# MyFirstDatewithR
# ExerciseBook

# Contents


**1 Introduction** **5**

**2 Your first R session** **7**
- 2.1 Aritmetic with R . . . 7
- 2.2 Assignment . . . 7

**3 Data Objects** **9**
- 3.1 Data Frames, Vectors and Factors . . . 9
- 3.2 Matrices . . . 11
- 3.3 Lists . . . 11

**4 Data Import from external sources** **13**
- 4.1 Text Files . . . 13
- 4.2 Excel Files . . . 14
- 4.3 Databases . . . 14

**5 Data Manipulation with R** **17**
- 5.1 Data . . . 17
- 5.2 `select()` . . . 19
- 5.3 `filter()` . . . 20
- 5.4 `arrange()` . . . 21

**6 Data Discovery with R** **25**
- 6.1 Data . . . 25
- 6.2 Descriptive statistics with `summarise()` and `group_by()` . . . 25
- 6.3 Multiple operations . . . 26

**7 Data Visualization with `ggplot2`** **29**
- 7.1 Data . . . 29
- 7.2 Scatterplot . . . 32
- 7.3 Barplot . . . 34
- 7.4 Histogram . . . 37
- 7.5 Boxplot . . . 39
- 7.6 Lineplot . . . 41

**8 Statistical models** **43**
- 8.1 Linear Models . . . 43

**9 Data Mining** **49**
- 9.1 Neural Networks . . . 49

# Chapter 1

# Introduction

In this document you will find some exercises about these sections:

- *Your First R session*
- *Data Objects*
- *Data Import from external sources*
- *Data Manipulation with R*
- *Data Discovery with R*
- *Data Visualization with R*
- *Statistical Models with R*
- *Data Mining with R*

# Chapter 2

# Your first R session

## 2.1 Aritmetic with R

### 2.1.1 Exercise 1

Calculate your body mass index dividing your body mass (kg) by the square of your body height (m) ($kg/m^2$)

```
65/1.7^2
```

```
## [1] 22.49135
```

## 2.2 Assignment

### 2.2.1 Exercise 1

a. Assign your age (in number) to `age` variable.

```
age <- 26
```

c. Print out the value of the variable `age`.

```
age # o print(age)
```

```
## [1] 26
```

d. Remove the variable `age` from the workspace, by using `rm()` function.

```
rm(age)
```

### 2.2.2 Exercise 2

Suppose you want to buy 10 roses and 8 sunflowers in a flower shop. The roses cost 3 euros each and the sunflowers 2 euros each.

a. Assign the total cost of roses to `roses_cost` variable and the total cost of sunflowers to `sunflowers_cost` variable.

```
roses_cost <- 10 * 3
sunflowers_cost <- 8 * 2
```

b. Calculate the total cost of flowers by adding `roses_cost` and `sunflowers_cost` variables and assign it to `flowers_cost` variable.

```
flowers_cost <- roses_cost + sunflowers_cost
```

c. Print out the value of the variable `flowers_cost`.

```
flowers_cost
```

```
## [1] 46
```

d. List the objects in the current R session, by using `ls()` function.

```
ls() # or objects()
```

```
## [1] "flowers_cost"    "roses_cost"      "show_solution"   "sunflowers_cost"
```

# Chapter 3

# Data Objects

## 3.1 Data Frames, Vectors and Factors

### 3.1.1 Exercise 1

a. Generate a data frame, named `df`, corresponding to:

```
df <- data.frame(country = c("Italy", "France", "China", "Japan", "Libya", "Cameroon"),
                 population = c(59801004, 64668129, 1382323332, 126323715, 6330159, 23924407),
                 continent = c("Europe", "Europe", "Asia", "Asia", "Africa", "Africa"),
                 stringsAsFactors = FALSE)
df
```

```
##    country population continent
## 1    Italy   59801004    Europe
## 2   France   64668129    Europe
## 3    China 1382323332      Asia
## 4    Japan  126323715      Asia
## 5    Libya    6330159    Africa
## 6 Cameroon   23924407    Africa
```

Remember to maintain character vectors as they are, specifiyng `stringsAsFactors = FALSE`.

```
df <- data.frame(country = c("Italy", "France", "China", "Japan", "Libya", "Cameroon"),
                 population = c(59801004, 64668129, 1382323332, 126323715, 6330159, 23924407),
                 continent = c("Europe", "Europe", "Asia", "Asia", "Africa", "Africa"),
                 stringsAsFactors = FALSE)
df
```

```
##    country population continent
## 1    Italy   59801004    Europe
## 2   France   64668129    Europe
## 3    China 1382323332      Asia
## 4    Japan  126323715      Asia
## 5    Libya    6330159    Africa
## 6 Cameroon   23924407    Africa
```

b. Supposing `dplyr` package is already installed, convert the previously defined data frame in tbl_df class.

```
require(dplyr)
```

```
df <- tbl_df(df)
```

### 3.1.2 Exercise 2

a. Generate a numeric vector, named `num_vec`, containing the values from 1 to 7.

```
num_vec <- c(1, 2, 3, 4, 5, 6, 7)
```

b. Genarate a character vector, named `char_vec` with the days of the week.

```
char_vec <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
```

c. Starting from the vector:

    ```
    fac <- c("F", "F", "M", "M", "F", "F", "M")
    ```

    Generate the corresponding factor, named `fac`, with two levels: “F” and “M”

```
fac <- factor(fac, levels = c("F", "M"))
fac
```

```
## [1] F F M M F F M
## Levels: F M
```

d. Generate a data frame, named `df2`, containing the previously defined: `num_vec`, `char_vec` and `fac`. Remember to maintain character vectors as they are, specifiyng `stringsAsFactors = FALSE`.

```
df <- data.frame(num_vec, char_vec, fac, stringsAsFactors = FALSE)
df
```

```
##   num_vec  char_vec fac
## 1       1    Monday   F
## 2       2   Tuesday   F
## 3       3 Wednesday   M
## 4       4  Thursday   M
## 5       5    Friday   F
## 6       6  Saturday   F
## 7       7    Sunday   M
```

e. Supposing `dplyr` package is already installed and loaded, convert the previously defined data frame in tbl_df class.

```
df <- df %>% tbl_df()
df
```

```
## # A tibble: 7 × 3
##   num_vec  char_vec    fac
##     <dbl>     <chr> <fctr>
## 1       1    Monday      F
## 2       2   Tuesday      F
## 3       3 Wednesday      M
## 4       4  Thursday      M
## 5       5    Friday      F
## 6       6  Saturday      F
## 7       7    Sunday      M
```

## 3.2 Matrices

### 3.2.1 Exercise 1

Generate a matrix, named `mat`, with 5 rows and 3 columns containing numbers from 1 to 15, using `matrix()` function.

```
mat <- matrix(1:15, nrow = 5, ncol = 3, byrow = TRUE)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
## [5,]   13   14   15
```

## 3.3 Lists

### 3.3.1 Exercise 1

Generate a list, named `my_list` that contains the following R elements:

```
char <- "Veronica"
mat <- matrix(1:9, ncol = 3)
log_vec <- c(TRUE, FALSE, TRUE, TRUE)
```

```
my_list <- list(char <- "Veronica",
               mat = matrix(1:9, ncol = 3),
               log_vec = c(TRUE, FALSE, TRUE, TRUE))
my_list
```

```
## [[1]]
## [1] "Veronica"
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## $log_vec
## [1]  TRUE FALSE  TRUE  TRUE
```

# Chapter 4

# Data Import from external sources

First of all, set your working directory in the *data* folder, using `setwd()` function, like in this example

```
setwd("C:/Users/Veronica/Documents/rbase/data")
```

We will work inside this folder.

## 4.1 Text Files

### 4.1.1 Exercise 1

a. Import text file named *"tuscany.txt"* and save it in an R object named `tuscany_df`.
   Open the text file before importing it to control if the first row contains column names and to control the field and the decimal separator characters. Remember to not import the character columns as factors.

```
tuscany_df <- read.table("tuscany.txt", header = TRUE, sep = "|",
                         dec=".", stringsAsFactors = FALSE)
```

b. Visualize the first rows of `tuscany_df`

```
head(tuscany_df)
```

```
##   id sex year_of_birth marital_status  income house_number
## 1  1   M          1969        married 16101.1       5144.0
## 2  2   M          1962         single 17220.0       6158.0
## 3  3   M          1965       divorcee 28801.9      10078.0
## 4  4   F          1968         single 25964.0      11133.7
## 5  5   M          1975        married 16522.5       5078.0
## 6  6   M          1977        married 18124.0       5115.0
##               city_name province provincial_acronym
## 1            Riparbella     Pisa                 PI
## 2              Capolona   Arezzo                 AR
## 3             Pomarance     Pisa                 PI
## 4               Cascina     Pisa                 PI
## 5              Quarrata  Pistoia                 PT
## 6 Castiglion Fiorentino   Arezzo                 AR
```

### 4.1.2 Exercise 2

a. Import text file named *"solar.txt"* and save it in an R object `solar_df`.
   Open the text file before importing it to control if the first row contains column names and to control the field and the decimal separator characters. Remember to not import the character columns as factors.

```
solar_df <- read.table("solar.txt", header = FALSE, sep = ",",
                       dec=".", stringsAsFactors = FALSE)
```

b. Visualize the first rows of `solar_df`.

```
head(solar_df)
```

```
##    V1    V2    V3    V4
## 1 gen 25677 24677 24567
## 2 feb 24044 25988 24376
## 3 mar 23877 24671 22455
## 4 apr 24377 23677 23670
## 5 mag 24581 25476 24999
## 6 giu 22154 21998 22451
```

## 4.2 Excel Files

### 4.2.1 Exercise 1

a. Import `iris` sheet of .xlsx file *"flowers.xlsx"* by using `read_excel` function of `readxl` package and save it in a R object named `flowers`.
   Remember to load `read_excel` package, supposing it is already installed.

```
require(readxl)
```

```
flowers <-read_excel(path = "flowers.xlsx", sheet = 'iris', col_names = TRUE)
```

b. Visualize the first rows of `flowers`

```
head(flowers)
```

```
## # A tibble: 6 × 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl>   <chr>
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

## 4.3 Databases

### 4.3.1 Exercise 1

a. Connect to *"plant.sqlite"* SQLite database, using `dbConnect()` function of `RSQLite` package. Save the connection in an R object, named `con`.

Remember to load `RSQLite` package, supposing it is already installed.

