Introduction to R

Wiebe Pestman, Loes Stukken, Merijn Mestdagh and Lisa Doove

May 8, 2017

The setup of this course is that, in rotation, there will be demonstrations and opportunities to do exercises. At the end there will be an opportunity to analyse your own data. The scheme is as follows:

- Demonstrations
  - Basics of R.
  - Some graphic routines.
  - Data types within R.
  - Importing data.
  - Some programming.
  - Making your own functions.
- Exercises
  - Small exercises referring to the demonstrations
  - Analysis of a real dataset.
- Analysis of your own data.

To install R on your computer, go to the home website of R, `http://www.r-project.org/`, and do the following:

- click `CRAN` in the left bar
- choose a download site
- choose Windows as target operation system (assuming you work on a windows computer)
- click `base`
- choose the most recent version and choose default answers for all questions

After finishing this setup, you should see an "R" icon on your desktop. Clicking on this would start up the standard interface. We recommend, however, to use the RStudio interface. To install RStudio, go to `http://www.RStudio.org/`, and do the following:

- click `Download RStudio`
- click `Download RStudio Desktop`
- click the installer recommended for you system
- download the `.exe` file and run it (choose default answers for all questions)

Please install the latest (free) version of RStudio

The RStudio interface consists of several windows:

- Bottom left: **console window** (also called **command window**). Here you can type simple commands after the $>$ prompt and R will then execute your command. This is the most important window, because this is where R actually does the things you want it to do.
- Top left: **editor window** (also called **script window**). Collections of commands (scripts) can be edited and saved. When you do not get this window, you can open it with `File` $\rightarrow$ `New` $\rightarrow$ `R script`. Just typing a command in the editor window is not enough, it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can select the command you want to run either by standing on the line in which the command resides or by selecting the command, in both cases you can then hit run in the upper right corner of the script file window or press CTRL+ENTER to send it to the command window.
- Top right: **workspace / history window**. In the workspace window you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- Bottom right: **files / plots / packages / help window**. Here you can open files, view plots (also previous plots), install and load packages or use the help function.

Working directory Your working directory is the folder on your computer in which you are currently working. When you ask RStudio to open a certain file, it will look in the working directory for this file, and when you tell RStudio to save a data file or figure, it will save it in the working directory.

Before you start working, please set your working directory to where all your data and script files are or should be stored.

Type in the command window: `setwd("directoryname")`. For example:

```
setwd("M:/IntroductionR/Exercises/")
```

Make sure that the slashes are forward slashes and that you do not forget the apostrophes. R is case sensitive, so carefully distinguish between upper and lower case characters.

Help/documentation Once R/RStudio is installed, there is a comprehensive built-in help system. At the program's command prompt you can use any of the following:

```
help.start()     # general help
help(plot)       # help about function plot
?plot            # same thing
apropos("plot") # list all functions containing string plot
example(plot)    # show an example of function plot
# To search for plot in help manuals and archived mailing lists:
RSiteSearch("plot")
```

The following links can also be very useful:

- `http://cran.r-project.org/doc/manuals/R-intro.pdf` A full manual.
- `http://www.statmethods.net/` Also called Quick-R. Gives very productive direct help. Also for users coming from other programming languages.
- `http://stackoverflow.com/` An open question and answer site for programmers. Also for users coming from other programming languages.
- Just using Google (type e.g. "R plot" in the search field) can also be very productive.

Some good practices for programming in R

- Store your commands in files, the so-called scripts.
- Comment your code using `#`.
- Strive for clarity and simplicity. For example, think about naming of objects (self-explainable but short).
- Dedicate one directory per project.

Contents

1 Some graphic routines

In this section you will learn:

- Commands to make plots in R.
- The difference between high-level and low-level commands.

Exercises

A most important graphical utility is the function `plot()`.

Exercise 1.

a) Create variables `x`, `y` in the R-workspace like this

```
x <- seq(from=1,to=10,by=0.1)
y <- x^2 - 10 * x + 12
```

b) now run the following command

```
plot(x,y)
```

Within the function `plot()` you can adjust several features of the plot. You can, for example, change the color, and tell R that it should draw a line instead of dots:

```
plot(x,y,type="l",col="blue")
```

c) Run the command `?plot` to see all the things you can adjust.

Exercise 2. This exercise introduces you to the possibilities of making histograms in R.

a) Define a sequence `x` consisting of 50 randomly chosen normally distributed numbers, by giving the command

```
s <- rnorm(50,10,2)
```

Then type `x` and press the key ENTER.

b) Now make a histogram by giving the command `hist(x)`.

