Introduction to Data Analytics Using R

Laura Vana lvana@wu.ac.at

Contents

1 Introduction **2**
1.1 Workshop . . . 2
1.2 R and RStudio . . . 3

2 The Basics **5**
2.1 Using R as a calculator . . . 6
2.2 Functions . . . 7
2.3 Assignments . . . 9

3 R Objects **10**
3.1 Data types . . . 10
3.2 Data structures . . . 12

4 Importing and saving data **19**
4.1 Working directory . . . 19
4.2 Importing data . . . 20
4.3 Saving data . . . 20

5 Data manipulation **21**
5.1 Sub-setting . . . 21
5.2 Duplicates . . . 22
5.3 Missing values . . . 23
5.4 Adding a new variable to a data frame . . . 23
5.5 Converting character variables to factors . . . 24

6 Data visualization **25**
6.1 Plotting functions . . . 25
6.2 Multi-panel Plots . . . 39
6.3 Exporting Plots . . . 40

7 Linear model in R **41**
7.1 Revisiting the `advertising` data set . . . 41
7.2 Sample descriptives . . . 42
7.3 Visualizing the multivariate relationships . . . 43
7.4 Simple linear regression . . . 47
7.5 Multiple linear regression . . . 51
7.6 Further topics . . . 54

8 The logistic model in R **59**
8.1 The `credit` data set . . . 60
8.2 Sample descriptives . . . 60
8.3 Simple logistic model . . . 65
8.4 Multiple logistic model . . . 67

9 Further details for printing, formatting and exporting regression results 70
9.1 Matrix of coefficients 70
9.2 Formatting coefficients 71

10 Appendix: R Markdown and R Notebooks 72

1 Introduction

1.1 Workshop

1.1.1 Who am I?

Laura Vana, PhD

- Postdoctoral researcher at the Institute for Statistics and Mathematics
- MSc in Quantitative Finance (WU), PhD in Statistics (WU)
- Lecturer in Statistics, Data Science, QRM, Finance (WU Wien, FH WKW, FH Campus)
- Organizer of the R-Ladies Vienna Meetup, Assistant organizer of the Vienna<-R Meetup group.

1.1.2 Who are you?

- Which is your favorite software for data analysis?
- What is your experience in programming?
- What are your expectations from this workshop?

1.1.3 Workshop goals and approach

After this workshop you will be able to

- navigate RStudio, interact with R and its extension packages;
- understand the different data structures in R;
- import and explore data sets in R;
- generate graphics for data visualization;
- run statistical models and interpret their results using built-in functions in R.

1.1.4 Workshop outline

Day 1

Day 1 is concerned with becoming familiar with getting data into R, doing some simple descriptive statistics, data manipulation and visualization.

9:30-16:30

- Presentation: Intro
- Interactive Session: The basics in R and RStudio
- – break 10:45-11:00 –
- Interactive Session: Introduction to R objects
- – lunch 12:30-13:30 –
- Interactive Session: Data manipulation and visualization.
- – break 15:15-15:30 –
- Work on your own data set

- – end –

Day 2

First half of Day 2 takes a look at performing linear and logistic regression using R. In the second half of Day 2 one gets to utilize all of the skills learned throughout the workshop by creating a complete statistical analysis using their own data set (descriptives, graphs, statistical modeling).

9:00-15:30

- Interactive Session: the linear model in R.
- – break 10:45-11:00 –
- Interactive Session: the logistic regression model in R.
- – lunch 12:30-13:30 –
- Work on your own data set
- – end –

1.2 R and RStudio

1.2.1 What is R?

- R is a language and environment for statistical computing and graphics
- R can be considered as a dialect the S language and environment.
- It is an object oriented programming language.
- R was first developed by Robert Gentleman and Ross Ihaka (University of Auckland, NZ) during the 1990s.
- R provides a wide variety of **statistical** (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and **graphical** techniques.
- Updated versions are available every 3-4 months
- The R Project for Statistical Computing webpage contains all necessary information.

1.2.2 Why use R?

- **Open source software** (General Public Licence - GPL);
- It is powerful, flexible and robust – it is being continuously developed by researchers and practitioners worldwide;
- State of the art graphics;
- It contains advanced and specialized statistical routines (mainly through the its extension packages);
- It is (almost) platform independent;
- It does not depend on a point-and-click interface...
- It is highly extensible (through its package ecosystem) and it allows users to add additional functionality by defining new functions.
- Can produce well-designed publication-quality plots.
- A lot of documentation is available.

1.2.3 Why NOT R?

- It is not a point-and-click interface :-)
- Command-based
- It can have a rather steep learning curve.
- Data sets must be read into memory, hence it can have some trouble with massive data sets (GBs).
- There are multiple ways of doing the same thing.

1.2.4 RStudio

- Integrated Development Environment (IDE) for R.
- Initially released in 2010, version 1.0 was released on 1 November 2016.
- Open source
- Makes the experience of working with R more user friendly by offering editor, compiler, debugger etc in the same place.
- Offers tool for version control (e.g., svn, git).
- Download at: http://www.rstudio.com/

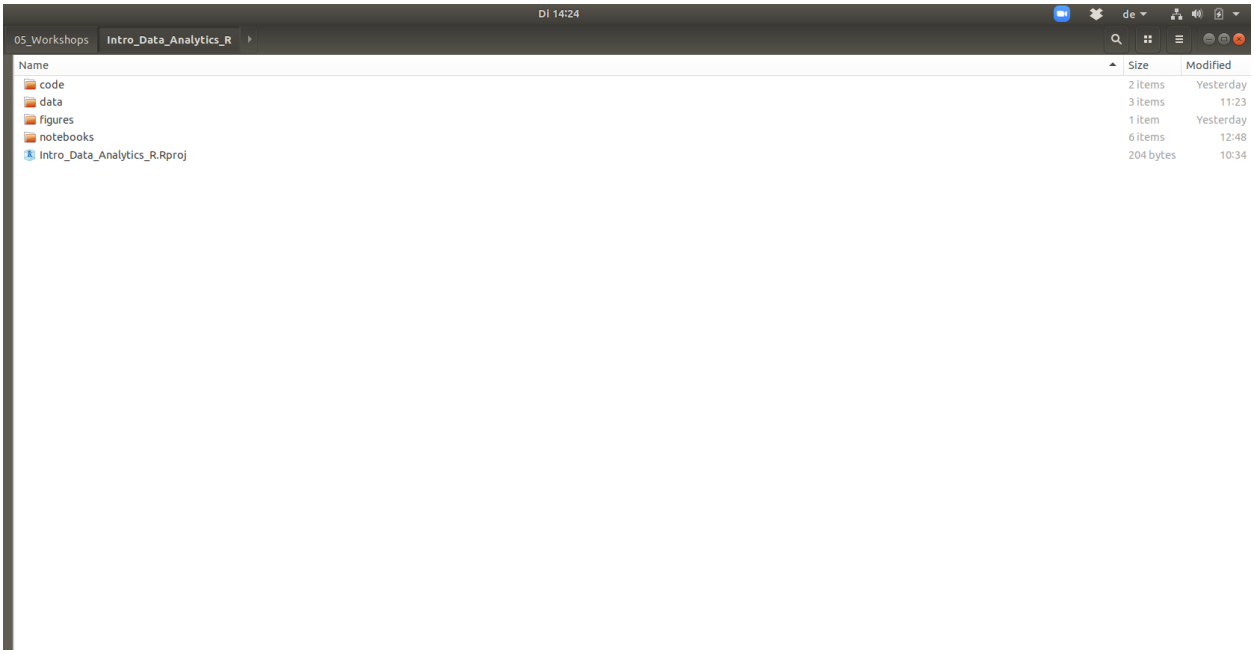
1.2.5 RStudio Interface

- The RStudio main window consists of five parts: a menu and four windows ('Panes')
- Using the drop-down menu, RStudio and R can be controlled and set-up.
- Pane 1 (top left) - Files and Data: Editing R-Code and display of data records
- Pane 2 (top right) - Workspace (shows all objects in the Workspace) and History (displays the complete code that has been written to the Console)
- Panel 3 (bottom right) - Files, Plots, Packages, Help
- Panel 4 (bottom left) - Console: Running R-Code
- The layout of the panels (both arrangement and content) can be changed in the options menu.

1.2.6 Using RStudio Projects

- RStudio Projects make it straightforward to divide your work into multiple contexts, each with their own working directory, workspace, history, and source documents.
- Projects will likely contain several files, including data, R scripts or even paper manuscripts.
- By keeping all resources in one place and "defining" this as an RStudio project will allow RStudio to find the files more easily as well as to switch easily among projects.
- Everyone has a personal style in defining the structure of folders but a recommended one is: `Projects -> Intro\_Data\_Analytics_R`
- In Intro_Data_Analytics_R you can also have more subfolders such as `data` , `code`, `notebooks`.
- *For WU laptops with Windows: Ensure that your project folder is not on the Desktop and not on the shared documents file! A good location would be on the C: drive: C:/Users/mmuster/*

Make sure that everytime you want to work with the materials here you open the .Rproj file. Here is a screenshot of my folder structure:



1.2.7 Resources

- All materials used in this workshop can be found at http://statmath.wu.ac.at/~vana/Intro_Data_Analytics_R
- Style and code conventions: https://google.github.io/styleguide/Rguide.html
- *R for Data Science* by Hadley Wickham and Garrett Grolemund: https://r4ds.had.co.nz/index.html
- *Hands-On Programming with R* by Garrett Grolemund: https://rstudio-education.github.io/hopr/

2 The Basics

We will start by interacting with R through the R console (in Pane 4).

When you type a command at the prompt (Symbol >) and hit Enter, your computer executes the command and shows you the results.

```
1 + 1
```

```
## [1] 2
```

R is printing for each line in the output the index of the first observation on the line in your result.

Some commands return more than one value:

```
1:100
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
```

```
##  [91]  91  92  93  94  95  96  97  98  99 100
```

If you type a command that R does not recognize, it will throw an error:

```
4 % 3
```

```
## Error: <text>:1:3: unexpected input
## 1: 4 % 3
##       ^
```

2.1 Using R as a calculator

```
3 + 3 ## Addition
```

```
## [1] 6
```

```
3 - 3  ## Substraction
```

```
## [1] 0
```

```
3 * 3  ## Multiplication
```

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

```
3 / 3 ## Divison
```

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

```
3 ^ 5 ## Powers
```

```
## [1] 243
```

```
243 ^ (1 / 5)
```

```
## [1] 3
```

Note that the R will not run anything that follows a hashtag on a line. This makes hashtags useful for adding comments and annotations to your code.

2.1.1 Relational Operators

Comparisons can also be performed in R:

```
3 < 5
```

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

```
3 > 5
```

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

```
3 <= 3
```

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

```
3 >= 5
```

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

```
3 == 5 ## Attention! double equal sign
```

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

```r
3 != 5 ## Not equal
```

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

2.1.2 Logical Operators

In addition one can also carry out operations like `AND`, `OR`

```r
(3 == 5) | (5 == 5) ## OR
```

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

```r
(3 != 5) & (4 > 3)  ## AND
```

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

2.2 Functions

R comes with many functions that you can use to do simple or sophisticated tasks. The data that we pass into the function is called the function's *argument*.

```r
round(1.234)
```

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

What if I want to round to two decimal points?

```r
args(round)
```

```
## function (x, digits = 0)
## NULL
```

I can change the argument `digits`. This argument has a default value of 0.

```r
round(1.234, digits = 2)
```

```
## [1] 1.23
```

Let us have a look at a more complicated function `sample()`, which can be used to randomly sample integer numbers:

```r
sample()
```

```
## Error in sample(): argument "x" is missing, with no default
```

```r
sample(10)
```

```
##  [1]  4  9  7  8  2  3 10  6  5  1
```

Clearly, we sampled the numbers from 1 to 10. We can dig deeper:

```r
args(sample)
```

```
## function (x, size, replace = FALSE, prob = NULL)
## NULL
```

If we want more information on what these arguments actually mean we can ask R for help:

```r
help("sample")
?sample
??"sampling" ## this allows us to also search for more 'vague' terms
```

which will open in RStudio a help page in the help section of Pane 3.

For example, we could sample from the numbers between 1 to 10 only 5 values with replacement:

```
sample(10, size = 5, replace = TRUE)
```

```
## [1] 8 8 6 6 2
```

Of course, user defined functions can also be created in R by using the syntax:

```
foo <- function() {}
```

which will build a function out of whatever R code you place between the braces.

For example if you want `foo` to return 1, you can type:

```
foo <- function() {1}
foo()
```

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

Now we could create a new function `foo2` which has `x` as an argument and will return a `TRUE` if `x > 10` and `FALSE` otherwise.

```
foo2 <- function(x) {
  x > 10
}
```

```
foo2(11)
```

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

```
foo2(9)
```

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

Exercise 1

In the console, perform the following operations:

1. Choose any number and add 2 to it.
2. Multiply the result by 3.
3. Subtract 6 from the answer.
4. Divide what you get by 3.
5. Compute the logarithm in base 2 of the number you get in Step 4.
6. Write a function `foobar` which simply returns the number you chose in Step 1.

Solution 1

```
log((((2 + 2) * 3) - 6)/3, base = 2)
```

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

```
foobar <- function() {2}
foobar()
```

```
## [1] 2
```

2.3 Assignments

So far what we typed into the console has not been saved into our computer's memory.

Objects can be created in R using assignments (`<-`) which allows them to be "saved" and reused later.

The object is saved and appears in Pane 2 in the Environment tab.

```
x <- 3
```

The object `x` is saved and appears in Pane 2 in Environment. But nothing is printed in the Console. For printing `x` you must type it in the console:

```
x
```

```
## [1] 3
```

Note that R is case-sensitive i.e., capital `X` is not the same as `x`:

```
X
```

```
## Error in eval(expr, envir, enclos): object 'X' not found
```

We can then perform operations on these objects:

```
exp(x)
```

```
## [1] 20.08554
```

or

```
y <- 4
x ^ 2 + y ^ 2
```

```
## [1] 25
```

```
z <- 5
z ^ 2 == x ^ 2 + y ^ 2
```

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

2.3.1 R Scripts

At this point it would be useful to have a better overview of the code and also to have a draft of what we have been doing so far. We can create a draft of our code by using an **R script**. You can open an R script in RStudio by going to `File > New File > R script` in the menu bar.

An R script is just a plain text file that you save R code in. It is recommended to start a script with some basic information for you to refer back to later. Start with a comment line (the line begins with a #) that tells you the name of the script, something about the script, who created it, and the date it was created.

Then, you can automatically execute a line of code in a script by clicking the Run button or by the shortcut `Ctrl + Enter`.

Exercise 2

Perform the following steps (time - 5 minutes)

1. Create a new R script
2. Copy the commands we typed before regarding `x`, `y` and `z` in the source editor: `x <- 3`; `y <- 4`; `z <- 5`; `z ^ 2 == x ^ 2 + y ^ 2`;

3. Insert comments at the beginning of the file with the necessary details (e.g., brief description, author, date)
4. Run the codes again.
5. Save the R script in the folder designated for our workshop under the name `my_first_r_file.R`.
6. Close R and re-open your script to see that it was saved.

---

3 R Objects

3.1 Data types

In R the following 6 data types are available:

- double (i.e., double precision number – R lingo for real number);
- integer
- character (sometimes referred to as string)
- logical/boolean (can only take values `TRUE` or `FALSE`)
- complex
- raw

We can find the type of each object by the function `typeof()`. Moreover, the functions `is.<type>()` (i.e., `is.double()`, `is.integer()`, `is.numeric()` for double or integer, `is.character()`, `is.logical`) will return a `TRUE` if the object is of type `<type>`.

3.1.1 Double

R will save any number that you type in R as a double:

```
a <- 1.02
typeof(x)
```

```
## [1] "double"
```

```
is.double(x)
```

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

```
is.numeric(x)
```

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

```
a <- 1
typeof(x)
```

```
## [1] "double"
```

3.1.2 Integers

Integers are numbers that can be written without a decimal component. This data type is more important for developers as it save memory and can be specified by typing a number followed by an uppercase `L`:

```
int <- 1L
typeof(int)
```

```
## [1] "integer"
```

```
is.integer(int)
```

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

```
is.numeric(int)
```

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

but

```
is.double(int)
```

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

3.1.3 Character

In R character objects can by specified as a character or string of characters surrounded by quotes (`""` or `''`):

```
w <- "Hello"
typeof(w)
```

```
## [1] "character"
```

```
is.character(w)
```

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

3.1.4 Logical

```
bool <- (4 > 5)
typeof(bool)
```

```
## [1] "logical"
```

```
is.logical(bool)
```

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

Exercise 3

What are the data types of the following objects?

- `x1 <- 10`
- `x2 <- "10"`
- `x3 <- (10 == 10)`
- `x4 <- "ten"`

- `x5 <- 10L`

Solution 3

`double`, `character`, `logical`, `character`, `integer`

3.1.5 Converting to other data types

One can covert the data types into other data types by using the functions `as.<type>()`: `as.double()`, `as.integer()`, `as.numeric()`, `as.character()`, `as.logical()`. For example:

```
bool_num <- as.numeric(bool)
bool_num
```

```
## [1] 0
```

```
as.numeric("10")
```

```
## [1] 10
```

```
as.character(10)
```

```
## [1] "10"
```

```
as.integer("1")
```

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

Some conversions might be unexpected:

```
as.integer(2.100) ## returns the closest smaller integer
```

```
## [1] 2
```

```
as.logical(2.100) ## returns TRUE for any number different than zero
```

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

```
as.logical(0) ## return FALSE for a zero
```

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

However, running the following command will return `NA` (not available):

```
as.numeric("Hello")
```

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

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

3.2 Data structures

The most important data structures in R are

- `vector`,
- `matrix`,
- `array`,
- `list`,

- `data.frame`.

3.2.1 (Atomic) vectors

An atomic vector is just a simple vector of data. You can make an atomic vector by grouping some values of data together with the combine function `c()`. Let's make a vector containing the Pythagorean triple:

```
pythag <- c(x, y, z)
pythag
```

```
## [1] 3 4 5
```

```
typeof(pythag)
```

```
## [1] "double"
```

Each atomic vector stores its values as a one-dimensional vector. The object `pythag` now is an R object, which has several attributes/characteristics such as `length`:

```
length(pythag)
```

```
## [1] 3
```

One could also assign names to the vector elements:

```
names(pythag) <- c("c1", "c2", "h")
pythag
```

```
## c1 c2  h
##  3  4  5
```

IMPORTANT: Each atomic vector can only store one type of data!

Exercise 4

What are the `typeof()` these vectors? What do you observe?

1. Create a vector `some_numbers` containing your age, your favorite number, the last digit of your phone number.
2. Create a vector `some_characters` containing your first name, your pet's name and the name of your favorite city.
3. Create a vector `some_thing` containing your first name, your age and whether or not you like ice-cream (`TRUE` vs `FALSE`).

Solution 4

```
##. 1
some_numbers <- c(30, 5, 8)
typeof(some_numbers)
```

```
## [1] "double"
```

```
##. 2
some_characters <- c("Laura", "Tup", "Vienna")
typeof(some_characters)
```

```
## [1] "character"
```

```r
##. 3
some_thing <- c("Laura", 30, TRUE)
typeof(some_thing)
```

```
## [1] "character"
```

3.2.1.1 Coercion If you try to put multiple types of data into a vector, R will convert the elements to a single type of data, a practice named *coercion*.

How does R coerce data types? `logical -> integer -> double -> character`

- If a character string is present in an atomic vector, R will convert everything else in the vector to character strings.
- If a vector only contains logicals and numbers, R will convert the logicals to numbers; every TRUE becomes a 1, and every FALSE becomes a 0

```r
aa <- c(TRUE, 1L , FALSE)
aa
```

```
## [1] 1 1 0
```

```r
bb <- c(TRUE, 1L , FALSE, 2.02)
bb
```

```
## [1] 1.00 1.00 0.00 2.02
```

```r
cc <- c(TRUE, 1L , FALSE, 2.02, "Hello")
cc
```

```
## [1] "TRUE"  "1"     "FALSE" "2.02"  "Hello"
```

3.2.1.2 Vectorization R is a vectorized programming language, i.e., operation are performed element by element:

```r
bb ^ 2
```

```
## [1] 1.0000 1.0000 0.0000 4.0804
```

```r
cc ^ 2 ## error as we can't exponentiate characters
```

```
## Error in cc^2: non-numeric argument to binary operator
```

```r
aa == 1
```

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

If we want to know at which index are the values that meet the condition, you can use the function `which()`:

```r
which(aa == 1)
```

```
## [1] 1 2
```

3.2.1.3 Selecting elements If you want to extract or replace elements of a vector, you can use the `[` operator:

```r
aa[3] # 3. Element of aa
```

```
## [1] 0
```

```r
aa[1:2] # 1. and 2. Element of aa
```

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

```r
aa[-1] # all Elements of aa except the first one
```

```
## [1] 1 0
```

```r
aa[3] <- 1000 # replace 3. Element by 1000
aa
```

```
## [1]    1    1 1000
```

3.2.2 Matrices

Matrices store values in a two-dimensional array, just like a matrix from linear algebra:

```r
m <- matrix(1:12, nrow = 3)
m
```

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

The function `matrix` will fill up the matrix column by column by default. Similar to atomic vectors, all the elements of the matrix must be of same type.

