Workshop 1 — Using R and R-Studio for the First Time

1 Introduction

1.1 What is R?

- R is a computer statistics system based on the programming language S.
- R has many "inbuilt" statistical commands (e.g. to do an Analysis of Variance). It also provides an environment for writing your own functions.
- Advantages of R over other statistics packages include flexibility, simplicity, and quality of graphical display.
- R is available free of charge and runs on Windows, Mac and Linux operating systems. You can download your own version from this clickable link:http://www.R-project.org.

Included in the document *Further Reading Week 1* (Moodle) is section called *A Quick History of* S-Plus *and* R

1.2 R and R-Studio

The statistics courses in this Data Science Program will use R extensively. We will be using R-Studio as the front end, which makes writing source code, handling data objects and graphics a little easier. All the computing is done by R itself and to use R-Studio, you have to install R first.

Other lecturers might use R without R-Studio; there is no difference in the code and output, so you will be able to use R-Studio in other courses, even if the the lecturer uses R directly. If you use Linux or a Mac at home then again there will be no difference in the code and output. In the exam all questions will be independent of operating system and front end software. In the R-Test the exercise sheet will be written assuming that you are using R-Studio .

R Studio is also free and available for Windows/Mac/Linux from this clickable link:
https://www.rstudio.com/products/rstudio/download/

2 Getting Started

In this section we will concentrate on getting started with R and R-Studio in the Maths-Lab. The following Set-Up instructions are specifically for the computers in the Lab.

2.1 Setting up a directory

Whenever you use a computer, it is good practice to organise your files in such a way that you can find them again later. This is accomplished by keeping related groups of files together in *directories* or *folders*. You have a folder on the Mathematics-Network. Use your login details given to you by the lecturer, don't forget to change your password the first time you login.

Open a *Windows Explorer* Window. There you will see a Network Drive with the letter `H:\\`
In this drive create a folder called `StatComp`. You will save all source code data and graphics files in this folder.

2.2 Starting R-Studio

Find the icon on the Windows Desktop called **Applications** or **Anwendungen**. Double click on the icon and scroll down until you find R-Studio Icon. Drag the Icon onto your desktop. Start the Program.

- The first time R-Studio is started a dialogue-box opens, so that you can input the path for the R executable files. Click on the option *choose a specific* ..... and *Browse*.
- An Explorer window will open. In the directory line near the top of the window type in the following: `\\pc04\Anwendungen\Tex\R\bin\x64`
- Click *OK* and the R-Studio window should open.

On the left side of the R-Studio window is a sub-window called *console* with the text

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

2.3 Expressions or assignments and arithmetic

Elementary R commands are either *expressions* or *assignments*.

- An *expression* is a command to simply display the result of a calculation, which is *not* retained in the computer's memory
- An *assignment* passes the result of a calculation to a variable name which is stored (but the result will not necessarily be printed out on the screen).

The > symbol is called the R prompt. Type the following simple commands after the > symbol, you do not need to type >, we use it to show that the following text is an R command.

An example of an *expression* is:

```
> 5 + 2.6
```

► Type an expression to calculate $1234 + 2468$.

An example of an `assignment` is:

```
> x <- 5 + 2.6
> x
```

Note the use of the assignment operator '<−' above. This reads as an arrow pointing to the object `x`. You can think of the command `x <- 5 + 2.6` as "*take the value of the expression* $5 + 2.6$*, and store it in the object called* `x`."

Typing `x` at the prompt displays the contents of that object. You can see all of your current R objects in the *Environment* window top left. ► Use the arithmetic operators −, $\star$ (multiplication) / (division) and ^ (exponentiation - "raising to a power") for the following calculations.

i) $87 - 105$

ii) 32.4×18

iii) $13/8$

iv) 6^3

2.4 The script window and script files

Your *console* probably takes up the whole left hand side of the R-Studio window. If this is the case select *File* > *New File* > R *Script* from the drop down menus. Now you see a window called *Untitled 1**, where you can enter and edit your R commands. Save the empty file with the name `Workshop1.R` in your `H:\\StatComp` folder, using
File > Save as .... It is useful to save this file from time to time Strg + S[1], as computers can crash.

