MAST90105: Lab and Workshop 1

1 Lab

Goals: (i) Getting started with R and RStudio; (ii) Basic exploratory data analyses; (iii) Basic graphics.

1.1 Introduction

Lately R has become a popular computational tool when comes to statistical applications. An increasing number of researchers in many disciplines, including social and biomedical sciences, choose R for their work. R is not just a statistical computing software, but rather a general purpose computing environment where many classical and modern statistical procedures have been implemented.

R is often the vehicle of choice for research in statistical methodology. Its open source nature allows statisticians to contribute new methods in form of packages which can be downloaded and installed for free at any time. Besides the ordinary packages supplied in the ”base” version of R, hundreds of contributed packages are available through the CRAN web site `http://CRAN.R-project.org`.

The main advantages of R are the fact that R is freeware and that there is a lot of help available online. It is quite similar to other programming packages such as MatLab (not freeware), but more user-friendly than programming languages such as C++ or Fortran. You can use R as it is, but for educational purposes we prefer to use R in combination with the RStudio interface (also freeware), which has an organized layout and several extra options.

1.2 Getting started

1.2.1 Install R and RStudio

To install R on your computer, go to the home website of R `http://CRAN.R-project.org`a nd do the following (assuming you work on a windows computer): (i) click download CRAN in the left bar; (ii) choose a download site; (iii) choose Windows as target operation system; (iv) click base and choose default answers for all questions. After finishing this setup, you should see an ”R” icon on you desktop. Clicking on this would start up the standard interface. We recommend, however, to use the RStudio interface. To install RStudio, go to `http://www.rstudio.com/` and do the following: click `products >RStudio`, click Download RStudio Desktop, click on the installer recommended for your system and run it (choose default answers for all questions).

1.2.2 RStudio layout

The RStudio interface consists of several windows

- Bottom left: console window (also called command window). Here you can type simple commands after the `>` prompt and R will then execute your command. This is the most important window, because this is where R actually does things.
- Top left: editor window (also called script window). Collections of commands (scripts) can be edited and saved. When you do not get this window, you can open it with `File> New R script`. Just typing a command in the editor window is not enough, it has to get into the command window before R executes the command. If you want to run a line from the script window (or the whole script), you can click `Run` or press `CTRL+ENTER` to send it to the command window.

- Top right: workspace / history window. In the workspace window you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been typed before.
- Bottom right files / plots / packages / help window. Here you can open files, view plots (also previous plots), install and load packages or use the help function.

You can change the size of the windows by dragging the grey bars between the windows.

1.2.3 Working directory

Your working directory is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory. Before you start working, please set your working directory to where all your data and script files are or should be stored.

From the command window: `setwd("directoryname")`. For example:

```
> setwd("M:/MyStuff/R/")
```

Make sure that the slashes are forward slashes and that you don't forget the apostrophes. R is case sensitive, so make sure you write capitals where necessary. From the RStudio menus you can go to `Session / Set working directory` or `Tools / Global options / General / Default working directory`.

1.2.4 Libraries

R can do many statistical and data analyses. They are organized in so-called packages or libraries. With the standard installation, most common packages are installed. To get a list of all installed packages, go to the packages window or type `library()`.

Alternatively, in RStudio you can use the console packages window (bottom right). If the box in front of the package name is ticked, the package is loaded (activated) and can be used. There are many more packages available on the R website. If you want to install and use a pack- age (for example, the package called `geometry`) you should: Install the package: click install packages in the packages window and type geometry or type

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

in the command window.

To load the package: check box in front of geometry or type

```
>library(geometry)
```

in the command window.

1.3 Examples of R commands

1.3.1 Using R as a calculator

We can warm up by using R as calculator:

```
> 62+5-7*9+15/3-2^2 #^means power
```

Note that commands after the symbol `#` are not executed. This can be used to include comments to lines of code. As any other calculator, R performs algebraic operations in the following order:

1. power
2. multiplication & division
3. summation and subtraction

For example, the next two commands give different results:

```
> 2+3/4^3
[1] 2.046875
> 2+(3/4)^3
[1] 2.421875
```

There are many mathematical functions that are already in R, ready to use. For example the exponen- tial function, `exp()`, the natural logarithm, `log()`, logarithm in base 2, `log2()`, the square root, `sqrt()` and trigonometric functions such as `sin()` and `cos()`.

```
> log(15)      # Logarithm of 15 using natural base
> sqrt(4)      # Square root of 4
> pi*4^2       # Area of a circle of radius 4
> a <- pi*4^2  # Create the variable "a"
> a            # Print "a"
pi*4^2
```

where `<-` represents an arrow pointing at the object receiving the value of the expression. Some people prefer te use `<-` instead of `=` (they do the same thing).

To remove all variables from R's memory, type

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

or click `clear all` in the workspace window. You can see that RStudio then empties the workspace window. If you only want to remove the variable `a`, you can type `rm(a)`.

1.3.2 Vectors and matrices

R organizes numbers in scalars (a single number 0-dimensional), vectors (a row of numbers, also called arrays 1-dimensional) and matrices (like a table 2- dimensional). The variable `a` you defined before was a scalar. To construct a vector named `x`, use the R command `c()` (it means "concatenate") as follows:

```
> x = c(4.1, -1.3, 2.5, -0.6, -21.7)
> x
[1]   4.1  -1.3   2.5  -0.6 -21.7
```

where `[1]` refers to the position of first the number in the line of the screen printout.

The following creates a new vector `z` by merging two vectors.

```
x = c(1,3,9,10)
x

## [1]  1  3  9 10

y = c(30,35,4)
y
```

```
## [1] 30 35  4

z = c(x,y)
z

## [1]  1  3  9 10 30 35  4
```

Above a new `z` is explicitly constructed by the concatenation function `c()`. R has enables us to generate commonly used sequences of numbers. For example `1:30` is the vector `c(1, 2, ..., 29, 30)`.

```
n = 5
1:n-1

## [1] 0 1 2 3 4

1:(n-1)

## [1] 1 2 3 4
```

Comparing the two sequences tells us that the priority of the operator '`:`' is higher than the '`-`' sign. In general it has higher priority than any other arithmetic operator ('`+`','`-`','`*`','`/`').

