Basic Concept of Statistics

Paolo Girardi and Livio Finos

8/10/2020

Contents

Basic use of R

Working-space and helps

R is an integrated software environment for data manipulation, computation and graphic representation. To start a session, it is necessary to perform a double mouse click on the icon of R or (RStudio). This will open the command window e the command prompt will be proposed:
>

The entities that R creates during a work session are called objects. These latter can be numbers, strings, vectors, matrices, functions, or more general structures. Such items are saved by name and stored in a dedicated area called workspace. At any time, it is possible check the objects available in the workspace using the command *ls()*

```
ls()
```

```
## character(0)
```

```
# empty workspace
```

I can remove an objects with the command *rm()*

```
rm(thing)
```

```
## Warning in rm(thing): oggetto "thing" non trovato
```

```
# attention... no thing in the working space
```

The working space can be saved and restore with the commands *save.image()* and *load()*

```
save.image("my_working_space.Rdata")
load("my_working_space.Rdata")
```

Files can be loaded and saved in a specific working directory in a local folder. We can use the functions *setwd()* and *getwd()* to set or to retrieve the folder location.

```
setwd("/Users/Paolo/My_Website/content/it/courses/PhD_neuroscience/")
getwd()
```

```
## [1] "/Users/Paolo/My_Website/content/it/courses/PhD_neuroscience"
```

For any request of help about R functions, a series of help function can be used

```
help(setwd)
?setwd
# and if I don't remember the function name help.search() or apropos()
apropos("setw")
```

```
## [1] "setwd"
```

Basic operation

R can be employed as a simple scientific calculator

```
1+1
```

```
## [1] 2
```

```
3/2
```

```
## [1] 1.5
```

```
1>2
```

```
## [1] FALSE
```

using a several number of local functions. Each function can be applied by means of round brackets with an argument inside

```
#squared root
sqrt(2)
```

```
## [1] 1.414214
```

```
#log - natural basis
log(10)
```

```
## [1] 2.302585
```

```
#exponential
exp(4)
```

```
## [1] 54.59815
```

```
#sin function
sin(pi)
```

```
## [1] 1.224647e-16
```

```
# the result is 0... pi is the greek pi constant
pi
```

```
## [1] 3.141593
```

```
#I can combine more functions
log(sqrt(2))*exp(4)
```

```
## [1] 18.92228
```

I can assign to an object values or results of operations as follows

```
x<-1
x
```

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

```
y<-3/2
y
```

```
## [1] 1.5
```

```
z<-1>2
z
```

```
## [1] FALSE
```

Vectors and Matrix

To create a vector, a basic function is *c()*

```
x<-c(1,2,3,9,12)
x
```

```
## [1]  1  2  3  9 12
```

or a sequence can be created in these two ways

```
x1<-1:20
x1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
x2<-seq(from=1,to=20,by=1)
x2
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
# the result is the same
```

Other useful functions are

```
#replicate
x<-rep(2,5)
x
```

```
## [1] 2 2 2 2 2
```

```
#multiplicate for scalar *
x<-1:5
x<-x*3
# x has been overwritten... pay attention!
x
```

```
## [1]  3  6  9 12 15
```

```
# other functions
sum(x)
```

```
## [1] 45
```

```
prod(x)
```

```
## [1] 29160
```

```
min(x)
```

```
## [1] 3
```

```
max(x)
```

```
## [1] 15
```

```
length(x)
```

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

A matrix can be define with command *matrix()*

```
mat<-matrix(data=1:9,nrow=3,ncol=3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
# by default elements are placed by col
```

and square brackets are used to select elements in a vector or matrix as follows

```
# in a vector
x[3]
```

```
## [1] 9
```

```
x[1:2]
```

```
## [1] 3 6
```

```
# in a matrix
mat[1,2]
```

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

```
mat[1:2,3]
```

```
## [1] 7 8
```

```
# creating subselection
x[-1] #dropping the first element
```

```
## [1]  6  9 12 15
```

```
mat[-1,] #for the first row
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    3    6    9
```

Type of ojects in R

In R we can define a many type of data. R can be automatically define an object on the basis of the object characteristics.
A numeric vector

```
x<-1:3
is(x)
```

```
## [1] "integer"             "double"              "numeric"
## [4] "vector"              "data.frameRowLabels"
```

```
is.numeric(x)
```

```
## [1] TRUE
```

