

Introduction to R

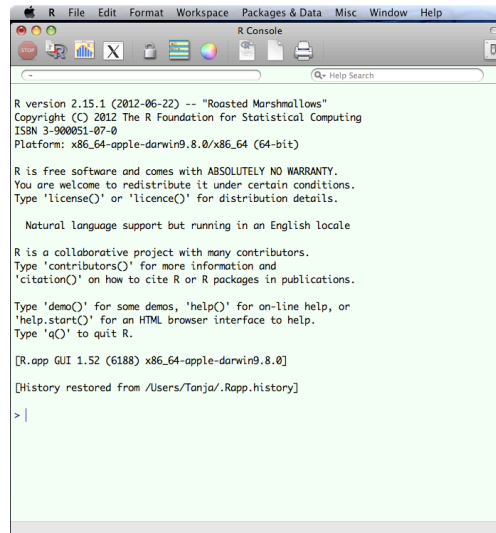
Tatjana Kecojević

September 22, 2016

The purpose of this section is to provide a basic overview of and introduction to R language and its environment for statistical computing and graphics. R is a public domain language for data analysis, fast becoming the lingua franca of quantitative research with some 9220 free specialised packages. R is a free, open-source data analysis package available for Windows, Mac OS X, and Unix/Linux systems, developed and maintained by R Development Core Team. You can download R from: <http://cran.r-project.org/>.

1 Getting Started

To run R you need to click on the R icon on your computer. When R is launched you will see a single window called R Console (Figure 1).



```
R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"
Copyright (C) 2012 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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.

[R.app GUI 1.52 (6188) x86_64-apple-darwin9.8.0]
[History restored from /Users/Tanja/.Rapp.history]

> |
```

Figure 1: *R Console*

The simplest way to get help in R is to click on the Help option on the toolbar of the R console. Alternatively, to get help type

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

which will provide a web-based interface to the help system.

There is a vast array of guidebooks on the web that will help you with R, but your first port of call should be CRAN's website

<http://cran.r-project.org/>

where you can find a number of manuals. Visiting CRAN's website is useful as you will be able to search through *Frequently Asked Questions* (FAQs) and *R News* which will keep you up to date with new useful articles, book reviews and dates of forthcoming releases.

To begin with, we can use R as a calculator:

```
> 5+9
[1] 14
> 13-2
[1] 11
> 18/6
[1] 3
> 4*7
[1] 28
> (5-3)^2/4
[1] 1
> 9^(1/2)*4
[1] 12
```

Note that you don't have to type the equals sign and that each answer has `[1]` in front. The `[1]` indicates that there is only one number in the answer. If the answer contains more than one number it uses numbering like this to indicate where in the 'group' of numbers each one is.

R provides a number of specialised data structures we will refer to as **objects**. To refer to an object we use a symbol. You can assign any object using the assignment operator `<-`, which is a composite made up from 'less than' and 'minus', with *no space between them!* Thus, we can create scalar constants, which we refer to as variables, and perform mathematical operations over them.

```
> x<-5
> y<-6
```

You can use objects in calculation in exactly the same way as as you have already seen numbers being used earlier:

```
> x+y
[1] 11
```

and you can store the results of the calculation done with the objects in another object:

```
> z<-x*y
> z
[1] 30
```

BUT, remember!!! Operator <- is a composite made up from 'less than' and 'minus', with no space between them!!!! Try to type:

```
x< -5
y< -6
and see what happens.
```

After you've created some objects in R you can get a list of them using `ls()` function:

```
> ls()
[1] "x" "y" "z"
```

You can also remove an object from R's 'workspace' using `rm()` function.

```
> rm(z)
> ls()
[1] "x" "y"
```

R is not like other conventional statistical packages like SAS, Minitab, SPSS, to name a few. It is more of a programming language designed for conducting data analyses. It comes with a vast number of ready-made blocks of code that will enable you to manipulate data, perform intricate mathematical calculations with data, carry out an array of statistical analysis ranging from simple to complex to extremely complex and it will facilitate the creation of fantastic graphs. These pre-made blocks of code are known as **functions**.

