R Basics

Barbara Guardabascio

What is R?

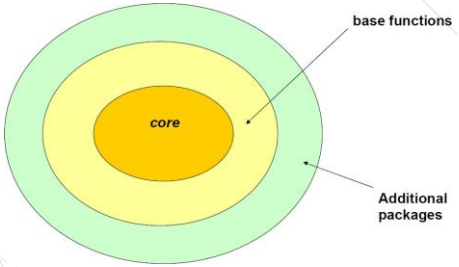
R is a language and free software environment widely used among statisticians and data miners for developing statistical software and data analysis.

(https://it.wikipedia.org/wiki/R_(software))

Main features:

- open-source
- First Version R 0.90.0 - 1999 November the 22th
- Last Version R 4.1.0 - 2021 May the 18th
- Official website https://www.r-project.org/

R Structure



- **core** can be modified only by software developers
- **base functions** include basic and new statistical methods
- R users could modify **base functions** or add new **functions**
- a set of **functions** developed to solve a given problem are generally released as a **package**

(https://cran.r-project.org/web/packages/)

Why R?

- ✓ **R pros**
  - Free (as in free beer and as in free spreech)
  - Versatile
  - Based on command lines: reproducible

- ✓ **R cons**
  - Based on command lines: not user friendly
  - (Still) slightly cumbersome with big data

Download R

https://cran.r-project.org/

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2017-11-30, Kite-Eating Tree) R-3.4.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Download R

https://cran.r-project.org/

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R for Windows

Subdirectories:

base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Download R

https://cran.r-project.org/

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R-3.4.3 for Windows (32/64 bit)

Download R 3.4.3 for Windows (62 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

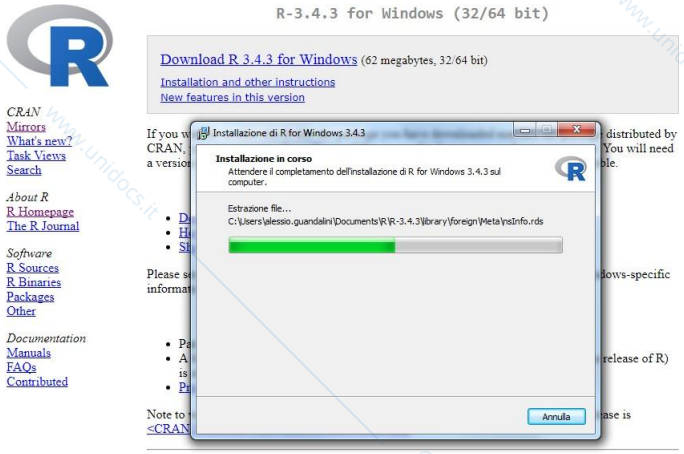
- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.htm.

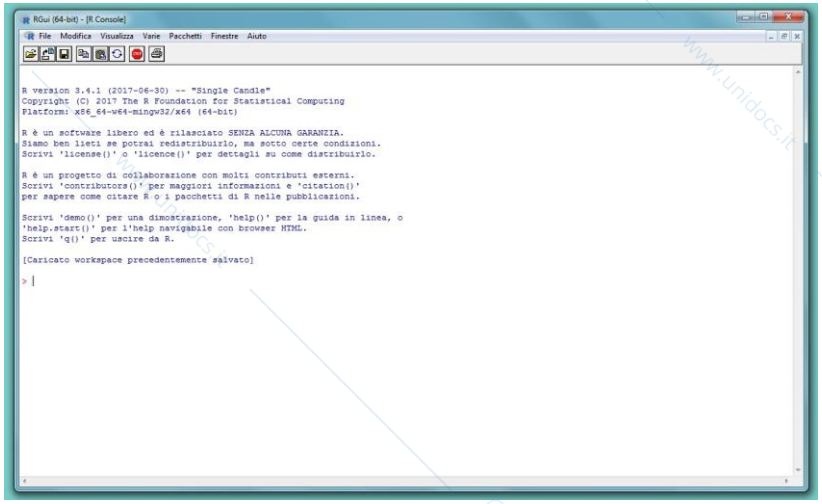
Last change: 2017-12-06

Download R

https://cran.r-project.org/



Download R



Download and install R packages

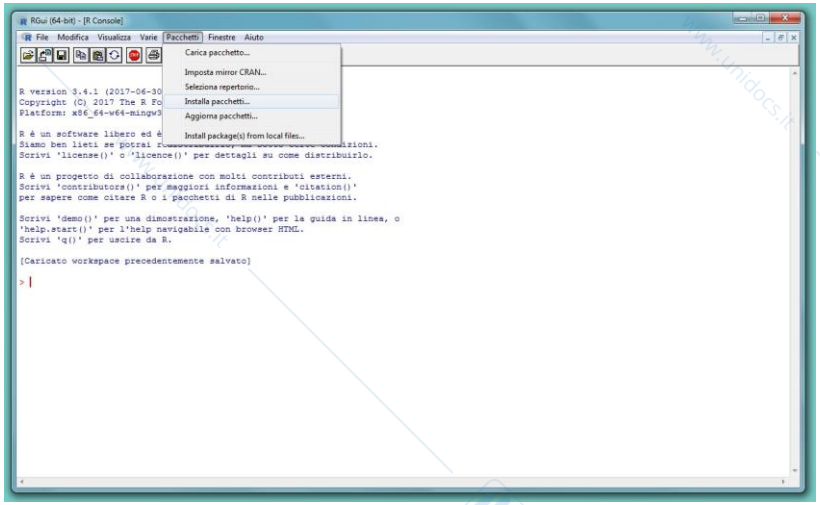
Three possible solutions:

1. Use directly **CRAN** (Comprehensive R Archive Network)
2. Install from **local files**
3. From **console**

```
install.packages("pack")
```

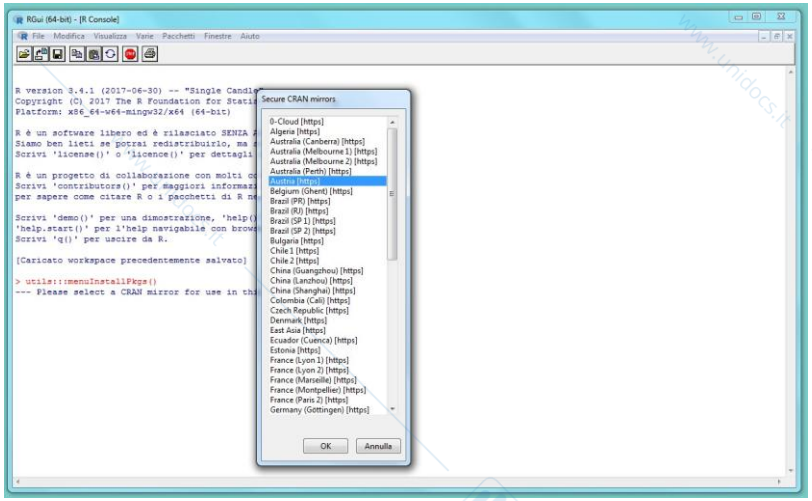
Download and install R packages

1 – From CRAN (Comprehensive R Archive Network)



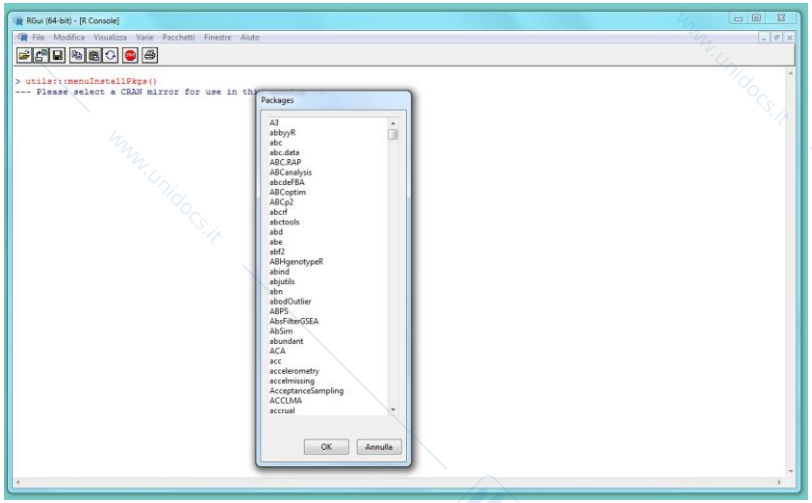
Download and install R packages

1 – From CRAN (Comprehensive R Archive Network)



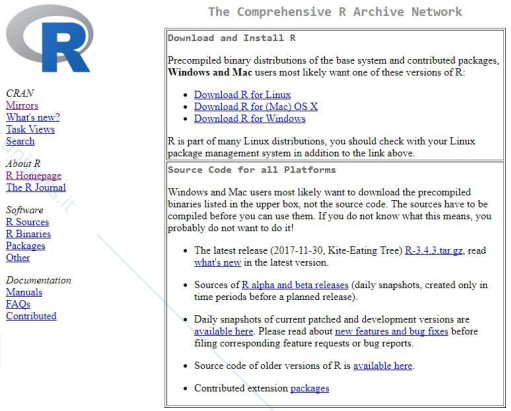
Download and install R packages

1 – From CRAN (Comprehensive R Archive Network)



Download and install R packages

2 – From local files



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

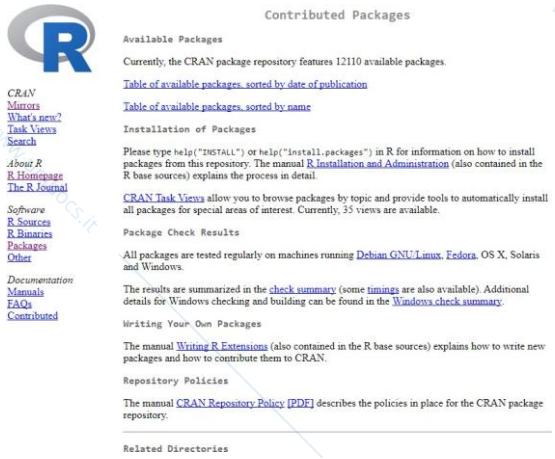
Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2017-11-30, Kite-Eating Tree) R-3.4.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Download and install R packages

2 – From local files



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Contributed Packages

Available Packages

Currently, the CRAN package repository features 12110 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type help("INSTALL") or help("install.packages") in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 35 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, OS X, Solaris and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Writing Your Own Packages

The manual Writing R Extensions (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Repository Policies

The manual CRAN Repository Policy [PDF] describes the policies in place for the CRAN package repository.

