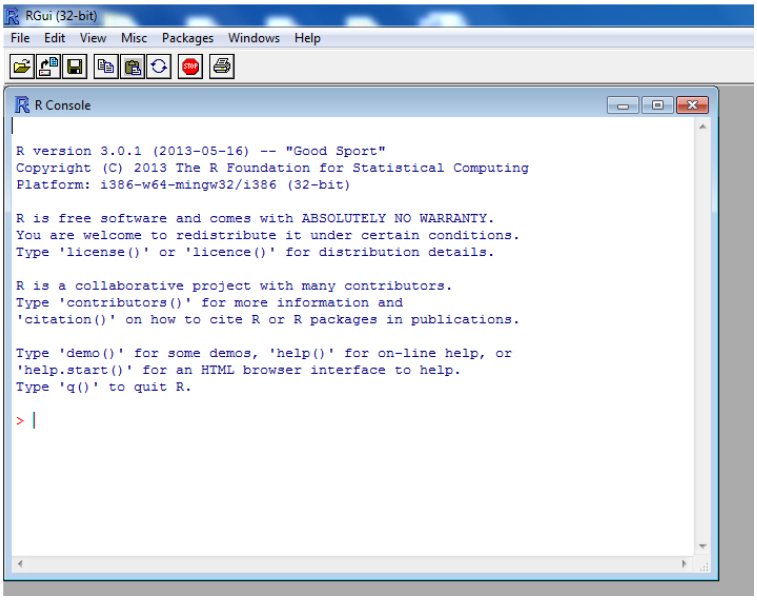
Introduction

R, is a free open statistical software package that is increasingly finding use in statistical analysis and is rapidly becoming the industry standard. In this chapter we will give a brief introduction to the statistical software package R. We will explain some of the features that are relevant to know in a introductory statistics course. For further reading, please refer to [2], [4], or [5], where [4] is part of the online R-documentation given in `http://cran.wustl.edu/doc/manuals/R-intro.pdf`. The book [5] is also available for free online.

The Command Window and R-scripts

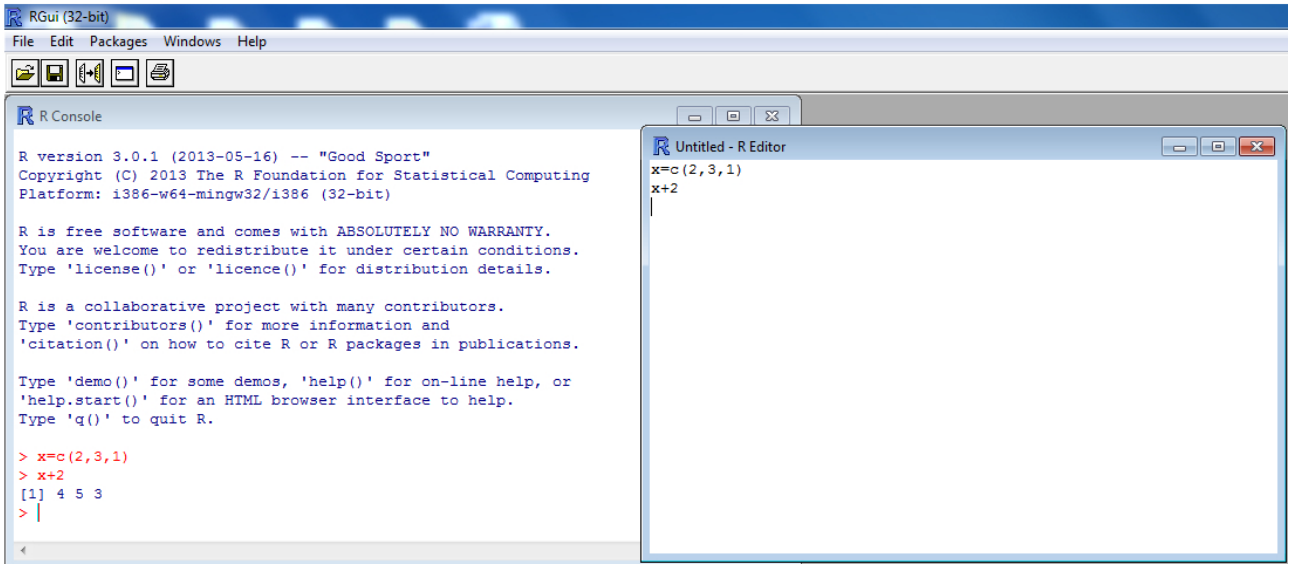
After we have installed R, we can access it by double click on its icon or enter the command "R" at the computer's command line. We will see the following window which is the console in R for where we enter the code.