Some important functions for handling matrices:

```r
m[2, ] # Extract 2. row
```

```
## [1]  2  5  8 11
```

```r
m[, 4] # Extract 4. column
```

```
## [1] 10 11 12
```

```r
m[1, 4] # Extract 4. Element in the 1st row
```

```
## [1] 10
```

```r
## Assign column names and row names
colnames(m) <- c("A", "B", "C", "D")
rownames(m) <- c("X", "Y", "Z")
m
```

```
##   A B C  D
## X 1 4 7 10
## Y 2 5 8 11
## Z 3 6 9 12
```

```r
dim(m) ## dimensions: number of rows and cols
```

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

```r
nrow(m) ## number of rows
```

```
## [1] 3
```

```r
ncol(m) ## number of columns
```

```
## [1] 4
```

```
cbind(c(1, 1, 1), m) ## bind a column to m
```

```
##     A B C  D
## X 1 1 4 7 10
## Y 1 2 5 8 11
## Z 1 3 6 9 12
```

```
rbind(c(1, 1, 1, 1), m) ## bind a row to m
```

```
##   A B C  D
##   1 1 1  1
## X 1 4 7 10
## Y 2 5 8 11
## Z 3 6 9 12
```

An important function for working with matrices is `apply()` which applies a function to each row (if `MARGIN = 1`) or each column (if `MARGIN = 1`).

```
apply(m, 2, mean) ## renders mean for each column
```

```
##  A  B  C  D
##  2  5  8 11
```

3.2.3 Lists

Lists are the most general data structures in Rand they group together R objects:

```
list1 <- list(100:102,
              "R", list(TRUE, FALSE))

list1
```

```
## [[1]]
## [1] 100 101 102
##
## [[2]]
## [1] "R"
##
## [[3]]
## [[3]][[1]]
## [1] TRUE
##
## [[3]][[2]]
## [1] FALSE
```

The double-bracketed indexes tell you which element of the list is being displayed.

The `lapply()` function is useful for performing operations on list objects and returns a list object of same length of original set.

```
lapply(list1, length)
```

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

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

The `sapply()` is very similar to `lapply()`. If the result of the functions applied on each element are of same length it will return a vector or a matrix (pay attention to coercion!). Otherwise it returns the list. For example,

```
sapply(list1, length) ## returns a vector
```

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

```
sapply(list1, summary) ## returns a list
```

```
## [[1]]
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   100.0   100.5   101.0   101.0   101.5   102.0
##
## [[2]]
##    Length     Class      Mode
##         1 character character
##
## [[3]]
##      Length Class  Mode
## [1,] 1      -none- logical
## [2,] 1      -none- logical
```

3.2.4 Data frames

Data frames are the two-dimensional version of a list. They are the most useful storage structure for data analysis in R. The difference between a matrix and a data frame is that in a data frame the columns can be of different type.

```
df <- data.frame(Group = c("A", "B", "A", "A", "B"),
                 Income = c(1000, 1100, 1200, 1300, 1400),
                 Age = c(20, 19, 54, 45, 24))
df
```

```
##   Group Income Age
## 1     A   1000  20
## 2     B   1100  19
## 3     A   1200  54
## 4     A   1300  45
## 5     B   1400  24
```

Make sure each vector is of same length!

To see the types of objects in each column you can use the function `str()`

```
str(df)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ Group : chr  "A" "B" "A" "A" ...
##  $ Income: num  1000 1100 1200 1300 1400
##  $ Age   : num  20 19 54 45 24
```

Note that `Group` is a *factor* variable. A `factor` is R's way of storing categorical information. A factor can also be created by applying the `factor()` function to an vector.

```r
gender <- c("M", "F", "M", "M", "M", "F")
typeof(gender)
```

```
## [1] "character"
```

```r
gender <- factor(c("M", "F", "M", "M", "M", "F"))
gender
```

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

```r
levels(gender)
```

```
## [1] "F" "M"
```

If we want to select a variable out of the data frame we typically use the `$` operator:

```r
df$Group
```

```
## [1] "A" "B" "A" "A" "B"
```

We can also perform operations on these variables such as taking the average income:

```r
mean(df$Income)
```

```
## [1] 1200
```

An important function for working with data frames is `apply()` which applies a function to each row (if `MARGIN = 1`) or each column (if `MARGIN = 1`).

```r
apply(df, 2, mean)
```

```
## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA

## Warning in mean.default(newX[, i], ...): argument is not numeric or logical:
## returning NA
```

```
##  Group Income    Age
##     NA     NA     NA
```

```r
## OR
apply(df, 2, function(x) length(x))
```

```
##  Group Income    Age
##      5      5      5
```

This returns an error as we cannot compute the mean of a categorical variable. We can select the second and third column of the data frame and then use the function `apply()`:

```r
apply(df[,2:3], 2, mean)
```

```
## Income    Age
## 1200.0   32.4
```

Note that `apply()` internally coerces the data frame to a matrix, which means the types are also coerced. Therefore, a better option would be using the function `sapply()` which applies the specified function to each column. For example: we can use the `sapply()` to identify the numeric variables and then use the resulting boolean vector for selecting the columns for which the mean should be computed:

```
id_num <- sapply(df, is.numeric)
id_num
```

```
##  Group Income    Age
##  FALSE   TRUE   TRUE
```

```
df[, id_num]
```

```
##   Income Age
## 1   1000  20
## 2   1100  19
## 3   1200  54
## 4   1300  45
## 5   1400  24
```

```
sapply(df[, id_num], mean)
```

```
## Income    Age
## 1200.0   32.4
```

The function `tapply()` is also very useful when working with data frame. It computes a measure (mean, median, min, max, etc..) or a function for each factor variable in a vector.

```
tapply(df$Income, df$Group, mean)
```

```
##        A        B
## 1166.667 1250.000
```

```
## more fancy :-)
sapply(df[, id_num], function(x) tapply(x, df$Group, mean))
```

```
##     Income      Age
## A 1166.667 39.66667
## B 1250.000 21.50000
```

For computing summary statistics for each column of the data frame the function `summary()` can be used:

```
summary(df)
```

```
##     Group               Income          Age
##  Length:5           Min.   :1000   Min.   :19.0
##  Class :character   1st Qu.:1100   1st Qu.:20.0
##  Mode  :character   Median :1200   Median :24.0
##                     Mean   :1200   Mean   :32.4
##                     3rd Qu.:1300   3rd Qu.:45.0
##                     Max.   :1400   Max.   :54.0
```

Typically data frames are not typed in by hand but rather imported/loaded into R.

4 Importing and saving data

4.1 Working directory

Each time you open R, it links itself to a directory on your computer, which R calls the working directory. This will be the folder containing the .Rproj file.

4.2 Importing data

R can open many types of files, including spreadsheets, text files, binary files and files from other statistical packages and software.

Data can be imported either from a local file or directly from a webpage.

Exercise 5

1. Go to the webpage and download to your computer `advertising.csv`. Save it in the `Intro_Data_Analytics_R/data/` folder.

The most useful commands for reading in data in R are `read.table()` and `read.csv()`. We need to supply the local path of the file to the e.g., `read.csv` function. Given that we are working with projects, we need to only supply the relative path i.e., path relative to the working directory:

```
advertising <- read.csv("data/advertising.csv")
```

The `head()` function can be used to print the first 6 observations of the data set:

```
head(advertising)
```

```
##   X.1 X    TV Radio Newspaper Sales NewspaperCat Country
## 1   1 1 230.1  37.8      69.2  22.1     (50,150]       A
## 2   2 2  44.5  39.3      45.1  10.4      (25,50]       B
## 3   3 3  17.2  45.9      69.3   9.3     (50,150]       B
## 4   4 4 151.5  41.3      58.5  18.5     (50,150]       B
## 5   5 5 180.8  10.8      58.4  12.9     (50,150]       A
## 6   6 6   8.7  48.9      75.0   7.2     (50,150]       A
```

```
str(advertising)
```

```
## 'data.frame':    100 obs. of  8 variables:
##  $ X.1         : int  1 2 3 4 5 6 7 7 8 9 ...
##  $ X           : int  1 2 3 4 5 6 7 7 8 9 ...
##  $ TV          : num  230.1 44.5 17.2 151.5 180.8 ...
##  $ Radio       : num  37.8 39.3 45.9 41.3 10.8 48.9 32.8 32.8 19.6 2.1 ...
##  $ Newspaper   : num  69.2 45.1 69.3 58.5 58.4 75 23.5 23.5 11.6 1 ...
##  $ Sales       : num  22.1 10.4 9.3 18.5 12.9 7.2 11.8 11.8 13.2 4.8 ...
##  $ NewspaperCat: chr  "(50,150]" "(25,50]" "(50,150]" "(50,150]" ...
##  $ Country     : chr  "A" "B" "B" "B" ...
```

The `Import Dataset` button in Pane 2 could also be used for the same purpose, but I do not recommend it. This is because the button generates a command in the R console with an absolute path.

4.3 Saving data

Data frames can be saved to a `.csv` by the function `write.csv()`

```
write.csv(advertising, file = "data/advertising_copy.csv", row.names = FALSE)
```

5 Data manipulation

Data manipulation includes sub-setting, eliminating duplicated or missing values and adding new variables to the existing data frame.

5.1 Sub-setting

As previously mentioned the operators `[`, `[[` or `$` can we used for sub-setting.

```
advertising[1:3, ]
```

```
##   X.1 X    TV Radio Newspaper Sales NewspaperCat Country
## 1   1 1 230.1  37.8      69.2  22.1     (50,150]       A
## 2   2 2  44.5  39.3      45.1  10.4      (25,50]       B
## 3   3 3  17.2  45.9      69.3   9.3     (50,150]       B
```

```
advertising$Sales[1:3, ] ## wrong - advertising$Sales is a one-dimensional object
```

```
## Error in advertising$Sales[1:3, ]: incorrect number of dimensions
```

```
advertising$Sales[1:3]
```

```
## [1] 22.1 10.4  9.3
```

We can also use the function `subset()` which returns the subsets of a vector, matrix or data frame which meet certain conditions.

```
subset(advertising, (NewspaperCat == "(50,150]"))
```

```
##    X.1  X    TV Radio Newspaper Sales NewspaperCat Country
## 1    1  1 230.1  37.8      69.2  22.1     (50,150]       A
## 3    3  3  17.2  45.9      69.3   9.3     (50,150]       B
## 4    4  4 151.5  41.3      58.5  18.5     (50,150]       B
## 5    5  5 180.8  10.8      58.4  12.9     (50,150]       A
## 6    6  6   8.7  48.9      75.0   7.2     (50,150]       A
## 14  13 13  23.8  35.1      65.9   9.2     (50,150]       B
## 15  13 13  23.8  35.1      65.9   9.2     (50,150]       B
## 19  16 16 195.4  47.7      52.9  22.4     (50,150]       B
## 20  17 17  67.8  36.6     114.0  12.5     (50,150]       B
## 21  18 18 281.4  39.6      55.8  24.4     (50,150]       B
## 24  21 21 218.4  27.7      53.4  18.0     (50,150]       B
## 57  54 54 182.6  46.2      58.7  21.2     (50,150]       A
## 59  56 56 198.9  49.4      60.0  23.7     (50,150]       B
## 65  62 62 261.3  42.7      54.7  24.2     (50,150]       B
## 79  76 76    NA  43.7      89.4   8.7     (50,150]       A
## 89  86 86 193.2  18.4      65.7  15.2     (50,150]       B
## 91  88 88 110.7  40.6      63.2  16.0     (50,150]       B
## 92  89 89  88.3    NA      73.4  12.9     (50,150]       B
## 93  90 90 109.8  47.8      51.4  16.7     (50,150]       B
## 96  93 93 217.7  33.5      59.0  19.4     (50,150]       A
## 97  94 94 250.9  36.5      72.3  22.2     (50,150]       A
## 99  96 96 163.3  31.6      52.9  16.9     (50,150]       A
```

Alternatively we could write this as:

```
advertising[(advertising$NewspaperCat == "(50,150]"), ]
```

```
##    X.1  X    TV Radio Newspaper Sales NewspaperCat Country
## 1    1  1 230.1  37.8      69.2  22.1     (50,150]       A
## 3    3  3  17.2  45.9      69.3   9.3     (50,150]       B
## 4    4  4 151.5  41.3      58.5  18.5     (50,150]       B
## 5    5  5 180.8  10.8      58.4  12.9     (50,150]       A
## 6    6  6   8.7  48.9      75.0   7.2     (50,150]       A
## 14  13 13  23.8  35.1      65.9   9.2     (50,150]       B
## 15  13 13  23.8  35.1      65.9   9.2     (50,150]       B
## 19  16 16 195.4  47.7      52.9  22.4     (50,150]       B
## 20  17 17  67.8  36.6     114.0  12.5     (50,150]       B
## 21  18 18 281.4  39.6      55.8  24.4     (50,150]       B
## 24  21 21 218.4  27.7      53.4  18.0     (50,150]       B
## 57  54 54 182.6  46.2      58.7  21.2     (50,150]       A
## 59  56 56 198.9  49.4      60.0  23.7     (50,150]       B
## 65  62 62 261.3  42.7      54.7  24.2     (50,150]       B
## 79  76 76    NA  43.7      89.4   8.7     (50,150]       A
## 89  86 86 193.2  18.4      65.7  15.2     (50,150]       B
## 91  88 88 110.7  40.6      63.2  16.0     (50,150]       B
## 92  89 89  88.3    NA      73.4  12.9     (50,150]       B
## 93  90 90 109.8  47.8      51.4  16.7     (50,150]       B
## 96  93 93 217.7  33.5      59.0  19.4     (50,150]       A
## 97  94 94 250.9  36.5      72.3  22.2     (50,150]       A
## 99  96 96 163.3  31.6      52.9  16.9     (50,150]       A
```

Let us make a subset of the data which contains NewspaperCat categories "(50,150]" and "(0,25]":

```
advertising_subset <- subset(advertising, NewspaperCat %in% c("(0,25]", "(50,150]"))
```

The dimensions of this data set are:

```
dim(advertising_subset)
```

```
## [1] 65  8
```

5.2 Duplicates

An application of sub-setting is when we only want to keep unique observations in our data set. The `duplicated()` function finds duplicate values and returns a logical vector that tells you whether the specified value is a duplicate of a previous value:

```
duplicated(advertising)
```

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

We can have a look at how many observations are duplicated and how many not:

```
id_dup <- duplicated(advertising) # this is a vector of T/F
table(id_dup)
```

```
## id_dup
## FALSE  TRUE
##    97     3
```

The following command gives us the indexes of the 3 duplicated observations:

```
which(id_dup)
```

```
## [1]  8 15 18
```

```
# these are the duplicated observations
advertising[which(id_dup), ]
```

```
##    X.1  X    TV Radio Newspaper Sales NewspaperCat Country
## 8    7  7  57.5  32.8      23.5  11.8       (0,25]       A
## 15  13 13  23.8  35.1      65.9   9.2     (50,150]       B
## 18  15 15 204.1  32.9      46.0  19.0      (25,50]       B
```

```
# we might want to eliminate them and assign the resulting data frame to another object
advertising_nodup <- subset(advertising, duplicated(advertising) == FALSE)
```

5.3 Missing values

In R missing values are marked as `NA`. They can by identified using the function `complete.cases()`, which returns a boolean which finds whether the rows are complete of not:

```
complete.cases(advertising_nodup)
```

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

The rows containing missing observations can easily be removed by:

```
advertising_nodup <- na.omit(advertising_nodup)
```

5.4 Adding a new variable to a data frame

New variables can easily be added to a data frame. For example, if we want to create a new variable in the `advertising_nodup` data set which contains the sum of the `TV`, `Radio` and `Newspaper` advertising budget we have:

```
advertising_nodup$Budget <- with(advertising_nodup, TV + Radio +
  Newspaper)
```

```
head(advertising_nodup)
```

```
##   X.1 X    TV Radio Newspaper Sales NewspaperCat Country Budget
## 1   1 1 230.1  37.8      69.2  22.1     (50,150]       A  337.1
## 2   2 2  44.5  39.3      45.1  10.4      (25,50]       B  128.9
## 3   3 3  17.2  45.9      69.3   9.3     (50,150]       B  132.4
```

```
## 4   4 4 151.5  41.3      58.5  18.5     (50,150]       B  251.3
## 5   5 5 180.8  10.8      58.4  12.9     (50,150]       A  250.0
## 6   6 6   8.7  48.9      75.0   7.2     (50,150]       A  132.6
```

5.5 Converting character variables to factors

Factors are R lingo for categorical variables. We can transform the existing variables to a factor by:

```
advertising_nodup$NewspaperCat <- as.factor(advertising_nodup$NewspaperCat )
advertising_nodup$Country <- as.factor(advertising_nodup$Country)
```

Exercise 6

1. Eliminate the first two columns of the advertising data frame.
2. What type of variables does the data frame contain?
3. Compute the mean of the variable `TV`.
4. Compute the standard deviation of the variable `Sales`.
5. Print the summary statistics for this data frame.
6. Create a new variable called `SalesBin` in the data frame that gets the value 1 if the sales are above 14 and zero otherwise.

Solution 6

```
## 1.
advertising <- advertising[, -c(1,2)]
## 2.
str(advertising)
```

```
## 'data.frame':    100 obs. of  6 variables:
##  $ TV          : num  230.1 44.5 17.2 151.5 180.8 ...
##  $ Radio       : num  37.8 39.3 45.9 41.3 10.8 48.9 32.8 32.8 19.6 2.1 ...
##  $ Newspaper   : num  69.2 45.1 69.3 58.5 58.4 75 23.5 23.5 11.6 1 ...
##  $ Sales       : num  22.1 10.4 9.3 18.5 12.9 7.2 11.8 11.8 13.2 4.8 ...
##  $ NewspaperCat: chr  "(50,150]" "(25,50]" "(50,150]" "(50,150]" ...
##  $ Country     : chr  "A" "B" "B" "B" ...
```

```
## 3.
mean(advertising$TV)
```

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

```
## 4.
sd(advertising$Sales)
```

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

```
## 5.
summary(advertising)
```

```
##        TV             Radio          Newspaper          Sales
##  Min.   :  5.4   Min.   : 1.40   Min.   :  0.30   Min.   : 4.80
##  1st Qu.: 69.9   1st Qu.:13.30   1st Qu.: 16.45   1st Qu.:10.60
##  Median :139.3   Median :26.70   Median : 31.40   Median :13.20
##  Mean   :145.2   Mean   :24.76   Mean   : 33.00   Mean   :14.36
```

```
##  3rd Qu.:215.6   3rd Qu.:35.80   3rd Qu.: 46.00   3rd Qu.:18.15
##  Max.   :293.6   Max.   :49.60   Max.   :114.00   Max.   :25.40
##  NA's   :1       NA's   :1                        NA's   :1
##  NewspaperCat         Country
##  Length:100         Length:100
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
##
```

```
## 6.
advertising$SalesBin <- as.numeric(advertising$Sales > 14)
```

---

6 Data visualization

In this part of the workshop, you will get familiar with the basics of using R for making plots and figures. More specifically, you will learn how to create the following graphics using R's base package `graphics`:

- scatterplots
- line plots
- bar plots
- box-and-whisker plots
- histograms
- spineplots.

Moreover, we will discuss the basics of how to change plot features or how to save the plots to an external file.

There are other R packages for plotting which can produce very nice graphs. The most used and noteworthy ones are `ggplot2` (Wickham and Chang 2016) and `lattice` (Sarkar 2017).

