Introduction to R

Marina Evangelou and Chris Hallsworth

R is a versatile statistical programming language widely used for data analysis and manipulation. It is characterised by its wide range of functions available that can be called using:

```
functionName(argument1, argument2, ...)
```

One of the advantages of R is that it is a free software where its users can create their own packages (libraries of functions) that can be made available to the wider R community. R provides comprehensive help with all its functions and datasets. You can access this by typing a question mark `?` followed by the function's name:

```
?functionName
```

This document introduces R's basic concepts, including R objects, classes, functions and plots. A set of exercises are included that you are strongly encouraged to attempt while working on this document. The introduction also includes a number of examples for using R with statistics. Further reading and training options are provided at the end.

1 Setting up R

Before you start you need to install R on your personal machine. For doing this, visit: https://cran.r-project.org where you will find instructions on how to download and install R on Windows, MAC and Linux devices.

It is important that you also use an editor for your scripts. One option is to use the default editors that exist for the Windows and MAC versions of R or to download other editors like:

- RStudio is a more full fledged editor (http://www.rstudio.com/). This is probably the best choice if you are not very experienced with R. One useful feature is its LaTeX integration.
- You can use the general purpose editors Emacs or XEmacs together with the extension ESS (http://ess.r-project.org/). Getting to use Emacs efficiently requires a bit of effort but once you have mastered it it can be very powerful (and as it is general purpose you will be able to use it for (most) other programming languages).

2 R basics

R can be used as a calculator that can perform simple arithmetic operations. Type the following R commands into the console, and check that their outcome is what would you expect:

```
4+5
9-7
4^2
```

Or it can be used to print messages like:

```
print("Hello World")
```

3 Objects

There are two equivalent ways of defining objects in R, either using the `=` or the `<-` operators. By typing either of the two commands below you will create an object in R named x that takes the integer value 9.

```
x<-9
x=9
```

In the same way that the object `x` is defined to take the value 9 (an integer value), it can be defined to have a different class, e.g.:

- Numeric representing real values e.g. x=10.343
- Character representing string values e.g. x="Monday"
- Logical representing boolean values e.g. x=TRUE
- Complex representing complex values e.g. x=2+3i

The function `class(x)` can be used for finding the class of an object `x`. The `as.` followed by the class name functions can be used for defining the class of the object `x`, for example, `as.integer(x)`, `as.numeric(x)` and `as.character(x)`. Equivalently, you can check if an object is of a specific class by using the `is.` followed by the class name functions, for example, `is.integer(x)` and `is.character(x)`.

Type the following R commands into the console, and check that they do what you expect:

```
x=10.3
y=3
z=as.integer(11)
class(y)
class(z)

is.numeric(y)
is.numeric(x)

w1=x>y; w1
w2=y>z; w2
class(w1)
class(w2)
```

By typing the command `ls()` you can see all the objects that are defined in the current R environment (https://cran.r-project.org/doc/manuals/r-release/R-intro.html#The-R-environment). The `;` separates commands on a single line.

4 Data Structures

Vectors

The simplest data structure in R is the numeric vector. The following R command creates a numeric vector named vec that has 5 numbers {2.3, 4, 1, 3, 7}:

```
vec=c(2.3,4,1,3,7)
```

Simple operations can be done on the vectors for example to select individual elements or subsets of the vector using square brackets, as follows:

```
vec[1]
vec[2:3]
```

Or you can omits elements of the vector:

```
vec[-1]
vec[-c(3,4)]
```

It is worth noting that indexing in R starts from one instead of zero as with some other programming langugages. Simple calculations can be done on a vector. The commands below compute the length of the vector and the average value of the vector and of subsets of it.

```
length(vec) # The length function computes the length of the vector
```

```
## [1] 5
```

```
length(c(vec,vec))
```

```
## [1] 10
```

```
mean(vec) # The  mean function computes the average value of the vector
```

```
## [1] 3.46
```

```
mean(vec[-c(1:2)])
```

```
## [1] 3.666667
```

Anything listed after the '#' symbol is regarded as a comment in R and it is not computed. Other types of objects in R include: matrices, factors, lists and data frames.

Exercises:

Create two vectors x and y in R that take the values: x=(5, 4, 2, 1, 10) and y=(2.3, 8.9, 9, 5, 4). Using R:

1. Compute the maximum and median values of each vector (*Hint:* Check the functions ?max and ?median).
2. Compute the output of x+y?
3. Find the entries of x that are greater than 3 (*Hint:* Check the function ?which).
4. Compute the sum of all entries of x that are greater than 3.

Factors

Factors in R are objects that are used to define a selection of categories that can be both ordered and unordered.

