraj_NA" "m_2018"  "f_2018"  "m_2019"  "f_2019"
```

```
# Preberemo vrednosti v podatkih, jim priredimo 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)
```

```
## # A tibble: 6 x 3
##   kraj_NA name   value
##   <chr>   <chr>  <int>
## 1 LJ      m_2018     8
## 2 LJ      f_2018     4
## 3 LJ      m_2019     2
## 4 LJ      f_2019     1
## 5 KR      m_2018    22
## 6 KR      f_2018    21
```

```
# 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
head(podatki_long)
```

```
## # A tibble: 6 x 4
##   kraj_NA value spol  leto
##   <chr>   <int> <chr> <chr>
## 1 LJ          8 m     2018
## 2 LJ          4 f     2018
## 3 LJ          2 m     2019
## 4 LJ          1 f     2019
## 5 KR         22 m     2018
## 6 KR         21 f     2018
```

Vizualizacija zemljevidov

Uporabljali bomo knjižnico `maps`. Z uporabo knjižnice `ggmap` lahko dostopamo do zemljevidov "Google maps", ki izgledajo res lepo, ampak je samo določeno število dostopov brezplačno. Moramo se registrirati in podati številko kreditne kartice. Knjižnica `maps` je brezplačna.

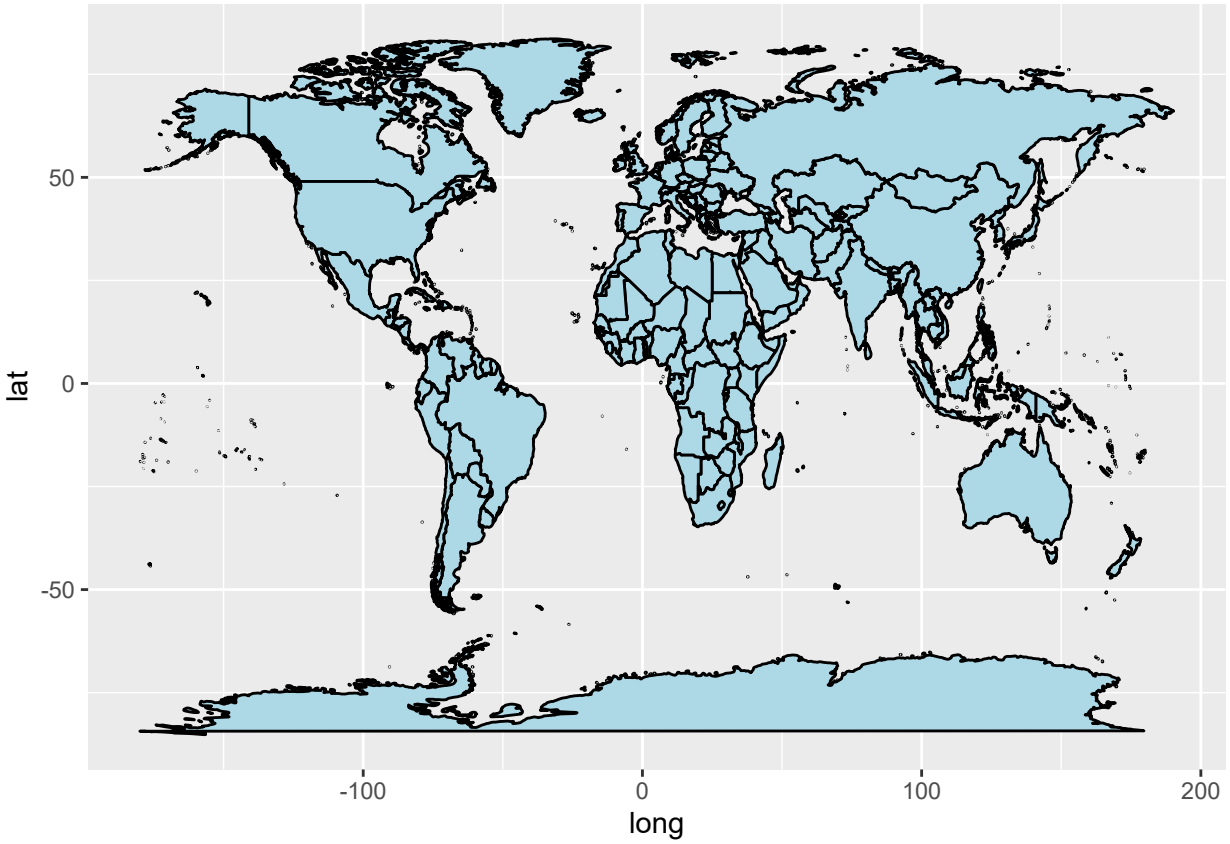
Lahko izrišemo zemlevid sveta:

```
library(maps)

svet <- map_data('world')
print(head(svet))
```

```
##        long      lat group order region subregion
## 1 -69.89912 12.45200     1     1  Aruba      <NA>
## 2 -69.89571 12.42300     1     2  Aruba      <NA>
## 3 -69.94219 12.43853     1     3  Aruba      <NA>
## 4 -70.00415 12.50049     1     4  Aruba      <NA>
## 5 -70.06612 12.54697     1     5  Aruba      <NA>
## 6 -70.05088 12.59707     1     6  Aruba      <NA>
```

```
ggplot() +
  geom_polygon( data=svet, aes(x=long, y=lat, group=group),
                color="black", fill="lightblue" )
```

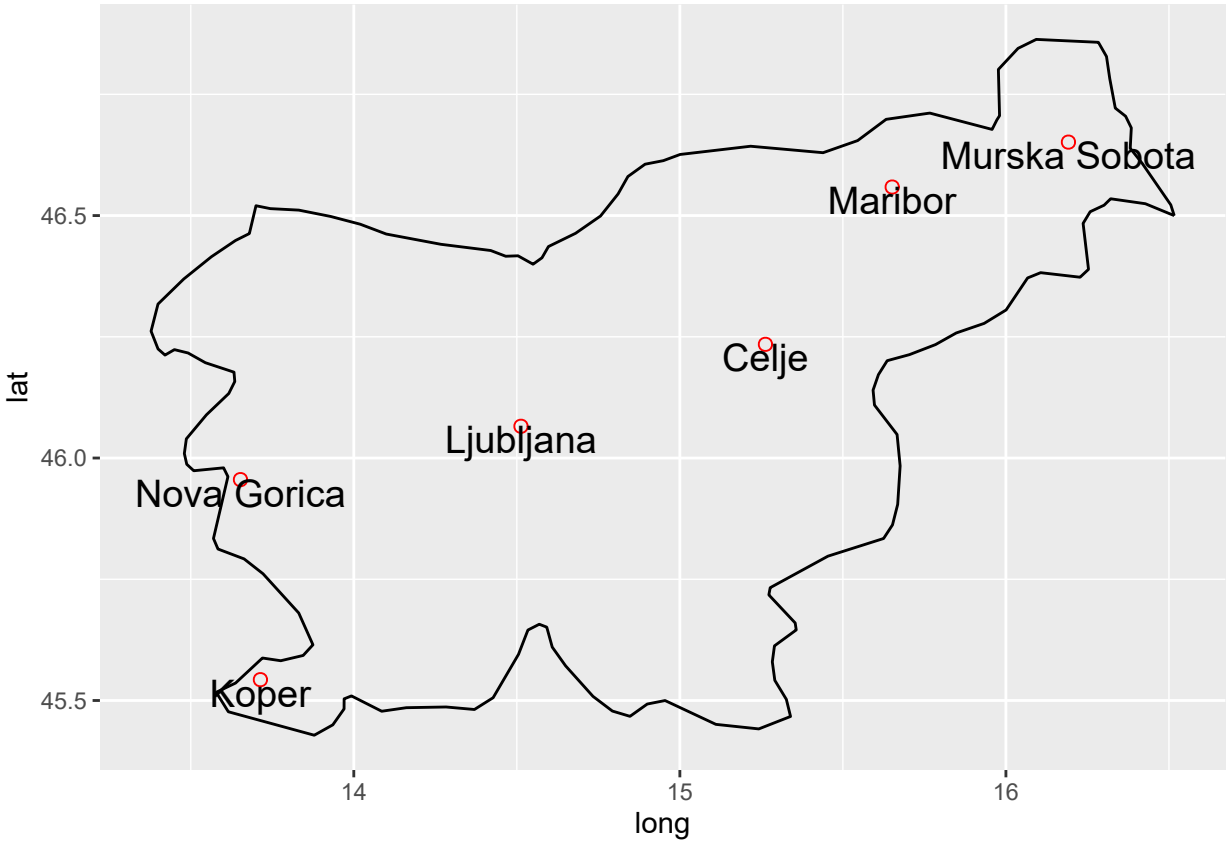


Lahko si izberemo tudi ožje področje, npr. Slovenijo. Na zemljevid lahko narišemo tudi poljubne točke (v napšem primeru jih shranimo v `data.frame` postaje).

