Rbeginner: R and RStudio – a first tour

TMA4268 Statistical Learning V2019. Module 1: INTRODUCTION TO STATISTICAL LEARNING

Mette Langaas and Martina Hall, Department of Mathematical Sciences, NTNU

week 2, 2019

Contents

Part A: Using RStudio - what are the different windows? **1**

Part B: Trying out R-commands **2**

Part C: Importing and exporting **3**
- Executing the commands in an R-file with `source` . . . 3
- Printing to and reading from a csv-file . . . 3
- Printing to and reading from other file formats . . . 4
- Exporting plots . . . 4

Part D: Packages - and functions **5**

Part E: More on vectors and matrices **6**

Part F: Plotting **8**

Part G: Writing a simple function **10**

Part H: Lists and data frames **10**
- Lists . . . 10
- Data frames . . . 11

What is next? **11**

(Latest changes: 31.12.18: first version for 2019)

R is a free software environment for statistical computing and graphics. It runs on a wide variety of UNIX platforms, Windows and MacOS. R can be downloaded from http://www.r-project.org/.

We recommend that you run R using the RStudio IDE (integrated development environment). RStudio can be downloaded from http://rstudio.org/.
Notice: you need to download both R and RStudio.

If you need help on installing R and RStudio on you laptop computer, contact orakel@ntnu.no. If you want to work at Nullrommet or Banachrommet at Matteland, R and RStudio is already installed for you.

Part A: Using RStudio - what are the different windows?

Start RStudio. Then you (probably) have the following four windows.

- **Source** (aka script window) - upper left window: where you write your code and keep track of your work.

- **Console** - lower left window: where the R commands are executed (so here is where you R installation lives). Sometimes also referred to as command window.
- **Environment/History/Connections/Presentation** - upper right window: the objects that you have in your workspace, and the commands you have executed, and more.
- **Files/Plots/Packages/Help/Viewer**- lower right: overview of your files, the plots you produce, the packages you have installed and loaded, and more.

Source window: (Make the source window active.) To start writing a script press File- New File- R Script. To open an existing file, press File- Open File- and select the file you want to open. The file will open in the source window. To save this file, press File- Save as- and go to the working directory there you want to put your TMA4268 R files and save the file as "name".R (example: `myRintro.R`). Files with R code usually have extension `.R`.

Console window: (Make the console window active.) To see your working directory, you can write `getwd()`, and you will get your location as output. You can also set your working directory to a certain folder of choise by writing `setwd("location")` (Example: `setwd("M:/Documents/TMA4268/")`). Now you are certain that all your files will be put in this folder.

Quitting: It is always important to be able to quit a program: when you are finished you may choose RStudio-Quit Rstudio (top menu outside of the windows). Alternatively, you may write `q()`in the console window to quit R (the parenthesis is because q is a function that can have arguments to be given within the parentheses and you call the function without any arguments). You will be asked if you want to save your script and workspave. If you want to reuse your script later (and of cause you want to do that - we aim at reproduciable research), you should save it! If you answer yes to "Save workspace image" all the objects you have created are found in a `.RData` file (more about objects soon). This could be useful if you don't want to run all the commands in the script again, because if you start R in the same working directory all the objects you have created will be automatically availble to you. More on objects next.

You can download the **RStudio IDE cheat sheet**: https://github.com/rstudio/cheatsheets/raw/master/rstudio-ide.pdf

Part B: Trying out R-commands

To exceute your commands, you can either type directly in the console or run the commands from the source window. In the **source window**, you can run the current line by pressing Ctrl and Enter (Windows) or CMD and Enter (MacOS), or you can run select lines by marking them and pressing Ctrl + Enter. You can also use the Run-button in the top right corner of the window to run selected lines or commands, and the Source-button in the top right corner to run everything in your Source window.

Q: Write and execute the following commands. What have you done mathematically here?

```
2 + 3
2 * 6
3 * 10^4 - 3 * 5^2
10^2 - 1
10^(2 - 1)
sqrt(9)
log(3, base = 10)
`?`(log)
log
log10(3)
log(3)
exp(34)
```

```
gamma(3)
factorial(5)
choose(10, 4)
1:4
c(1, 2, 3, 4)
seq(from = 1, to = 4, by = 1)
sum(1:5)
prod(1:5)
heights = c(192, 185, 174, 195, 173)
shoes = c(46, 43, 40, 45, 40)
ratio <- heights/shoes
ratio
```

Here we have created three objects: `heights`, `shoes` and `ratio`. Observe: we can both use `=` and `<-` for assigning content to an object. Notice now that the objects you assigned values to (`heights, shows, ratio`) appear in the Enviroment window (sorted as Data, Values or Functions, but you should only have Values so far).