Elements in vectors can be ad- dressed by standard `[i]` indexing. For example

```
z[2]

## [1] 3

z[1:4]

## [1]  1  3  9 10

z[c(1,2,6)]

## [1]  1  3 35

z[2] = 2015
```

In the last line one of the elements is replaced with a new number. The function `seq()` is another useful tool for generating sequences. The first two arguments, if given, specify the beginning and end of the sequence.

```
seq(1,2, by=0.1)

##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0

seq(1,2, length=20)

##  [1] 1.000000 1.052632 1.105263 1.157895 1.210526 1.263158 1.315789
##  [8] 1.368421 1.421053 1.473684 1.526316 1.578947 1.631579 1.684211
## [15] 1.736842 1.789474 1.842105 1.894737 1.947368 2.000000
```

A matrix is can be created from scratch using the function `matrix()`. For example, to define the matrix

$$A = \begin{pmatrix} 2 & 4 & -1 \\ -1 & 2 & 3 \end{pmatrix}$$

one can use the function `matrix()` to a vector where the elements of A are listed column by column

```
A =  matrix(c(2,-1,4,2,-1,3), nrow=2) # create a matrix
A

##      [,1] [,2] [,3]
## [1,]    2    4   -1
## [2,]   -1    2    3
```

Matrix-operations are similar to vector operations:

```
A[1,2]   # extract the element in first 1st row 2nd column

## [1] 4

A[1, ]   # extract the first row

## [1]  2  4 -1

mean(A)  # sample average for all the elements

## [1] 1.5
```

Elements of a matrix can be addressed in the usual way: `[row,column]`. When you want to select a whole row, you leave the spot for the column number empty (the other way around for columns of course). The last line shows that many functions also work with matrices as arguments.

1.3.3 Characters

character vectors/matrices are used frequently in R, for example as plot labels. When needed they can be entered by quotes characters. For example:

```
u1 = c( "male"   "female")
u2 = c("apple",  "pear", " kiwi", "orange")
> u1
[1] "male"   "female"
> u2
[1] "apple"   "pear" " kiwi" "orange"
```

A useful function to combine numeric and character vectors is `paste()` function takes an arbitrary number of arguments and concatenates them one by one into character strings. The arguments are by default separated in the result by a single blank character, but this can be changed by the named parameter, `sep=string`, which changes it to string, possibly empty. For example

```
> labels = paste(c("X","Y"), 1:10, sep="")
> labels
[1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7" "Y8" "X9" "Y10"
```

1.4 Functions

Things are automated in so-called functions. Some functions are standard in R or in one of the packages. Later you will learn how to program your own func- tions. Important basic functions to know are:

- `max` and `min`: select the largest and smallest elements of a vector
- `range` is a function whose value is a vector of length two, namely c(`min(x)`,`max(x)`)
- `sort` returns a vector of the same size as the original vector with the elements arranged in increasing order
- `length` is the number of elements in a vector
- `sum` gives the total of the elements in vector
- `prod` gives the produc of the elements in vector
- `dim` gives the dimensions of multi-dimensional array
- Given a vector `x`, two important statistical functions are `mean(x)` which calculates the sample mean, which is the same as `sum(x)/length(x)`

The function`rnorm` is a standard R function which creates random sam- ples from a normal distribution

```
> z = rnorm(10)  # generate a vector with 10 observations from N(0,1)
> z
 [1] -0.3418484  1.3550846  0.6249231 -0.7459091  0.3754317 -0.1544511
 [7]  0.2986792  1.2771058  2.0498478  0.2596015
```

Here `rnorm` is the function and the 10 is an argument specifying how many random numbers you want, in this case 10 numbers (typing `n=10` instead of just `10` would also work). Note that entering the same commands again produces 10 new random numbers. Instead of typing the same text again, you can also press the upward arrow key "↑" to access previous commands. When you use a function to compute a mean of a vector, you will type:

```
> mean(z)         # computes the sample mean of z
> mean(z)
[1] 0.4998465
```

Within the brackets you specify the arguments. Arguments give extra information to the function. In this case, the argument says of which set of numbers (vector) the mean should computed. If you want 10 random numbers out of normal distribution $N(\mu = 10, \sigma^2 = 2^2)$ you can type

```
> x = rnorm(10, mean = 10, sd=2)
> x
 [1]  5.457800 10.408367 10.611804  6.750633  8.988588  9.057470  7.662196
 [8] 10.595350  9.955573 10.403548
```

Thus the same function (`rnorm`) may have different interfaces and that R has so called named arguments(in this case `mean` and `sd`). By the way, the spaces around the "," and "=" symbols do not matter. Comparing this example to the previous one also shows that for the function `rnorm` only the first argument (the number 10) is compulsory, and that R gives default values to the other so-called optional arguments. RStudio has a nice feature: when you type `rnorm(` in the command window and press TAB, RStudio will show the possi

1.4.1 Getting help

There is a vast amount of (free) documentation and help available. Some help is automatically installed. Typing in the console window the command

```
> help(boxplot)
```

gives help on the `rnorm` function. It gives a de scription of the function, possible arguments and the values that are used as default for optional arguments. Typing

```
> example(boxplot)
```

gives some examples of how the function can be used.

