The statistical software R

Luca Frigau

University of Cagliari

Ph.D. course Quantitative Methods
A.A. 2016/2017

Outline

What is R?

- R is a dialect of S language
- R is consistent and integrated
- R is both a language *object oriented* and a software

A brief history...

- S was developed in 1975-1976 at Bell Laboratories
- S was developed by John Chambers together with Rick Becker and Allan Wilks
- From S came out: the S-Plus software developed and marketed by Insightful, and the statistical environment open source R
- R was developed by Ross Ihaka and Robert Gentlemen at Auckland University (New Zealand)

S philosophy

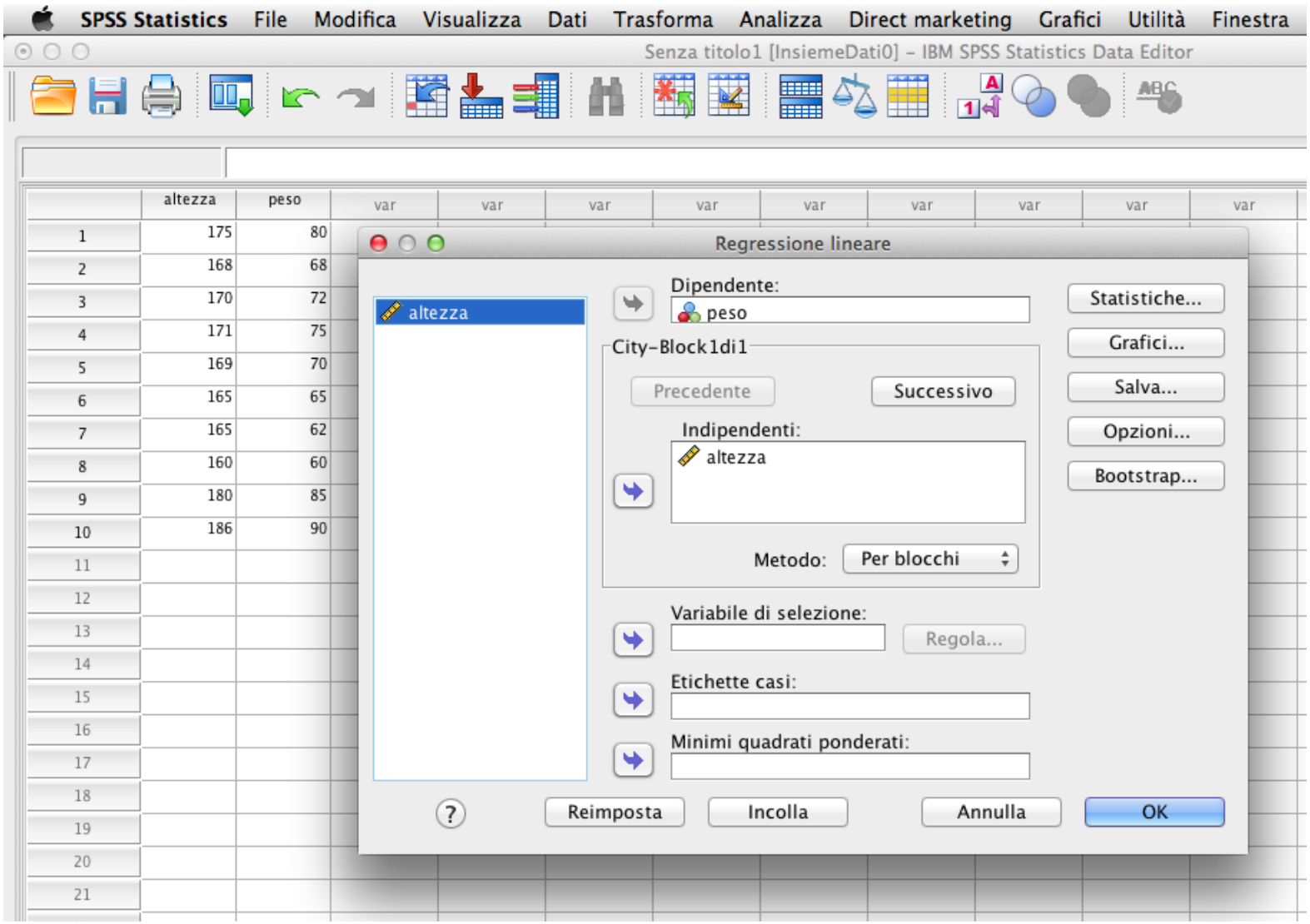
In *Stages in the Evolution of S*, John Chambers wrote