The function `c` combines values into a vector (concatenate). Also, all the commands you have run are reported in the History window.

If you want to add comments, you do that by starting with a hashtag symbol:

```
# now we quit
q()
```

Save your work as `myRintro.R` - and we will try to open that in Part C.

Part C: Importing and exporting

Executing the commands in an R-file with `source`

You ended Part B by quitting RStudio, now open RStudio again, and open the file `myRintro.R` in your source window. Then either write:

```
source("myRintro.R")  #given that your working directory is wher myRintro.R is saved
```

or source with the source button in the upper right corner of the source window.

It is also possible to source a file from the internet, for example a version of Part B can be sources from the TMA4268 catalog:

```
source("https://www.math.ntnu.no/emner/TMA4268/2019v/1Intro/RintroPartB.R",
    echo = TRUE)
```

Here `echo=TRUE` echoes the commands being run- in addition to the results of the commands.

Reading and writing data into R may be a bit tricky if the format of the data is not defined exactly, so here we just show how to read three nice formats.

You may also choose Enviroment (window upper right) -Import Data set - and get help.

Printing to and reading from a csv-file

Q: Study the R script below and find out what is done in each line of the script.

```r
n = 1000
ds = matrix(rnorm(n), ncol = 10)
colnames(ds) = paste("Variable", 1:10, sep = "")
write.csv(ds, file = "stnorm.csv", row.names = FALSE)  #do not want 1:100 as rownames
getwd()
list.files()  #files in the folder of getwd()
dss = read.csv(file = "stnorm.csv", header = TRUE)
dim(dss)
typeof(dss)
head(dss)
```

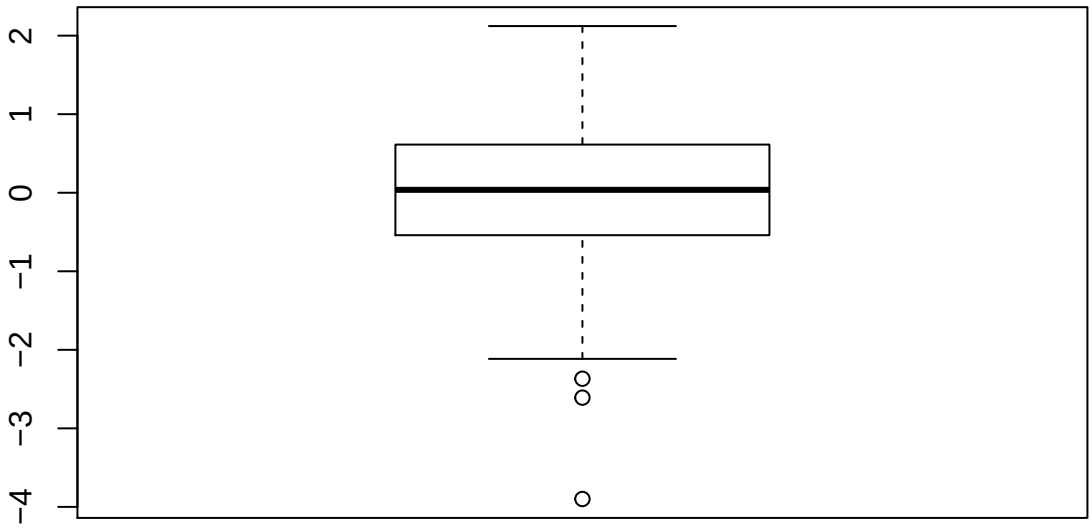
Printing to and reading from other file formats

There exists may packages to read different type of input data, and reading `xls` files can be done using the package `readxl`.

Exporting plots

We will talk more about generating random data from different distribution in `Rintermediate.R` and more about plotting in Part F. However, the following commands draws 100 realizations from the standard normal distribution and makes a boxplot. Write these commands and run them.

```r
ds = rnorm(100)
boxplot(ds)
```



You how should have a boxplot in your Plots window - looking similar to the one shown here. **Q**: What are the different parts of this boxplot? Hint: median, 1 and 3rd quantiles. To read what the function `boxplot` does you may write `help(boxplot)` or just `?boxplot`. How are the whiskers defined?

Now, we want to export this plot - maybe to be put on a webpage or just for fun (we will use R Markdown for our compulsory exercises and will then not need to export plots).

