INTRODUZIONE A R

Prof. Francesco Pappalardo
francesco.pappalardo@unict.it

Anno Accademico 2024/25

UNIVERSITÀ
degli STUDI
di CATANIA

COS'È R ?

R è un software OpenSource, che può essere definito come un sistema di analisi statistica e un linguaggio di programmazione.

Esso è il più utilizzato a livello mondiale dai ricercatori in campo statistico.

E' possibile scaricarlo gratuitamente sotto i vincoli della GPL ed è disponibile per diverse tipologie di hardware e sistemi operativi come: Unix, Linux, Windows, macOS.

Oltre al programma di base, vengono messe a disposizione anche una serie di moduli/librerie aggiuntive ed è possibile consultare la manualistica (in lingua inglese) sul sito ufficiale.

CARATTERISTICHE DI R

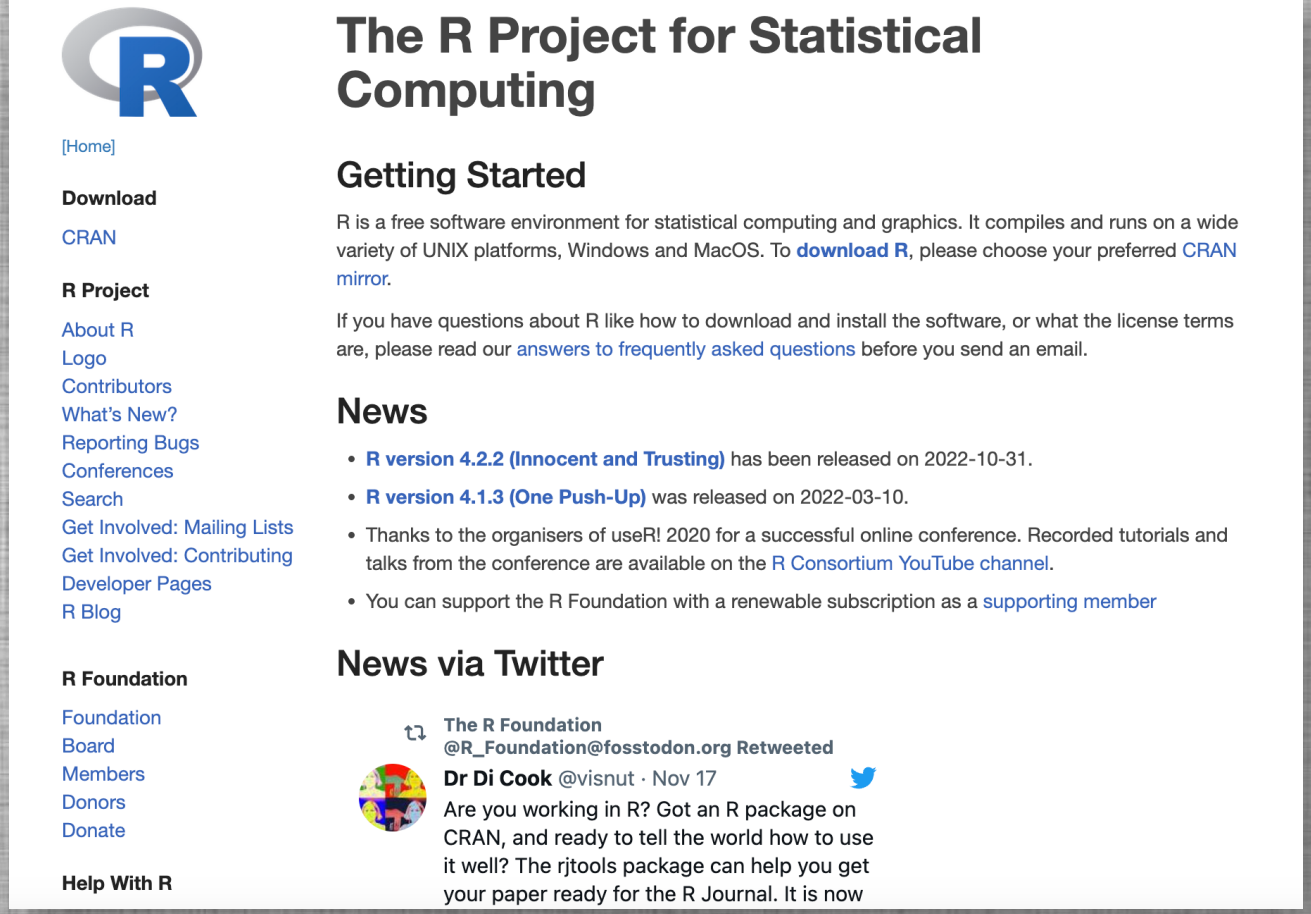
Le sue caratteristiche principali sono:

- Facilità nella gestione e manipolazione dei dati.
- Mette a disposizione una suite di strumenti per calcoli su vettori, matrici e molte altre operazioni complicate.
- Insieme di strumenti integrati per l'analisi statistica.
- Molteplici potenzialità grafiche.
- Possibilità di utilizzare un linguaggio di programmazione orientato ad oggetti.

PERCHÉ USARE R ?

- OpenSource, Multipiattaforma
- Esegue l'intero flusso di lavoro (codice) nello stesso punto:
  - Pre-elaborazione dei dati
  - Analisi dei dati
  - Visualizzazione dei dati
- Ricerca riproducibile organizzando l'analisi in vari script
  - Script: sequenza di istruzioni in linguaggio R salvate in un file di testo con estensione **.R (ex. mioScript.R)**
- Accesso ai metodi analitici organizzati in pacchetti

R - DOWNLOAD



[Home]

Download

CRAN

R Project

About R
Logo
Contributors
What's New?
Reporting Bugs
Conferences
Search
Get Involved: Mailing Lists
Get Involved: Contributing
Developer Pages
R Blog

R Foundation

Foundation
Board
Members
Donors
Donate

Help With R

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.

If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News

- **R version 4.2.2 (Innocent and Trusting)** has been released on 2022-10-31.
- **R version 4.1.3 (One Push-Up)** was released on 2022-03-10.
- Thanks to the organisers of useR! 2020 for a successful online conference. Recorded tutorials and talks from the conference are available on the R Consortium YouTube channel.
- You can support the R Foundation with a renewable subscription as a supporting member

News via Twitter

The R Foundation @R_Foundation@fosstodon.org Retweeted

Dr Di Cook @visnut · Nov 17

Are you working in R? Got an R package on CRAN, and ready to tell the world how to use it well? The rjtools package can help you get your paper ready for the R Journal. It is now

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

COS'È R-STUDIO ?

R-Studio è un ambiente di sviluppo integrato (IDE) per R, ed è un software open-source che gira su Gnu/Linux, Mac OS X e Windows.