We wanted users to be able to begin in an interactive environment, where they did not consciously think of themselves as programming. Then as their needs became clearer and their sophistication increased, they should be able to slide gradually into programming, when the language and system aspects would become more important.

http://www.stat.bell-labs.com/S/history.html

Metalanguage VS Software

Software



Metalanguage VS Software

Metalanguage

```
> height = c(175, 168, 170, 171, 169, 165, 165, 160, 180, 186)
> weight = c(80, 68, 72, 75, 70, 65, 62, 60, 85, 90)
> model = lm(formula = weight ~ height, x = TRUE, y = TRUE)
> model

Call:
lm(formula = weight ~ height, x = TRUE, y = TRUE)

Coefficients:
(Intercept)       height
   -143.696        1.266
```

Metalanguage VS Software

Metalanguage

lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Arguments

`formula`	an object of class `"formula"` (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Should be `NULL` or a numeric vector. If non-NULL, weighted least squares is used with weights `weights` (that is, minimizing `sum(w*e^2)`); otherwise ordinary least squares is used. See also 'Details',
`na.action`	a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.
`method`	the method to be used; for fitting, currently only `method = "qr"` is supported; `method = "model.frame"` returns the model frame (the same as with `model = TRUE`, see below).
`model, x, y, qr`	logicals. If `TRUE` the corresponding components of the fit (the model frame, the model matrix, the response, the QR decomposition) are returned.
`singular.ok`	logical. If `FALSE` (the default in S but not in R) a singular fit is an error.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one are specified their sum is used. See `model.offset`.
`...`	additional arguments to be passed to the low level regression fitting functions (see below).

R system

R system is divided into 2 conceptual parts:

1. The 'base' R system
2. The remainder

R usability is divided into different packages:

- The 'base' R system includes the following packages: **base, utils, stats, datasets, graphics, grDevices, methods**.
- To enhance its usability it is possible to install more packages, such as **boot, class, cluster, code tools, foreign, KernSmooth, lattice, mgcv, name, rpart, survival, MASS, spatial, nnet, Matrix**.

Potentiality and diffusion

- 4914 packages are available in CRAN repository. They are developed by users and programers of all the world
- Many manuals and tutorials
- 452 blogs

Software	**Blogs**
R	452
SAS	40
Stata	8
Others	0-3

Table: Blogs for each software (May 2013). *Source r4stats*

Logic

In math...

$$y = \sqrt{x}$$

...in R

```
> object <- function(parameter1, parameter2)
```

Assignment and comments

- The operator <- assigns an output to an object

```
> x <- 6          # 6 is assigned to x
> print(x)        # print x
[1] 6

> "hello" -> y    # the word 'hello' is assigned to y
> y               # print y
[1] "hello"
```

- The character # indicates a comment

Objects

R object can be defined as a data box

R 'atomic' classes of objects:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

Objects

- Vector
- Matrix
- Dataframe
- ...

Vectors

c() function is used for creating vector objects

```
> x <- c(0.61, 0.76)      # numeric
> x <- c(4,5,6,7,8)       # numeric
> x <- 4:8                # integer
> x <- c(4L,5L,6L,7L,8L)  # integer
> x <- c(4+0i, 3+6i)      # complex
> x <- c("a", "b", "c")   # character
> x <- c(TRUE, FALSE)     # logical
> x <- c(T, F)            # logical
```

If a mixing vector is built, i.e. a vector made up of objects of different classes, R applies a coercion in order to have elements of a same class

```
> y <- c(1.7, "hello")    # character
> y <- c(TRUE, 8)         # numeric
> y <- c("hello", TRUE)   # character
```

Matrix

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
> m <- matrix(nrow = 4, ncol = 2)
> m
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
[3,]   NA   NA
[4,]   NA   NA
> dim(m)
[1] 4 2
> attributes(m)
$dim
[1] 4 2
```

Matrices can be made up of elements of a same class as vectors

Matrix

Matrices are built *column-wise*, starting in the 'upper left' corner and running down the columns