R has all the standard mathematical functions that you might ever need: `sin`, `cos`, `tan`, `asin`, `atan`, `log`, `log10`, `exp`, `abs`, `sqrt`, `factorial`... To use them, all you need to do is to type the function and put the name of the object (argument) you would like to use the function for in brackets.

```
> sqrt(144)
[1] 12
> log10(8)
[1] 0.90309
> log10(100)
[1] 2
> log(100)
[1] 4.60517
> exp(1)
[1] 2.718282
> pi
[1] 3.141593
> sin(pi/2)
[1] 1
> abs(-7)
```

```
[1] 7
> factorial(3)
[1] 6
> exp(x)
[1] 148.4132
> log(y, 2)
[1] 2.584963
```

You can use expression as argument of a function:

```
> trunc(x^2+z/y)
[1] 30
> log((100*x-y^2)/z)
[1] 2.738687
```

You can have nested functions and you can use functions in creating new objects:

```
> round(exp(x), 2)
[1] 148.41
> p<-abs(floor(log((100*x-y^2)/exp(z))))
> p
[1] 24
```

2 Vectors and Matrices

When analysed data you are more likely to be working with lots of numbers/variables. It would be much more convenient to keep all of those number/variables as an object. Variables can be of a different type: logical, integer, double, string are some examples. Variables with one or more values *of the same type* are **vectors**. Hence, a variable with a single value (known to us as a scalar) is a vector of length 1. We can assign to vectors in many different ways:

- generated by R using the colon symbol (:) as a sequence generated operator or by using built in function `rep` for replicating the given number for a given number of times

```
> x<-1:10
> x
[1] 1 1 1 1 1 1 1 1 1 1
> x<-rep(1,10)
> x
[1] 1 1 1 1 1 1 1 1 1 1
```

- generated by the user by using concatenation function `c` or using function `scan` that allows you to enter one number at a the time, When using `scan` to indicate that the input is complete you need to press the Enter button.

```

> x<-c(2, 6, 4, 2, 3, 7, 1, 5, 9, 8)
> x
[1] 2 6 4 2 3 7 1 5 9 8
> y<-scan()
1: 8
2: 1
3: 4
4: 7
5: 3
6:
Read 5 items
> y
[1] 8 1 4 7 3

```

- created as a sequence of numbers. For example to generate a sequence of numbers from 1 to 10, with increments of 0.2 type

```

> seq(1,10,0.2)
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6
[10] 2.8 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4
[19] 4.6 4.8 5.0 5.2 5.4 5.6 5.8 6.0 6.2
[28] 6.4 6.6 6.8 7.0 7.2 7.4 7.6 7.8 8.0
[37] 8.2 8.4 8.6 8.8 9.0 9.2 9.4 9.6 9.8
[46] 10.0

```

R can easily perform arithmetic with vectors as it does with scalars. You have already seen that R contains a number of operators. There is a list of some of them you are likely to use the most:

- arithmetic: `+`, `-`, `*`, `\`, `%%`
- relational: `>`, `>=`, `<=`, `==`, `!=`
- logical: `!`, `&`, `|`

Thus, just as we can use those operators over the scalars we can use them when dealing with the vectors and/or a combination of both.

```

> x<-rep(1,10)
> y<-1:10
> x
[1] 1 1 1 1 1 1 1 1 1 1
> y
[1] 1 2 3 4 5 6 7 8 9 10
> c(x, y)
[1] 1 1 1 1 1 1 1 1 1 1 1 2 3 4 5 6
[17] 7 8 9 10
> x+y
[1] 2 3 4 5 6 7 8 9 10 11
> x+2*y
[1] 3 5 7 9 11 13 15 17 19 21
> x^2/y

```

```

[1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
[6] 0.1666667 0.1428571 0.1250000 0.1111111 0.1000000
> z<-(x+y)/2
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
> z<-c(z, rep(1, 3), c(100, 200, 300))+1
> z
[1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
[9] 6.0 6.5 2.0 2.0 2.0 101.0 201.0 301.0
> p<-2.5
> z*p
[1] 5.00 6.25 7.50 8.75 10.00 11.25 12.50
[8] 13.75 15.00 16.25 5.00 5.00 5.00 252.50
[19] 502.50 752.50

```

It can get messy with all the objects you create. To keep your house in order it is useful to check from time to time what is there. Remember, to list all the objects you have created, use function `ls()`:

```

> ls()
[1] "p" "x" "y" "z"

```

To access a specific element of a vector you would use index inside a single square bracket `[]` operator. The following shows how to obtain a vector member. The vector index is 1-based, thus use the index position 4 to access the fourth element.

```

> y<-c(9, 3, 7, 2, 9, 2, 1, 5, 4, 6)
> y
[1] 9 3 7 2 9 2 1 5 4 6
> y[4]
[1] 2
> y[27]
[1] NA

```

Note that missing values in R are represented by the symbol `NA` (*not available*) or `NaN` (*not a number*) for undefined mathematical operations. Here, `NA` would be shown if an index is out-of-range.