6.1 Plotting functions

In R's base graphics systems there are:

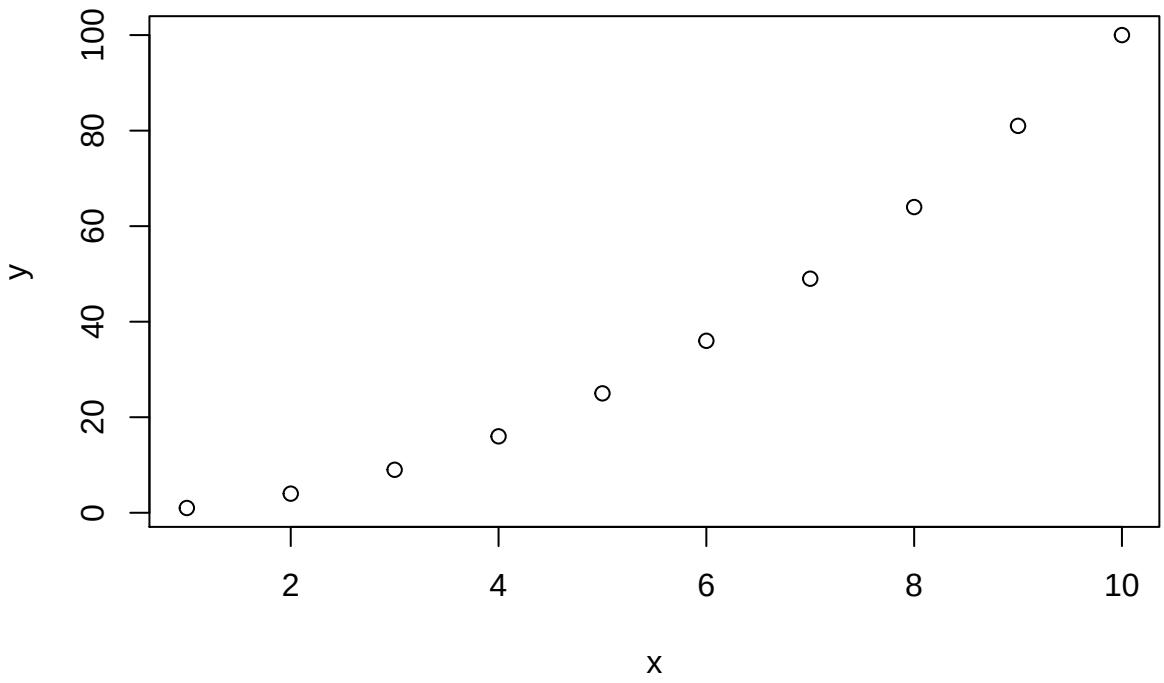
- High level plotting functions: used to make a new graph. When this high level plotting function is executed, the current plot (if any available) is written over. Examples: `plot()`, `barplot()`.
- Low-level plotting functions: used to modify existing plots and can only be executed if a graph has already been produced by a high-level plotting function. Examples: `points()`, `lines()`.

The **graphics device** in R is the area in which a plot is actually displayed (in RStudio the "Plots" tab in Pane 3). One can also create a new device for plotting (such as a pdf file). R will plot on the active device, which is the most recently created device. There can only be one active device at a time.

6.1.1 Scatterplots

Let us create a two variables, `x` which contains the integers from 1 to 10 and `y` which is the square of `x`. The two variables can be plotted in a scatterplot using the high-level `plot()` function:

```r
x <- 1:10
y <- x ^ 2
plot(x, y)
```



The `plot()` function has many arguments or *graphical parameters* which can be found by

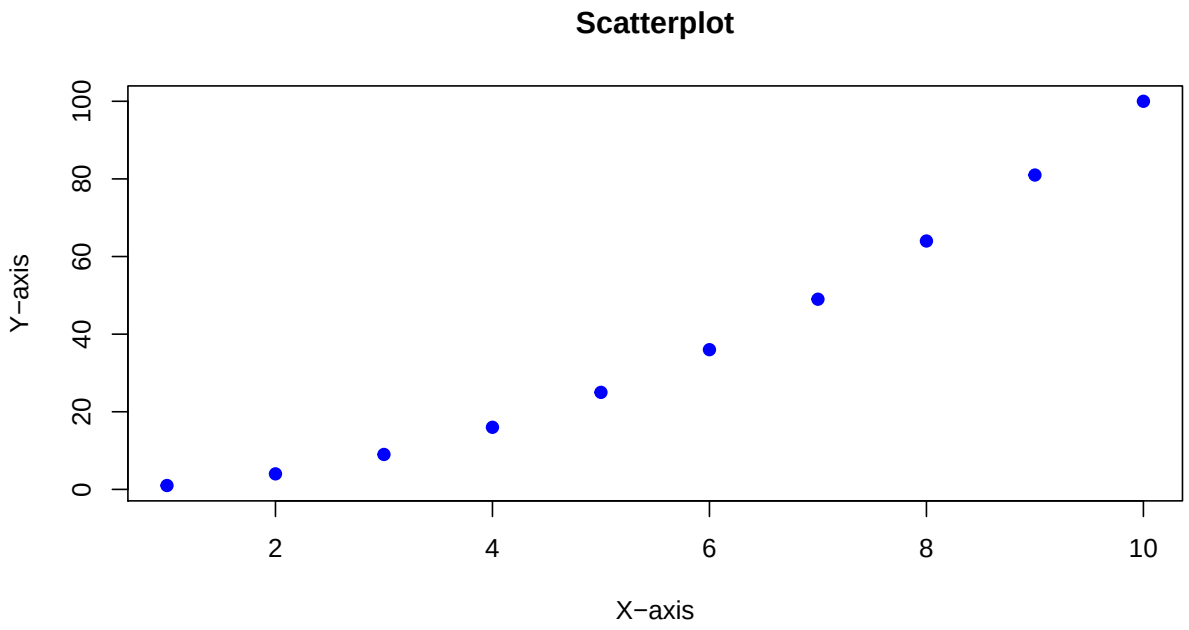
```r
?plot
## OR
?par
```

The graphical parameters of the graphics device can be changed by using the `par()` function. Once you change the settings in `par()`, they will remain that way until you start a new device.

A blog post on this topic that I always go back to is https://www.r-bloggers.com/r-plot-function-the-options/.

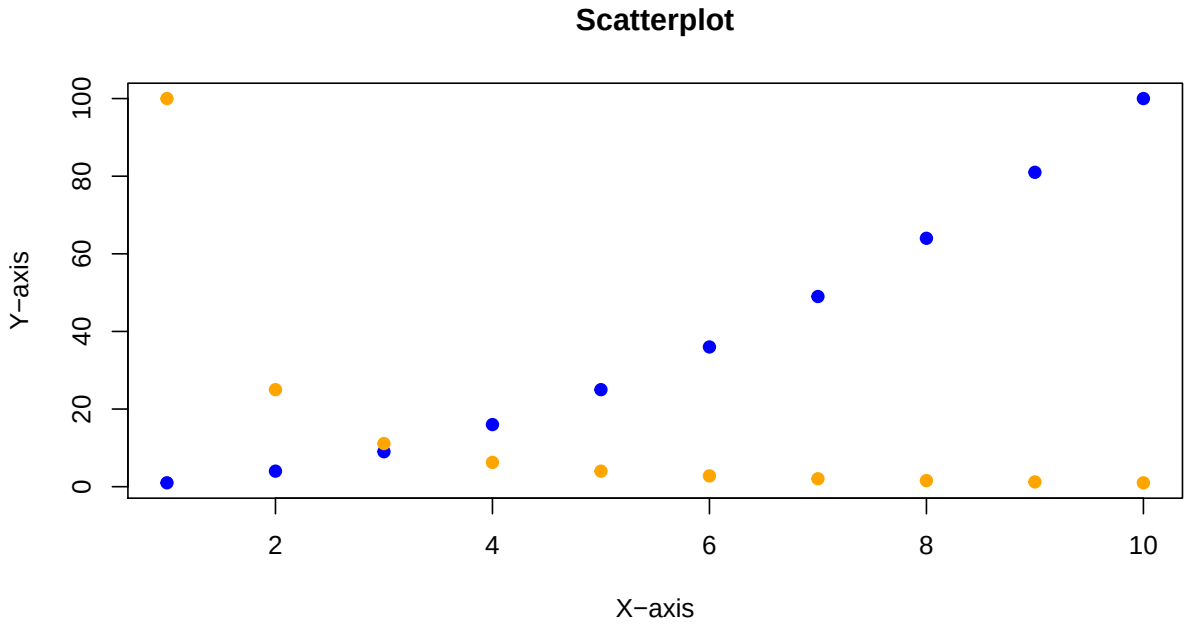
For example, a title, the axes labels and the type and color of points can be changed by:

```r
plot(x, y,
     main = "Scatterplot", ## title
     ylab = "Y-axis",
     xlab = "X-axis",
     pch = 19, ## type of point
     col = "blue")
```



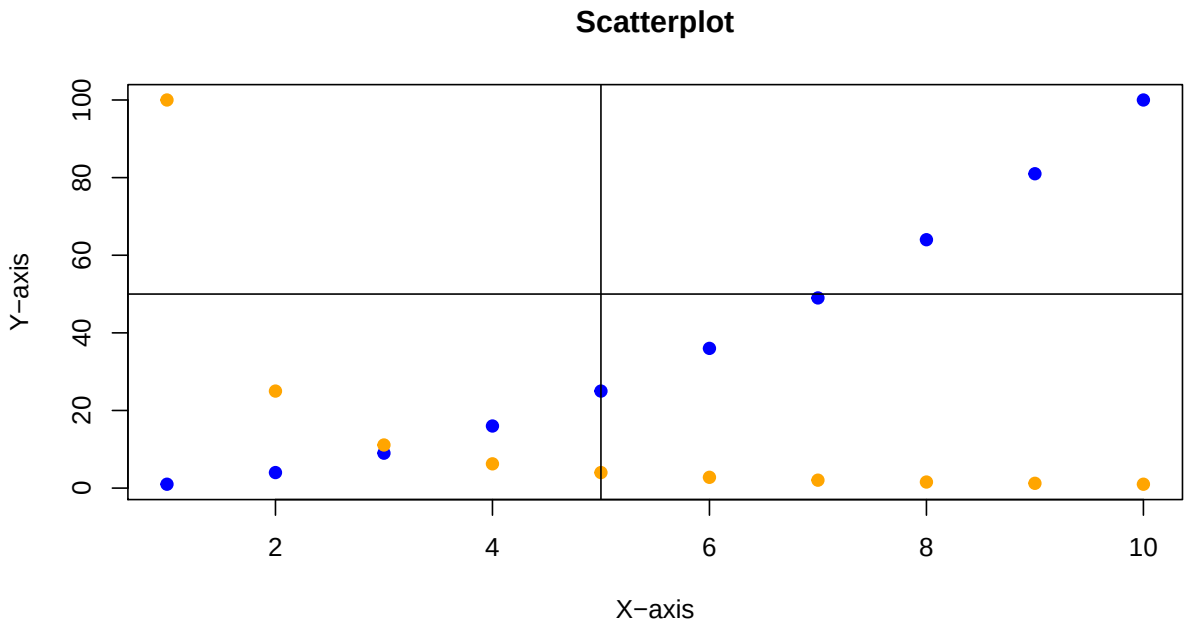
Now let us modify the initial scatterplot by adding more points to it, which would correspond to the function $f(x) = 100 \times x^{-2}$.

```
z <- 100 * x ^ (-2)
## call low level function
points(x, z, col = "orange", pch = 19)
```



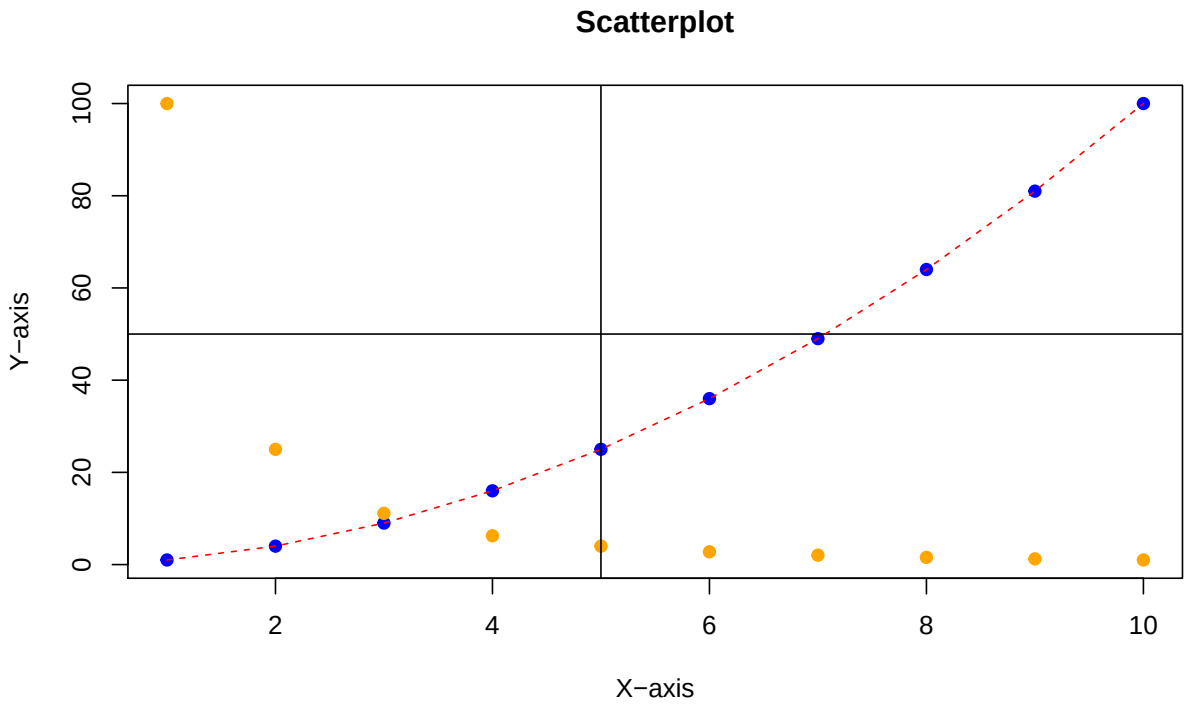
Add a horizontal line to the plot at $y = 50$ and a vertical one at $x = 5$.

```
abline(h = 50)
abline(v = 5)
```



Draw a dashed red line through the blue points at

```
lines(x, y, lty = 2, col = "red")
```

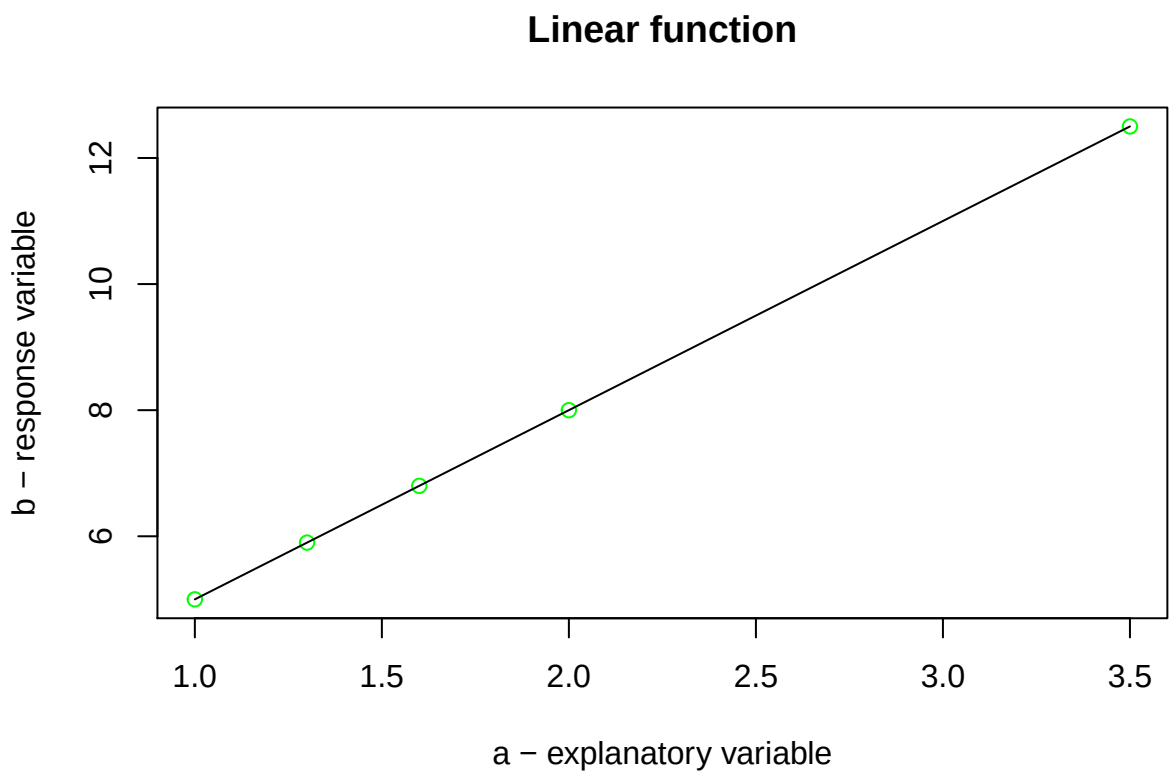


Exercise 7

1. Create the variables
$$a = \{1, 1.3, 1.6, 2, 3.5\}$$
$$b = 2 + 3a$$
2. Create a scatterplot of a and b with the name "Linear function", axes labels "a - explanatory variable" and "b - response variable". Color the points in green.
3. Add a line through the points using the `lines()` low level plotting function.

Solution 7

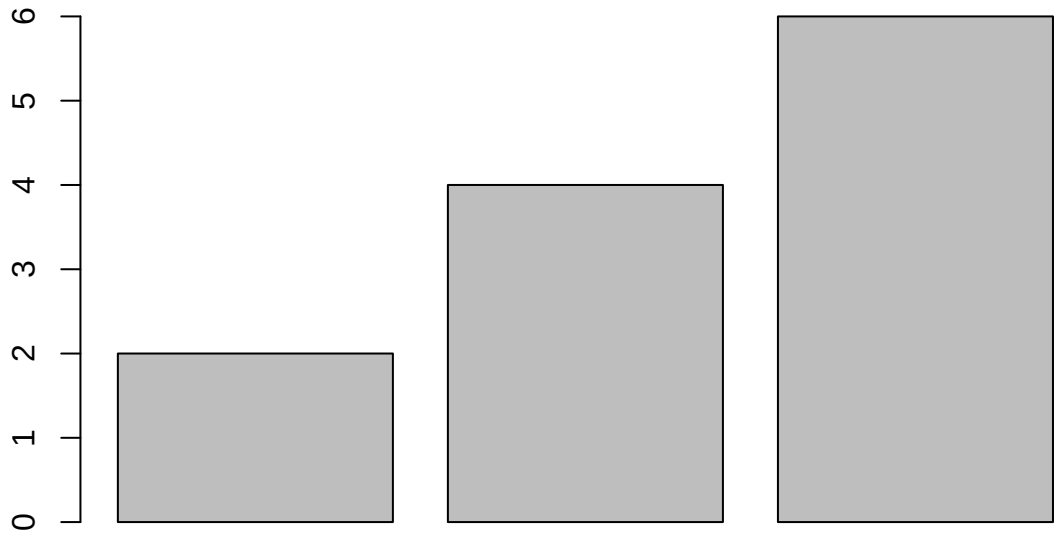
```
a <- c(1, 1.3, 1.6, 2, 3.5)
b <- 2 + 3 * a
plot(a, b, main = "Linear function",
     xlab = "a - explanatory variable",
     ylab = "b - response variable",
     col = "green")
lines(a, b)
```



6.1.2 Bar charts

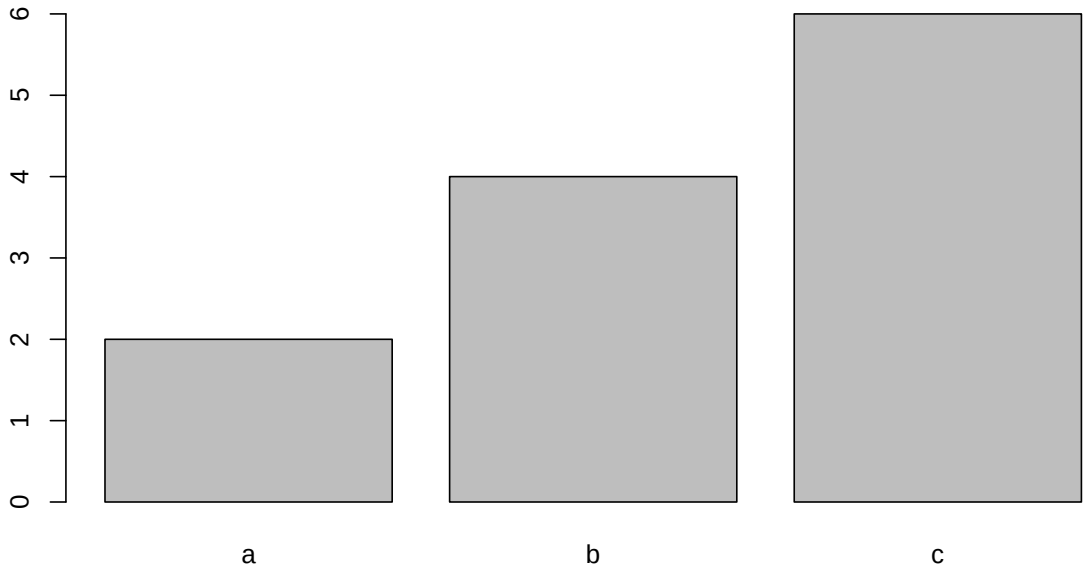
R has a `barplot()` function for creating bar charts. We will have a look at two common variations: single bars per group and multiple bars per group.

```
x1 <- c(2, 4, 6)
barplot(x1)
```