For example, suppose that we are recording pets in three categories: "dogs","cats" and "other"; and a factor vector, *pet*, has been observed with the following entries:

```
pet=as.factor(c("dogs","dogs","other","cats","cats","dogs"))
pet
```

```
## [1] dogs  dogs  other cats  cats  dogs
## Levels: cats dogs other
```

```
levels(pet)
```

```
## [1] "cats"  "dogs"  "other"
```

```
pet[1]
```

```
## [1] dogs
## Levels: cats dogs other
```

```
table(pet) #The table function produces a frequency table of the observations
```

```
## pet
##  cats  dogs other
##     2     3     1
```

Factors can be modified to include additional levels. For example, we can have an additional level "goldfish". Before adding any new observations we need to add the "goldfish" level to the levels of the vector pet:

```
levels(pet)=c(levels(pet),"goldfish")
```

Exercise:

When running the following R code a warning message is produced. What is the error message? How can you correct the code so that the warning message is not produced?

bedroom=as.factor(c("bed","wardrobe","desk","bed","bed","desk","wardrobe"))
bedroom[c(3,4)]="chair"

Matrices

Matrices are a two-dimensional representation of data elements in R. The columns and the rows of a matrix can represent different observations, for example:

```
A=matrix(1:9, nrow=3, ncol=3)
```

Similarly to vectors, we can do basic arithmetic calculations on a matrix, including matrix multiplication and subsetting of elements. Type the following R commands into the console, and check their outcome:

```
B=matrix(1:9,nrow=3,ncol=3)

##Arithmetic calculations of a matrix
A+B
A%*%B

##Print columns and rows of a matrix
A[,3]
B[2,]
B[1,3]
diag(A) #The diagonal of matrix A is printed
```

Vectors and matrices can be combined to form new matrices using the cbind and rbind commands that column and row bind the objects together, respectively.

Type the following R commands into the console, and check their outcome:

```
x=c(1:3)
y=c(4:6)
z=cbind(x,y); is.matrix(z); dim(z) #Column bind
z=rbind(x,y); is.matrix(z); dim(z);#Row bind
```

Exercises:

Add a fourth column to the matrix A created above with entries: {"M","F","M"}.

1. What is the outcome?
2. What is the class of each column of the updated A matrix?
3. What is the class of matrix A?

Data Frames

A data frame in R is a list of vectors of equal length. Each vector can have different types of data stored in it. We can index data frames like a matrix or like a list. Type the following R commands on your console, and check that they do what you expect:

```
x=c(4,3,2,1)
y=c(20,10,20,10)
df=as.data.frame(cbind(x,y))
df

df[1,1]
df[,1]

class(df); dim(df)
colnames(df)
rownames(df)
is.numeric(df[,1]); class(df[,1])
is.numeric(df[,2]);
```

As the column names of the data frame `df` are: `x` and `y` we can use the `$` operator to subset entries of the data frame. For example:

```
df$x
df$y[1:2]
```

R has several built-in data frames, for example:

```
mtcars
```

(see ?mtcars for more infomation)

Exercises:

Let's explore the mtcars data frame in R.

1. What are the dimensions of the mtcars data frame?
2. Create a new data frame with the entries of cyl column greater than 6
3. Construct a frequency table for the carb variable
4. Create a frequency table of the cyl and gear variables together

Lists

A list can contain a combination of anything you want, including nested lists:

```
x=list(1,list("hat","coat"),c(3,4,5),list(q="?",answer=24))
x
```

List indexing can be used to extract a sub-list

```
x[2]
x[2:4]
x[[2]]
```

Exercises:

1. What does the '[[]]' (double square brackets) allow you to access that the '[]' (square brackets) do not in the above example?
2. What are the outputs of the commands: 'x[2][[1]]' and 'x[2][1]'?

5 Functions

R has a number of built-in functions for performing different types of calculations or other operations. Functions in R can be called using `functionName(argument_1, argument_2, ...)`. So far we have discussed the use of the `class`, `mean`, `which`, `cbind`,`table` functions amongst others.

As discussed R provides comprehensive help with all built-in functions that you can access by typing a question mark followed by the function's name, for example, `?functionName`. This will bring up a help window where information about the syntax of the function, available options for the input arguments and examples applying the function are provided. It is recommended that when working with unfamiliar functions you check their help pages for further information and the best places to use them.

