Ex01

Andrea Nicolai

5/4/2020

EXERCISE 1

Given the following lakes, make a dataframe out of this table after having put all the data in their respective vectors.

```
names <- c("Loch Ness", "Loch Lomond", "Loch Morar", "Loch Tay", "Loch Awe", "Loch Maree", "Loch Ericht
volume <- c(7.45, 2.6,2.3,1.6,1.2,1.09,1.08,1.07,0.97,0.79,0.77,0.75,0.35)
area <- c(56,71,27,26.4,39,28.6,18.6,16,19,19.5,12.4,16,22.5)
length <- c(39,36,18.8,23,41,20,23,16,15.7,28,12.9,19.3,27.8)
max_depth <- c(230,190,310,150,94,114,156,162,134,128,151,109,49)
mean_depth <- c(132,37,87,60.6,32,38,57.6,70,51,40,43.4,46.5,15.5)

scottish.lakes <- data.frame(names,volume,area,length,max_depth,mean_depth)
```

The largest volume lake is:

```
scottish.lakes[order(scottish.lakes$volume, decreasing = TRUE),][1,1]
```

```
## [1] Loch Ness
## 13 Levels: Loch Arkaig Loch Awe Loch Ericht Loch Katrine ... Loch Tay
```

```
head(scottish.lakes, n=1)['names']
```

```
##       names
## 1 Loch Ness
```

The lowest volume lake is:

```
scottish.lakes[order(scottish.lakes$volume),][1,1]
```

```
## [1] Loch Shin
## 13 Levels: Loch Arkaig Loch Awe Loch Ericht Loch Katrine ... Loch Tay
```

```
#scottish.lakes[order(scottish.lakes$volume),]
#head(scottish.lakes, n=1)
```

The largest area lake is:

```
scottish.lakes[order(scottish.lakes$area, decreasing = TRUE),][1,1]
```

```
## [1] Loch Lomond
## 13 Levels: Loch Arkaig Loch Awe Loch Ericht Loch Katrine ... Loch Tay
```

The smallest area lake is:

```
scottish.lakes[order(scottish.lakes$area),][1,1]
```

```
## [1] Loch Katrine
## 13 Levels: Loch Arkaig Loch Awe Loch Ericht Loch Katrine ... Loch Tay
```

The two largest area lakes is:

```r
scottish.lakes[order(scottish.lakes$area),]
```

```
##            names volume area length max_depth mean_depth
## 11 Loch Katrine   0.77 12.4   12.9       151       43.4
## 8    Loch Lochy   1.07 16.0   16.0       162       70.0
## 12  Loch Arkaig   0.75 16.0   19.3       109       46.5
## 7   Loch Ericht   1.08 18.6   23.0       156       57.6
## 9  Loch Rannoch   0.97 19.0   15.7       134       51.0
## 10   Loch Shiel   0.79 19.5   28.0       128       40.0
## 13    Loch Shin   0.35 22.5   27.8        49       15.5
## 4      Loch Tay   1.60 26.4   23.0       150       60.6
## 3    Loch Morar   2.30 27.0   18.8       310       87.0
## 6    Loch Maree   1.09 28.6   20.0       114       38.0
## 5      Loch Awe   1.20 39.0   41.0        94       32.0
## 1     Loch Ness   7.45 56.0   39.0       230      132.0
## 2   Loch Lomond   2.60 71.0   36.0       190       37.0
```

```r
head(scottish.lakes, n=2)
```

```
##          names volume area length max_depth mean_depth
## 1   Loch Ness   7.45   56     39       230        132
## 2 Loch Lomond   2.60   71     36       190         37
```

Total area covered by water is:

```r
sum(scottish.lakes$area)
```

```
## [1] 372
```

EXERCISE 2:

```r
#install.packages(c("DAAG","tibble"), type="source")
```

Getting info info on package content:

```r
help(package="DAAG")
```

Getting info on ‘ais’ package and making a table out of that:

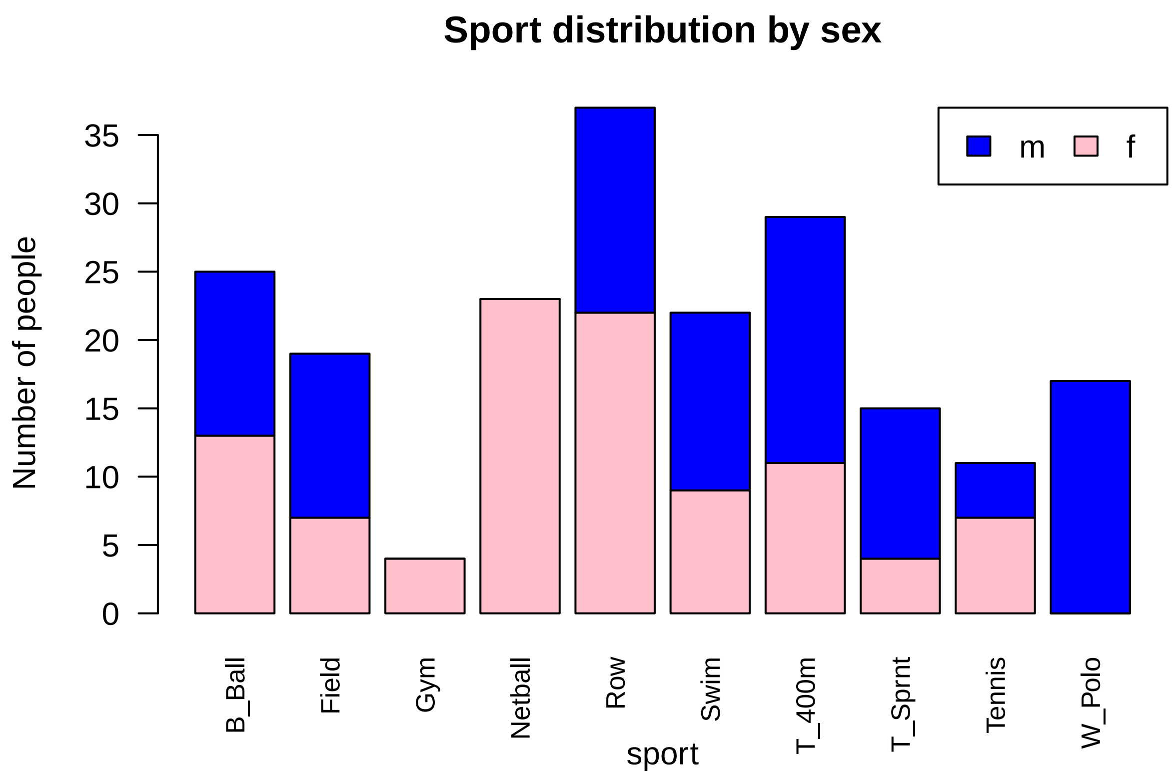
```r
library(tibble)
str(DAAG::ais)
```

```
## 'data.frame':    202 obs. of  13 variables:
##  $ rcc   : num  3.96 4.41 4.14 4.11 4.45 4.1 4.31 4.42 4.3 4.51 ...
##  $ wcc   : num  7.5 8.3 5 5.3 6.8 4.4 5.3 5.7 8.9 4.4 ...
##  $ hc    : num  37.5 38.2 36.4 37.3 41.5 37.4 39.6 39.9 41.1 41.6 ...
##  $ hg    : num  12.3 12.7 11.6 12.6 14 12.5 12.8 13.2 13.5 12.7 ...
##  $ ferr  : num  60 68 21 69 29 42 73 44 41 44 ...
##  $ bmi   : num  20.6 20.7 21.9 21.9 19 ...
##  $ ssf   : num  109.1 102.8 104.6 126.4 80.3 ...
##  $ pcBfat: num  19.8 21.3 19.9 23.7 17.6 ...
##  $ lbm   : num  63.3 58.5 55.4 57.2 53.2 ...
##  $ ht    : num  196 190 178 185 185 ...
##  $ wt    : num  78.9 74.4 69.1 74.9 64.6 63.7 75.2 62.3 66.5 62.9 ...
##  $ sex   : Factor w/ 2 levels "f","m": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sport : Factor w/ 10 levels "B_Ball","Field",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
tib_dataframe <- as_tibble(DAAG::ais)
```

Creating a table grouping the data by gender and by sport, and then creating a barplot out of that:

```r
  sport_vs_sex <- table(DAAG::ais[c("sex", "sport")])
  barplot(sport_vs_sex, main="Sport distribution by sex",
          xlab="sport", cex.names=0.8,  ylab="Number of people", col=c("pink","blue"), las =2,
          legend=TRUE, args.legend = list( x = "topright", ncol = 2))
```



Is there any missing values in the dataframe? And how many, if so?

```r
any(is.na.data.frame(tib_dataframe))
```

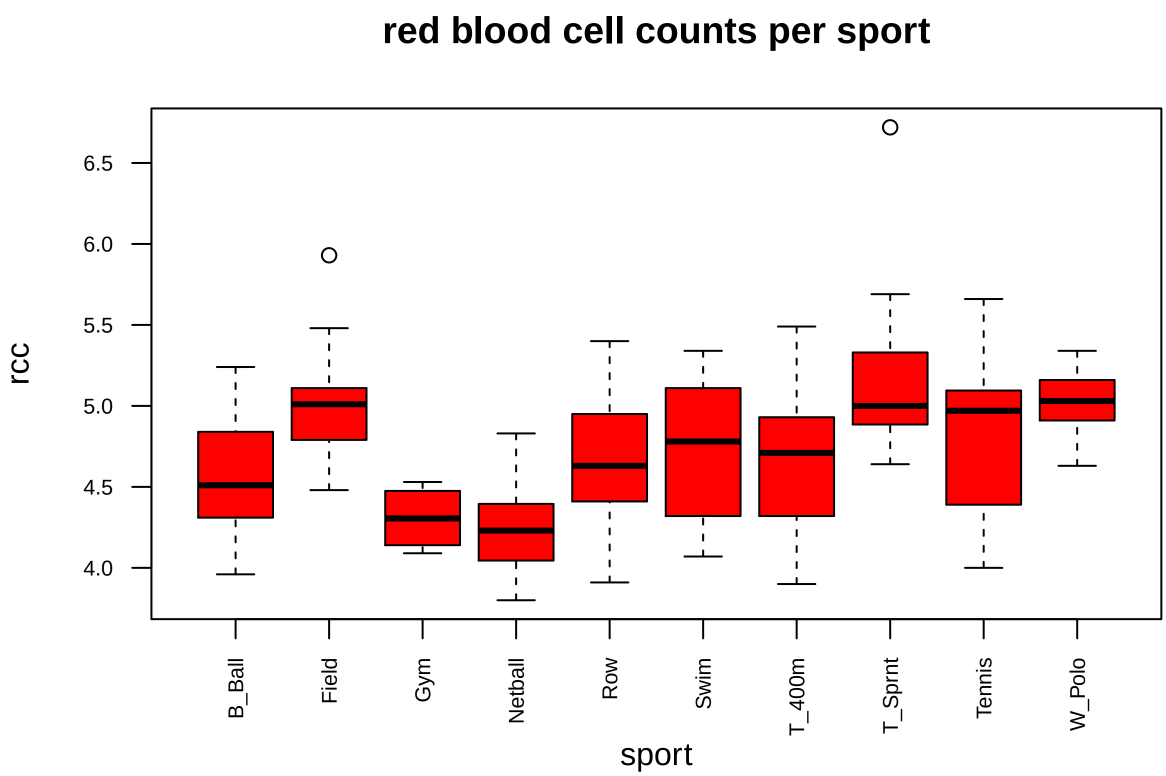
```
## [1] FALSE
```

```r
which(is.na(DAAG::ais))
```