We observe there are no names for the bars. This is due to the fact that the vector was not named.

```
barplot(x1, names.arg = c("a", "b", "c"))
```

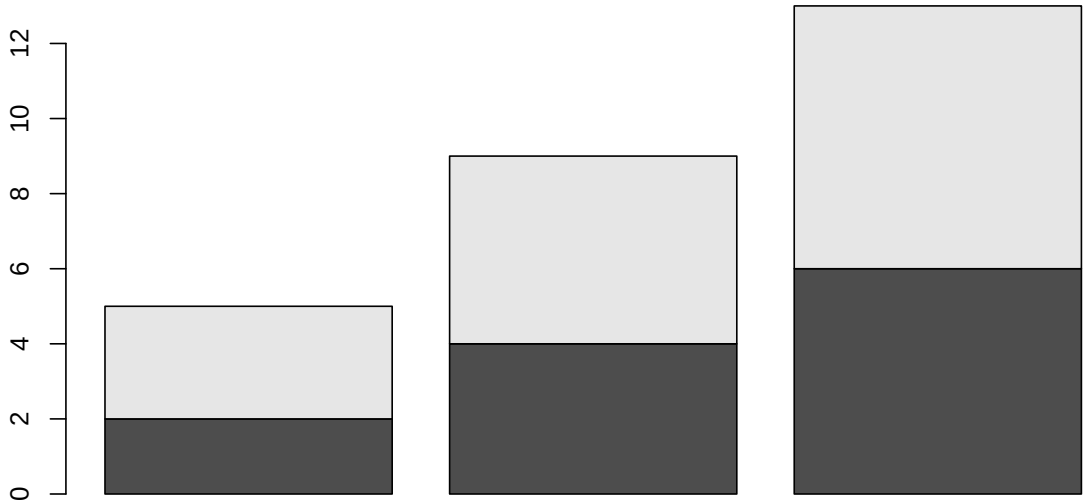


If we want multiple bars per group the input argument of the `barplot()` function will be a matrix:

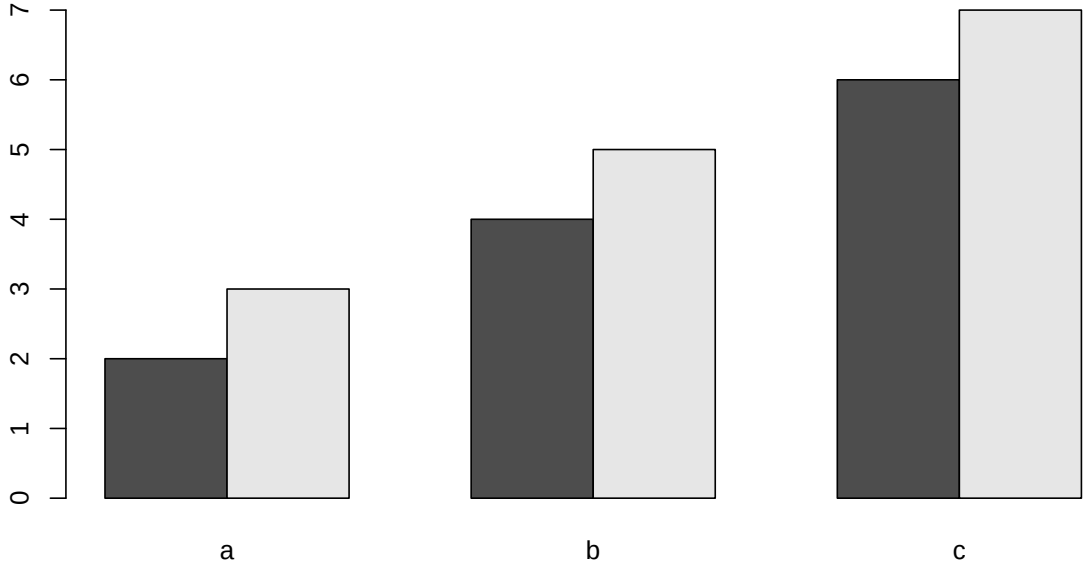
```
x2 <-  c(3,5,7)
x3 <- rbind(x1, x2)
x3
```

```
##    [,1] [,2] [,3]
## x1    2    4    6
## x2    3    5    7
```

```
barplot(x3)
```




```r
## or nicer:
barplot(x3, names.arg = c("a", "b", "c"), beside = TRUE)
```

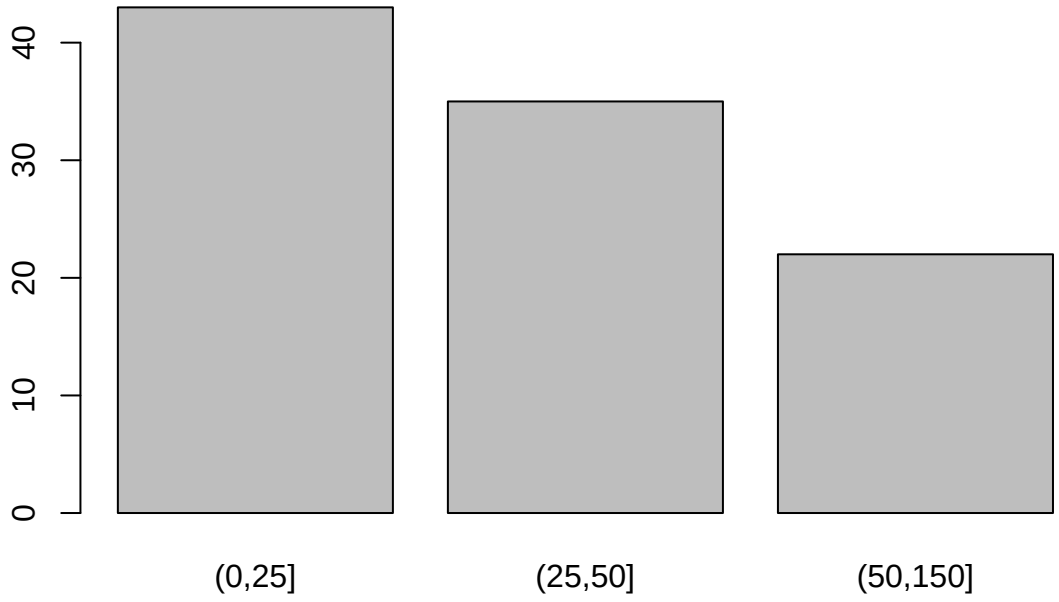


Exercise 8

1. Run the following command in R and assign it to an object `z1`: `sample(1:4, size = 20, replace = TRUE)`
2. Compute the frequencies for this vector by using the function `table()`.
3. Draw a bar chart of the absolute frequencies of the vector `z1`. Use a different color for each bar.

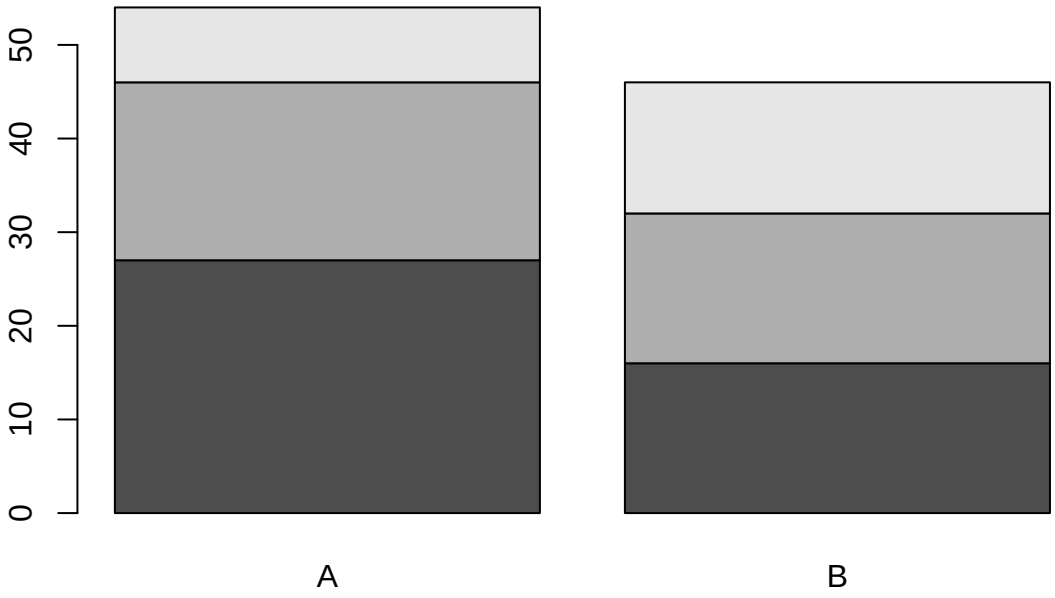
Let us build a barplot of `NewspaperCat` variable in our data set. First we need to do the frequency table of the variable:

```
tab <- table(advertising$NewspaperCat)
barplot(tab)
```



Let us build a stacked barplot of `NewspaperCat` and the `Country` variable in our data set. First we need to do the contingency table of the two variables:

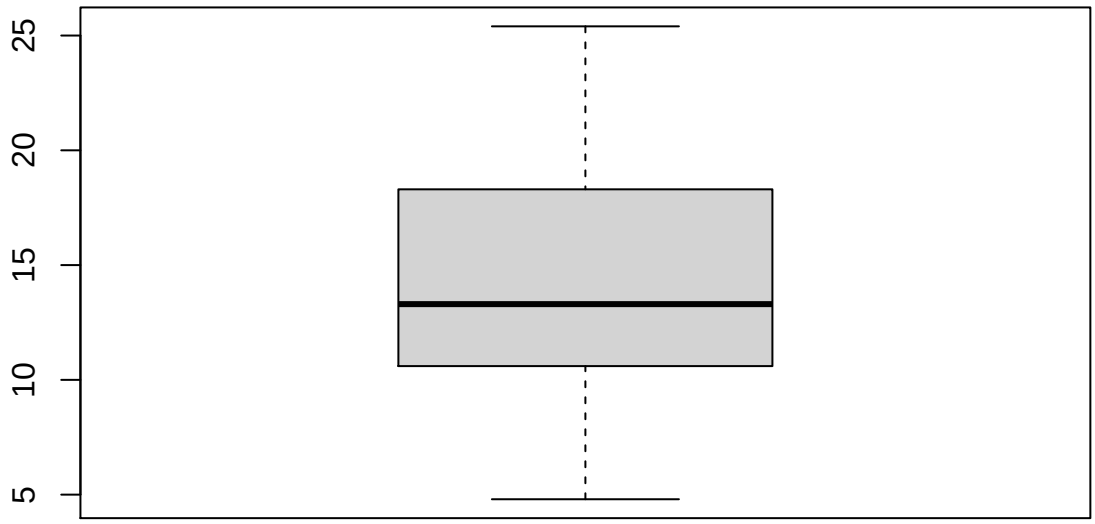
```
tab2 <- xtabs(~ NewspaperCat + Country, data = advertising)
barplot(tab2)
```



6.1.3 Boxplots

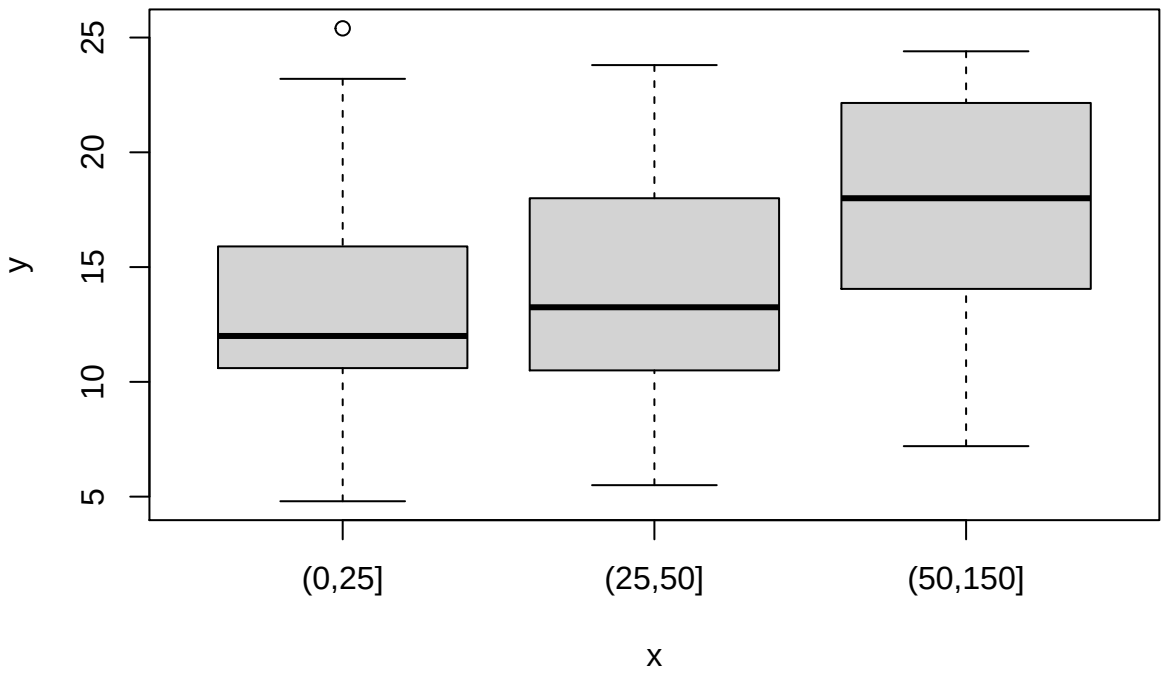
Boxplots can be created using the `boxplot()` function:

```
boxplot(advertising_nodup$Sales)
```



Assume we want to create a boxplot of the Sales in each group of the `NewspaperCat`. If we supply the generic `plot()` function with a continuous response (y) variable and a categorical predictor (x) variable, it will automatically make parallel boxplots:

```
plot(x = advertising_nodup$NewspaperCat,
     y = advertising_nodup$Sales)
```



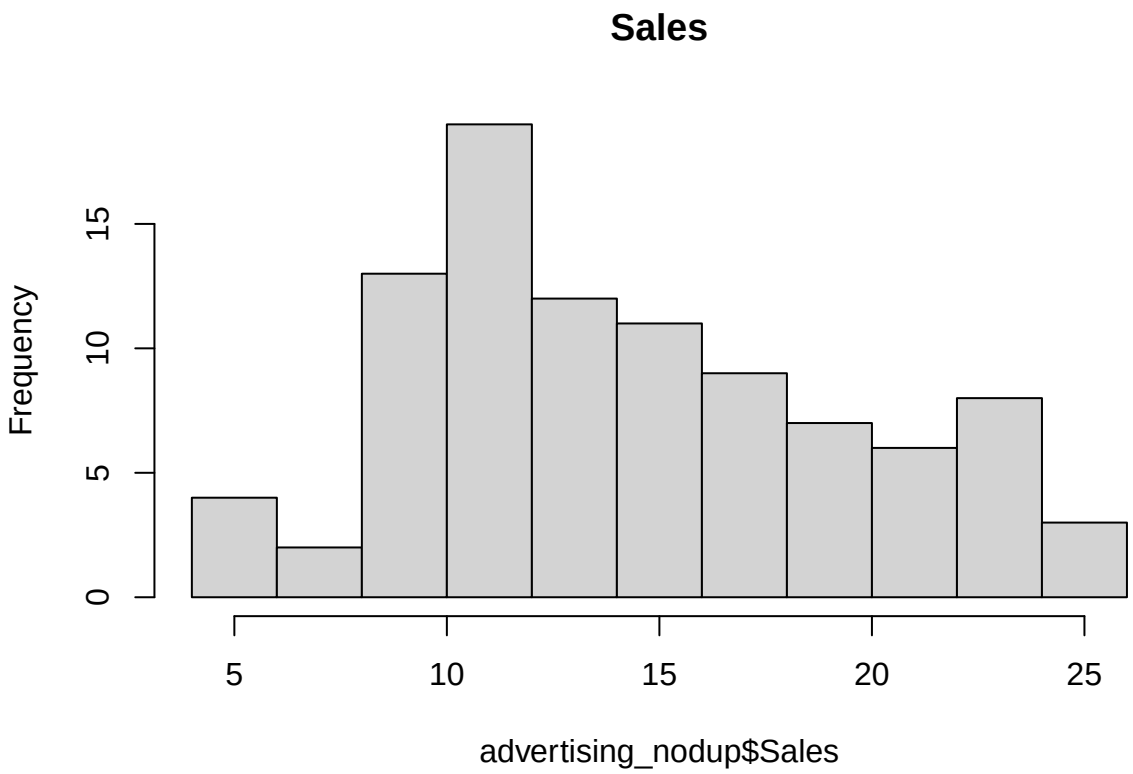
Alternatively, the formula syntax `y ~ x` in R can be used:

```
plot(Sales ~ NewspaperCat, data = advertising_nodup)
```

6.1.4 Histograms

Another graphical representation for showing the distribution of numerical variable is histograms.

```
hist(advertising_nodup$Sales, main = "Sales")
```



Note that `hist()` automatically selects the number of bins based on the range and resolution of the data. This can be changed through the `breaks` argument. Multiple histograms can be created using the e.g., `lattice` or `ggplot2` packages.

6.1.5 Spineplots

Spineplots are a useful graphical representation for contingency tables. Assume `x3` is a contingency table for two categorical variables with 2 and 3 groups respectively:

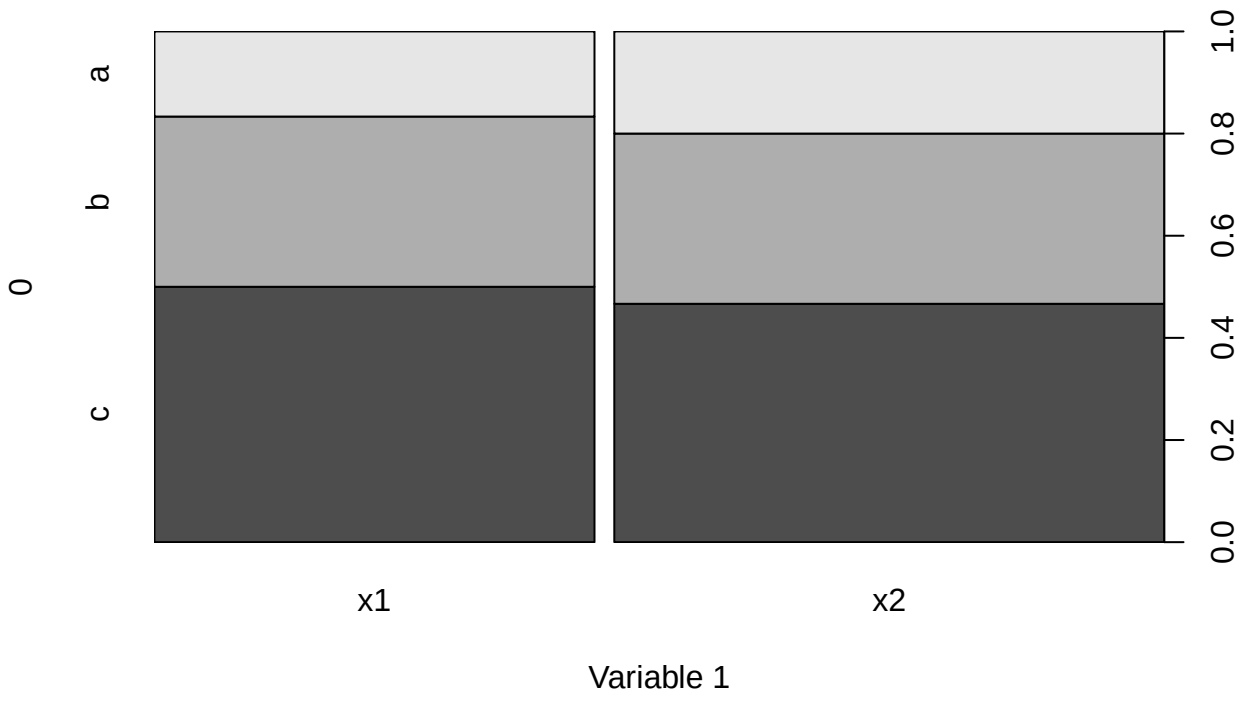
```
x3
```

```
##    [,1] [,2] [,3]
## x1    2    4    6
## x2    3    5    7
```

```
## let us name the columns of x3
colnames(x3) <- c("a", "b", "c")
```