R offers the flexibility of creating your own functions. New functions in R can be defined by:

$$\text{functionName} = \text{function(argument1, argument2, ...)} \ \{ \ \text{expression} \ \}$$

For example we can write our own function for computing the average value of a numeric vector:

```
new_average=function(V){
  new_ave=sum(V)/length(V)
  return(new_ave)
    }
```

We can check if our new function has the same output as the mean function in R:

```
x=c(4,3,2,1); new_average(V=x); mean(x)
```

```
## [1] 2.5
```

```
## [1] 2.5
```

```
new_average(3:20); mean(3:20)
```

```
## [1] 11.5
```

```
## [1] 11.5
```

Multiple arguments can be passed in functions. Some of which you might want to prespecify, in the example below y is prespecified to take the value 1. We can call the function with or without the y statement and we will not have an error:

```
new_add=function(x,y=1){
  x+y
}
new_add(x=5)
```

```
## [1] 6
```

```
new_add(x=5,y=1)
```

```
## [1] 6
```

```
new_add(x=c(2,3),y=2)
```

```
## [1] 4 5
```

If the y value was not prespecified in the definition of the function, the first command: `new_add(x=5)` would have led to an error.

You should note a few things about creating your own functions:

1. Any new variables created within a function are local and are not available within the global R environment. For example, in the new_average function the object new_ave is only available within the function and not globally. You can check this by running `ls()` on your console.
2. You can allow for one function to pass on arguments to another function by adding the `...` argument (https://cran.r-project.org/doc/manuals/r-release/R-intro.html#The-three-dots-argument). The `...` argument is a useful argument when you do not want to prespecify any arguments that are used by other function within your creted function. For example check the R output of the following R commands:

```
col_plot=function(x,y,...){
  print(x+y)
  plot(x,y,...)
}

col_plot(1:5,1:5,col="red")
col_plot(1:5,1:5,cex=2)
```

Exercise:

What does the following R function achieve?

```
sumpos=function(x){
wh.pos=which(x>5)
sum(x[wh.pos])
}
```

6 Loops

If you would like to perform the same set of actions repeatedly it is a good idea to use loops for this. R has a number of loop constructions available: `for`, `while` and `repeat`. Depending on the nature of the problem that you are working a different construction will be more suitable. The examples below illustrate how to code the three loops in R:

```
# for loop
for(i in 1:4){
  print(i)
}
```

```
# while loop
i=1
while(i<4){
  i=i+1
  print(i)
}
```

```
# repeat loop
repeat{
  print(i)
  i=i+1
  if(i>4){
    break
  }
}
```

The `if` command in the `repeat` loop checks if a statement is TRUE or FALSE. In the case that it is TRUE then the expression within the brackets is evaluated. The `break` command terminates the `repeat` loop.

Further information about conditional execution (`if`, `if else`, `else`) statements and loop constructions can be found at: https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Conditional-execution

Exercise:

Construct a for-loop in R for computing the average value of each column of the mtcars data frame.

Other R built-in functions for evaluating expressions on vectors, matrices, data frames and lists include the sapply, apply and lapply functions.

The apply function can be used to compute a specific function on either the columns or the rows of a matrix. The outcome of the function is a vector of values. For example:

```
A=matrix(1:9, nrow=3, ncol=3)
apply(A,2,mean) #The average value of each column of matrix A is returned
```

```
## [1] 2 5 8
```

```
# Instead of a pre-defined function, we can use our own functions.
# Below the average value of each row of matrix A is returned computed our own function
# that has as an input statement vector i
apply(A,1,function(i){return(sum(i)/length(i))})
```

```
## [1] 4 5 6
```

Check `?sapply` and `?lapply` for further information on how to use them to repetitively perform the actions of a function on the elements of a vector.

Exercise:

Write an R code for computing the median value of each row of the mtcars data frame using the sapply function.