```
> m <- matrix(1:12, nrow = 3, ncol = 4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Matrix

Matrices can be built even through *column-binding* or *row-binding*, with the function **cbind()** and **rbind()**

```
> x <- c(1,2,3,4)
> y <- c(7,8,9,10)
> cbind(x, y)
     x  y
[1,] 1  7
[2,] 2  8
[3,] 3  9
[4,] 4 10

> rbind(x, y)
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    7    8    9   10
```

Data frame

Data frame features:

- it stores tabular data
- it can store objects of different classes
- it has a special attribute called row.names

```
> x <- data.frame(years = c(21,34,67,45), sex = c("M","M","F","M"))
> x
  years sex
1    21   M
2    34   M
3    67   F
4    45   M
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Missing values

Missing values are denoted by

- NA (*not available*)
- NaN (*not a number*) for undefined mathematical operations

Functions for identifying NA and NaN

```
> x <- c(2,3,0/0,NA)
> is.nan(x)                       # function for NaN
[1] FALSE FALSE  TRUE FALSE
> is.na(x)                        # function for NA
[1] FALSE FALSE  TRUE  TRUE
```

A NaN value is also NA but the converse is not true

Data importation

physical

logical

Data importation

Functions

- **read.table()**
- **read.csv()**
- **read.csv2()**
- **read.delim()**
- **read.delim2()**

Main parameters

- header
- sep
- dec
- row.names
- col.names

Data importation

read.table {utils} R Documentation

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Object structure

Several functions exist to get information about object structure

str()

To get information about object structure

```
> years = c(21L,34L,67L,45L,33L)
> sex = c("M","M","F","M","F")
> status = c(TRUE,TRUE,TRUE,FALSE,FALSE)
> weight = c(81.4,72.3,54.8,76.4,47.3)
> x <- data.frame(years,sex,status,weight)

> str(x)
'data.frame': 5 obs. of  4 variables:
 $ years : int  21 34 67 45 33
 $ sex   : Factor w/ 2 levels "F","M": 2 2 1 2 1
 $ status: logi  TRUE TRUE TRUE FALSE FALSE
 $ weight: num  81.4 72.3 54.8 76.4 47.3
```

Object structure

summary()

To get general statistics

```
> summary(x)
     years     sex      status             weight
 Min.   :21   F:2   Mode :logical   Min.   :47.30
 1st Qu.:33   M:3   FALSE:2         1st Qu.:54.80
 Median :34         TRUE :3         Median :72.30
 Mean   :40         NA's :0         Mean   :66.44
 3rd Qu.:45                         3rd Qu.:76.40
 Max.   :67                         Max.   :81.40
```

Installation

1. The R Project for Statistical Computing
2. Comprehensive R Archive Network (CRAN)
3. Choose the nearest *location* for downloading
4. Install it following instructions

Working directory

Working directory is the default fold for swapping physical files between R and PC

- **getwd()** gets the working directory path
- **setwd()** sets the working directory path
- **dir()** provides a list of files located in the working directory

Workspace

Workspace is the current R working environment and includes any user-defined objects

- **ls()** provides a list of all objects in workspace

```
> a <- 4
> b <- 6
> ls()
[1] "a" "b"
```

- **rm()** removes objects from workspace

```
> rm(a)              # object a is removed
> rm(list = ls())    # all objects are removed
```

Library

About libraries it is necessary to separate

- installed libraries
- loaded libraries

install.packages() for installing new libraries

require() or **library()** for loading libraries already installed

Subsetting

Several operators allow to extract subsets of R objects

- $ is used to extract elements by name

```
> x <- data.frame(years = c(21,34,67,45), sex = c("M","M","F","M"))
> x
  years sex
1    21   M
2    34   M
3    67   F
4    45   M

> x$years
[1] 21 34 67 45

> x$sex
[1] M M F M
Levels: F M
```

Subsetting

- [ ] returns an object of the same class as the original. It can be used to select more than one element

```
> x
  years sex
1    21   M
2    34   M
3    67   F
4    45   M
> x[3,1]
[1] 67