```
require(RSQLite)
```

```
con <- dbConnect(RSQLite::SQLite(), "plant.sqlite")
```

b. See the list of available tables in *"plant.sqlite"* db, using `dbListTables()` function.

```
dbListTables(con)
```

```
## [1] "PlantGrowth"
```

c. See list of fields in *"PlantGrowth"* table of *"plant.sqlite"* db, using `dbListFields()` function.

```
dbListFields(con, name = "PlantGrowth")
```

```
## [1] "weight" "group"
```

d. Send query to *"PlantGrowth"* table of *"plant.sqlite"* which select the records with `weight` greater than 5.5.

```
dbGetQuery(con, "SELECT * FROM PlantGrowth WHERE weight >= 5.5")
```

```
##   weight group
## 1   5.58  ctrl
## 2   6.11  ctrl
## 3   5.87  trt1
## 4   6.03  trt1
## 5   6.31  trt2
## 6   5.54  trt2
## 7   5.50  trt2
## 8   6.15  trt2
## 9   5.80  trt2
```

e. Disconnect from the database, using `dbDisconnect()` function.

```
dbDisconnect(con)
```

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

# Chapter 5

# Data Manipulation with R

Load `dplyr` package, supposing it is already installed.

```
require(dplyr)
```

## 5.1 Data

All the following exercises are based on the `nycflights13` data, taken from the `nycflights13` package. So first of all, install and load this package

```
install.packages("nycflights13")
require(nycflights13)
```

The `nycflights13` package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013: 336,776 flights in total.

```
ls(pos = "package:nycflights13")
```

```
## [1] "airlines" "airports" "flights"  "planes"   "weather"
```

To help understand what causes delays, it includes a number of useful datasets:

- `flights`: information about all flights that departed from NYC
- `weather`: hourly meterological data for each airport;
- `planes`: construction information about each plane;
- `airports`: airport names and locations;
- `airlines`: translation between two letter carrier codes and names.

Let us explore the features of `flights` datasets, which will be used in the following exercises.

```
data("flights")
```

### 5.1.1 flights

This dataset contains on-time data for all flights that departed from NYC (i.e. JFK, LGA or EWR) in 2013. The data frame has 16 variables and 336776 observations. The variables are organised as follow:

- Date of departure: `year`, `month`, `day`;

- Departure and arrival times (local tz): `dep_time`, `arr_time`;
- Departure and arrival delays, in minutes: `dep_delay`, `arr_delay` (negative times represent early departures/arrivals);
- Time of departure broken in to hour and minutes: `hour`, `minute`;
- Two letter carrier abbreviation: `carrier`;
- Plane tail number: `tailnum`;
- Flight number: `flight`;
- Origin and destination: `origin`, `dest`;
- Amount of time spent in the air: `air_time`;
- Distance flown: `distance`.

```
dim(flights)
```

```
## [1] 336776     16
```

```
head(flights)
```

```
##   year month day dep_time dep_delay arr_time arr_delay carrier tailnum flight
## 1 2013     1   1      517         2      830        11      UA  N14228   1545
## 2 2013     1   1      533         4      850        20      UA  N24211   1714
## 3 2013     1   1      542         2      923        33      AA  N619AA   1141
## 4 2013     1   1      544        -1     1004       -18      B6  N804JB    725
## 5 2013     1   1      554        -6      812       -25      DL  N668DN    461
## 6 2013     1   1      554        -4      740        12      UA  N39463   1696
##   origin dest air_time distance hour minute
## 1    EWR  IAH      227     1400    5     17
## 2    LGA  IAH      227     1416    5     33
## 3    JFK  MIA      160     1089    5     42
## 4    JFK  BQN      183     1576    5     44
## 5    LGA  ATL      116      762    5     54
## 6    EWR  ORD      150      719    5     54
```

```
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  16 variables:
##  $ year     : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ dep_delay: num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ arr_delay: num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier  : chr  "UA" "UA" "AA" "B6" ...
##  $ tailnum  : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ flight   : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ origin   : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest     : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance : num  1400 1416 1089 1576 762 ...
##  $ hour     : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ minute   : num  17 33 42 44 54 54 55 57 57 58 ...
```

## 5.2 select()

### 5.2.1 Exercise 1

Extract the following information:

- month;
- day;
- air_time;
- distance.

```
select(flights, month, day, air_time, distance)
```

```
## # A tibble: 336,776 × 4
##    month   day air_time distance
##    <int> <int>    <dbl>    <dbl>
## 1      1     1      227     1400
## 2      1     1      227     1416
## 3      1     1      160     1089
## 4      1     1      183     1576
## 5      1     1      116      762
## 6      1     1      150      719
## 7      1     1      158     1065
## 8      1     1       53      229
## 9      1     1      140      944
## 10     1     1      138      733
## # ... with 336,766 more rows
```

```
# flights %>% select(month, day, air_time, distance)
```

### 5.2.2 Exercise 2

Extract all information about `flights` except hour and minute.

```
select(flights, -c(hour, minute))
```

```
## # A tibble: 336,776 × 17
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
## 2   2013     1     1      533            529         4      850            830
## 3   2013     1     1      542            540         2      923            850
## 4   2013     1     1      544            545        -1     1004           1022
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 9 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, time_hour <dttm>
```

```r
# flights %>% select(-c(hour, minute))
```

### 5.2.3 Exercise 3

Extract `tailnum` variable and rename it into `tail_num`

```r
select(flights, tail_num=tailnum)
```

```
## # A tibble: 336,776 × 1
##    tail_num
##       <chr>
## 1    N14228
## 2    N24211
## 3    N619AA
## 4    N804JB
## 5    N668DN
## 6    N39463
## 7    N516JB
## 8    N829AS
## 9    N593JB
## 10   N3ALAA
## # ... with 336,766 more rows
```

```r
# flights %>% select(tail_num=tailnum)
```

## 5.3 `filter()`

### 5.3.1 Exercise 1

Select all flights which delayed more than 1000 minutes at departure.

```r
filter(flights, dep_delay > 1000)
```

```
## # A tibble: 5 × 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     9      641            900      1301     1242           1530
## 2  2013     1    10     1121           1635      1126     1239           1810
## 3  2013     6    15     1432           1935      1137     1607           2120
## 4  2013     7    22      845           1600      1005     1044           1815
## 5  2013     9    20     1139           1845      1014     1457           2210
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# flights %>% filter(dep_delay > 1000)
```

### 5.3.2 Exercise 2

Select all flights which delayed more than 1000 minutes at departure or at arrival.

```r
filter(flights, dep_delay > 1000 | arr_delay >1000)
```

```
## # A tibble: 5 × 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     9      641            900      1301     1242           1530
## 2  2013     1    10     1121           1635      1126     1239           1810
## 3  2013     6    15     1432           1935      1137     1607           2120
## 4  2013     7    22      845           1600      1005     1044           1815
## 5  2013     9    20     1139           1845      1014     1457           2210
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# flights %>% filter(dep_delay > 1000 | arr_delay >1000)
```

### 5.3.3 Exercise 3

Select all flights which took off from "EWR" and landed in "IAH".

```r
filter(flights, origin == "EWR" & dest == "IAH")
```

```
## # A tibble: 3,973 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
## 2   2013     1     1      739            739         0     1104           1038
## 3   2013     1     1      908            908         0     1228           1219
## 4   2013     1     1     1044           1045        -1     1352           1351
## 5   2013     1     1     1205           1200         5     1503           1505
## 6   2013     1     1     1356           1350         6     1659           1640
## 7   2013     1     1     1527           1515        12     1854           1810
## 8   2013     1     1     1620           1620         0     1945           1922
## 9   2013     1     1     1725           1720         5     2045           2021
## 10  2013     1     1     1959           2000        -1     2310           2307
## # ... with 3,963 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# flights %>% filter(origin == "EWR" & dest == "IAH")
```

## 5.4 arrange()

### 5.4.1 Exercise 1

Sort the flights in chronological order.

```r
arrange(flights, year, month, day)
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     1      517            515         2      830            819
```

```
## 2   2013     1     1      533            529         4      850            830
## 3   2013     1     1      542            540         2      923            850
## 4   2013     1     1      544            545        -1     1004           1022
## 5   2013     1     1      554            600        -6      812            837
## 6   2013     1     1      554            558        -4      740            728
## 7   2013     1     1      555            600        -5      913            854
## 8   2013     1     1      557            600        -3      709            723
## 9   2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# flights %>% arrange(year, month, day)
```

### 5.4.2 Exercise 2

Sort the flights by decreasing arrival delay.