– Next, pass the command

```
hist(s,col="blue")
```

In the same was as with plot, you can change the characteristics of the histogram. For example:

```
hist(s,col="blue",border="red")
```

and check the difference with the foregoing

c) Next, pass the command

```
hist(s,breaks=10,col="red")
```

and watch the output in the graphic window.

d) Define a sequence `b` of breakpoints by giving the command

```
b <- c(3.5,5.5,7.5,9.5,11.5,13.5,15.5,17.5)
```

e) Now run the command

```
hist(s,breaks=b,col="red")
```

and see the effect in the graphic window.

f) Ask R for help about histograms by giving the command `?hist`.

Exercise 3. This exercise teaches you how to make boxplots in R.

a) Pass the command

```
u <- rnorm(50,10,2)
```

Then type `u` and press ENTER.

b) Determine the median of the sequence `u` by giving the command `median(u)`.

c) Determine all quartiles by giving the command `quantile(u)`.

d) Give the command `boxplot(u)` and compare the graphic window to your results in b) and c).

e) To learn some more on making boxplots in R, just type `?boxplot` and press the key ENTER. For example, try to get colors in your boxplots.

In general there are two important types of plotting commands: *high level* and *low level*.
High level commands are for example `plot()`, `hist()`, `boxplot()`. They set up coordinate axes and the type of graphics.
Low level commands are for example `points()`, `lines()`, `text()`, `legend()`. They are used to add stuff once everything is set up by a high level command.

Exercise 4. The aim of this exercise is to show you the difference between high and low level commands.

a) Create a new variable `z` in the R-workspace like this

```
z <- 10*sin(x)
```

b) Run the two high level commands

```
plot(x,y,type="l",col="blue")
plot(x,z,type="l",col="red")
```

c) Run the high level command

```
plot(x,y,type="l",col="blue")
```

Then pass the low level commands

```
lines(x,z,col="red")
legend(8,-9, legend=c("Men","Women"),
                    col=c("red","blue"),pch=c(16,16))
```

2 Some basic statistical tests

In this section you will learn:

- How to run t-tests (all flavours) in R.

Exercises

Exercise 1. This exercise will teach you how to conduct t-tests (in all flavours) in R.

a) Generate a sequence `holland` of 30 numbers, drawn from a normally distributed population with mean 175 and standard deviation 10. Do this by giving the command

```
holland <- rnorm(30,175,10)
```

Think of the sequence `holland` as if it were a sample of 30 height measurements in the Dutch male population. Then pass the command

```
t.test(holland,mu=178)
```

and explain the output R returns. Should the null hypothesis $H_0 : \mu = 178$ be rejected?

b) Generate a sequence `france` of 50 numbers by giving the command

```
france <- rnorm(50,170,10)
```

c) Now pass the command

```
t.test(holland,france)
```

and explain the output R returns. Which null hypothesis is tested here? Is it rejected? Also run the command

```
boxplot(holland,france)
```

This can be considered the graphical counterpart of the t-test you just conducted.

d) Generate a sequence `male` and a sequence `female` by giving the two commands

```
male <- rnorm(50,176,10)
```

```
female <- rnorm(50,174,10)
```

Assume that the sequences are paired in the sense that they present height measurements in 50 man-woman couples. Give the command

```
t.test(male,female,paired=TRUE)
```

and explain the output R returns. Which null hypothesis is tested here? Is it rejected?

3 Dealing with datasets

Basically in R there are the functions `load()` and `load()` to save and load data in R-native format. Besides this there are extensive possibilities to import data that is stored in 'foreign' formats, such as Excel formats or SPSS formats. Importing data can be very time-consuming. Through the latest versions of RStudio this process has been simplified considerably. In this section you will learn:

- How to save, load, import and export datasets in R.
- Some basic commands in order to get some information from your dataset and workspace.

Exercises

Reading in data

Exercise 1. In this section we would like to read in the dataset `bloodtype.txt`. The first thing to do is to find out where this dataset resides, that is to say, to find the computer path to this dataset. On a windows machine this is usually something of the form

```
X:\folder\subfolder\subsubfolder\bloodtype.txt
```

where `X` stands for the drive that contains the file in question. Once you know this, there are two ways to get the data into the R-workspace. The first way is through

```
Blood <- read.table("X:/folder/subfolder/bloodtype.txt",
                                                header=TRUE,sep="")
```

The second way is to set the home directory to the folder `X:/folder/subfolder` and then use `read.table()` in a simpler form:

```
setwd("X:/folder/subfolder")
Blood <- read.table("bloodtype.txt",header=TRUE,sep="")
```

In the above `header=TRUE` tells R to read in the names of the columns (the headers) in the txt-file. Moreover, `sep=";"` tells R which separator is used in the datafile. When you have a txt-file (like here) it may be a space and then you should use `sep=" "` with a csv-file it is usually ";" The data is stored in an R-object called `Blood`. Be aware that this object is *not* the file `bloodtype.txt`.