> x[c(1,4),2]
[1] M M
Levels: F M
```

Subsetting

```
> x[2,]
  years sex
2    34   M

> x[x$years>30,1]
[1] 34 67 45

> x[x$years < 30 | x$years > 60,]
  years sex
1    21   M
3    67   F

> x[-c(1,4),2]
[1] M F
Levels: F M
```

Subsetting

Logical operators	
&	AND
|	OR
!	NOT
==	equals
!=	not equal to
<	less than
>	greater than

Data importation

The five classical functions

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)
```

Data importation

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Data importation

Is it possible to import data directly from an Excel file? **Definitely!!!**

```
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL,
  startRow=NULL, endRow=NULL, colIndex=NULL,
  as.data.frame=TRUE, header=TRUE, colClasses=NA,
  keepFormulas=FALSE, encoding="unknown", ...)

read.xlsx2(file, sheetIndex, sheetName=NULL, startRow=1,
  colIndex=NULL, endRow=NULL, as.data.frame=TRUE, header=TRUE,
  colClasses="character", ...)
```

Data importation

```
> install.packages("xlsx")
> library(xlsx)

> dati <- read.xlsx2("dati.xlsx",1)
```

It is better to import data from .csv or .txt files for efficiency issue and for avoiding problems connected with Excel file structure (formulas, etc.)

Importation time is less

Data importation

Main parameters in data importation

Parameter	**Description**
file	file name
header	it indicates whether the file contains the names of the variables as its first line
sep	the field separator character
dec	the character used in the file for decimal points
row.names	a vector of row names

Data exportation

For exporting tabular data several function can be used

The three most common function are **write.table()**, **write.csv()** and **write.csv2()**

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")

write.csv(...)
write.csv2(...)
```

Many parameters are the same of data importation functions

Data exportation

It is possible to export data even in Excel format, of course!

```
write.xlsx(x, file, sheetName="Sheet1",
  col.names=TRUE, row.names=TRUE, append=FALSE, showNA=TRUE)

write.xlsx2(x, file, sheetName="Sheet1",
  col.names=TRUE, row.names=TRUE, append=FALSE, ...)
```

The efficiency problem keeps staying

Object attributes

To get object attributes

names()

To get the names of an object

```
> years = c(21L,34L,67L,45L,33L)
> sex = c("M","M","F","M","F")
> status = c(TRUE,TRUE,TRUE,FALSE,FALSE)
> weight = c(81.4,72.3,54.8,76.4,47.3)
> x <- data.frame(years,sex,status,weight)
> names(x)
[1] "years" "sex" "status" "weight"
```

dim()

To retrieve the dimension of an object

```
> dim(x)
[1] 5 4
```

Object attributes

class()

To retrieve the class of an object

```
> class(x)
[1] "data.frame"
> class(x$years)
[1] "integer"
```

length()

To get the length of vectors

```
> length(x$weight)
[1] 5
```

Explicitly coercion

It is possible to set objects as specific classes by **as.*** functions

```
> x <- 4:7
> class(x)
[1] "integer"

> as.numeric(x)
[1] 4 5 6 7

> as.logical(x)
[1] TRUE TRUE TRUE TRUE

> as.character(x)
[1] "4" "5" "6" "7"

> as.complex(x)
[1] 4+0i 5+0i 6+0i 7+0i
```

Explicitly coercion

No sense explicitly coercions generate NAs

```
> x <- c("a", "b", "c")

> as.numeric(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion

> as.logical(x)
[1] NA NA NA
```

Factors

Factors are composed objects. They features each element is assigned to a specific level.

Factors describe elements with a finite number of values. They are used for representing categorical data

```
> x <- factor(c("M", "F", "F", "M", "M"))
> x
[1] M F F M M
Levels: F M
> class(x)
[1] "factor"

> a <- c(1,2,3,2,1)
> a
[1] 1 2 3 2 1
> a <- as.factor(a)
> a
[1] 1 2 3 2 1
Levels: 1 2 3
```

Vector reference

In some case it is useful to reference a specific vector of a data frame. As previously said it is common to use $ operator

```
> x$weight
[1] 81.4 72.3 54.8 76.4 47.3
```

As an alternative, it is possible to use **attach()** function, in order to reference directly its name