A matrix

```
mat<-matrix(data=1:9,nrow=3,ncol=3)
is(mat)
```

```
## [1] "matrix"    "array"     "structure" "vector"
```

A char vector (a vector of letters or even not numbers)

```
label<-c("white","red","black")
is(label)
```

```
## [1] "character"           "vector"              "data.frameRowLabels"
## [4] "SuperClassMethod"
```

I can combine numbers and characters in a list

```
list<-list(x,label)
list
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "white" "red"   "black"
```

```
list[[1]] # first element of a list with double square brackets
```

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

and rename each single element

```
names(list)<-c("numbers","colours")
```

We can combine number and characters in a data.frame

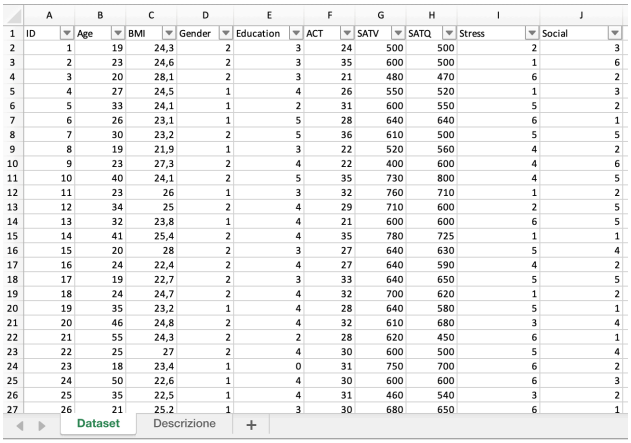
```
data<-data.frame(numbers=x,colours=label)
data # the result is a typical dataset format
```

```
##   numbers colours
## 1       1   white
## 2       2     red
## 3       3   black
```

Import a dataset

R permits to import data in several format and from other statistical softwares (STATA, SPSS, SAS, EXCEL, etc..). When R import a file it creates a data.frame object. For each format there are specific functions. We are going to explore the most used functions.
However, a beginner user can follow a guided importation process from
File > Import Dataset > and then to select the importing format.

A classical format for dataset is the text (extension csv, txt, dat).
Text can be imported in R with the function *read.csv()* or *read.table()*. This dataset called "test" collected the results on proficieny test (SAT and ACT) in a sample of 150 students. Students are by row, while characteristics by column. In Excel we have this output:



	A	B	C	D	E	F	G	H	I	J
1	ID	Age	BMI	Gender	Education	ACT	SATV	SATQ	Stress	Social
2	1	19	24,3	2	3	24	500	500	2	3
3	2	23	24,6	2	3	35	600	500	1	6
4	3	20	28,1	2	3	21	480	470	6	2
5	4	27	24,5	1	4	26	550	520	1	3
6	5	33	24,1	1	2	31	600	550	5	2
7	6	26	23,1	1	5	28	640	640	6	1
8	7	30	23,2	2	5	36	610	500	5	5
9	8	19	21,9	1	3	22	520	560	4	2
10	9	23	27,3	2	4	22	400	600	4	6
11	10	40	24,1	2	5	35	730	800	4	5
12	11	23	26	1	3	32	760	710	1	2
13	12	34	25	2	4	29	710	600	2	5
14	13	32	23,8	1	4	21	600	600	6	5
15	14	41	25,4	2	4	35	780	725	1	1
16	15	20	28	2	3	27	640	630	5	4
17	16	24	22,4	2	4	27	640	590	4	2
18	17	19	22,7	2	3	33	640	650	5	5
19	18	24	24,7	2	4	32	700	620	1	2
20	19	35	23,2	1	4	28	640	580	5	1
21	20	46	24,8	2	4	32	610	680	3	4
22	21	55	24,3	2	2	28	620	450	6	1
23	22	25	27	2	4	30	600	500	5	4
24	23	18	23,4	1	0	31	750	700	6	2
25	24	50	22,6	1	4	30	600	600	6	3
26	25	35	22,5	1	4	31	460	540	3	2
27	26	21	25,2	1	3	30	680	650	6	1