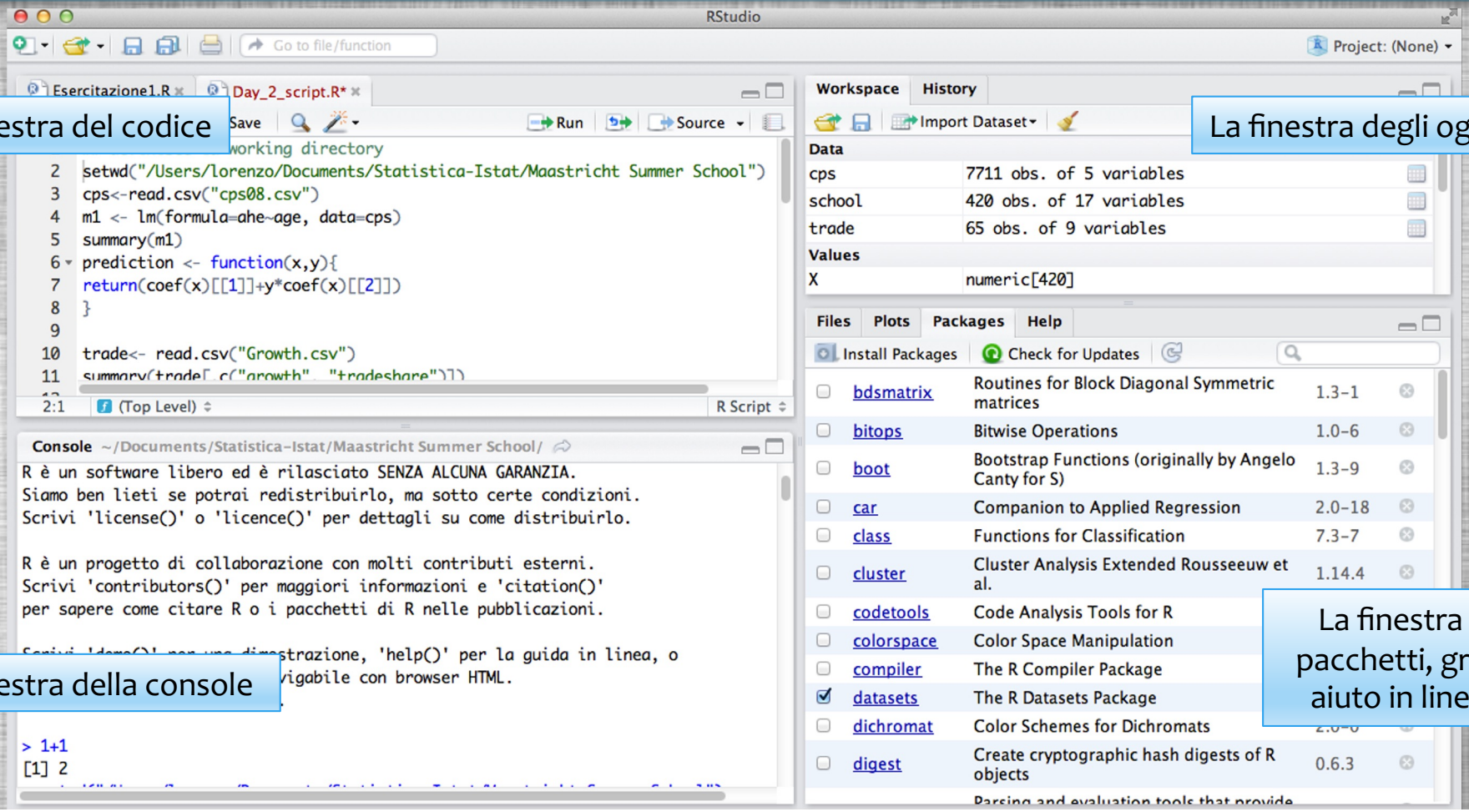
Permette di:

- Scrivere script (codice) ed eseguirli direttamente
- Installare pacchetti dalla console
- Interagire con i grafici (plot) prodotti

L'ambiente di lavoro di R-Studio è costituito da quattro finestre:

1. La finestra del codice (scrivere-eseguire script);
2. La finestra della console (riga di comando - output);
3. La finestra degli oggetti (elenco oggetti-cronologia dei comandi);
4. La finestra dei pacchetti, dei grafici, dell'aiuto in linea.

R-STUDIO – INTERFACCIA



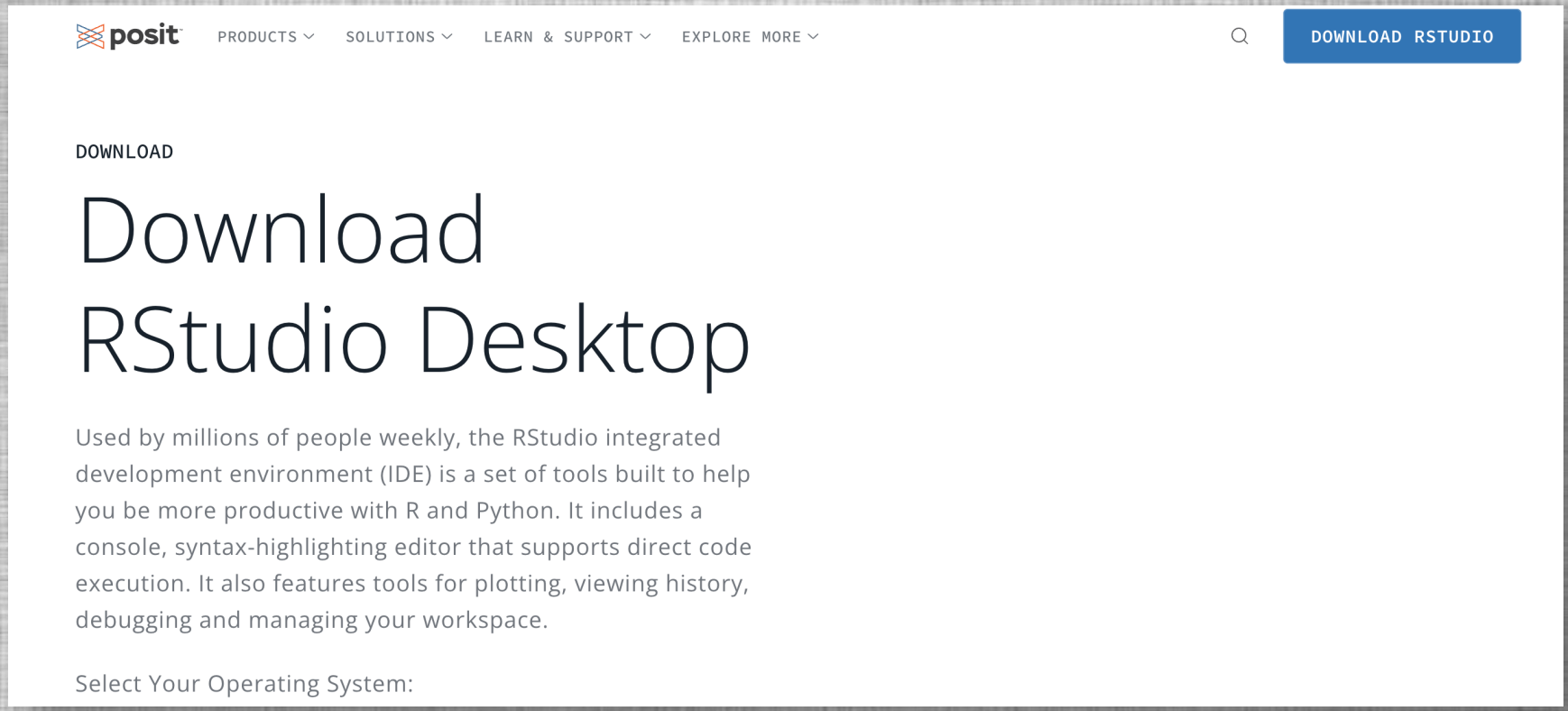
La finestra del codice

La finestra degli oggetti

La finestra della console

La finestra dei pacchetti, grafici, aiuto in linea…

R-STUDIO – WEBSITE & DOWNLOAD



https://posit.co/download/rstudio-desktop/

R - PACKAGES

Esistono due repository principali di pacchetti: **CRAN** e **Bioconductor**

- **CRAN:** è una rete di server che memorizza versioni identiche e aggiornate di codice e documentazione per R.

- **Bioconductor:** è un progetto software open source per l'analisi di dati genomici.
  - Più di 1200 pacchetti
  - Argomenti principali: sequenziamento, microarray, proteomica, etc.
  - I pacchetti sono descritti da vignette e manuali

R – PACKAGES (cont.)

Per poter utilizzare le funzioni che un packages mette a disposizione bisogna prima installarlo con il seguente comando da console:

install.packages(“nome_package”)

```
install.packages("ggplot2")
```

e successivamente verificare se quest’ultimo è stato installato correttamente caricando la libreria all’interno del progetto con questo comando

library(“nome_package”)

```
library(ggplot2)
```

R – PACKAGES (cont.)