```
slo <- "Slovenia"
slo.map <- map_data("world", region = slo)

postaje <- data.frame(Mesta = c("Ljubljana", "Celje", "Maribor", "Murska Sobota", "Nova Gorica", "Koper

ggplot() +
  geom_path(data = slo.map, aes(x = long, y = lat))+
  geom_point(data=postaje, aes(x=long, y=lat), colour="Red",pch=1, size=2) +
theme(legend.position = "none") + geom_text(data = postaje, aes(long, lat, label=Mesta),
                                            size=5, vjust = 1) #v just premakne napis, da ne pokriva to
```



Funkcija `geom_path` izriše konturo medtem, ko ima polygon tudi notranjost (lahko določimo barvo).

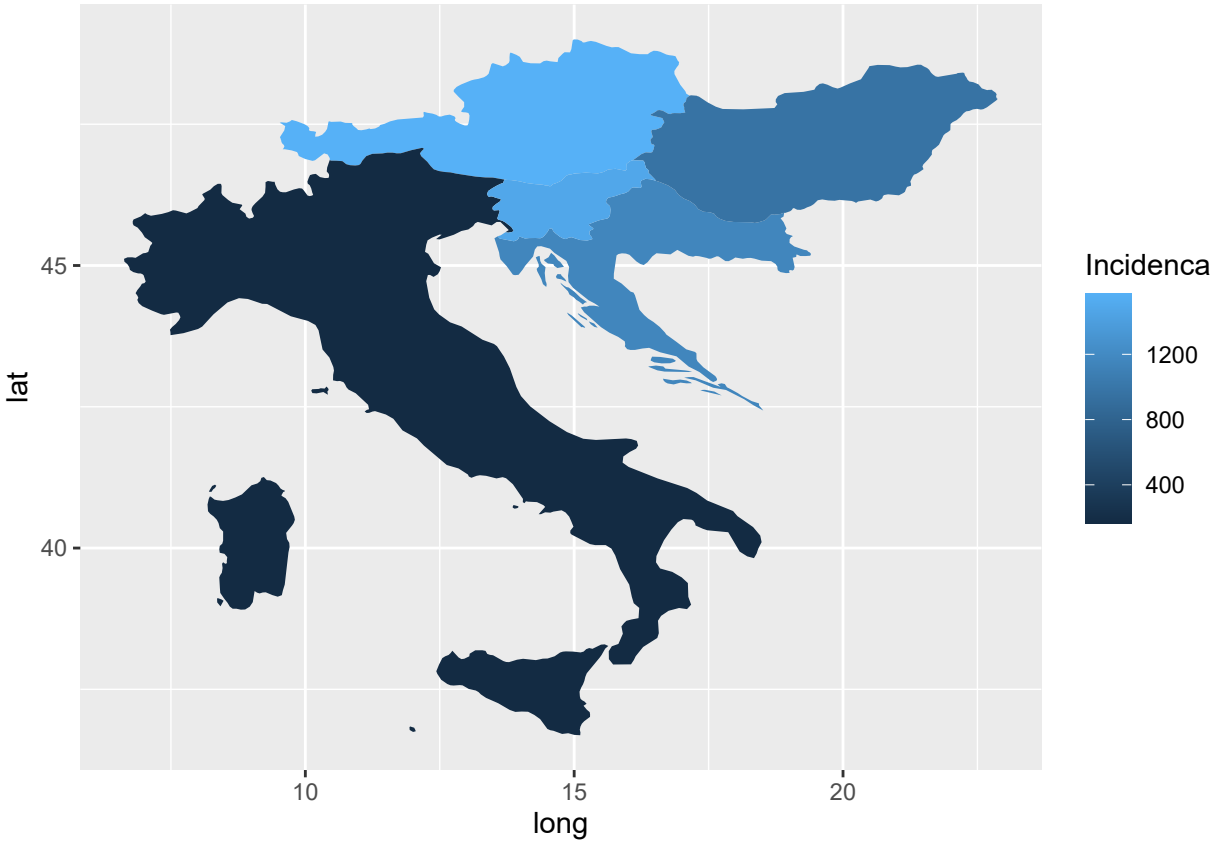
Naslednji primer uporabe zemljevida je barvanje različnih področji glede na neke lastnosti. Poglejmo primer 7-dnevnega povprečja covid primerov na milijon prebivalcev (incidenca). Incidence moramo najprej izračunati.