You can also obtain a desirable selection of the elements of a vector by specifying a query within the index brackets `[]`:

```

> y
[1] 9 3 7 2 9 2 1 5 4 6
> y[y>5]
[1] 9 7 9 6
> y[y>10]
numeric(0)
> y[y!=2]
[1] 9 3 7 9 1 5 4 6

```

In R you can evaluate functions over the entire vectors which helps to avoid from looping.

```
> max(y)
[1] 9
> range(y)
[1] 1 9
> mean(y)
[1] 4.8
> var(y)
[1] 8.4
> sort(y)
[1] 1 2 2 3 4 5 6 7 9 9
> cumsum(y)
[1] 9 12 19 21 30 32 33 38 42 48
```

To obtain a description of a function you need to type a question mark, `?`, in front of the name of the function. You might find this particularly useful when you start applying more complicated functions, as help will often provide you not only with the detailed description of the function's input/output arguments, but practical illustrative examples on how the function can be used and applied.

```
> ?mean
```

When data is arranged in two dimensions rather than one we have **matrices**. In R function `matrix()` creates matrices:

```
> ma1 <- matrix(c(1, 0, -20, 0, 1, -15, 1, -1, 0), nrow=3,
+ ncol=3, byrow=T)
> ma1
      [,1] [,2] [,3]
[1,]     1     0  -20
[2,]     0     1  -15
[3,]     1    -1     0
```

The individual numbers in a matrix are called the *elements* of the matrix. Each element is uniquely defined by its particular *row number* and *column number*. To determine the dimensions of a matrix use function `dim()`

```
> dim(ma1)
[1] 3 3
```

An element at the i^{th} row, j^{th} column of a matrix can be accessed by indexing inside square bracket operator `[i, j]`.

```
> ma1[2,3]
[1] -15
```

The entire i^{th} row or entire j^{th} column of a matrix can be extracted as

```
> ma1[3, ]
```

```
[1] 1 -1 0
> mal[,2]
[1] 0 1 -1
```

Standard scalar algebra, which deals with operations on single numbers, has a set of well established rules for handling manipulations involving addition, subtraction, multiplication and division. In a broadly similar fashion a set of rules has been developed to enable us to manipulate matrices. However, introducing those rules is beyond the scope of this session and they are covered in the complementary set of notes *Matrix Methods in R*.

2.1 Data Types

The examples we have used so far are all dealing with numbers (quantitative numerical data). Those of you familiar with programming will know that **numerical objects** can be classified as *real*, *integer*, *double* or *complex*. To check if an object is numeric and of what type it is, you can use `mode()` and `class()` functions respectively.

```
> x<-1:10
> mode(x)
[1] "numeric"
> class(x)
[1] "integer"
```

In R to enter **strings of characters** as objects you need to enter them using quote marks around them. By default R expects all inputs to be numeric and unless you use quote marks around the strings you wish to enter, it will consider them as numbers and subsequently will return an error message.

```
> x<-c("UK", "Spain", "France", "Germany", "Italy")
[1] "UK"      "Spain"    "France"   "Germany"  "Italy"
> mode(x)
[1] "character"
```

It is common in statistical data to have *attribute* also known as *categorical variables*. In R such variables should be specified as **factors**. Attribute variable has a set of *levels* indicating possible outcomes. Hence, to deal with `x` as an attribute variable with five levels we need to make it a factor in R.

```
> x<-factor(x)
> x
[1] UK      Spain    France   Germany  Italy
Levels: France Germany Italy Spain UK
```

Note that R codes the factor levels in their alphabetical order. However, attribute variables are usual coded and you would usually enter them as such.

```
> quality <- factor(c(3, 3, 4, 2, 2, 4, 0, 1))
> levels(quality)
```



```
[1] "0" "1" "2" "3" "4"
> quality
[1] 3 3 4 2 2 4 0 1
Levels: 0 1 2 3 4
```

You might need to deal from time to time with **logical** data type. This is when something is recorded as *TRUE* or *FALSE*. It is most likely that you would use this data type when checking what type of data the variable is that you are dealing with. For example

```
> is.numeric(x)
[1] FALSE
> is.factor(x)
[1] TRUE
```

3 Data Frames

Statistical data usually consists of several vectors of equal length and of various types that resemble a table. Those vectors are interconnected across so that data in the same position comes from the same experimental unit, ie. observation. R uses **data frame** for storing this kind of data table and it is regarded as primary data structure.