```
> attach(x)
The following objects are masked _by_ .GlobalEnv:

    years, status, sex, weight

> weight
[1] 81.4 72.3 54.8 76.4 47.3
```

For removing the effect of **attach()** it is necessary to use **detach()** function

Object and its attributes modification

Object attribute modification: factor levels

```
> x <- factor(c("M", "F", "F", "M", "M"))
> levels(x)
[1] "F" "M"
> levels(x) <- c("Female", "Male")     # level modification
> x
[1] Male   Female Female Male   Male
Levels: Female Male
```

Object and its attributes modification

Object attribute modification: variable name

```
> x <- data.frame(ye = c(21L,34L,67L,45L,33L),
+ se = c("M","M","F","M","F"),
+ st = c(T,T,T,F,F),
+ we = c(81.4,72.3,54.8,76.4,47.3))

> names(x)
[1] "ye"  "se" "st" "we"
> names(x) <- c("years", "sex", "status", "weight")
> names(x)
[1] "years" "sex" "status" "weight"
```

Cutting in classes

```
> weight <- c(81.4,72.3,54.8,76.4,47.3)
> class(weight)
[1] "numeric"

> weight2 <- cut(weight, breaks = c(0,40,60,80,100))
> weight2
[1] (80,100] (60,80]  (40,60]  (60,80]  (40,60]
Levels: (0,40] (40,60] (60,80] (80,100]
> class(weight2)
[1] "factor"
```

Cutting in classes and ordered

```
> ages <- c(84,65,54,47,68,34)

> ageClasses <- ifelse(ages > mean(ages), "old", "young")
> class(ageClasses)
[1] "character"

> (ageClasses <- as.factor(ageClasses))
[1] old   old   young young old   young
Levels: old young

> (ageClasses <- ordered(ageClasses, levels = c("young", "old")))
[1] old   old   young young old   young
Levels: young < old

> class(ageClasses)
[1] "ordered" "factor"
```

Basic statistics

Elementary operations

```
> x <- c(3,4,6,7,12,4)

> sum(x)       # sum
[1] 36
> prod(x)      # product
[1] 24192
> cumsum(x)    # cumulative sum
[1]  3  7 13 20 32 36
> cumprod(x)   # cumulative product
[1]     3    12    72   504  6048 24192
```

Basic statistics

Elementary statistics

```
> summary(x)    # General statistics
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00    4.00    5.00    6.00    6.75   12.00
> min(x)        # Minimum
[1] 3
> max(x)        # Maximum
[1] 12
> median(x)     # Median
[1] 5
> mean(x)       # Mean
[1] 6
> sd(x)         # Standard deviation
[1] 3.286335
> var(x)        # Variance
[1] 10.8
```

Basic statistics

mean()

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

```
> a <- c(4,6,7,12,5,7,3,2,9,88,NA)
> mean(a)
[1] NA
> mean(a, na.rm = T)
[1] 14.3
> mean(a, na.rm = T, trim=.1)
[1] 6.625
```

Basic statistics

var(), sd()

```
var(x, na.rm = FALSE)
sd(x, na.rm = FALSE)
```

```
> a <- c(4,6,7,12,5,7,3,2,9,88,NA)
> var(a)
[1] NA
> var(a, na.rm = T)
[1] 679.1222

> sd(a)
[1] NA
> sd(a, na.rm = T)
[1] 26.05997
```

Variance is *unbiased*, denominator $= (n - 1)$

Basic statistics

cor()