Now, read in the data using one of the two options above.

Once the data is on board you may try out the following commands to get some basic information about the object `Blood` and the stuff it contains.

```
colnames(Blood)
rownames(Blood)
dim(Blood)
head(Blood)
str(Blood)
summary(Blood)
```

a) Now run the command `ls()`. What do you get? The command `ls()` is a fast way to see how many and which objects are active in your workspace and are thus available to work with. If you have read in the bloodtype.txt file in the way described above, the object `Blood` should be one of them.

b) Now run the command

```
remove(list=ls())
```

Your workspace is now empty. Pass the command `ls()` to check this.

Exercise 2. R has a native file format. In this exercise you will learn how to save objects in files of this particular format.

a) Define a sequence `x`, consisting of 50 normally distributed numbers, by giving the command

```
x <- rnorm(50,10,2)
```

b) Define a second object `y` in your workspace by giving the command

```
y <- runif(200,2,10)
```

c) Now by running the command below you can save the variables you just created

```
save(x,y,file="X:/john.RData")
```

where `X` stands for the drive letter of your USB-key.

d) Clean your workspace. Check this by means of the command `ls()`. Then pass the command

```
load(file="X:/john.RData")
```

and check your workspace again.

e) Try to open your file `john.RData` in WordPad.

f) Clean up your workspace and create a new variable `x` by the command

```
x <- 1:12
```

Then run the command

```
write(x,file="X:/peter.txt")
```

This command saves your variable `x` in the form of a text file. Try to open it in WordPad. Note that the file `peter.txt` could be imported for example in SPSS.

g) Now, try out the next command

```
write(x,file="X:/peter.txt",ncolumn=3)
```

and check the result in WordPad.

h) Just play around a bit to become familiar to the commands you learnt in this exercise. It is also wise to make a summary of all the commands you learnt up to now.

4 Making sequences

In this section you will learn how to create sequences in the R-workspace. In R-language, sequences are often called *vectors*. The most basic way to create sequences is through the function `c()`. For example, when running the command

```
jan <- c(1,5,12,1578,8625,45)
```

an R-object `jan` is created that contains the numbers $1, 5, 12, 1578, 8625, 45$ in this specific order. Similarly, through the command

```
piet <- c("cat","dog","pig","spider")
```

an R-object `piet` is created that contains the words *cat, dog, pig, spider* in precisely in this order.

A sequence (or vector) of subsequent natural numbers can be put in like this

```
klaas <- 3:12
```

The object `klaas` then contains the numbers $3, 4, 5, \ldots, 10, 11, 12$ in this order.

Generally, sequences with a constant step-size can be created through the function `seq()`. For example, the vector `klaas` that was created above could also be created like this:

```
klaas <- seq(from=3,to=12,by=1)
```

If you want a different step-size, do something like this:

```
seq(from=1,to=600,by=2)
```

This defines the sequence $1, 3, 5, 7, \ldots, 597, 599$. Type `?seq()` to see the possibilities this function offers and to see (at the end) some examples.

If you want to put in a sequence that contains structural repetitions, then the function `rep()` is often useful. To learn how this function works, run the following commands and observe the corresponding output.

```
rep(c(0,1),times=2)
rep(c(0,1),each=2)
rep(c(0,1),c(4,3))
rep(c("cat","dog","pig","spider"),c(1,3,2,1))
```

Type `?rep()` to learn more about this function. Examples are always at the end.

5 Matrices and dataframes

In this section you will learn:

- How to navigate in matrices and datasets and how to modify them.
- How to equip rows and columns with names.

Exercises

Exercise 1. This exercise will teach you how to make matrices, how to modify them and how to navigate in them.

a) Start the program R and pass the command `x <- 1:12`. Then run the commands

```
a <- matrix(x,ncol=4)
b <- matrix(x,ncol=4,byrow=TRUE)
c <- matrix(x,nrow=6)
```

Print the objects `a`, `b` and `c` on your screen and compare them mutually.

b) Now pass the command `d <- a[3,2]`. Compare `d` to `a`.

c) Define an object `e` by `e <- a[,3]` and compare `e` to `a`.

d) Define an object `f` by `f <- a[2,]` and compare `f` to `a`.

e) Give the command

```
g <- a[,-3]
```

and compare `g` to `a`.

f) Give the command

```
h <- a[-2,]
```

and compare `h` to `a`.

g) Give the command `a[2,3] <- 99` and observe the effect on `a`.