Related Directories

Download and install R packages

2 – From local files

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

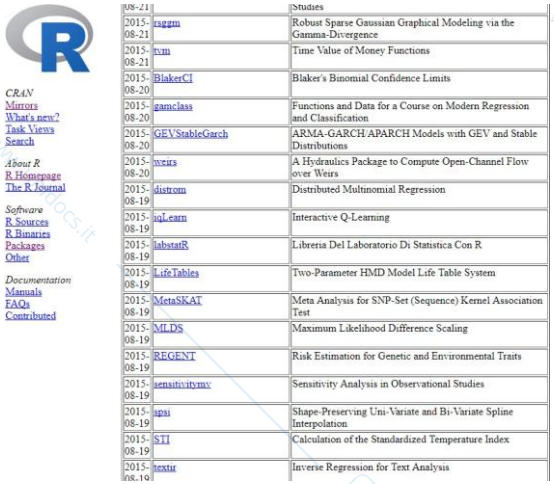
Documentation
Manuals
FAQs
Contributed

Available CRAN Packages By Date of Publication

Date	Package	Title
2018-01-30	ABPS	The Abnormal Blood Profile Score to Detect Blood Doping
2018-01-30	BeSS	Best Subset Selection in Linear, Logistic and CoxPH Models
2018-01-30	BoSSA	A Bunch of Structure and Sequence Analysis
2018-01-30	DataVisualizations	Visualizations of High-Dimensional Data
2018-01-30	desctable	Produce Descriptive and Comparative Tables Easily
2018-01-30	errorist	Automatically Search Errors or Warnings
2018-01-30	FENmlm	Fixed Effects Nonlinear Maximum Likelihood Models
2018-01-30	findR	Find R Scripts, R Markdown, PDF and Text Files by Content with Pattern Matching
2018-01-30	ggQC	Quality Control Charts for 'ggplot'
2018-01-30	gnorm	Generalized Normal Exponential Power Distribution
2018-01-30	iotables	Importing and Manipulating Symmetric Input-Output Tables
2018-01-30	mlegp	Maximum Likelihood Estimates of Gaussian Processes
2018-01-30	morse	Modelling Tools for Reproduction and Survival Data in Ecotoxicology
2018-01-30	mr.raps	Two Sample Mendelian Randomization using Robust Adjusted Profile Score
2018-01-30	onewaytests	One-Way Tests in Independent Groups Designs

Download and install R packages

2 – From local files



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

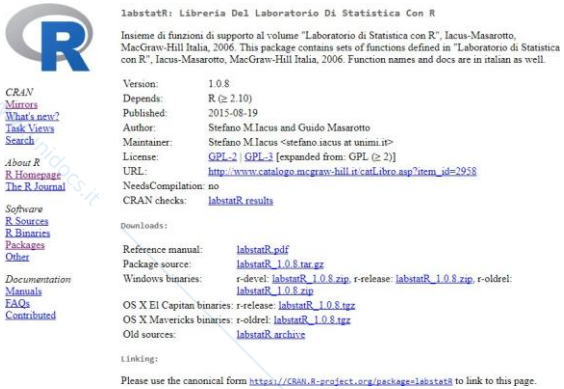
Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

		Studies
2015-08-21	rsggm	Robust Sparse Gaussian Graphical Modeling via the Gamma-Divergence
2015-08-21	tvm	Time Value of Money Functions
2015-08-20	BlakerCI	Blaker's Binomial Confidence Limits
2015-08-20	gamclass	Functions and Data for a Course on Modern Regression and Classification
2015-08-20	GEVStableGarch	ARMA-GARCH/APARCH Models with GEV and Stable Distributions
2015-08-20	weirs	A Hydraulics Package to Compute Open-Channel Flow over Weirs
2015-08-19	distrom	Distributed Multinomial Regression
2015-08-19	iqLearn	Interactive Q-Learning
2015-08-19	labstatR	Libreria Del Laboratorio Di Statistica Con R
2015-08-19	LifeTables	Two-Parameter HMD Model Life Table System
2015-08-19	MetaSKAT	Meta Analysis for SNP-Set (Sequence) Kernel Association Test
2015-08-19	MLDS	Maximum Likelihood Difference Scaling
2015-08-19	REGENT	Risk Estimation for Genetic and Environmental Traits
2015-08-19	sensitivitymv	Sensitivity Analysis in Observational Studies
2015-08-19	spsi	Shape-Preserving Uni-Variate and Bi-Variate Spline Interpolation
2015-08-19	STI	Calculation of the Standardized Temperature Index
2015-08-19	textir	Inverse Regression for Text Analysis

Download and install R packages

2 – From local files



labstatR: Libreria Del Laboratorio Di Statistica Con R

Insieme di funzioni di supporto al volume "Laboratorio di Statistica con R", Iacus-Masarotto, MacGraw-Hill Italia, 2006. This package contains sets of functions defined in "Laboratorio di Statistica con R", Iacus-Masarotto, MacGraw-Hill Italia, 2006. Function names and docs are in italian as well.

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Version:	1.0.8
Depends:	R (≥ 2.10)
Published:	2015-08-19
Author:	Stefano M.Iacus and Guido Masarotto
Maintainer:	Stefano M.Iacus <stefano.iacus at unimi.it>
License:	GPL-2 | GPL-3 [expanded from: GPL (≥ 2)]
URL:	http://www.catalogo.mcgraw-hill.it/catLibro.asp?item_id=2958
NeedsCompilation:	no
CRAN checks:	labstatR results

Downloads:

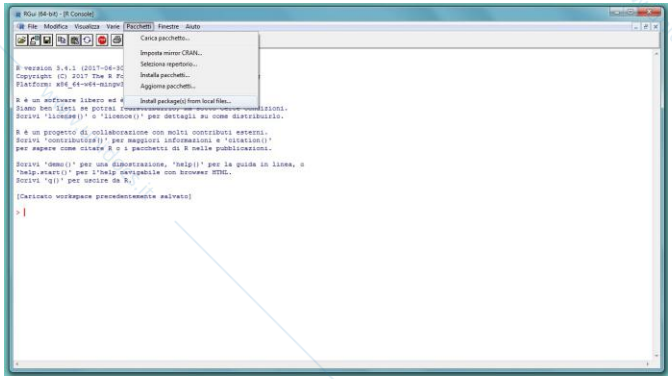
Reference manual:	labstatR.pdf
Package source:	labstatR_1.0.8.tar.gz
Windows binaries:	r-devel: labstatR_1.0.8.zip, r-release: labstatR_1.0.8.zip, r-oldrel: labstatR_1.0.8.zip
OS X El Capitan binaries:	r-release: labstatR_1.0.8.tgz
OS X Mavericks binaries:	r-oldrel: labstatR_1.0.8.tgz
Old sources:	labstatR archive

Linking:

Please use the canonical form https://CRAN.R-project.org/package=labstatR to link to this page.

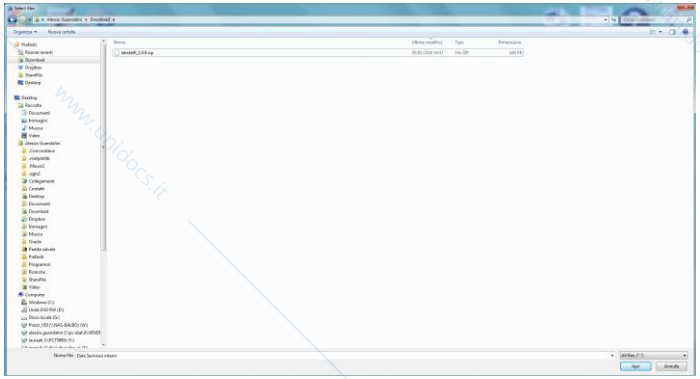
Download and install R packages

2 - From local files



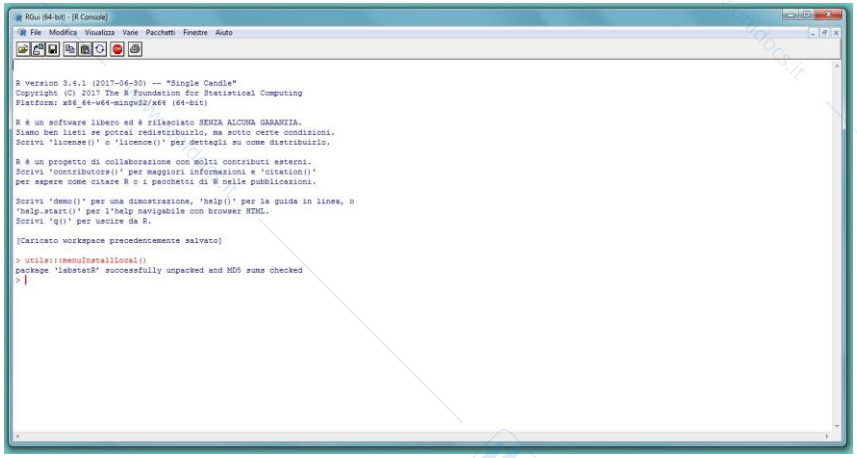
Download and install R packages

2 - From local files

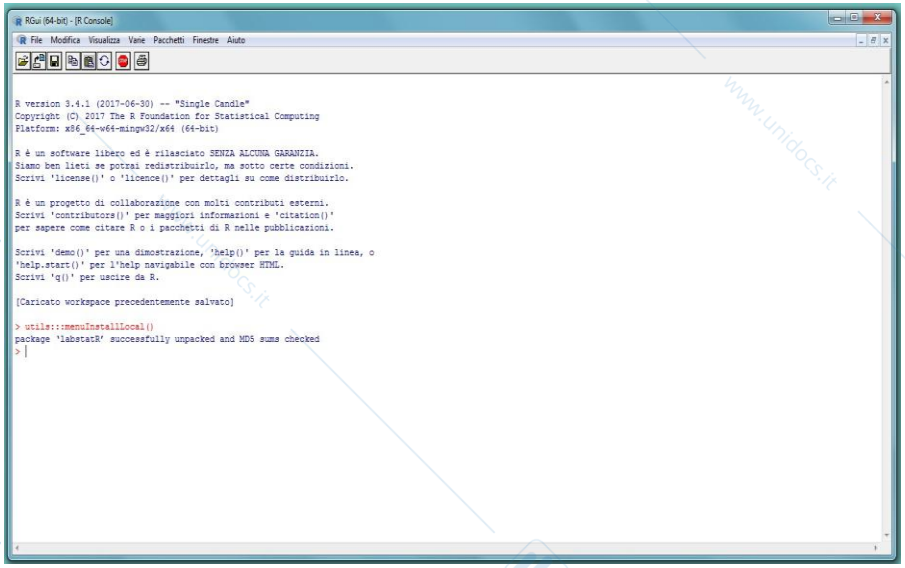


Download and install R packages

2 - From local files



Download and install R packages



Load packages in R

Once packages have been dowloaded to use them they need to be loaded in R.