```
cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
```

```
> x <- c(3,4,6,7,12,4)
> y <- c(43,65,72,54,6,89)
> cor(x,y)
[1] -0.7382895
```

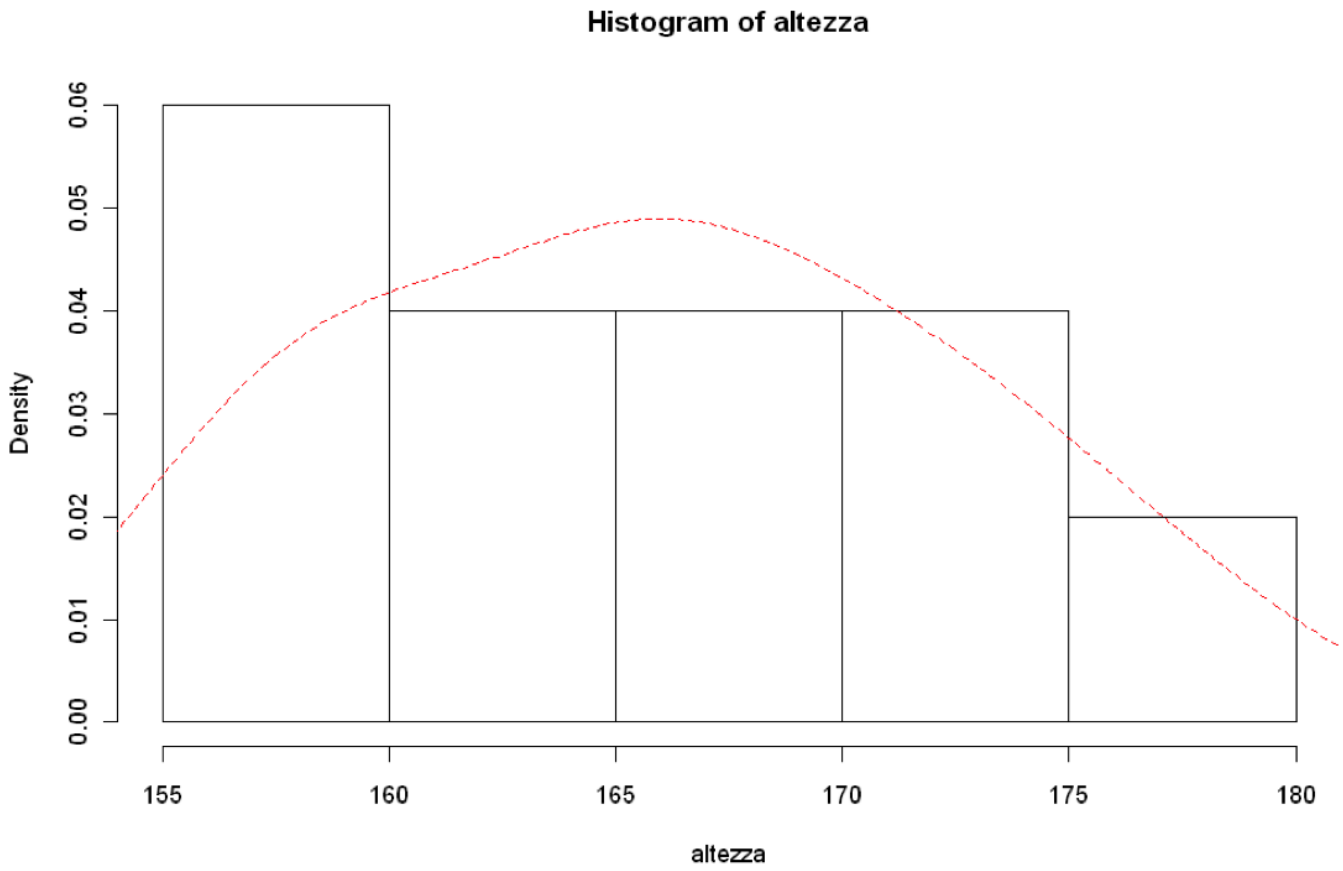
Basic statistics

```
> x <- c(3,4,6,7,12,4)

> quantile(x)                    # Quantiles
   0%   25%   50%   75%  100%
 3.00  4.00  5.00  6.75 12.00

> quantile(x, probs = 0.02)     # Specific quantile
 2%
3.1

> table()      # Contingency table
> ftable()     # "Flat" contingency table
> length()     # Length of vector
```

Basic plots

hist()

```
> x <- c(3,4,6,7,12,4)
> hist(x)
```



Basic plots

pie()

```
> x <- c("M", "M", "M", "F", "F")
> pie(x)