Exercise 2. We can navigate in a similar way in datasets (a dataset can be seen as a large matrix).

a) For example after reading in the `bloodtype.txt` dataset you can access how many people have blood type A (the first row in the dataset) by passing the command (if the dataset, is currently not in you workspace, then read the data in again using the commands explained in §3.

```
Blood[1,]
```

b) To assess the bloodtypes for all the men (the first column in the dataset) one can type in:

```
Blood[,1]
```

An alternative way to bring this about is by passing the command:

```
Blood$male
```

The following command does not work (at least not straight-away):

```
male
```

This is so because, by default, R does not search data through the headers of an object. You can make R do this by running the command

```
attach(Blood)
```

Once you have done this, R will recognize the header `male` and it will show its content when running this header as a command. You may then, for example, also run a command like this

```
table(male)
```

So in order to use the variable name to obtain the scores on that variable you should attach the dataset!

c) To obtain the number of men that have bloodtype A you may run one of the commands:

```
Blood[1,1]
Blood$male[1]
male[1]
```

The last of these commands only works if you have attached the dataset! From now on we assume that you have attached the object `Blood` so that we can use its headers as if they were variables in the workspace. Also note that when you refer to the header/name of the column instead of to the dataset, you refer to a vector instead of to a matrix. Hence, since a vector is one-dimensional, you only have to define the position of blood type A.

6 Boolean operators

In this section you will learn:

- How to use boolean operators to select datacells with a particular value on a variable.
- How to combine these boolean operators to select data with multiple restrictions.

Exercises

For these exercises you need to read in the datasets `bloodtype.txt` and `perulung.txt`. You can use the commands you learned in §3.

Exercise 1. In some cases you want to access data-cells that have a particular value on another variable. You can then use so-called boolean operators stored in logical vectors. As an example, one of the columns in `bloodtype.txt` dataset is the variable `female`. By running the command

```
female==25
```

you get a sequence (a vector) of length 4 (=length of female), exclusively containing the Boolean expressions `TRUE` or `FALSE`. Similarly one could run the commands

```
female!=25
female<30
female<=30
```

Exercise 2. If you wish to know what the age is for all girls in the `perulung.txt` dataset, then pass a command of type

```
age[gender=="1"]
```

Only the age of the people for which `gender=="1"` has value `TRUE` will be shown then.

Exercise 3. It is possible to combine logical vectors. For example, if you want to know the age of the boys larger than 120 cm, do the following:

```
age[gender=="0" & height>120]
```

Only the age of people for which `gender="0"` *and* `height>120` will be shown then. Similarly

```
age[gender=="0" | height>120]
```

will show the ages of people for which `gender="0"` *or* `height>120` is true.

7 Built-in Functions

In this section you will learn:

- How to store the result of an arithmetic operation in a new variabele.
- How to use built-in functions for easy calculation

Exercises

Exercise 1. The variables in `Blood` and any other dataset may be subjected to arithmetic operations. For example, if we want to calculate the total number of people that have a specific bloodtype, we can use the following command:

```
male+female
```

Of course these numbers may be stored in a new variable, say a variable `Total`, like this:

```
Total <- male+female
```

By running the command `ls()` or `objects()` you may check that the variable `Total` is now a member of the workspace.

Exercise 2. R has a variety of built-in functions, that can be applied to all kinds of variables. Here are some of them:

```
sum() #calculates sum
mean() #calculates mean
prod() #calculates product
var() #calculates variance
sd()  #calculates standard deviation
```

Thus, for example, the command `sum(female)` will return the total number of women in the bloodtype dataset. As another example, the command

```
TotalA <- sum(Blood[1,1:2])
```

will store the total number of people that have bloodtype A in a variable `TotalA`. In the above the expression 1:2 indicates that you want to calculate the sum over the columns 1 to 2 (in this case across male and female).

The following functions are often handy tools:

```
length() #can be used to calculate the length of a variable
min() #determines minimum score of a variable
max() #determines maximum score of a variable
range() #determines the minimum and maximum score of a variable
colMeans() # calculates the mean for the columns
rowMeans() # calculates the mean for the rows
colSums() #calculates the sum for the columns
rowSums() #calculates the sum for the rows
```

8 A little programming

In this section you will learn:

- How to exploit the idea of so-called `for` loops.
- How to use `while` loops.