[1] In case you don't already know Strg is the *Control* key on a German keyboard

From now on, type in all your commands into the script window, and run them using Strg + Enter while the cursor is on the command. If you have an error correct the code and re-run it.

2.5 Arithmetic continued and mathematical functions

There are many mathematical functions in R, which are intuitive to use. Here are a few important ones To calculate the square root use the function `sqrt()`

```
> sqrt(729)
```

► Calculate $\sqrt{3^2 + 4^2}$.

`> exp(3)` is the R-Function to calculate e^3.

`log()` is a function which calculates the natural logarithm, i.e. logarithm to base e.
► What is the logarithm of 7.4?

The `log()` function accepts an optional argument called `base=`.

```
> log(3, base=10)
```

► What is the logarithm for 100 base 10 and log 64 to base 2?

`> abs(-5))` returns the absolute value.

3 Simple Manipulations, Numbers and Vectors

3.1 Creating vectors in R

The command `c()` (for *combine*) creates R vectors from numbers, objkects or other vectors .

```
> x <- c(2.3,1.2,2.4)
> x

> y <- c(x,9.0,x)
> y
```

We often want to create integer sequences. The expression `1:n` denotes the sequences $1, 2, \ldots, n-1, n$.

```
> z <- 1:5
> z
```

This notation is very important in R, you will be using it a lot.

More generally, `seq(i,j,k)` is a sequence from i to j in steps of k.

```
> seq(0,10,by=2)
```
outputs the even numbers between 0 and 10.

Assign the first 10 Fibonacci numbers $1, 1, 2, 3, 5, 8, 13, 21, 34, 55$ to the vector called `fibonacci`
```
> fibonacci<-c(1,1,2,3,5,8,13,21,34,55)
```

In the script window type the following *comment* above the previous command.
```
#fibonacci is a vector containing the first 10 Fibonacci numbers.
```
The hash symbol # indicates that the following text is a comment and not an R-Command. If you run this text R will not report an error. This helps you make your code readable by a human, by inserting lines that explain what's going on.

As the Workshop continues, annotate your code with section names and add your own comments.

3.2 R vector arithmetic

You can also use the arithmetic operators +, −, ⋆, / and ^ on vectors. Vector operations are done element by element.

```
> x <- c(2,3)
> y <- c(1,4,5,6)
> 2*x
> 2 + x
> y^2
```
N.B. for multiplication the asterisk ⋆ is required: `2x` gives an error.

► Use `:` and ^ to output the first 5 square numbers.

If the vectors have different lengths the shorter is „recycled" until it has the same length as the longer vector. Make sure you understand the output of these two commands
```
> x + y
> x*y
```

Square brackets are used to access an element in a vector. The 7th Fibonacci number is:
```
> fibonacci[7]
```

Add the 7th and 8th Elements of the vector `fibonacci` together
```
> fibonacci[7]+fibonacci[8]
```

Run a similar command to calculate the 11th Fibonacci number.

Now add the 11th Fibonacci number to the end of the vector.

```
> fibonacci<-c(fibonacci,???)
```

Throughout the course `???` indicates code that is missing and you have to complete.

Most mathematical functions accept a vector as the argument. The output is either the function applied to each element or a single answer as appropriate.
E.g. to output the Exponential of every number in the sequence $-1, 0, 1, \ldots, 5$

```
> exp(-1:5)
```

The function `mean()` outputs one value, the mean (average) of the input vector.

```
> mean(1:10)
```

► What is the log of each number in the sequence $-1, 0, 1, \ldots, 5$? Why do get an odd warning?

► Assign the sequence $1, 2, \ldots 10$ to an object. Use this to calculate $\sum_{i=1}^{10} i$.
Hint: How do you say the symbol „$\sum$" in English? There is an R-Function with this name.