You may save the plot (for example as pdf or svg) Export in the Plots window, or alternatively you may write

```r
dev.copy2pdf(file = "box.pdf")
```

The will produce a pdf-file thtat is saved to your working directory.

A third solution `pdf("box.pdf"); boxplot(ds); dev.off()`to make a file named box.pdf with the boxplot, then it is possible to save many plots together in one pdf-file - just add more plots before closing the pdf-file with `dev.off()`.

Part D: Packages - and functions

R is a free and open source program. Everyone can contribute with making functions (like `boxplot` and `rnorm`) and packages (collection of functions and data sets), and there exists a *lot* of available packages. Some are already included in the default R session, like the package `stats` that includes many basic functions for doing statistics.

Every statistical researcher who would like to to get their new statistitical methods used will make an R package and distribute it with their article (on the new method). Most books also comes with R packages with data sets and functions. Our ISL book has the package `ISLR`, hosted on the most widely used service for R packages: CRAN. See the official page for the package here: https://cran.r-project.org/web/packages/ISLR/index.html.

To install an R package from CRAN you go to the Packages tab and see if the packages is already available on your computer. If you see ISLR in this list just press the square next to ISLR to load the package into R.

If you don't see ISLR you will have to download it from CRAN. Do this by either pressing Install on the top left corner of the Packages window, CRAN is already filled as "Install from" and then write "ISLR" as the name of the package to install, and nice to have chosen "install dependencies" (then packages that ISLR depend on will also be installed). Then press "Install". You might have to select from different mirrors for CRAN - choose Norway, and you are good to go. Then "ISLR" should pop up in the list of packages installed, and you tick (in the square) to load the package into R.

Alternatively, in the console (or source) window you may write:

```
install.packages("ISLR")
library(ISLR)  # to make the package available in the current session
```

Now the packages is installed and loaded to the current session. Remember that whenever starting a new session, you need to reload the packages you want to use, using the `library()` function, or ticking the square next to ISLR in the Packages window.

```
library(help = "ISLR")
```

Q: Look at the contents of the `ISLR` to see that only data sets are available - you may also see that by selecting the name ISLR in the Packages window. To know more about the data set named `NCI60` either just select the data set in the Packages window, or write `help(NCI60)` after ISLR is loaded. What can you say about `NCI60`?

Another package that we will use is `car`. **Q**: Install the `car` package from CRAN, check the content of the package (data sets and functions) and investigate

We will be using a lot of packages in this course, and the ones we use will be listed on the start of each module page. We would assume that you have loaded these packages if you want to reproduce that statistical analyses on the module pages.

Before start using the functions of the package, it is often a good idea to visit the help pages of the package to see which functions and data sets are available, how they are used, what they calculate, and the output they give, etx.. These pages are found in the Help window to the left or typing `?name`in the console, (ex. `?mean`).

In the stats package, you find functions for making and evaluating distributions. We use the function `rnorm` to sample independent data from the univariate normal distribution.

```
rnorm  #lists the function code
`?`(rnorm  #help pages for the function
```

```
)
rnorm()  #gives error
rnorm(n = 100, mean = 0, sd = 1)  #draw random samples from this distribution
`?`(lm  # more to see, will be what we use to perform linear regression
)
```

Part E: More on vectors and matrices

R can handle both numeric and non numeric data. The concatenate c-function can handle both numeric and non-numeric data, but be careful when mixing them.

Q: Go through theses commands and see what is produced.

```
x = c(1, 2, 3)
typeof(x)
y = c("a", "b", "c")
typeof(y)

u = c("1", "2", "3")
typeof(u)
v = as.numeric(u)
typeof(v)

z = c("red", 1, "yellow", 2)
typeof(z)
# w = z - 1 this gives error
```

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

Logical operators are also available, == for equality, != for not equal to, >= for greater than or equal to, etc.

```
gender = factor(c("male", "female", "female", "male"))
gender
sum(gender == "male")
table(gender)
```

```
## [1] male   female female male
## Levels: female male
## [1] 2
## gender
## female   male
##      2      2
```

Useful for vectors:

```
x = 1:5
x = seq(1, 5, length = 5)
x = c(1, 2, 3, 4, 5)
2 %in% x
6 %in% x
x[2] = 10
```

```
x[3:4] = 0
x[-2] = 1
x[c(1, 4)] = 4
x[x > 4] = 10
y = log(x)
z = exp(y)
z = z + y
y = x * y
z = y/x
a = t(x) %*% y  # t(): transpose
min(x)
max(x)
sum(x)
mean(x)
var(x)
length(x)
sort(x)
order(x)
sort(x) == x[order(x)]
```

Notice the length of your vectors when doing calculations with two vectors.

```
x = 1:5
y = 2
x - y
5 * x

z = 10:15
w = 1:2
z - w
```

```
## [1] -1  0  1  2  3
## [1]  5 10 15 20 25
## [1]  9  9 11 11 13 13
```

What happens here?

