

Didattica integRativa

EseRcitazioni pRatiche con il software R

dott. Luca Menghini Ph.D.

Assegnista di ricerca, Dipartimento di Psicologia,
Università degli Studi di Bologna

luca.menghini3@unibo.it

Analisi dei dati in ambito di comunità

Corso di laurea magistrale in Psicologia di comunità, della promozione
del benessere e del cambiamento sociale

Università degli Studi di Padova
Anno Accademico 2021 - 2022




Mi presento

- **2014: Triennale in Scienze Psicologiche Sociali e del Lavoro @uniPD**

“Biofeedback training per la gestione dello stress nei contesti organizzativi”

- **2016: Magistrale in Psicologia Sociale, del Lavoro e della Com. @uniPD**

“Un Protocollo di Assessment Psicofisiologico per la Valutazione del Rischio Stress lavoro-correlato”

- **2017: Tirocinio post-laurem presso Inside Performance** (stress management & biofeedback in contesti organizzativi e sportivi) + **Laboratorio di psicofisiologia @uniPD** (processamento dati e accuratezza sensori actigrafici) ← 

- **2017-2021: Dottorato in Scienze Psicologiche @uniPD**

“Workplace stress in real time: Towards the psychophysiological assessment of stressors and strain under ecological conditions”

- **2020: Esperienza di ricerca all'estero @SRI International (CA, USA)**

Accuratezza e uso Sleep Consumer Technology, Relazioni giornaliere tra sonno e stress

- **2021: Assegno di ricerca @uniBO**

“State workaholism as a predictor of daily fluctuations in blood pressure, emotional exhaustion, and sleep quality”

Obiettivi delle eseRcitazioni

- Acquisire competenze di base nell'uso del software R
- Consolidare le conoscenze apprese nel corso
- Implementare le tecniche analitiche apprese durante il corso utilizzando il software R su dataset reali
- Svolgere insieme gli esercizi propedeutici all'esame

Le slide e tutti materiali usati nelle eseRcitazioni verranno di volta in volta caricati e aggiornati sulla repository all'indirizzo

<https://github.com/Luca-Menghini/eseRcitazioni>

Programma incontri facoltativi

- **Giorno 1 (Get started):** Installare R e RStudio, acquisire confidenza con l'interfaccia del software, e alcuni comandi di base
- **Giorno 2 (R objects):** Vettori, fattori, matrici e dataframe
- **Giorno 3 (R workspace):** Worskpace e working directory, caricare un dataset e calcolare le principali statistiche descrittive
- **Giorno 4 (Graphics):** Visualizzare i dati e interpretare i principali tipi di grafici
- **Giorno 5 (Linear Models):** Modelli di regressione lineare
- **Giorno 6 (Linear Mixed-Effects and Structural Equation Models):** Regressione multilivello e path analysis

Get started

Il linguaggio R



- R è un linguaggio e un ambiente di programmazione per il **calcolo statistico** e la **visualizzazione grafica** dei dati
- basato sul 'linguaggio S' (Becker & Chambers, 1984), usato per creare il software S-Plus e poi R, creato da Ross Ihaka e Robert Gentleman, nel 1996
- oggi sviluppato da un gruppo di ricerca internazionale (*R Core Team*), che aggiorna periodicamente (ogni anno) il programma di base (*Base R*)
- progressiva ed esponenziale aggiunta di nuovi pacchetti (*packages*) che ne estendono le funzionalità

Il linguaggio R



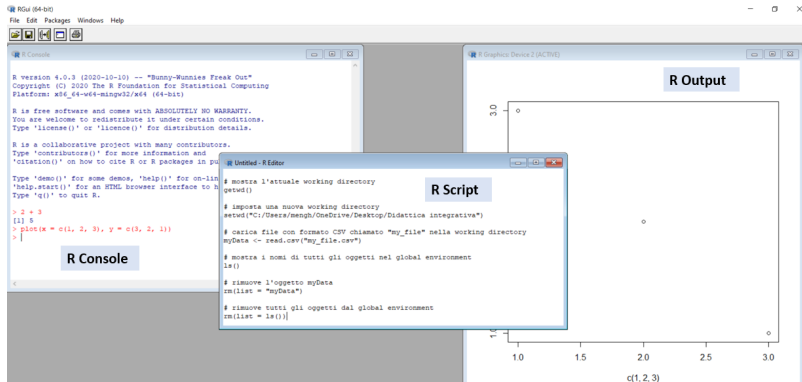
- ampia varietà di tecniche statistiche (es. modelli lineari e non lineari) e grafiche (es. pacchetto **ggplot2** - **link a lezione dedicata**)
- pensato per essere **semplice** ma al contempo in grado di generare **output di alta qualità** (grafici, tabelle e report con equazioni e simboli matematici, ecc.); funzioni di default ottimizzate + possibilità di avere il **pieno controllo**
- **software gratuito** (GNU General Public License), **open source** (ogni funzione è documentata e visibile in dettaglio) che funziona su tutti i **principali sistemi operativi**: Windows, MacOS, e UNIX (es. Linux)
- enorme comunità di utenti (per qualsiasi problema, basta googlare ;-))