E’ possibile installare e gestire pacchetti di R anche utilizzando la funzione BiocManager:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install(version = "3.14")
```

Esso garantisce che i pacchetti vengano scaricati da Bioconductor nel rispetto della versione di R in uso

OPERAZIONI NUMERICHE

Esempi

```
>3+2*4
  [1] 11
>3^2
  [1] 9
>(8\2)^2
  [1] 16
```

simbolo	operazione
+	somma
-	sottrazione
/	divisione
*	moltiplicazione
^	elevamento a potenza

OPERAZIONI NUMERICHE

```
> sqrt(9)
[1] 3
```

Radice quadrata

```
> log(1)
[1] 0
```

Logaritmo naturale

```
> exp(0)
[1] 1
```

Esponenziale

OPERAZIONI LOGICHE

Esempi

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

```
>(4*3)^2==4*3^2
  [1] FALSE
```

simbolo	operazione
<,>	minore di, maggiore di
<=, >=	minore o uguale a, maggiore o uguale a
==	uguale a
!=	diverso da
&	AND
|	OR
!	NOT

GLI OGGETTI IN R

Ogni entità che il programma crea e manipola è definita un oggetto, che può essere un numero, una variabile, una funzione, o più in generale, strutture costruite a partire da tali componenti.

I diversi tipi di variabili che R può gestire si distinguono in: numeri reali, numeri complessi, stringe di testo, e valori logici.

ASSEGNAZIONE

Per assegnare un valore ad un variabile (oggetto):

nome_variabile <- valore **Oppure** *nome_variabile = valore*

Esempio:

```
> x <- 4
> x
[1] 4
```

Così abbiamo creato la variabile **x** che contiene il valore 4;

*Per **modificare** il valore della variabile **x**, basta **riassegnarla**.*

GESTIONE DEGLI OGGETTI: ls()

Il comando **ls()** fornisce una lista di oggetti presenti nel workspace.

Esempio

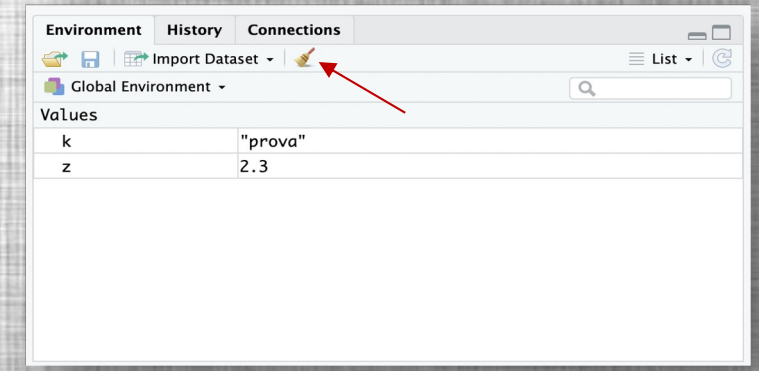
```
> x <- 4
> y <- "ciao"
> z <- 2.3
> ls()
[1] "x" "y" "z"
> k <- "prova"
> ls()
[1] "k" "x" "y" "z"
```

GESTIONE DEGLI OGGETTI: rm()

Il comando **rm()** cancella gli oggetti presenti nel workspace.

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

E' possibile rimuovere tutti gli oggetti nel workspace tramite il comando presente nella finestra degli oggetti.



SISTEMA DI AIUTO IN LINEA

E' possibile aprire una pagina del manuale, digitando uno dei seguenti comandi:

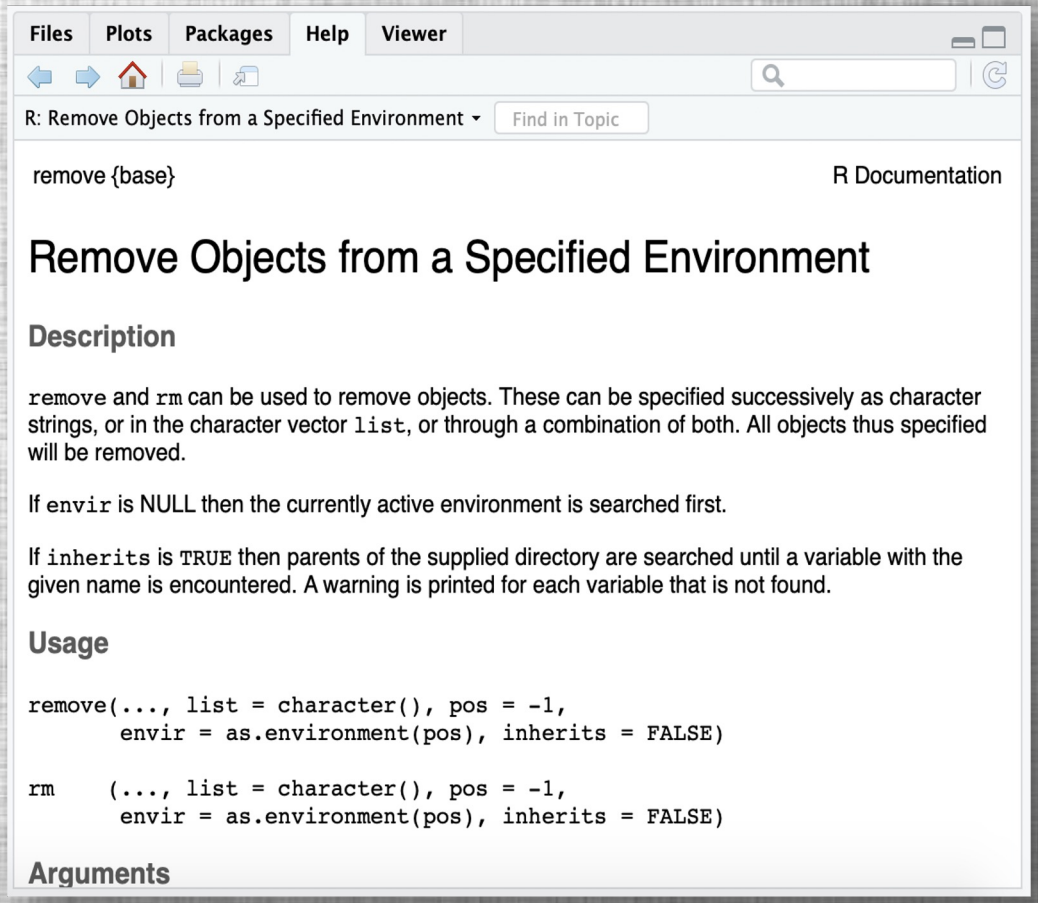
help('nome_comando') **Oppure** *?'nome_comando'*

Esempio

```
> help('rm')
>
> ?rm
```

CONTENUTO DI UNA PAGINA DEL MANUALE

- **Description:** descrizione generale
- **Usage:** sintassi del comando
- **Arguments:** dettagli sui parametri del comando
- **Details:** dettagli tecnici
- **Value:** risultato dell'esecuzione
- **References:** riferimenti bibliografici
- **See Also:** comandi collegati
- **Examples:** istruzioni eseguibili



WORKING DIRECTORY (WD)

La **working directory** (*wd*) è la directory dove R salva la sessione di lavoro, ovvero 2 file:

1. **.Rdata:** che contiene tutti gli oggetti della workspace (tipo i dati)
2. **.Rhistory:** che contiene la cronologia degli ultimi *n* comandi eseguiti dal programma stesso

E' possibile cambiare la working directory, basta andare nel menù **Session** e cliccare su **Set Working Directory -> Choose Directory**

WORKING DIRECTORY (WD)

- **getwd()**: mostra il path della cartella nel computer che è stata automaticamente selezionata al momento dell'installazione del programma.
- **setwd()**: per modificare lo spazio di lavoro, e cambiare directory. In alternativa, dal menu "File", si sceglie l'opzione "Cambia directory" e si sceglie la la cartella.

GESTIONE DEI FILE: SALVATAGGIO E LETTURA

- **Salvare area di lavoro:** permette di salvare gli oggetti all'interno dei file .Rdata attraverso il menù **Session** e cliccare su **Save Workspace As…**

```
>
> save.image("~/Desktop/workspace.RData")
>
```

- **Lettura area di lavoro:** permette di caricare gli oggetti salvati all'interno del file .Rdata attraverso il menù **Session** e cliccare su **Load Workspace…**

```
>
> load("~/Desktop/workspace.RData")
>
```

COSTRUTTI – IF-THEN-ELSE

```
if (test_expression) {
statement1
} else {
statement2
}
```

```
x <- -5
if(x > 0){
print("Non-negative number")
} else {
print("Negative number")
}
```

```
if (test_expression1) {
statement1
} else if (test_expression2) {
statement2
} else {
statement4
}
```

```
x <- 0
if (x < 0) {
print("Negative number")
} else if (x > 0) {
print("Positive number")
} else
print("Zero")
```

COSTRUTTI – FOR

```
for (val in sequence){
  statement
}
```

```
x <- c(2,5,3,9,8,11,6)
count <- 0
for (val in x) {
  if(val %% 2 == 0){
    count = count+1
  }
}
print(count)
```

TIPI DI DATI

R lavora con dati strutturati. Le strutture di dati contemplate sono:

- **variabili**: contengono un singolo dato
- **vettori**: insieme lineare di elementi omogenei per tipologia (tutti numeri, tutte stringhe, ecc.)
- **fattori**: vettore utilizzato per classificare o suddividere in livelli gli elementi di un altro vettore
- **array e matrici**: sono vettori multidimensionali, aventi due dimensioni, righe e colonne. Gli array sono insiemi di numeri con p dimensioni
- **data frame**: sono matrici bidimensionali dove ogni colonna puo` avere un tipo di dato diverso dalle altre
- **liste**: insieme di elementi non omogenei tra loro

TIPI DI DATI - FACTORS

I fattori sono vettori utilizzati per classificare o suddividere in livelli gli elementi di un altro vettore di pari lunghezza. Per trasformare un vettore, contenente le etichette dei livelli, in "fattore" si usa la funzione factor().