Dataset | Descrizione | +

Figure 1: The dataset in Excel

We save them in a CSV format and import with the function *read.csv()* in R.

```
test<-read.csv("test.csv",sep=";",header=T,dec=",")
head(test) #the first 6 rows
```

```
##   ID Age  BMI Gender     Education ACT SATV SATQ Stress Social
## 1  1  19 24.3      F     secondary  24  500  500      2      3
## 2  2  23 24.6      F     secondary  35  600  500      1      6
## 3  3  20 28.1      F     secondary  21  480  470      6      2
## 4  4  27 24.5      M        degree  26  550  520      1      3
## 5  5  33 24.1      M upper primary  31  600  550      5      2
## 6  6  26 23.1      M   post-degree  28  640  640      6      1
```

```
names(test)
```

```
##  [1] "ID"        "Age"       "BMI"       "Gender"    "Education" "ACT"
##  [7] "SATV"      "SATQ"      "Stress"    "Social"
```

Reproducible Statistical Analysis with R-Markdown

Why use R-Markdown

The use of RStudio with R-Markdown provides the basis to edit text and executable R code is the same text file.
R-Markdown permits to
- create HTML, PDF, or MS Word output;
- use beamer, ioslides, and slidy presentations; - manage tables, figures and bibliographies; - create a customizable environment.

There are a lot of sites. More about R Markdown:

- rmarkdown.rstudio.com
- online reference guide

Create an RStudio project

Create an Rmd report

From RStudio, create a new R Markdown file

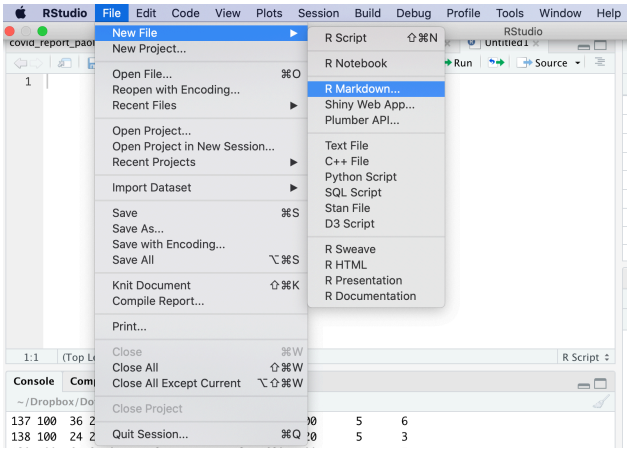


Figure 2: rmarkdown.rstudio.com

Select HTML output (for now). We can change it later.

An untitled R Markdown file is created with some default text and R code.

File -> Save As to the project directory with an Rmd suffix, for example, `test-report.Rmd`.

Click `Knit HTML` to render the document in HTML.

The report appears in your RStudio viewer (or can be opened in other HTML viewer).