7 Plotting

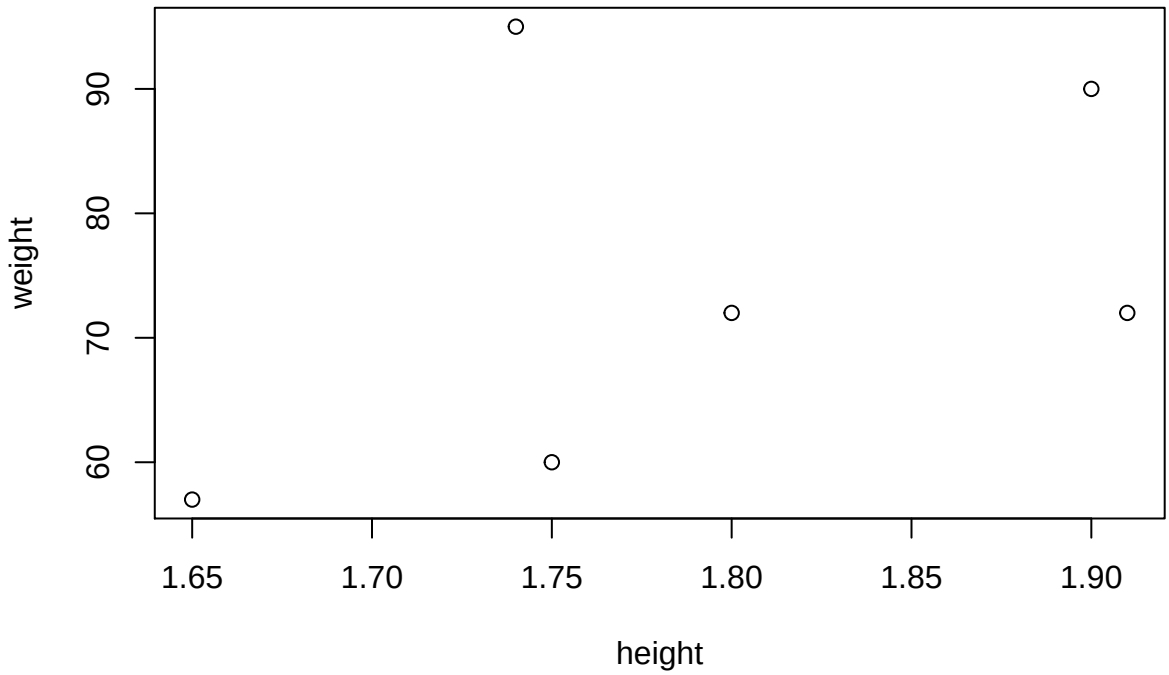
R can produce a wide variety of plots and graphics. This section presents some common types of plots that arise when working with data, and how to tweak the output so that it looks presentable.

Input a small height and weight dataset by pasting the two lines below.

```
weight=c(60, 72, 57, 90, 95, 72)
height=c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

To get a simple scatter plot illustrating these variables, use the plot command:

```
plot(height, weight)
```



R plots can be customised extensively. Try modifying the arguments of the plot function (e.g.`cex`, `col`) to check their effect on the plot:

```
plot(height, weight,pch=2,col="red",cex=2, main="scatter plot")
```

Add the `xlab` and `ylab` arguments to the command above to change the labels on the two axes.

The `hist` built-in function can be used to produce a histogram of the height data.

```
hist(height)
```

A nicer-looking histogram can be obtained by specifying the number of bins manually. Check the argument `breaks=k` in the `hist` function.

For looking at experimental data, the *box and whisker plot*, often just called a box plot, is a useful tool. It shows the first and third quantiles of a numerical dataset as the bottom and top of a box, and a line inside the box represents the median of the data. Whiskers at the top and bottom of the box show a multiple of the interquartile range - this is where "most" of the data should be. Any data points outside the whisker region are displayed as individual points. These individually displayed points may be outliers (depending on the situation).

A box plot of the height data can be produced using the `boxplot` function:

```
boxplot(height)
```

Exercises:

1. Produce a histogram of the mpg variable in the mtcars data frame.
2. Produce a box plot of the disp variable in the mtcars data frame.
3. Write an R code for producing histograms for all columns of the mtcars data frame (Hint: use one of the loop constructions). To create all the plots in a single page you can use the command 'par(mfrow=c(3,4))' before the first plot.

You can get a quick glimpse of R's plotting capabilities by calling the graphics demo command

```
demo(graphics)
```

For more advanced visualisation techniques in R you can explore the ggplot2 package (https://cran.r-project.org/web/packages/ggplot2/index.html). Further guidance on how to use the ggplot2 can be found at: https://r4ds.had.co.nz/data-visualisation.html.

8 Importing/ Exporting Data

R can handle data in a wide variety of formats. For the smallest datasets, you can enter the variables directly as vectors, as we have done above. Most commonly, you will have data in a file that needs to be read in to R.

To read data in a text file you can use the `read.table` command, and to save data created in R you can use the `write.table` command. For example:

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
write.table(mtcars[,1:5],file="my_mtcars.txt",row.names=FALSE,col.names=TRUE,quote=FALSE)

new.mtcars=read.table("my_mtcars.txt",header=TRUE)
head(new.mtcars)
```

```
##    mpg cyl disp  hp drat
## 1 21.0   6  160 110 3.90
## 2 21.0   6  160 110 3.90
## 3 22.8   4  108  93 3.85
## 4 21.4   6  258 110 3.08
## 5 18.7   8  360 175 3.15
## 6 18.1   6  225 105 2.76
```

If the data files that you need to read are comma-separated files you can use the `read.csv` command to read the file or you can amend the `read.table` to include the argument `sep=","`. In addition, if there are character values in the file that you are importing in R you can choose whether the character entries will be read as factors

using the argument `stringsAsFactors=TRUE`. For further information about the functions check `?read.table` and `?read.csv`.