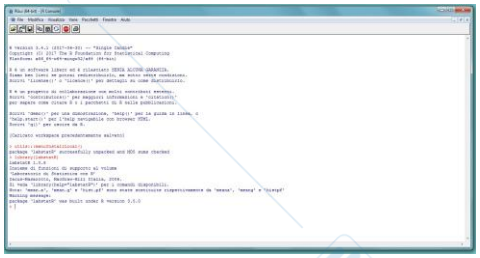
Two possible solutions:

1. From drop down menu:

   `Packages > Load … package selection`

2. From console:

   `library("package.name")`



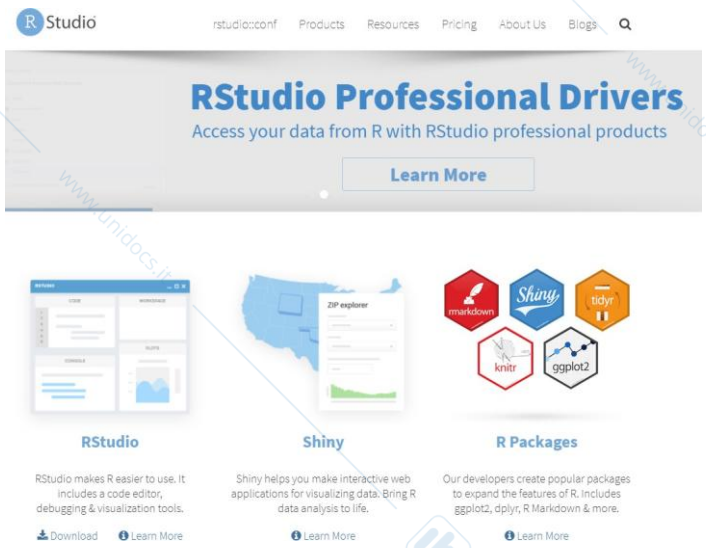
RStudio is an integrated development environment (IDE) for R

Main features

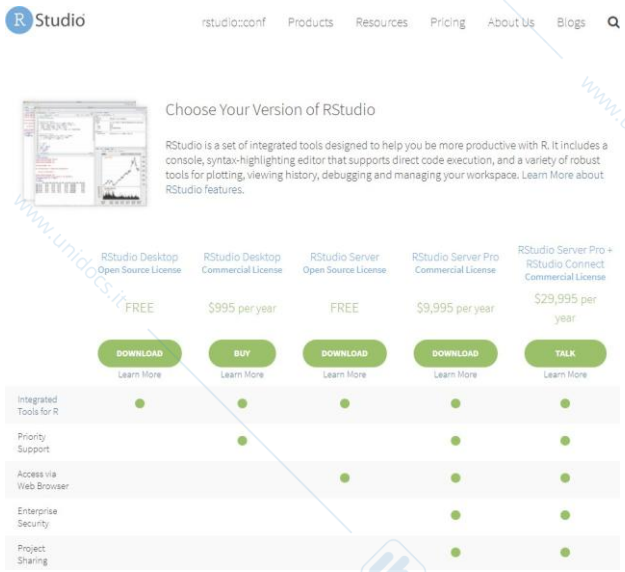
- ✓ Open source
- ✓ Two formats:
  - *Desktop*
  - *Server*

Official website https://www.rstudio.com/

R Studio - Download

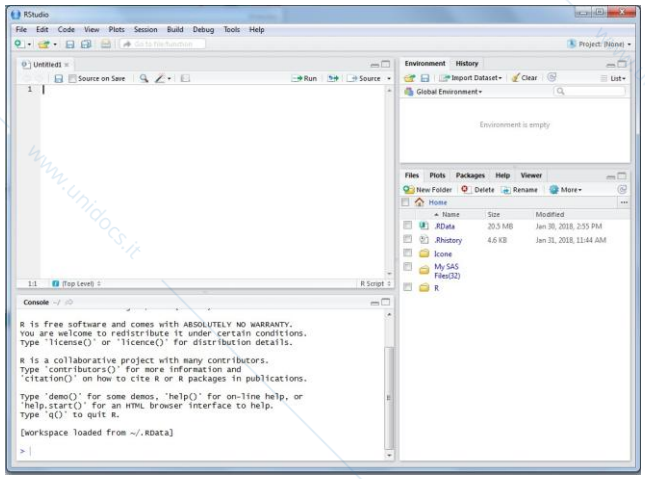


R Studio - Download

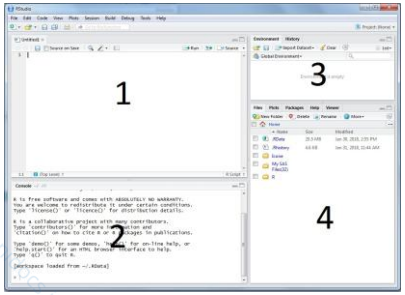


R Studio - Download

R Studio - Environment



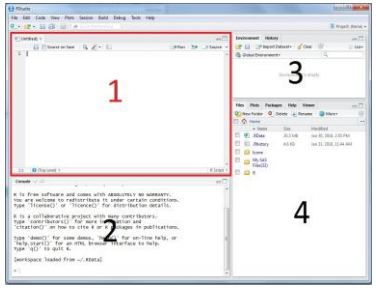
R Studio - Environment



4 sections:

1. Script
2. Console
3. Enviroment/History
4. Files/Plots/Packages/Help/Viewer

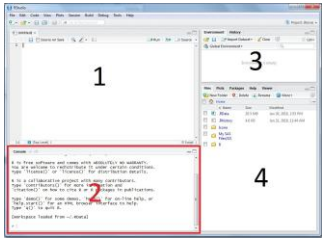
R Studio - Environment



Script

- Is text file which includes the command to run
- The file can be saved and run again without re-writing its command lines
- Once saved, its format is .R

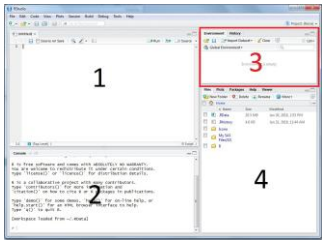
R Studio - Console



Console

The environment in which commands are placed and where results can be observed

R Studio - History



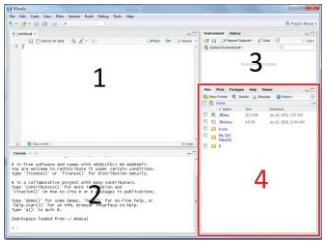
Enviroment

It provides a **list** of all active **objects** in the session

History

It is the timeline of all the **commands** that have been run in the session

R Studio – Plots/Packages/Help



Plots

Is the window in which **plots** are listed

Packages

Allows you to manage the installation and loading of **packages**

Help

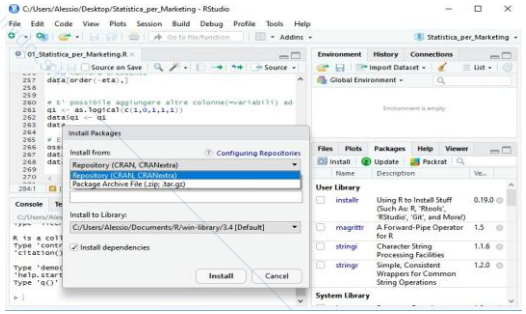
Is the window in which **help tabs** are displayed

R Studio – Install packages

To install packages in R Studio is sufficient a clic on «Install» displayed in the Packages window.

So that you can install a package:

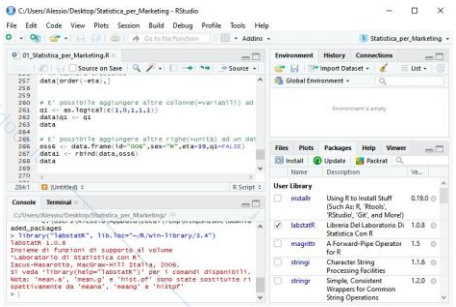
- Directly from CRAN (Comprehensive R Archive Network)
- From a local file.



R Studio – Load Packages

To load **packages** in **R Studio** is sufficient:

tick the box with the package name in the "Packages" window



- From console

```
library(package.name)
```

R programming: preliminaries (Summary)

Install: http://www.r-project.org/

Manual: https://cran.r-project.org/manuals.html

Quit: `q()`;

- Getting help
  `help('plot')`
  `?plot`
- Lots on the web (ask Google)

- Comments: `#` symbol (useful in text files with your commands to reproduce an analysis)

- Extended functionalities in libraries. Some come with the vanilla installation (e.g., `library(MASS)`)

Working directory (*01_SL_Intro_R.R*)

The working directory is where **R** looks for files and saves the results.
To set the working directory in **R** is possible to use the command line `getwd()`

```
> getwd()
[1] "C:/Users/guardabascio/Documents"
```

To change the working directory it is possible to use the command line `setwd()` specifing the path of the new working directory

```
>  setwd("C:/Users/guardabascio/Desktop")
```

Let's check the path to be sure it is changed:

```
>  setwd()
[1] "C:/Users/guardabascio/Desktop"
```

Workspace

The workspace collects all the **objects** created during the work session, as well as the syntax used to create them and saved in the history.

When the program is closed, a **dialog window** asks if you want to save the workspace. If you accept, **two unnamed files** with an extension are saved in the work-book:

1. `.Rdata`
2. `.Rhistory`

The default **R** setting foresees that the next time the program is opened, the previously saved workspace is automatically loaded.

Variable: information in an object, give it a name (can not start with number or special character)

Variables are case sensitive, so “A” and “a” are two different variables.

Assignment: <- and =

- A<-3; a<-4
- Everything stays in a workspace which can be saved on the current directory (save.image())
- To list the workspace, ls()
- To erase everything, rm(list=ls())

Arithmetic Operators in R

Sum "+" o "sum()"

```
> 2+2
[1] 4
> sum(2,2)
[1] 4
```

Minus "-"

```
> 4-2
[1] 2
```

Help

To have access to the documentation pages for all **R** functions, it is possible to use the help operator.

To recall the help operator it is possible to use two different command lines:

- ?function.name

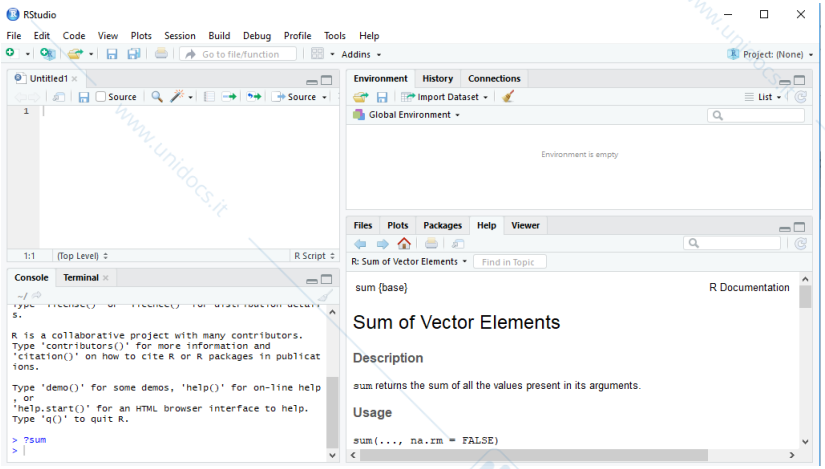
```
> ?sum
```

- Help(function.name)

```
> help(sum)
```

Help

Help information are displayed in the "Help" section



sum {base} R Documentatio

Sum of Vector Elements

Description

`sum` returns the sum of all the values present in its arguments.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`
: numeric or complex or logical vectors.

`na.rm`
: logical. Should missing values (including `NaN`) be removed?

Details

This is a generic function: methods can be defined for it directly or via the `Summary` group generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the first argument.

If `na.rm` is `FALSE` an `NA` or `NaN` value in any of the arguments will cause a value of `NA` or `NaN` to be returned, otherwise `NA` and `NaN` values are ignored.

Help

Help documentation pages present the following structure:

- **Description:** Function Description.
- **Usage:** Function syntax.
- **Arguments:** explanation of all the syntax function elements. All the options and their default values are detailed in this section.
- **Details:** Other explanations related to the usage of the function.
- **Value:** Description of input and output format.
- **References:** Literature.
- **See Also:** List of other related functions.
- **Examples:** Some simple examples for better understanding how to use the function.