Spineplots (also known as mosaic plots for categorical variables) are a generalization of stacked bar plots where the areas of rectangular tiles are proportional to the frequencies in the contingency table:

```
spineplot(x3, xlab = "Variable 1")
```



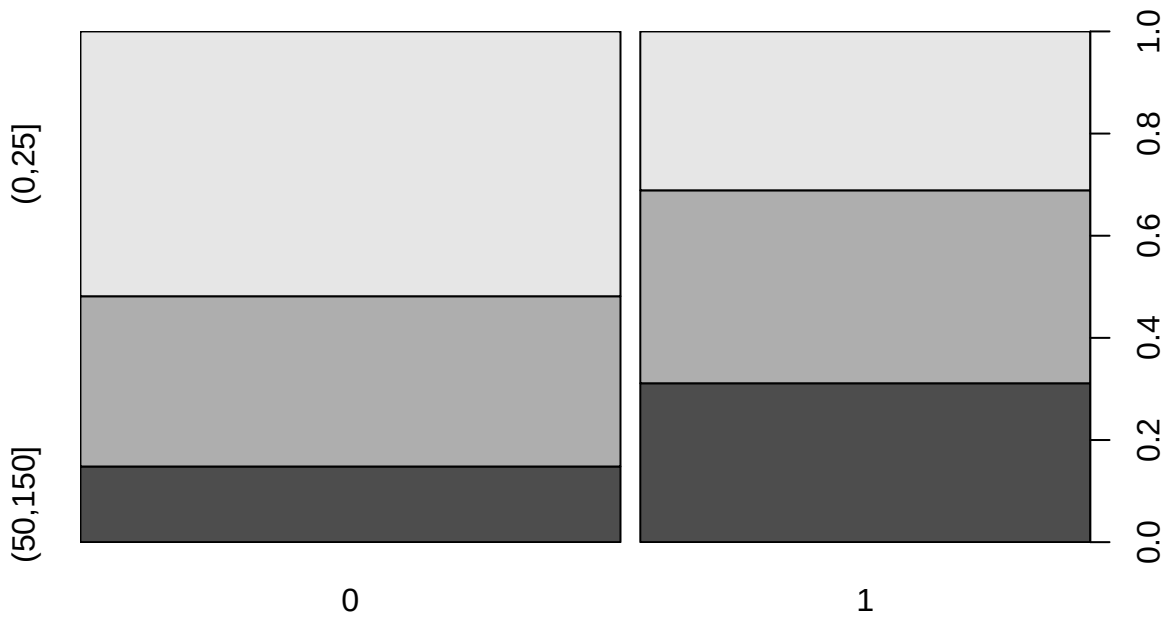
Exercise 9

1. Create a contingency table for the variables `NewspaperCat` and `SalesBin` in the `advertising` data set using the `table(x, y)` function.
2. Produce a spineplot.

```
tab <- table(advertising$SalesBin,
             advertising$NewspaperCat)
tab
```

```
##
##     (0,25] (25,50] (50,150]
##   0     28      18        8
##   1     14      17       14
```

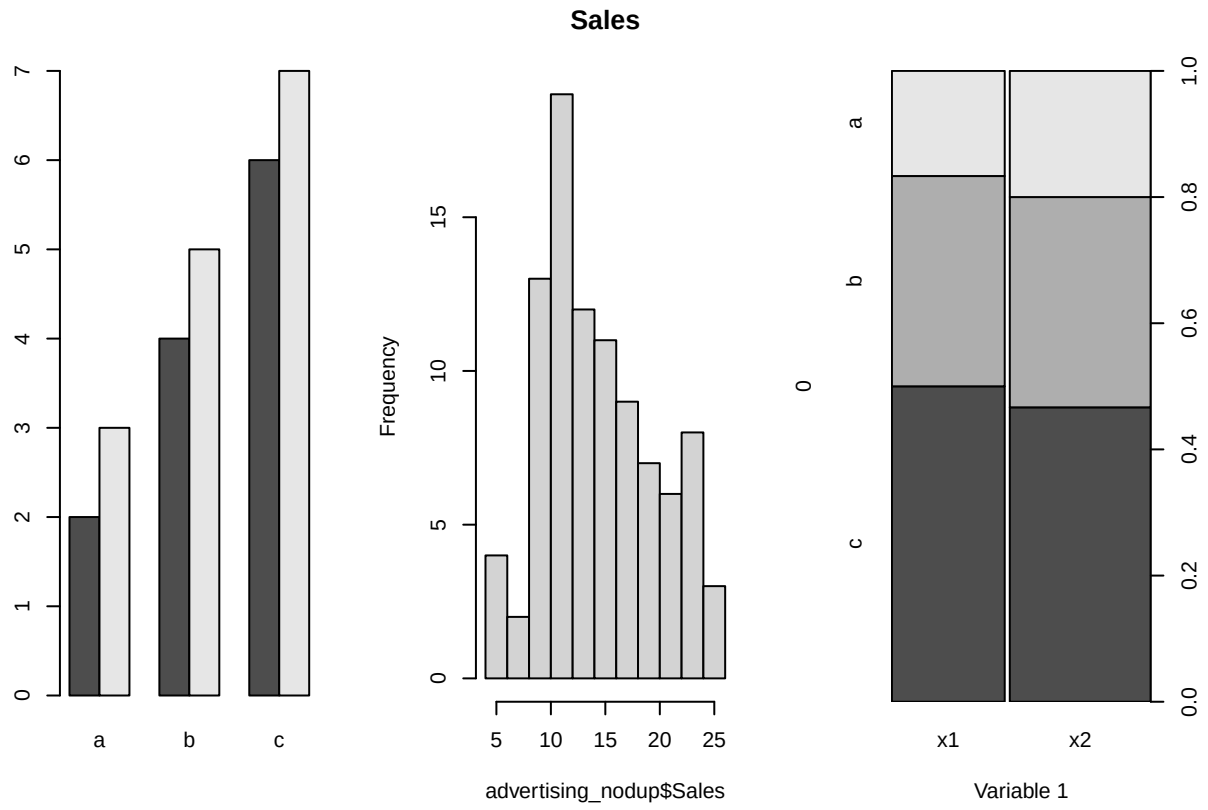
```
spineplot(tab)
```



6.2 Multi-panel Plots

Multi-panel plot allows for multiple plotting regions to show up simultaneously within the same plotting device. First, the layout of the plotting region must be changed. The easiest way to set up the device for multi-panel plotting is by using the `mfrow` argument in the `par()` function:

```
op <- par(mfrow = c(1, 3))
barplot(x3, names.arg = c("a", "b", "c"), beside = TRUE)
hist(advertising_nodup$Sales, main = "Sales")
spineplot(x3, xlab = "Variable 1")
```




```
par(op) ## reset settings
```

6.3 Exporting Plots

There are two main ways to save permanent copies of R plots.

- The first is a point-and-click method that saves the plots in low-resolution. In the RStudio built-in graphics device: Right above the plot, click `Export > Save as Image`.
- The second method produces cleaner-looking high-resolution plots with code that can be embedded in the script by using the functions `png()`, `pdf()`, `jpeg` etc. :

```
# step 1: Make a pixels per inch object
ppi = 600

# step 2: Call the figure file creation function
png("TestFigure.png", h = 8 * ppi, w = 8 * ppi, res = ppi)

# step 3: Run the plot
spineplot(x3, xlab = "Variable 1", ylab = "Variable 2")

# step 4: Close the device
dev.off()
```

```
## pdf
##   2
```

This will save the image in the working directory.

Exercise 10 - Work on your own data set

1. Import in your data set into R.
2. Make sure it is correctly imported. What type of variables does the data frame contain?
3. Print the summary statistics for this data frame.
4. Are there missing values or duplicated values? You can either eliminate them or try to replace them with another value (i.e., imputation).
5. Compute other important statistics for the most important variables (maybe standard deviation).
6. Plot the most important variables for the analysis.

---

7 Linear model in R

Source:

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York:Springer, 2013.

http://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf

The R notebook in `notebooks/advertising.Rmd` contains the analysis of the advertising data including processing, descriptives and linear modeling. The following contains excerpts of this notebook.

7.1 Revisiting the `advertising` data set

The data set contains statistics about the sales of a product in different markets together with advertising budgets in each of these markets for different media channels: **TV**, **Radio** and **Newspaper**.

Further details:

- The Sales are in thousands of units.
- The budgets are in thousands of dollars.
- Additionally, the variable `NewspaperCat` is a categorization of the newspaper budget into three categories: below 25,000, between 25,000 and 50,000, above 50,000.

```
advertising <- read.csv("data/advertising.csv")
```

Let's have a look at header of the data set using the function `head()`, displaying the first 6 rows of the `data.frame`:

```
head(advertising)
```

```
##   X.1 X    TV Radio Newspaper Sales NewspaperCat Country
## 1   1 1 230.1  37.8      69.2  22.1     (50,150]       A
## 2   2 2  44.5  39.3      45.1  10.4      (25,50]       B
## 3   3 3  17.2  45.9      69.3   9.3     (50,150]       B
## 4   4 4 151.5  41.3      58.5  18.5     (50,150]       B
## 5   5 5 180.8  10.8      58.4  12.9     (50,150]       A
## 6   6 6   8.7  48.9      75.0   7.2     (50,150]       A
```

The first two columns are redundant, as they only contain the index of the observation. This is also contained in the `rownames()` of the `data.frame`. We should hence eliminate them.

```
advertising <- advertising[, -c(1:2)]
```

Inspect the `structure` of the data.frame:

```
str(advertising)
```

```
## 'data.frame':    100 obs. of  6 variables:
##  $ TV          : num  230.1 44.5 17.2 151.5 180.8 ...
##  $ Radio       : num  37.8 39.3 45.9 41.3 10.8 48.9 32.8 32.8 19.6 2.1 ...
##  $ Newspaper   : num  69.2 45.1 69.3 58.5 58.4 75 23.5 23.5 11.6 1 ...
##  $ Sales       : num  22.1 10.4 9.3 18.5 12.9 7.2 11.8 11.8 13.2 4.8 ...
##  $ NewspaperCat: chr  "(50,150]" "(25,50]" "(50,150]" "(50,150]" ...
##  $ Country     : chr  "A" "B" "B" "B" ...
```

Convert characters to factors:

```
advertising$NewspaperCat <- factor(advertising$NewspaperCat)
advertising$Country      <- factor(advertising$Country)
```

Eliminate any missing values:

```
advertising <- na.omit(advertising)
```

Eliminate any duplicates:

```
advertising <- subset(advertising, duplicated(advertising) == FALSE)
```

We left with

```
nrow(advertising)
```

```
## [1] 94
```

observations.

7.2 Sample descriptives

7.2.1 Summary statistics

```
summary(advertising)
```

```
##        TV             Radio         Newspaper          Sales
##  Min.   :  5.40   Min.   : 1.40   Min.   :  0.30   Min.   : 4.80
##  1st Qu.: 74.85   1st Qu.:12.95   1st Qu.: 15.93   1st Qu.:10.62
##  Median :145.10   Median :26.20   Median : 30.65   Median :13.30
##  Mean   :148.66   Mean   :24.52   Mean   : 31.71   Mean   :14.46
##  3rd Qu.:216.70   3rd Qu.:36.15   3rd Qu.: 44.65   3rd Qu.:18.23
##  Max.   :293.60   Max.   :49.60   Max.   :114.00   Max.   :25.40
##    NewspaperCat Country
##  (0,25]  :41    A:52
##  (25,50] :34    B:42
##  (50,150]:19
##
##
##
```

7.2.2 Correlation

What is the correlation (Pearson's empirical correlation coefficient) between the variables in the first four columns of the data set? (We omit the categorical variable).

```
cor(advertising[, 1:4])
```

```
##                    TV      Radio   Newspaper     Sales
## TV         1.00000000 0.08810839 -0.04152654 0.7894696
## Radio      0.08810839 1.00000000  0.46366815 0.6019071
## Newspaper -0.04152654 0.46366815  1.00000000 0.1876800
## Sales      0.78946961 0.60190706  0.18767998 1.0000000
```

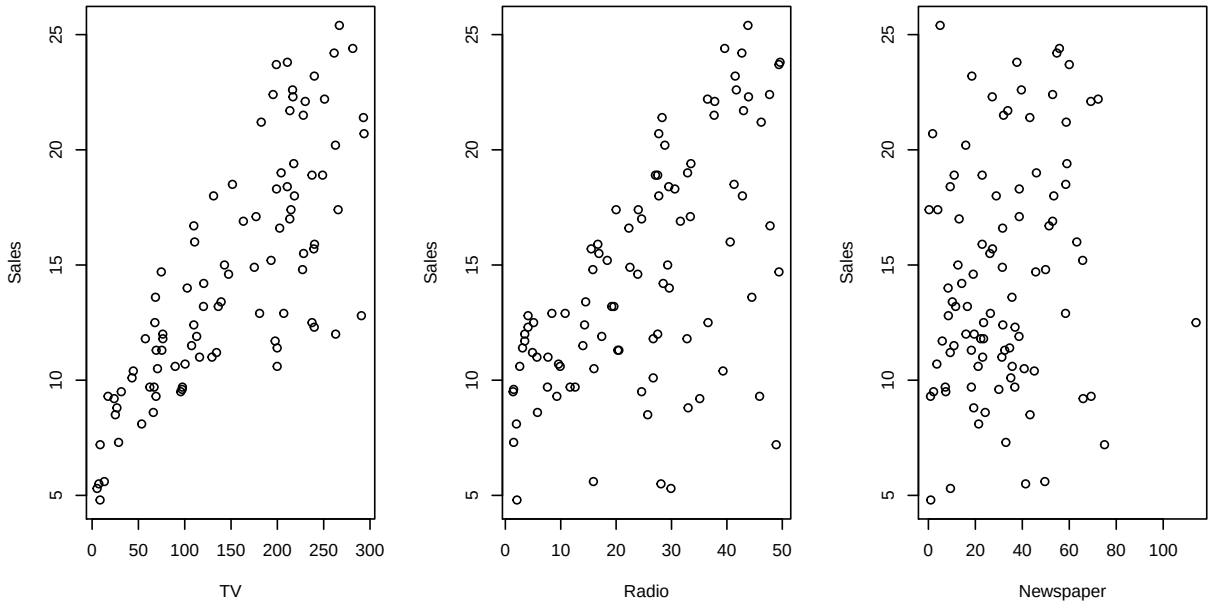
Let us perform a correlation test for `Sales` and `TV`:

```
cor.test(~ Sales + TV, data = advertising)
```

```
##
##  Pearson's product-moment correlation
##
## data:  Sales and TV
## t = 12.337, df = 92, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6986008 0.8552766
## sample estimates:
##       cor
## 0.7894696
```

7.3 Visualizing the multivariate relationships

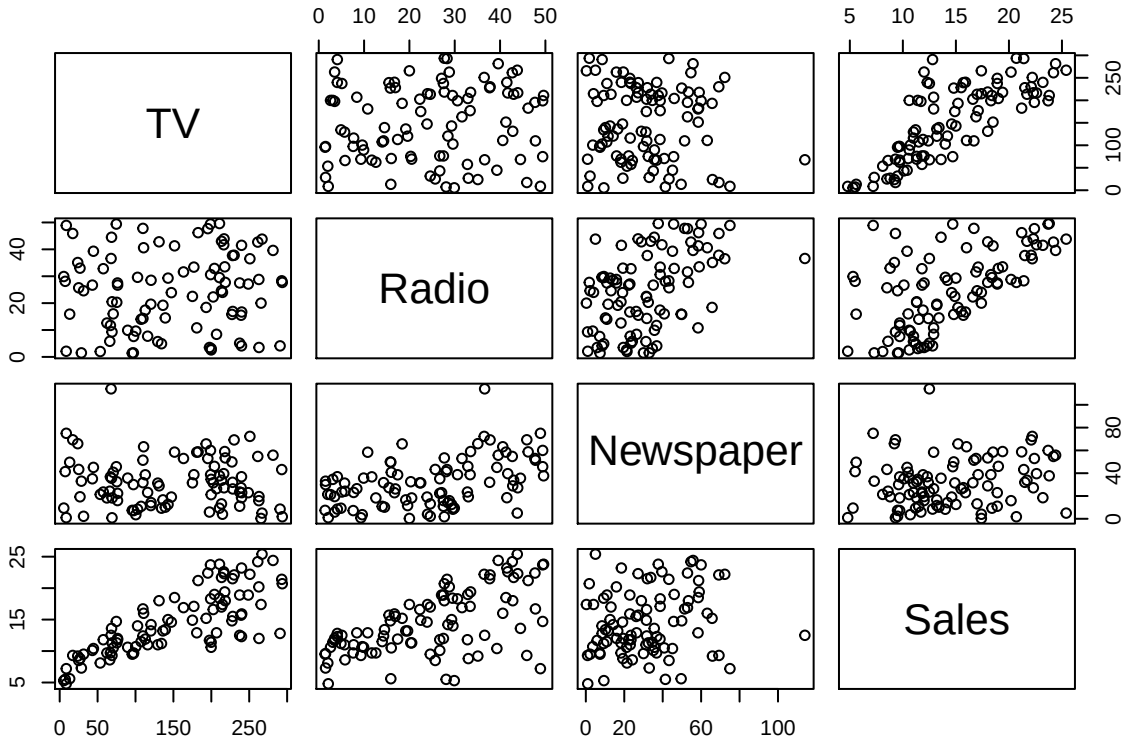
```
## plot(y ~ x)
op <- par(mfrow = c(1, 3))
plot(Sales ~ TV, data = advertising)
plot(Sales ~ Radio, data = advertising)
plot(Sales ~ Newspaper, data = advertising)
```




```
par(op)
```

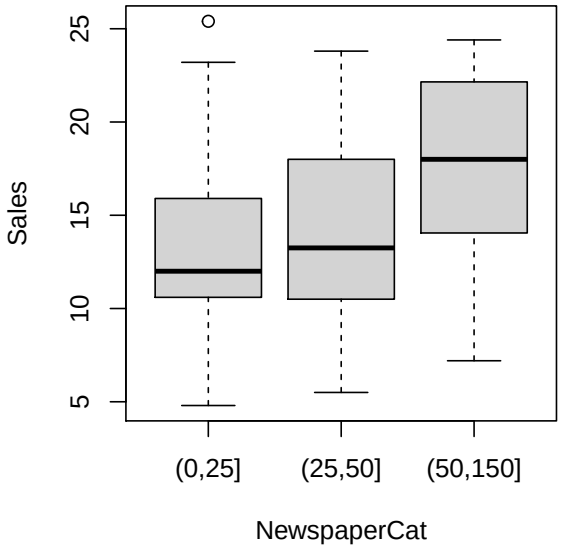
Scatterplots for all pairs of variables in a data frame can be produced by:

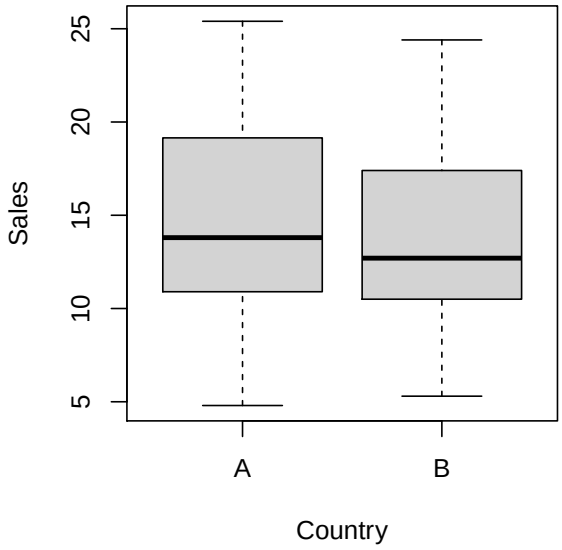
```
## plot scatterplots for all pairs of numeric variables
pairs(advertising[c("TV", "Radio", "Newspaper", "Sales")])
```



Let us check the relationship between sales and the categorical variables:

```
op <- par(mfrow = c(1,2))
plot(Sales ~ NewspaperCat, data = advertising) # boxplot
plot(Sales ~ Country, data = advertising) # boxplot
```






```
par(op)
```

We can perform ANOVA to check whether there are significant differences:

```
m_aov_news <- aov(Sales ~ NewspaperCat, data = advertising)
summary(m_aov_news)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## NewspaperCat  2  230.4  115.21   4.852 0.00995 **
## Residuals    91 2160.9   23.75
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
m_aov_country <- aov(Sales ~ Country, data = advertising)
summary(m_aov_country)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## Country      1   15.7   15.74    0.61  0.437
## Residuals   92 2375.5   25.82
```

If we want to check for the joint additive significance of the two categorical variables in explaining sales, we can use:

```
m_aov_country_news_add <- aov(Sales ~ Country + NewspaperCat, data = advertising)
summary(m_aov_country_news_add)
```

```
##              Df Sum Sq Mean Sq F value  Pr(>F)
## Country       1   15.7   15.74   0.671 0.41471
## NewspaperCat  2  265.3  132.66   5.658 0.00485 **
## Residuals    90 2110.2   23.45
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we want to check for the interaction effect of the two categorical variables in explaining sales, we can use:

```r
m_aov_country_news_inter <- aov(Sales ~ Country * NewspaperCat, data = advertising)
summary(m_aov_country_news_inter)
```

```
##                      Df Sum Sq Mean Sq F value  Pr(>F)
## Country               1   15.7   15.74   0.661 0.41839
## NewspaperCat          2  265.3  132.66   5.570 0.00528 **
## Country:NewspaperCat  2   14.4    7.18   0.301 0.74053
## Residuals            88 2095.9   23.82
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

7.4 Simple linear regression

- Goal: predict a quantitative variable Y on the basis of a single predictor X by assuming an approximately linear relationship.
- We assume:

$$Y = \underbrace{\beta_0 + \beta_1 X}_{linear\ function} + \underbrace{\epsilon}_{error\ term}, \tag{1}$$

  where β_0 and β_1 are unknown **parameters** or **coefficients**.
- Given some estimates $\hat{\beta}_0$ and $\hat{\beta}_1$, we can predict future values of Y by using the regression line:

$$\widehat{y} = \hat{\beta}_0 + \hat{\beta}_1 x,$$

  where $\hat{y}$ indicates a prediction of Y on the basis of X being equal to x.
- The coefficients can be estimated using the ordinary least squares principle (OLS).