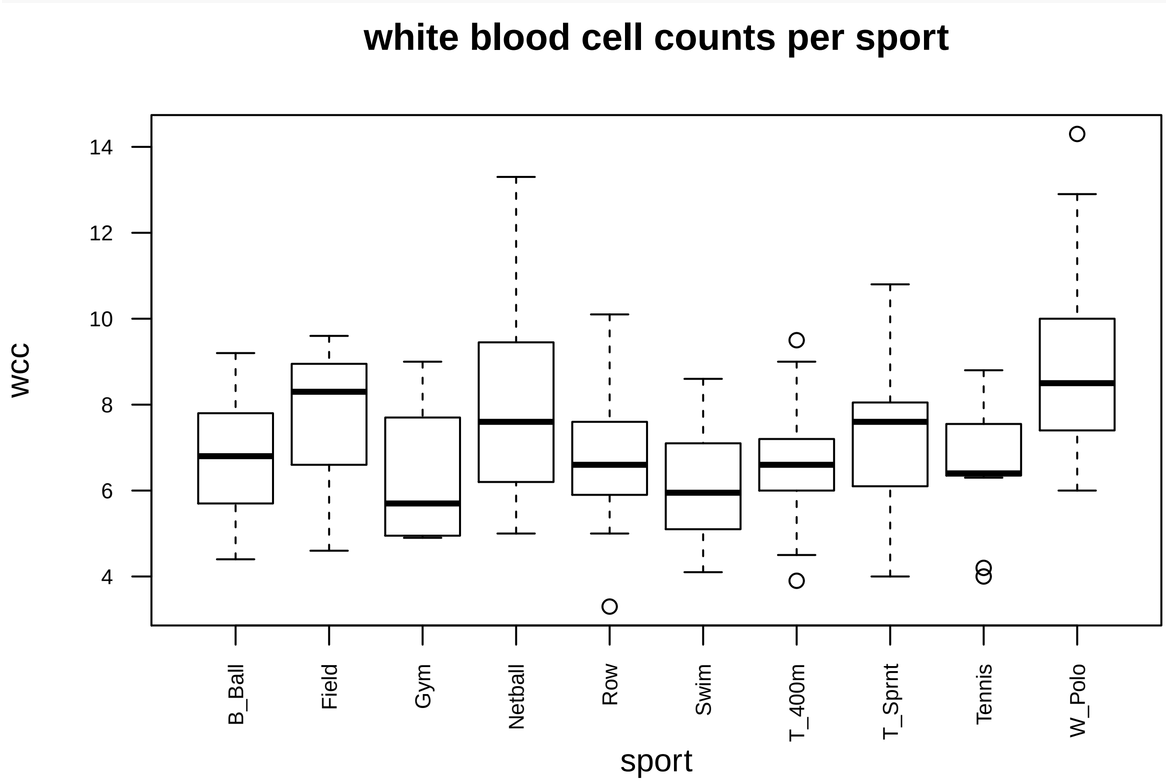
```
## integer(0)
```

Now let's produce boxplots of the main blood variables ('red blood cell counts', 'white blood cell counts','hematocrit' and 'hemoglobin concentration'), for different kind of sports:

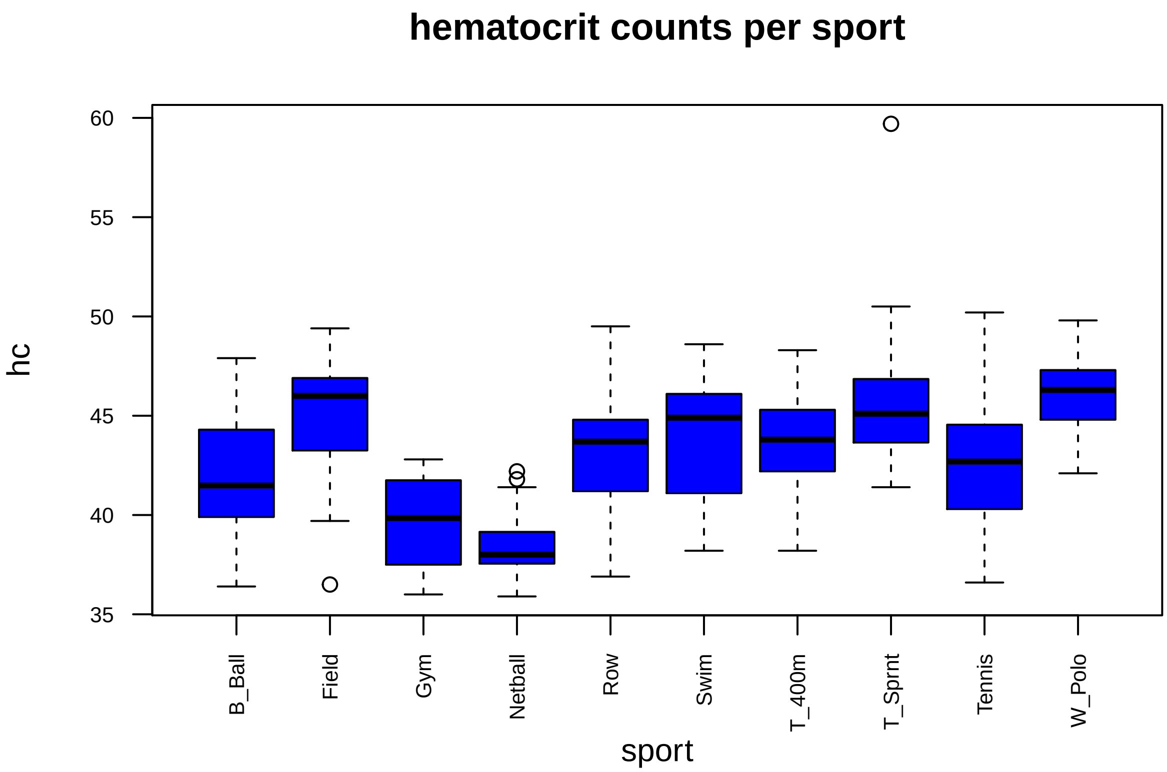
```r
boxplot(tib_dataframe$rcc~tib_dataframe$sport,
        main = 'red blood cell counts per sport', xlab = 'sport', ylab = 'rcc',
        col = 'red', border = 'black', las = 2, cex.axis=0.7)
```