```
arrange(flights, desc(arr_delay))
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1     9      641            900      1301     1242           1530
## 2   2013     6    15     1432           1935      1137     1607           2120
## 3   2013     1    10     1121           1635      1126     1239           1810
## 4   2013     9    20     1139           1845      1014     1457           2210
## 5   2013     7    22      845           1600      1005     1044           1815
## 6   2013     4    10     1100           1900       960     1342           2211
## 7   2013     3    17     2321            810       911      135           1020
## 8   2013     7    22     2257            759       898      121           1026
## 9   2013    12     5      756           1700       896     1058           2020
## 10  2013     5     3     1133           2055       878     1250           2215
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# flights %>% arrange(desc(arr_delay))
```

### 5.4.3 Exercise 3

Sort the flights by origin (in alphabetical order) and decreasing arrival delay.

```
arrange(flights, origin, desc(arr_delay))
```

```
## # A tibble: 336,776 × 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1   2013     1    10     1121           1635      1126     1239           1810
## 2   2013    12     5      756           1700       896     1058           2020
## 3   2013     5     3     1133           2055       878     1250           2215
## 4   2013    12    19      734           1725       849     1046           2039
## 5   2013    12    17      705           1700       845     1026           2020
```

```
## 6   2013    11     3      603            1645       798      829            1913
## 7   2013     2    24     1921             615       786     2135             842
## 8   2013    10    14     2042             900       702     2255            1127
## 9   2013     7    21     1555             615       580     1955             910
## 10  2013     7     7     2123            1030       653       17            1345
## # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# flights %>% arrange(origin, desc(arr_delay))
```

# Chapter 6

# Data Discovery with R

Load `dplyr` package, supposing it is already installed.

```
require(dplyr)
```

## 6.1 Data

Also these exercises are based on the `nycflights13` data, taken from the `nycflights13` package.
Load `nycflights13` package, supposing it is already installed.

```
require(nycflights13)
```

The `nycflights13` package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013: 336,776 flights in total. For more information see *Data Manipulation with R* section.

The following exercises refers to `flights` dataset:

```
data("flights")
```

## 6.2 Descriptive statistics with `summarise()` and `group_by()`

### 6.2.1 Exercise 1

Calculate the mean delay at arrival (`arr_delay` variable). Remember to add `na.rm=TRUE` option to all calculations.

```
summarise(flights, mean_delay = mean(arr_delay, na.rm=TRUE))
```

```
## # A tibble: 1 × 1
##   mean_delay
##        <dbl>
## 1   6.895377
```

```
# flights %>% summarise(mean_delay = mean(arr_delay, na.rm=TRUE))
```

### 6.2.2 Exercise 2

Calculate the summary (minimum, first quartile, median, mean, third quartile, maximum and standard deviation) of delay at departure (`dep_delay` variable) for flights.
Remember to add `na.rm=TRUE` option to mean calculations.

```
flights %>%
  summarise(min=min(dep_delay, na.rm = TRUE),
            first_q=quantile(dep_delay, prob = 0.25, na.rm = TRUE),
            median=median(dep_delay, na.rm = TRUE),
            mean=mean(dep_delay, na.rm =TRUE),
            third_q=quantile(dep_delay, prob = 0.75, na.rm = TRUE),
            max=max(dep_delay, na.rm = TRUE),
            sd=sd(dep_delay, na.rm =TRUE))
```

```
## # A tibble: 1 × 7
##     min first_q median     mean third_q   max       sd
##   <dbl>   <dbl>  <dbl>    <dbl>   <dbl> <dbl>    <dbl>
## 1   -43      -5     -2 12.63907      11  1301 40.21006
```

### 6.2.3 Exercise 3

Calculate minimum and maximum delay at departure (`arr_delay` variable) for flights by month.
Remember to add `na.rm=TRUE` option to all calculations.

```
flights %>%
  group_by(month) %>%
  summarise(min_delay = min(dep_delay, na.rm=TRUE),
     max_delay = max(dep_delay, na.rm=TRUE))
```

```
## # A tibble: 12 × 3
##    month min_delay max_delay
##    <int>     <dbl>     <dbl>
## 1      1       -30      1301
## 2      2       -33       853
## 3      3       -25       911
## 4      4       -21       960
## 5      5       -24       878
## 6      6       -21      1137
## 7      7       -22      1005
## 8      8       -26       520
## 9      9       -24      1014
## 10    10       -25       702
## 11    11       -32       798
## 12    12       -43       896
```

## 6.3 Multiple operations

### 6.3.1 Exercise 1

For each destination (`dest` variable), compute the mean delay at arrival (`arr_delay` variable) and filter the mean delays greater than 30 minutes.
Remember to add `na.rm=TRUE` option to mean calculations.

```
flights %>% group_by(dest)  %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm=TRUE)) %>%
  filter(mean_arr_delay > 30)
```

```
## # A tibble: 3 × 2
##    dest mean_arr_delay
##   <chr>          <dbl>
## 1   CAE       41.76415
## 2   OKC       30.61905
## 3   TUL       33.65986
```

### 6.3.2 Exercise 2

Filter the observations recorded on June 13 and count the number of flights (use `n()` function inside `summarise()`) for each destination. Then sort the result in ascending order.

```
flights %>% filter(month == 6 & day == 13)  %>%
  group_by(dest) %>%
  summarise(n_flight = n()) %>%
  arrange(n_flight)
```

```
## # A tibble: 89 × 2
##     dest n_flight
##    <chr>    <int>
## 1    ABQ        1
## 2    ACK        1
## 3    ALB        1
## 4    AVL        1
## 5    BGR        1
## 6    BHM        1
## 7    BUR        1
## 8    CAE        1
## 9    GRR        1
## 10   MSN        1
## # ... with 79 more rows
```

# Chapter 7

# Data Visualization with ggplot2

Load ggplot2 package, supposing it is already installed.

```
require(ggplot2)
```

## 7.1 Data

### 7.1.1 iris

Almost all the following exercises are based on the iris dataset, taken from the datasets package. It is a base package so it is already installed and loaded.

```
data("iris")
```

This dataset gives the measurements in centimeters of length and width of sepal and petal, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

iris dataset contains the following variables:

- Sepal.Length: length of iris sepal
- Sepal.Width: width of iris sepal
- Petal.Length: length of iris petal
- Petal.Width: width of iris petal
- Species: species of iris

```
dim(iris)
```

```
## [1] 150   5
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

### 7.1.2 mpg

Some of the exercises are based on `mpg` dataset, taken from the `ggplot2` package.

```r
data("mpg")
```

This dataset contains the fuel economy data from 1999 and 2008 for 38 popular models of car. `mpg` dataset contains the following variables:

- `manufacturer`
- `model`
- `displ`: engine displacement, in litres
- `year`
- `cyl`: number of cylinders
- `trans`: type of transmission
- `drv`: drivetrain type, f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- `cty`: city miles per gallon
- `hwy`: highway miles per gallon
- `fl`: fuel type

```r
dim(mpg)
```

```
## [1] 234  11
```

```r
head(mpg)
```

```
## # A tibble: 6 × 11
##   manufacturer model displ  year   cyl      trans   drv   cty   hwy    fl
##          <chr> <chr> <dbl> <int> <int>      <chr> <chr> <int> <int> <chr>
## 1         audi    a4   1.8  1999     4   auto(l5)     f    18    29     p
## 2         audi    a4   1.8  1999     4 manual(m5)     f    21    29     p
## 3         audi    a4   2.0  2008     4 manual(m6)     f    20    31     p
## 4         audi    a4   2.0  2008     4   auto(av)     f    21    30     p
## 5         audi    a4   2.8  1999     6   auto(l5)     f    16    26     p
## 6         audi    a4   2.8  1999     6 manual(m5)     f    18    26     p
## # ... with 1 more variables: class <chr>
```

```r
str(mpg)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    234 obs. of  11 variables:
##  $ manufacturer: chr  "audi" "audi" "audi" "audi" ...
##  $ model       : chr  "a4" "a4" "a4" "a4" ...
##  $ displ       : num  1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
```

```
## $ year         : int  1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
## $ cyl          : int  4 4 4 4 6 6 6 4 4 4 ...
## $ trans        : chr  "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
## $ drv          : chr  "f" "f" "f" "f" ...
## $ cty          : int  18 21 20 21 16 18 18 18 16 20 ...
## $ hwy          : int  29 29 31 30 26 26 27 26 25 28 ...
## $ fl           : chr  "p" "p" "p" "p" ...
## $ class        : chr  "compact" "compact" "compact" "compact" ...
```

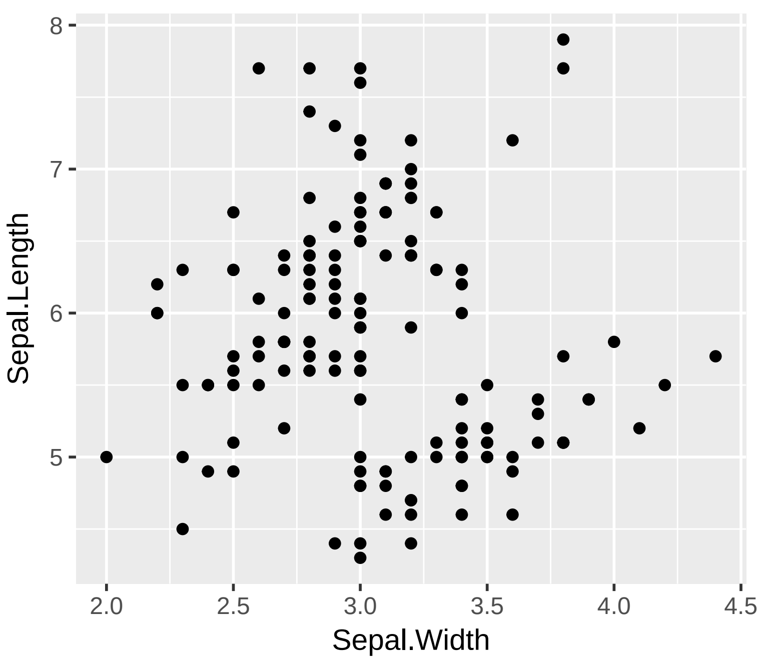
## 7.2 Scatterplot

### 7.2.1 Exercise 1

Let us consider `iris` dataset.