Also you can save your current R environment by exiting R with `q("yes")` or you can save a subset of the defined objects of the environment using:

```
save(x,y, file = "filename.RData")
```

and you can retrieve previously saved data using:

```
load("FILEDIRECTORY/filename.RData")
```

You need to be careful when importing/ exporting data as you need to properly specify the directory where the files (data) are located.

9 R Packages

R functions are stored in R packages. The standard R functions, like `print`, `plot`, etc are included in the base R packages. For using functions and/or datasets that are not stored in the standard R packages you will need to install the relevant R package.

```
install.packages("packageName")
```

and for using the relevant package after installation you will need to run:

```
library("packageName")
```

10 Additional Exercises

1.

Type the following R command on your console:

```
z=seq(10,20,2)
```

1. What is the class of z?
2. What is the length of z?
3. What are the entries of z?
4. How will z change if the following command was typed instead: 'z=as.character(seq(10,20,2))'

If you are not familiar with the seq function check it using ?seq.

2.

Missing values in R are coded as NA. Create the following vector in R:

```
c(4,2,NA,10,20,NA,NA,3)
```

Write the relevant R commands for:

1. Finding the entries of the vector that are non-missing.
2. Computing the sum of the vector excluding its missing values.

3.

Type the following R commands on your console:

```
x=sample(c(1:20),9,rep=FALSE)
A=matrix(x,nrow=3,ncol=3)
y=sample(c(1,20),9,rep=TRUE)
B=matrix(y,nrow=3,ncol=3)
```

1. What does the 'rep' argument do in the sample function?
2. Write the relevant R commands for:
   (a) Adding the two matrices A and B
   (b) Adding the diagonals of matrices A and B
   (c) Creating a new matrix C that has 4 columns: the first 3 of which are the columns of A and the fourth column contains the last 3 entries of vector y
   (d) Creating a new matrix D that has 4 rows that combines matrix B and the first three entries of vector x
   (e) Computing the inverse of matrix A
3. What happens if you try to add matrices A and D?

4.

Type the following R commands on your console to create a data frame with the following elements:

```
animal=c("cat","dog")
weight=c(10,12)
df=data.frame(animal,weight)
df; class(df)
```

Write the R commands for performing the following:

1. Replace the weight of dog with the number 15 instead of 12
2. Adding a third column with the height of the two animals 72 and 90 cm for cat and dog respectively
3. Add a third animal to the data frame: frog with weight=1kg and height=10cm
4. Assign your own names to the rows of the data frame df

5.

One of the available datasets in the `MASS` library (one of the R built-in libraries) is the Animals dataset. Load the data frame Animals on your R environment:

```
library(MASS)
data(Animals)
Animals[1:5,]
?Animals #For further information about the dataset
```

1. How many animals are included in the dataset? Are there any duplicates?
2. Compute descriptive statistics for body and brain weight.
3. Create box-plots of body and brain weights.
4. Create histograms of body and brain weights.
5. Prepare a plot of body weight against brain weight. The x-axis and y-axis labels of the graph should be named: "Animals Body Weight kg" and "Animals Brain Weight g". The title of the plot should be:"Animals".
6. Run parts 1-5 to the subset of the dataset that includes only the animals with body weight less than 20000 kg and brain weight less or equal to 1000 g.

6.

Write your own R functions:

1. For computing the variance of a vector. You can use the formula of the standard deviation:

$$\frac{\sum_i (x_i - \bar{x})^2}{n-1}$$

   Be careful to deal with missing values in your function.
2. For producing a plot of two vectors y against x where the data points are illustrated by red triangles in the graph.
3. For computing frequency tables for the columns of a data frame that are characters. Test your code on the 'sleep' dataset.

4. For computing the factorial of an integer. Try doing this both with recursion and without.

7.

The following R code when ran produces a number of errors. Correct the errors and run the code:

```
rm(list=ls())0 #This command ensures that the R environment is cleaned.

my_fun=function(x,y=5){
  return(x+y)
)

j=1;
while(j<=4)
  x=my_fun(i,k)
  if(j==2){print("Hello World")k"}
  print x
}
```

1. What is the objective of the R code?
2. Re-write the code by replacing the while loop with a for loop.

11 Statistics and R

R as a statistical programming language has a number of built-in functions for conducting statistical tests and computations.

Such functions include evaluating the cumulative distribution function ($P(X \leq x)$; CDF), density and quantile functions of a number of distributions (e.g. Normal, Beta, Binomial, Poisson) and for simulating data from these distributions. R functions for computing the probability density of a distribution begin with a `d`, for the CDF with a `p`, for the quantile function with `q` and for simulating data with `r`. For further information and for the list of available distributions see https://cran.r-project.org/doc/manuals/r-release/R-intro.html#Probability-distributions.