Let us consider a study of share prices of companies from three different business sectors. As part of the study a random sample (n=15) of companies was selected and the following data was collected:

```
> share_price<-c(880, 862, 850, 840, 838, 799, 783, 777,
+ 767, 746, 692, 689, 683, 661, 658)
> profit<-c(161.3, 170.5, 140.7, 115.7, 107.9, 135.2,
+ 142.7, 114.9, 110.4, 98.9, 90.2, 80.6, 85.4, 91.7, 137.8)
> sector<-factor(c(3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1,
+ 1, 1)) # 1:IT, 2:Finance, 3:Pharmaceutical
#
> share_price
[1] 880 862 850 840 838 799 783 777 767 746 692 689
[13] 683 661 658
> profit
[1] 161.3 170.5 140.7 115.7 107.9 135.2 142.7 114.9
[9] 110.4 98.9 90.2 80.6 85.4 91.7 137.8
> sector
[1] 3 3 3 3 3 2 2 2 2 2 1 1 1 1 1
Levels: 1 2 3
```

Rather than keeping this data as a set of individual vectors in R, it would be better to keep whole data as a single object, i.e. data frame.

```
> share.data<-data.frame(share_price, profit, sector)
> share.data
```

	share_price	profit	sector
1	880	161.3	3
2	862	170.5	3
3	850	140.7	3
4	840	115.7	3
5	838	107.9	3
6	799	135.2	2
7	783	142.7	2
8	777	114.9	2
9	767	110.4	2
10	746	98.9	2
11	692	90.2	1
12	689	80.6	1
13	683	85.4	1
14	661	91.7	1
15	658	137.8	1

Individual vectors from the data frame can be accessed using \$ symbol:

```
> share.data$sector
[1] 3 3 3 3 3 2 2 2 2 2 1 1 1 1 1
Levels: 1 2 3
```

This notation for accessing variables in data frames can become rather gruelling when having to type it repeatedly. By attaching data frame to R *search path*, variables from that data frame can be accessed by simply giving their names.

```
> attach(share.data)
```

From now on if you type the name of a variable in the attached data frame you do not have to tell R the name of the data frame in which it can be found. You can always check what is in *system's search path* by using `search()`:

```
> search()
[1] ".GlobalEnv"          "share.data"
[3] "tools:RGUI"          "package:stats"
[5] "package:graphics"    "package:grDevices"
[7] "package:utils"        "package:datasets"
[9] "package:methods"     "Autoloads"
[11] "package:base"
```

`.GlobalEnv` is the *workspace* and `package:base` is the *system library*, i.e. *package* where all standard functions are defined. The rest of the so called *base packages* contain the basic statistical routines. Assuming that you are connected to the internet, you can install a package using `install.packages()`. Note that if you are working behind a firewall system, it will try to prevent you from adding new packages. To enable installation of new packages type the following:

```
> setInternet2()
> chooseCRANmirror()
```

You will be asked to select the *mirror* nearest to you for fast downloading (any UK would be fine), then everything else is automatic.

```
install.packages("ggplot2")
```

Once you have installed a package to use it you have to load it to *system's search path* by simply typing

```
> library(ggplot2)
```

Many packages inside and outside the standard *R* distribution, come with built-in data sets. For example, *ggplot* contains *economics* data set:

```
> data(economics)
> economics[1:5,]
      date    pce    pop psavert uempmed unemploy
1 1967-06-30 507.8 198712    9.8    4.5    2944
2 1967-07-31 510.9 198911    9.8    4.7    2945
3 1967-08-31 516.7 199113    9.0    4.6    2958
4 1967-09-30 513.3 199311    9.8    4.9    3143
5 1967-10-31 518.5 199498    9.7    4.7    3066
```