7.4.1 Fitting the model in R

Let us fit a simple linear regression `Sales ~ TV` using OLS using the `lm()` function:

```r
fitTV <- lm(Sales ~ TV, data = advertising)
```

The fitted coefficients are:

```r
fitTV
```

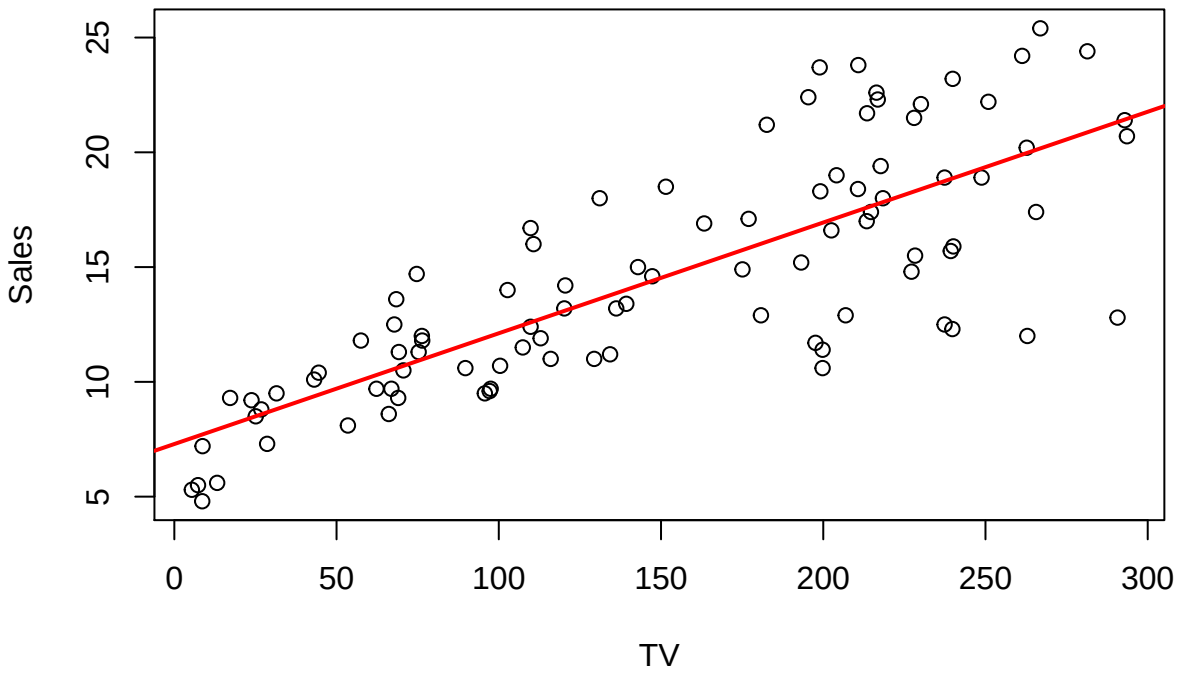
```
##
## Call:
## lm(formula = Sales ~ TV, data = advertising)
##
## Coefficients:
## (Intercept)           TV
##     7.29194      0.04825
```

The fitted regression line is:

$$\widehat{Sales} = 7.29194 + 0.04825 \cdot TV \tag{2}$$

We can add the fitted regression line to the scatterplot by `abline()`:

```r
plot(Sales ~ TV, data = advertising)
abline(fitTV, col = "red", lwd = 2)
```

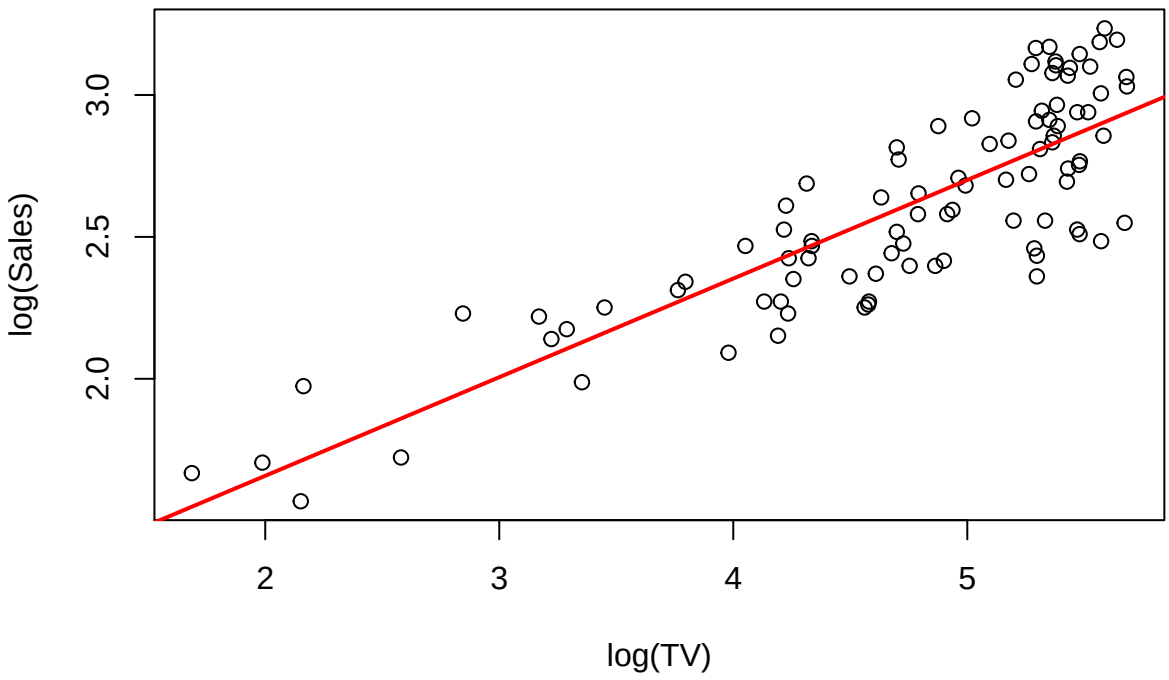


The linear relationship seems to fit well, except for the left hand side of the plot, where for low advertising budgets the linear relationship does not seem to hold. A linear regression on the scale might be more appropriate:

```
fitTVlog <- lm(log(Sales) ~ log(TV), data = advertising)
```

Plot the regression line on the log scale:

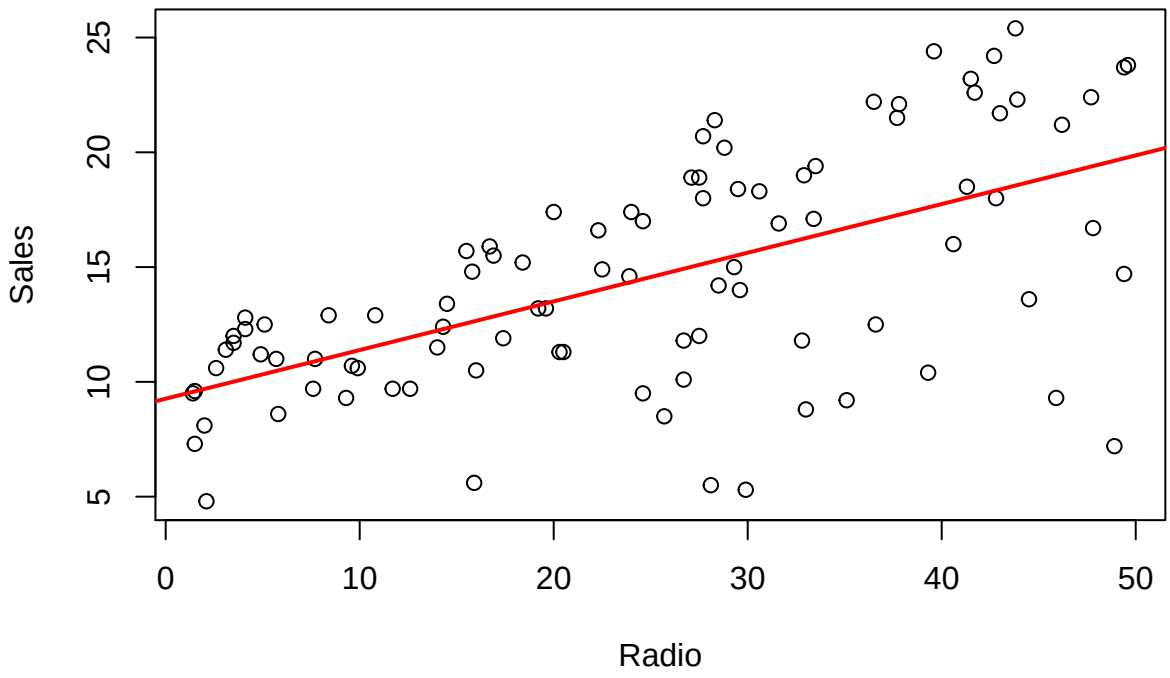
```
## plot on the log scale
plot(log(Sales) ~ log(TV),  data = advertising)
abline(fitTVlog,  col = "red", lwd = 2)
```



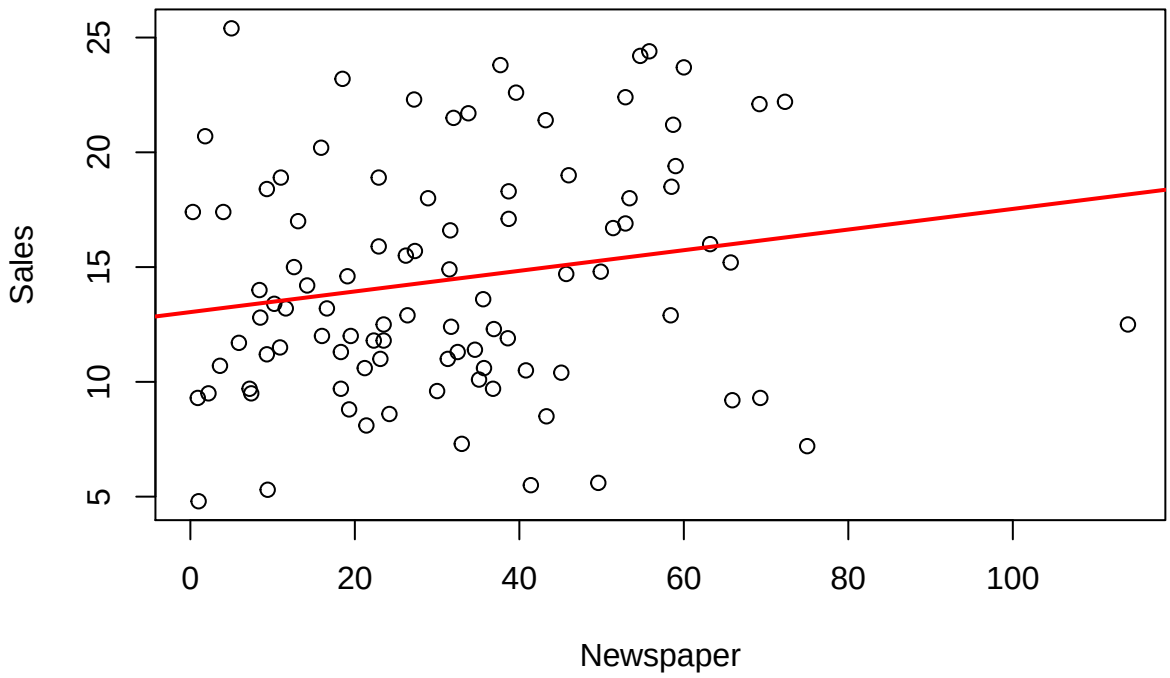
Exercise 11

1. Fit a simple regression model for `Sales ~ Radio` and `Sales ~ Newspaper`

```
fitRadio <- lm(Sales ~ Radio,
               data = advertising)
plot(Sales ~ Radio, data = advertising)
abline(fitRadio, lwd = 2, col = "red")
```




```
fitNewspaper <- lm(Sales ~ Newspaper, data = advertising)
plot(Sales ~ Newspaper, data = advertising)
abline(fitNewspaper, lwd = 2, col = "red")
```



2. Produce the scatterplot together with regression line for both models.

7.5 Multiple linear regression

The multiple linear regression model is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon.$$

We now estimate a model with all advertising budgets as variables:

```
fitAll <- lm(Sales ~ TV + Radio + Newspaper,
             data = advertising)
```

```
fitAll
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper, data = advertising)
##
## Coefficients:
## (Intercept)           TV        Radio    Newspaper
##    3.254695     0.045164     0.195392    -0.009315
```

The regression model is:

$$\widehat{Sales} = 3.254695 + 0.045164 \cdot TV + 0.195392 \cdot Radio + -0.009315 \cdot Newspaper$$

7.5.1 Summary of the linear model

When applied to an `lm` object, the `summary` function creates and prints the following output:

```
summary(fitAll)
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper, data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3037 -0.8401  0.1988  1.1080  2.7679
##
## Coefficients:
##              Estimate Std. Error t value        Pr(>|t|)
## (Intercept)  3.254695   0.439685   7.402 0.0000000000683 ***
## TV           0.045164   0.001948  23.182         < 2e-16 ***
## Radio        0.195392   0.012659  15.435         < 2e-16 ***
## Newspaper   -0.009315   0.008584  -1.085           0.281
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.546 on 90 degrees of freedom
## Multiple R-squared:  0.9101, Adjusted R-squared:  0.9071
## F-statistic: 303.5 on 3 and 90 DF,  p-value: < 2.2e-16
```

The standard output contains the call and the summary statistics of the residuals.

Moreover, for each estimated coefficient we have the `Estimate`, the standard error of the coefficients `Std. Error`, the value t-statistic (`t value`) used in the t-test

$$H_0 : \beta_j = 0$$
$$H_A : \beta_j \neq 0$$

and the corresponding p-value `Pr(>|t|)` by using the result that the test statistic of this hypothesis test follows a t-distribution under the null hypothesis.

Other goodness-of-fit statistics such as the `Residual standard error`, `Multiple R-squared`, `Adjusted R-squared` and the `F-statistic` corresponding to the test $H_0 : \beta_1 = \beta_2 = \ldots = \beta_P = 0$ are also provided.

The residuals can be extracted from the linear model by

```
res <- resid(fitAll)
summary(res)
```

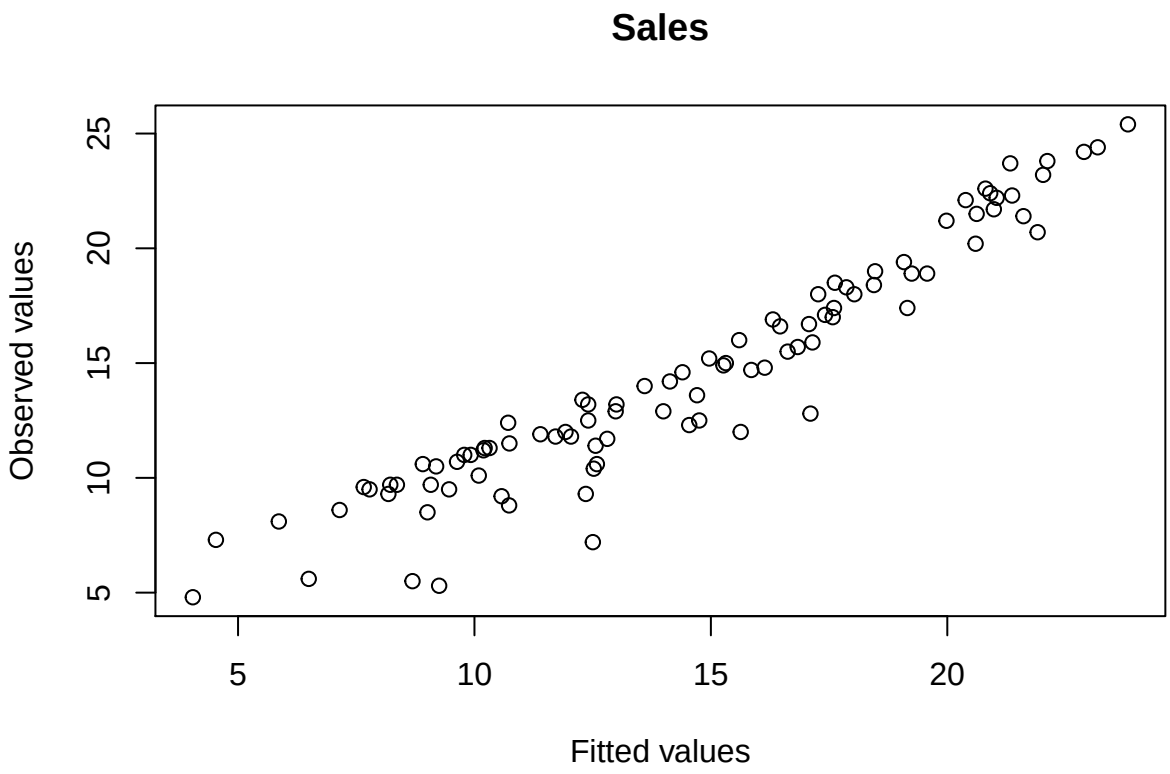
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.3037 -0.8401  0.1988  0.0000  1.1080  2.7679
```

The fitted values can be extracted from the linear model by

```
yhat <- fitted(fitAll)
```

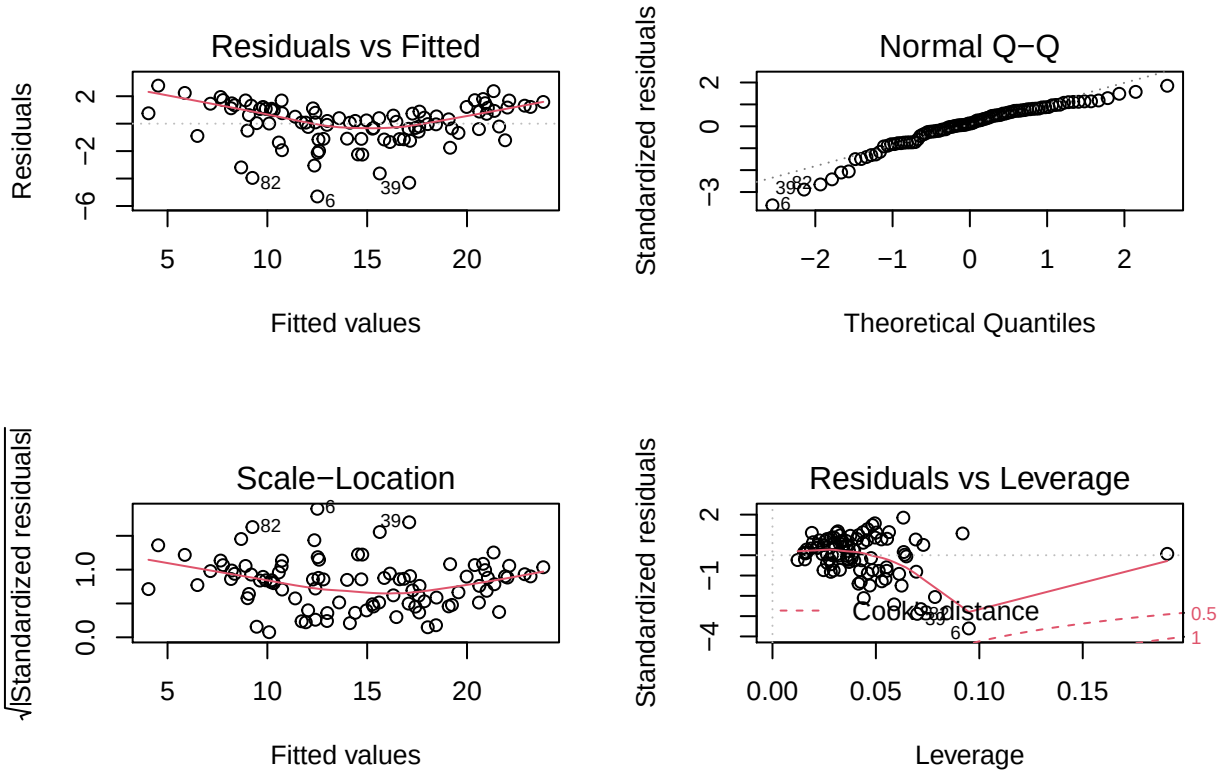
Plot the fitted values vs. the original values:

```
plot(yhat, advertising$Sales, ylab = "Observed values", xlab = "Fitted values",
     main = "Sales")
```



7.5.2 Diagnostic plots

```
par(mfrow = c(2, 2))
plot(fitAll)
```



Four diagnostic plots are produced by default with the `plot()` function:

- **Fitted values vs residual plot** - a good fit would be indicated by a random plot.
- **Normal qqplot** - checking the normality of residuals (important for validity of standard errors); if the residuals are not normally distributed, the standard errors might be biased and hence the hypotheses test also.
- The **Scale-Location** plot shows if residuals are spread equally along the ranges of predictor (for the assumption of homoskedasticity).
- **Residuals vs Leverage**: Identification of influential observations. One might eliminate them from the data set, repeat the analysis and check whether the results have changed.

7.6 Further topics

7.6.1 Model selection

One can compare models linear model based on various statistics, e.g., Akaike information criterion (AIC). Based on AIC, the model with the lowest AIC is selected.

Exercise 12

1. Fit the regression model for `Sales ~ TV + Radio` and save it as an object named `fitTVRadio`.