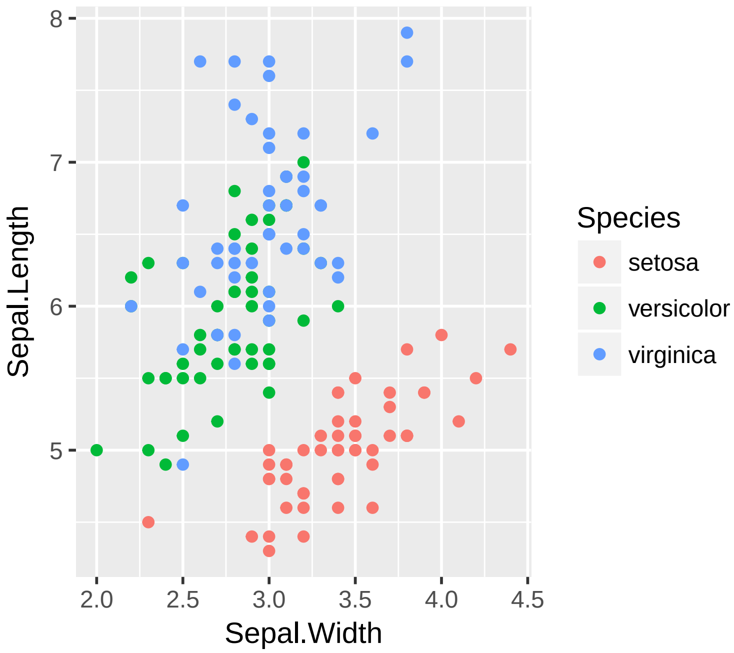
a. Generate a scatterplot to analyze the relationship between `Sepal.Width` and `Sepal.Length` variables.

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length)) +
        geom_point()
pl
```



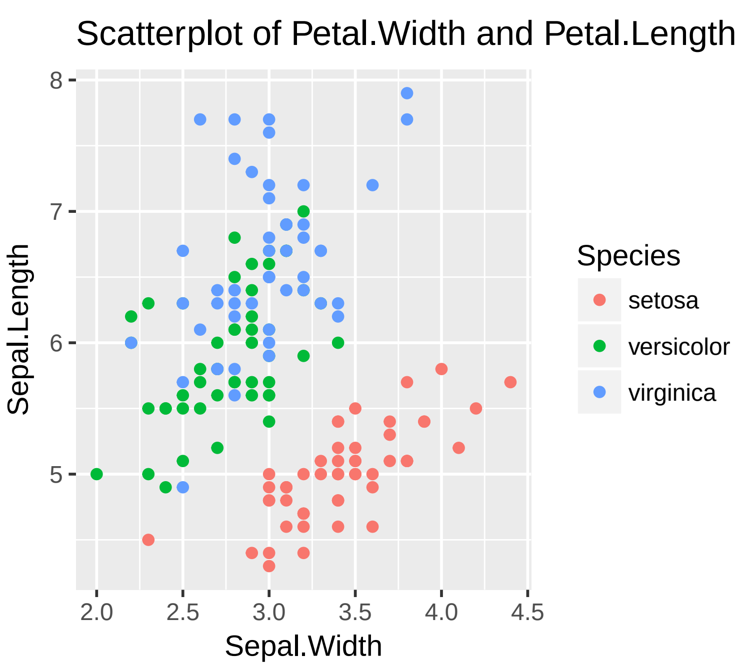

b. Map `Species` to `colour` in `aes()`.

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length, colour=Species)) +
  geom_point()

pl
```



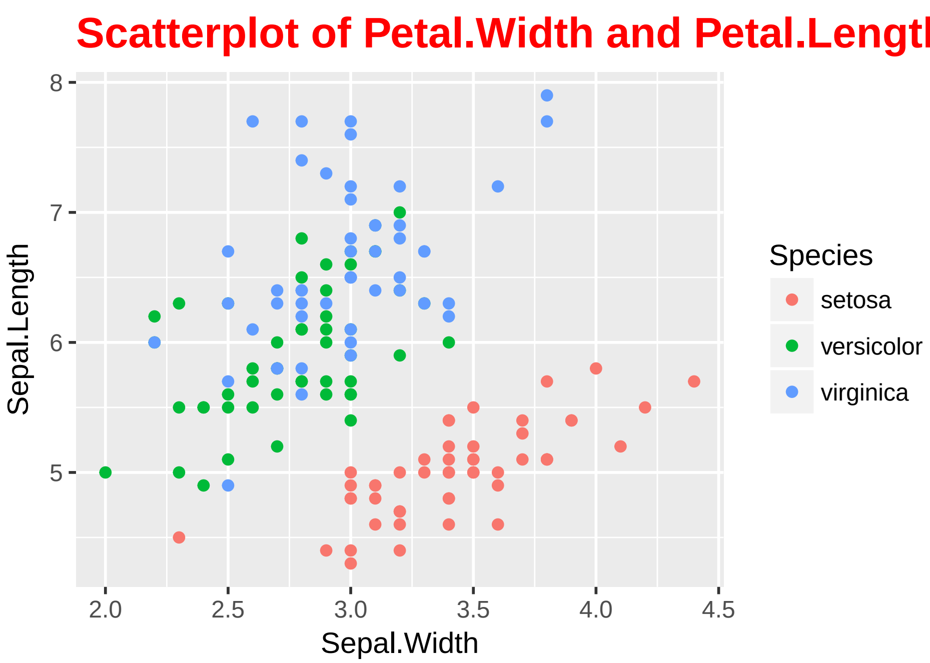

c. Add the title to the plot: `"Scatterplot of Petal.Width and Petal.Length"` (use `ggtitle()` function).

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length, colour=Species)) +
  geom_point() +
  ggtitle("Scatterplot of Petal.Width and Petal.Length")

pl
```




d. Customize plot title by adding `theme(plot.title = element_text())` to the plot and setting `colour` argument to `"red"`, `size` to 16 and `face` to `"bold"`.

```
pl <- ggplot(data = iris, mapping = aes(x=Sepal.Width, y=Sepal.Length, colour=Species)) +
  geom_point() +
  ggtitle("Scatterplot of Petal.Width and Petal.Length") +
  theme(plot.title = element_text(colour = "red", size = 16, face = "bold"))

pl
```



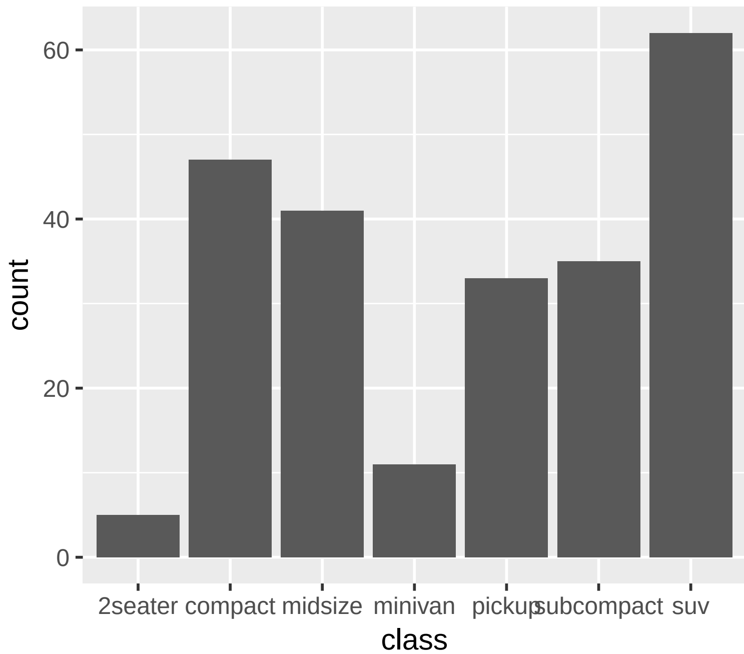

## 7.3 Barplot

### 7.3.1 Exercise 1

Let us consider `mpg` dataset.

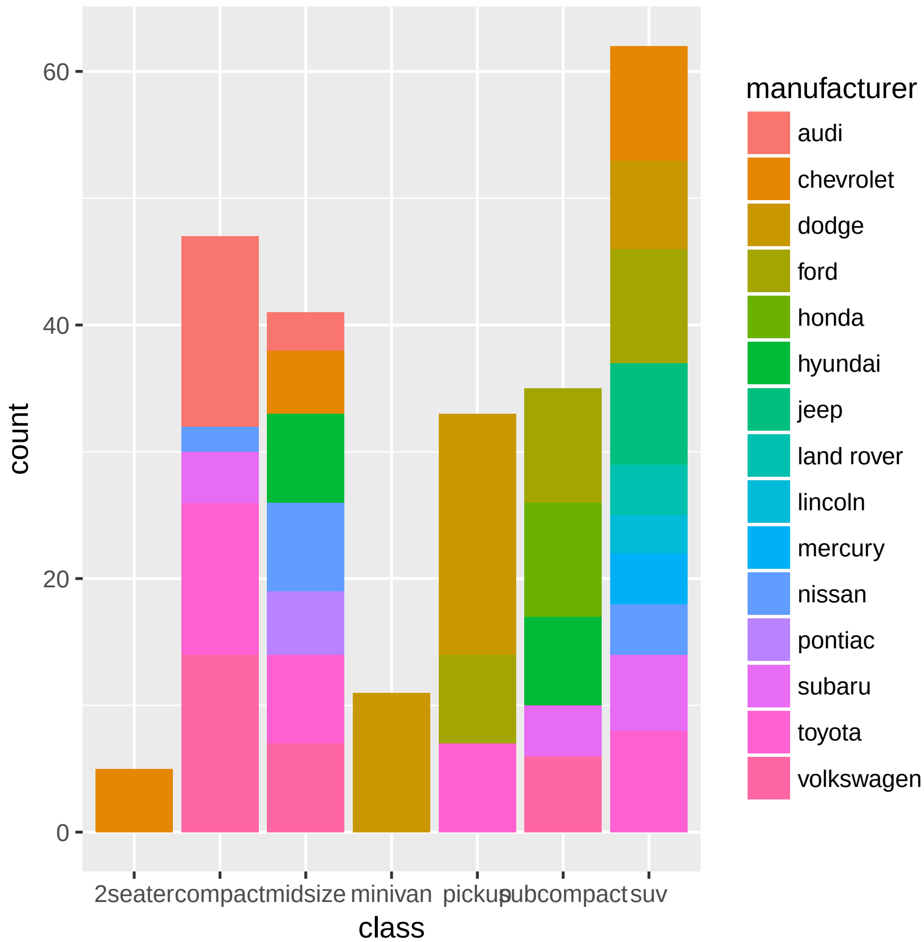
a. Represent graphically with a barplot the number of cars for each class.