Exercises

Exercise 1.

a) Define an object `a` by giving the command $a < -3$. Then pass the command `print(a)`. This command 'prints' the object `a` to the screen. The same could be established by typing `a` and press ENTER. In programming the command 'print' is often to be used, as we will see hereafter.

b) Here is your first program in R; just type the next command and then press ENTER:

```
for (k in 1:8) {print(k)}
```

c) Next try the command

```
for (k in 1:8) {print(k^2)}
```

d) Define the object `x` as `x <- 1:12` and the object (matrix) `a` by

```
a <- matrix(x,ncol=4)
```

Give the command `sum(a[,2])` and explain the value returned by R.

e) Give the command

```
for (k in 1:4) {sum(a[,k])}
```

What did you expect to see? Why is this not visible on the screen?

f) Next pass the command

```
for (k in 1:4) {print(sum(a[,k]))}
```

Explain what you see on the screen.

g) Give the command

```
for (k in c(2,1,4)) {print(sum(a[,k]))}
```

Explain the values R returns.

Exercise 2. Clean up your workspace and create a new variable `x` by the command

```
x <- 1:64
```

Then pass the command

```
a <- matrix(x,ncol=8)
```

a) Give the next command:

```
for (k in 1:8) {mean(a[k,])}
```

Next pass the command

```
for (k in 1:8) {print(mean(a[k,]))}
```

b) Define an object `m` by giving the command `m <- rep(0,8)`. Print `m` on your screen. Next, pass the command

```
for (k in 1:8) {m[k]=mean(a[k,])}
```

Print `m` again on your screen.

c) Give the command

```
for (k in 1:8) {m[k]=a[k,k]}
```

and print `m` on your screen.

d) Give the command

```
for (k in 1:8) {print(t.test(a[,k],mu=12))}
```

Why was the additional command 'print' necessary here?

9 ANOVA and linear regression

In this section you will learn:

- How to run an ANOVA in R.
- How to run a regression analysis in R.

Exercises

Exercise 1. The plain text file `cars.txt` contains data that will be needed in this exercise. Read the data in in R by running the command:

```
cars <- read.table(file="cars.txt",header=TRUE)
```

The dataset `cars.txt` is a real dataset about cars. The first column presents the weight of the cars in kilograms. The second the distance (in kilometers) one can drive on one litre of fuel. The third column presents a self-explaining classification of the cars.

a) Run an ANOVA by giving the command

```
myfirstanova <- aov(km ~ factor(class))
```

Then pass the command

```
summary(myfirstanova)
```

Explain the output R returns.

b) Ask R for the residuals by giving the command

```
r <- residuals(myfirstanova)
```

Now type `r` and press ENTER.

Exercise 2. This exercise will teach you how to run a regression analysis in R.

a) If it is not anymore in your workspace, get again the dataset `cars.txt` in R. Give the command

```
plot(weight,km)
```

and observe the results in the graphic window.

b) Give the command

```
john <- lm(km ~ weight)
```

Then pass the command

```
summary(john)
```

and explain the output R returns.

c) Probably you also want to have the regression line in the plot you made in a). Give the command

```
abline(coefficients(john))
```

to bring this about. Type `help(abline)` to get more information about the function `abline()`.

d) Ask R for the residuals of the regression analysis by giving the command

```
r <- residuals(john)
```

10 Making your own functions

In R virtually all elementary mathematical functions are available by default. Nevertheless you may very well encounter situations where you need a user-defined function. Suppose, for example, that you want to have a function

$$x \mapsto \frac{8}{1+x^2}\, e^{-2x}$$

available in R. To make such a function in your workspace, first make a decision how to name the function. If you decide to name it, say, `john` then pass the command

```
john <- function(x) {(8/(1+x^2))*exp(-2*x)}
```

By this action the function `john` is available in your workspace. As an example, to get the value of `john` for `x` $= 1.2$, pass the command

```
john(1.2)
```

The value of the function `john` in `x` $= 1.2$ is then printed to the screen. The function `john` allows for vectorial input. As an example, define a vector `myx` like this

```
myx <- c(1.2,1.7,2.2,2.5,2.9)
```

Now pass the command

```
john(myx)
```

This will print a set of function values to the screen. These function values, of course, can also be stored in, say, an object `y` :

```
y <- john(myx)
```

A plot of the function values could then be obtained like this:

```
plot(myx,y)
```

Functions can be saved for usage in subsequent sessions. Saving it in the root of drive X can be done like this

```
save(file="X:/myfun.RData",john)
```

By this action the function `john` is saved in a file named `myfun.RData`. You can save more than just one function in a file. If there were two more functions in your workspace, say `peter` and `nicholas`, then the three functions could be saved in one file as follows

```
save(file="X:/myfun.RData",john,peter,nicholas)
```

By this the two functions are saved in a file named `myfun.RData` in the root of drive `X`. In a subsequent session you can load the two functions in your workspace like this

```
load("X:/myfun.RData")
```

The functions `john`, `peter`and `nicholas` are then again at your disposal.