```
okolica <- c('Slovenia','Italy', 'Croatia', 'Austria', 'Hungary')
okolica.map <- map_data("world", region = okolica)

#7-dnevno povprečje v torek
covid_torek <- c(3106, 9866, 4615, 14004, 9435)
#prebivalci po državah
prebivalci <- c(2.1, 59.5, 4, 8.9, 9.8)
incidenca_tabela <- data.frame(Drzava = okolica, Incidenca = (covid_torek/prebivalci))


#tabeli okolica.map dodamo prazen stolpec Incidence
okolica.map <- cbind(okolica.map, Incidenca=rep(NA, nrow(okolica.map)))
# Vnesemo incidence v tabelo okolica.map po državah
for (d in okolica){
  okolica.map[okolica.map$region == d, 'Incidenca'] <-
    incidenca_tabela[incidenca_tabela$Drzava == d, 'Incidenca']
}

ggplot(data = okolica.map, aes(x = long, y = lat)) +
  geom_polygon(aes(group=group, fill=Incidenca))
```



Bioconductor

Bioconductor je odprtikodni projekt, ki vsebuje funkcije in pakete za analizo bioloških testov. Celoten projekt je razvit v R-ju in se posodablja dvakrat letno. Trenutna verzija (3.14) vsebuje okoli 3000 paketov. Ti paketi niso na voljo preko funkcije `install.packages()`, saj ima projekt svoj repozitorij.

Za instalacijo paketov za Bioconductor verzija 3.8 potrebujete najprej paket `BiocManager` z uradnega cran repozitorija. Tega najprej namestite z naslednjim ukazom. Funkcija `requireNamespace` preveri, če je paket že naložen. Če je, se bo instalacija preskočila. If stavek sicer ni potreben, bomo pa v tem dokumentu pustili sintakso, kakršno boste dobili tudi na njihovi uradni strani, da se boste v prihodnje lažje razumeli namen teh dodatnih ukazov.

```
if (!requireNamespace("BiocManager"))
    install.packages("BiocManager")
```

```
## Loading required namespace: BiocManager
```

Za instalacijo Bioconductorja in ostalih paketov, ki so na voljo pa od sedaj naprej uporabljate paket BiocManager. Instalirajmo najprej Bioconductor.

```
BiocManager::install()
```

```
## Bioconductor version 3.14 (BiocManager 1.30.16), R 4.1.2 (2021-11-01)
```

```
## Old packages: 'backports', 'digest', 'igraph', 'rlang', 'stringi', 'tibble',
##   'utf8', 'xfun'
```

Zgornji ukaz instalira najnovejšo verzijo Bioconductorja. Če iz kakršnega koli razloga potrebujete starejšo verzijo uporabite parameter `version`. Primer:

```
BiocManager::install(version = "3.12")
```

Opazite še novo sintakso *ime paketa*::*ime funkcije*. S dvojnim dvopičjem lahko uporabimo funkcijo nekega paketa brez, da bi ga predhodno naložili z ukazom `library()`, kar je uporabno, če potrebujemo funkcijo paketa le enkrat.

Če imate paket BiocManager in Bioconductor že naložen lahko preverite naloženo verzijo z naslednjima ukazoma.

```
#verzija paketa
packageVersion("BiocManager")
```

```
## [1] '1.30.16'
```

```
#verzija bioconductorja
BiocManager::version()
```

```
## [1] '3.14'
```

Sedaj si poglejmo, kako naložimo pakete. Najprej si na uradni strani poglejmo seznam paketov, ki so nam na voljo. Recimo, da nas po opisu *Quasispecies Diversity* zanima paket `QSutils`. Za podrobnejši opis kliknite na ime paketa in odprla se vam bo stran z osnovnimi podatki. Na tej strani so pod naslovom Documentation tudi zelo uporabni HTML in PDF dokumenti, ki vedno vključujejo tudi osnovni primer uporabe teh paketov.

Paket namestimo z ukazom:

```
BiocManager::install("QSutils")
```

```
## Bioconductor version 3.14 (BiocManager 1.30.16), R 4.1.2 (2021-11-01)
```

```
## Warning: package(s) not installed when version(s) same as current; use `force = TRUE` to
##   re-install: 'QSutils'
```

```
## Old packages: 'backports', 'digest', 'igraph', 'rlang', 'stringi', 'tibble',
##   'utf8', 'xfun'
```

Nekaj osnovnih napotkov paketa lahko dobite tudi z ukazom `browseVignettes()`. Ta ukaz deluje tudi za ostale pakete, ki niso del Bioconduktorja.

```
browseVignettes("QSutils")
```

Naložimo paket.

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which.max, which.min
```

```
## Loading required package: S4Vectors

## Loading required package: stats4

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:tidyr':
##
##     expand

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following object is masked from 'package:grDevices':
##
##     windows

## Loading required package: XVector

## Loading required package: GenomeInfoDb

##
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':
##
##     strsplit
```

Sedaj naredimo en svoj primer s tem paketom. Najprej naložimo podatke o genih, ki so zapisani v .fasta obliki.

```
gene <- ReadAmplSeqs("./data_raw/nucleus_gene.fast", type="DNA")
gene$hseqs
```

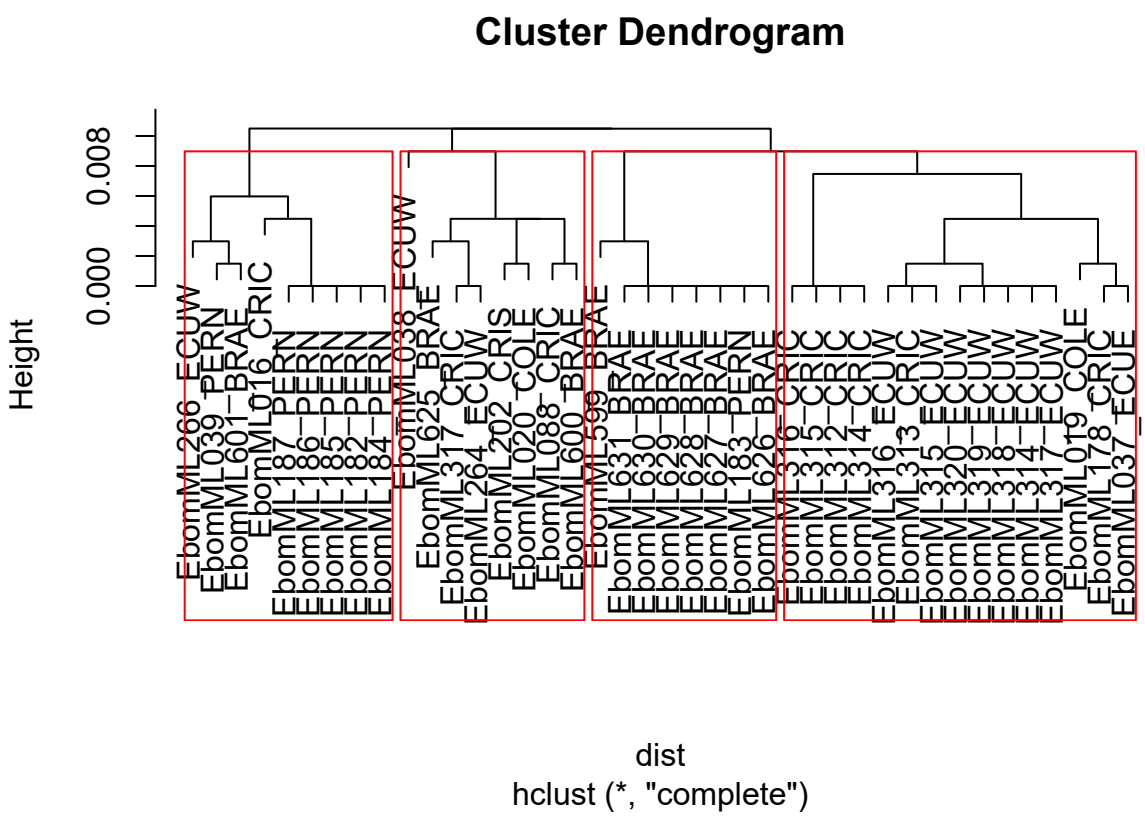
```
## DNAStringSet object of length 40:
##      width seq                                                names
##  [1]   673 AGCCTTCTCTCTTGGTGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML016_CRIC
##  [2]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML088_CRIC
##  [3]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML178_CRIC
##  [4]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML202_CRIS
##  [5]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML312_CRIC
##  ...   ... ...
## [36]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML627_BRAE
## [37]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML628_BRAE
## [38]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML629_BRAE
## [39]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML630_BRAE
## [40]   673 AGCCTTCTCTCTTGGCGGTTTGG...GATTTTAGATCGGCAGCTCGAT EbomML631_BRAE
```