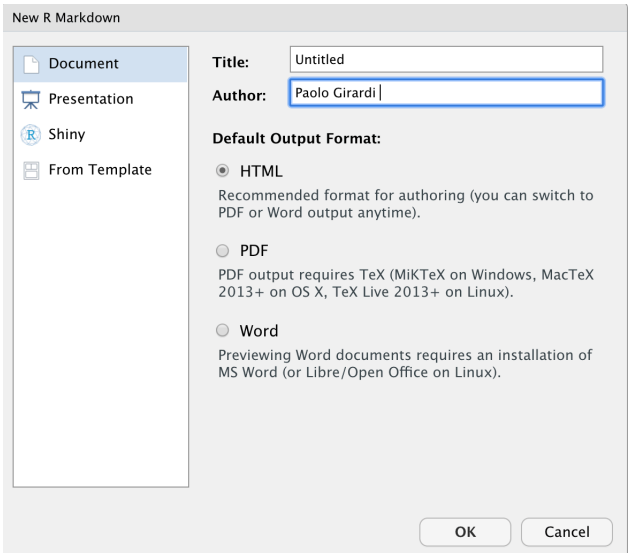


Figure 3: rmarkdown.rstudio.com

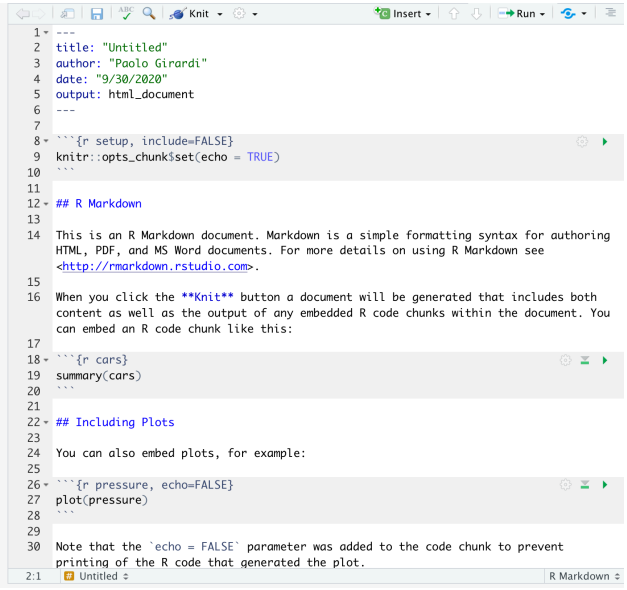


Figure 4: rmarkdown.rstudio.com

Compare the markup to the output

Compare the Rmd markup to the HTML output. For example,

- markup `<http://rmarkdown.rstudio.com>` creates a link, http://rmarkdown.rstudio.com
- markup `**Knit**` produces a bold typeface, **Knit**
- single backtick markup ‘ produces highlighted inline code `Knit`.
- markup `*Knit*` produces an italic typeface, *Knit*

The code-chunk markup

echoes the R code in the HTML document, executes the *summary()* function, and writes the result to the output.

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

The next code chunk includes an `echo=FALSE` argument that prevents printing the R code chunk to the output.

However, the code is executed and the graph is printed to the output document.

What the software is doing

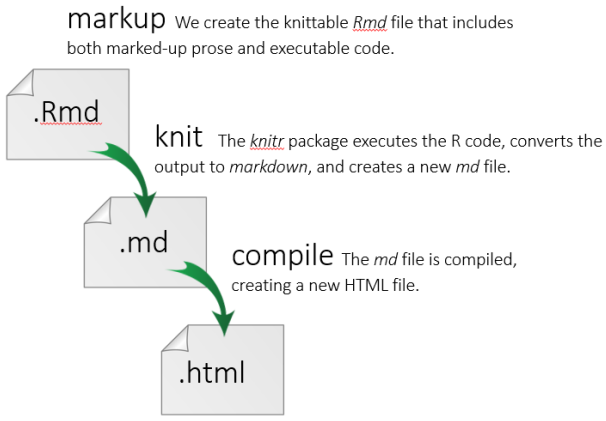


Figure 5: rmarkdown.rstudio.com

The resulting output file is placed in the same directory as your Rmd file.

Changing the output format

The YAML header or front-matter in the Rmd file controls how the file is rendered. (YAML: YAML Ain’t Markup Language)

Let’s change the title to *Test Report*.

The `output:` option recognizes three document types:

- html_document
- pdf_document
- word_document

You can type these directly in the Rmd YAML header or you can use the RStudio `Knit` pulldown menu

---

Formatting the output

Articles on the RStudio website for formatting output.

- Formatting an HTML document
- Formatting a PDF document
- Formatting a Word document

Markdown basics

Section headings

Figure 6: rmarkdown.rstudio.com

Emphasis

Figure 7: rmarkdown.rstudio.com

Itemize

Sub-items begin with 4 spaces.
Every line ends with two spaces.

Figure 8: rmarkdown.rstudio.com

Enumerate

Sub-items begin with 4 spaces.
Every line ends with two spaces.