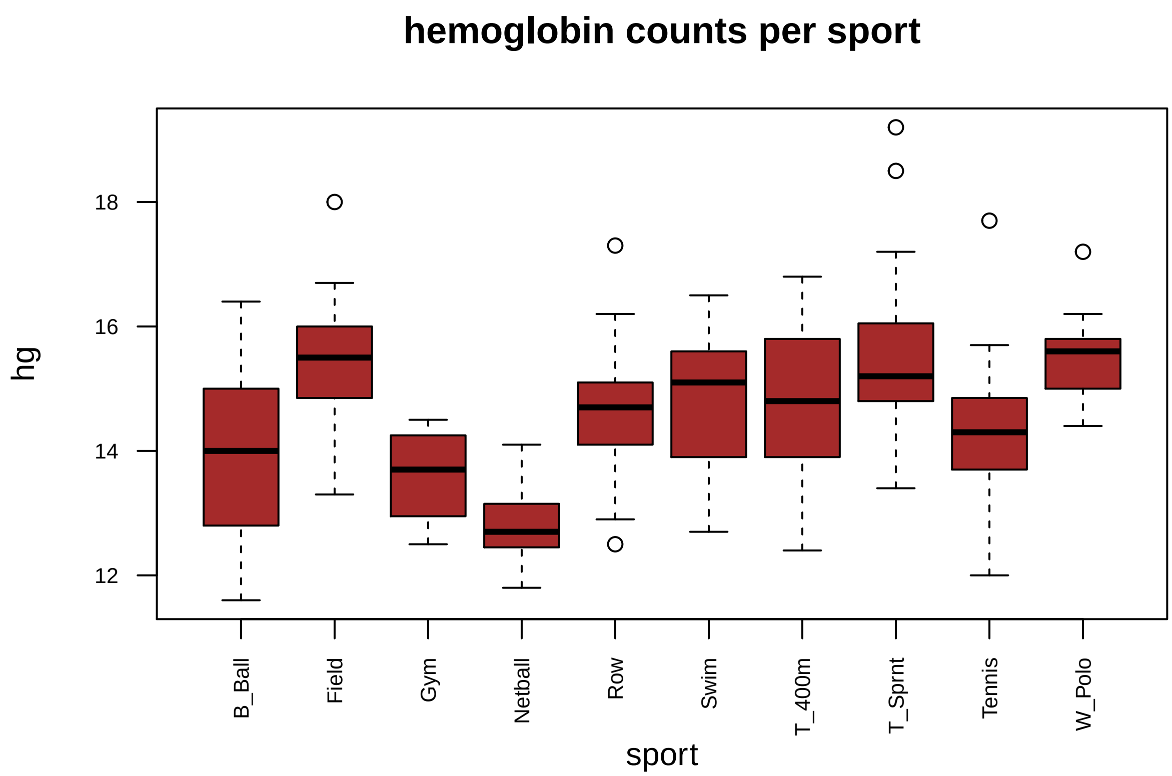

```r
boxplot(tib_dataframe$wcc~tib_dataframe$sport,
        main = 'white blood cell counts per sport', xlab = 'sport', ylab = 'wcc',
        col = 'white', border = 'black', las = 2, cex.axis=0.7)
```



```r
boxplot(tib_dataframe$hc~tib_dataframe$sport,
        main = 'hematocrit counts per sport', xlab = 'sport', ylab = 'hc',
        col = 'blue', border = 'black', las = 2, cex.axis=0.7)
```

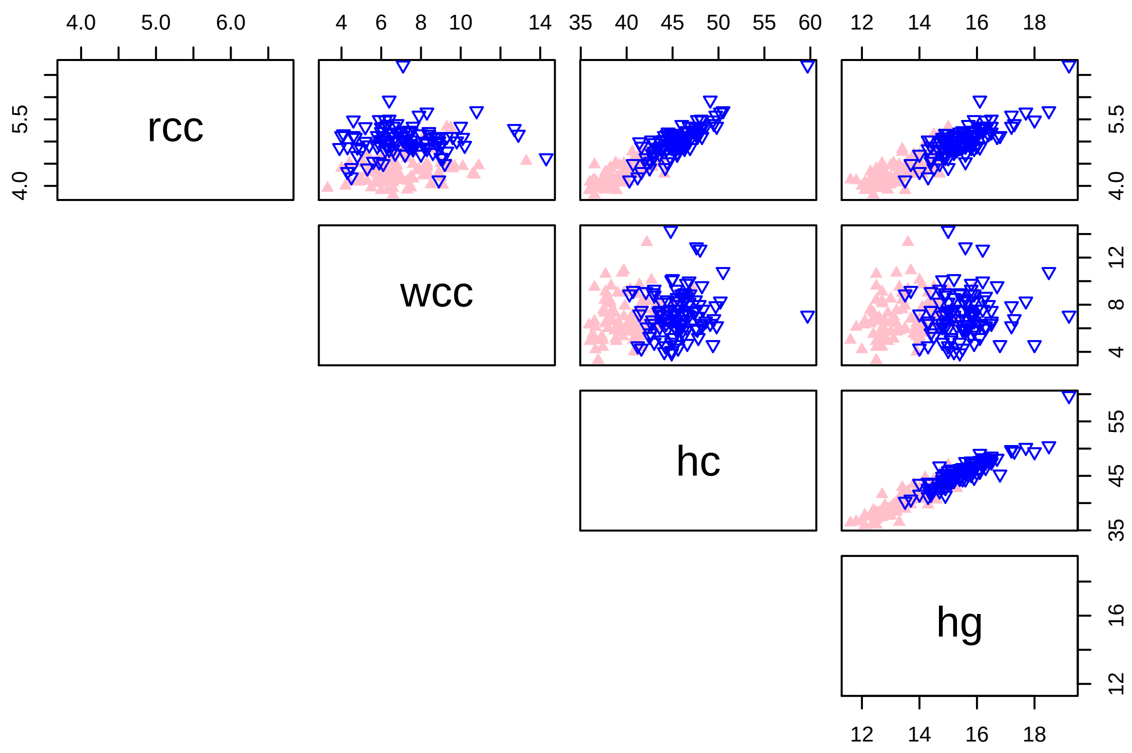



```r
boxplot(tib_dataframe$hg~tib_dataframe$sport,
        main = 'hemoglobin counts per sport', xlab = 'sport', ylab = 'hg',
        col = 'brown', border = 'black', las = 2, cex.axis=0.7)
```