```
pl <- ggplot(mpg, aes(class)) +
  geom_bar()
pl
```




b. Represent graphically with a barplot, the distribution of manufacturer for each class (map `manufacturer` variable to `fill`).

```
pl <- ggplot(mpg, aes(class, fill=manufacturer)) +
  geom_bar()
pl
```


60
40
20
0
count
2seater
compact
midsize
minivan
pickup
subcompact
suv
class
manufacturer
audi
chevrolet
dodge
ford
honda
hyundai
jeep
land rover
lincoln
mercury
nissan
pontiac
subaru
toyota
volkswagen

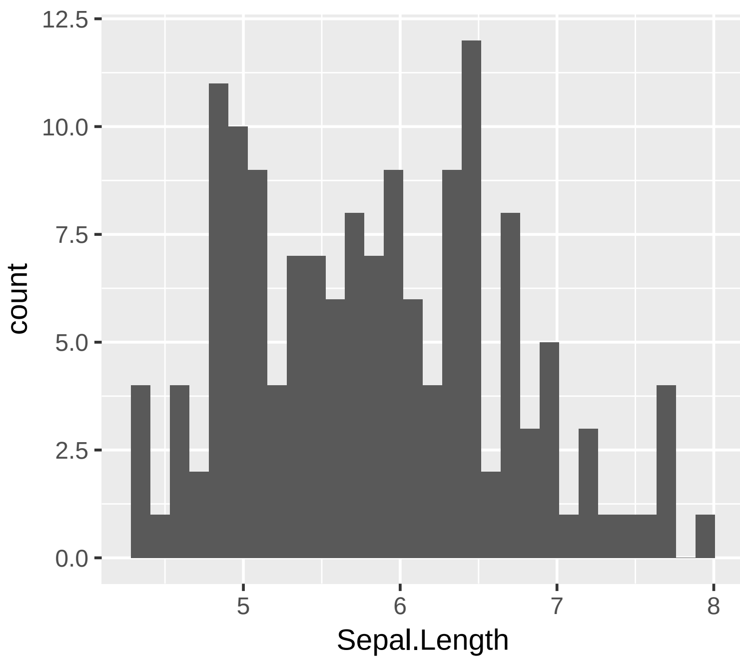
## 7.4 Histogram

### 7.4.1 Exercise 1

Let us consider `iris` dataset.

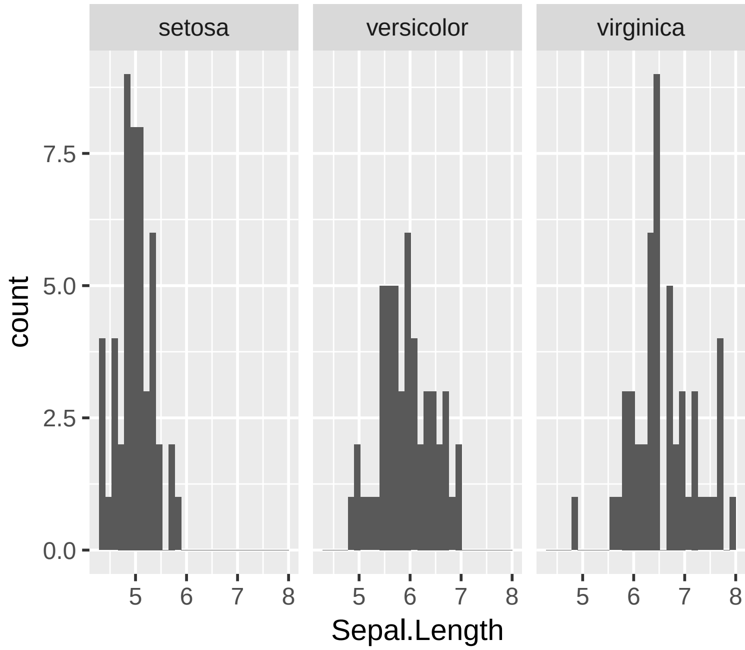
a. Represent the distribution of `Sepal_Length` variable with an histogram.

```
pl <- ggplot(data=iris, aes(x=Sepal.Length)) +
    geom_histogram()
pl
```




b. Represent each level of `Species` variable in a different panel. Use `facet_grid()` function.

```
pl <- ggplot(data=iris, aes(x=Sepal.Length)) +
    geom_histogram() +
  facet_grid(. ~ Species)
pl
```


setosa
versicolor
virginica
count
7.5
5.0
2.5
0.0
5
6
7
8
Sepal.Length

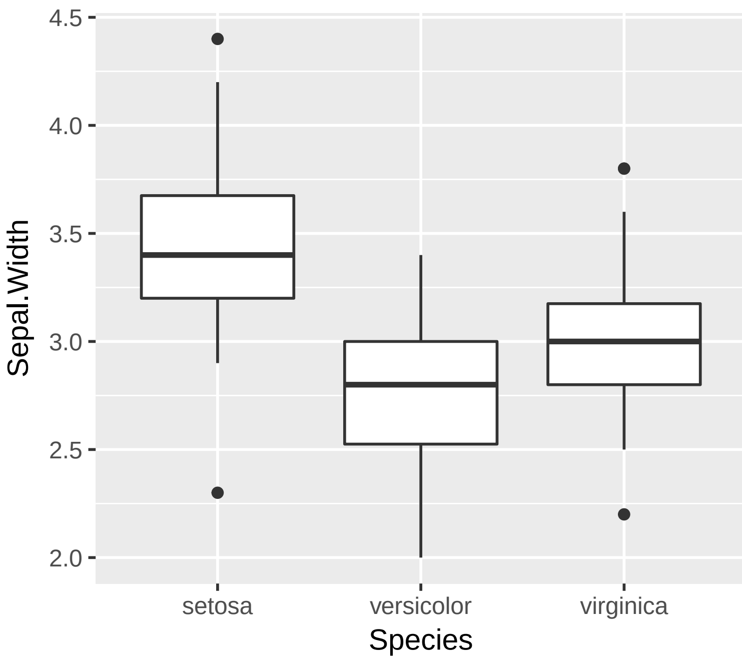
## 7.5 Boxplot

### 7.5.1 Exercise 1

Let us consider `iris` dataset.

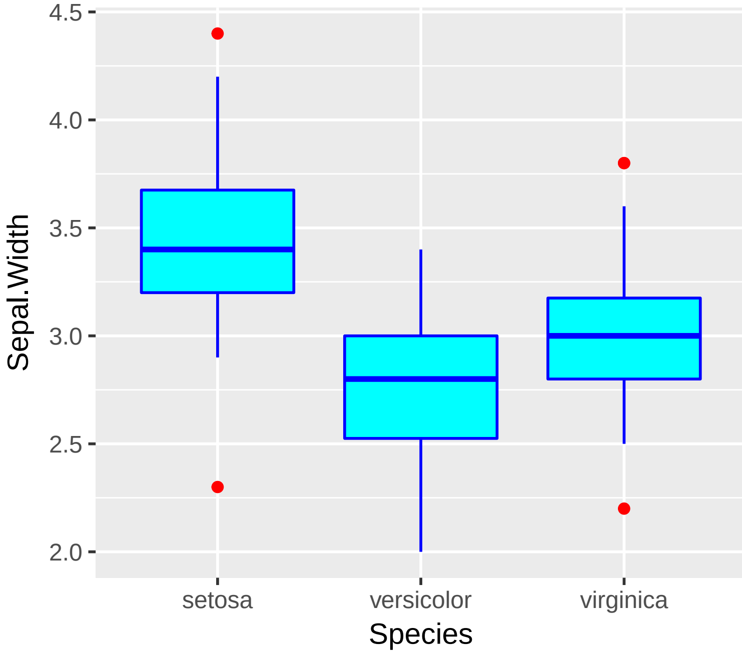
a. Build a boxplot to compare the differences of sepal width accordingly to the type of iris species.