► Calculate the sum and the average of the first 11 Fibonacci numbers.

3.2.1 Operator precedence

It's important to understand the order in which R does things. For the standard arithmetic operations +, −, ⋆, / and ^, R follows standard computing rules:

Priority	Operator
highest 1	^
2	:
3	⋆ and /
lowest 4	+ and −

```
> 4+2*3
[1] 10
> 2^2*3
[1] 12
```

Note, however, that it is difficult to read the commands above. In general, it is good practice to make explicit the order in which you *want* the calculations to be performed, using brackets:

```
> 4+(2*3)
[1] 10
> (4+2)*3
[1] 18
```

Not only does this make the commands easier to read; it also guarantees that R is not making any decisions for you. Much of the time, the decisions it makes will be very intuitive. However, this is not

always the case and it's always best to err on the side of caution when you're anywhere near a computer. Compare

```
> x <- 3
> 1:x
```

with

```
> 1:x+2
```

You might expect the last command to produce the sequence 1 to $x + 2$ i.e. 1,2,3,4 and 5. However, in R, the sequence operator `:` has higher precedence than *any* of the arithmetic operators. So R interprets the last command above as `(1:x)+2`. If we want a sequence from 1 to $x + 2$, we have to enter

```
> 1:(x+2)
```

3.3 Types of vector

All the R objects you have seen so far have been *numeric*. R also understands vectors consisting of *characters*, *logical values* (true or false) and *factors*. These are called *data types*, which you will learn about in Lecture 2. Factors are used to assign the values to groups, and are important in procedures such as analysis of variance. For example, patients in a clinical trial may be given one of three different doses (low, medium and high) of a drug; in an agricultural experiment, plots of land may be treated with one of five different fertilisers.

Example 4.3

```
> mywords <- c("This","is","a","vector","of,"characters")
> mywords
[1] "This"     "is"       "a"    "vector"    "of"       "characters"
> mylogical <-  mywords == "is"
> mylogical
[1] FALSE TRUE FALSE FALSE
> myfactor <- factor(c("Low","Medium","High","Low","Low"))
> myfactor
[1] Low    Medium High   Low    Low
Levels: High Low Medium
```

3.4 Indexing/subsetting vector components

Often we wish to refer to a single element of a vector, or some subset of its elements. To index components of the vector `x`, we use the form `x[...]`.

Example 4.4

```
> x <- c(1,3,4,7)
```

```
> x[1]
> fibonacci[8]
> x[2:4]
> fibonacci[c(3,8)]
> x[-2]
> x[x > 3.5]
```

Notice the odd notation `x[-2]`. This means omit the second element of `x`.

► Type a command which outputs all elements except for the 5th element of the `fibonacci` vector.

3.5 More on functions

Notice that most R functions, e.g. `seq()`, are followed by parentheses which usually contain additional information for the function.

Writing a command name without parentheses simply makes R write out the source code for the function.

Example 3.1 continued

```
> seq
function (...)
UseMethod("seq")
<bytecode: 0x0000000004dc7838>
<environment: namespace:base>
```

In the command `log(3, base=10)` there are two arguments. The second is a so called "*named argument*". Named arguments are usually used to specify an option, in this example the base of the logarithm. The *default* value of `base` is e. The order in which *named arguments* are specified is not important.

```
> log(3, base=10)
> log(base=10, 3)
```

Arguments may be unnamed, but have to be specified in the right order. This is usually intuitive. Here are a few examples:

```
> seq(1,7)
> seq(from=1,to=7,by=2)
> seq(by=2,to=7,from=1)
> seq(1,7,2)
> seq(2,7,1)
```

```
> seq(from=1,to=25,length=4)
> seq(1,25,4)
```

Notice the difference in the last two commands. The 3rd Argument in `seq()` is '`by`' not '`length`'

4 Random Numbers and Probability Distributions

Type the following command in your script window and run it:

```
> rbinom(10,5,0.5)
```

Now repeat the command several times. What do you notice about the output? Each time the command is run, it generates 10 *random* numbers. The random numbers come from the binomial distribution with parameters $n = 5$ and $p = 0.5$. By default the output is different each time.