However, often you will have to load data from an out source, such as spreadsheet or database or maybe another statistical package. Loading data into *R* is not difficult and the key command you will use will be `read.table`. Commonly, data is saved in a text file (for example: *mydata.txt*) and to load this data into *R* you would type the following:

```
mydata <- read.table("c:/mydata.txt", header=TRUE)
```

Sometimes you will still find it easier to manipulate and organise your data using *EXCEL*. If this is the case you can save the spreadsheet as a *csv* file (Comma Separated Values file) that can be loaded into *R* by using `read.csv` command.

```
mydata <- read.csv("c:/mydata.csv", header=TRUE)
```

Note that when you ask *R* to load a data file you need to specify the exact path of where the file is saved by giving the full path name in quotes.

Remember to keep your house in order! When you are done with an attached data frame you should remove it from *system's search path* using `detach()` .

```
> detach(share.data)
> search()
[1] ".GlobalEnv"          "package:ggplot2"
[3] "tools:RGUI"           "package:stats"
[5] "package:graphics"    "package:grDevices"
[7] "package:utils"        "package:datasets"
[9] "package:methods"     "Autoloads"
[11] "package:base"
```

If you are going to quit your R session right away this is not necessary as quitting detaches everything that was attached.

4 Graphs

R has many great functions for producing high quality plots. To get the idea of what type of plots it is possible to produce in R type the following:

```
> demo(graphics)
```

and keep pressing the return button on your keyboard until you scan through them all.

Let us use some of the basic functions for creating plots that will open a graph window in which the plot will be shown. Type the following and see what happens. Try to explain to yourself what the graph is showing.

```
> y<-rnorm(1:30, 2, 4)
> plot(1:30, cumsum(y))
> lines(1:30, cumsum(y), type="l", col="red")
> hist(y)
> hist(y, 10)
> x<-seq(-4, 4, 0.1)
> plot(x, dnorm(x), type="l", lty=2, col="red",
+ ylab="Probability Density", xlab="x")
> boxplot(x, col="orange", horizontal=T)
```

Graphs are usually used to provide visualisation of specific features contained within the data, thus they can help us to communicate information. Often we can inspect the main characteristic of the data by using the appropriate plots. Questions that you should seek the answers to before embarking on producing any kind of plot are:

- What type of information one (or more) variables are providing: bar chart or histogram?
- How two variables are related: boxplots or scatterplots?

In other words, to use the appropriate plot we need to understand the problem under investigation for which the data is collected and to recognise clearly what each of the variables in the data is measuring.

5 Your Turn

Before you start remove everything from your *workspace* in R by typing:

```
> rm(list=ls())
```

Practise by doing the following set of exercises:

1. Carry out the following calculations:
 - 4^5
 - Add 7 to 9 and then divide the answer by 2
 - Subtract π from 5 and raise answer to the power of 3 and then add 2.79
 - Divide 6^8 by 4.7 and then divide the answer by 2
2. Create an object called "X1", which is 99.
3. Create an object called "X2", which is answer to 4^5 .
4. Multiply X1 and X2 and calculate the square root of the answer which should be stored as a new object called "X3".
5. Calculate the *log* to the base of 10 of X1 and round it to the second decimal place.
6. Create vectors called "x1" and "x2", where vector x1 consists of numbers: 1, 4, 7, 9, 11, 12, 13, 15 and 18 and vector x2 of numbers: 1, 1, 1, 2, 2, 2, 3, 3, 3.
7. Subtract x2 from x1.
8. Create a new vector called "x3" by combining vectors x1 and x2.
9. Calculate mean and variance of x3.
10. Calculate medians for the three vectors.
11. Create a matrix called "m1" with the following elements:

$$m1 = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}$$
12. Use a subscript to find the 2^{nd} number in vector x1 and x2 and element in the 2^{nd} row and 3^{rd} column in matrix m1.
13. Add the 5^{th} number in vector x1 to the element in matrix m1 which is in 1^{st} row and 1^{st} column.
14. Calculate the mean and the variance of all numbers in x3 that are less than 13.
15. Calculate the mean and the variance of all numbers in x3 that are greater than or equal to 3 and that are less than 12.

6 What is Data

There are many situations in modern business and science where data is collected and analysed. The key ideas of data analysis are important in understanding the information

provided by such data. In this section we will look into a set of methods to enable data to be explored with the objective of summarising and understanding the main features of the variables contained within the data.