Let's look at the examples below:

Suppose that $X \sim Poisson(4)$. We can compute $P(X \leq 3)$ using:

```
ppois(3,lambda=5,lower.tail=TRUE)
```

```
## [1] 0.2650259
```

Or you can draw 5 values from a Uniform(0,1) distribution using:

```
runif(5,min=0,max=1)
```

```
## [1] 0.051457463 0.179099351 0.008170631 0.967664950 0.362049386
```

Binomial Distribution

This section looks at the binomial distribution and binomial test in R.

Suppose that a coin is thrown 20 times and the number of heads is recorded. The number of heads obtained is considered to follow a Binomial distribution with $n = 20$ trials and with probability of success $p = 0.5$. Using R we can compute the P(Number of heads obtained ≤ 15):

```
  pbinom(15,20,0.5)
```

```
## [1] 0.994091
```

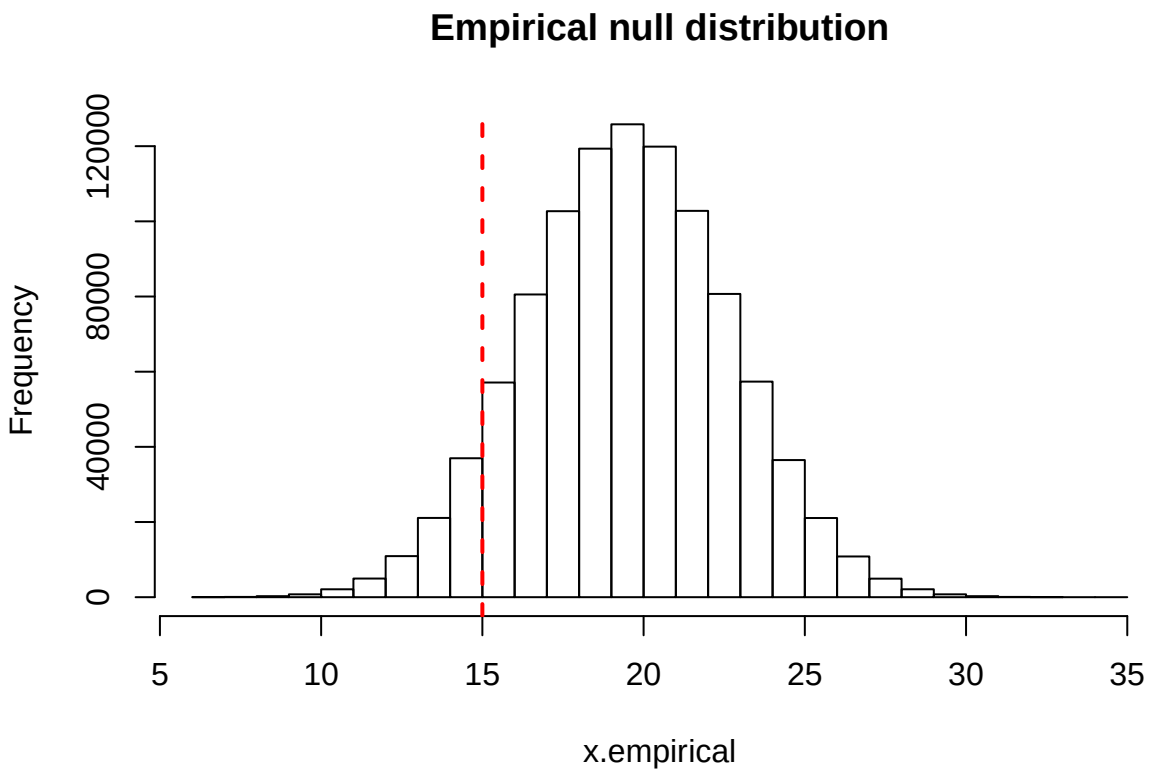
Let's look at a different experiment. Suppose we have observed 15 heads (successes) out of 40 flips of a coin (trials). We want to know whether we have evidence that the true probability p of success for each trial is smaller than 0.5. We might also say that we want to know whether the difference between the observed probability and 0.5 is *statistically significant*. The significance level (often called the p value) is a measure of how likely our observed result would be, if the *null hypothesis* of $p = 0.5$ were true. Specifically, the significance level is the probability of obtaining as extreme a result as we did, if 0.5 is the true success probability. The following exercises attempt to make this notoriously tricky concept a bit clearer.