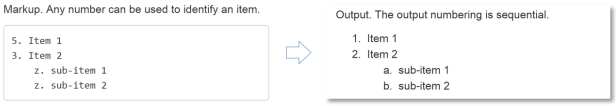
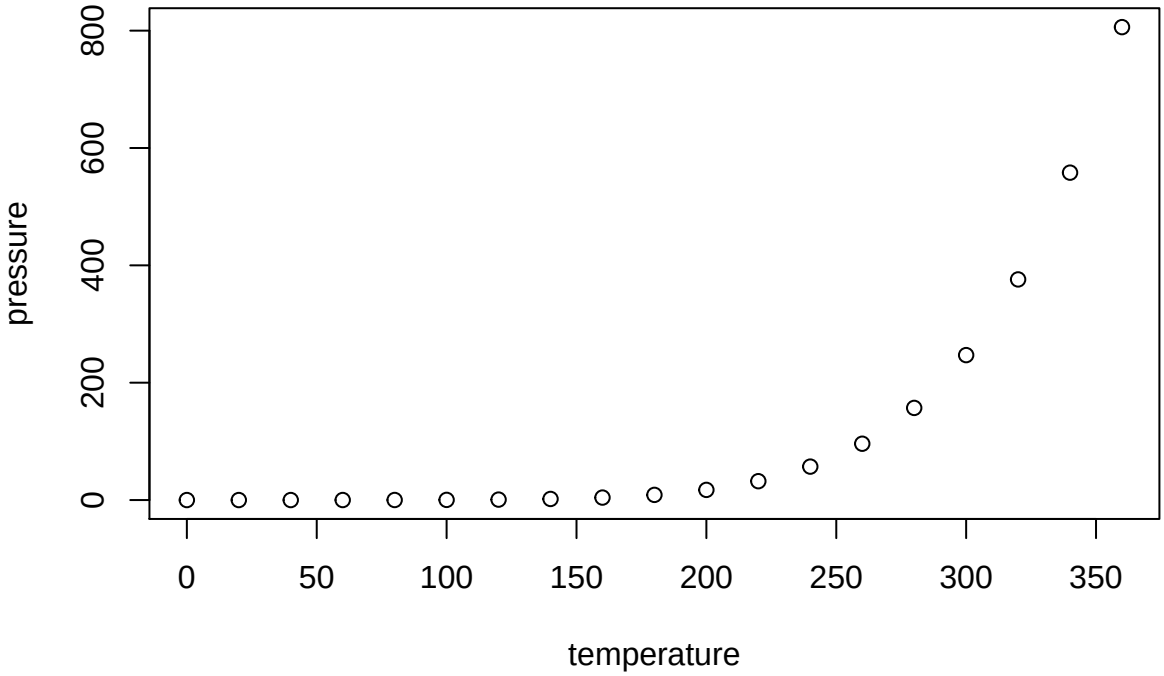


Figure 9: rmarkdown.rstudio.com

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Data visualization and base statistics with R

The normal distribution

R has some basic functions for calculating density, cumulative distribution function and quantiles for many distributions of interest. It is also possible to generate achievements' pseudo-random from the distribution. For example, considering the distribution normal standard, there are 4 main functions:

- dnorm (x) calculates the density value in x;
- pnorm (x) calculates the value cumulative distribution function into x;
- qnorm (p) computes the quantile of level p;
- rnorm (n) generates a sample from a normal standard of size n (N(0,1)).

The prefix (d, p, q and r) descriminates the type of function associated to the random variable. R contains some functions related to several random variables by default. In particular

```
## Warning in rbind(c("norm", "normal", "mean, sd", "0, 1"), c("lnorm", "log-
## normal", : number of columns of result is not a multiple of vector length (arg
## 4)
```

norm	normal	mean, sd	0, 1
lnorm	log-normal	meanlog, sdlog	0, 1
t	t di Student	df	-
chisq	chi-quadrato df	-	chisq
f	F	df1, df2	-, -
unif	uniform	min, max	0, 1
exp	exponential	rate	1
gamma	gamma	shape, scale	-, 1
binom	binomial	size, prob	-,-
pois	Poisson	lambda	-

Other random variables can be added with "external R-Packages" or built by yourself. Some example of the functions related to the normal