Instead of entering the code directly into the R-console, it is convenient to create a R-script allowing us to edit and save the code. The R-scripts are created using a text editor. The directions to create a R-script using the default R-editor are as follows:

- First click **File** and select **New Script**.
- Now enter the code into the **R-editor** that pops up.
- Save the script by clicking on the R-editor to make it the active window and then hit **File** and select **Save as**.
- To open the script, hit **File** and select **Open script**.

- To run the code we can copy and paste the code into the R-console from the R-editor. Alternatively, highlight the code in the R-editor and hit **Edit** and select **Rune line or selection** or select **Run all**.



The Basics

The Command Line

After starting the computer program, the command line with the sign " > " appears as in

```
>
```

The commands are executed when we hit *enter*:

```
> 1+1
[1] 2
```

R prints out the line [1] 2. The meaning of the symbol [1] will be clearer in the next example.

We wish to generate 10 random numbers uniformly spread between 0 and 1. We enter:

```
> runif(10)
 [1] 0.05331820 0.23554952 0.41056000 0.90195242 0.04207462 0.49889091
 [7] 0.64395255 0.48124812 0.34240936 0.78748225
```

Here [7] indicates that the number 0.64395255 is the seventh observation. (Notice that you will get a different result if you try this command since $runif()$ is a random number generator).

R works as a calculator in that we can enter commands as:

```
> 2*3
[1] 6

> 6/2
[1] 3
```

To calculate e^4, enter

```
> exp(4)
[1] 54.59815
```

Assignments
To assign a value to a variable, we use the command $<-$ or $=$. To assign the value 3 to the variable x, enter

```
> x <-3
```

or

```
> x=3
```

We will use the notation "=" throughout the book for assignments of variables.
Now that the value 3 is stored in the variable x, we can include it in calculations. To add 4 to x, enter

```
> x+4
[1] 7
```

Vectors
To create an array of numbers we can use the commands **c()**, **seq()**, or **rep()**.
To create an array of the following numbers, $2, 3, 1, 4$, assigned to x, enter

```
> x=c(2,3,1,4)
```

We call this array of numbers a *vector* and we call x a *vector variable*. To view the values stored in x, enter

```
> x
[1] 2 3 1 4
```

To generate an array of equally spaced numbers from -1 to 1 in steps of 0.1, enter

```
> seq(-1,1,0.1)
 [1] -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1  0.0  0.1  0.2  0.3  0.4
[16]  0.5  0.6  0.7  0.8  0.9  1.0
```

To create a vector of 1's repeated 5 times enter

```
> rep(1,5)
[1] 1 1 1 1 1
```

To create a vector with entries $1, 2$ repeated 5 times, enter

```
> rep(c(1,2),5)
 [1] 1 2 1 2 1 2 1 2 1 2
```

Here we put the numbers 1 and 2 in a vector such that the vector gets repeated 5 times using the command **c()**.

After we have created the vector, we can perform calculations on it. Enter

```
> x=c(2,3,1,4)
> y=c(1,4,5,2)
> x+y
[1] 3 7 6 6
> 3*x
[1]  6  9  3 12
```

To find the sum and mean of the components of the vector x, enter

```
> sum(x)
[1] 10
> sum(x)/length(x)
[1] 2.5
```

R has the following built-in command for the mean:

```
> mean(x)
[1] 2.5
```

By entering

```
> x=c(2,3,1,4)
> sort(x)
[1] 1 2 3 4
```

the values of x gets sorted in increasing order.
To access the second component of the vector x, enter

```
> x=c(2,3,1,4)
> x[2]
[1] 3
```

To access the second and fourth components of the vector x, enter

```
> x=c(2,3,1,4)
> x[c(2,4)]
[1] 3 4
```

We observe that we have created a subvector of x.

