# Introduction to R

R is a free software for data analysis, which is widely used in ecological science (among other applications). There exist vast amounts of study materials for all proficiency levels, including:

1. Statistikatarkvara R õpetus (Mait Raag, Raivo Kolde) <http://andmeteadus.github.io/2015/rakendustarkvara_R/> (Estonian, quite long)
2. Sissejuhatus statistikapaketti R (Märt Möls) <http://www-1.ms.ut.ee/mart/MC2013/SissejuhatusR.pdf> (Estonian, 22 pages)
3. Statistikapakett R (Ants Kaasik) <http://lepo.it.da.ut.ee/~tammarut/Rkasutamine.txt> (Estonian, about 1 page)
4. An Introduction to R (W. N. Venables, D. M. Smith and the R Core Team) <http://cran.r-project.org/doc/manuals/R-intro.pdf> (English, very long and comprehensive “”introduction””)

Remember, Google is always a useful companion.

This document contains guiding notes for the very first steps in R, including

1. installing R
2. basic functions
3. getting data into and out of R
4. basic plots

## Installing R

1. go to <https://cran.r-project.org/>
2. click “Download R for Windows” (or Linux or Mac, if necessary)
3. click “base”
4. download the installer for the most current version (R 3.5.2 under <https://cran.r-project.org/bin/windows/base/R-3.5.2-win.exe> )
5. run the installer. It is sufficient just to click Next enough times.

If you ever want to **update** your R, then it amounts to installing (the new version of) R for the first time (i.e. follow the previous instructions).

### Installing RStudio

Plain R has quite an ascetic appearance. Therefore I think it is convenient to use some additional user interface on top of R, e.g., RStudio. You can install it as follows:

1. go to <https://www.rstudio.com/products/rstudio/download/>
2. download a suitable installer, e.g., [RStudio 1.1.463 - Windows Vista/7/8/10](https://download1.rstudio.org/RStudio-1.1.463.exe) (<https://download1.rstudio.org/RStudio-1.1.463.exe>)
3. run the installer

## Basic features of R and RStudio

After installing R and RStudio, you can use R through RStudio. If you open RStudio, something similar should appear:



In the left half, you can type commands and see the results.

In the upper right corner, you can see the list of all the variables, datasets, and other R objects you have created.

In the lower right corner, you can currently see a help file. However, there are many other tabs available there, e.g. plots appear to that corner.

### R as calculator. Functions

Click the left half of the RStudio window, to confirm that the cursor is blinking in front of the > symbol: R is now waiting for your commands.

The following commands will yield the obvious results (you may wander what is the default base of logarithm):

(1+3)\*2

3/5

2\*\*3

sqrt(4)

log(10)

sum(1,2, 3)

log(10, base=10)

Notice that square root, logarithm and sum are obtained through typing the name of the procedure followed by something inside brackets. Such procedures are called **functions**. This is direct analogy to functions as they are written in math classes: name of the function followed by argument(s) of that function. A function can have more than one argument, in which case the arguments are separated by commas. Arguments have names, in which case you can specify the argument by writing its name followed by an equal sign and then the desired value. Arguments can have default values, in which case they do not need to be specified at all. **Almost everything we do in R is achieved by the use of such functions.**

If you want to know more about a function (e.g., the format of its arguments), you can ask for help:

?log

Help file opens in the lower right corner of the window.

Notice that auto-complete also works in RStudio (lower left corner, not the script window), so that a list of names of all functions (and other objects) beginning with what you have already written is displayed as you type. So, if you hastily press Enter after typing ?log, then you may instead get the help file for the function logLik.

### Saving scripts

It is good practice (to say the least) to save your work in R, the data manipulation and statistical analysis commands in particular. The commands can be saved into an ordinary text file (for clarity, they bear the extension .R instead of .txt) called “a script”. You can create a new script by New -> Script:



A fourth section opens for writing the script:



Anything you want to write into the script file that is not meant for R (comments, explanations, section titles, etc) are called “comments” and they must be followed by a hash symbol #. RStudio supports a basic sectioning functionality based on comments. Namely, comments followed by at least four hashtags or hyphens are considered section headers and included into a table of contents immediately below the script window.