Arithmetic Operators in R

Product "*" o "prod()"

```
> 4*2
[1] 8
> prod(4,2)
[1] 8
```

Ratio "/"

```
> 4/2
[1] 2
```

Arithmetic Operators in R

Power "^"

```
> 4^2
[1] 16
```

Square root "sqrt()"

```
> sqrt(4)
[1] 2
```

n^{th} power "^"

```
> 4^5
[1] 1024
```

n^{th} root "^"

```
> 1024^(1/5)
[1] 4
```

R Objects

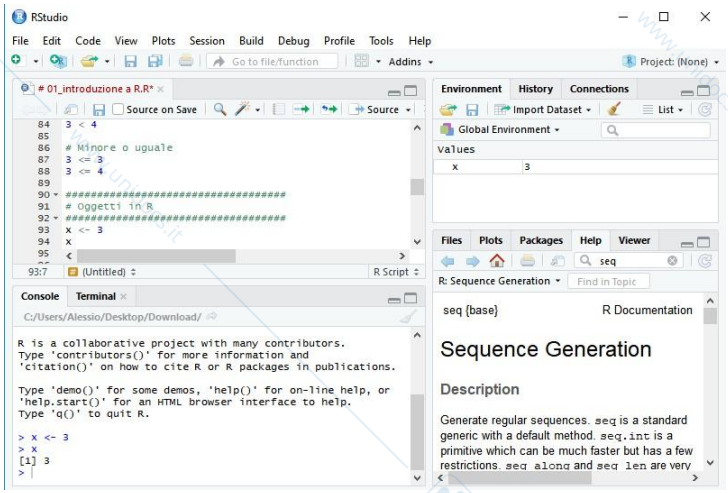
- Any element created and/or changed during a work session is an object.

- To create an object should be used "<-"

```
> x <- 3
> x
[1] 3
```

R Objects

Once defined, the object *x*, it appears in the "Environment"



R Objects

```
> (y <- 4); (z <- "a")
[1] 4
[1] "a"
```

To see the list of all the active objects it is possible to use the command line `ls()`

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

To remove one object from the list the command line to be used is `rm()`

```
> rm(ls="y")              # just one
> rm(list=c("x","z"))     # more than one
> rm(list=ls())           # all together
```

R - Objects

- `vector`: vector (NON algebric)
- `factor`: vector with categorical variables
- `matrix`: matrix
- `array`: multi – dimensional data object
- `list`: list
- `ts`: time series data object
- `data.frame`: list with a matrix structure (element for variable)

Vector

In **R**, vector means a list of elements of the same type. The main types of vectors in **R** are:

- **Integer numbers** (`integer`)
- **Real numbers** (`numeric`)
- **Characters** or **strings** (`character`)
- **Logical numbers** (`logical`)
- **Factors** (`factor`)
- **Complex numbers** (`complex`)

With the command line `class()` it is possible to know what kind of vector or object we are dealing with.

Integer number vector

A vector is generally created with `c()`, listing the elements of the vector in brackets, separated by a comma. For example:

$$c(x_1, x_2 \ldots)$$

```
> int <- c(1,5,7)
> int
[1] 1 5 7
> class(int)
[1] "numeric"
```

To get an integer number vector, each number should be followed by an `L`

```
> intl <- c(1L,5L,7L)
> intl
[1] 1 5 7
> class(intl)
[1] "integer"
```

Integer number vector

Equivalentely we can get integer number vectors:

```
>C(1L,2L,3L,4L)
[1] 1 2 3 4
> c(1:4)
[1] 1 2 3 4
> 1:4
[1] 1 2 3 4
```

Another useful function is `rep()`

```
>rep(3,6)
[1] 3 3 3 3 3 3
>rep(1:3,2)
[1] 1 2 3 1 2 3
>rep(1:3,each=2)
[1] 1 1 2 2 3 3
```

Integer number vector

The function `length()` provides the number of the vector elements

```
> int
[1]1 5 7
> length(int)
[1] 3
```

The function `str()` provides the kind of vector and shows an overview of the vector elements

```
> str(int)
num[1:3] 1 5 7
```

Real number vector

Like integer number vectors is possible to create a real number vector:

```
> real<- c(1,5.3,7.4)
> real
[1] 1 5.3 7.4
> class(real)
[1] "numeric"
```

Moreover we canget real number vectors using the function `seq()`

```
> seq(from=1,to=3,by=0.5)
[1] 1.0  1.5  2.0  2.5  3.0
>seq(1,3,0.5)
[1] 1.0  1.5  2.0  2.5 3.0
```

String or character vector

Also string or character vectors can be created using the same command line

```
> txt <- c("head", "tail")
> txt
[1] "head" "tail"
> class(txt)
[1] "character"
```

The function `nchar()` counts the number of character included in each string of the vector

```
> x <- c("Roma","Milano","Firenze", "", " ")
> nchar(x)
[1] 4 6 7 0 1
```

String or character vector

The function `paste()` is used to construct complex string

```
> paste("VAR",1:4,sep="")
[1] "VAR1" "VAR2" "VAR3" "VAR4"
> paste("VAR",1:4,sep=".")
[1] "VAR.1" "VAR.2" "VAR.3" "VAR.4"
```

Other useful functions to work with strings:

- `substring()` selects a portion of a string using position
- `tolower()` converts to lowercase
- `toupper()` converts to uppercase
- `match()` searches for a string in a vector
- `grep()` searches for a string in a vector (partial match are allowed)
- `pmatch()` searches for a string (partial matches are allowed from the beginning)
- `sub()` replaces a part of a string

String or character vector

```
> substring("abcdef",2,4)
[1] "bcd"
> tolower("ABCDEF")
[1] "abcdef"
> toupper("abcdef")
[1] "ABCDEF"
> match("Milano",x)
[1] 2
> grep("a",x)
[1] 1 2
> pmatch("Ro",x)
[1] 1
> sub("a","**",x)
[1] "Rom**" "Mil**no" "Firenze" "" " "
```

Comparison operators in **R**

Equal "=="

```
> 3 == 3
[1] TRUE
> 3 == 4
[1] FALSE
```

Different "!="

```
> 3 != 3
[1] FALSE
> 3 != 4
[1] TRUE
```

Comparison operators in R

Greater than "> "

```
> 3 > 3
[1] FALSE
> 4 > 3
[1] TRUE
```

Greater or equal to ">="

```
> 3 >= 3
[1] TRUE
> 3 >= 4
[1] FALSE
```

Comparison operators in **R**

Lower than "< "

```
>  3 < 3
[1] FALSE
>  3 < 4
[1] TRUE
```

Lower than or equal to "<="

```
>  3 <= 3
[1] TRUE
>  3 <= 4
[1] TRUE
```

Logical Vector

Through the logical operators it is possible to create vectors of logical elements

```
> x <- 1:5
> y <- x < 3
> y
[1] TRUE TRUE FALSE FALSE FALSE
> class(y)
[1] "logical
```

Factor Vectors

Used for dealing with categorical data

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

`levels()` and `nlevels()` functions provide, respectively the type and the number of type included in the vector

```
> levels(sex)
[1] "factor"
> nlevels(sex)
[1] 2
```

Factor Vectors

A factor vector can be got as result of the `cut()` function.

This function is used to get categorical variables from numerical ones.

```
> x <- 1:8
> y <- cut(x,c(1,4,8),include.lowest=T,labels=c("1-4","5-8"))
> y
[1] 1-4 1-4 1-4 1-4 5-8 5-8 5-8 5-8
Levels: 1-4 5-8
```

How to select vector elements

It is possible to select vector elements:

- Looking for their position

```
Vector.name[position]
```

```
> x[4]
[1] 4
```

- Using a criterion to select the elements

```
Vector.name[which(logical operator)]
```

```
> x[which(x>4)]
[1] 5 6 7 8
```

Matrix

A matrix is a set of elements oredered by rows and columns. The command line to create a matrix in **R** is **matrix**

```
> A<- matrix(1:6,ncol=2,nrow=3)
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> class(A)
[1] "matrix"
```

Matrix

It is possible to get some information about a matrix:

- `dim()` dimension (number of rows and columns)

```
> dim(A)
[1] 3 2
```

- `nrow()` number of rows

```
> nrow(A)
[1] 3
```

- `ncol()` number of columns

```
> ncol(A)
[1] 2
```

- `legnth()` number of matrix elements

```
> length(A)
[1] 6
```

How to select matrix elements

Matrix elements can be selected:

- In reference to their own position:

  matrix.name[row.number,column.number]

```
> A[3,2]
[1] 6
```

With matrix objects is possible to compute some matrix algebra, like i.e. to transpose a matrix (t()) or getting the vector product (%*%)

```
> B<- matrix(1:6,ncol=2,nrow=3,byrow=TRUE)
> t(A)%*% B
     [,1] [,2]
[1,]   22   28
[2,]   49   64
```

Lists

Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using list() function.

```
> list <- list(integers=int,reals=real,factor=sex,matrix=A)
> list
$integers
[1] 1  5  7
$reals
[1] 1.0  5.3  7.4
$factor
[1] M F M F F
Levels: M F
$matrix
     [,1]    [,2]
[1,]    1       4
[2,]    2       5
[3,]    3       6
```

List

The list elements can be given names and they can be accessed using these names

```
list(name1=object1, name2=object2,...)
```

It is possible to recall an object:

- Using "$"

```
list.name$object
```

```
>list$factor
[1] M FM FF
Levels: M F
```

- Alternatively using:

```
list.name[object.position]
```

```
>list[3]
[1] M FM FF
Levels: M F
```

List structure

The function `str()` shows the list structure

```
> str(list)
$integers  :  num  [1:3]  1  5  7
$reals     :  num  [1:3]  1  5.3  7.4
$factor    :  Factor w/ 2  levels "M","F": 1 2 1 2 2
$matrix    :  int     [1:3,1:2]  1  2  3  4  5  6
```

To remove an object froma list:

`list.name[-object.position]`

```
> listar <- list[-3]
$integers  :  num  [1:3]  1  5  7
$reals     :  num  [1:3]  1  5.3  7.4
$matrix    :  int     [1:3,1:2]  1  2  3  4  5  6
```

Add more objects in a list

To add one or more objects:

```
> list$character <-txt
> str(list)
$integers    :  num   [1:3]  1   5   7
$reals       :  num   [1:3]  1   5.3   7.4
$factor      :  Factor w/ 2  levels "M","F": 1 2 1 2 2
$matrix      :  int     [1:3,1:2]  1  2  3  4  5  6
$character   :  chr     [1:2] "head" "tail"
```

dataframe

A **dataframe** is a list of vectors of equal length. It is represented as a table whose:

- **Rows** represent the observations
- **Columns** represent the variables

The top line of the table, called the **header**, contains the column names. Each horizontal line afterward denotes a **data row**, which begins with the name of the row, followed by the data. Each element of a row defines a **cell**.

In other word a **dataframe** is like a matrix which includes, differently from the previous one, even non-numeric objects.

Infact, objects in a dataframe are allowed to be simultaneously numeric, integer, logical factor. Character objects are automatically converted into a factors

dataframe

To create a dataframe it is possible to use the function data.frame()
Example:

- Id code
- Sex
- Age

```
> id  <- c("001","002","003","004","005")
> sex <- factor(c("M","F","M","F","F"), levels=c("M","F"))
> age <- c(33,25,61,21,47)
> data <- data.frame(id,sex,age)
> data
    id   sex  age
1  001    M   33
2  002    F   25
3  003    M   61
4  004    F   21
5  005    F   47
> class(data)
[1] "data.frame"
```

dataframe

It is possible to get some information about a dataframe:

- `dim()` dimension (number of rows and columns)

```
> dim(data)
[1] 5 3
```

- `nrow()` number of rows (=*units,observations*)

```
> nrow(data)
[1] 5
```

- `ncol()` number of columns (=*variables)*

```
> ncol(data)
[1] 3
```

- `legnth()` number of dataframe elements

```
> length(data)
[1] 3
```

How to select elements from a dataframe

- Column(=*variable*)

```
> data$age
[1] 33 25 61 21 47
> data[,3]
[1] 33 25 61 21 47
> data[, "age"]
[1] 33 25 61 21 47
```

- Row(=*unit*)

```
> data[3,]
3 003 M 61
> class(data[3,])
[1] "data.frame"
```

In a dataframe rows are classified as "data.frame"

How to select elements from a dataframe

It is possible to select some units in relation to their features:

```
> data[which(data$sex=="F"),]
   id sex age
2 002   F  25
4 004   F  21
5 005   F  47
```

In this case we select from dataframe just female (`sex=="F"`).