Now we want to make some scatter plot correlations of the same blood variables using different colors and symbols for the two genders in the sample

```
my_colors <- c('pink', 'blue')
pairs(tib_dataframe[1:4], col = my_colors[tib_dataframe$sex],
      pch = c(17,25)[tib_dataframe$sex], lower.panel=NULL)
```



EXERCISE 3

Initializing everything by the script given by professor:

```r
needed_packages <- c( "lubridate" , "readxl" , "curl")
already_installed <- needed_packages %in% installed.packages()
for ( pack in needed_packages [!already_installed]) {
  message (paste(" To be installed : " ,pack,sep = " " ))
  install.packages( pack )
}
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

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

```r
library(readxl)
library(curl)
url <- "https://www.ecdc.europa.eu/sites/default/files/documents/"
fname <- "COVID-19-geographic-disbtribution-worldwide-"
date <- lubridate::today() - 1
ext = ".xlsx"
target <- paste( url , fname , date , ext , sep = "" )
message ( "target:" , target )
```

```
## target:https://www.ecdc.europa.eu/sites/default/files/documents/COVID-19-geographic-disbtribution-wo
```

```r
tmp_file <- tempfile ( "data" , "/tmp" , fileext = ext )
tmp <- curl::curl_download ( target , destfile = tmp_file )
covid <- readxl::read_xlsx ( tmp_file )
```

Let's examine the loaded tibble structure:

```r
str(covid)
```

```
## tibble [8,905 x 10] (S3: tbl_df/tbl/data.frame)
##  $ dateRep                : POSIXct[1:8905], format: "2020-04-05" "2020-04-04" ...
##  $ day                    : num [1:8905] 5 4 3 2 1 31 30 29 28 27 ...
##  $ month                  : num [1:8905] 4 4 4 4 4 3 3 3 3 3 ...
##  $ year                   : num [1:8905] 2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 ...
##  $ cases                  : num [1:8905] 35 0 43 26 25 27 8 15 16 0 ...
##  $ deaths                 : num [1:8905] 1 0 0 0 0 0 1 1 1 0 ...
##  $ countriesAndTerritories: chr [1:8905] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...
##  $ geoId                  : chr [1:8905] "AF" "AF" "AF" "AF" ...
##  $ countryterritoryCode   : chr [1:8905] "AFG" "AFG" "AFG" "AFG" ...
##  $ popData2018            : num [1:8905] 37172386 37172386 37172386 37172386 37172386 ...
```

Now we want to create a sub-tibble containing only the last day and produce a table with all the countries with number of deaths or number of new cases greater than 200:

```r
last_day <- subset(covid, dateRep == date)
high <- last_day[last_day$cases > 200 | last_day$deaths > 200, ]
#high <- subset(last_day, cases > 200 | deaths > 200)
```

We then select the top 10 countries, in terms of cases, and plot the total number of cases as a function of time. Plot the total number of deaths as a function of time. In order to compare the different curves, normalize the first date-time plot to the same t_0 value. First 10 top countries are:

```
highest <- head(high[order(-high$cases),], 10)
cat(paste(c("The 10 states with the largest number of cases at 31st of March are, in order: ")))
```

```
## The 10 states with the largest number of cases at 31st of March are, in order:
```

```
highest$countriesAndTerritories
```

```
##  [1] "United_States_of_America" "Germany"
##  [3] "Spain"                    "Iran"
##  [5] "Italy"                    "France"
##  [7] "United_Kingdom"           "Turkey"
##  [9] "Belgium"                  "Canada"
```

We now want to reaorder the dataset starting from the oldest data available for those countries:

```
covid <- covid[order(covid$dateRep),]
```

Here we want to find which can be the so-called t0, in order to plot data starting from the same date.

```
t0 <- min(covid$dateRep)
for (i in highest$countriesAndTerritories) {
  data_country <- (subset(covid, countriesAndTerritories == i))
  ifelse( min(data_country$dateRep) > t0, t0 <- min(data_country$dateRep), t0 )
}
```

Then we shift the data by 4 days since the Turkey has been missing data on a couple of days after the just-found t0:

```
shift_date <- t0 + as.difftime(4, units = "days")
```

Now we want to create a dataframe for the daily cases and rename its columns:

```
cases_dataframe <- tibble(unique(covid$dateRep[covid$dateRep >= shift_date]))
for (i in highest$countriesAndTerritories) {
  cases_country <- (subset(covid, (countriesAndTerritories == i) & (dateRep >= shift_date)))['cases']
  print(i)
  cases_dataframe <- cbind(cases_dataframe, cases_country )
}
```

```
## [1] "United_States_of_America"
## [1] "Germany"
## [1] "Spain"
## [1] "Iran"
## [1] "Italy"
## [1] "France"
## [1] "United_Kingdom"
## [1] "Turkey"
## [1] "Belgium"
## [1] "Canada"
```