```
> SESSO = factor(SESSO)
> SESSO
[1] F M M F M F
Levels: F M

> PROV = factor(PROV)
> PROV
[1] RM RM TO NA TO NA
Levels: NA RM TO

> #per ottenere l'elenco dei livelli presenti nel fattore
> levels(PROV)

> is.factor(SESSO)
[1] TRUE
```

VETTORI: ASSEGNAZIONE

E' possibile assegnare un vettore con il seguente comando:

```
> x <- c(22, 5, 2019)
>
> x
[1]   22    5 2019
```

Per verificare se la variabile *x* è un vettore, basta eseguire il comando:

```
> is.vector(x)
[1] TRUE
```

VETTORI: ASSEGNAZIONE

```
> #†definizione di un vettore (colonna)
> x = c(1,2,3,4,5,6,7,8,9,10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> t(x) # trasposto (vettore riga)

> z = c("pippo", "topolino", "pluto", "paperino")
> z
[1] "pippo" "topolino" "pluto" "paperino"

> w = c(x, z) # concateno due vettori
> w
  [1] "1"        "2"        "3"        "4"        "5"        "6"
  [7] "7"        "8"        "9"        "10"       "pippo"    "topolino"
 [13] "pluto"    "paperino"
```

FUNZIONI SU UN VETTORE

R dispone di tante funzioni il cui argomento (parametro) è un vettore.

```
> x <- c(22, 5, 2019)
>
> mean(x)
[1] 682
>
> sum(x)
[1] 2046
>
> min(x)
[1] 5
```

```
> x <- c(22, 5, 2019)
>
> max(x)
[1] 2019
>
> length(x)
[1] 3
>
> range(x)
[1]    5 2019
```

ASSEGNAZIONE DEL VALORE DI UNA FUNZIONE

Il risultato di una funzione può anche essere memorizzato all'interno di una variabile.

```
> sommax <- sum(x)
>
> sommax
[1] 2046
```

Il tipo di variabile dipende dal valore restituito che la funzione restituisce.

```
> typeof(sommax)
[1] "double"
```

OPERAZIONI ARITMETICHE SUI VETTORI

Una normale **operazione binaria** (+ , - , * , / , ^) avente per argomenti due vettori, viene eseguita elemento per elemento.

```
> x <- c(1, 2, 3)
>
> y <- c(4, 5, 6)
>
> x + y
[1] 5 7 9
>
> x * y
[1]  4 10 18
```

OPERAZIONI LOGICHE SUI VETTORI

La stessa regola che vale per le operazioni aritmetiche vale anche per le operazioni logiche (vedi slide 12).

```
> x <- c(1, 2, 3)
>
> y <- c(2, 1, 5)
>
> x > y
[1] FALSE  TRUE FALSE
> x < y
[1]  TRUE FALSE  TRUE
```

VETTORI DI DIVERSA LUNGHEZZA

Se un'operazione aritmetica viene eseguita tra vettori di differente lunghezza, il vettore più corto viene ripetuto tante volte quante è necessario per eguagliare la lunghezza del vettore più lungo.

```
> x <- c(1, 2)
>
> y <- c(3, 4, 5, 6)
>
> x + y
[1] 4 6 6 8
```

N.B. Un messaggio di avviso viene dato solo se la lunghezza del vettore non è multipla di quella del vettore corto.

```
> x <- c(1, 2)
>
> y <- c(3, 4, 5)
>
> x + y
[1] 4 6 6
Warning message:
In x + y : longer object length is not a multiple of shorter object length
```

ARITMETICA CON I VETTORI

I vettori possono essere utilizzati in qualunque espressione aritmetica. In questo caso l'operazione è applicata a tutti gli elementi del vettore.

```
> campione <- c (6, 1, 5, 9, 4, 7, 8, 2, 5, 8)

> campione^2 #elevamento al quadrato
[1] 36  1 25 81 16 49 64  4 25 64

> min(campione)  #valore minimo
[1] 1

> max(campione)  #valore massimo
[1] 9

> length(campione)  #numero di valori (lunghezza del vettore)
[1] 10

> sum(campione) #somma degli elementi
[1] 55
```

ESTRAZIONE DI ELEMENTI DA UN VETTORE

E' possibile estrarre un sottovettore da un vettore, facendo seguire al nome del vettore la parte da estrarre (cioè la posizione dell'elemento da estrarre) racchiusa da parentesi quadre.

```
> x <- c(4, 5, 6)
>
> x[2]
[1] 5
>
> x[c(1,3)]
[1] 4 6
```

ESTRAZIONE DI ELEMENTI DA UN VETTORE (2)

In modo completare al precedente metodo possiamo indicare, anziché gli elementi da estrarre, quelli da ignorare, indicandoli con indici negativi.

```
> x <- c(4, 5, 6)
>
> x[-c(1,3)]
[1] 5
```

ESTRAZIONE DI ELEMENTI DA UN VETTORE (3)

Un altro metodo per estrarre elementi da un vettore è utilizzare delle operazioni logiche all'interno delle parentesi quadre.

```
> x <- c(3, 5, 1, 9)
>
> x[x < 4]
[1] 3 1
```

```
> x <- c(1, 0, 1, 0)
>
> x[x==1]
[1] 1 1
```

CREAZIONE VETTORI

- **seq(from = 1, to, by, length)** : genera un vettore che crea una sequenza regolare di numeri da from fino a to; by indica il passo della sequenza, mentre length la lunghezza.

- **rep(x, times)**: genera un vettore in cui un oggetto (x), numero o vettore, viene ripetuto un numero di volte pari a times

CREAZIONE VETTORI (esempio)

```
> x <- seq(from=1, to=5, by=1)
>
> x
[1] 1 2 3 4 5
>
> x <- 1:5
>
> x
[1] 1 2 3 4 5
>
> x <- seq(from=1, to=5, length=5)
>
> x
[1] 1 2 3 4 5
```

CREAZIONE VETTORI (esempio)

```
> rep(x = 1, times = 5)
[1] 1 1 1 1 1

> rep(c(1, 2), 2)
[1] 1 2 1 2
```

OPERATORI LOGICI

```
<     #minore
<=    #minore o uguale
>     #maggiore
>=    #maggiore o uguale
==    #uguale
!=    #diverso
&     #è l'intersezione ("e")
|     #è l'unione ("o")
!     #è la negazione ("non")
```

OPERATORI LOGICI

```
> 2 * 2 == 4
[1] TRUE

>2*2>4
[1] FALSE

> 2 * 2 >= 4
[1] TRUE

> 2 * 2 != 4
[1] FALSE

> a=1:10
> a <= 5
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE

> a[a>5]
[1]  6  7  8  9 10
```

OPERATORI LOGICI

```
> a[a<3 | a>8]
[1]  1  2  9 10

> a[a != 10]
[1] 1 2 3 4 5 6 7 8 9

> 0 / 0
[1] NaN

> 1/0
[1] Inf

> x <- c(1,2,NA,3)
> mean(x)
[1] NA
```

NA – valori mancanti

Spesso i dati su cui lavoriamo sono incompleti: i valori di alcune celle (alcune variabili di alcune osservazioni) sono mancanti. Per rappresentare i valori mancanti, R utilizza la costante NA (Not Available). Tecnicamente, NA è un vettore logico di lunghezza 1.