Sort units in a dataframe

To sort dataframe units it is possible to use the command line:

```
data.frame.name[order(variabile.name),]
```

```
> data[order(age),]
   id   sex  age
4 004    F    21
2 002    F    25
1 001    M    33
5 005    F    47
3 003    M    61
```

With the previous instruction, the dataframe is sorted in increasing order of the variable. To provide a decreasing order to the dataframe, is sufficient to write

```
data.frame.name[order(-variable.name),]
```

Adding a column to dataframe

To add a new column to a dataframe it is possible to use the same function used to add an object to a list.

```
> ql <- as.logical(c(1,0,1,1,1))
> ql
[1] TRUE FALSE TRUE TRUE TRUE
> data$ql <- ql
> data
  id sex age    ql
1 001   M  33  TRUE
2 002   F  25 FALSE
3 003   M  61  TRUE
4 004   F  21  TRUE
5 005   F  47  TRUE
```

Adding a dataframe line

To combine several vectors, matrices and/or dataframes by rows, could be used the function `rbind()`.
Notice that each row is classified as a data.frame in R

```
> oss6 <- data.frame(id="006",sex="M",age=39,q1=FALSE)
> data <- rbind(data,oss6)
> Data
  id sex age    q1
1 001   M  33  TRUE
2 002   F  25 FALSE
3 003   M  61  TRUE
4 004   F  21  TRUE
5 005   F  47  TRUE
6 006   M  39 FALSE
```

How to import a dataset in R

Importing data into **R** is fairly simple. Depending on the file extention you can import:

- `.csv` (comma separated values)
  `read.csv()` `read.csv2()`
- `.txt` (Tab delimited file)
  `read.delim()`
- `.xls` o `.xlsx` (Microsoft Excel)
  `read_excel()` using the package *`readxl`*
  `read.xlsx()` using the package *`openxlxs`*
- `.dta` (Stata file)
  `read.dta()`

Import a dataset in **R**

Let's write on our browser the following URL

http://goo.gl/HKnl74

We get immediately open a dataset including **500 household** who attended an amusement park and **8 variables**:

1. *Weekend,* the park has been visited during the weekend?
2. *child*, number of children coming in the park
3. *Distance,* distance traveled to reach the park
4. *Rides*, rating (0 - 100)
5. *Games*, rating (0 - 100)
6. *Wait*, rating (0 - 100)
7. *Clean*, rating (0 - 100)
8. *Overall*, rating (0 - 100)

Save the file in the working directory and open it in **R Studio**.

Dataset park

```
> park
# A tibble: 500 x 8
   weekend num.child  distance rides games  wait clean overall
     <chr>     <dbl>     <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
 1     yes         0 114.64826    87    73    60    89      47
 2     yes         2  27.01410    87    78    76    87      65
 3      no         1  63.30098    85    80    70    88      61
 4     yes         0  25.90993    88    72    66    89      37
 5      no         4  54.71831    84    87    74    87      68
 6      no         5  22.67934    81    79    48    79      27
 7     yes         1  26.64956    77    73    58    85      40
 8      no         0  11.30019    82    70    70    83      30
 9      no         0  14.11403    90    88    79    95      58
10     yes         3  21.35994    88    86    55    88      36
# ... with 490 more rows
```

In this dataset there are different kind of variables:

- Qualitative variables
- Quantiative discrete variables
- Quantative continous variables

Qualitative Variables

Qualitative variables, aslo called categorical variables, are **not numerical** variables. They describe data that fit into categories.

Dummy variables: take only the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome.

Polytomic variables: consider qualitative variables with more than 2 modalities.
With this kind of variables it is possible to compare:

- = and ≠ with **disconnected quantities** (i.e. hair color, profession…)
- =, ≠, > or < with ordered quantitative variables (i.e. evaluation "insufficient", "sufficient", "discreet", "good", "distinct", "excellent")

Frequency Distribution

To describe a qualitative variable it is possible to look at its frequency distribution. The function `table()` applied to a single variable produces the distribution of absolute frequencies of the variable itself .

```
> tabw <- table(park$weekend)
 no  yes
259  241
```

To add the total, it is sufficient to apply the `addmargins()` function to the output of the `table()` function which has been appropriately saved as an object

```
> tabw <- table(park$weekend)
> addmargins(tabw)
 no  yes  Sum
259  241  500
```

Distribution of relative frequencies or percentage

To get the **relative frequency distribution** it is possible to use the function prop.table()to be applied on the output of the function table()

```
> rel <- prop.table(tabw)
> addmargins(rel)
   no       yes     Sum
0.518     0.482     1.000
```

To get the **percentage distribution**, it is sufficient to multiply by 100 the results coming from the relative frequencies table

```
> perc <- rel*100
> addmargins(perc)
   no       yes       Sum
 51.8     48.2     100.0
```

Quantitative discrete variables

A **discrete quantitative variable** is a variable whose values are the result of a count.

Both **absolute** and **relative frequency distribution** are obtained as seen for categorical variables

```
> tabc <- table(park$num_child)
> tabc
  0   1   2   3   4   5
151  66 143  68  47  25
 prop.table(tabc)
    0     1     2     3     4     5
0.302 0.132 0.286 0.136 0.094 0.050
```

Double entry Table

A **double-entry table** is a table in which the modes of two variables cross.
What we get is the **joint distribution of absolute frequencies**

```
> tabcw <- table(park$num_child,park$weekend)
   no yes
0  74  77
1  37  29
2  77  66
3  34  34
4  25  22
5  12  13
```

Double entry Table

To get the colum/row total values it is sufficient to use the function `addmargins()`

```
> tabcw <- table(park$num_child,park$weekend)
     no  yes  Sum
 0   74   77  151
 1   37   29   66
 2   77   66  143
 3   34   34   68
 4   25   22   47
 5   12   13   25
Sum 259  241  500
```

Each **row** represents the family distribution by day of visit, **given** the number of children
Each **column** represents the family distribution by number of children, **given** the day of visit

Double entry Table

Frequency distribution in a double – entry table

```
> rel2 <- tabcw/dim(park)[1]
> addmargins(rel2)

          no   yes   Sum
  0   0.148 0.154 0.302
  1   0.074 0.058 0.132
  2   0.154 0.132 0.286
  3   0.068 0.068 0.136
  4   0.050 0.044 0.094
  5   0.024 0.026 0.050
  Sum 0.518 0.482 1.000
```

The sum of all relative frequencies in a double – entry table is equal to 1.

Conditional Distributions

Distribution of household **relative frequencies** by day of visit, given the number of children on the tour

```
> prop.table(tabcw,margin=1)

          no       yes
0 0.4900662 0.5099338
1 0.5606061 0.4393939
2 0.5384615 0.4615385
3 0.5000000 0.5000000
4 0.5319149 0.4680851
5 0.4800000 0.5200000
```

Each row represents a frequency distribution. For example, the first row of the table shows the distribution of families by day of visit, conditionally on families with 0 children.

Conditional Distributions

Distribution of household **relative frequencies** by number of children in the trip, given the day of visit.

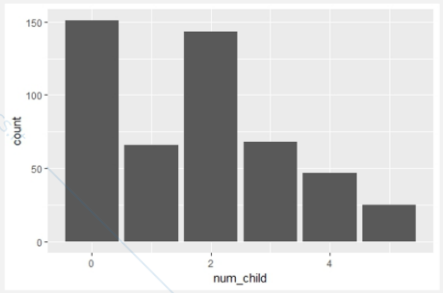
```
> prop.table(tabcw,margin=2)

           no        yes
0 0.28571429 0.31950207
1 0.14285714 0.12033195
2 0.29729730 0.27385892
3 0.13127413 0.14107884
4 0.09652510 0.09128631
5 0.04633205 0.05394191
```

Each column represents a frequency distribution. For example, the first column of the table shows the distribution of families by children in the tour, conditionally on visit day.

Barplot

To graph the frequency distributions we use the bar chart (geom_bar())

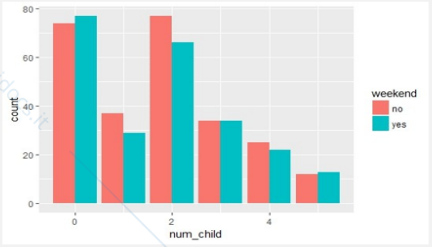
```
> ggplot(data=park, aes(x=num_child)) + geom_bar()
```



Barplot

It is possible to **distinguish the bars** according to whether the visit took place over the weekend or not:

```
> ggplot(data=park, aes(x=num_child, fill=weekend)) +
    geom_bar(position=position_dodge())
```

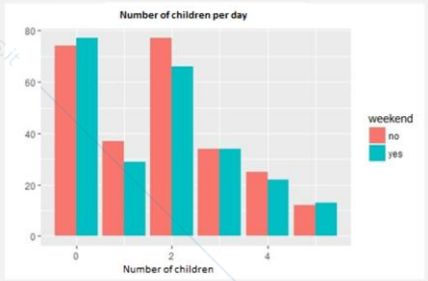


Each **pair of bars** represents the distribution of families by day of visit **given** the number of children.

Barplot

It is possible to add a title and changing the name of the axes

```
>ggplot(data=park, aes(x=num_child, fill=weekend)) +
    geom_bar(position=position_dodge()) +
    xlab("Number of children") +ylab("") +
    ggtitle("Number of children per day")
```



Continuous Quantitative Variables

A continuous variable is a variable where the scale is continuous and not made up of discrete steps.