The code (or syntax) that defines functions can also be stored in plain text files. For example, by means of a text editor one could save the code

```
john <- function(x) {(8/(1+x^2))*exp(-2*x)}
peter <- function(x) {0.5+log(x/(1+x))}
nicholas <- function(x) {1+sin(3*x)}
```

in a text file named `myfun.txt`. On a Windows machine this can be done, for example, in the program 'WordPad'. It can also be done in the program 'MS-Word' but then you must save the result as a plain text file, *not* as a file in MS document format! To read in your function from the text file `myfun.txt` to your workspace, pass the command

```
source(file="X:/path/to/myfun.txt")
```

Here `X` is the drive on which the file `myfun.txt` resides, followed by the complete path to this file.

11 Closing RStudio

When you close R and RStudio, you will be asked whether you want to save your workspace. If you do so the contents of the workspace will be saved in a file with extension `.Rdata`. There are very few situations in which saving the whole workspace is a good idea. However, saving parts of the workspace can be useful. To save your workspace, click "Workspace", then "Save Workspace As". Choose the folder in which you want to store the file, then click save. Saving part of your commands (code) is extremely useful. This can be done in an .R file. To save your script-file, click "File", then "save as". Choose the folder in which you want to store the file, then click save.

12 Analysis of a simple dataset in R

In this section you are asked to study the dataset `Beers.cvs`, thereby using some of the things you learned in the previous sections.

So let's talk beers! The beer dataset includes data for 4 groups of people (young men, young women, old men and old women) that did a bunch of tasks including:

- Tasting 17 beers and trying to guess which one they were drinking.
- Filling out a questionnaire in which they were asked to indicate:
  - how many alcoholic beverages they consume each week
  - the number of beers they consume each week
  - the different sorts of beers they drink each week
  - the different sorts of beers they drink each year
  - the level of beer knowledge they thought they had on a scale ranging from 1 (poor) to 5 (expert)

The first thing to do is to find out where this dataset resides, that is to say, to find the computer path to this dataset. Start by reading in the dataset as an object `Beerdata`, and attach the object `Beerdata` to the workspace by `attach(Beerdata)`.

By now you should be able to do the following exercises:

Exercise 1. How old is subject 33? (use two different ways for finding your answer)

Exercise 2. Can you find the number of beers each subject drinks each week? (use two different ways for finding you answer)

Exercise 3. How many different sorts of beers do women that have a `Beerknowledge` score over 2 drink each year?

Exercise 4. Run the command `max(NumbBeerConsumpWeek)` and interpret the output it returns.

Exercise 5. Run the command `colMeans(Beerdata[,5:9])` and interpret the output it returns.

Exercise 6. Calculate the mean, variance and standard deviation of the variables `NumbAlcConsumpWeek` and `Age`.

Exercise 7. Use an easy way to calculate the variable `NameScore` (=the total score of each subject on the beer naming task).

Exercise 8. Calculate the mean, variance and standard deviation of the variable `Age` for all women.

Exercise 9. Calculate the mean, variance and standard deviation of the variable `NumbAlcConsumpweek` for all women older than 30.

Exercise 10. How many subjects have a score higher than 3 on the variable `Beerknowledge`?

Exercise 11. Make a function that calculates the percentage correct in naming for a specific beer.

Exercise 12. Use a for loop to determine the number of subjects that have a score higher than 3 on the variable `Beerknowledge`.

Exercise 13. Run the following code and observe what it does:

```
Groupnew <- c()
for (i in 1:length(Gender))
{
  if (Gender[i]=="m")
    {Groupnew[i]=1}
  else
    {Groupnew[i]=2}
}
```

Why is it necessary to start the code with `Groupnew <- c()`?

Exercise 14. In this exercise we play a bit with `plot()` by just running it in several ways on the variables `NumbDifferentBeerYear` and `NameScore`

a) Run the command

```
plot(NumbDifferentBeerYear,NameScore)
```

b) Change the color of the plot to green (run `?plot`, if you need help)

c) Now give the plot a name and label the axes

d) Change the plotting symbol (use the internet if you need help)

e) Change the limits of the x and y axes

Exercise 15. Make a histogram for the variable `NumbAlcConsumpWeek`. Change the colors of the bins and the colors of the borders Add labels and a title to the histogram. For help pass the command `?hist()`.

