Statistical Computing
Master Data Science
Winter Semester 2019/20 **Prof. Tim Downie**

BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Introduction to R: Further Reading

This document includes some important aspects of using R. The Examples are given as explanations. Unlike in the Workshops **you are not required to type in all the commands yourself**.

1 Using Files and dirctories

1.1 Current working directory

When you ask R to access information from a file, by default it assumes that the file is located in the *working directory*[1]. R-Studio displays the current working directory next to the word **console** at the top of the console window. There is also an R command to access the current working directory.

```
> getwd()
```

It is helpful to change your working directory to where you want to access data files and save script files graphics etc. The easiest way to do this is via the menus
Session > Set Working Directory > Choose Directory

However if you want to re-run your source code at a later date, then it is better to do this by adding an R-Command at the start of the script file.

```
> setwd("H:/StatComp/")
```

The syntax here is somewhat annoying. For unimportant historical reasons R uses / and windows uses \ to seperate the directory names in the path.

1.2 Reading data from files

Understanding data formats, data manipulation and importing data is an essential part of data science. You will learn much more on this topic in other courses and later on in this course. Today we cover just the basics so that we are not limited to data files which come with the R installation.

Which command you use to read in data depends on which format the data is in. For text files the usual functions are `read.table()` and `read.csv()`. Other possibilities are `read.csv2()`,

[1]directory=folder

`read.fwf()` or `scan()`. There are also functions to read in data in a format that was saved by another statistical software such as SPSS or SAS.

A good summary of reading in data for different formats can be found at (link) `https://www.datacamp.com/community/tutorials/r-data-import-tutorial#gs.oeoGUXk`.

Note that the external "text" files must be in ASCII format. Today most numerical data sets are originally created electronically rather than typed in using a keyboard. If you do need to create a dataset by hand then I suggest using Excel and saving it as a comma separated version `.csv`. Word is not recommended as it is too easy for the layout to go wrong.

Example 7.1

Suppose that your current working directory contains a file named `somedata.txt`, which contains the following:

```
 2  100.0
 3   44.5
```

In R,

```
> x <- read.table("somedata.txt",header=FALSE)
> x
  V1    V2
1  2 100.0
2  3  44.5
```

1.3 Reading commands from a file

Typically, a sequence of many commands is required for an analysis. By storing them in a script file, the analysis can be repeated or modified later on.

While you're working out how to do the analysis, type the commands into the RStudio script file and run each line individually. To execute lots of commands in one go by highlighting the code you want to run and typing $\boxed{\textit{Ctrl}}$+$\boxed{\textit{Enter}}$ ($\boxed{\textit{Strg}}$+$\boxed{\textit{Enter}}$) or clicking on the $\boxed{\textit{Run}}$ button at the top of the source file.

Another way is to source the whole file. If you click on $\boxed{\textit{Source}}$ the script file will be saved and then run using the `source()` command. You can use also use `source(...)` directly at the command prompt or in a script file to run another script file.

For example, suppose the *current working directory* contains a file `somecommands.r`, whose contents are as follows:

```
dat <- c(3,3,3)
print(dat)
```

(with a carriage return after the last line in the file). You can run these commands by typing

```
> source("somecommands.r")
[1] 3 3 3
```

or

```
> source("somecommands.r",echo=TRUE)
> dat <- c(3,3,3)
> print(dat)
[1] 3 3 3
```

1.3.1 Commenting your programs

To make your programs readable by a human, it is helpful to insert lines that explain what's going on. These are called *comments* in programming jargon. To include comments in an R program, use the `#` character: R will ignore anything on a line after encountering this character. Here is a simple R program:

```
#
# This is a very simple program that illustrates the value of comments.
# It starts by asking the user to input their name ...

username <- readline(prompt="Please input your name: ")

# ... and then it says hello to them, personally.

cat(paste("Hello, ",username,"!\n",sep=""))
```

Even though you don't know all the commands, the program is (hopefully) perfectly intelligible because the comments explain what's going on.