The main features of continuous quantitative variables could be applied also to discrete quantitative variables with many modalities.
Park dataset includes the following variables:

- *distance* – continuous variable
- *rides* – quantitaive discrete variable with more than 100 modalities
- *games* – quantitaive discrete variable with more than 100 modalities
- *wait* – quantitaive discrete variable with more than 100 modalities
- *clean* – quantitaive discrete variable with more than 100 modalities
- *overall* – quantitaive discrete variable with more than 100 modalities

Statistical Position Indices

- minimum

```
> min(park$distance)
[1] 0.53
```

- maximum

```
> max(park$distance)
[1] 239.19
```

- mean

```
> mean(park$distance)
[1] 31.04748
```

- median

```
> median(park$distance)
[1] 19.02
```

Statistical Position Indices

- percentiles

```
> quantile(park$distance, probs=0.4) #40th percentile
   40 %
14.752
```

```
> quantile(park$distance, probs=c(0.25, 0.5, 0.75)
    25%      50%      75%
10.320   19.020   39.582
```

```
> quantile(park$distance, probs=0:10/10)
     0%     10%     20%     30%     40%     50%     60%
  0.530   5.716   8.430  11.782  14.752  19.020  26.558
    70%     80%     90%    100%
 35.409  47.020  68.020 239.190
```

Variability Indices

- Variance

```
> var(park$distance)
[1] 1098.62
```

- Standard Deviation

```
> sd(park$distance)
[1] 33.14543
```

- Interquartile Range

```
> IQR(park$distance)
[1] 29.2625
```

- Median Absolute Deviation

```
> mad(park$distance)
[1] 17.25746
```

Summary

Great part of the information provided by the position index is provided by the function summary()

```
> summary(park$distance)
   Min. 1st Qu.  Mean   Median 3rd Qu.   Max.
0.5267 10.3181 19.0191 31.0475 39.5821 239.1921
```

Asimmetry

Given the value of mean and median it is possible to have some useful information:

- Median = Mean → simmetry
- Median < Mean → positive asymmetry
- Median > Mean → negative asymmetry

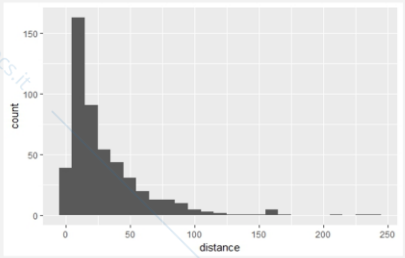
More robust results are provided considering:

- Position index
  - median
- Variability index
  - Interquartile range
  - MAD

Histogram

The frequency distribution by classes of a continuous variable can be represented by a histogram (geom_histogram)

```
> ggplot(data=park, aes(x=distance)) + geom_histogram(binwidth=10)
```

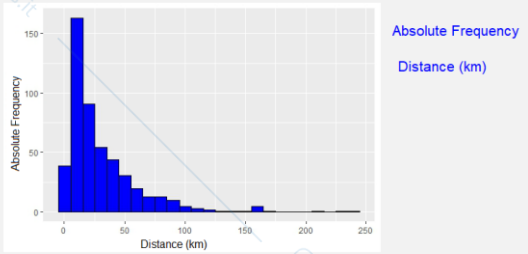


Histogram

There are two functions to customize the colors of the charts :

- Color: for changing the color of the outlines
- Fill: for changing the color of the bars

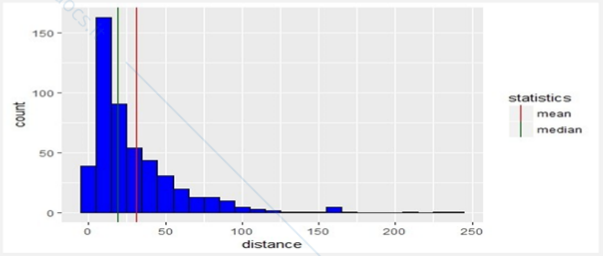
```
> ggplot(data=park, aes(x=distance)) +
    geom_histogram(binwidth=10, fill="blue", color="black")+
    xlab("Distance (km)") + ylab("Absolute Frequency")
```



Histogram

Let's add the mean and the median in the histogram

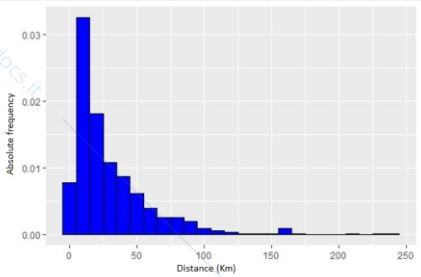
```
> ggplot(data=park, aes(x=distance)) +
  geom_histogram(binwidth=10,fill="blue", color="black")+
  geom_vline(aes(xintercept=median(park$distance),
  color="median")) + geom_vline(aes(xintercept=mean(park$distance)
  color="mean")) + scale_color_manual(name="statistics",
  values=c(median="blue",mean="red"))
```



Histogram

It is possible to represent the frequency distribution with a plot

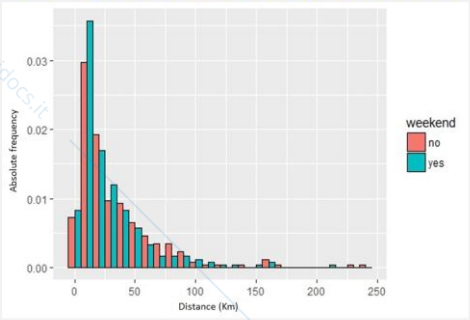
```
> ggplot(data=park, aes(x=distance, y=..density..)) +
  geom_histogram(binwidth=10, fill="blue", color="black")+
  xlab("Distance (km)") + ylab("Absolute Frequency") )
```



Histogram

Comparing two frequency distributions:

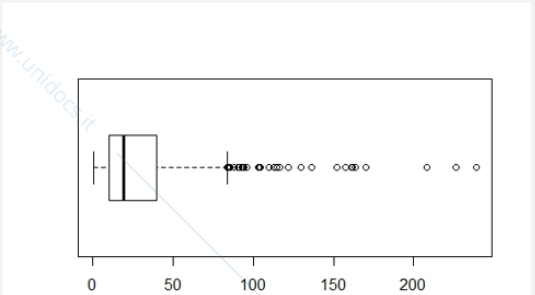
```
> ggplot(data=park, aes(x=distance, y=..density..)) +
  geom_histogram(aes(fill=weekend),binwidth=10,color="black")
  +xlab("Distance (km)") + ylab("Absolute Frequency")
```



Boxplot

The **boxplot** contains many synthetic aspects of a frequency distribution

```
> boxplot(park$distance, horizontal=TRUE)
```



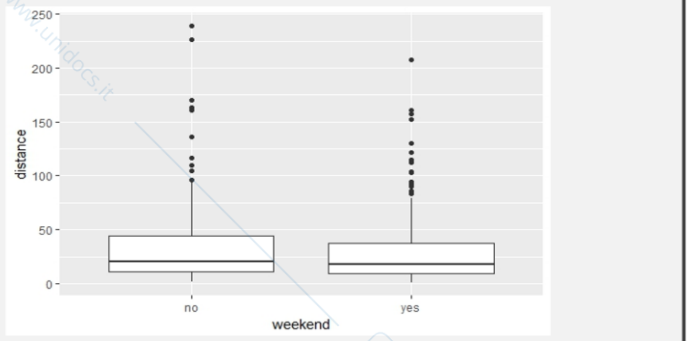
Boxplot

- The **box** of the plot is a rectangle which encloses the middle half of the sample, with an end at each quartile (Q1 and Q3).
- The length of the box is thus the **interquartile range** of the sample.
- A line is drawn across the box at the **sample median**.
- **Whiskers** sprout from the two ends of the box until they reach the values:
  - ✓ $Q_1 - 1.5\,IQR$
  - ✓ $Q_3 + 1.5\,IQR$
- **Dots** are values outside the whiskers that represent possible outliers

Boxplot

Let us compare the distribution of the travel length based on the day of visit (`geom_boxplot`)

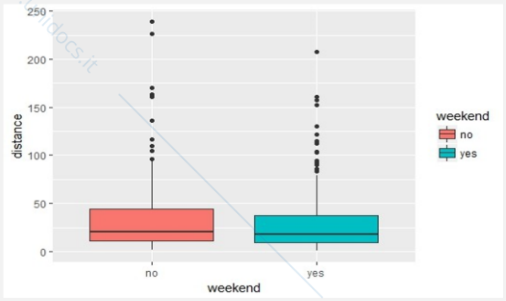
```
> ggplot(data=park, aes(x=distance, y=..density..)) +
  geom_plot()
```



Boxplot

Let us compare the distribution of the travel length based on the day of visit (geom_boxplot)

```
> ggplot(data=park, aes(x=distance, y=..density..,
  fill=weekend)) +geom_plot()
```



Multivariate Analysis

Each individual variable can be analyzed separately or:

- Together with other variables **simultaneously**
- It is possible to study the **relationship** between two or more variables

Simultaneous analysis of several variables

Indicators of a **quantitative variabile** distribution could be compute for several variables simutaneously

```
> park_q <- select(park,distance:overall)
> head(park_q)
   distance rides games wait clean overall
1 114.64826    87    73   60    89      47
2  27.01410    87    78   76    87      65
3  63.30098    85    80   70    88      61
4  25.90993    88    72   66    89      37
5  54.71831    84    87   74    87      68
6  22.67934    81    79   48    79      27
```

Simultaneous analysis of several variables

To compute the **mean** there is the function `colMeans`

```
> colMeans(park_q)
distance    rides    games     wait    clean  overall
31.04748 85.84600 78.66600 69.89600 87.89800 51.25800
```

To compute the other index it is helpful the function `summarise_all()` included in the package **dplyr**

```
>summarise_all(park_q,funs(median)
    distance  rides  games  wait  clean  overall
[1]  19.02     86     78     70     88     50
```

In `funs()` we provide the name of all the functions to be applied to all the variables of the dataset. This function works only with quantitative variables.

%>% (*pipe – operator*)

To compute an index for quantitative variables we had to save a new dataframe.
To avoid this double save it is possible to use the % > % pipe (Ctrl/Cmd+Shift+M) whose sintax is:

argument % > % function. name

```
> mean(1:10)
[1] 5.5
> 1:10 % > % mean
[1] 5.5
```

%>% (pipe – operator)

Let's compute the median of the quantitative variables only:

```
> Park % > %
> select(distance:overall) % > %
> summarize_all(funs(median))
  distance     rides     games      wait     clean   overall
  31.04748  85.84600  78.66600  69.89600  87.89800  51.25800
```

Segment Analysis

A **segment** is a sub-set of a sample (or population) whose units are **homegeneous** with respect to one or more variables (**segmentation variables**)

The function `group_by()`, in **dyplyr**, allows the choice of the segmentation variable (i.e. **day of visit**)

```
> park %>%
+ group_by(weekend) %>%
+ summarize(sat=mean(overall))
# A tibble: 2 x 2
  weekend   sat
  <fct>   <dbl>
1 no       52.1
2 yes      50.4
```

Segment Analysis

For number of children

```
> park %>%
+   group_by(num.child) %>%
+   summarize(sat=mean(overall),sd_sat=sd(overall),cv_sat=sd_sat/sat)
# A tibble: 6 x 4
  num.child   sat sd_sat cv_sat
      <int> <dbl>  <dbl>  <dbl>
1         0  40.7   11.0  0.270
2         1  56.8   13.9  0.244
3         2  56.0   16.5  0.294
4         3  55.8   15.5  0.278
5         4  52.6   15.6  0.297
6         5  58.1   13.4  0.230
```

Segment Analysis

It is possible to apply the same function to several variables

```
> park %>%
    group_by(weekend) %>%
    summarize_all(funs(mean))
# A tibble: 2 x 8
  weekend num.child distance rides games  wait clean overall
  <fct>       <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>
1 no           1.75     33.3  86.1  78.9  69.9  88.1    52.1
2 yes          1.73     28.6  85.6  78.4  69.9  87.7    50.4
```

Export data in R

To save the result tables provided by the analysis

```
> sat_park <- park %>%
> group_by(num.child) %>%
> summarize(sat=mean(overall),sd_sat=sd(overall),cv_sat=sd_sat/sat)
```

To do this we need to give a name to our object in this case

sat_park

Export data with **R**

It is possible to save data in several formats:

- `.csv` (comma separated values)

```
write.csv(dataframe, "file.name.csv")
write.csv2(dataframe,"file.name.csv")
```

- `.txt` (Tab delimited file)

```
write.table(dataframe, "file.name.txt",
sep="\t",colnames=FALSE)
```

- `.xlsx` (Microsoft Excel)

```
write.xlsx() package openxlxs
write.xlsx(dataframe, "file.name.xlsx")
```

- `.dta` (Stata file)

```
write_dta() package haven
write_dta(dataframe, "file.name.dta", version=14)
Package haven
```

Customers

Let's write on the browser the following URL

`goo.gl/PmPkaG`

Immediately we get a dataset of 1000 customers of a multi-channel seller.