```
incompleto <- c(1, 2, NA, 10, 3)
# is.na mi dice se l'elemento è NA
is.na(incompleto)

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

NA – valori mancanti

```
1 + NA
## [1] NA
NA == 5
## [1] NA
```

NA tende a *contagiare* i valori non NA nel caso di operazioni logiche e aritmetiche:

```
TRUE | NA
## [1] TRUE
TRUE & NA
## [1] NA
FALSE | NA
## [1] NA
FALSE & NA
## [1] FALSE
```

MATRICI

Una matrice è una raccolta di elementi di dati disposti in un layout rettangolare bidimensionale. Gli elementi di una matrice sono tutti dello stesso tipo. Una matrice può anche essere considerata come un array di 2 dimensioni o un vettore con attributo *dim* e lunghezza *righe*colonne*

```
> A = matrix(
   c(2, 4, 3, 1, 5, 7), # the data elements
   nrow=2,              # number of rows
   ncol=3,              # number of columns
   byrow = TRUE)        # fill matrix by rows

> A                      # print the matrix
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

MATRICI (cont.)

Esistono molti modi per visualizzare ed accedere agli elementi di una matrice.

```
> A[2, 3] # element at 2nd row, 3rd column

[1] 7

> A[2, ] # the 2nd row

[1] 1 5 7

> A[ ,3] # the 3rd column

[1] 3 7

> A[ ,c(1,3)] # the 1st and 3rd columns

     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

MATRICI

```
(matrice1 <- matrix(c(11,12,13,14,21,22,23,24,31,32,33,34),nrow=4,ncol=3))
```

```
##      [,1] [,2] [,3]
## [1,]   11   21   31
## [2,]   12   22   32
## [3,]   13   23   33
## [4,]   14   24   34
```

```
# dim restituisce la dimensione (righe, colonne)
dim(matrice1)
```

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

```
# la lunghezza del vettore sottostante (r*c)
length(matrice1)
```

```
## [1] 12
```

MATRICI

```
# as.vector restituisce il vettore sottostante
as.vector(matrice1)

##  [1] 11 12 13 14 21 22 23 24 31 32 33 34

# t traspone la matrice:
# le righe diventano colonne, e viceversa
t(matrice1)

##      [,1] [,2] [,3] [,4]
## [1,]   11   12   13   14
## [2,]   21   22   23   24
## [3,]   31   32   33   34
```

MATRICI

Attraverso le funzioni colnames() e rownames() posso assegnare o richiamare i nomi delle colonne e delle righe.

```
# colnames assegna i nomi alle colonne
colnames(matrice1) <- c("C_A", "C_B", "C_C")
# rownames assegna i nomi alle righe
rownames(matrice1) <- c("R1","R2","R3","R4")
matrice1
```

MATRICI

```
##     C_A C_B C_C
## R1   11  21  31
## R2   12  22  32
## R3   13  23  33
## R4   14  24  34
```

LISTE

Una lista è una collezione ordinata ed eterogenea di oggetti anche complessi. Poiché la lista può contenere altre liste, attraverso la lista si possono costruire delle strutture ad albero. Per creare una lista, si usa la funzione list() :

```
lista1 <-list(
    matrice=matrix(10:18,nrow=3),
    saluto=rep("ciao",3),
    colazione=c("cappuccio","brioche"))
```

LISTE

```
lista1
```

```
## $matrice
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
##
## $saluto
## [1] "ciao" "ciao" "ciao"
##
## $colazione
## [1] "cappuccio" "brioche"
```

```
str(lista1)
```

```
## List of 3
##  $ matrice  : int [1:3, 1:3] 10 11 12 13 14 15 16 17 18
##  $ saluto   : chr [1:3] "ciao" "ciao" "ciao"
##  $ colazione: chr [1:2] "cappuccio" "brioche"
```

LISTE

Estrazione dati da liste:

```
lista1 <- list(nomi=c("Marco","Alessia"), amici=c("Luigi","Francesco","Raffaella"),
lista1$nomi

## [1] "Marco"   "Alessia"

# la sintassi [[]] estrae i valori dell'oggetto selezionato
lista1[[2]]

## [1] "Luigi"     "Francesco" "Raffaella"

# la sintassi [] estrae l'oggetto
lista1[2]

## $amici
## [1] "Luigi"     "Francesco" "Raffaella"
```

DATAFRAME

Il dataframe è una particolare lista e rappresenta la matrice dei dati in cui ad ogni riga corrisponde una osservazione e ad ogni colonna una variabile.
Tra le diverse opzioni disponibili, è possibile costruire un data.frame direttamente con la funzione **data.frame()**

```
x<-data.frame(a=1:4, sesso=c("M","F","F","M"))# crea un dataframe
```

	a	sesso
1	1	M
2	2	F
3	3	F
4	4	M

La differenza più importante fra data.frame e matrice è che mentre nella seconda tutti gli elementi sono dello stesso tipo, nel data.frame le colonne possono essere di tipo diverso.

DATAFRAME (cont.)

Se voglio conoscere le dimensioni del dataframe:

```
> dim(x)

[1] 4 2
```

Se voglio aggiungere, ad esempio, la variabile età al dataframe:

```
x$eta <- c(2.5,3,5,6.2) #aggiungi una variabile
```

DATAFRAME

Per selezionare una parte del dataframe possiamo effettuare le seguenti operazioni:

```
X[,1:3] #selezionare tutte le righe del dataset e le prime tre variabili

X[1:5,]# per selezionare le prime 5 righe e tutte le colonne
```

Oppure possiamo utilizzare le operazioni logiche:

```
X[giorni_vacanza<3 & età>21,]
```

DATAFRAME

Estrazione dati da dataframe:

```
nome <- c(”luigi”,”mario”,”antonella”,”luca”)
anno <- c(1956,1945,1972, 1976)
condizione <- c(”exp”,”controllo”,”exp”,”controllo”)
soggetti <- data.frame (nome,anno,condizione)
soggetti
```

```
##        nome anno condizione
## 1     luigi 1956        exp
## 2     mario 1945  controllo
## 3 antonella 1972        exp
## 4      luca 1976  controllo
```

```
# stessa sintassi delle matrici
# tutte le righe, colonna 3
soggetti[,3]
```

```
## [1] exp       controllo exp       controllo
## Levels: controllo exp
```

```
# riga 3, tutte le colonne
soggetti[3,]
```

```
##        nome anno condizione
## 3 antonella 1972        exp
```

```
# cella 3,3
soggetti[3,3]
```

```
## [1] exp
## Levels: controllo exp
```

DATAFRAME

Estrazione dati da dataframe:

```
# il data frame è una lista, e dunque vale anche la sintassi $
soggetti$anno

## [1] 1956 1945 1972 1976

# il data frame è una lista, e dunque posso selezionare per colonna
soggetti[2]

##   anno
## 1 1956
## 2 1945
## 3 1972
## 4 1976
```

FUNZIONI NOTE

Funzione **sample (v)**: mescola l'ordine del vettore v.
Funzione **sample (v, n)**: estrae n valori casuali dal vettore v.

```
# mescola il vettore
(mischiato <- sample (vettore1))
```

```
##  [1]  3  6  2  5  4  9  1  7 10  8
```

```
# sample (v,n) fa il sample e restituisce i primi n numeri
sample(vettore1, 4)
```

```
## [1]  6  9  4 10
```

```
# se lo rifaccio, avrò numeri differenti
sample(vettore1, 4)
```

```
## [1] 3 5 1 2
```

```
# con replace = TRUE i valori estratti vengono rimessi nell'urna
sample(1:4,10,replace = TRUE)
```

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

FUNZIONI NOTE

Ordinare un vettore:

```
# ordina un vettore
# decreasing = TRUE ordine decrescente
sort(mischiato)

##  [1]  1  2  3  4  5  6  7  8  9 10

order(mischiato)

##  [1]  7  3  1  5  4  2  8 10  6  9

mischiato[order(mischiato)]

##  [1]  1  2  3  4  5  6  7  8  9 10
```

FUNZIONI NOTE

cbind e rbind creano delle matrici legando vettori o matrici in colonne (cbind) o in righe (rbind).

```
# rbind prende gli argomenti e li mette una riga dopo l'altra
rbind(c(1,2),c(3,4),c(5,6))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

```
# incolonna gli argomenti
cbind(c(1,2),c(3,4),c(5,6))
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