Character vectors
A vector of text strings is called a *character vector*. We specify the elements in quotes as in

```
> c("pink","blue", "green")
[1] "pink"  "blue"  "green"
```

Logical vectors
A vector that consists of TRUE and FALSE is called a *logical vector*. Here is an example:

```
> c(F,T,T,F)
[1] FALSE  TRUE  TRUE FALSE
```

However, we normally do not create a logical vector in this manner. They are rather involved in expressions as shown in the following example:

```
> x=c(2,1,6,7,3)
> x<5
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

R returns **TRUE** for entries in x that are smaller than 5 and **FALSE** for entries that are greater than or equal to 5.

Data frames
An important part in statistics and data analysis is working with data frames. To create a data frame with column names weight (in kg) and height (in cm), enter

```
> height=c(155,177,164,180)
> weight=c(65,83,74,115)
> d=data.frame(height,weight)
> d
  height weight
1    155     65
2    177     83
3    164     74
4    180    115
```

Indexing
To access the element in the first row and second column which is the weight for subject number 1, enter

```
> d[1,2]
[1] 65
```

To access all the elements in row 1, enter

```
> d[1,]
  height weight
1    155     65
```

which is the data for subject number 1.

To access all the elements in column 1, which are heights for all subjects, enter

```
> d[,1]
[1] 155 177 164 180
```

To access the first three row elements, enter

```
> d[1:3,]
  height weight
1    155     65
2    177     83
3    164     74
```

which returns the weights and heights for the first three subjects.
To extract the data in the data frame for which height is greater than 164, enter

```
> d[d$height>164,]
  height weight
2    177     83
4    180    115
```

Entering **d$height**> 164 returns a logical vector with the value TRUE for the entries where height is greater than 164 as in

```
> d$height>164
[1] FALSE  TRUE FALSE  TRUE
```

More on Data frames
Enter

```
> Puromycin
   conc rate     state
1  0.02   76   treated
2  0.02   47   treated
.. ...    ..   .......
12 1.10  200   treated
13 0.02   67 untreated
.. ...    ..   .......
22 0.56  158 untreated
23 1.10  160 untreated
```

The data **Puromycin** is a built-in data set in R [1] and [3]. The data shows the reaction rate for an enzymatic reaction versus the concentration of substrate for cells treated with Puromycin or untreated cells.
To extract the individual variable **rate** from the data frame **Puromycin**, we use the $ notation as in

```
> Puromycin$rate
 [1]  76  47  97 107 123 139 159 152 191 201 207 200  67  51  84  86  98 115 131
[20] 124 144 158 160
```

If we wish to group the **rate** into vectors separated by **untreated** cells and **treated** cells, enter

```
> tr.cells=Puromycin$rate[Puromycin$state=="treated"]
> tr.cells
 [1]  76  47  97 107 123 139 159 152 191 201 207 200
> untr.cells=Puromycin$rate[Puromycin$state=="untreated"]
> untr.cells
 [1]  67  51  84  86  98 115 131 124 144 158 160
```

In the first command above, we extract the value of the variable **rate** from the dataframe **Puromycin** for which the **state** is **treated**. We name this vector of observations for **tr.cells**.

Attach and Detach
We can use the command **attach** to extract every variable from the dataframe as in

```
> attach(Puromycin)
```

Now we can type

```
> rate
 [1]  76  47  97 107 123 139 159 152 191 201 207 200  67  51  84  86  98 115 131
[20] 124 144 158 160
```

and avoid using the $ notation.
To detach the variable from the search path, we add the command

```
> detach(Puromycin)
```

Subsets of the data frame
To access the subsets of elements of the data frame **Puromycin** for which **rate** < 100, enter