2 Scripts

You can store your commands in files called scripts. These have the extension `.R` , e.g. `foo.R`. You can open an editor window to edit these files by clicking `File` and `New` or `Open file...` You can run (send to the console window) part of the code by selecting lines and pressing CTRL+ENTER or click `Run` in the editor window. If you do not select anything, R will run the line your cursor is on. You can always run the whole script with the console command

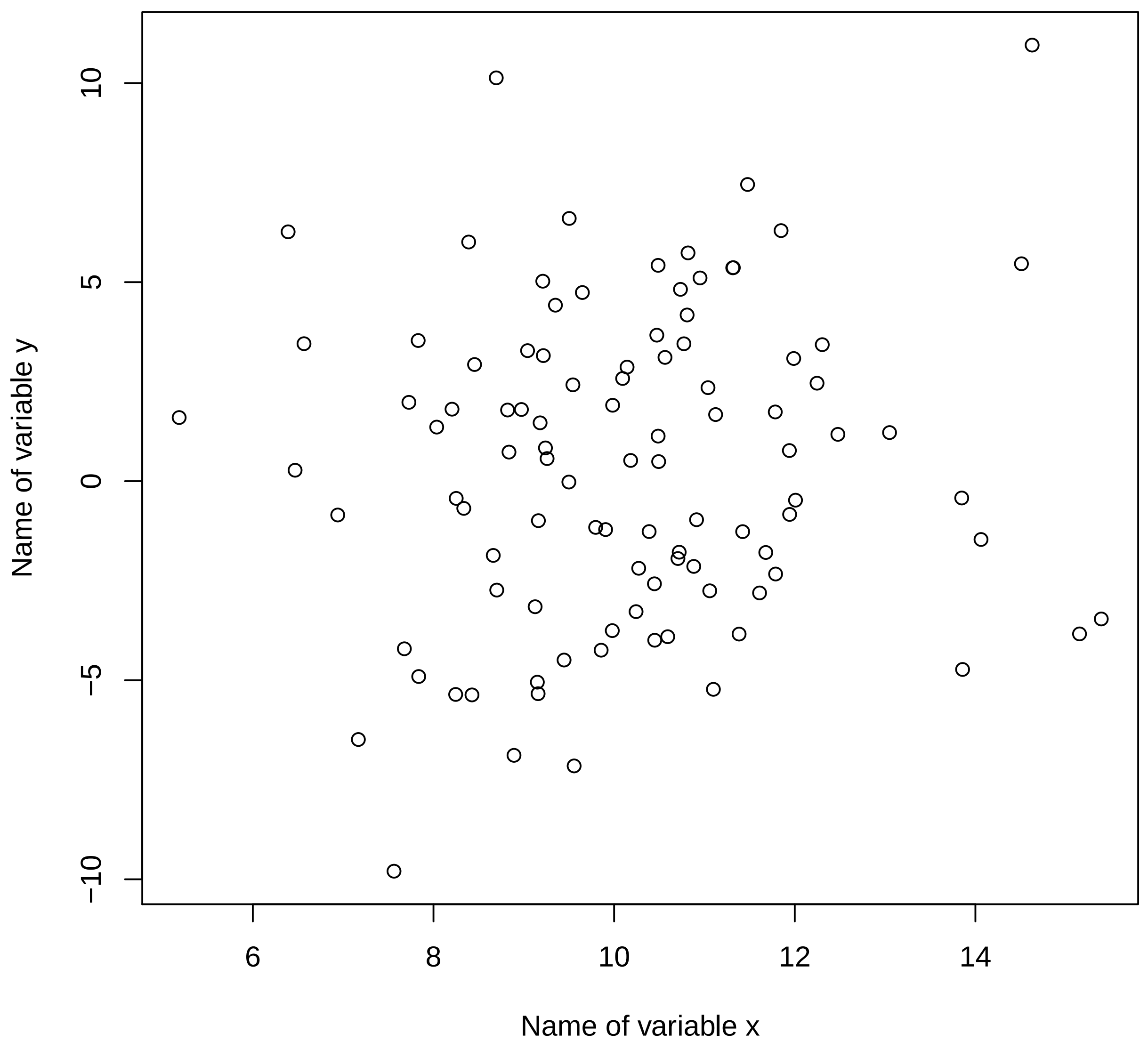
```
>source(foo.R)
```

or press CTRL+SHIFT + S.

2.1 Graphics

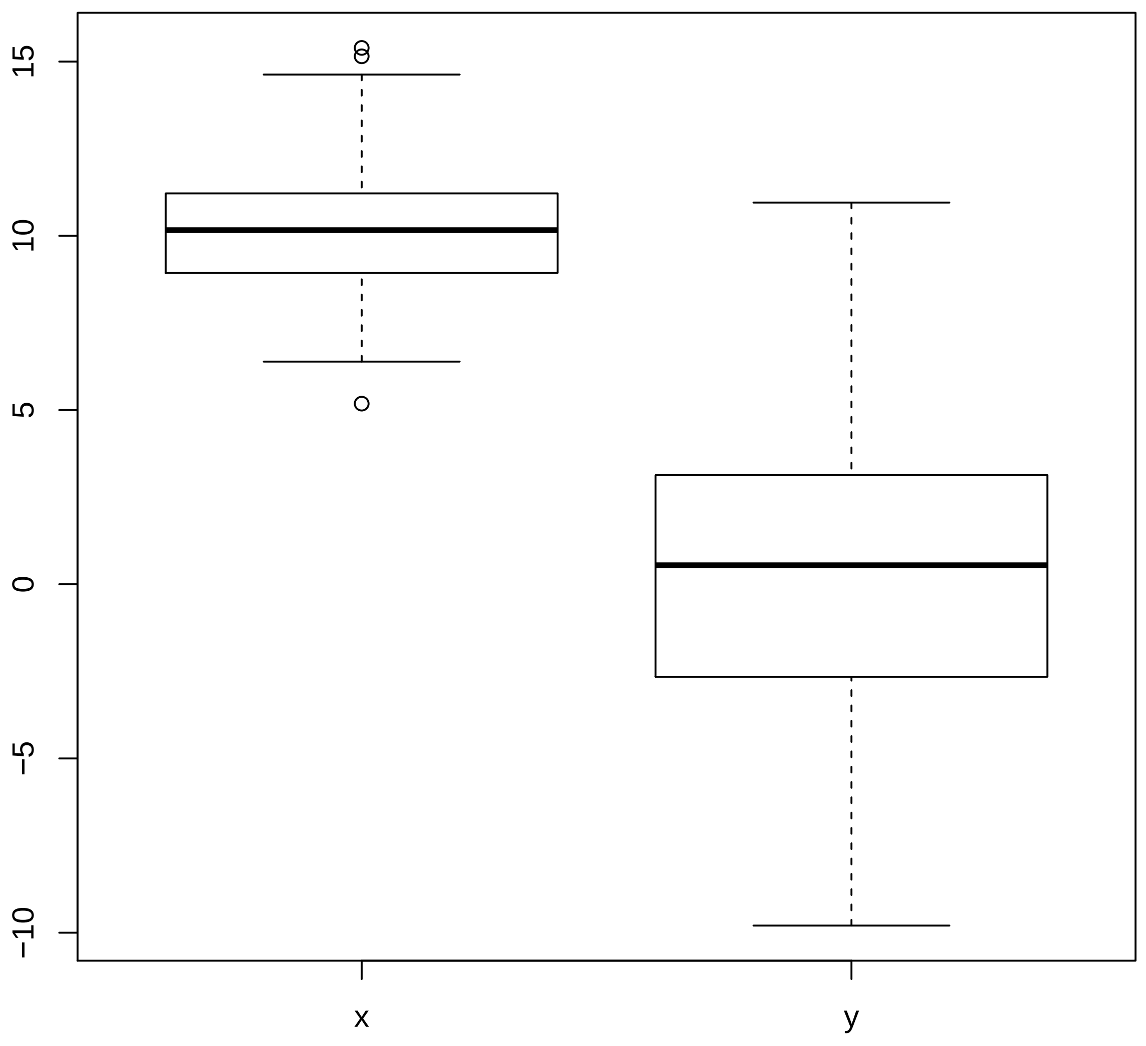
Plots are an important statistical output, so it is not surprising that R can be used to make arbitrarily beautiful graphs. The following is a very simple example:

```r
x = rnorm(100, mean = 10, sd=2)     # sample of size 100 from N(10,4)
y = rnorm(100, mean = 0, sd =4)     # sample of size 100 from N(0,16)
plot(x,y, xlab="Name of variable x", ylab="Name of variable y")
```