```
pl <- ggplot(data=iris, aes(x=Species, y=Sepal.Width)) +
  geom_boxplot()

pl
```



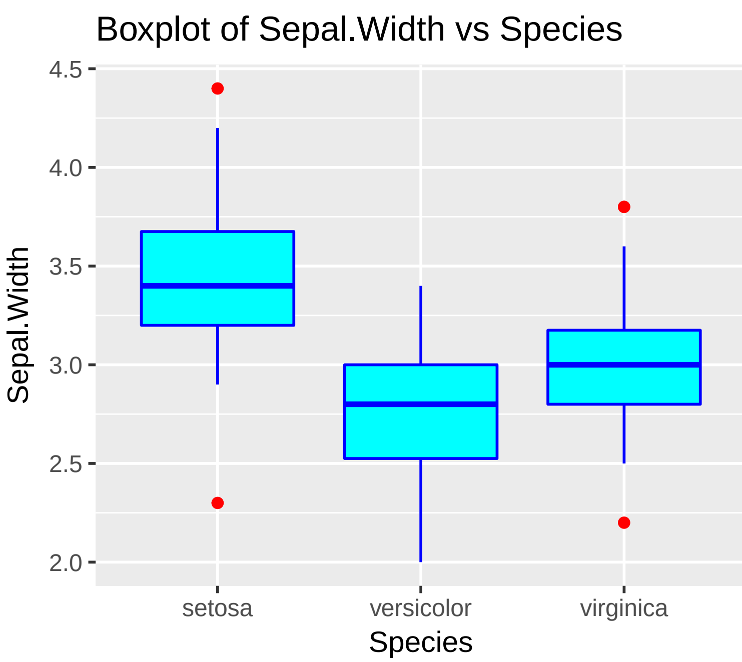

b. Set the fill colour of boxes as `"#00FFFF"`, the lines colour of boxes as `"#0000FF"` and the outliers colour as `"red"`.

```
pl <- ggplot(data=iris, aes(x=Species, y=Sepal.Width)) +
  geom_boxplot(fill="#00FFFF", colour="#0000FF", outlier.colour = "red")

pl
```




c. Add the plot title: `"Boxplot of Sepal.Width vs Species"`.

```
pl <- ggplot(data=iris, aes(x=Species, y=Sepal.Width)) +
  geom_boxplot(fill="#00FFFF", colour="#0000FF", outlier.colour = "red") +
  ggtitle("Boxplot of Sepal.Width vs Species")
pl
```



## 7.6 Lineplot

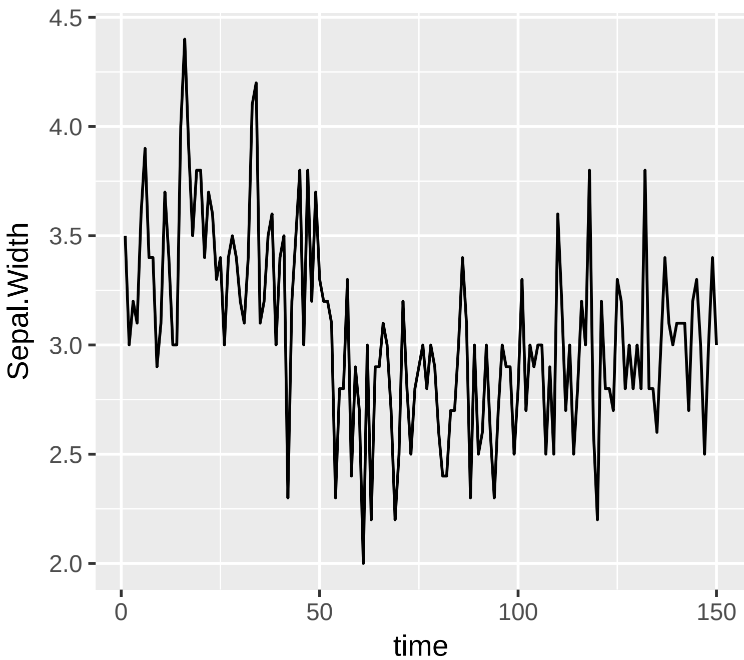
### 7.6.1 Exercise 1

Let us suppose that the observations on `iris` are taken along time.
So let us consider the following dataset, named `iris2`, in which `time` variable is added:

```
require(dplyr)
iris2 <- iris %>% mutate(time=1:150)
```

a. Build a lineplot to visualize the measures of `Sepal.Length` variable along time.

```
ggplot(data = iris2, mapping = aes(y=Sepal.Width, x= time)) +
  geom_line()
```



# Chapter 8

# Statistical models

Before starting the exercises, load the following libraries, supposing they are already installed.

```
require(dplyr)
require(ggplot2)
require(qdata)
```

## 8.1 Linear Models

### 8.1.1 Exercise 1

The number of impurities (lumps) present in the containers of paint depends on the rate of agitation applied to the container. A researcher wants to determine the relation between the rate of agitation and the number of lumps, so he conducts an experiment. He applies different rates of agitation (`Stirrate`) to 12 containers of paint and he counts the number of impurities (lumps) present in the containers of paint (`Impurity`).

```
data(paint)
head(paint)
```

```
## # A tibble: 6 × 2
##   Stirrate Impurity
##      <int>    <dbl>
## 1       20      8.4
## 2       38     16.5
## 3       36     16.4
## 4       40     18.9
## 5       42     18.5
## 6       26     10.4
```

a. Let us compute the main descriptive statistics of `Impurity`.

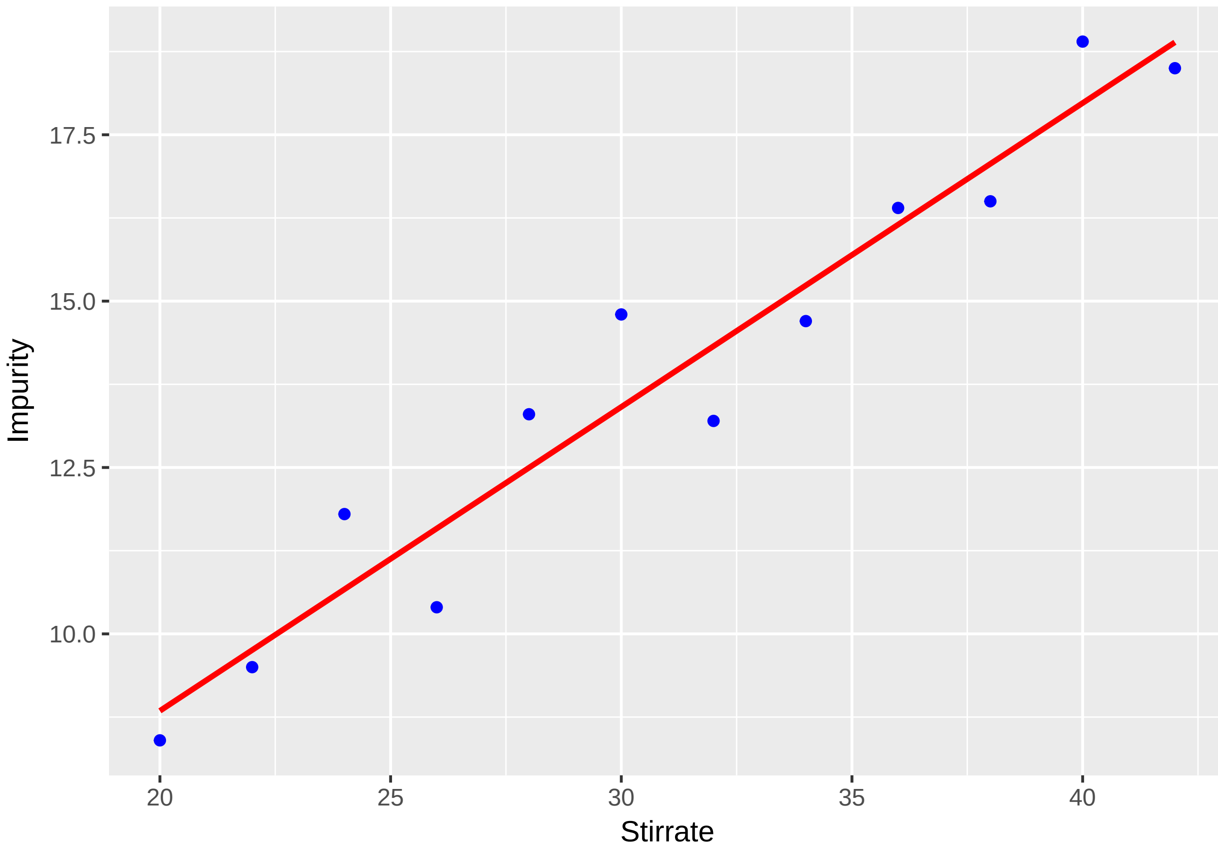
```
# Descriptive Statistics
summary_stat <- paint %>% summarise(n=n(),
  min = min(Impurity),
  first_qu = quantile(Impurity, 0.25),
  median = median(Impurity),
  mean = mean(Impurity),
  third_qu = quantile(Impurity, 0.75),
  max = max(Impurity),
```

```
  sd = sd(Impurity))

print(summary_stat)
```

```
## # A tibble: 1 × 8
##       n   min first_qu median     mean third_qu   max       sd
##   <int> <dbl>    <dbl>  <dbl>    <dbl>    <dbl> <dbl>    <dbl>
## 1    12   8.4    11.45     14 13.86667   16.425  18.9 3.407567
```

b. Let us graphically represent the relation between `Impurity` and `Stirrate` variables (add regression line to the scatterplot).