We will start by defining the population. The **population** is the set of all people/objects of interest in the study being undertaken. Usually populations are very large, and in some cases may be conceptual in the sense that they cannot be completely enumerated physically. The majority of data analysis is carried out on a **sample** drawn from the population, and the fundamental problem is to use sample data to draw inferences about the population.

In statistical terms the whole data set is called the population. This represents *perfect information* however in practice it is often impossible to enumerate the whole population. The analyst therefore takes a sample drawn from the population and uses this information to make judgements (inferences) about the population.

Clearly if the results of any analysis are based on a sample drawn from the population, then if the sample is going to have any validity, then the sample should be chosen in a way that is fair and reflects the structure of the population. The process of sampling to obtain a representative sample is a large area of statistical study. The simplest model of a representative sample is a *random sample*, a sample chosen in such a way that each item in the population has an equal chance of being included in the sample. As soon as sample data is used, the information contained within the sample is *imperfect* and depends on the particular sample chosen. The key problem is to use this sample data to draw valid conclusions about the population with the knowledge of and taking into account the *error due to sampling*.

Usually the data will have been collected in response to some design problem, in the hope of being able to glean some pointers from this data that will be helpful in the analysis of the problem. Data is commonly presented to the data analyst in this way with a request to analyse the data.

Before attempting to analyse any data, the analyst should:

- Make sure that the problem under investigation is clearly understood, and that the objectives of the investigation have been clearly specified. The only way to obtain this information is to ask questions, and keep asking questions until satisfactory answers have been obtained.
- Before any analysis is considered the analyst should make sure that the individual variables making up the data set are clearly understood.

A starting point is to examine the characteristics of each individual variable in the data set. The way to proceed depends upon the type of variable being examined.

The variables can be one of two broad types:

- 1) **Attribute variable**: has its outcomes described in terms of its characteristics or attributes;
- 2) **Measured variable**: has the resulting outcome expressed in numerical terms.

6.1 Statistical Distribution

The concept of the **statistical distribution** is central to statistical analysis. This concept relates to the population and conceptually assumes that we have perfect information, the exact composition of the population is known. However, as you are most likely to deal with the sample data you will be looking at sample distribution, based on which you will be drawing conclusions about the population.

If you want to look at the distribution of an attribute variable, you will look at the frequency of occurrence of each level using a bar chart (Figure 2). If the variable under discussion is a measured type then distribution of this variable across its possible range of values may look like in Figure 3, where the area under the curve from one height value to another measures the relative proportion of the population having height in that range. In other words, the graph in Figure 3 indicates how the range of heights is distributed over the possible range of values.

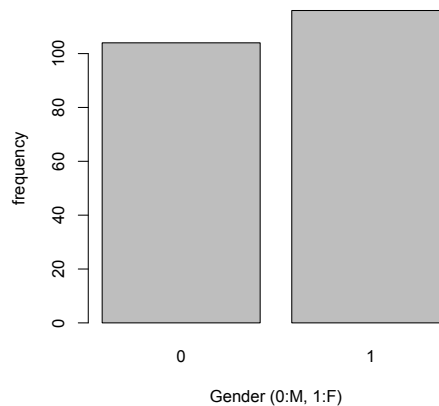


Figure 2: Bar Chart for attribute variable Gender

A statistical distribution for a measured variable can be summarised using three key descriptions:

- the centre of the distribution
- the width of the distribution
- the symmetry of the distribution

The common measures

- of the centre of a distribution are the *mean* and the *median* and
- of the width of a distribution are the *standard deviation* and the *Inter-Quartile Range (IQR)*.

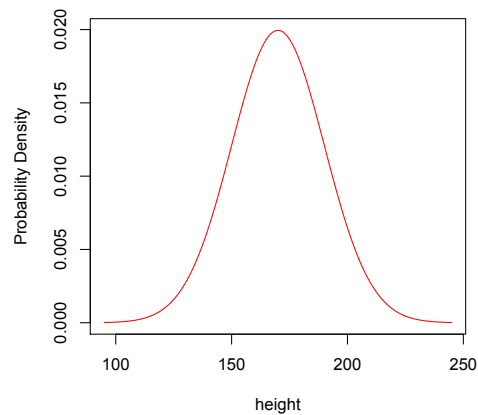


Figure 3: *Density Plot for measured variable Height*

Symmetry of the distribution is assessed by measuring *skewness* and "peakedness" *kurtosis*. However, the simplest way to assess symmetry is to compare the mean and the median. For a perfectly symmetric distribution the mean and the median will be exactly the same.