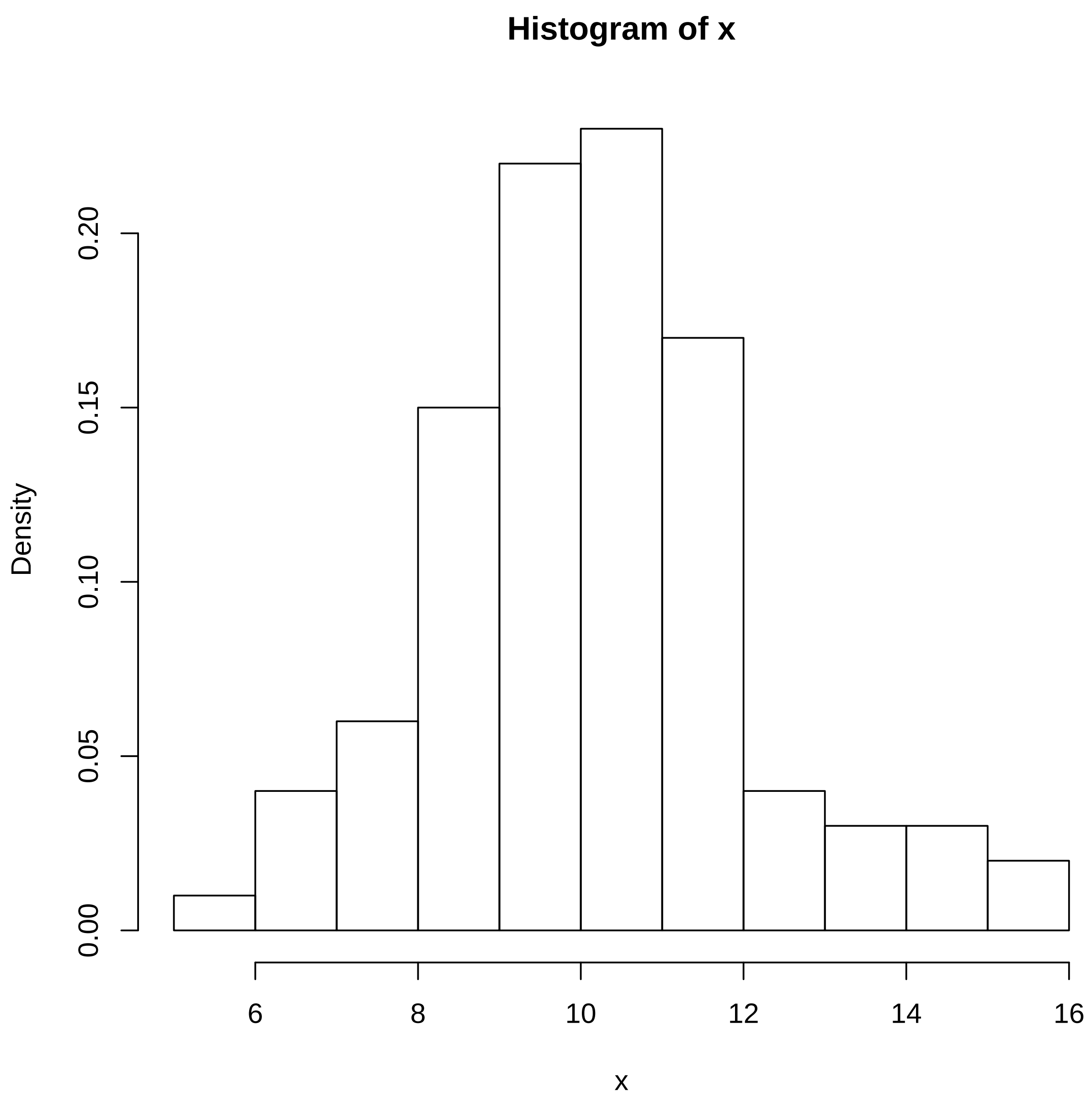
In the first two line, random numbers are assigned to the variables `x` and `y`, which become a vectors. In the second line, pairs of values are plotted in the plots window. The following creates a paired boxplot:

```
boxplot(x,y, names = c("x", "y"))  # creates paired boxplots
```