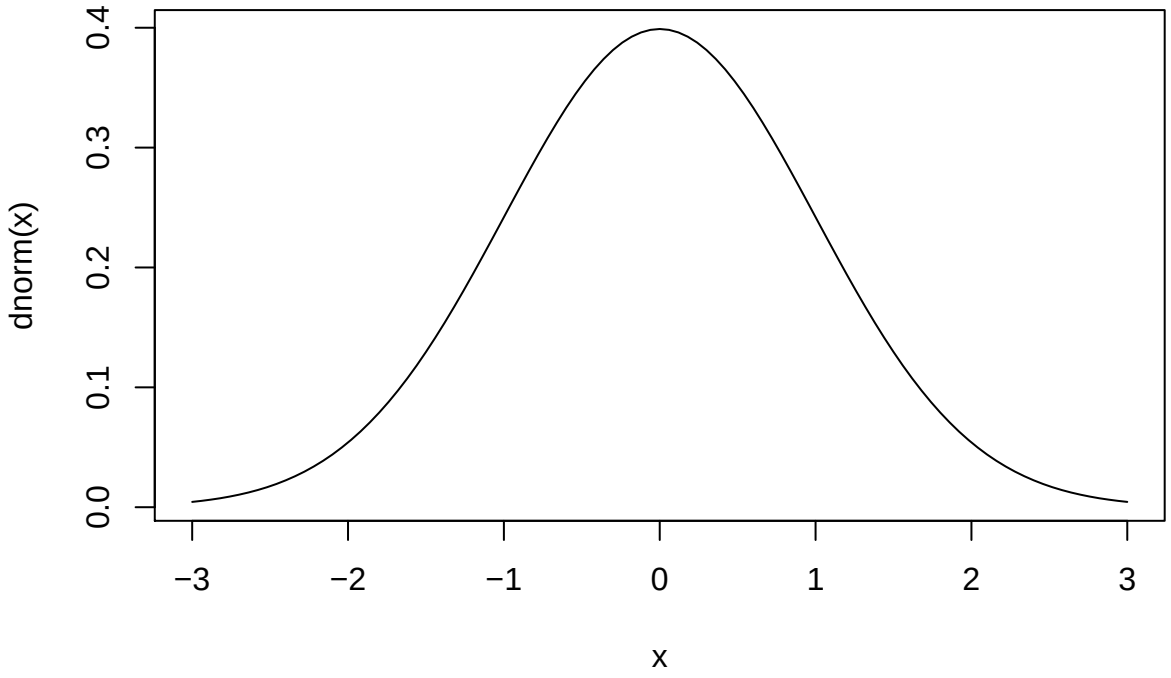
```
## [1] 0.3989423
```

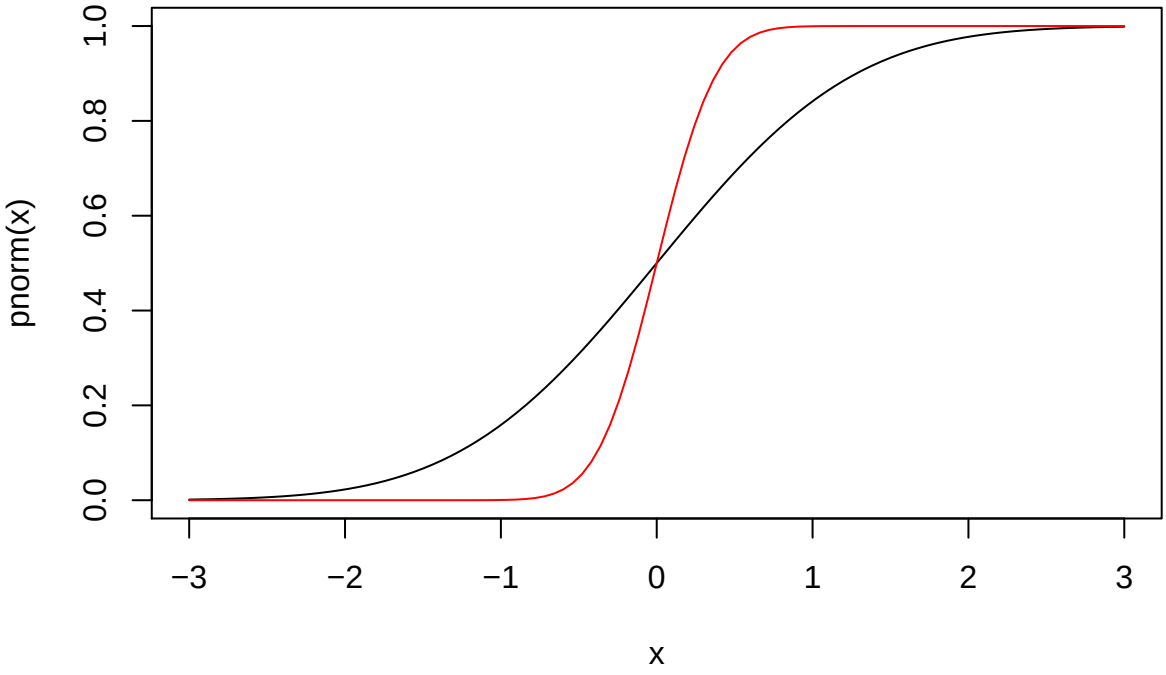
```
## [1] 0.5
```

```
## [1] 0
```

```
## [1] -0.9726606 -1.1934622 -1.1272200
```

```
## [1] -0.6264538  0.1836433 -0.8356286
```