IMPORTAZIONE DEI DATI

```
# Read CSV into R
MyData <- read.csv(file="file.csv",
  header=TRUE,
  sep=",")
```

```
library(gdata)                          # load gdata package
mydata = read.xls("mydata.xls")  # read from first sheet
```

```
mydata = read.table("mydata.txt") # read text file
```

PLOT – CREAZIONE DI UN GRAFICO (1)

Un grafico è un oggetto grafico complesso. Gli oggetti grafici complessi sono composti da:

- uno spazio grafico (bidimensionale)
- degli oggetti grafici (complessi o semplici)
- delle strutture di riferimento (titoli, legende, assi, etichette, annotazioni), che permettono l'interpretazione e facilitano la lettura dei dati

PLOT – CREAZIONE DI UN GRAFICO (1)

Gli oggetti grafici rappresentano:

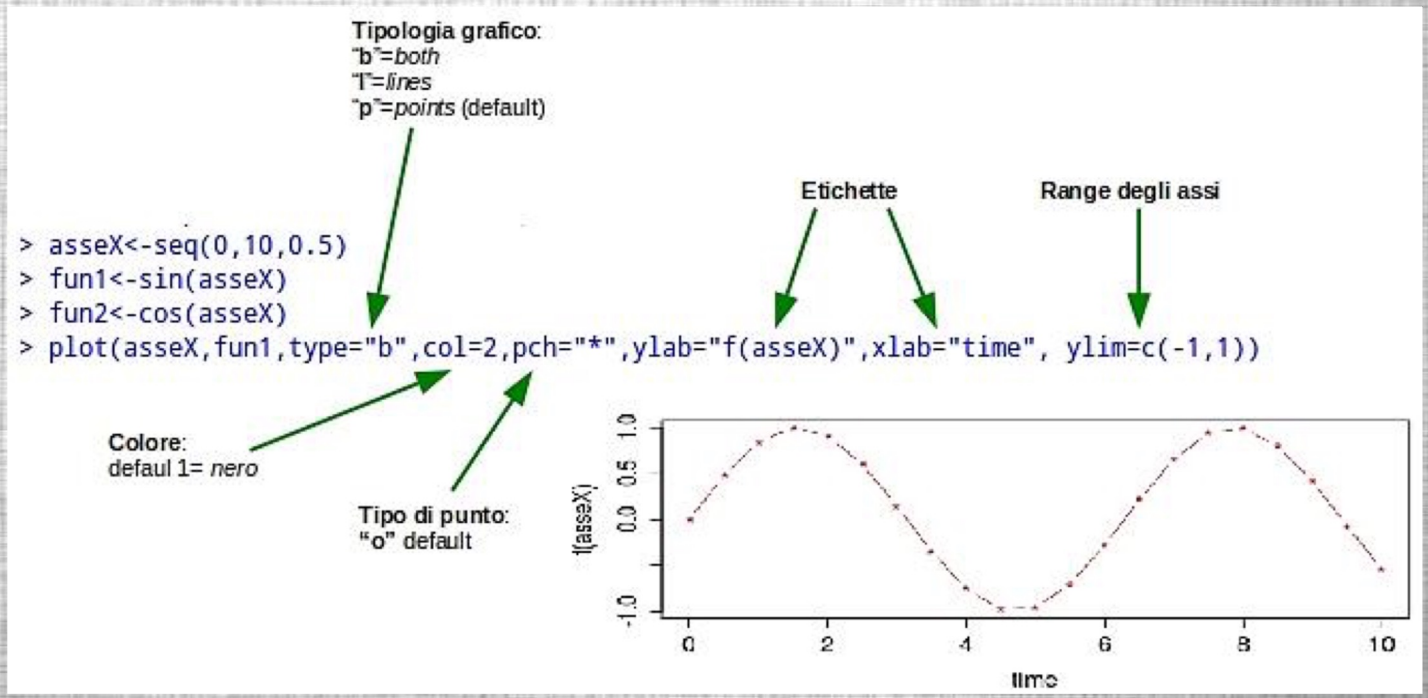
- osservazioni
- statistiche
- relazioni (fra osservazioni o fra statistiche)

Gli oggetti grafici semplici sono formati da figure geometriche: punti (0d), linee o curve (1d), rettangoli o aree (2d), da icone o da elementi testuali

PLOT – CREAZIONE DI UN GRAFICO (1)

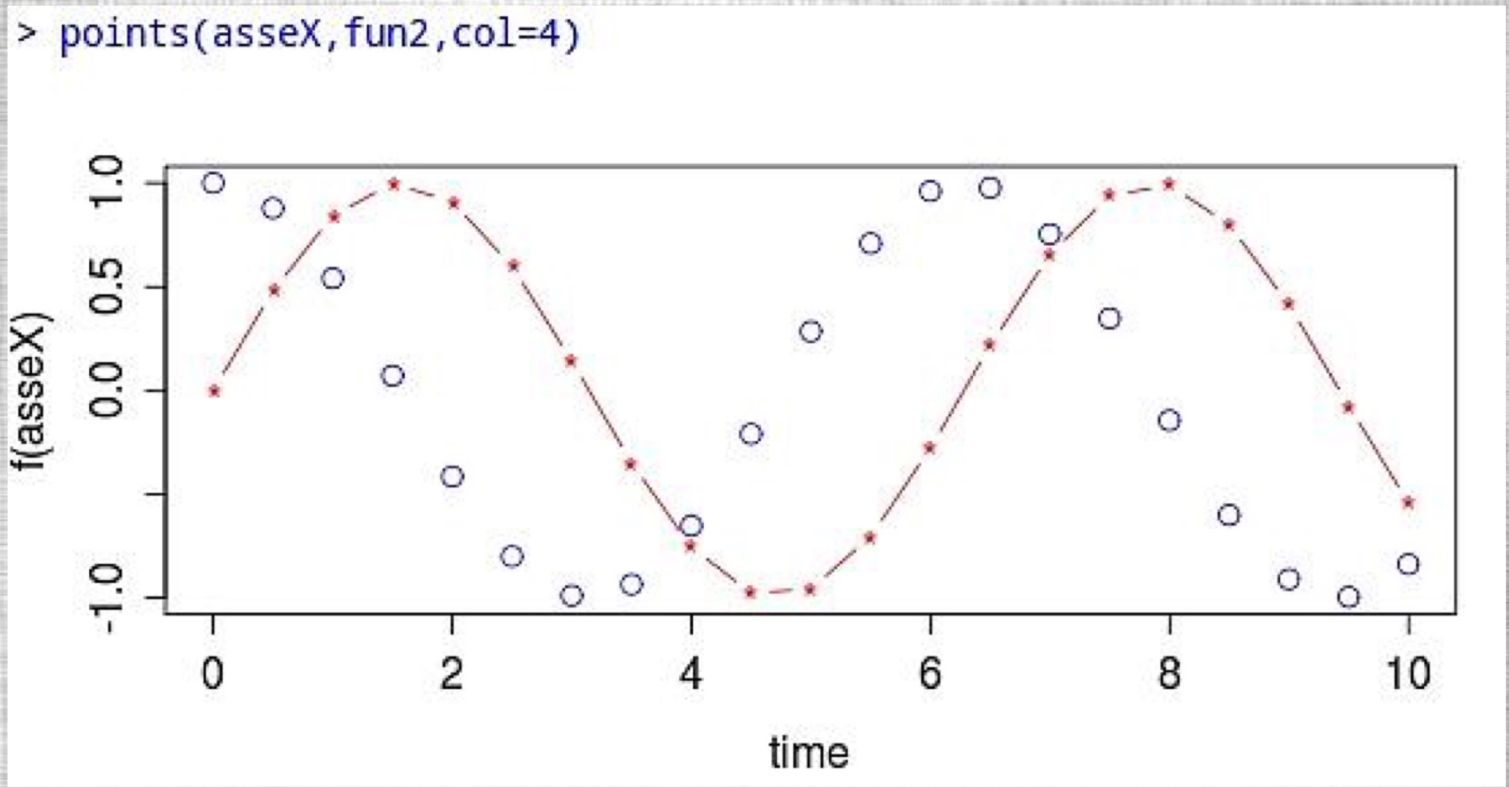
Le funzioni principali per disegnare un grafico sono:

- **plot**()
- **point**()
- **lines**()