The script is not saved yet. Not surprisingly, you can save the script using the Save symbol in the upper menu (or shorthand Ctrl+S). In some computers (e.g. my computer) the file extension .R is not added automatically and you should type it manually (i.e. type R-intro-1.R as the file name).

The last piece of puzzle is getting the commands from the script into R (to get the results). There are many options:

* to execute one line of script, click anywhere on the line to place the cursor there, and
	+ press Ctrl+R (or, in some computers, Ctrl+Enter)
	+ click the Run button in the upper right corner of the script window
* to execute any other portion of script, highlight it and
	+ press Ctrl+R (or, in some computers, Ctrl+Enter)
	+ click the Run button in the upper right corner of the script window

Please remember that if you write something into the script (upper left part), then it is not automatically executed. You should use one of the previous options (Ctrl+Enter or sth) to make R actually read the script and do what it says.

### Variables and data types

You can tell R to remember various objects for later use. E.g.,

x = -2.5

will create a variable x and populate it with the number -2.5 (decimal point, not comma). Observe that the upper right corner (Environment) now contains x with value -2.5.

tekst = "Kassid on siledad."

will create a variable tekst and populate it with the text.

You can use the apostrophe ‘ (just left of Enter on most keyboards) instead of the double quote “ for creating character variables.

To create (and over-write) variables, you can use the lesser-sign and minus-sign <- (like an arrow) instead of an equal sign =. This visually emphasizes that the right-hand-side object is put into the container (name) left of the arrow.

Variable names can’t begin with a number, but almost any other sequence of letters, numbers, points and underscores is OK.

**Variable names and R in general is case-sensitive.** E.g.

sqrt(4)

works, but

Sqrt(4)

SQRT(4)

give errors, because sqrt, Sqrt and SQRT are three different objects for R and only the first happens to be meaningful at the moment.

#### Vector

Vector is simply a sequence of data points (numeric or character data). Vectors can be created with the function c (abbreviated from “concatenate” or “combine”, the former is an alias for c):

z <- c(1.5,2,3,4)

w <- c("kass", "koer", "kala", "kala")

You can do various operations with vectors. Some of them are rather obvious:

sum(z)

mean(z)

sd(z) # standard deviation

var(z) # variance

table(w)

length(w)

Otherwise, generally, R tries to do operations element-by-element.

2\*z

z\*\*2

log(z)

z+z

#### Matrix

Matrix is a matrix, not much to add. Matrices can be created with the function matrix.

#### Data frame

A data frame is just a table of data. Data frames can be created by joining vectors into (named) columns with the function data.frame, e.g.:

a <- data.frame(animal=w, smoothness=z)

## Accessing specific data elements

Elements of vectors and matrices can be accessed with square brackets. There are many ways to specify the elements you need:

z[1]

z[c(1,3)]

z[1:3] # see what 1:3 by itself does, too

z[c(T,T,F,F)] # see what c(T,T,F,F) by itself does, too

z[z < 3] # see what z < 3 by itself does, too

z[w == "kala"] # see what w == "kala" by itself does, too

You can leave elements out with a minus sign in front of the indexes:

z[-c(1,3)]

For matrixes, you should use two indices separated by a comma: first for row, second for column.

zmat <- matrix(z, nrow=2)

zmat

zmat[1,2]

zmat[1, ] # get the whole first row

For data frames, there are two options. To access one column of a data frame as a vector, use the dollar sign:

a$smoothness

Then you can use all of the methods described for vectors on that one column:

a$smoothness[a$animal == "kala"]

You can also use square brackets right on the data frame, somewhat like for a matrix:

a[1, 2]

a[c(T, F, F, F), ]

a[c(T, F; F, F), 2]

a[a$animal == "kass", ]

For data frames, you can also specify columns by its name:

a[, c("smoothness")]

## Getting data into R

Here, the main function is read.table, which reads data from (a text) file and creates a data frame. Let’s try this from scratch:

### Toy example

Open Notepad. Write the following text there and save the file with name testdata (the last empty line too). Remember where you saved the file!

nimi,sugu,kaal

Mark,mees,85

Mati,mees,100

Kati,naine,60

Mari,naine,75

Go to R(Studio). Tell R where to look for all the files to read in or where to write the files that should be written, i.e., the **working directory**. Of course, you should use the folder you saved the data file, which is presumably different than where I saved my data.

setwd("C:/Users/Kasutaja/Documents")

You can get the address of the desired directory by **right-clicking** on any file that is already in that directory, clicking on Properties (Atribuudid in Estonian), and then copy-pasting the Location (Asukoht in Estonian).