Prebrani podatki so tipa `list`, ki vsebuje vektor `nr` in 40 vzorcev gena v `hseqs`.

Ker je podatkov relativno malo jih poskušamo združiti v gruče s hierarhičnim gručenjem in prikažimo rezultat v obliki dendrograma.

```
# Izračunamo razdalje med sekvencami
dist <- DNA.dist(gene$hseqs, pairwise.deletion = T, model = "K80")
```

```
# Hirarhično gručenje
clusters <- hclust(dist)
# Prikažemo razdalje z dendrogramom
plot(clusters)
# Na dendogramu obkrožimo štiri skupine
rect.hclust(clusters, k=4, border="red")
```

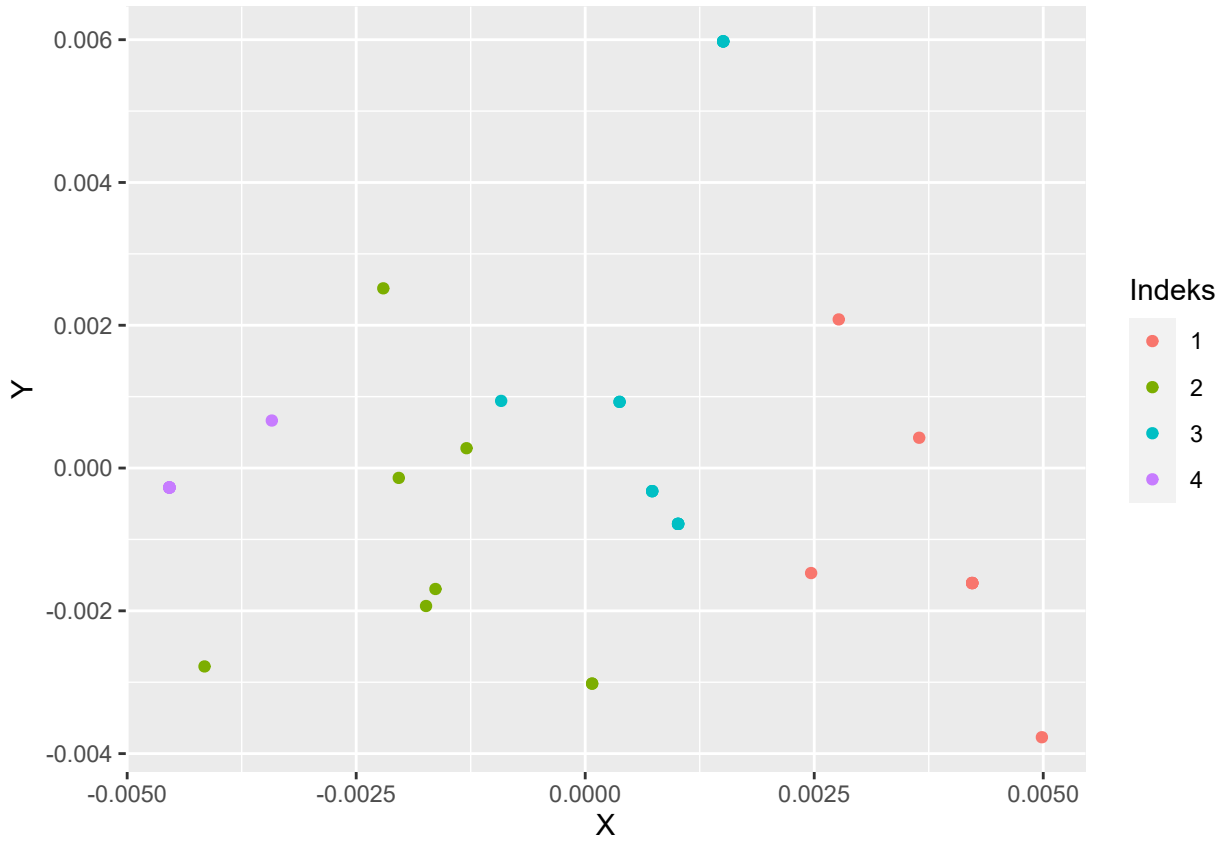


Rezultati so tukaj različni glede na uporabljeno mero za izračun razdalj, ki jo podamo z parametrom `model`. Ostale možnosti si lahko pogledate v dokumentaciji z ukazom `?DNA.dist`. Naredimo še vizualizacijo gruč v 2d prostoru za štiri skupine. Če želite dendogram porezati na neki določeni višini v spodnji kodi namesto `k = 4` uporabite `h = višina`.

```
# Razdelimo na štiri skupine
cluster_ids <- cutree(clusters, k = 4)

#Pripravimo podatke v obliki tabele
podatki <- as.data.frame(cmdscale(dist)) #predstavimo podatke v 2d obliki
names(podatki) <- c("X", "Y")
podatki$Indeks <- as.factor(cluster_ids) #dodamo indekse

#Izrišemo  vzorce na graf
ggplot(podatki, aes(x = X, y = Y, colour = Indeks)) +
  geom_point()
```



Nekaj krajših sekvenc vsebuje tudi sam paket. Naložimo jih:

```
filepath<-system.file("extdata","ToyData_RVReads.fna", package="QSutils")
sekvenca <- ReadAmplSeqs(filepath,type="DNA")
head(sekvenca$nr)
```

```
## [1] 44582  2319  1360   231   100    95
```

```
sekvenca$hseqs
```

```
## DNAStringSet object of length 1162:
##         width seq                                               names
##     [1]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
##     [2]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
##     [3]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
##     [4]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
##     [5]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
##     ...   ... ...