Let us look at the following study of company share price:

```
#=====
#
# A business analyst is studying share prices of companies
# from three different business sectors. As part of the
# study a random sample (n=60) of companies was selected
# and the following data was collected:
#
#- Share_Price: The market value of a company share
#- Profit: The company annual profit
#- RD: Company annual spending on research and development
#- Turnover: Company annual total revenue
#- Competition: A variable coded:
# 0 if the company operates in a very competitive market
# 1 if the company has a great deal of monopoly power
#- Sector: A variable coded:
# 1 if the company operates in the IT business sector;
# 2 if the company operates in the Finance business sector;
# 3 if the company operates in the Pharmaceutical business
# sector.
#- Type: A variable coded:
```



```

# 0 if the company does business mainly in Europe;
# 1 if the company trades globally.
#
#=====

> companyd <- read.csv("SHARE_PRICE.csv", header=T)
# load package {car}
> library(car)
# we want a sample of 'some' of the observations from
# the data set.
> some(companyd)
  Share_Price Profit    RD Turnover Competition Sector Type
7          808  102.0  91.3    212.0           1      3     1
16         739  130.1 102.5    280.7           1      3     1
18         737  100.4  71.2    201.4           0      3     1
30         602   69.8  63.6    155.8           1      3     1
34         587  103.1 101.9    187.7           0      3     0
43         513   50.9  48.8     92.5           0      1     0
47         494   37.5  87.5     34.5           0      1     0
51         443   28.9  88.6     64.5           0      2     1
52         433   60.7  77.9    126.4           0      2     1
59         148    3.9  39.2     46.6           0      1     1
# -----
# To check how big the data set is, we can use function dim()
> dim(companyd) # dimension of company data
[1] 60  7
# Data set has 60 observations and 7 variables
#=====
#
> summary(companyd) # Get the key summary statistics for each
+ variable
  Share_Price      Profit          RD      Turnover
Min.   :101.0  Min.   : 2.90  Min.   : 39.20  Min.   : 30.3
1st Qu.:501.2  1st Qu.: 59.73  1st Qu.: 75.78  1st Qu.:112.3
Median :598.5  Median : 88.85  Median : 90.60  Median :173.5
Mean   :602.8  Mean   : 84.76  Mean   : 89.64  Mean   :170.2
3rd Qu.:739.8  3rd Qu.:106.62  3rd Qu.:104.15  3rd Qu.:216.6
Max.   :880.0  Max.   :170.50  Max.   :152.60  Max.   :323.3

  Competition      Sector      Type
Min.   :1  Min.   :0.0
1st Qu.:0.0  1st Qu.:1  1st Qu.:0.0
Median :0.5  Median :2  Median :0.5
Mean   :0.5  Mean   :2  Mean   :0.5
3rd Qu.:1.0  3rd Qu.:3  3rd Qu.:1.0
Max.   :1.0  Max.   :3  Max.   :1.0

```

```

#
#
# BUT!!! Variables: 'Comparison', 'Sector' and 'Type' are
# attribute variables?! We need to let R know this!
# To encode a measured variable as an attribute variable
# we can use function factor(variable_name).
> companyd[, 5] <- factor(companyd[, 5])
> companyd[, 6] <- factor(companyd[, 6])
> companyd[, 7] <- factor(companyd[, 7])
#
> summary(companyd)
  Share_Price  Profit      RD      Turnover
Min.   :101.0  Min.   :  2.90  Min.   : 39.20  Min.   : 30.3
1st Qu.:501.2  1st Qu.: 59.73  1st Qu.: 75.78  1st Qu.:112.3
Median :598.5  Median : 88.85  Median : 90.60  Median :173.5
Mean   :602.8  Mean   : 84.76  Mean   : 89.64  Mean   :170.2
3rd Qu.:739.8  3rd Qu.:106.62  3rd Qu.:104.15  3rd Qu.:216.6
Max.   :880.0  Max.   :170.50  Max.   :152.60  Max.   :323.3

Competition Sector  Type
0:30             1:20  0:30
1:30             2:20  1:30
                3:20

# -----
# Alternatively, to get an individual summary for measured variable
# type:
> sapply(companyd[,1:4], summary)
      Share_Price Profit      RD Turnover
Min.      101.0   2.90  39.20    30.3
1st Qu.   501.2  59.73  75.78   112.4
Median    598.5  88.85  90.60   173.5
Mean      602.8  84.76  89.64   170.2
3rd Qu.   739.8 106.60 104.20   216.6
Max.      880.0 170.50 152.60   323.3
# -----
#
# To focus on the centre of the distributions for the measured
# variables you can ask only for the rows showing mean and
# median to be displayed.
> sapply(companyd[,1:4], summary)[3:4, ]
      Share_Price Profit      RD Turnover
Median    598.5   88.85  90.60   173.5
Mean      602.8   84.76  89.64   170.2
#
# To observe spread of the data we can use standard deviation