You can specify a so called *seed* for the random number generator. This fixes the sequence so that you can replicate the sequence of random numbers.

```
> set.seed(100)
> rbinom(10,5,0.5)
```

The output is

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

► Repeat both commands and you will this time get the same output.

The function `dbinom()` outputs probabilities from the Binomial distribution and `pbinom()` cumulative probabilities (c.d.f).

Examples:

The probability of three heads from 5 coin tosses is

```
> dbinom(3,5,0.5)
```

The probability of three heads or fewer from 5 coin tosses is

```
> pbinom(3,5,0.5)
```

► Find the probability that, from 4 die rolls, a six is rolled at most twice.

5 The Iris Dataset

The Iris dataset has measurements from three species of iris (a type of flower). This is a classic data set which is often used as a first-example data set in machine learning, despite being a small data set.

First of all load the dataset, which comes with the R installation.

```
> data("iris")
```

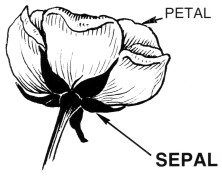
In the Environment window (top right) you should see the text **iris** and `promise`.

Run the command

```
> names(iris)
```

which returns the variable names of the Iris dataset.

Notice that the Environment window has changed and next to **iris** you can see „150 obs. of 5 variables". Click on the word `iris` in this window and you now see the data in the top left window. The dataset is a matrix with 150 rows (observations) and 5 columns (variables). The variable names are: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width` and `Species`.



A dataset in this matrix format is called a `data.frame`.

To read more about the Iris dataset type

```
> help(iris)
```

or

```
> ?iris
```

Most R commands and datasets have a help page, which can be accessed usinh `help()` or `?`.

```
> summary(iris)
```

produces a summary of each variable. Use the output to answer the following questions. YOU can either type the answers into your script file as comments or open a Word document to write the solutions.

i) How many flowers with species *Versicola* are in the data frame?

ii) What is the average petal length?

A variable in a `data.frame` can be accessed via:

`data.frame.Name$variablen.name`, e.g.

```
> mean(iris$Petal.Length)
```

► How long is the longest sepal length? (Hint: `max()`)

Create a histogram of the variable `Sepal.Length`

```
> hist(iris$Petal.Length)
```

► Edit your `hist()` command with the following arguments `xlab="Petal length in cm"` and `main="Histogram of Iris petal length"`

Create a "*dot plot*" or "*strip chart*" for the variable `Sepal.Length` grouped by `Species`.

```
> stripchart(Sepal.Length~Species,data=iris)
```

► Click on *Zoom* in the graphics window. What is the data being displayed here?

► Use the diagram to estimate the longest sepal in the Setosa species.

The following command displays the petal width against the petal length in a scatter diagram. The colour of each point corresponds to the species.

```
> plot(iris$Petal.Length,iris$Petal.Width,col=iris$Species,pch=16)
```

6 Tidying Up

At the end of each R session it is good practice to tidy up your source code and save all necessary files or edited data.

► Scroll up to the start of the script window and add the comment.

```
#Workshop 1 - Using RStudio for the First Time
```

► Find the command `> data("iris").`
Add the comment `#Section 5 - the iris dataset` directly above this line.

► Save the script file (source file) again: *Strg* + *S* or *File > Save as*. You can return to this source file in a later R session and re-run some or all of the commands.

► Leave R-Studio by typing the command:

```
> q()
```

`q` means *quit*. When R asks you *Save workspace image ...?*, click on **Don't save**!

► Feierabend![2] Please don't forget to read the *Further Reading* document before next Friday.

[2] I'll use the word Feierabend, as I think it's a much nicer word than the English equivalent "home-time"