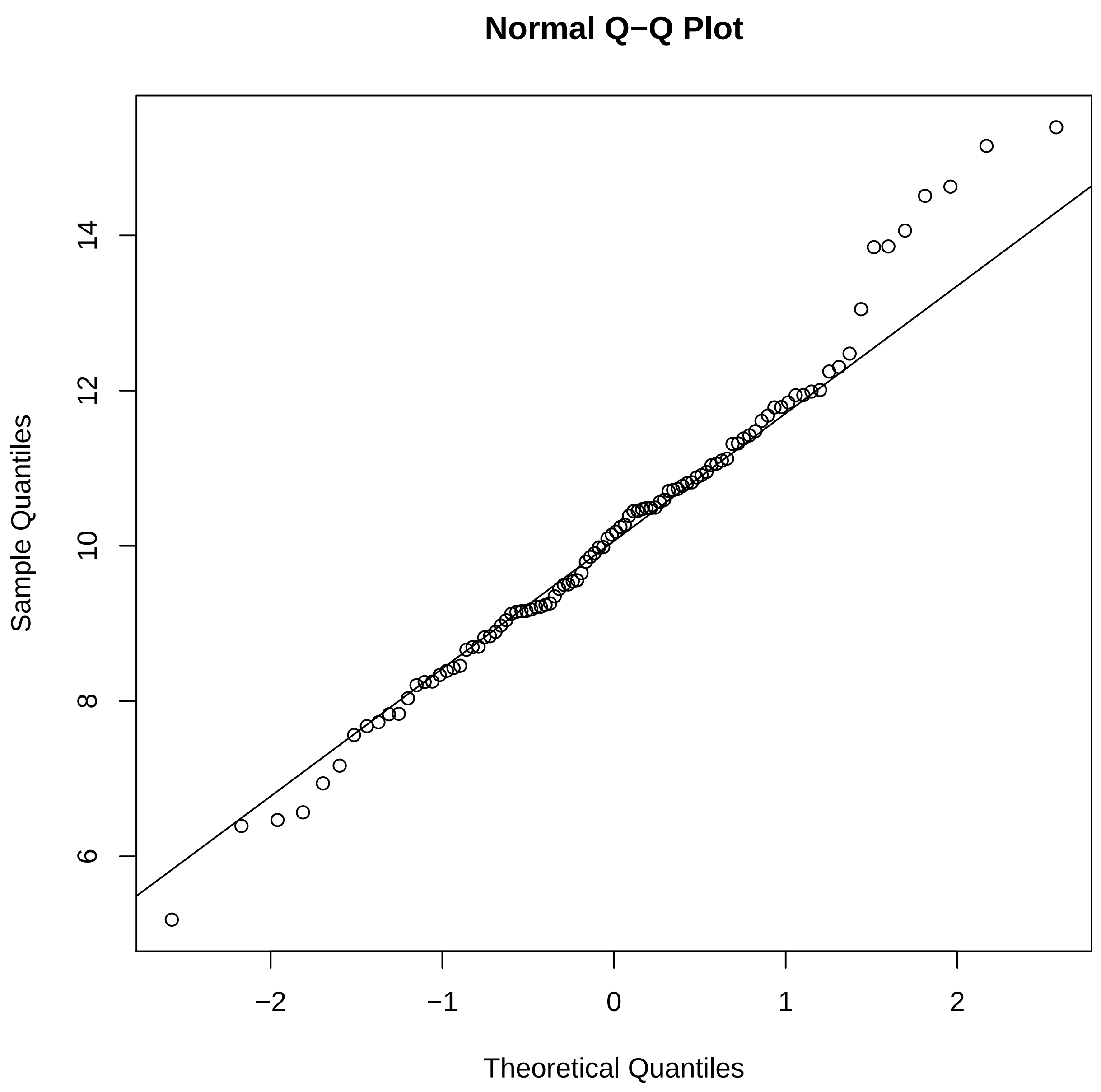
Other simple examples are the histogram and normal QQ plots, generated by the simple command

```
hist(x, freq=FALSE, nclass=10)
```



The second argument requires a density histogram (`freq=TRUE` shows relative frequencies), while the third argument specifies the number of bins.

```
qqnorm(x)
qqline(x)
```