```
fitTVRadio <- lm(Sales ~ TV + Radio, data = advertising)
```

Let's inspect the AIC of the two models:

```
AIC(fitAll, fitTVRadio)
```

```
##            df      AIC
## fitAll      5 354.5712
## fitTVRadio  4 353.7932
```

The model with two independent variables fits better.

Automated approaches that search through a subset of all models are typically employed when performing model selection. The function `step()` in R used a stepwise algorithm based on the AIC as a default.

- Forward selection: start with null model (a model that contains an intercept but no predictors) and progressively add new variables until some stopping rule is satisfied.
- Backward selection: start with all variables in the model; remove the variable with e.g., the largest p-value; continue until some stopping rule is satisfied.
- Mixed selection: a combination of forward and backward selection.

```
## forward selection based on AIC
step(fitAll, direction = "forward")
```

```
## Start:  AIC=85.81
## Sales ~ TV + Radio + Newspaper
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper, data = advertising)
##
## Coefficients:
## (Intercept)           TV        Radio    Newspaper
##    3.254695     0.045164     0.195392    -0.009315
```

```
## backward selection based on AIC
step(fitAll, direction = "backward")
```

```
## Start:  AIC=85.81
## Sales ~ TV + Radio + Newspaper
##
##             Df Sum of Sq     RSS     AIC
## - Newspaper  1      2.81  217.91  85.033
## <none>                    215.09  85.811
## - Radio      1    569.36  784.46 205.439
## - TV         1   1284.39 1499.48 266.341
##
## Step:  AIC=85.03
## Sales ~ TV + Radio
##
##         Df Sum of Sq     RSS     AIC
## <none>                217.91  85.033
## - Radio  1    682.98  900.88 216.448
## - TV     1   1307.03 1524.93 265.923
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio, data = advertising)
```

```
##
## Coefficients:
## (Intercept)           TV        Radio
##     3.08815      0.04536      0.18894
```

```
## forward-backward selection based on AIC
step(fitAll, direction = "both")
```

```
## Start:  AIC=85.81
## Sales ~ TV + Radio + Newspaper
##
##             Df Sum of Sq     RSS     AIC
## - Newspaper  1      2.81  217.91  85.033
## <none>                    215.09  85.811
## - Radio      1    569.36  784.46 205.439
## - TV         1   1284.39 1499.48 266.341
##
## Step:  AIC=85.03
## Sales ~ TV + Radio
##
##             Df Sum of Sq     RSS     AIC
## <none>                    217.91  85.033
## + Newspaper  1      2.81  215.09  85.811
## - Radio      1    682.98  900.88 216.448
## - TV         1   1307.03 1524.93 265.923
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio, data = advertising)
##
## Coefficients:
## (Intercept)           TV        Radio
##     3.08815      0.04536      0.18894
```

- Mixed selection: a combination of forward and backward selection.

```
## forward-backward selection based on AIC
step(fitAll, direction = "both")
```

```
## Start:  AIC=85.81
## Sales ~ TV + Radio + Newspaper
##
##             Df Sum of Sq     RSS     AIC
## - Newspaper  1      2.81  217.91  85.033
## <none>                    215.09  85.811
## - Radio      1    569.36  784.46 205.439
## - TV         1   1284.39 1499.48 266.341
##
## Step:  AIC=85.03
## Sales ~ TV + Radio
##
##             Df Sum of Sq     RSS     AIC
## <none>                    217.91  85.033
## + Newspaper  1      2.81  215.09  85.811
## - Radio      1    682.98  900.88 216.448
## - TV         1   1307.03 1524.93 265.923
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio, data = advertising)
##
## Coefficients:
## (Intercept)           TV        Radio
##     3.08815      0.04536      0.18894
```

7.6.2 Qualitative explanatory variables

We have a look at the categorical newspaper variable:

```
table(advertising$NewspaperCat)
```

```
##
##   (0,25]  (25,50] (50,150]
##       41       34       19
```

Let us fit a linear regression with the categorical variable as explanatory variable:

```
fitNewsCat <- lm(Sales ~ NewspaperCat, data = advertising)
```

Categorical covariates (explanatory variables) are transformed to dummy variables (which take on 0 or 1 values) why used in a regression setting. By default R uses the treatment contrasts, namely $K-1$ dummy variables are created, where K is the number of groups of the categorical variable. In our example we have:

$$d_{i1} = \begin{cases} 1 & \text{if the newspaper budget is in the range (25, 50]} \\ 0 & \text{otherwise} \end{cases}$$

and

$$d_{i2} = \begin{cases} 1 & \text{if the newspaper budget is in the range (50, 150]} \\ 0 & \text{otherwise} \end{cases}$$

Note that we have not created any dummy variable for the first group (0, 25] as this is considered a baseline. The model estimated above becomes:

$$\widehat{Sales} = \beta_0 + \beta_1 d_{i1} + \beta_2 d_{i2} = \begin{cases} \beta_0 + \beta_1 & \text{if the newspaper budget is in the range (25, 50]} \\ \beta_0 + \beta_2 & \text{if the newspaper budget is in the range (50, 150]} \\ \beta_0 & \text{if the newspaper budget is in the range (0, 25]} \end{cases}$$

```
summary(fitNewsCat)
```

```
##
## Call:
## lm(formula = Sales ~ NewspaperCat, data = advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.2737  -3.5960  -0.6824   3.7882  12.1024
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)           13.2976     0.7610  17.473  < 2e-16 ***
## NewspaperCat(25,50]    0.8936     1.1303   0.791  0.43123
```

```
## NewspaperCat(50,150]   4.1761     1.3524   3.088  0.00267 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.873 on 91 degrees of freedom
## Multiple R-squared:  0.09636,    Adjusted R-squared:  0.0765
## F-statistic: 4.852 on 2 and 91 DF,  p-value: 0.00995
```

Interpretation of the coefficients

- Category (0, 25] is the baseline.
- The intercept tells us that the average sales in category (0,25] are at $13.2976 \times 1000 = 13297.6$ units.
- The coefficient estimate $\hat{\beta}_1$ (`NewspaperCat(25,50]`) gives the additional amount of sales in category (25,50] on top of the baseline i.e., being in category (25,50] adds $0.8936 \times 1000 = 893.6$ more units on top of the baseline of 13297.6 units. In category (25, 50] the average sales are hence $13297.6 + 893.6 = 14191.2$ units.
- The coefficient estimate $\hat{\beta}_2$ (`NewspaperCat(50,150]`) gives the additional amount of sales in category (50, 150] on top of the baseline, i.e., being in category (50, 150] adds$4.1761 \times 1000 = 4176.1$ more units on top of the baseline of 13297.6 units. In category (50, 150] the average sales are hence 17473.7 units.

7.6.3 Interactions

The linear models above assume that advertising medium is independent of the amount spent on the other media, i.e., in model `fitTVRadio` one unit increase in TV ad budget increases sales by 3.0881474 regardless of how much has been spent on radio. However, it could be that spending on radio advertising actually increases the effectiveness of TV advertising or vice-versa. The corresponding interaction model is given by:

$$\widehat{Sales} = \beta_0 + \beta_1 TV + \beta_2 Radio + \beta_3 TV \times Radio \tag{3}$$

$$\widehat{Sales} = \beta_0 + (\beta_1 + \beta_3 Radio) TV + \beta_2 Radio \tag{4}$$

$$\widehat{Sales} = \beta_0 + \beta_1 TV + (\beta_2 + \beta_3 TV) Radio \tag{5}$$

We now estimate the corresponding model in `R`:

```
fitInter <- lm(Sales ~ TV * Radio, data = advertising)
## OR
fitInter <- lm(Sales ~ TV + Radio + TV:Radio,
               data = advertising)
```

```
summary(fitInter)
```

```
##
## Call:
## lm(formula = Sales ~ TV + Radio + TV:Radio, data = advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8778 -0.3658  0.1161  0.5174  1.2969
##
## Coefficients:
##               Estimate Std. Error t value         Pr(>|t|)
## (Intercept) 7.09654960 0.33887671  20.941          < 2e-16 ***
## TV          0.01707461 0.00210081   8.128 0.00000000000222 ***
## Radio       0.02707757 0.01204117   2.249            0.027 *
```

```
## TV:Radio    0.00111106 0.00007207  15.417          < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8155 on 90 degrees of freedom
## Multiple R-squared:  0.975,  Adjusted R-squared:  0.9741
## F-statistic:  1169 on 3 and 90 DF,  p-value: < 2.2e-16
```

We can compare the model with no interaction vs the model with interaction based in AIC:

```
AIC(fitTVRadio, fitInter)
```

```
##            df      AIC
## fitTVRadio  4 353.7932
## fitInter    5 234.3229
```

The interaction model clearly fits better to the data.

7.6.4 Predictions

Let us consider a new market, for which 20000 USD have been spent on TV and 12000 USD on radio advertising. What would be the predicted sales for this market?

```
x_new <- data.frame(Sales = NA, TV = 20, Radio = 12)
```

One uses the `predict()` function:

```
predict(fitInter, newdata = x_new)
```

```
##        1
## 8.029626
```

For also extracting the confidence intervals of this prediction we add the `interval = "confidence"` argument:

```
predict(fitInter, newdata = x_new, interval = "confidence", level = 0.95)
```

```
##        fit      lwr      upr
## 1 8.029626 7.625995 8.433257
```

For extracting the prediction intervals we add the `interval = "prediction"` argument:

```
predict(fitInter, newdata = x_new, interval = "prediction", level = 0.95)
```

```
##        fit      lwr      upr
## 1 8.029626 6.360026 9.699226
```

8 The logistic model in R

By means of the logistic regression model the probability of occurrence of a certain event Y is modeled as a function of a series of independent variables $X_1, \dots X_p$. The independent variables may be categorically and/or numeric, the dependent variable is categorical. If it contains more than two categories, the multinomial logistic regression can be used. If they only use two categories it is the binary logistic regression (on which will to concentrate here).

The R notebook in `notebooks/credit.Rmd` contains the analysis of the credit data including processing, descriptives and linear modeling. The following contains excerpts of this notebook.

8.1 The `credit` data set

The `credit.csv` data contains 10000 observations on the following 4 variables.

- default: a categorical variable with groups `No` and `Yes` indicating whether the customer defaulted on their debt
- student: a categorical variable with groups `No` and `Yes` indicating whether the customer is a student
- balance: the average balance that the customer has remaining on their credit card after making their monthly payment
- income: income of customer

Source: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York:Springer, 2013.

Exercise 13

1. Go to the webpage and download to your computer `credit.csv`.

```
credit <- read.csv(url("http://statmath.wu.ac.at/~vana/Intro_Data_Analytics_R//Intro_Data_Analytics_R_P
```

```
str(credit)
```

```
## 'data.frame':    10000 obs. of  4 variables:
##  $ default: chr  "No" "No" "No" "No" ...
##  $ student: chr  "No" "Yes" "No" "No" ...
##  $ balance: num  730 817 1074 529 786 ...
##  $ income : num  44362 12106 31767 35704 38463 ...
```

Any missing values?

```
sum(complete.cases(credit))
```

```
## [1] 10000
```

8.2 Sample descriptives

```
summary(credit)
```

```
##    default            student             balance           income
##  Length:10000       Length:10000       Min.   :   0.0   Min.   :  772
##  Class :character   Class :character   1st Qu.: 481.7   1st Qu.:21340
##  Mode  :character   Mode  :character   Median : 823.6   Median :34553
##                                        Mean   : 835.4   Mean   :33517
##                                        3rd Qu.:1166.3   3rd Qu.:43808
##                                        Max.   :2654.3   Max.   :73554
```

We can convert the default and the student variables to a factor, such that R knows it is dealing with categorical variables:

```
credit$default <- factor(credit$default)
credit$default <- factor(credit$default)
```

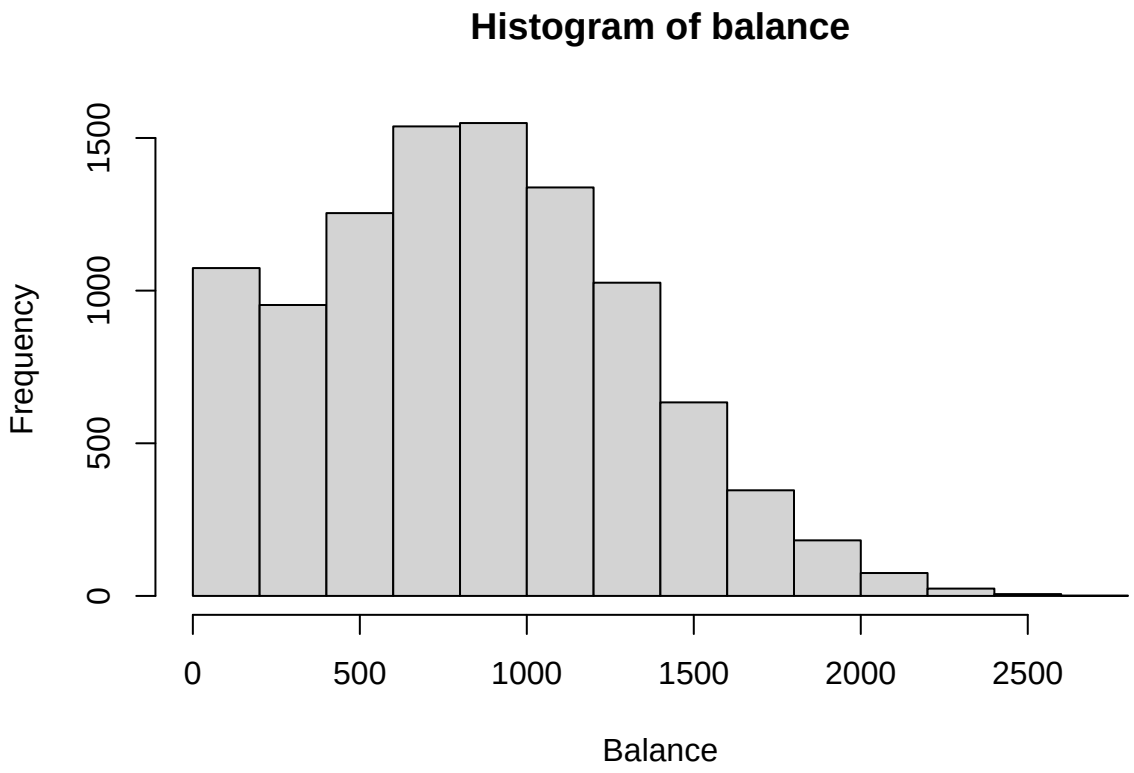
Exercise 14

For the `credit` data set generate the following graphics:

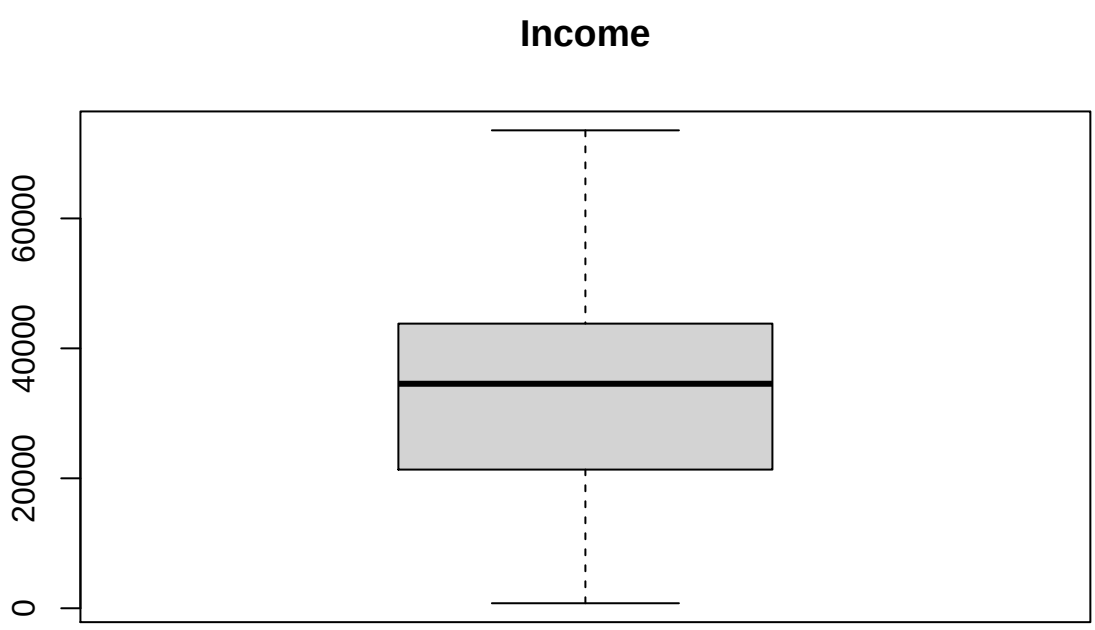
1. A histogram of the `balance` variable
2. A boxplot of the `income` variable
3. A bar plot of the absolute frequencies of the `student` variable
4. A spineplot showing the distribution of the student for each default class (`No`/`Yes`).

Solution 14

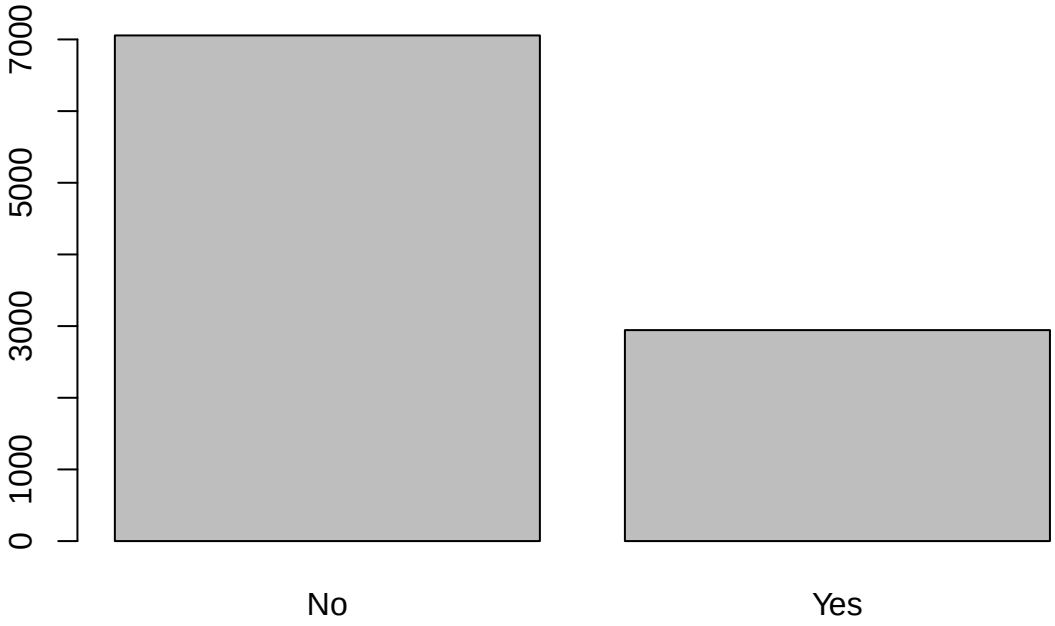
```
# 1
hist(credit$balance, main = "Histogram of balance",
     xlab = "Balance")
```



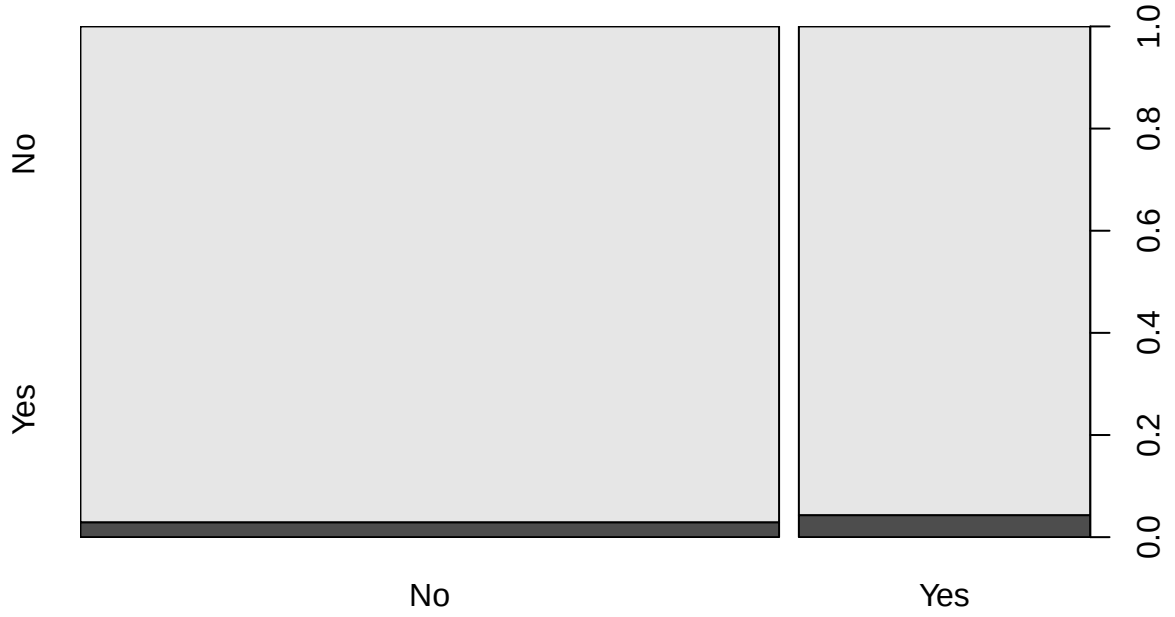