```

```

# and/or Inter Quartile Range
> sapply(companyd[,1:4], sd)
Share_Price      Profit          RD      Turnover
177.28461      37.76443      24.13231      75.72712
> sapply(companyd[,1:4], IQR)
Share_Price      Profit          RD      Turnover
238.500      46.900      28.375      104.250
#
#=====
#
# To explore the distributions of the variables visually
# you should use the appropriate graphs.
# Usually you use a pie chart or a bar plot if you want to
# visualise an attribute variable.
#
> barplot(table(companyd[, 5]), xlab="Commpetition",
+ ylab="frequency")
> barplot(table(companyd[, 7]), xlab="Type", ylab="frequency")
#
> pie(table(companyd[, 6]), labels=names(companyd$Sector),
+ col=c(3, 7, 5), main="Sector")
#
# Histogram is appropriate when you have a measured variable
# to graphically explore.
#
> hist(companyd[, 1], xlab="Share Price", main="Histogram of
+ Share Price", col="gray")
#
# If you would like to see the density smoothing of the
# histogram, on your histogram you will plot the
# probability density rather than the frequency of the
# measured variable, over which you can superimpose
# a kernel density smoothing line.
#
> hist(companyd[, 1], xlab="Share Price", main="Histogram of
+ Share Price", col="gray", prob=T)
> lines(density(companyd[, 1]), col="red")
# Or you can have a kernel density smoothing as an
# individual plot.
> par(mfrow=c(2, 2)) # splits the graph window into 2 rows
+ and 2 columns
> plot(density(companyd[,1]), main="Density Function of
+ Share Price")
> plot(density(companyd[,2]), main="Density Function of Profit")
> plot(density(companyd[,3]), main="Density Function of R&D")
> plot(density(companyd[,4]), main="Density Function of Turnover")

```

```

> par(mfrow=c(1, 1)) # puts the graph window back onto a single plot
#=====
# To investigate the possible relationship between attribute
# and measured variables you can use a box plot:
#
> boxplot(Share_Price ~ Competition, data = companyd,
+ boxwex = 0.25, main="Share Price vs Competition",
+ xlab = "Share_Price", ylab = "Competition",
+ col=c("palegreen3", "orchid3"), horizontal=T)
#
# If you are interested in analysing a potential
# relationship between measured variables use a scatter
# plot:
#
> plot(companyd$Profit, companyd$Share_Price, cex=.6,
+ main="Scatterplot of Share Price by Profit")
#
#
# Note: the plot() function gives a scatterplot of two
# numerical variables. The first variable listed will be
# plotted on the horizontal axis and the second on the
# vertical axis, ie. you 'feed' as the arguments first
# variable representing X and then variable
# representing Y.
# -----

```

7 Your Turn

Use *birthwt* data from MASS package in R.

- i What type of information variables are providing. Provide key information about each of the variables and use the appropriate plot to illustrate your findings.
- ii How two variables are related: boxplot or scatterplot? Illustrate the potential relationships between the two variables using appropriate graphs.
- iii Write down questions that you could answer with this data.

8 Further Reading

One of the great things about R that makes it so powerful to use, is the freely available excellent documentation that you can access not just from CRAN, but from other websites created by a vast community of R enthusiasts. Here are a couple of them that you can explore for yourself:

Good for quick and useful tips <http://www.ats.ucla.edu/stat/r/>

More comprehensive - for those
of you already familiar with R <http://www.statmethods.net/>

9 Acknowledgment

We would like to thank Ian McGowan and George Rawlings for letting us adapt their teaching material.