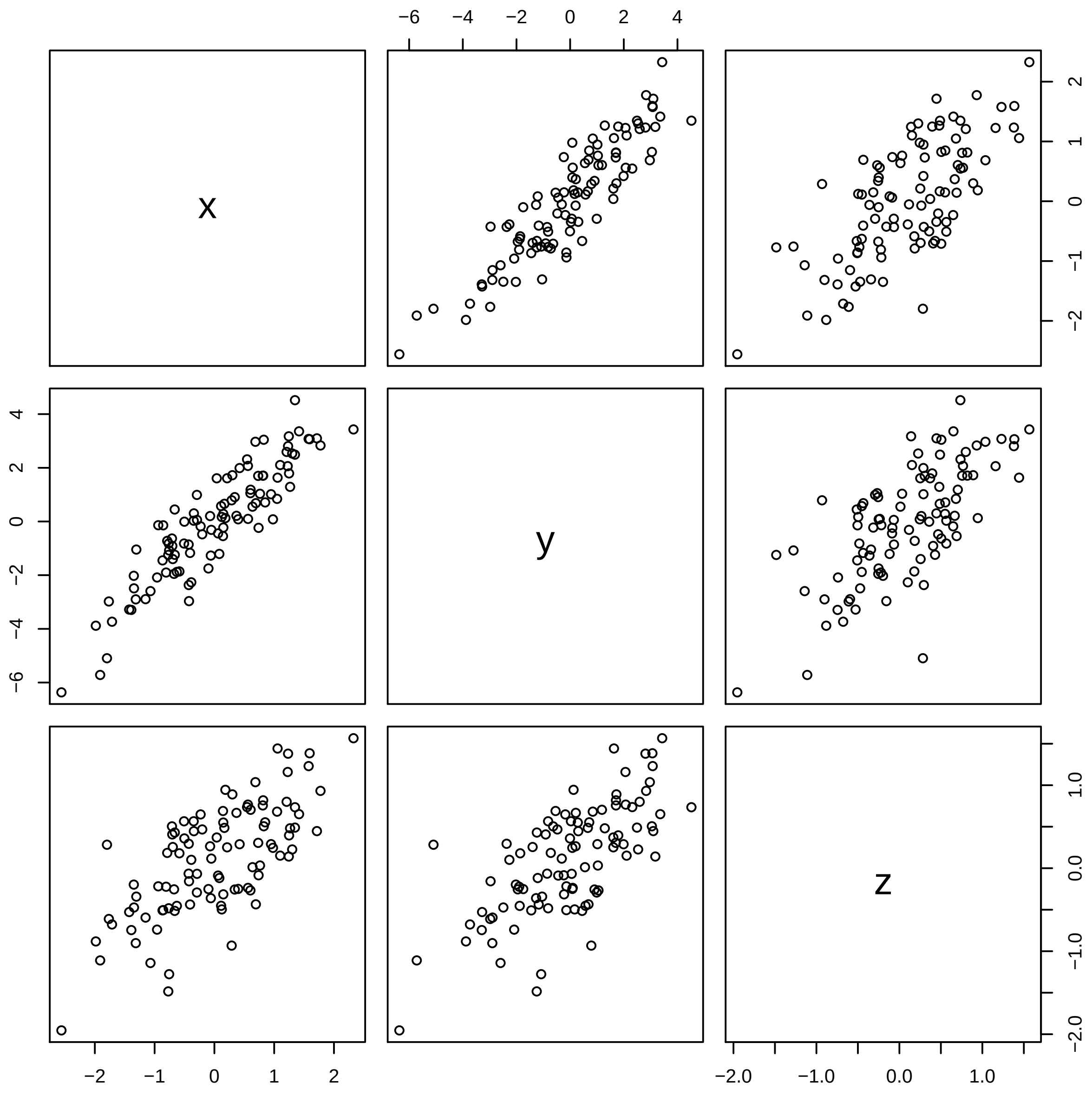
2.2 Other data structures

Datasets are often organised in data frames. A data frame is a matrix with names above the columns. This is nice, because you can call and use one of the columns without knowing in which position it is.

```
x = rnorm(100)       # generate 3 vectors of lenght 100
y = 2*x   + rnorm(100, 0, 0.8)
z = 0.5*x + rnorm(100, 0, 0.5)
t = data.frame(x,y,z) # create a dataframe
summary(t$x)          # summary statistics for x

##       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -2.558543 -0.698218  0.071943  0.009131  0.744542  2.326406

plot(t)               # scatter plot matrix
```



Another basic structure in R is a list. The main advantage of lists is that the objects forming the list do not have to be of the same length, unlike in data frames.

```
L = list(one=1, two=c(1,2), five=seq(0, 1, length=5) )
L

## $one
## [1] 1
##
## $two
## [1] 1 2
##
## $five
## [1] 0.00 0.25 0.50 0.75 1.00

L$five + 10

## [1] 10.00 10.25 10.50 10.75 11.00
```

2.3 Reading and writing data

A simple way to load a data set from text file or url in RStudio `tools>import dataset`. There are many ways to write data from within the R environment to files, and to read data from files. The following lines illustrate some basic steps using console commands:

```
# construct and store a simple data frame
t = data.frame(x=c(1,2,3), y = c(30,20,10))
t

##   x  y
## 1 1 30
## 2 2 20
## 3 3 10

write.table(t, file="mydata.txt", row.names=FALSE) # save the data frame as a file
t2 = read.table(file="mydata.txt", header=TRUE)    # load the saved data frame
t2

##   x  y
## 1 1 30
## 2 2 20
## 3 3 10
```

The argument `row.names=FALSE` prevents that row names are written to the fille. Nothing is specified about `col.names`, so the default option `col.names=TRUE` is chosen and column names are written to the file. When reading a file note that the column names are also read if `header=TRUE`. The data frame also appears in the workspace window.

R has several additional packages for reading external data format. For example, the function `read.csv()` reads .csv formatted text files. Another example is the package `foreign` which contains functions such as `read.dta(), write.dta()` and `read.spss(), write.spss()` dealing with *stata* and *spss* formats, respectively.

2.4 Missing data

When you work with real data, you will en- counter missing values. When a data point is not available, you write `NA` instead of a number.

```
x= c(rnorm(10), NA, rnorm(2))
```

Computing statistics of incomplete data sets is strictly speaking not possible. Therefore, R will say that it doesn't know what the smallest value of `x` is:

```
min(x)

## [1] NA
```

If you do not care about missing data and want to compute the statistics use the additional argument `na.rm=TRUE`

```
min(x, na.rm=TRUE)

## [1] -1.06531

mean(x, na.rm=TRUE)

## [1] -0.1837402
```

2.5 Conditional execution and loops

In R commands can be grouped together by braces signs `{ expression1; expression2; ... }` and the result of the group expression is the result of the last expression being evaluated. We can use single expressions or groups of expression to construct conditional statements with the following syntax

```
> if  (expression1)  expression2  else  expression3
```

where `expression1` is a condition resulting in a logical value true or false. If `expression1` is true, then `expression2` is evaluated. If `expression1` is false, then `expression3` is evaluated instead. Let us consider a sample of $n = 10$ observations from a normal $N(0, 1)$

```
x = rnorm(10)
```

The following code checks whether the mean is greater than the median and prints a string with the outcome.

```
if ( mean(x) > median(x) )
{
"The mean is greater than the median"
}else{
"The mean is smaller than the median"
}

## [1] "The mean is smaller than the median"
```

There is also a vectorized version of the if-else statement that uses the `ifelse()` function. In this case the form is `ifelse(condition,a,b)`.