Error in pie(x) : 'x' values must be positive.
```

Basic plots

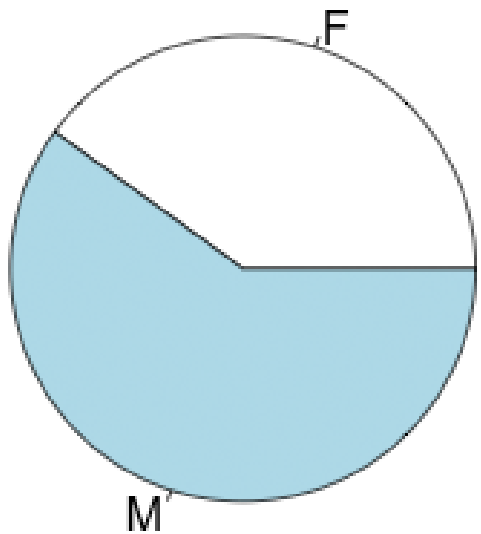
table() function uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

```
> x <- c("M", "M", "M", "F", "F")

> table(x)
x
F M
2 3
```

Basic plots

pie()

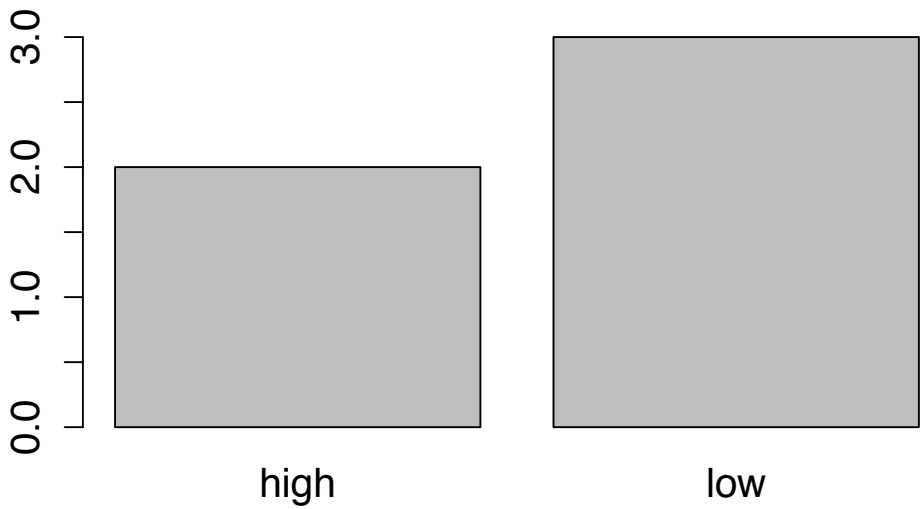
```
> pie(table(x))
```



Basic plots

barplot()

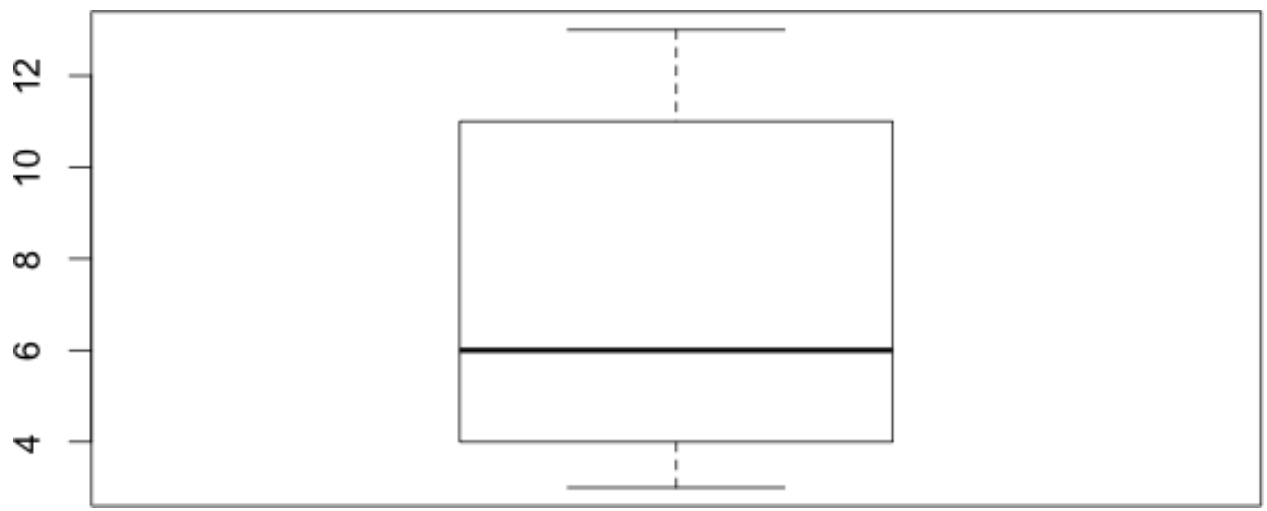
```
y <- c("high", "high", "low", "low", "low")

barplot(table(y))
```



Basic plots

boxplot()