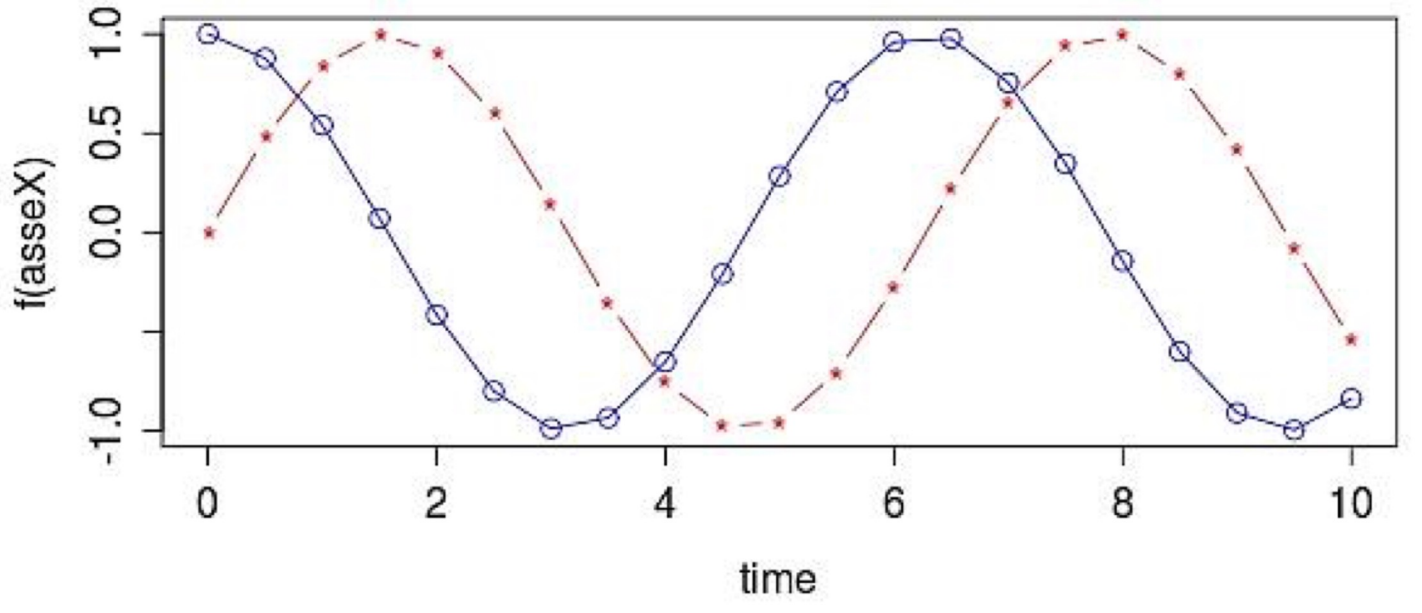
PLOT – CREAZIONE DI UN GRAFICO (2)

```
> points(asseX,fun2,col=4)
```



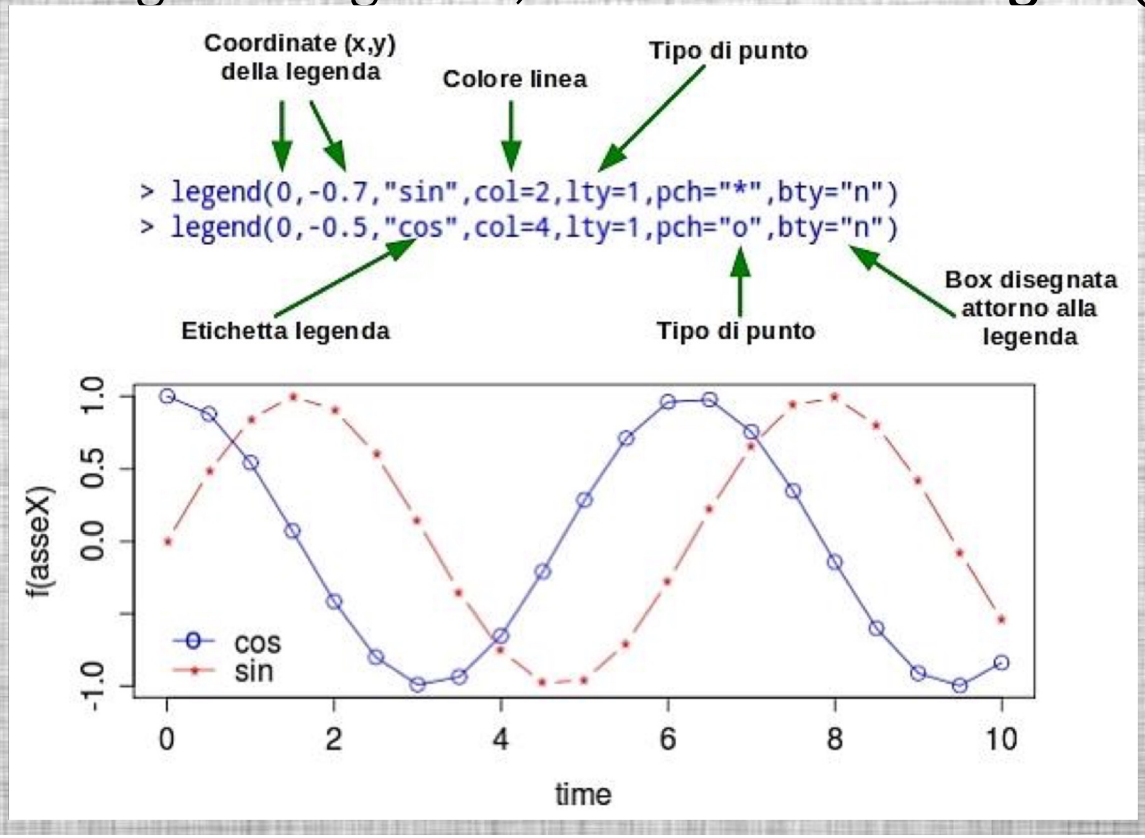
PLOT – CREAZIONE DI UN GRAFICO (3)

```
> lines(asseX,fun2,col=4)
```



PLOT – LEGENDA

Per aggiungere una legenda al grafico, esiste la funzione **legend**().



PLOT – SALVARE UN GRAFICO

Per salvare un grafico invece di visualizzarlo, si può usare una delle seguenti funzioni:

Funzione	Salva in
pdf("mygraph.pdf")	pdf file
png("mygraph.png")	png file
jpeg("mygraph.jpg")	jpeg file
bmp("mygraph.bmp")	bmp file
postscript("mygraph.ps")	postscript file

Esempio:

```
jpeg("myplot.jpg")
plot(x)
dev.off()
```

PLOT AVANZATI – ISTOGRAMMA

Dato un vettore di valori, per creare un istogramma si può usare la funzione "**hist**".

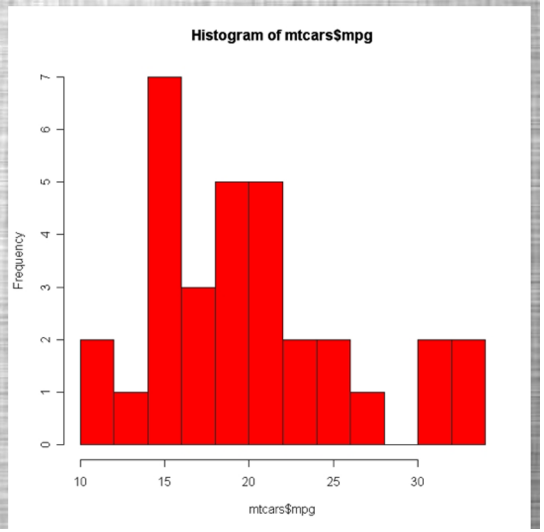
Sintassi:

hist(*x*, breaks="*Sturges*", ...)

- **x**: è un vettore di valori
- **breaks**: può essere un numero per indicare il numero di bins, un vettore con i punti di taglio, o il nome di un algoritmo.

Esempio:

```
hist(mtcars$mpg, breaks=12, col="red")
```



PLOT AVANZATI – BAR PLOT

È possibile creare grafici a barre con il comando "**barplot**".