For each custmer, **11 variables** have been observed:

- `age`, age of the customer
- `credit_score`, creditworthiness
- `email`, does the customer have an email address registered on the site?
- `distance_to_store`, distance to store
- `online_visits`, `online_trans`, `online_spend`, number of visits, purchases and total expenditure on the retailer's website
- `store_trans`, `store_spend`, number purchases and total expenditure on the retailer's store
- `sat_service`, `sat_selection`, level of satisfaction with the service and product selection, on a scale from 1 (totally unsatisfied) to 5 (totally satisfied)

Customers

Save and load a ***.txt*** file

```
> customers<- as.tibble(read.delim("E:/LUISS/Lezioni/customers.txt"))
> customers
# A tibble: 1,000 x 12
   cust.id   age credit.score email distance.to.store online.visits
     <int> <dbl>        <dbl> <fct>             <dbl>         <int>
 1       1  22.9          631 yes                2.58            20
 2       2  28.0          749 yes               48.2            121
 3       3  35.9          733 yes                1.29            39
 4       4  30.5          830 yes                5.25             1
 5       5  38.7          734 no                25.0             35
 6       6  42.4          686 yes               18.5              1
 7       7  28.3          725 yes                9.34             1
 8       8  36.5          749 yes                6.92            48
 9       9  30.4          690 no                12.8              0
10      10  26.6          787 no                 6.62            14
# ... with 990 more rows, and 6 more variables: online.trans <int>,
#   online.spend <dbl>, store.trans <int>, store.spend <dbl>,
#   sat.service <int>, sat.selection <int>
```

Data-set Analysis

```
> summary(customers)
    cust.id           age         credit.score   email     distance.to.store
 Min.   :   1.0   Min.   :19.34   Min.   :543.0   no :186   Min.   :  0.2136
 1st Qu.: 250.8   1st Qu.:31.43   1st Qu.:691.7   yes:814   1st Qu.:  3.3383
 Median : 500.5   Median :35.10   Median :725.5             Median :  7.1317
 Mean   : 500.5   Mean   :34.92   Mean   :725.5             Mean   : 14.6553
 3rd Qu.: 750.2   3rd Qu.:38.20   3rd Qu.:757.2             3rd Qu.: 16.6589
 Max.   :1000.0   Max.   :51.86   Max.   :880.8             Max.   :267.0864

 online.visits     online.trans      online.spend      store.trans
 Min.   :  0.00   Min.   :  0.000   Min.   :   0.00   Min.   : 0.000
 1st Qu.:  0.00   1st Qu.:  0.000   1st Qu.:   0.00   1st Qu.: 0.000
 Median :  6.00   Median :  2.000   Median :  37.03   Median : 1.000
 Mean   : 28.29   Mean   :  8.385   Mean   : 170.32   Mean   : 1.323
 3rd Qu.: 31.00   3rd Qu.:  9.000   3rd Qu.: 177.89   3rd Qu.: 2.000
 Max.   :606.00   Max.   :169.000   Max.   :3593.03   Max.   :12.000

  store.spend      sat.service   sat.selection
 Min.   :  0.00   Min.   :1.00   Min.   :1.000
 1st Qu.:  0.00   1st Qu.:3.00   1st Qu.:2.000
 Median : 30.05   Median :3.00   Median :2.000
 Mean   : 47.58   Mean   :3.07   Mean   :2.401
 3rd Qu.: 66.49   3rd Qu.:4.00   3rd Qu.:3.000
 Max.   :705.66   Max.   :5.00   Max.   :5.000
                  NA's   :341    NA's   :341
```

Scatterplot

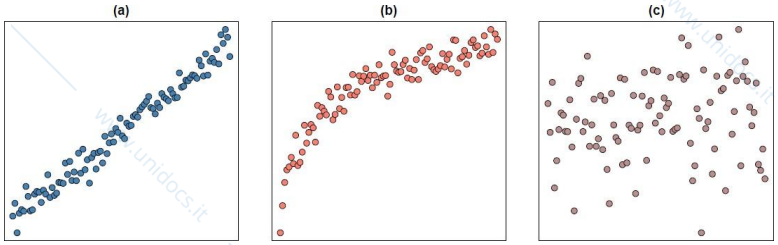
A **scatterplot** is a graphic tool used to display the relationship between two quantitative variables.

- Each dot on the scatterplot represents one observation from the dataset whose cohordinates (x_i, y_i) define the observed values for the variables X and Y.

A scatterplot provides several useful information about:

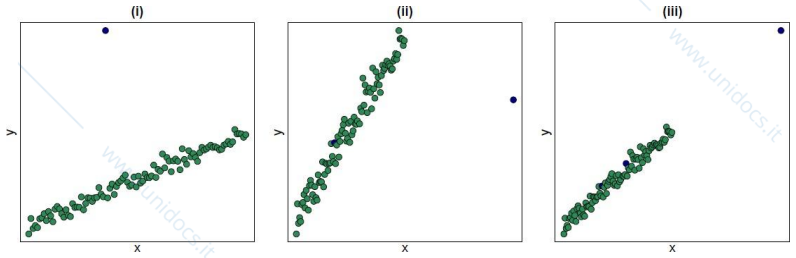
- The presence of a possible **relation** (linear, logarithmic, quadratic) between the two variables
- The presence of **outliers** with respect to one of the variables considered, to both of them or with respect to their relation.

Relation between two variables



(a) Linear (increasing)
(b) Logarithmic (increasing)
(c) No relation

Outliers



(i) Outlier with respect to the relation between x and y and with respect to y, but not with respect to x.

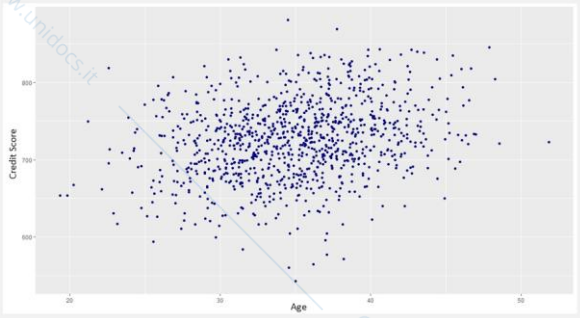
(ii) Outlier with respect to the relation between x and y and with respect to x, but not with respect to y.

(iii) Outlier with respect to x and y but not with respect to their relation.

Scatterplot

Let's create a scatterplot between age and credit.score using the function (`geom_boxplot`)

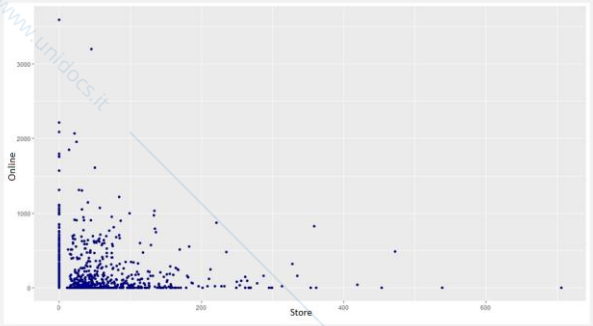
```
> ggplot(customers, (x=age, y=credit.score)) +
  geom_point(color="darkblue")+ xlab("Age")+ylab("Credit score")
```



Scatterplot

Or between store.spend and online.spend

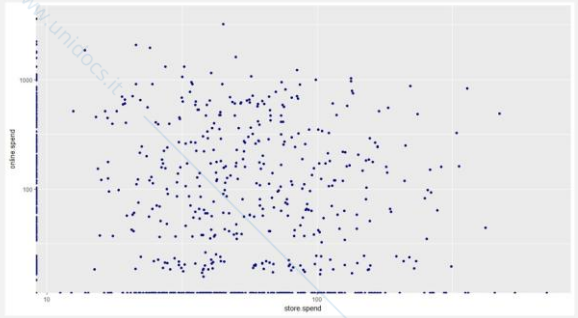
```
> ggplot(customers, (x=store.spend, y=online.spend)) +
  geom_point(color="darkblue") + xlab("Store")+ylab("Online")
```



Scatterplot

Let's change in logarithmic scale both variables

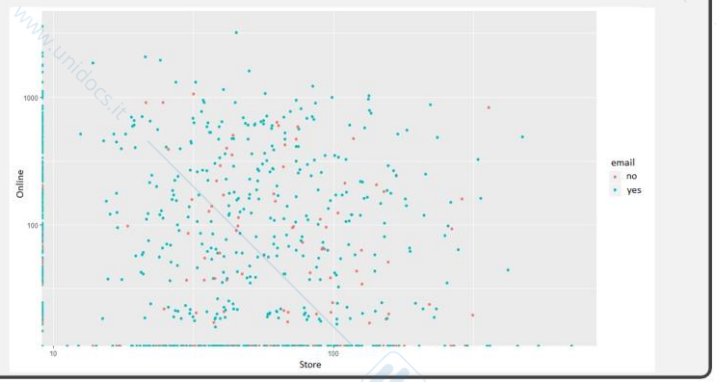
```
> ggplot(customers, (x=store.spend, y=online.spend)) +
  geom_point(color="darkblue")+scale_x_log10()+scale_y_log10()
```



Scatterplot

Let's distinguish customers based on whether their email is registered on the site or not

```
> ggplot(customers, (x=store.spend, y=online.spend)) +
  geom_point(aes(color=email)) + scale_x_log10()+scale_y_log10() +
  xlab("Store")+ylab("Online")
```



Scatterplot

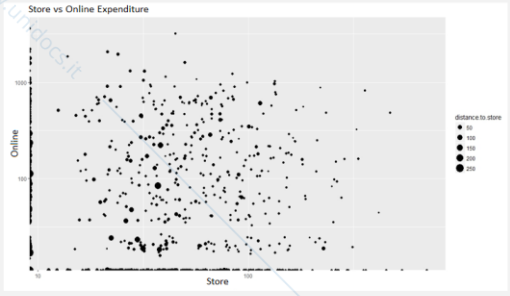
Let's insert the labels in the graph, this time as a list (labs)

```
> ggplot(customers, (x=store.spend, y=online.spend)) +
  geom_point(aes(color=email))+scale_x_log10()+scale_y_log10()+
  xlab("Store")+ylab("Online")
```

bubbleplot

The *bubbleplot* is a scatterplot whose dot dimension is proportional to a third variable (`distance_to_store`).

```
> ggplot(customers, (x=store.spend, y=online.spend)) +
  geom_point(aes(size=distance_to_store)) + scale_x_log10() +
  scale_y_log10()+ labs(list(title="Store vs Online (in log)
  expenditure",x="Store",y="Online", color="Email\nregistered"))
```