```
> P.subset=subset(Puromycin,rate<100)
> P.subset
   conc rate     state
1  0.02   76   treated
2  0.02   47   treated
3  0.06   97   treated
13 0.02   67 untreated
14 0.02   51 untreated
15 0.06   84 untreated
16 0.06   86 untreated
17 0.11   98 untreated
```

sapply
To perform a calculation of each of the quantitative variables in a data frame, we can use the **sapply** command. Consider again the data frame **Puromycin**. We will demonstrate how to calculate the mean of the observations in a data frame:

```
> sapply(Puromycin,mean,na.rm=T)
       conc        rate       state
  0.3121739 126.8260870          NA
Warning message:
In mean.default(X[[3L]], ...) :
  argument is not numeric or logical: returning NA
```

If we add the argument **na.rm=T**, then R will remove the missing values from the calculation. We obtained a Warning message since the entries in **state** are not numerical values.

Matrices
To create a 2×3 matrix, we enter:

```
> matrix(c(1,3,2,7,5,2),nrow=2)
     [,1] [,2] [,3]
[1,]    1    2    5
[2,]    3    7    2
```

Explanation. The code can be explained as follows:

- The entry **nrow=2** creates a matrix with 2 rows. The matrix is filled column-wise.

Adding the entry **byrow=T** results in a row-wise filling of the entries as in

```
> matrix(c(1,3,2,7,5,2),nrow=2,byrow=T)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    7    5    2
```

Creating Functions
A function is on the form

```
f=function(arg_1,arg_2,...,arg_n){expression}
```

The expression uses arg_1, arg_2, ..., arg_n as inputs and calculates a value. To call the function, we invoke the command

```
f(input_1,input_2,..., input_n)
```

and the function f returns its value. Here is an example. Create a function that adds the means of two samples of observations.

```
> f=function(x1,x2){
+ x1bar=mean(x1)
+ x2bar=mean(x2)
+ mean=x1bar+x2bar
+ }
> data1=c(2,5,7,3,2)
> data2=c(1,3,2,8,3)
> mean.value=f(data1,data2)
> mean.value
[1] 7.2
```

Explanation. The code can be explained as follows:

- The arguments in the function f are $x1$ and $x2$. The function calculates the means of $x1$ and $x2$ and store the values in **x1bar** and **x2bar**, respectively.
- The function calculates the sum of the means and store its value in **mean**.

- The inputs to the function f are data1 and data2. We call the function f by adding the command **f(data1,data2)** and store the returned value in **mean.value**.
- Notice that R includes a + sign when it adds a line break. You **should not** type in the + sign when you write the code.

Plotting points and graphs of functions
R uses the command **plot()** to plot points and functions. We wish to create two vectors x and y and plot y against x:

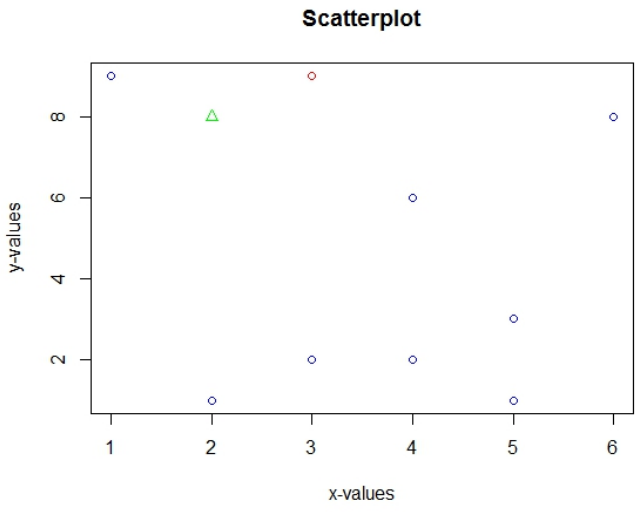
```
> x=c(2,4,6,1,3,5,4,5,3)
> y=c(1,6,8,9,2,3,2,1,2)
> plot(x,y,main="Scatterplot",xlab="x-values",ylab="y-values",col="blue")
```

Explanation. The code can be explained as follows:

- Adding the entry main=" " gives a title.
- Adding the entry xlab=" " labels the x-axis.
- Adding the entry ylab=" " labels the y-axis.
- Adding the entry **col** colors the points in a specified color.

We can add points to the plot as in