Sintassi:

barplot(*height***, names.arg=***NULL***, horiz=***FALSE***, xlab=***NULL***, ylab=***NULL***, ...)**

height: è un vettore o matrice che descrive le barre del grafico

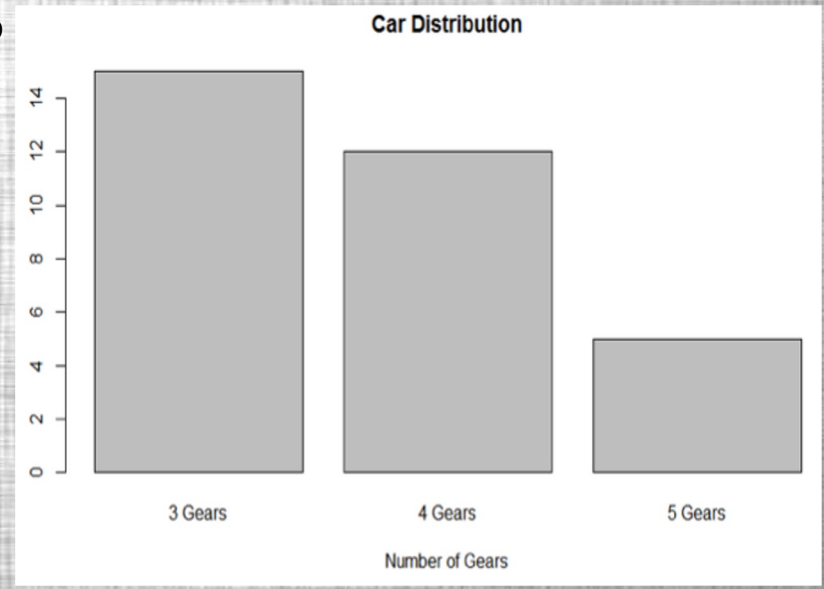
names.arg: è un vettore di nomi da mostrare per ogni barra

horiz: è un booleano che indica l'orientamento del grafico

xlab, **ylab**: sono le etichette sull'asse x o y.

Esempio:

```
counts <- table(mtcars$gear)
barplot(counts, main="Car Distribution",
        names.arg=c("3 Gears", "4 Gears", "5 Gears"),
        xlab="Number of Gears")
```



PLOT AVANZATI – GRAFICI A TORTA

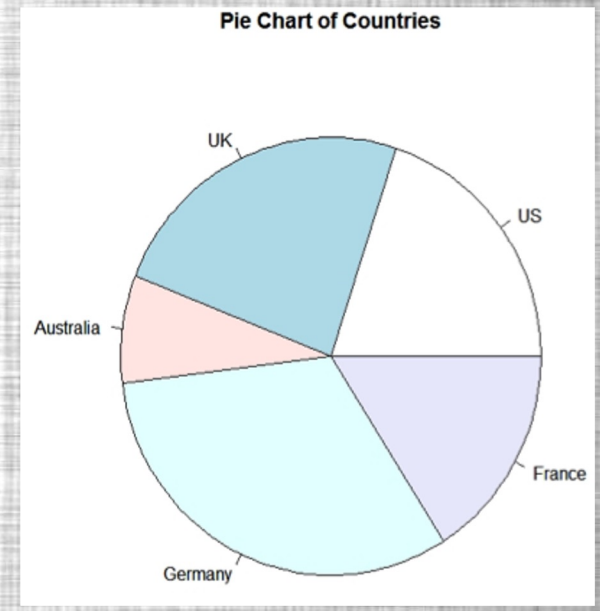
È possibile creare grafici a torta con il comando "**pie**".

Sintassi:

pie(*slices***, labels=***NULL***, main=***NULL***, ...)**

slices: è un vettore che descrive le porzioni del grafico

labels: è un vettore di nomi per ogni porzione del grafico

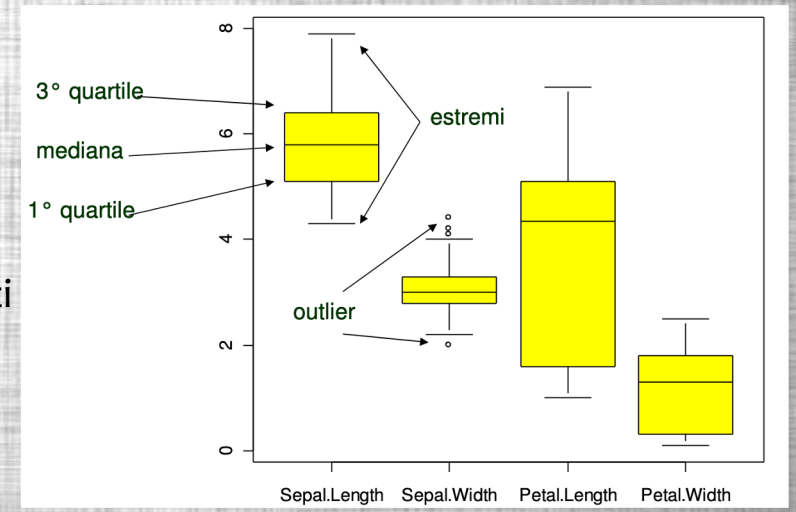


Esempio:

```
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```

PLOT AVANZATI - BOXPLOT

I BoxPlot forniscono una descrizione grafica sintetica di un insieme di dati utilizzando semplici statistiche. Sono una modalità efficace per valutare visivamente se ci sono delle differenze fra diversi gruppi.



Sintassi:

boxplot(*formula***, data=***NULL***, ...)**

formula: un modello a partire dal quale definire l'insieme dei dati

data: il dataframe contenente i dati da elaborare

Esempio:

```
boxplot(iris[,1:4], col="yellow", main="Title plot",
        xlab="Label x", ylab="Label y")
```

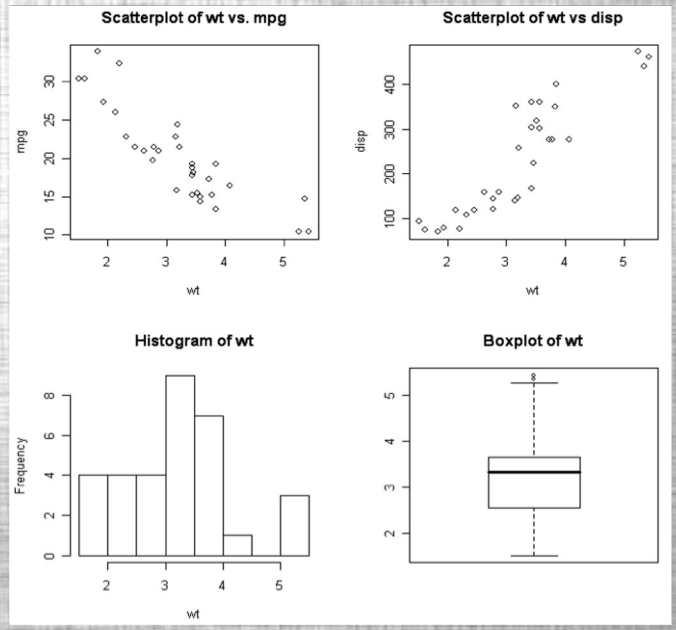
PLOT AVANZATI – COMBINARE GRAFICI

Per combinare più grafici in una singola immagine si può impostare il parametro "**mfrow**" del sottosistema grafico tramite il comando "**par**".

Il parametro è un vettore di due elementi che rappresentano rispettivamente il numero di righe e colonne del grafico.

Esempio:

```
# 4 figures arranged in 2 rows and 2 columns
attach(mtcars)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
```



ESERCIZI

Costruire un vettore **x** con elementi: 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5, 6, 6.5, 7, 7.5, 8, 8.5, 9, 9.5 e 10.
Per **x** calcolare: lunghezza, somma degli elementi, prodotto degli elementi, media e varianza.

Dai vettori:

```
x <- c(3, 5, 9, 1, 2, 10, 12, 24, 6)
y <- c(0.15, 0, 0.32, 0.51, 0.18, 0.22, 0.6, 0.98, 0.12)
z <- c(123, 415, 981, 643, 1080, 89, 46, 75, 910)
```

si costruisca la matrice che ha come colonne i tre vettori.
Si calcoli poi la media degli elementi di **z** che sono minori di 900.

Si costruisca il dataframe:

```
> x
    a    b
1   2    I
2   3   II
3   4  III
```

A partire dal dataframe creato si acceda alla seconda colonna di `x`.