Scatterplot Matrix

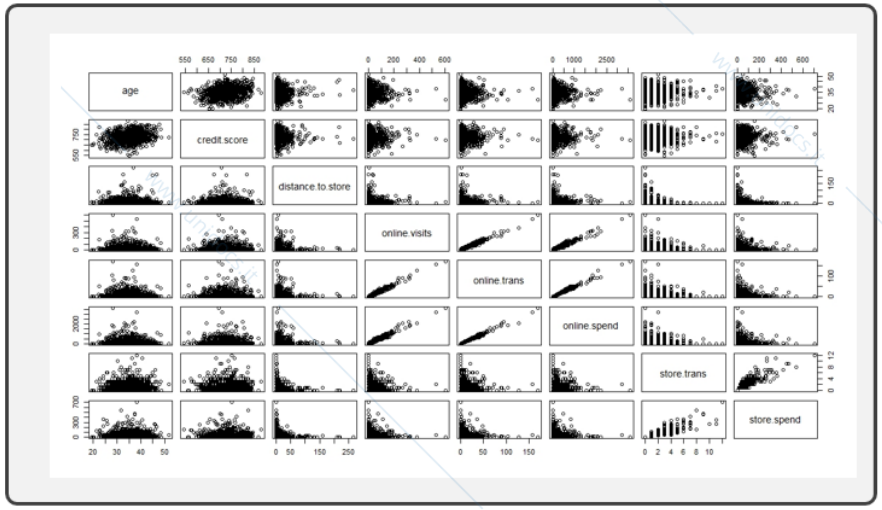
The Scatterplot matrix contains the scattering diagrams of all the quantitative variables of a data matrix

```
> customers_q <- select(customers, age,
  credit.score,distance.to.store:store.spend)
> pairs(customers_q)
```

Or

```
> customers  %  > %
> select(customers,age,credit.score,distance.to.store:store.spend)% > %
> pairs()
```

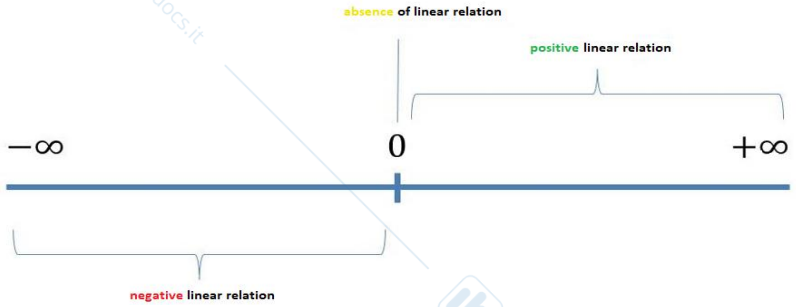
Scatterplot Matrix



Covariance

Scatterplot provides a graphical information about the realtionship between two variables. An index that summarize this realation is the **covariance**, $Cov(X, Y)$

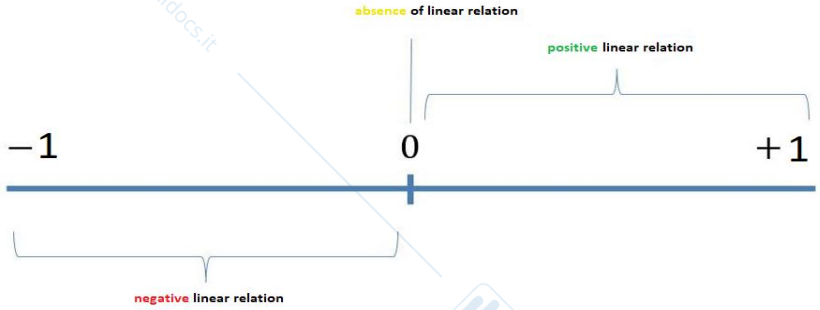
```
> cov(costumers$age,costumers$credit_score)
[1] 63.23443
```



Correlation

As **covariance** depends on measurement unit and order of magnitude of the two variables, it is difficult to assess the goodness of the linear relation using this coefficient. For this reason it is better to refer to **correlation coefficient**, r(X, Y) which does not depends on these two factors.

```
> cor(costumers$age,costumers$credit_score)
[1] 0.2545045
```



Correlation coefficient (r) interpretation

If the variables are (approximately) **normally distributed,** it is possible to consider the Cohen's rule of thumb for which:

- if $|r| \approx 0.1$, the linear relation is absent or weak
- if $|r| \approx 0.3$, the linear relation is moderate
- if $|r| \approx 0.5$, the linear relation is high
- if $|r| = 1$, there is a perfect linear relation

If the two variables are not normally distributed it could be useful to apply some transformation (like logarithm) before computing the coefficient.

Correlation Matrix

The Correlation Matrix is a square matrix in which

- Number of rows = number of columns =number of quantitative variables
- The generic element r_{ij} represents the correlation between the variable X_i and X_j

$$R = \begin{pmatrix} 1 & r_{12} & \dots r_{1j} & \dots & r_{1p} \\ r_{21} & 1 & \dots r_{2j} & \dots & r_{2p} \\ \dots & \dots & \dots \dots & \dots & \dots \\ r_{i1} & r_{i2} & \dots r_{ij} & \dots & r_{ip} \\ \dots & \dots & \dots \dots & \dots & \dots \\ r_{p1} & r_{p2} & \dots r_{pj} & \dots & 1 \end{pmatrix}$$

Where $r_{ij} = r_{ji}$
and $r_{ii} = r_{jj} = 1$

Correlation Matrix

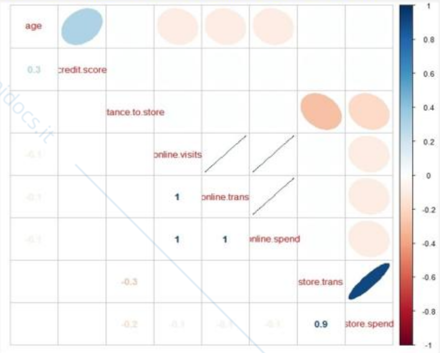
It is possible to get the correlation matrix using the function `cor()` to be applied to a set of **quantitative variables**

```
> cor_cust <- customers %>%
+   select(age,credit.score,distance.to.store:store.spend) %>%
+   cor() %>% round(1)
> cor_cust
                  age credit.score distance.to.store online.visits
age               1.0          0.3              0.0          -0.1
credit.score      0.3          1.0              0.0           0.0
distance.to.store 0.0          0.0              1.0           0.0
online.visits    -0.1          0.0              0.0           1.0
online.trans     -0.1          0.0              0.0           1.0
online.spend     -0.1          0.0              0.0           1.0
store.trans       0.0          0.0             -0.3           0.0
store.spend       0.0          0.0             -0.2          -0.1
                 online.trans online.spend store.trans store.spend
age                      -0.1         -0.1         0.0         0.0
credit.score              0.0          0.0         0.0         0.0
distance.to.store         0.0          0.0        -0.3        -0.2
online.visits             1.0          1.0         0.0        -0.1
online.trans              1.0          1.0         0.0        -0.1
online.spend              1.0          1.0         0.0        -0.1
store.trans               0.0          0.0         1.0         0.9
store.spend              -0.1         -0.1         0.9         1.0
```

Graphical representation of the correlation matrix

The function `corrplot.mixed()` provided by the package `corrplot` allows to represent graphically the correlation matrix

```
> corrplot.mixed(cor_cust,upper="ellipse",
+ addgrid.col=colorpanel(50,"red","grey","blue"))
```



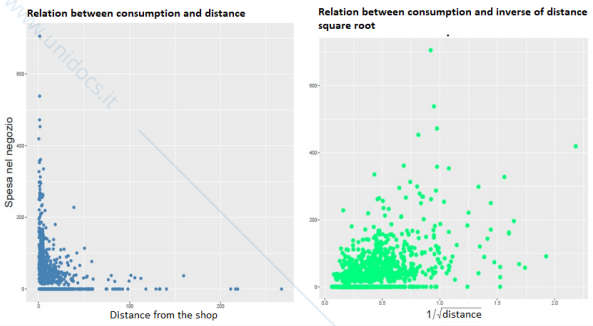
Variable transformations

When the correlation between two variables is not too high, it simply means that there is no linear relationship, however it is possible to find some different kind of relationship.

Variable	Useful transformation
Turnover, prices	$log(x)$
Distance	$\frac{1}{x}; \frac{1}{x^2}; log(x)$
Positive asymmetric distributions	$\sqrt{x}; log(x)$
Negative asymmetric distributions	x^2

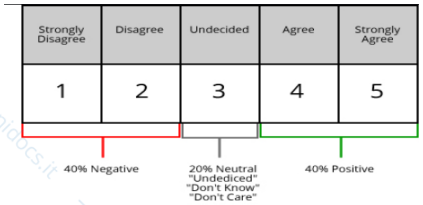
bubbleplot

```
> cor(customers$distance.to.store,customers$store.spend)
[1] -0.2414949
> customers$inv_sq_distance<-/sqrt(customers$distance.to.store)
> cor(customers$inv_sq_distance,customers$store.spend)
[1] 0.4843334
```



Likert Scale

The **Likert Scale** is a tool useful to measure an **attitude** or **opinion based on the** degree of agreement or disagreement of the interviewee in relation to a set of information:



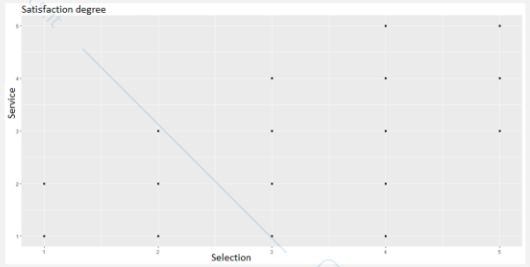
In `customers dataset` there are two variables which use the **Likert scale at 5 item**

- `sat_service`, custmer satistaction about the service, 1 (strongly unsatisfied) to 5 (strongly satisfied)
- `sat_selection`, custmer satistaction for the product selection, 1 (strongly unsatisfied) to 5 (strongly satisfied)

Likert Scale

`sat_service` and `sat_selection` are ordered categorical variables. Measuring the degree of association can be complicated. If we represent the scatterplot dots are **superimposed** and it is difficult evaluate the **density of each pair of answers**

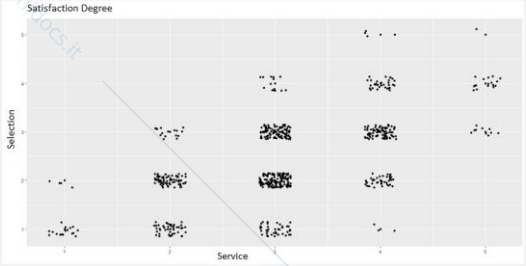
```
> ggplot(customers, aes(x=sat.service, =sat.selection)) +
  geom_point() + labs(list(x="Servce", y="Selection",
  title="Satisfaction degree"))
```



Likert Scale

To better distinguish the points it is possible adding to each point a **random noise** (width and height), using the function geom_jitter()

```
> ggplot(customers, aes(x=sat.service,y=sat.selection))
  + geom_point()+geom_jitter(width=0.15,height=0.15)+
  labs(list(x="Service",y="Selection",title="Satisfaction
  Degree"))
```



References

- http://www.statmethods.net/: with an introduction to **R** and several practical examples.
- Chang W. (2012), R Graphics Cookbook. Practical Recipes for Visualizing, O'Reilly Media. http://www.cookbook-r.com/Graphs/
- Chapman C., Mc Donnel Feit E. (2015), **R** for Marketing Research and Analytics, Springer. http://r-marketing.r-forge.r-project.org/
- Wickham H., Grolemund G. (2016), **R** for Data Science, O'Reilly. http://r4ds.had.co.nz/