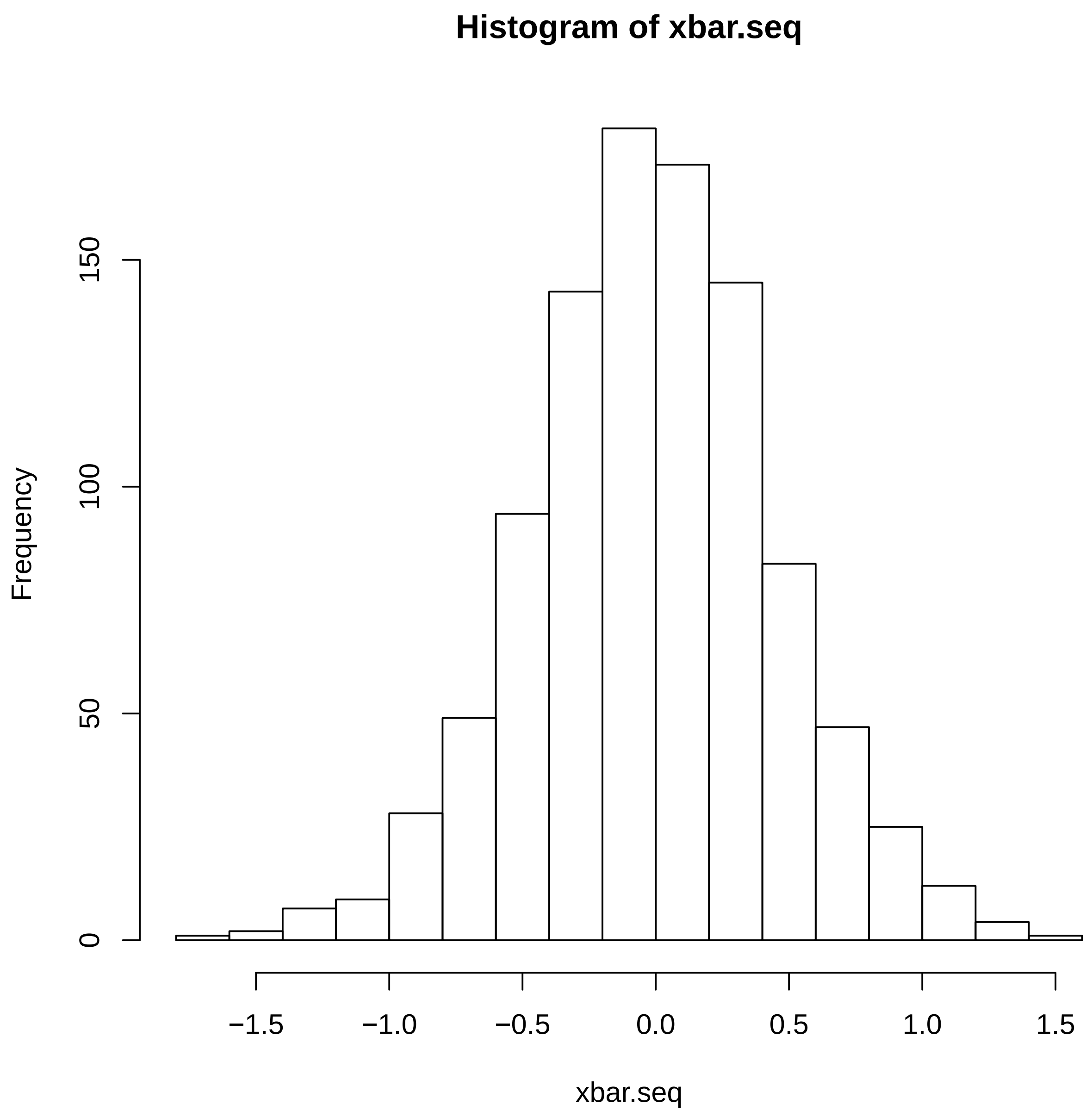
Suppose we want to carry out a task a fixed number of times, say B. To this end, it is convenient to construct for-loops. In a for-loopyou specify what has to be done and how many times. To tell how many times, you specify a so-called counter.

```
> for  (name in expression1)  expression2
```

where `name` is a loop variable and `expression1` is a vector expression (in many cases a sequence like `1:100` or so). From example, consider the drawing $B = 1000$ samples of size $n = 5$ from a $N(0, 1)$. For each sample, we compute $\hat{x}$, and store the results in a vector of length B

```
B = 1000                          # number of runs
n = 5                             # sample size
xbar.seq = 1:B                    # a vector of size to be filled with means
for (i in 1:B)
```

```
     {
     sample = rnorm(5)
     xbar.seq[i] = mean(sample)
     }
hist(xbar.seq)                      # plot the results
```



2.6 Writing your own functions

The R language allows the user to create new functions. These are true R functions that are stored in a special internal form and may be used in further expressions and so on. In the process, the language gains enormously in power, convenience and elegance, and learning to write useful functions is one of the main ways to make your use of R comfortable and productive. A function is defined by an assignment of the form