```
colnames(cases_dataframe)[1] <- "Date"
colnames(cases_dataframe)[2:ncol(cases_dataframe)] <- highest$countriesAndTerritories
```

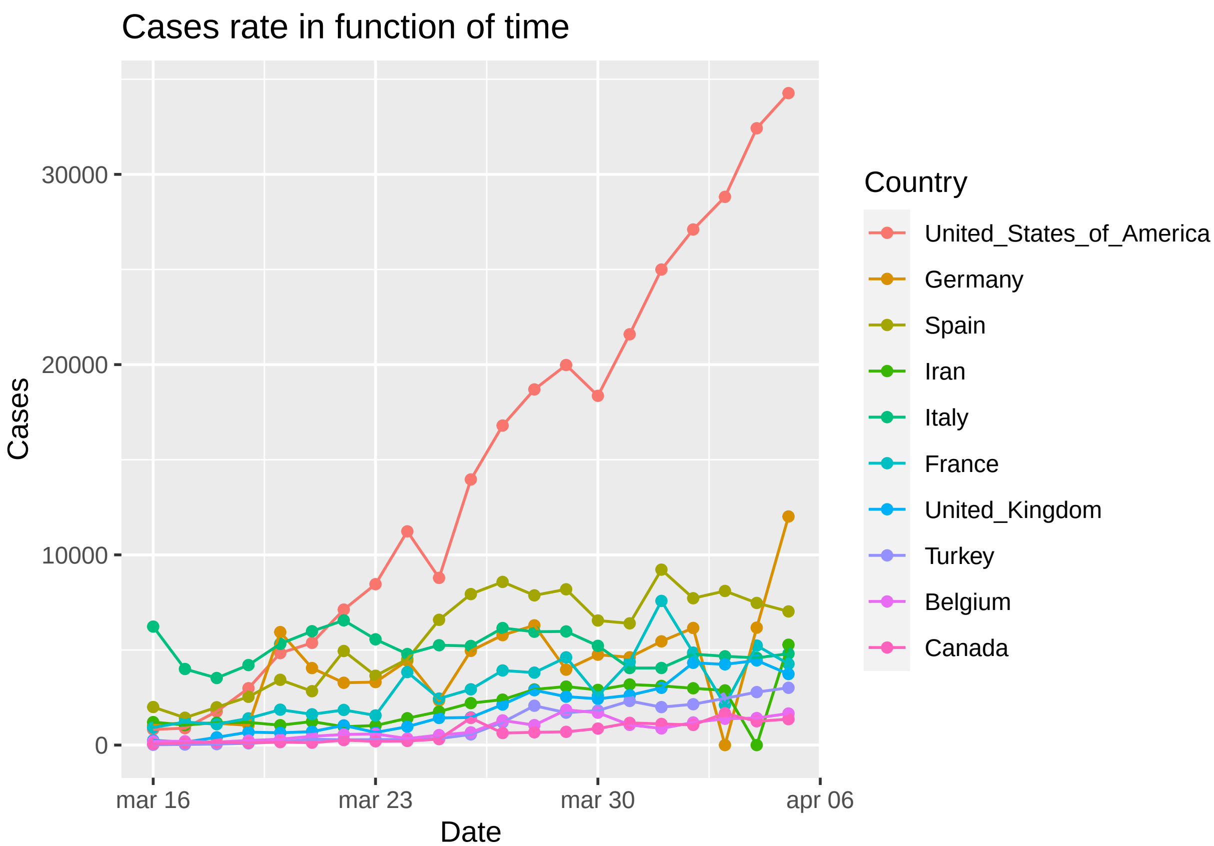
Now we want to create a dataframe for the daily deaths and rename its columns:

```
deaths_dataframe <- tibble(unique(covid$dateRep[covid$dateRep >= shift_date]))
for (i in highest$countriesAndTerritories) {
  deaths_country <- (subset(covid, (countriesAndTerritories == i) & (dateRep >= shift_date)))['deaths']
```

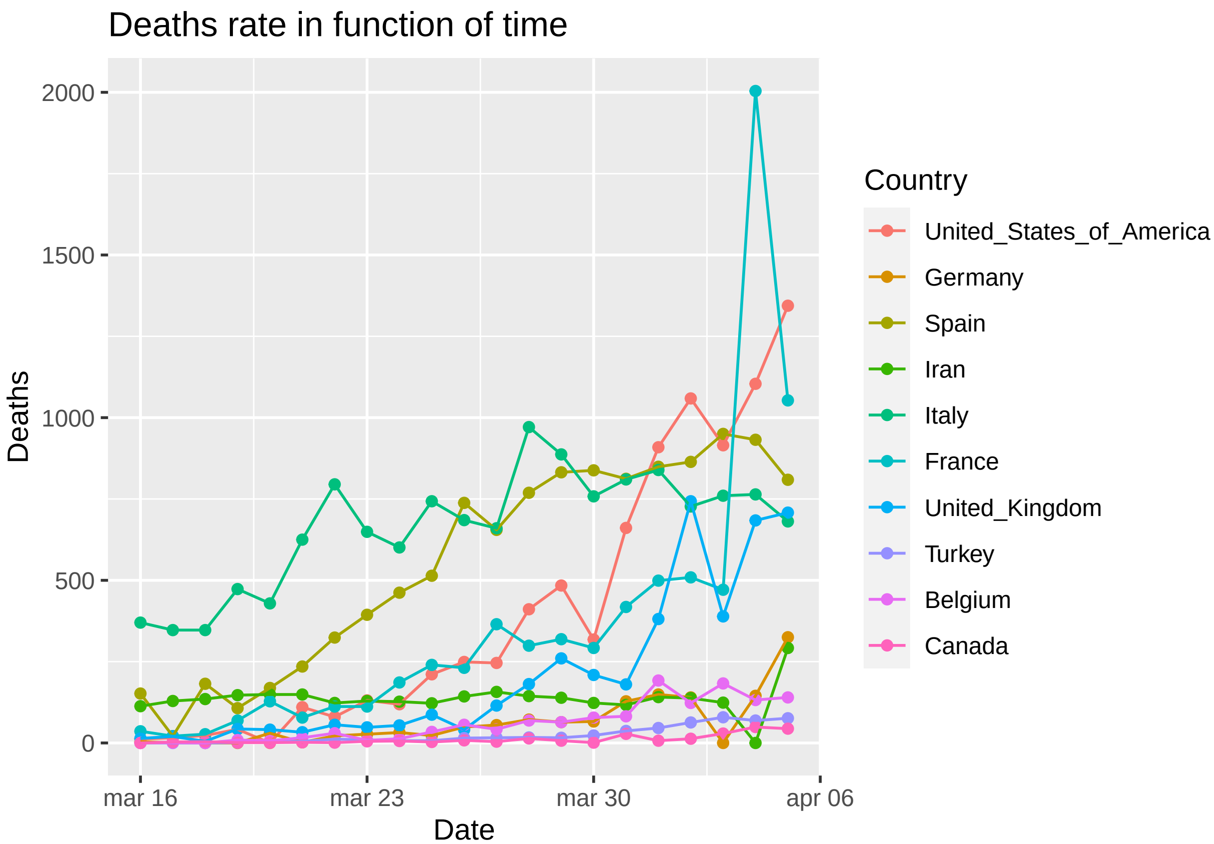
```
  deaths_dataframe <- cbind(deaths_dataframe, deaths_country )
}
colnames(deaths_dataframe)[1] <- "Date"
colnames(deaths_dataframe)[2:ncol(deaths_dataframe)] <- highest$countriesAndTerritories
```

Plot functions