R does not like backslashes \ in file paths. Please replace all backslashes \ with slashes / or with double backslashes \\.

Mac users don’t need the C: part.

Let’s read in the data. The data frame that is created should be saved into a variable, let’s name it testdata. The function read.table wants to know the name of the file to read, the character that separates data fields, and the fact whether the first line of the file holds the variable names. In short:

testdata <- read.table("testdata.txt", header=TRUE, sep=",")

The data can be printed to screen if you just run the name of the variable:

testdata

There are many further details you can specify for reading in the data (?read.table can be very useful).

### Reading data from Excel

It is unfortunately not extremely easy to “read data from excel in R” (although, you can google it and find solutions). Currently we will follow a “foolproof” schema by first saving the data from Excel to some simpler text format, and then reading it in with the read.table function.

Open the Excel file repedata18eng.xls, downloadable from the course page <http://lepo.it.da.ut.ee/~tammarut/repedata18eng.xls>

Click File -> Save As… and select Save as type: Text (Tab delimited). Click OK.

Close Excel. Click Yes/OK/Save as many times as necessary (four times for me). Again, pay attention to where you save the file. For convenience, please save it into the same folder as the file testdata.txt.

You may look at the data in Notepad. The file is should be in the file repedata18eng.txt, and it should be tab-delimited.

Read in the data with the following command:

repedata <- read.table("repedata18eng.txt", header=T, sep="\t")

Notice that the boolean value TRUE can also be written as T (analogously, FALSE is the same as F).

Also notice that sep="\t" means tab-delimited.

### Some common problems

1. Your data itself contains the separating symbol. In this case, no data frame is created. E.g.

father,mother,children

Martin,Mari,Uku,Kalle

Preventive action: use a different symbol in your data (e.g. a semicolon) or use a different data delimiter (e.g. a tab).

2. Your data contains an apostrophe (or other quoting characters in inappropriate places).

 E.g., a warning message (“incomplete final line found” or sth) may be given:

sender,subject,time

markgimbutas@gmail.com,Mark's homework,2019-01-01 16:45

E.g., an error may be given:

nimi,sugu,kaal

Mati',mees',100

Kati,naine,60

E.g., no warning is given, but the data is wrong:

nimi,sugu,kaal

Mati',mees,100

Kati',naine,60

Solution: Add quote = "\"" to the list of arguments.

If you are bored, you may try to find out what this argument actually specifies. Hint: ?read.table

…

…

## Getting data out of R

It may be desirable to get some processed datasets out of R. For this purpose, R has a function similar to read.table, and it is called write.table. Run and examine the following example:

write.table(testdata, "testdata.csv", sep=";", row.names = F)

The file is written to the working directory. If you can’t remember where it was, you can ask for the current working directory by

getwd()

## Packages

In addition to the basic functionalities we have examined so far, R has numerous collections of functions that are designed for some specific purpose (e.g., survival analysis, linear mixed models, generalized linear mixed models, graphics, text analysis, …). These collections are called **packages** and they are usually not available for you when you initially start R, or they may even not be downloaded to your computer yet.

An example of such an R package that is not initially present in your computer when you install R is ggplot2. It contains many useful functions to draw graphs. To use these functions, you should first install the package to your computer:

install.packages("ggplot2")

Now the package is downloaded (you do not need to run this previous command again), but it is not yet available. To make it available, you should run

library(ggplot2)

once every time when you start R. Then you can use functions like

qplot(data=repedata, x=time, y=shell, color=treatment, group=individual, geom="line")+ theme\_bw()

## Plots and how to save them

R has a powerful plotting system.

Plots can be saved manually to numerous formats. If the plot opens in a separate window, then select File -> Save as and select the desirable format. If the plot opens in the lower right corner, click the Export -> Save as Image… button and proceed.

Plots can also be saved automatically (e.g. if you need a for-cycle to make you tens of very similar plots). E.g.:

png("testdataplot.png")

qplot(data=testdata, fill=sugu, x=sugu, geom="bar")

dev.off()