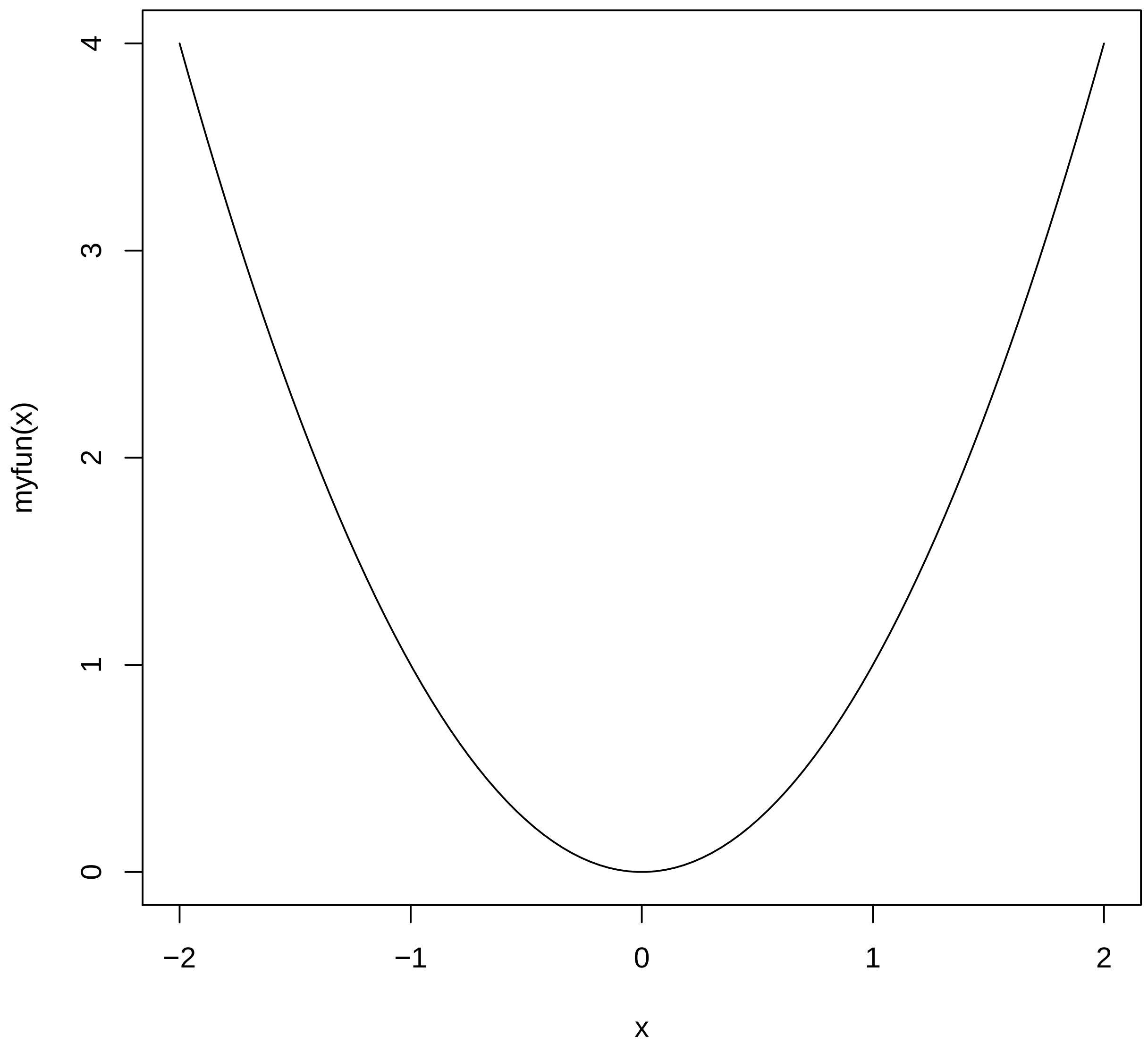
```
> namefunction <- function(argument_1, argument_2, ...) { expression }
```

For example

```
myfun  = function(x)           # Specifies function name and argument
{
y = x^2                        # Specifies what the function should do
return(y)                      # Returned value
}
myfun(1.5)                     # Computes 1.5^2

## [1] 2.25

x = seq(-2,2, length=100)      # Plots the new function
plot(x, myfun(x), type="l" )
```



Next write a function to compute the median. A simple algorithm is to sort the data and take the middle element.

```r
mymedian <- function(x){
n = length(x)
return(sort(x)[(n+1)/2])
}
```

For even sample sizes we should take the average of the two middle values

```r
mymedian <- function(x)
{
n<-length(x)
if (n %% 2==1) ## odd
sort(x)[(n+1)/2]
        else { ## even
middletwo = sort(x)[(n/2)+0:1]
return(mean(middletwo))
}
}
x = rnorm(10)
mymedian(x)

## [1] 0.1190559

median(x)

## [1] 0.1190559
```

3 Workshop

1. Box A contains one red and one black marbles. Box B contains one green and one white marbles. Consider the experiment of drawing a marble at random from Box A, transferring it to Box B and then drawing a marble from Box B at random.

   (a) Write down the associated sample space.

   (b) List the sample point(s) that comprise the event that a red ball is selected from Box B.

   (c) List the sample point(s) that comprise the event that a white ball is selected from Box B.

   (d) List the sample points that comprise the event that a green ball is not selected from Box B.

2. Let $A_n = [0, 1 + \frac{1}{n})$, $n = 1, 2, \cdots$, be a sequence of events (i.e. intervals). Find $A_1 \cap A_2$, $A_1 \cap A_2 \cap A_3$ and $A_1 \cap A_2 \cap \cdots \cap A_{100}$. Then guess the result of $\cap_{n=1}^{\infty} A_n$.

3. A six-sided die is to be rolled. However, it is known that the die is loaded so that a 1 and 6 are equally likely and are three times as likely to occur as any of the other sides. What are the probabilities for each of the six sides to be observed?

4. Each day a machine produces 100 items of certain product. Assume that 10 of these item are defective on a particular day. What is the probability that a random sample of 5 items to be selected from the outputs will contain 3 defectives?

5. If we had a choice of two airlines, we would possibly choose the airline with the better "on time performance". So consider Alaska and America West using data reported by Arnold Barnett, "How Numbers Can Trick You," in *Technology Review*, 1994.

Airline	Alaska Airlines	America West
	Relative Frequency	Relative Frequency
Destination	On Time	On Time
Los Angeles	$\frac{497}{559} = 0.889$	$\frac{694}{811} = 0.856$
Phonex	$\frac{221}{233} = 0.948$	$\frac{4840}{5255} = 0.921$
San Diego	$\frac{212}{232} = 0.914$	$\frac{383}{448} = 0.855$
San Francisco	$\frac{503}{605} = 0.831$	$\frac{320}{449} = 0.713$
Seattle	$\frac{1841}{2146} = 0.858$	$\frac{201}{262} = 0.767$
Five-city Total	$\frac{3274}{3775} = 0.867$	$\frac{6438}{7225} = 0.891$

(a) For each of the five cities listed, which airline has the better on time performance?

(b) Combining the results, which airline has the better on time performance?

(c) Interpret your results.

6. (a) Calculate $\sum_{r=0}^{n} \binom{n}{r}$ for $n = 0, 1$ and 2. Then guess the result of $\sum_{r=0}^{n} \binom{n}{r}$ for general n and prove it.

(b) Calculate $\sum_{r=0}^{n} (-1)^r \binom{n}{r}$ for $n = 1, 2$ and 3. Then guess the result of $\sum_{r=0}^{n} (-1)^r \binom{n}{r}$ for general n and prove it.