```
library("ggplot2")
library("reshape2")
cases_melted <- melt(cases_dataframe ,  id.vars = 'Date', variable = "Country")
ggplot(data= cases_melted , aes(x=as.Date(Date), y=value, color=Country)) +
       geom_point() + geom_line() +
       labs(title = "Cases rate in function of time", x = "Date", y = "Cases")
```




```
deaths_melted <- melt(deaths_dataframe ,  id.vars = 'Date', variable = "Country")
ggplot(data = deaths_melted , aes(x=as.Date(Date), y=value, color = Country)) +
  geom_point() + geom_line() +
  labs(title = "Deaths rate in function of time", x = "Date", y = "Deaths")
```



Since Brazil and Israels hold some missing data, let's delete them from the countries found before in order to have a longer-range date graphs.

```
countries <- highest$countriesAndTerritories[highest$countriesAndTerritories != 'Turkey']
countries <- countries[countries != 'Israel']

t0 <- min(covid$dateRep)
for (i in countries) {
  data_country <- (subset(covid, countriesAndTerritories == i))
  ifelse( min(data_country$dateRep) > t0, t0 <- min(data_country$dateRep), t0 )
}
shift_date <- t0
```

Samefunctions as before in order to create dataframes for the plots:

```
cases_dataframe <- tibble(unique(covid$dateRep[covid$dateRep >= shift_date]))
for (i in countries) {
  cases_country <- (subset(covid, (countriesAndTerritories == i) & (dateRep >= shift_date)))['cases']
  print(i)
  cases_dataframe <- cbind(cases_dataframe, cases_country )
}
```

```
## [1] "United_States_of_America"
## [1] "Germany"
## [1] "Spain"
## [1] "Iran"
## [1] "Italy"
## [1] "France"
## [1] "United_Kingdom"
```

```
## [1] "Belgium"
## [1] "Canada"
```

```r
colnames(cases_dataframe)[1] <- "Date"
colnames(cases_dataframe)[2:ncol(cases_dataframe)] <- countries

deaths_dataframe <- tibble(unique(covid$dateRep[covid$dateRep >= shift_date]))
for (i in countries) {
  deaths_country <- (subset(covid, (countriesAndTerritories == i) & (dateRep >= shift_date)))['deaths']
  deaths_dataframe <- cbind(deaths_dataframe, deaths_country )
}
colnames(deaths_dataframe)[1] <- "Date"
colnames(deaths_dataframe)[2:ncol(deaths_dataframe)] <- countries
```

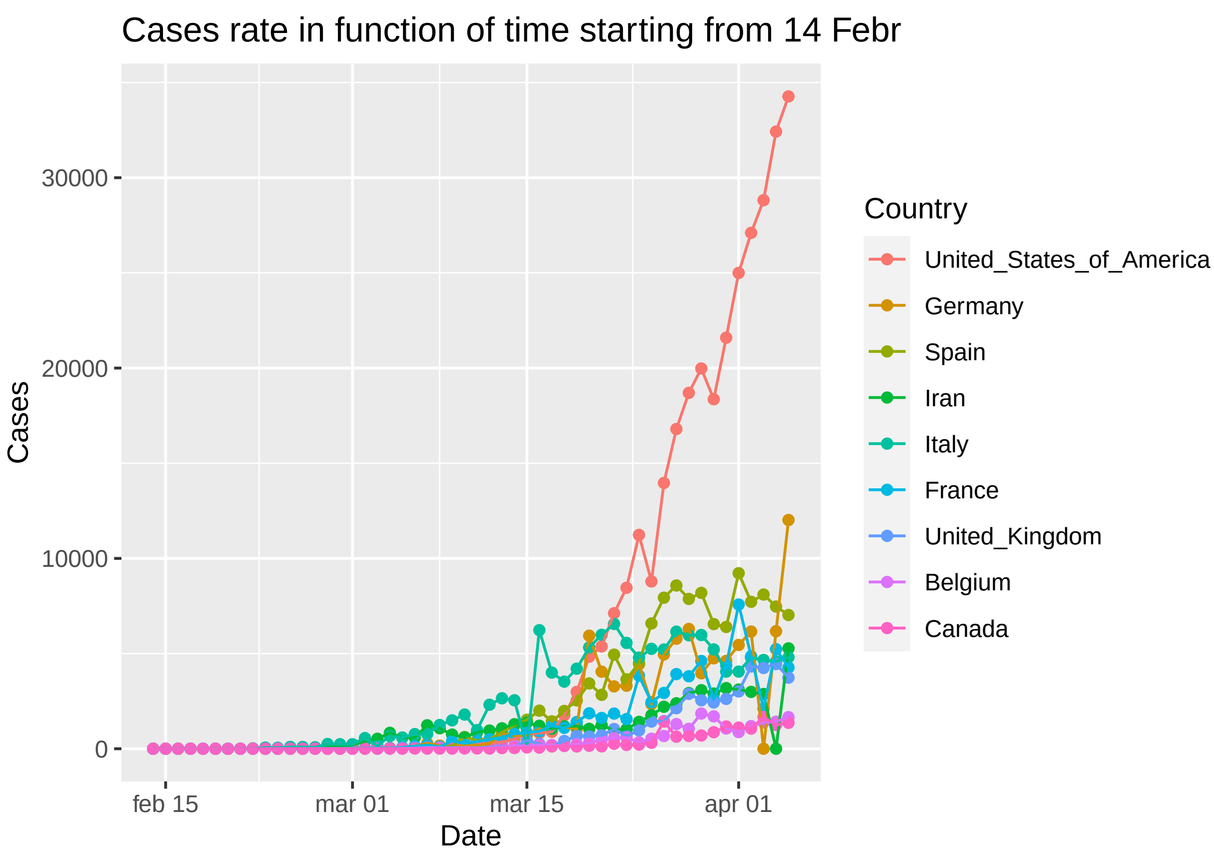
Let's choose as offset (i.e. starting point), 14th of February:

```r
offset <- as.Date("2020-02-14")
```

Same functions for plotting as before:

```r
cases_melted <- melt(cases_dataframe ,  id.vars = 'Date', variable = "Country")
ggplot(data= cases_melted , aes(x=as.Date(Date), y=value, color=Country)) +
  geom_point() + geom_line() + scale_x_date(limits = c(offset, date)) +
  labs(title = "Cases rate in function of time starting from 14 Febr", x = "Date", y = "Cases")
```

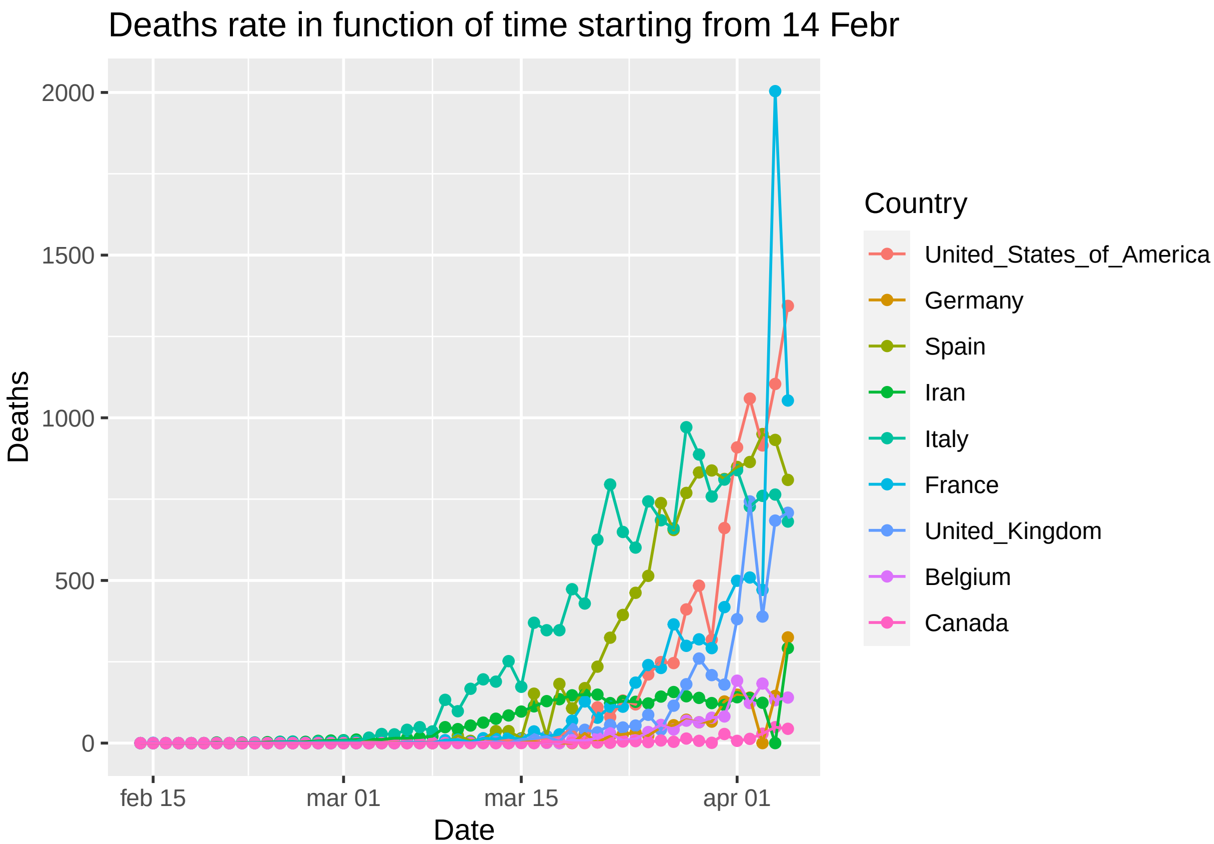
```
## Warning: Removed 405 rows containing missing values (geom_point).

## Warning: Removed 405 row(s) containing missing values (geom_path).
```



```r
deaths_melted <- melt(deaths_dataframe ,  id.vars = 'Date', variable = "Country")
ggplot(data = deaths_melted , aes(x=as.Date(Date), y=value, color = Country)) +
  geom_point() + geom_line() + scale_x_date(limits = c(offset, date)) +
  labs(title = "Deaths rate in function of time starting from 14 Febr", x = "Date", y = "Deaths")
```

```
## Warning: Removed 405 rows containing missing values (geom_point).
```

```
## Warning: Removed 405 row(s) containing missing values (geom_path).
```



“‘

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.