For matrices:

```
A = matrix(1:6, nrow = 3, ncol = 2)
A
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```
B = matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
B
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
A %*% B  # matrix multiplication
A * t(B)
A %*% t(A)
A^2
```

```
##      [,1] [,2] [,3]
## [1,]   17   22   27
## [2,]   22   29   36
## [3,]   27   36   45
##      [,1] [,2]
## [1,]    1   16
## [2,]    4   25
## [3,]    9   36
##      [,1] [,2] [,3]
## [1,]   17   22   27
## [2,]   22   29   36
## [3,]   27   36   45
##      [,1] [,2]
## [1,]    1   16
## [2,]    4   25
## [3,]    9   36
```

The functions `cbind` (column bind) and `rbind` (row bind) can also be used to create matrices:

```
x1 = 1:3
x2 = c(7, 6, 6)
x3 = c(12, 19, 21)
A = cbind(x1, x2, x3)  # Bind vectors x1, x2, and x3 into a matrix.
# Treats each as a column.
A = rbind(x1, x2, x3)  # Bind vectors x1, x2, and x3 into a matrix.
# Treats each as a row.
```

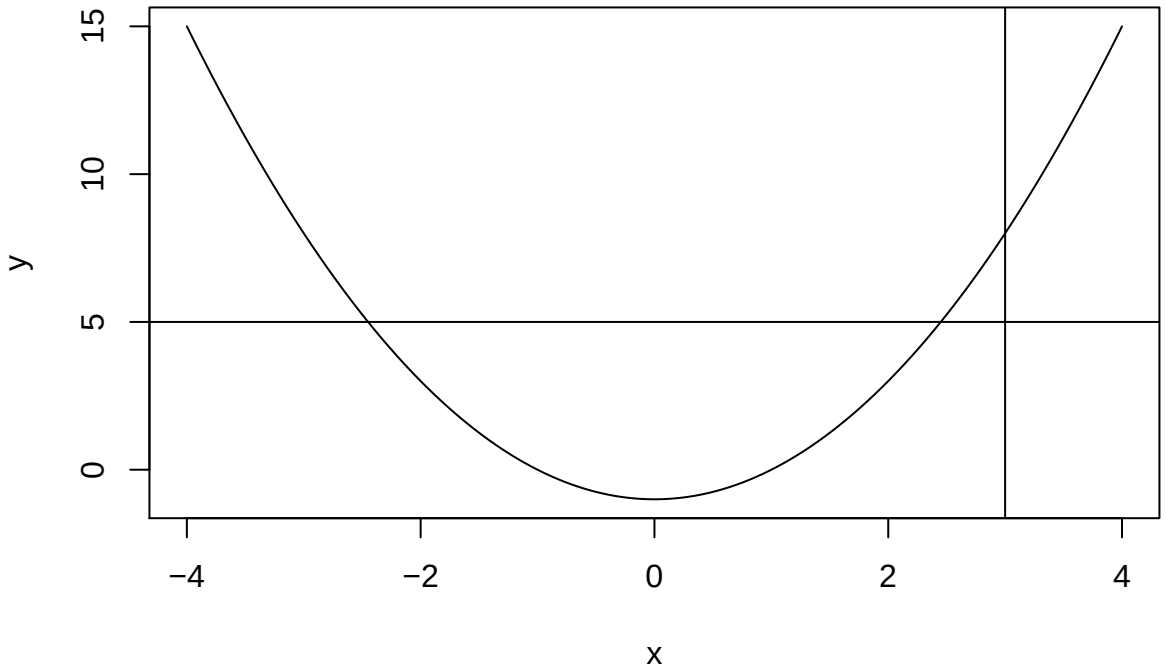
Other matrix commands are

```
dim(A)  # get the dimensions of a matrix
nrow(A)  # number of rows
ncol(A)  # number of columns
apply(A, 1, sum)  # apply the sum function to the rows of A
apply(A, 2, sum)  # apply the sum function to the columns of A
sum(diag(A))  # trace of A
A = diag(1:3)
solve(A)  # inverse of A, in general solve(A,b) solves Ax=b wrt x
det(A)  # determinant of A
```

Part F: Plotting

Create a plot

```
x <- seq(-4, 4, length = 500)
y <- x^2 - 1
plot(x, y, type = "l", main = "My plot", xlab = "x", ylab = "y")
abline(v = 3)
abline(h = 5)
```



To draw the plot in the way you want, check the help pages of the plot function to see which input values you can change to make your plot look the way you want to.