```
# 2
boxplot(credit$income, main = "Income")
```




```
# 3
barplot(table(credit$student))
```




```
# 3
spineplot(table(credit$student, credit$default))
```

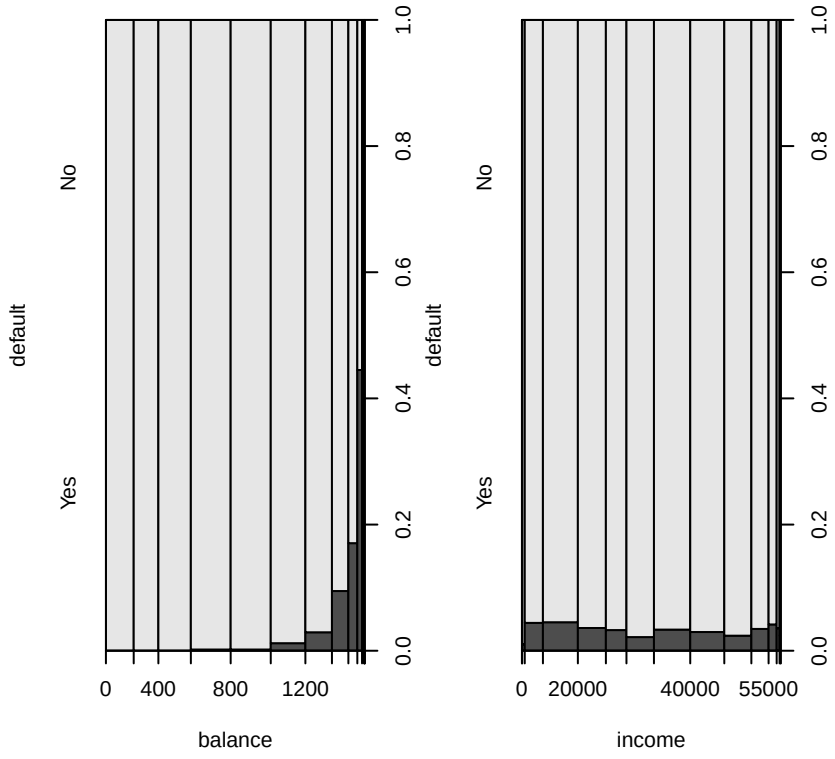


We can investigate the dependence of default on the variables graphically:

```
par(mfrow = c(1, 3))
plot(default ~ balance, data = credit) # spineplot
plot(default ~ income, data = credit) # spineplot
plot(default ~ student, data = credit) # spineplot
```

```
## Warning in spineplot.default(x, y, ...): NAs introduced by coercion
```

```
## Error in hist.default(x = c(NA_real_, NA_real_, NA_real_, NA_real_, NA_real_, : invalid number of 'bi
```



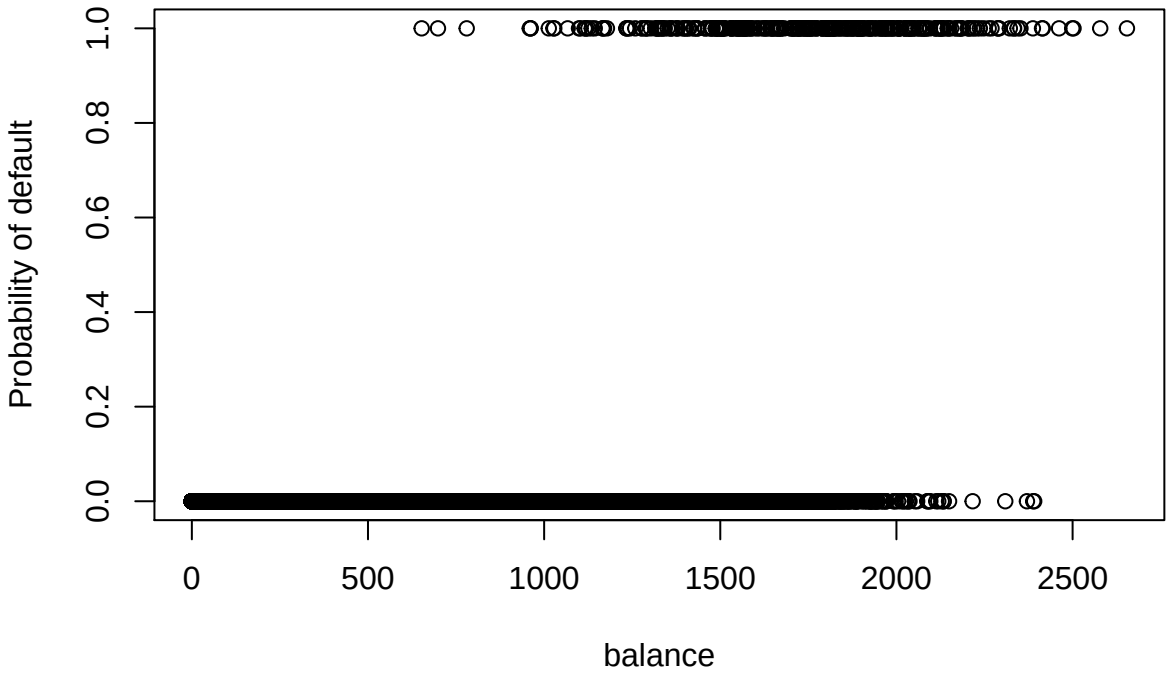
8.3 Simple logistic model

Let us have a look at the probability of default in relation to the balance variable. We can recode the `default` variable as a binary variable with 0 for `No` and 1 for `yes`.

```
credit$defaultBin <- (credit$default == "Yes")
```

Now we can generate a scatterplot of the probability of default and balance:

```
plot(defaultBin ~ balance, data = credit,
     ylab = "Probability of default")
```



We could estimate a linear regression for these data, but this would be inappropriate as the possible values would lie between $-\infty$ and ∞ while we know that the probability should lie between zero and one. A model which can deal with this feature is the logistic regression model, where the probability is given by the logistic function:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}, \qquad \log\left(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)}\right) = \beta_0 + \beta_1 X$$

where $p/(1 - p)$ represents the odds of event $Y = 1$. The interpretation of the coefficients is the following: a one unit increase in X leads to a β_1 log-odds ratio (which implies an odds ratio of e^{β_1}).

8.3.1 Fitting the model

In R the logistic model can be estimated using the `glm()` function:

```
fitLogitBalance <- glm(defaultBin ~ balance, data = credit,
                       family = binomial())
```

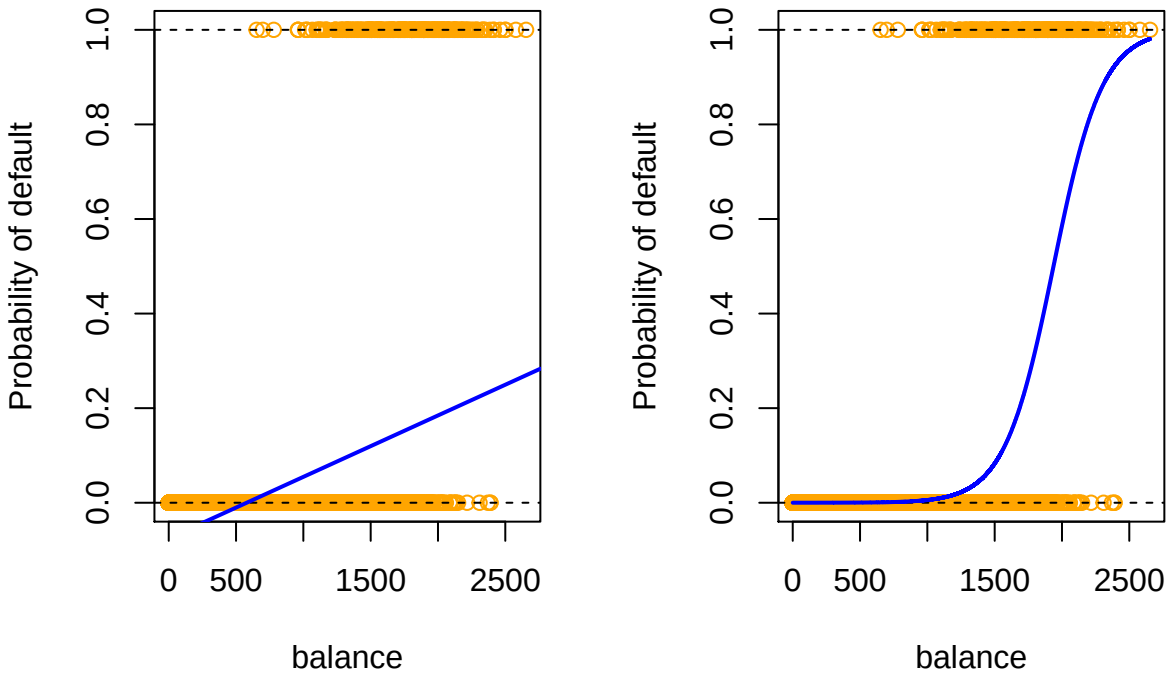
```
fitLogitBalance
```

```
##
## Call:  glm(formula = defaultBin ~ balance, family = binomial(), data = credit)
##
## Coefficients:
## (Intercept)      balance
##  -10.651331     0.005499
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9998 Residual
## Null Deviance:       2921
```

```
## Residual Deviance: 1596  AIC: 1600
```

We see that $\hat{\beta}_1 = 0.0055$; this indicates that an increase in balance is associated with an increase in the probability of default. One monetary unit increase in balance leads to a 0.0055 log-odds ratio (which implies an odds ratio of $e^{0.0055} = 1.0055152$). This means the odds of default increase by $(0.0055 - 1) \times 100 = 0.5515153$ % for every additional balance monetary unit.

After estimating the coefficients using maximum likelihood, we plot the (wrongly) estimated regression line and logistic regression curve:



8.4 Multiple logistic model

In the multiple logistic regression model, the probability is given by the logistic function:

$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_P X_P}}{1 + e^{\beta_0+\beta_1 X_1+\ldots+\beta_P X_P}}, \qquad \log\left(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_P X_P$$

We now include all available variables as independent variables in the model:

```
fitLogitAll <- glm(default ~ balance + income + student, data = credit,
                   family = binomial())
```

```
fitLogitAll
```

```
##
## Call:  glm(formula = default ~ balance + income + student, family = binomial(),
##     data = credit)
```

```
##
## Coefficients:
##   (Intercept)        balance         income     studentYes
## -10.869045196    0.005736505    0.000003033   -0.646775807
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9996 Residual
## Null Deviance:       2921
## Residual Deviance: 1572  AIC: 1580
```

We see that $\hat{\beta}_1 = 0.0057$; this indicates that an increase in balance is associated with an increase in the probability of default; similarly for the `income` variable, but the effect is rather small. The coefficient estimate $\hat{\beta}_3 = -0.6468$ indicates that, for same levels of balance and income, being a student is associated with a decrease in the probability of default.

8.4.1 Summary of the logistic model

```
summary(fitLogitAll)
```

```
##
## Call:
## glm(formula = default ~ balance + income + student, family = binomial(),
##     data = credit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4691  -0.1418  -0.0557  -0.0203   3.7383
##
## Coefficients:
##                   Estimate    Std. Error z value Pr(>|z|)
## (Intercept) -10.869045196   0.492255516 -22.080  < 2e-16 ***
## balance       0.005736505   0.000231895  24.738  < 2e-16 ***
## income        0.000003033   0.000008203   0.370  0.71152
## studentYes   -0.646775807   0.236252529  -2.738  0.00619 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.5  on 9996  degrees of freedom
## AIC: 1579.5
##
## Number of Fisher Scoring iterations: 8
```

The standard output of the `glm()` function contains the call and the summary statistics of the deviance residuals, the estimated coefficients (`Estimate`), the standard error of the coefficients `Std. Error`, the value z-statistic (`z value`) used in the test

$$H_0 : \beta_j = 0$$
$$H_A : \beta_j \neq 0$$

and the corresponding p-value `Pr(>|z|)` by using the result that the test statistic of this hypothesis test follows asymptotically a standard normal distribution under the null hypothesis.

Other goodness-of-fit statistics such as the `Null deviance`, which shows how well the response variable is predicted by a model that includes only the intercept, the `Residual deviance` and the `AIC` are also provided.

The deviance is a measure of lack of fit of a generalized linear model and is calculated as $-2(\log lik(\text{Saturated model}) - \log lik(\text{Model}))$.

One can use the `step()` function for model selection:

Exercise 15

1. For the `credit` data set perform the model selection procedure using the `step()` function.

```
fitLogitStep <- step(fitLogitAll)
```

```
## Start:  AIC=1579.54
## default ~ balance + income + student
##
##           Df Deviance    AIC
## - income   1   1571.7 1577.7
## <none>         1571.5 1579.5
## - student  1   1579.0 1585.0
## - balance  1   2907.5 2913.5
##
## Step:  AIC=1577.68
## default ~ balance + student
##
##           Df Deviance    AIC
## <none>         1571.7 1577.7
## - student  1   1596.5 1600.5
## - balance  1   2908.7 2912.7
```

```
summary(fitLogitStep)
```

```
##
## Call:
## glm(formula = default ~ balance + student, family = binomial(),
##     data = credit)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4578  -0.1422  -0.0559  -0.0203   3.7435
##
## Coefficients:
##                Estimate  Std. Error z value   Pr(>|z|)
## (Intercept) -10.7494959   0.3691914 -29.116    < 2e-16 ***
## balance       0.0057381   0.0002318  24.750    < 2e-16 ***
## studentYes   -0.7148776   0.1475190  -4.846 0.00000126 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2920.6  on 9999  degrees of freedom
## Residual deviance: 1571.7  on 9997  degrees of freedom
## AIC: 1577.7
```

```
##
## Number of Fisher Scoring iterations: 8
```

2. Print the summary of the results. Which model has been chosen?

8.4.2 Predictions

Let us consider a new customer, a student with a credit card balance of 1500 USD and an annual income of 40000 USD. What would be the predicted probability of default?

```
x_new <- data.frame(balance = 1500, income = 40000, student = "Yes")
```

Using the `predict()` function we get:

```
predict(fitLogitStep, newdata = x_new)
```

```
##         1
## -2.857217
```

The predicted number is clearly not a probability. It is instead on the scale of the linear predictors. For more information we can check the help page:

```
?predict.glm
```

For getting the probability of default we need to change the argument `type` to `type = "response"`:

```
predict(fitLogitStep, newdata = x_new, type = "response")
```

```
##          1
## 0.05430945
```

Exercise 16 - analyze your own data set

Continuing the analysis from yesterday, you should focus on the statistical modeling aspect.

1. Be mindful about how you can use the methods discussed today for your own application/dataset.
2. Estimate a linear model and/or a logistic regression model. Interpret the result.

At the end of the workshop, you should have an R script or an R Notebook which contains a complete statistical analysis: sample description, graphics for visualization, a statistical model and a summary of the results.

---

9 Further details for printing, formatting and exporting regression results

9.1 Matrix of coefficients

The coefficient matrix resulting from the `lm` and `glm` objects can be extracted by (`summary()` returns another object):

```
tab_coef <- summary(fitLogitStep)$coef
tab_coef
```

```
##                   Estimate  Std. Error    z value      Pr(>|z|)
## (Intercept) -10.749495878 0.369191361 -29.116326 2.230782e-186
## balance       0.005738104 0.000231847  24.749526 3.136911e-135
## studentYes   -0.714877620 0.147519010  -4.846003  1.259734e-06
```

We could export this table to a `.csv` file:

```
write.csv(tab_coef, file = "tab_coef_fitLogit.csv")
```

9.2 Formatting coefficients

The `stargazer` R package can be used for producing publication ready tables in `text`, `html` or `latex`.

```
if (!require("stargazer")) install.packages("stargazer");
```

```
## Loading required package: stargazer
```

```
##
## Please cite as:
```

```
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
```

```
##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

```
library(stargazer)
```

```
stargazer(fitTV, type = "text")
```

```
##
## ===============================================
##                         Dependent variable:
##                     ---------------------------
##                                Sales
## -----------------------------------------------
## TV                            0.048***
##                               (0.004)
##
## Constant                      7.292***
##                               (0.665)
##
## -----------------------------------------------
## Observations                    94
## R2                              0.623
## Adjusted R2                     0.619
## Residual Std. Error       3.129 (df = 92)
## F Statistic           152.202*** (df = 1; 92)
## ===============================================
## Note:               *p<0.1; **p<0.05; ***p<0.01
```

```
stargazer(list(fitTV, fitAll), type = "text")
```

```
##
## ========================================================================
##                                   Dependent variable:
```

```
##                       ------------------------------------------------
##                                             Sales
##                                (1)                        (2)
## ---------------------------------------------------------------------
## TV                          0.048***                   0.045***
##                              (0.004)                    (0.002)
##
## Radio                                                  0.195***
##                                                         (0.013)
##
## Newspaper                                              -0.009
##                                                         (0.009)
##
## Constant                    7.292***                   3.255***
##                              (0.665)                    (0.440)
##
## ---------------------------------------------------------------------
## Observations                  94                         94
## R2                           0.623                      0.910
## Adjusted R2                  0.619                      0.907
## Residual Std. Error     3.129 (df = 92)            1.546 (df = 90)
## F Statistic         152.202*** (df = 1; 92) 303.522*** (df = 3; 90)
## =====================================================================
## Note:                                    *p<0.1; **p<0.05; ***p<0.01
```

Best way for getting tables ready for MS Word is by saving the html format in a local file (thanks Markus!). Then opening it with the browser and copy pasting it in MS Word.

```
stargazer(list(fitTV, fitAll), type = "html",
          out = "tab_coef_models.html")
```

10 Appendix: R Markdown and R Notebooks

Resources: * http://rmarkdown.rstudio.com/ * https://bookdown.org/yihui/rmarkdown/notebook.html