Exercise 16. Make a boxplot for the variable NameScore

Exercise 17. Now make a boxplot for the variable NameScore but for each group (gender) separately

Exercise 18. Calculate a correlation between `NameScore` and `Beerknowledge` and test whether it is greater than 0. (use the R built-in help system or the internet if you need help)

Exercise 19. Test whether there is a significant difference in naming score between men and women.

Exercise 20. Is there a significant difference in `Beerknowledge` between men and women?

Exercise 21. Are people with a high score on the variable `Beerknowledge` better in naming beers than people with a low score on the variable `Beerknowledge` Hint: Split the subjects into two groups by means of the median or mean.

Exercise 22. Fit the following model using the `lm()` function to do regression analysis.

$$\texttt{NameScore} = \beta_0 + \beta_1 * \texttt{Beerknowledge} + \beta_2 * \texttt{NumbDifferentBeerYear},$$

Exercise 23. Use an ANOVA to test whether there is a difference in naming score between the 4 groups: young men, old men, young women, old women. Look at the degrees of freedom, something went wrong! What?

This is important! The grouping variable `Group` has values 1,2,3,4 but these numbers do not actually represent a number; they represent a *category*. For this reason R should be told that it should consider these numbers as categories. Use the function `factor()` to solve this problem:

```
Group.factor <- factor(Group)
```

Now redo the analysis.

Exercise 24. Test the main effect of `Gender`, the main effect of `Beerknowledge` and their interaction.

Answers to some of the exercises in §12

Exercise 12.1 First find out in which column `Age` resides. For example, type `colnames(Beerdata)` to learn that it is the third column. Then type `Beerdata[33,3]`. Alternatively, provided the object `Beerdata` is attached to the workspace, you may simply pass the command `Age[33]`. In both cases you will arrive at an age of 24 year.

Exercise 12.2 Just type `NumbBeerConsumpWeek` or `print(NumbBeerConsumpWeek)`. Alternatively, use a command of type `Beerdata[,k]`, where `k` is the column number belonging to the variable `NumbBeerConsumpWeek`.

Exercise 12.3 Type

```
NumbDifferentBeerYear[Gender=="f" & Beerknowledge > 2].
```

The result is

8 10 5 5 10 100 3 5 4 12 6 5 20 1 3 4 4 2

Exercise 12.4 The result is 80. Type `ID[NumbBeerConsumpWeek==80]` to find the ID of the person(s) drinking this amount weekly.

Exercise 12.5 The command `colMeans(Beerdata[,5:9])` returns the mean of the following variables:
`NumbAlcConsumpWeek`
`NumbBeerConsumpWeek`
`NumbDifferentBeerWeek`
`NumbDifferentBeerYear`
`Beerknowledge`

Exercise 12.6 For the variable `NumbAlcConsumpWeek`, pass respectively the following commands

```
mean(NumbAlcConsumpWeek)
var(NumbAlcConsumpWeek)
sd(NumbAlcConsumpWeek)
```

This will lead to the answers 11.73262, 132.5373 and 11.51248. As an additional exercise, check that `sqrt(132.5373)` leads to the standard deviation 11.51248. For the variable `Age` the answers are 28.68874, 147.4111 and 12.1413.

Exercise 12.7 For example, the following will do the job

```
NameScore <- rowSums(Beerdata[,10:27])
```

Exercise 12.8 Pass the following commands

```
mean(Age[Gender=="f"])
var(Age[Gender=="f"])
sd(Age[Gender=="f"])
```

This will result in the numbers 29.17557, 160.0997, 12.65305 respectively.

Exercise 12.9 Run

```
mean(NumbAlcConsumpWeek[Gender=="f" & Age>30])
var(NumbAlcConsumpWeek[Gender=="f" & Age>30])
sd(NumbAlcConsumpWeek[Gender=="f" & Age>30])
```

This will return the numbers 6.484848, 27.1482, 5.210394.

Exercise 12.10 For example, type

```
sum(Beerknowledge>3)
```

Alternatively you could pass the command

```
table(Beerknowledge)
```

It can be read off that there are 37 subjects that have a score higher than 3 on the variable `Beerknowledge`.