## [1158]   336 TCGTGGCCACTGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
## [1159]   336 TCGTGGCCATCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
## [1160]   336 TCGTGGCTACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
## [1161]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
## [1162]   336 TCGTGGCCACCGCATTGCGTCA...CTGGAGGAGATGCGCACCGCC 15457.Myc.178.554...
```

Paket *QSutils* ponuja tudi funkcije za analizo teh podatkov. *Opozorilo:* Spodaj opisani stavki mogoče vsebujejo napake, glede na nepozavanje domene :)

Poglejmo si mero negotovosti Shannon in verjetnost (GiniSimpson), da dva naključna vzorca iz populacije pripadata različnim haplotipom.

```
Shannon(sekvenca$nr)
```

```
## [1] 2.510248
```

```
GiniSimpson(sekvenca$nr)
```

```
## [1] 0.5351748
```

Z analizo razdalj med haplotipi lahko izračunamo tudi druge statistike. Poglejmo povprečno frekvenco mutacije in povprečno diverziteto nukleotidov.

```
razdalje <- DNA.dist(sekvenca$hseqs,model="K80")
MutationFreq(razdalje)
```

```
## [1] 0.004017996
```

```
NucleotideDiversity(razdalje)
```

```
##             [,1]
## [1,] 0.007762838
```

Razhroščevanje (debugging)

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` (*next*) se browser pomakne na naslednji ukaz, ki se bo izvajal. Z ukazom `c` (*continue*) mu povemo naj nadaljuje izvajanje do konca oz. dokler ne pride do naslednjega klica `browser()`. Iz browserja lahko prekinemo izvajanje zanke in se vrnemo v osnovno konzolo s tipko **escape** ali če napišemo `Q` (*Quit*).

Če nam težava dela določena funkcija in jo ne želimo spreminjati z dodajanjem ukazov `browser()` lahko v browser preidemo tudi ob klicu funkcije z ukazom `debug()`.

```
mnozi <- function(a, b){
  return(a*b)
}
deli <- function(a, b){
  return(a/b)
}
debug(deli) #označimo funkcijo deli za razhroščevanje
mnozi(2, 3) #se izvede normalno
deli(2, 3) #na prvi vrstici funkcije se pokliče browser()
```

Če želimo sprostiti funkcije iz razhroščevalnika uporabimo `undebug()`.

```
undebug(deli)
```