The package `ggplot2`is a powerful tool for making nice plots. In this package, the function `qplot()`can be used to compute the basic graphics from the plot function - however it is better to use the grammar of graphics that is the core of the `ggplot2` package - if you want to learn more about the grammar of graphics you should start by reading Chapter 3 of the book `R for Data Science`.

Before diving in to this, the list below shows some basic tools for plotting using both `plot()`and `qplot()` (the last from the `ggplot2` package).

Description	Base Graphics	ggplot2
Plot y versus x using points	plot(x, y)	qplot(x, y)
Plot y versus x using lines	plot(x, y, type = “l”)	qplot(x, y, geom = “line”)
Plot y versus x using both points and lines	plot(x, y, type = “b”)	qplot(x, y, geom = c(“point”, “line”))
Boxplot of x	boxplot(x)	qplot(x, geom = “boxplot”)
Side-by-side boxplot of x and y	boxplot(x, y)	qplot(x, y, geom = “boxplot”)
Histogram of x	hist(x)	qplot(x, geom = “histogram”)

Plotting will be an important part of any statistical analysis course.

Part G: Writing a simple function

When starting a function, you should start with the name of the function and state if the function takes input values. Then you write the function code inside the branches . Remember to return the output of the function using `return()`.

```
myfunction <- function(x, y) # myfunction is the name, x and y are the names of the inputs
{
    n <- c(length(x), length(y))
    m <- c(sum(x), sum(y))
    p <- m/n
    return(p)
}
```

Q: If `x` and `y` are two vectors of different lengths - what does then the function return?

To start using the function, you must first run it through the console so that it is in your enviroment (mark and run). Then you call the function name and give your inputs like this.

```
a = 1:10
b = seq(from = 0.1, to = 1, length = 10)
p = myfunction(x = a, y = b)  #assign output to a variable p
p
```

```
## [1] 5.50 0.55
```

You can also use `if/else` sentences, `for/while`-loops and `print()`.

```
lett = c("a", "b", "c", "d", "e", "f", "g", "h")
for (i in 1:length(lett)) {
    print("Now we work with:")
    print(lett[i])
    if (lett[i] == "b") {
        print(lett[i])
    } else {
        if (lett[i] == "d") {
            print(lett[i])
        } else {
            print("not b or d")
        }
    }
}
```

While loopes can be written in a similar mannar, using `while` instad of `for`.

Part H: Lists and data frames

Lists and data frames are good tools for storing and accessing your data.

Lists

Using a list, there is no restrictions to the type of data you want to store.

```
a = c("male", "female", "male", "male")
b = matrix(c(1:6), ncol = 2)
```

```
c = rnorm(100, mean = 0, sd = 1)
my_list = list(a = a, b = b, c = c)
```

The list `my_list` now consists of three objects, `a, b` and `c`. To access the data in you list, you write

```
my_list[[1]]  #a
my_list[[2]]  #b
my_list[[3]]  #c
```

or

```
my_list$a  #a
my_list$b  #b
my_list$c  #c
```

To access the second element in the object `a`, you write `my_list[[1]][2]` or `my_list$a[2]`.

Data frames

When using a data frame, you need all your elements in the data frame to be of equal length.

```
Sick = c(0, 1, 1, 0, 0, 0, 1, 0)
Age = c(50, 15, 39, 35, 26, 20, 10, 69)
Sex = factor(c("male", "female", "female", "male", "male", "male", "female",
    "male"))
df = data.frame(Sick = Sick, Age = Age, Sex = Sex)
```

To access the vectors in the data frame,

```
df$Sick
df$Age
df$Sex
```

Similar to a list, we access elements in the data frame using `df$Sex[2]`. If your data frame is very large, it is easier to view typing `View(df)`.

What is next?

You may now move to Rintermediate.html to see how R can be used on topics that should already be familiar to you from TMA4240/TMA4245 Statistics - or similar courses. Or, if you did ST1201 you can look at an overview of how the methods in ST1201 can be performed in R: ST1201inR.html