First, simulate a large number of draws from the null distribution, i.e. many sets of 40 trials, where each trial is a success with probability 0.5. Each dataset is then just a number between 0 and 40 - the number of successes.

```
        n=1e+6
        x.empirical=rbinom(size=40,n=n,p=0.5)
```

We can review the null distribution using a histogram. Also we can add a vertical line that presents the value observed in the real data 'x.test'.

```
x.test=15
hist(x.empirical,main="Empirical null distribution")
abline(v=x.test,col="red",lty=2,lwd=2)
```



Does the observed value look plausible if the null distribution is indeed the true distribution? We can evaluate how plausible it is by computing:

```
sum(x.empirical<=x.test)/n
```

```
## [1] 0.077166
```

Exercises:

1. What is your conclusion? Would you reject the null hypothesis?
2. How does the result above compare to instead conducting a Binomial test for examining whether the probability of success is less than 0.5? R can do this for us using 'binom.test(x.test,n=40,alternative="less")'.
3. How would the results of the two tests would be differnt if 'x.test' was equal to 5?

Normal Distribution

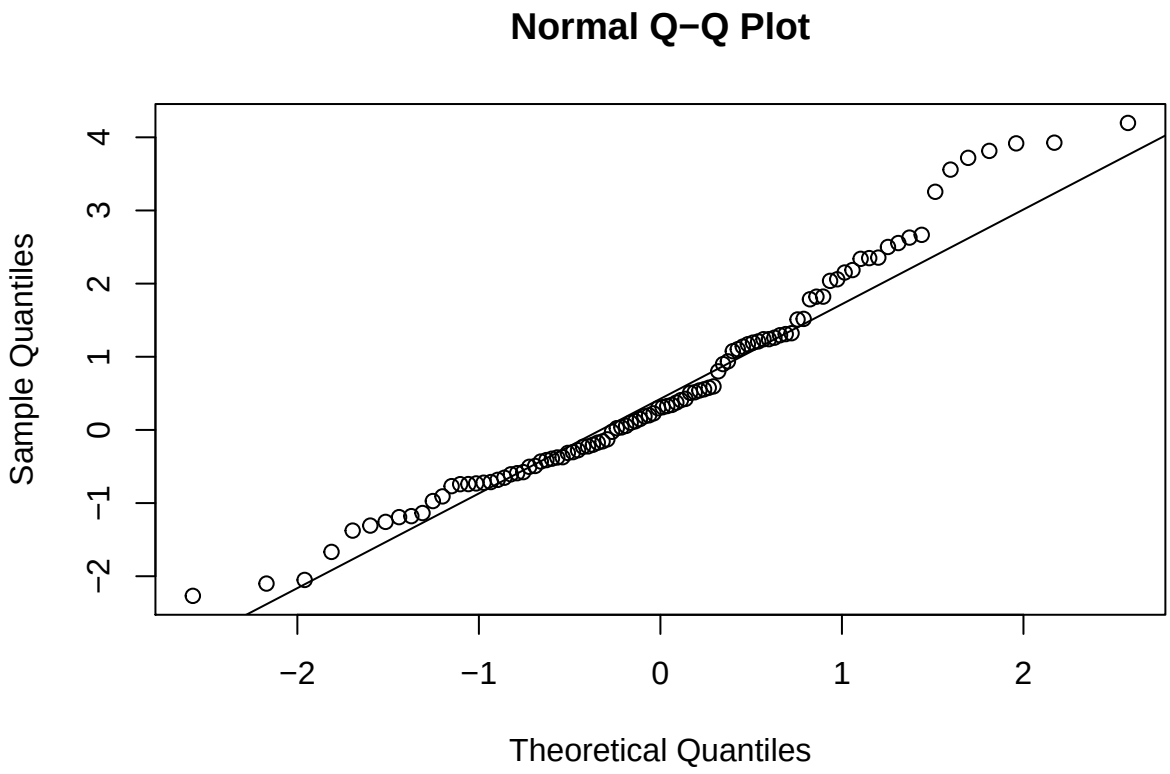
This section is devoted on Normally distributed data and statistical analysis for testing the distribution of such data.

Suppose we have two populations one drawn from a N(1,1.6) distribution and the second one from a N(0.5,1.5) distribution. Each population has 100 observations. We are interested in testing whether the two populations follow the same distribution.

```r
x=rnorm(100,mean=0.5,sd=1.5)
y=rnorm(100,mean=1,sd=1.6)
```