Exercise 12.11 The following code will do the job

```
percentagecorrect <- function(x){
p <- (sum(x)/302)*100
return(p)
}
```

For example, when running `percentagecorrect(Belle.Vue.Kriek)` the function returns the percentage of 23.50993%.

Exercise 12.12 For example the following loop will do the job

```
S <- 0
 for (k in 1:302)
 {
   if (Beerknowledge[k]>3){S<-S+1}
 }
```

Then print the contents of `S` to the screen. Without a `for`-loop, the command

```
sum(Beerknowledge>3)
```

does the same job.

Exercise 12.13 The loop defines a new variable `Groupnew`. This new variable is actually the variable `Gender` with the difference that male subjects are coded by the string `1` and female subjects by `2`.

Exercise 12.14 b) Run `plot(NumbDifferentBeerYear,NameScore, col="green")`

c) For example, run

```
plot(NumbDifferentBeerYear,NameScore, col="green",
         main="Beerplot", ylab="Total score",
            xlab="Number of different beers per year")
```

d) For example, run

```
plot(NumbDifferentBeerYear,NameScore, col="green",
       main="Beerplot", ylab="Total score",
         xlab="Number of different beers per year", pch=20)
```

e) For example, run

```
plot(NumbDifferentBeerYear,NameScore, col="green",
       main="Beerplot", ylab="Total score",
        xlab="Number of different beers per year", pch=20,
         ylim=c(0,10), xlim=c(0,120))
```

Exercise 12.15 For example, run

```
hist(NumbAlcConsumpWeek, col="green", border="red",
             main=Histogram of alcoholic consumption",
                    xlab="Alcoholic consumption per week")
```

Exercise 12.16 Run `boxplot(NameScore)`

Exercise 12.17 Run `boxplot(NameScore~Gender)`

Exercise 12.18 For example, type `cor.test(NameScore, Beerknowledge)`

Exercise 12.19 For example, type `t.test(NameScore~Gender)`

Exercise 12.20 Run a t-test as follows:

```
t.test(Beerknowledge~Gender)
```

The output will be like this:

```
Welch Two Sample t-test

data:  Beerknowledge by Gender
t = -12.4396, df = 292.322, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.3711293 -0.9965329
sample estimates:
mean in group f mean in group m
       1.664122        2.847953
```

The p-value is less than 2.2×10^{-16}. Hence there is a very significant difference between women and men as to `Beerknowledge`. To visualize this, run for example:

```
boxplot(Beerknowledge~Gender)
```

Exercise 12.21 First split the subjects into two groups: a group that has a score above the mean and a group that has a score below the mean. A quick way to bring this about is as follows

```
Beerknowledge0 <- (Beerknowledge > mean(Beerknowledge))
```

You may also use a `for`-loop to get `Beerknowledge0`. The variable `Beerknowledge0` is a dichotomized version of `Beerknowledge`. It may serve the purpose of a grouping variable in a t-test. The following code will do the job

```
t.test(NameScore~Beerknowledge0)
```

Exercise 12.22 Run

```
lm(NameScore~Beerknowledge+NumbDifferentBeerYear)
```

Exercise 12.23 First define a new variable `Group`.

```
Group <- c()
for (i in 1:length(Gender))
{if(Gender[i]=="m"&Age[i]<=mean(Age)) {Group[i]=1}
 if(Gender[i]=="m"&Age[i]>mean(Age)) {Group[i]=2}
 if(Gender[i]=="f"&Age[i]<=mean(Age)) {Group[i]=3}
 if(Gender[i]=="f"&Age[i]>mean(Age)) {Group[i]=4}
}
```

Run the analysis by `aov(NameScore~Group)`.

Next run

```
Group.factor <- factor(Group)
aov(NameScore~Group.factor)
```

Exercise 12.24 Two equivalent ways to specify the model with interaction are:

```
fit <- lm(NameScore~Gender*Beerknowledge)
fit <- lm(NameScore~Gender+Beerknowledge+Gender:Beerknowledge)
```

Next run `summary(fit)`.

Bibliography

- Dalgaard, P.; Introductory Statistics with R (second edition). Springer Verlag, Berlin (2008).
- Larget, B.; R for Introductory Statistics. (free downloadable from: http://www.stat.wisc.edu/ larget /stat371/rhelp.pdf)
- Torfs, P. & Brauer, C.; A (very) short introduction to R. (free downloadable from: http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf)
- Venables, W. N. & Ripley, B. D.; Modern applied statistics with S-plus. (Springer Verlag, Berlin, 1997)
- Venables, W. N.; An Introduction to R. (free downloadable from: http://cran.r-project.org/doc /manuals/R-intro.pdf)