```
pl <- ggplot(data = paint, mapping = aes(x = Stirrate, y=Impurity)) +
  geom_point(color="blue") +
  geom_smooth(method = "lm", colour="red", se = FALSE)

print(pl)
```




```
# A simple straight linear regression is a good choice to describe the relation
```

c. Let us compute a simple linear regression between `Impurity` and `Stirrate`.

```
# Fit the linear model
fm <- lm(formula = Impurity ~ Stirrate, data = paint)
```

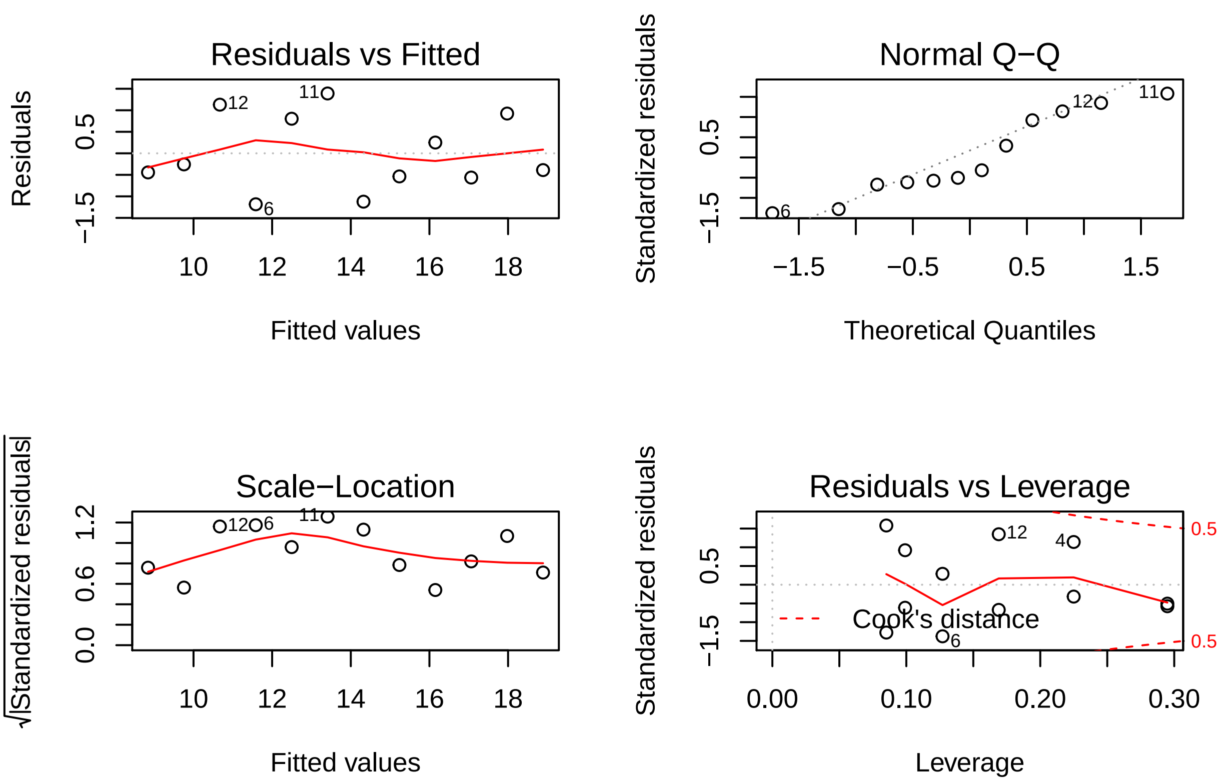
d. Does `Stirrate` influence `Impurity`? How? Let us analyze the model fitted by using `summary()` function.

```
summary(fm)
```

```
##
## Call:
## lm(formula = Impurity ~ Stirrate, data = paint)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1834 -0.5432 -0.3233  0.8333  1.3900
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.28928    1.22079  -0.237    0.817
## Stirrate     0.45664    0.03844  11.880 3.21e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9193 on 10 degrees of freedom
## Multiple R-squared:  0.9338, Adjusted R-squared:  0.9272
## F-statistic: 141.1 on 1 and 10 DF,  p-value: 3.211e-07
```

e. Let us check (final) models residuals.

```r
# Residuals analysis
op <- par(mfrow = c(2,2))
plot(fm)
```




```r
par(op)
```

### 8.1.2 Exercise 2

A pressure switch has a membrane whose thickness (in mm) influences the pressure required to trigger the switch itself. The aim is to determine the thickness of the membrane for which the switch "trig" with a pressure equal to 165 ± 15 KPa. 25 switches with different thickness (`DThickness`) of the membrane was analysed, measuring the pressure at which each switch opens (KPa) (`SetPoint`).

```
data(switcht)
head(switcht)
```

```
## # A tibble: 6 × 2
##   DThickness SetPoint
##        <dbl>    <dbl>
## 1        0.9  223.523
## 2        0.6  157.131
## 3        0.5  149.307
## 4        0.8  200.146
## 5        0.8  199.974
## 6        0.7  166.919
```

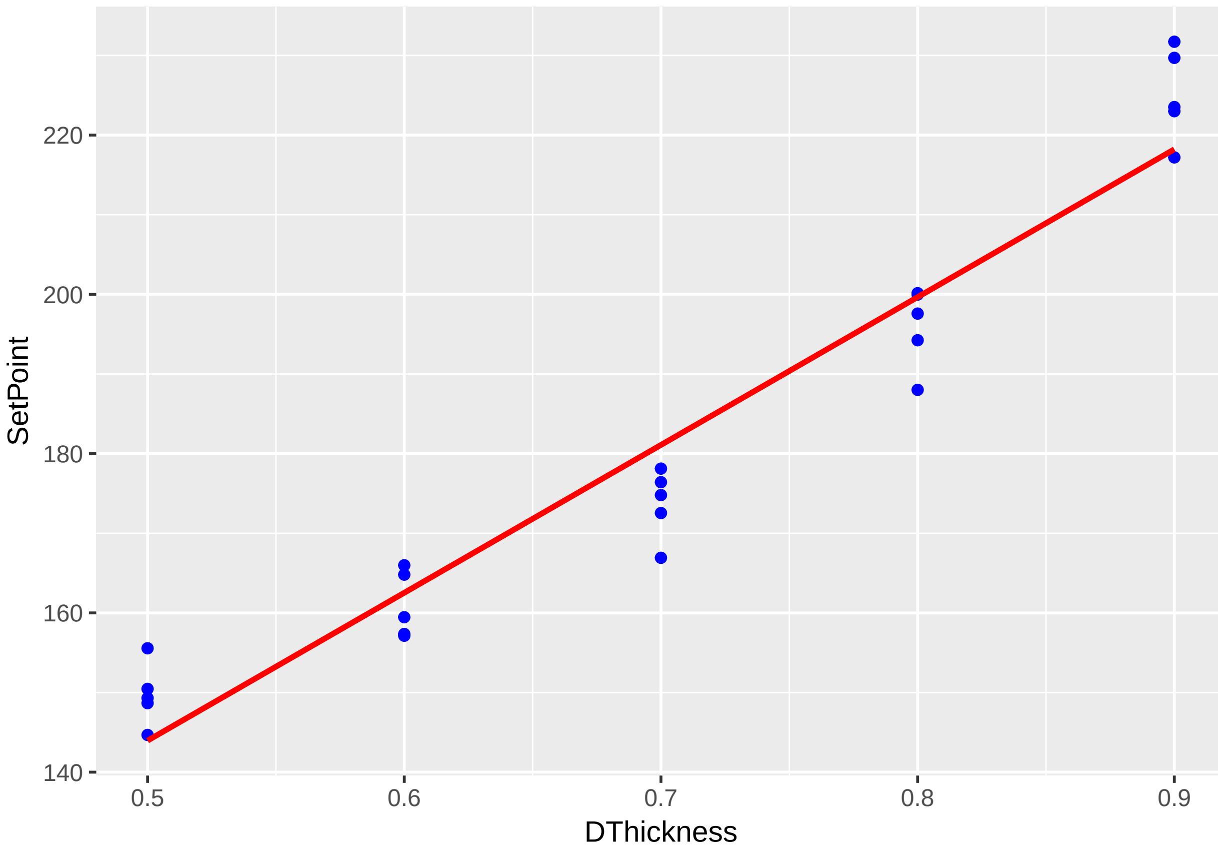
a. Let us compute the descriptive statistics of `SetPoint` variable.

```
# Descriptive Statistics
summary_stat <- switcht %>% summarise(n=n(),
  min = min(SetPoint),
  first_qu = quantile(SetPoint, 0.25),
  median = median(SetPoint),
  mean = mean(SetPoint),
  third_qu = quantile(SetPoint, 0.75),
  max = max(SetPoint),
  sd = sd(SetPoint))

print(summary_stat)
```

```
## # A tibble: 1 × 8
##       n     min first_qu  median     mean third_qu     max       sd
##   <int>   <dbl>    <dbl>   <dbl>    <dbl>    <dbl>   <dbl>    <dbl>
## 1    25 144.674  157.353 174.796 181.0909  199.974 231.725 27.70451
```

b. Let us graphically represent the relation between `DThickness` and `SetPoint`(add regression line to the graph).

```
pl <- ggplot(data = switcht, mapping = aes(x = DThickness, y=SetPoint)) +
  geom_point(color="blue") +
  geom_smooth(method = "lm", colour="red", se = FALSE)

print(pl)
```




c. Let us compute a linear regression between `DThickness` and `SetPoint` and check the residuals of the fitted model.