To build a function in R

R permits to build personal functions.
The structure is similar to other programming codes. The function *function()* permits to define a new function. Here an example that returns the area

```
## [1] 40
```

We can expand this function calculating the perimeter and the area returning a list

```
## $area
## [1] 40
##
## $perimeter
## [1] 28
```

Basic statistics function with R

From the last imported dataset test

```
test<-read.csv("test.csv",sep=";",header=T,dec=",")
head(test) #the first 6 rows
```

```
##   ID Age  BMI Gender     Education ACT SATV SATQ Stress Social
## 1  1  19 24.3      F     secondary  24  500  500      2      3
## 2  2  23 24.6      F     secondary  35  600  500      1      6
## 3  3  20 28.1      F     secondary  21  480  470      6      2
## 4  4  27 24.5      M        degree  26  550  520      1      3
## 5  5  33 24.1      M upper primary  31  600  550      5      2
## 6  6  26 23.1      M   post-degree  28  640  640      6      1
```

Useful functions to visualize a dataset are:

- *View()*: to visualize a dataset like in rows and columns

- *str()*: to analyse the structure of a dataset
- *names()*: to obtain the name of each variabile in a vector

```
#View(test)
str(test)
```

```
## 'data.frame':    150 obs. of  10 variables:
##  $ ID       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age      : int  19 23 20 27 33 26 30 19 23 40 ...
##  $ BMI      : num  24.3 24.6 28.1 24.5 24.1 23.1 23.2 21.9 27.3 24.1 ...
##  $ Gender   : Factor w/ 2 levels "F","M": 1 1 1 2 2 2 1 2 1 1 ...
##  $ Education: Factor w/ 6 levels "degree","lower primary",..: 5 5 5 1 6 3 3 5 1 3 ...
##  $ ACT      : int  24 35 21 26 31 28 36 22 22 35 ...
##  $ SATV     : int  500 600 480 550 600 640 610 520 400 730 ...
##  $ SATQ     : int  500 500 470 520 550 640 500 560 600 800 ...
##  $ Stress   : int  2 1 6 1 5 6 5 4 4 4 ...
##  $ Social   : int  3 6 2 3 2 1 5 2 6 5 ...
```

```
names(test)
```

```
##  [1] "ID"        "Age"       "BMI"       "Gender"    "Education" "ACT"
##  [7] "SATV"      "SATQ"      "Stress"    "Social"
```

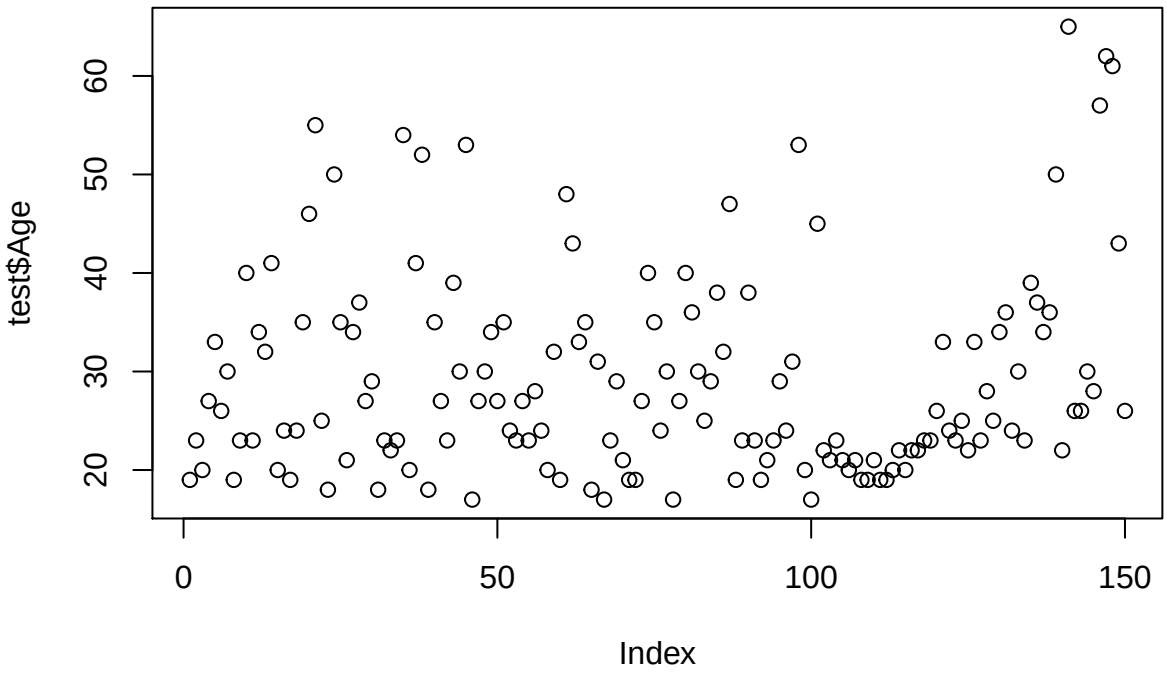
Some useful functions for basic statistics:

- *summary()*: compute a 5 number of Tukey + mean for numeric variables or frequency for categorical variables
- *plot()*: an object sensitive function, perform a barplot for categorical or dispersion diagram for numeric variables
- others: *sd()* compute standard deviation, *length()* the number of element in a vector, *dim()* the dimensions of a dataset or array, *median()* compute the median, *quantile()* calculates the quantile of a vector, *scale()* standardize a numeric vector, *IQR()* interquantile range

```
summary(test$Age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.00   22.00   26.00   29.22   34.00   65.00
```

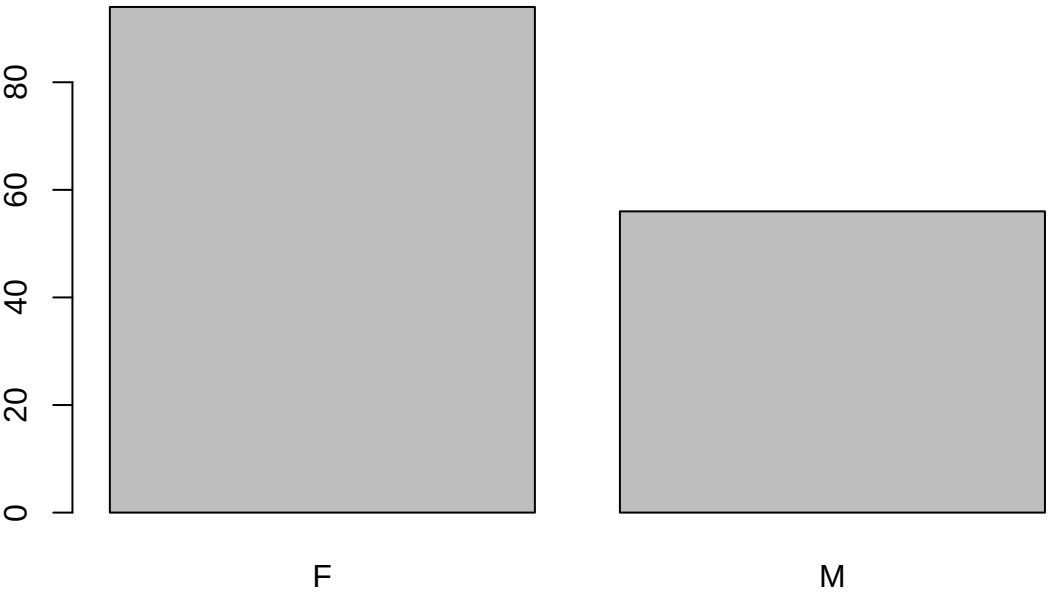
```
plot(test$Age)
```




```
summary(test$Gender)
```

```
##  F  M
## 94 56
```

```
plot(test$Gender)
```



Now we try to build a function that extract from a numeric vector mean, sd, median, IQR and the length:

```
fun_sum<-function(x){
c(M=mean(x),SD=sd(x),Me=median(x),IQR=IQR(x),n=length(x))
}
fun_sum(test$Age)
```

```
##         M        SD        Me       IQR         n
##  29.22000  10.39416  26.00000  12.00000 150.00000
```