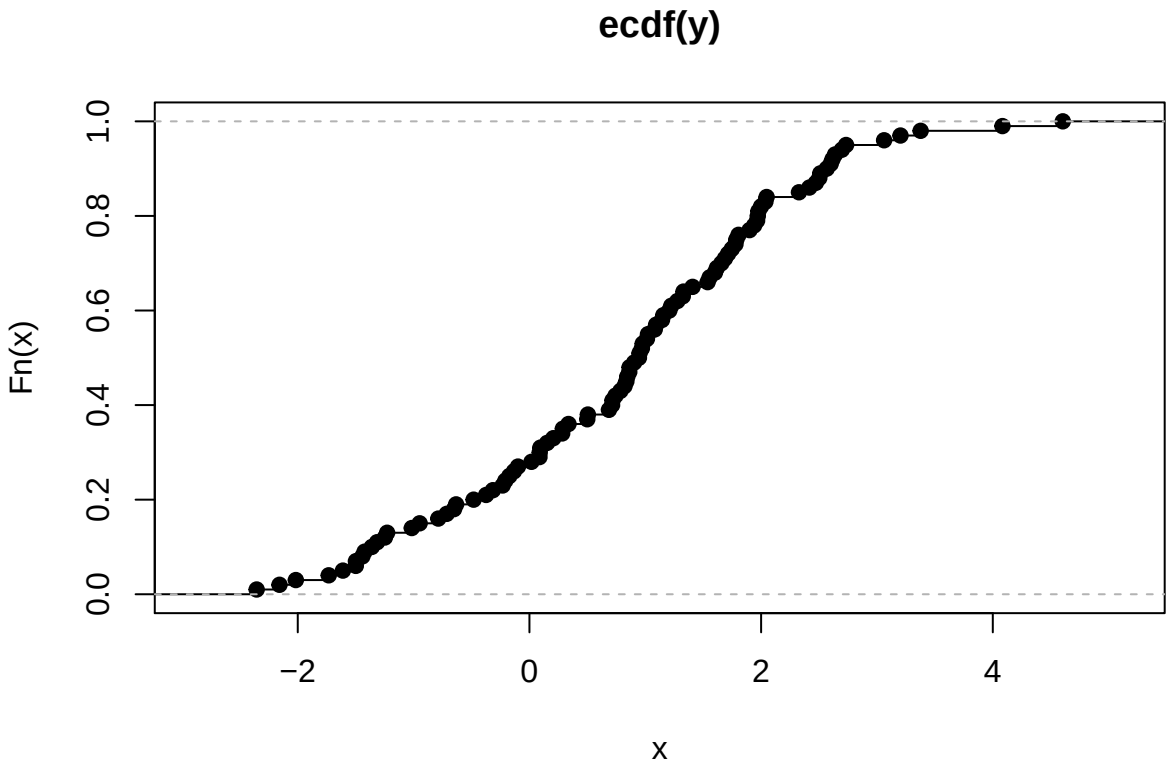
Before starting any statistical analysis we should visualise the data. Similarly to before we can plot the samples using a histogram and/or we can use the Quantile-Quantile plot to examine the distribution of the samples:

```r
qqnorm(x)
qqline(x) #Check ?qqline
```



In addition, we can plot the empirical cumulative density function (ECDF) of the data:

```r
plot(ecdf(y))
```



R provides a comprehensive list of statistical tests in the `stats` package (see Section 9 for further information). The `stats` package is normally loaded within R (https://stat.ethz.ch/R-manual/R-devel/library/stats/html/stats-package.html).

There are a number of possible ways of testing if two populations follow the same distribution or if they have similar characteristics. Below we illustrate how to apple some of them in R.

Each one of the possible tests makes different assumptions about the two populations. One approach would be to use the unpaired t-test that assumes that the two populations follow a Normal distribution for testing if the means of the two populations are equal. A second approach would be to use the non-parametric Wilxocon (Mann-Whitney) test that tests if the two populations follow the same distribution by assuming that there is a common underlying distribution amongst the two datasets. A third possible way is to compare the ECDFs of the two populations through a Kolmogorov-Smirnov test. The examples below illustrate the application of the different tests:

If both `x` and `y` follow a Normal distribution, we can test if they have the same mean using an unpaired t-test:

```
t.test(x,y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -1.3711, df = 198, p-value = 0.1719
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.6775466  0.1217904
## sample estimates:
## mean of x mean of y
## 0.5476518 0.8255299
```

As the t-test does not test for equality of variances we can use the F-test to test for equality in the variances (Check '?var.test'). If the two variances are found to be the same (i.e. do not reject the null of equal variances) we can amend the t.test above to assume for equality of variances:

```
##
##  Two Sample t-test
##
## data:  x and y
## t = -1.3711, df = 198, p-value = 0.1719
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.6775466  0.1217904
## sample estimates:
## mean of x mean of y
## 0.5476518 0.8255299
```

We can test if x is Normally distributed using the Kolmogorov-Smirnov test:

```
ks.test(x,"pnorm",mean=mean(x),sd=sd(x))
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  x
## D = 0.10688, p-value = 0.2034
## alternative hypothesis: two-sided
```

Exercises:

1. Test the null hypothesis: $H_0 : x \sim N(0, 1)$.
2. Test if 'x' and 'y' follow the same distribution using the non-parametric test: wilcox.test(x,y).
3. Test if 'x' and 'y' follow the same distribution using the Kolmogorov-Smirnov test.