```
# Fit the linear model
fm1 <- lm(formula = SetPoint ~ DThickness, data = switcht)
```

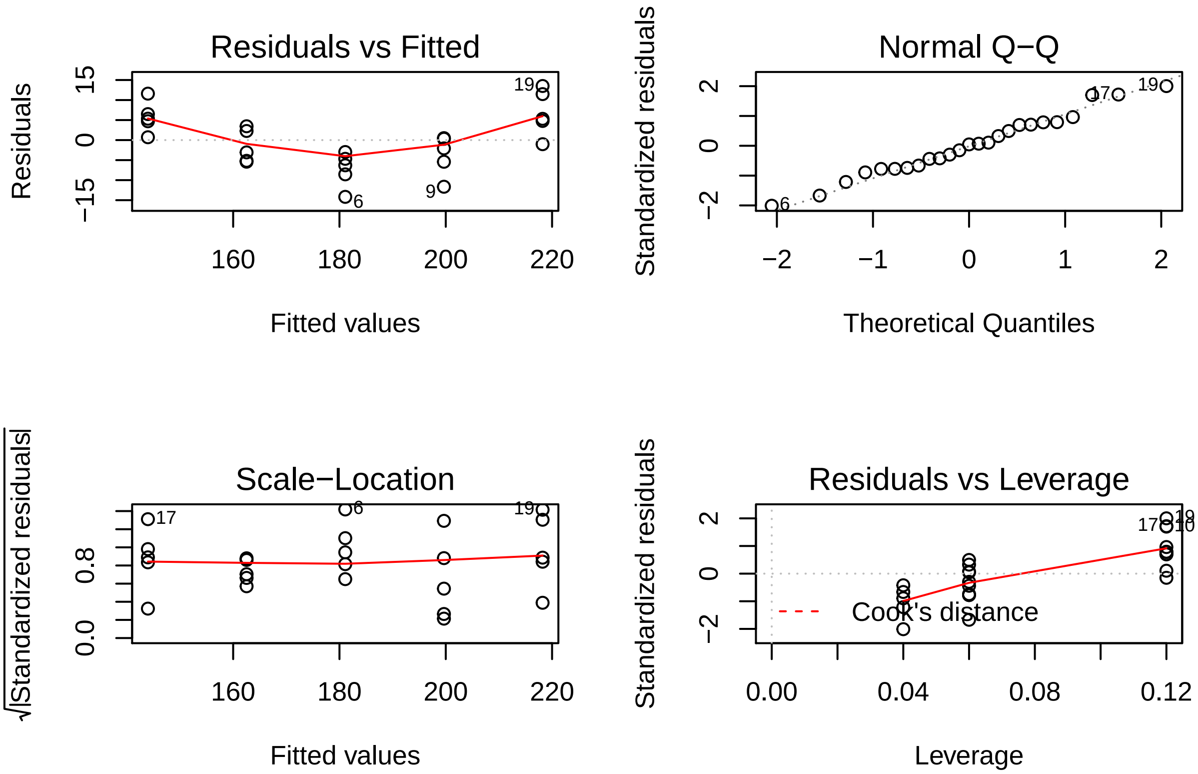
d. Does `DThickness` influences `SetPoint`? Let us analyze the model fitted by using `summary()` function.

```
summary(fm1)
```

```
##
## Call:
## lm(formula = SetPoint ~ DThickness, data = switcht)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.1719  -5.1742   0.3194   4.7807  13.5067
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   51.145      7.266   7.039 3.58e-07 ***
## DThickness   185.637     10.174  18.246 3.54e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.194 on 23 degrees of freedom
## Multiple R-squared:  0.9354, Adjusted R-squared:  0.9326
## F-statistic: 332.9 on 1 and 23 DF,  p-value: 3.542e-15
```

e. Let us check (final) models residuals.

```
# Residuals analysis
op <- par(mfrow = c(2,2))
plot(fm1)
```




```
par(op)
```

# Chapter 9

# Data Mining

Before starting the exercises, load the following libraries, supposing they are already installed.

```
require(qdata)
require(dplyr)
require(ggplot2)
require(nnet)
```

## 9.1 Neural Networks

### 9.1.1 Exercise 1

Consider `iris` dataset.

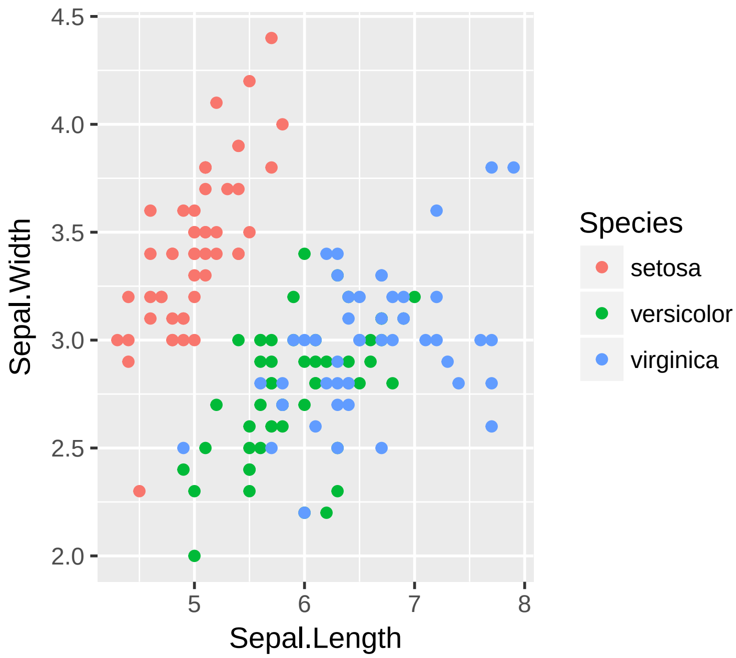
```
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

A botanist wants to to find a prediction model to assess the probability of belonging to a specific species, for each flower, based on its sepal and petal features.

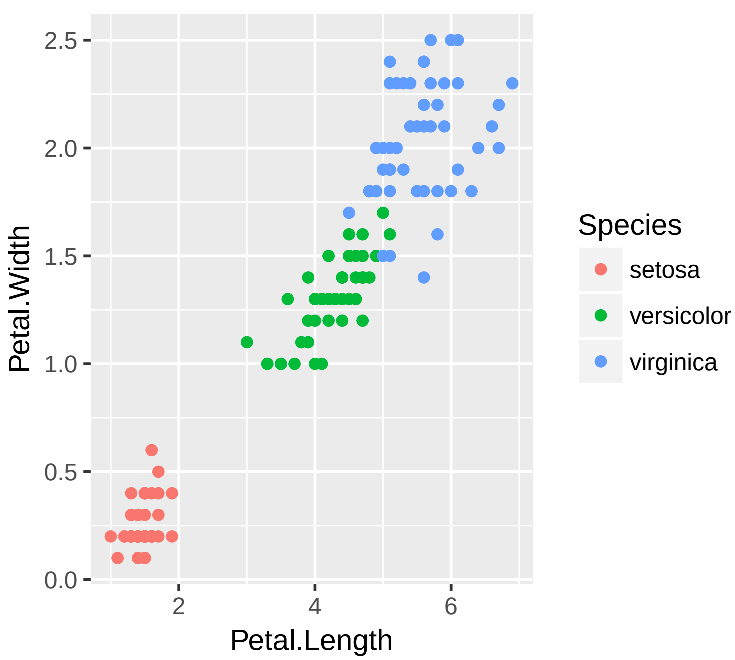
a. Analyze the relationship between `Species` and the other variables of `iris` dataset. The following lines of code produces a scatterplot of `Sepal.Length` and `Sepal.Width` by `Species`.

   ```
   ggplot(data=iris, mapping=aes (x=Sepal.Length, y=Sepal.Width, colour=Species)) +
     geom_point()
   ```




Generate a scatterplot to analyze the relationship between `Petal.Length` and `Petal.Width` by `Species`. Comment the results.

```
ggplot(data=iris, mapping=aes (x=Petal.Length, y=Petal.Width, colour=Species)) +
  geom_point()
```




b. Divide the dataset in train and test dataset in this way:

```
set.seed(1)
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
train <- iris[samp,]
test <- iris[-samp,]
```

and estimate a Neural Network model on train sample to assess the probability of belonging to a specific species, for each flower, based on its measures of `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`. Use `nnet()` function and set the `size` (number of units in the hidden layer) to 2.

```
nn_mod <- nnet(Species ~ ., data = train, size = 2)
```

```
## # weights:  19
## initial  value 88.417449
## iter  10 value 35.898332
## iter  20 value 25.562049
## iter  30 value 12.718791
## iter  40 value 4.169553
## iter  50 value 1.495034
## iter  60 value 0.042383
## iter  70 value 0.003180
## iter  80 value 0.002247
## iter  90 value 0.002168
## iter 100 value 0.002115
## final  value 0.002115
## stopped after 100 iterations
```

c. Use `predict()` function to gain the predictions on test sample. Add `type = "class"` argument to `predict()` function. Add the prediction estimated to `test` dataset.

```
test$pr <- predict(object = nn_mod, newdata = test, type = "class")
```

d. Built a frequency table to compare the original distribution of `Species` and that predicted in `test` data. Comment the results.

```
test %>%
  group_by(Species, pr) %>%
  summarise(n = n()) %>%
  mutate(freq = paste(round(n/sum(n)* 100, 2), "%"))
```

```
## Source: local data frame [5 x 4]
## Groups: Species [3]
##
##      Species         pr     n  freq
##       <fctr>      <chr> <int> <chr>
## 1     setosa     setosa    25 100 %
## 2 versicolor versicolor    23  92 %
## 3 versicolor  virginica     2   8 %
## 4  virginica versicolor     4  16 %
## 5  virginica  virginica    21  84 %
```