Get into the habit of commenting your programs clearly. This applies not just to R but to all types of coding. If you have assessments where you need to write code (includung the tests in this course) you can only gain high marks if the lecturer can understand what you've written!

1.4 Saving output to a file

When you run commands in RStudio, results are by default written to the screen. The `sink()` command is used to direct output to a file instead of the screen: this file can then be printed and examined. If you have stored all of your R commands in a file called `somecommands.r`, this enables you to print a 'clean' copy of the results:

```
> sink("somecommands.res")
> source("somecommands.r",echo=TRUE)
> sink()
```

Now you will not see anything on screen when you run `source("somecommands.r")`. The results are written straight to `somecommands.res`. The final `sink()` command tells R to stop writing to this file and return output to the screen. N.B. R-Studio does not at the moment allow you to save the console window file.

1.5 Current Workspace

All assigned variables (or any other R *objects*) are automatically stored by the computer in memory, in your R *workspace* also known as *Global Environment*, until you finish your R session. When you finish, R gives you the option to save your workspace for future use.

Experienced R users recommended that you you do not save your workspace for two reasons. If you always save it, all objects will remain in the workspace until overwritten or explicitly deleted by the command `rm()` (for *remove*). This means that, after a year or so, your R files can take up a lot of memory, and can slow down R, and you end up with lots of objects, the prupose of which you have long forgotten.

A rarer but more serious problem of reloading your past objects is that your code could be relying on an object that was defined 6 months ago and not defined in your current source code. This means that when you do eventually run the code in a clean environment or if you pass the code onto another user then it will not do what you expect it to.

In the Environment Window, top right, you can see all the objects in the current "workspace". To output your workspace objects in the R console, type `objects()`.

Example continued

```
> x <- 8
> x
[1] 8
> y <- 3.1415
> objects()
[1] "x" "y"
> rm(x)
[1] "y"
```

2 Graphics

Many R commands produce high-quality graphical output. This can be saved to a variety of file formats for inclusion in documents, web pages etc. For example:

```
> x <- seq(-5,5,0.1)
> y <- x^2
> plot(x,y,type="lines")
> dev.copy(jpeg,"quadratic.jpg",quality=95)
JPEG
   3
> dev.off()
X11
  2
> dev.copy(postscript,"quadratic.ps",paper="a4",horizontal=TRUE)
postscript
         3
> dev.off()
X11
  2
>
```

File formats '`jpeg`' and '`png`' may be used to produce graphics that can be imported into Microsoft Word documents. The postscript file format is useful for inclusion into LaTeX documents (more about LaTeX when you do your projects in the summer, probably), and for printing without any loss of quality. Postscript files can be sent directly to many printers — most printers at UCL support this.

3 Using Packages

There are hundreds of extensions to R called packages, which are publicly available. These packages have been contributed by R users, who have developed R functions for specific purposes.

An example is the package `survival`. This includes many functions used in the statistical field called "survival analysis" and was contributed by Terry Therneau, a statistician who has contributed many extensions to R.

You will probably need to use many R extensions in this degree Programme, but they are easy to install and use.

3.1 Installing a Package

The R code to install a package is

```
> install.packages("packagename")
```

This asks you to choose a mirror site (somewhere local i.e. in Germany is best). The installation might take a minute or so and install some other "dependency packages" but this should be all you need to do.

In some instances you might have problems with administrative rights not being able to write files to certain directories. If you are at home and on a windows PC and have this problem quit R , right click on the R-Studio Icon and run as administrator. If you are on a Linux OS login as super-user and run R-Studio. When installing a package in the Beuth Maths-Lab as part of the Workshop, your lecturer will give you the instructions specific to that network.

You will only need to install specific package once on each computer.

To use the functions inside a package run one of the two commands

```
> library(packagename)
```

or

```
> require(packagename)
```

Quotes (`"`) around `packagename` are optional. The difference between `library()` and require() is minimal and not important at this stage.