```
> points(3,9,col="red")
> points(2,8,pch=2,col="green")
```



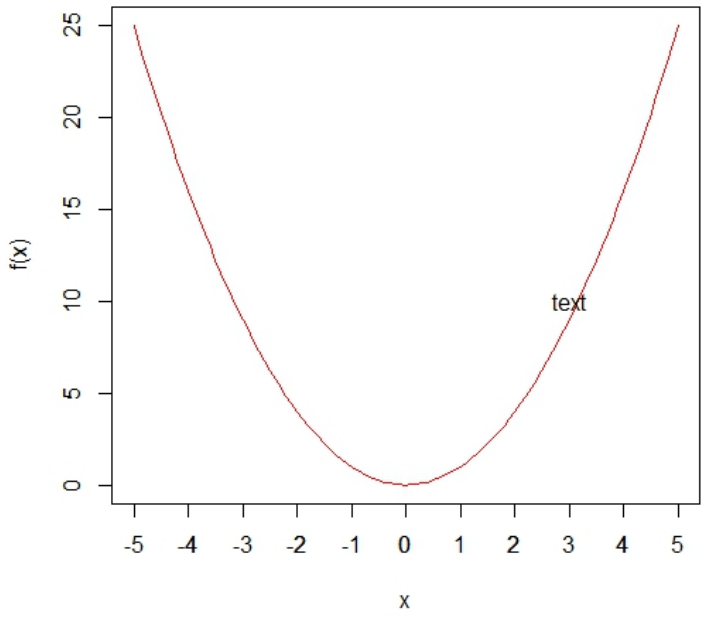
This plot is called a scatterplot and will be explained more thoroughly in a later chapter.

Explanation. The code can be explained as follows:

- The first point added has coordinates $(3, 9)$ and is plotted in red.
- The second point added has coordinates $(2, 8)$ and is plotted in green. The entry **pch** specifies the type of point to use (i.e. circle, plus, triangle, etc.). The range of possible values are 0-25 for symbols and 32-255 for characters. The default is 1.

To graph the function $f(x) = x^2$ from $x = -5$ to $x = 5$, we can do the following:

```
> x=seq(-5,5,0.1)
> f=function(x){x^(2)}
> plot(x,f(x),type="l",col="red")
```



Explanation. The code can be explained as follows:

- We create the function f defined on $-5, -4.9, ..., 4.8, 4.9, 5$.
- We plot the function f and add the entry **type="l"**. (Note that this is the letter "l"). This will draw a line between the points defined by the sequence x. The default in R is **type="p"** which plots points.

We can specify the tick marks and add labels to the graph as illustrated using the graph above:

```
> axis(1,at=seq(-5,5,1))
> text(3,10,labels="text")
```

Explanation. The code can be explained as follows:

- The "1" in the command **axis(1,at=seq(-5,5,1))** indicates that the axis we are adding tick marks to is the x-axis. The tick marks are located at the x-values

$$-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5$$

  which is specified by the entry **at=seq(-5,5,1)**.

- The command **text(3,10,labels="text")** adds a text or character, positioned at the coordinate $(3, 10)$.

for loops
A *for loop* is on the form

```
for(i in expression_1 ){expression_2},
```

where **expression**_1 is a vector of indexed values which is often a sequence, and **expression**_2 might consists of several sub expressions given in terms of **i**. For each **i** running through the indexed values in **expression**_1, a value for the expression_2 is calculated. Here is an example of a **for-loop**: We wish to calculate the product

$$\prod_{k=1}^{i}(1 - k/9) \text{ for } i = 1, 2, ..., 10,$$

and enter the results into a data frame. Proceed as follows:

```
> x=numeric(10)
> for (i in 1:10){x[i]=prod(1-(1:i)/9)}
> i=c(1:10)
> data.frame(i,x)
    i            x
1   1 0.8888888889
2   2 0.6913580247
3   3 0.4609053498
4   4 0.2560585277
5   5 0.1138037901
6   6 0.0379345967
7   7 0.0084299104
8   8 0.0009366567
9   9 0.0000000000
10 10 0.0000000000
```

Explanation. The code can be explained as follows:

- The command **x=numeric(10)** initializes the vector x with 0 in each of its 10 entries.

- The entry **for(i in 1:10)** indicates that we run through the loop for values indexed by $1, 2, ...10$. For each of the 10 run, we compute a value **x[i]** of the product.

- We enter the values of **x[i]** into a data frame for i=1,2,...,10 by adding the command **data.frame(i,x)**.

Reading data from text files
If we have a large dataset we will usually import this data into R from an external file rather than enter the data manually into R. There are several functions in R that can be used to read in data. If the data are stored in a data frame, which is highly recommended, we can use the functions **read.table()** or **read.delim()**. The data should be entered into a file using for example *Notepad* for Windows or *emacs* for unix/Linux. The file should be saved as a text file. We can also enter the data into an excel file and convert this file into a text-file.

We show here an example on how to read an external file into R. The file is an excel-file called **flowers.xlsx** which can be found in the folder called **Introduction**. We convert it into a text-file, **flowers.txt**. The file has the following content:

```
Price   Type
10      Roses
8       Tulips
4       Heliconia
9       Roses
9       Tulips
8       Tulips
```

Here are the steps to follow:

- Save the excel-file called **flower.xlsx** as **flower.txt** and store the file in the folder called **Introduction**. (In Windows, you can create a text file by hitting **save as** and select **Text(MS-DOS)**.)

- To read the data into R and store the data in the data frame called z, we do:

```
> z=read.table("C:/Introduction/flowers.txt",header=T)
> z
  Price      Type
1    10     Roses
2     8    Tulips
3     4 Heliconia
4     9     Roses
5     9    Tulips
6     8    Tulips
```

  Notice that we are using forward slashes (/) in the command **read.table()**. Also notice that your directories are different from what is given here depending on where you store you files on your computer. To find the location of the file on your computer, right click on the file **flowers.txt** and go to **properties**. You can then see the location of the file. Remember to change the backslashes (\) into forward slashes (/) in R. The directory of the file should be given in quotes in **read.table()**.

The option **header=T** specifies that the first line is a header and that the names of the variables in the file will be preserved in R.

Alternatively, we can use the function **read.delim()** as in

```
> z=read.delim("C:/Introduction/flowers.txt")
```

By default this command reads Tab-delimited files.

If the data are separated with a comma, we can use the function **read.csv()** to read in the data [2].

Built-in data sets
R has several built-in data sets. To see the available data sets, enter

```
> data()
```

To access the built-in data set **Orange**, enter

```
> data(Orange)
```

Library and Packages (For more advanced R-users).
Data sets and R-functions are stored in libraries of packages. To load the package **boot**, enter

```
> library(boot)
```

Some of the packages and the basic statistical functions and data sets are part of the R source code and are immediate available when R is installed. Other packages can be downloaded from CRAN. To install a particular package, say the package **plotrix**, which contains functions that can perform 3D plot, go to the menu in R and click on **Packages** and select **Install package(s)**. Then choose the CRAN mirror. For example, we could choose **USA(IA)**. Then select the package **plotrix**. Now we can enter

```
> library(plotrix)
```

and use the R-functions contained in this package. We will return to this package in chapter 1.

Help in R
R has an extensive online documentation. To get help in R, enter

```
> help.start()
```

To get help with a particular function, say, **t.test**, enter

```
> help(t.test)
```

or

```
> ??t.test
```

References

[1] D. M. Bates and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*, Wiley, Appendix A1.3, 1988

[2] P. Dalgaard, *Introductory Statistics with R*, Second Edition, Statistics and Computing, Springer, 2008

[3] M. A. Treloar, *Effects of Puromycin on Galactosyltransferase in Golgi Membranes*, M.Sc. Thesis, U. of Toronto, 1974

[4] W. N. Venables, D. M. Smith, and the R Core Team, *An Introduction to R, Notes on R: A Programming Environment for Data Analysis and Graphics*, Version 3.0.2 (2013-09-25)

[5] J.Verzani, *simpleR - Using R for Introductory Statistics*, Chapman & Hall/CRC, The R Series, 2005

00