```
x <- c(3,4,6,7,12,4,10,4,12,13,5)
boxplot(x)
```



Basic plots

stem()

```
> x <- c(3,24,26,7,12,24,20,4,12,13,5)
> stem(x)

The decimal point is 1 digit(s) to the right of the |

  0 | 34
  0 | 57
  1 | 223
  1 |
  2 | 044
  2 | 6
```

Bivariate plots

Conditional Boxplot

A quantitative variable respect to categories of a qualitative variable

```
> age<-c(31,53,72,22,43,83,32)
> sex<-c("M","M","F","F","F","M","F")

> boxplot(age ~ sex)
> boxplot(split(age,sex))

> split(age,sex)
$F
[1] 72 22 43 32

$M
[1] 31 53 83
```

$\sim$ is got keying ALT + 126 (windows) o ALT + 5 (mac OS)

Grafici Bivariate plots

Mosaic

Two or more qualitative variables

```
> sex <- c("M","M","M","F","F","F","M","F","M")
> income<-c("low","low","medium","high","low",
+ "high","low","medium","high")

> plot(table(income,sex))

> plot(table(sex,income))
```

Do you note some difference?

General plot options

par() can be used to set graphical parameters

```
> par(bg = "green")      # Window background color

> par(cex = 0.8)         # Dimension of plotting text and symbols

> par(mfrow = c(2, 1))   # For dividing window
>                        # (here in 2 rows and 1 column)
```

General plot options

IMPORTANT

- Parameters managed by **par()** must be set before plot creation

```
> par(bg = "green")
> plot(...)
```

- When you close the *device*, **par()** parameters get back to default

General plot options

par() can manage several parameters

```
> ?par
```

It is possible to set more parameters together

```
> par(mfrow = c(2, 1), bg = "lightgreen", cex=0.8)
> plot(x)
> plot(y)
```

General plot options

To add a title or a subtitle: **main()** and **sub()**

```
> hist(x, main = "Title", sub = "Subtitle")
```

In a pinch it is possible to use **title()**

```
> plot(x)
> title(main = "Title", sub = "Subtitle")
```

General plot options

To change line or point style **lty**, **pch** and **lwd** parameters are used

```
> z <- rnorm(100)

> plot(density(z), lty=2)          # lty draws the lines

> plot(z, pch=2)                   # pch draws the points

> plot(density(z), lty=2, lwd=2) # lwd draws both the line and point width
```

General plot options

pch values

General plot options

To set colors it is possible

- to use **rainbow** function

```
> sex <- factor(c("F","F","M","F"))

> pie(table(sex), col = rainbow(2))

> pie(table(sex), col = rainbow(length(levels(sex))))
```

- to set them manually

```
> pie(table(sex), col = c("pink","blue"))
> colors()        # it gets color list
```

Debugging

Indications that something is not right

- **message**: A generic notification/diagnostic message produced by the message function; execution of the function continues
- **warning**: An indication that something is wrong but not necessarily fatal; execution of the function continues; generated by the warning function
- **error**: An indication that a fatal problem has occurred; execution stops; produced by the stop function

Debugging

warning

```
> log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
```

error

```
> plot()
Error in xy.coords(x, y, xlabel, ylabel, log) :
  l'argomento "x"  non è specificato e non ha un valore predefinito
```

Data saving

R uses two kind of outputs

- **workspace**: it has a filename extension .RData and stores data and objects created during working session

```
> save.image(file = "output.RData") # save all workspace
> save(x,y,z, file = "xyz.RData")   # save objects x y z

> load("xyz.RData")                 # load saved objects
```

- **history**: it has a filename extension .Rhistory and stores list of all code keyed during working session

```
> savehistory(file = "crono.Rhistory")   # save history

> loadhistory("crono.Rhistory")          # load history
```

frigau@unica.it