Scaricare e installare R



1. Scaricare R dal sito <https://www.r-project.org/>
CRAN (*Comprehensive R Archive Network*): rete di server che offrono le versioni aggiornate e la relativa documentazione
2. Cliccare sulla voce **CRAN** nel menu *Download* a sinistra, selezionare un mirror (es. [il primo](#), oppure [quello dell'Università di Padova](#)), quindi il proprio sistema operativo (Linux, MacOS, o Windows)
3. Installare R aprendo il file .exe (Windows) o .pkg (MacOS) appena scaricato, oppure seguire i comandi in base alla propria versione di Linux

L'interfaccia di Base R



- **R Console:** per scrivere (>) ed eseguire (tasto Enter) velocemente dei comandi
- **R Script** (menu File > New R Script): per scrivere, modificare e salvare sequenze di comandi (salvati con formato .R)
- **Outputs** (es. plot): finestre che si aprono lanciando il relativo comando

Alcuni comandi elementari

Commenti (#)

```
# questo è un commento
```

Semplici operazioni matematiche

```
2 + 2 # addizione
```

```
[1] 4
```

```
2 * 2 # moltiplicazione
```

```
[1] 4
```

```
log(3) # logaritmo naturale
```

```
[1] 1.098612
```

```
exp(1) # funzione esponenziale
```

```
[1] 2.718282
```

Espressioni più lunghe (con **parentesi tonde**)

```
sqrt(5) * ( (4 - 1/2)^2 - pi/2^(1/3) )
```

```
[1] 21.81623
```

Assegnare valori a degli **oggetti** (<-)

```
x <- 3 # creo oggetto 'x' associato al valore 3
```

```
x # stampo il valore di x
```

```
[1] 3
```

I nomi degli oggetti possono includere lettere, numeri, trattini bassi e punti (es. pippo, pippo32, pippo.32, pippo_32)

```
pippo_32 <- x / 3
```

```
pippo_32 # stampo il valore di pippo_32
```

```
[1] 1
```

R è sensibile alle maiuscole!

Mentre non è sensibile agli spazi

```
3+2
```

```
[1] 5
```

```
3 + 2
```

```
[1] 5
```

Hands on: Operazioni aritmetiche con R

Calcola il risultato delle seguenti operazioni utilizzando R ([soluzioni](#)):

Source: <https://psicostat.github.io/Introduction2R/first-comands.html#esercizi>

$$1. \frac{(45+21)^3 + \frac{3}{4}}{\sqrt{32 - \frac{12}{17}}}$$

$$2. \frac{\sqrt{7-\pi}}{3(45-34)}$$

$$3. \sqrt[3]{12 - e^2} + \ln(10\pi)$$

$$4. \frac{\sin(\frac{3}{4}\pi)^2 + \cos(\frac{3}{2}\pi)}{\log_7 e^{\frac{3}{2}}}$$

$$5. \frac{\sum_{n=1}^{10} n}{10}$$

Extra: Assegna il risultato dell'operazione 4 all'oggetto **x**, il risultato della 5 all'oggetto **y**, e calcola la somma **x + y**

RStudio



- RStudio è un ambiente di sviluppo integrato per R, che lo integra con un'**interfaccia grafica ottimizzata** per facilitarne l'utilizzo (es. accesso a file e oggetti, grafici, dataset, ecc.) presentando tutto in un'unica finestra
- fondato da J J Allaire nel 2009 (scritto con linguaggio Java e C++), gestito e sviluppato da gruppo di ricerca internazionale (gli stessi di **tidyverse** e **shiny**)
- **gratuito e open source** (GNU General Public License) + versioni a pagamento

Scaricare e installare RStudio



NB: soltanto **dopo** aver installato R

1. Scaricare RStudio dal sito <https://rstudio.com>
2. Cliccare sulla voce **Download** nel menu in alto, selezionare la versione gratuita (FREE) di **RStudio Desktop**, quindi il proprio sistema operativo
3. Installare RStudio aprendo il file appena scaricato

L'interfaccia di RStudio

The screenshot displays the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. Below the menu is a toolbar with icons for opening files, saving, running, and other functions. The main window is divided into four panes:

- Source:** Contains an R script with the following code:

```
1 sleep # stampo dataset sleep
2
3
4 plot(extra-group,data=sleep) # boxplot
5
6
7
```
- Console:** Shows the output of the executed code, which is a table of sleep data for two groups (1 and 2). The output is as follows:

group	extra	sleep
1	0.8	1
1	0.0	1
1	2.0	1
1	1.9	2
1	0.8	2
1	1.1	2
1	0.1	2
1	-0.1	2
1	4.4	2
1	5.5	2
1	1.6	2
1	4.6	2
1	3.4	2
2	0.8	1
2	0.0	1
2	2.0	1
2	1.9	2
2	0.8	2
2	1.1	2
2	0.1	2
2	-0.1	2
2	4.4	2
2	5.5	2
2	1.6	2
2	4.6	2
2	3.4	2
- Environment:** Displays the R Objects in the workspace. It shows three objects: 'ab' (36 obs. of 3 variables), 'abilities' (36 obs. of 3 variables), and 'att' (210 obs. of 2 variables).
- Plots:** Shows a boxplot of the 'extra' variable for the two groups. The x-axis is labeled 'group' with values 1 and 2. The y-axis is labeled 'extra' with values -1, 1, 3, and 5. The boxplot shows that the distribution of 'extra' values is shifted higher for group 2 compared to group 1.

- **Source:** R Scripts (.R), documenti e presentazioni (.Rmd), applicazioni (.app), ecc. Per lanciare uno o più comandi, selezionali e premi Ctrl + Enter oppure clicca sul tasto Run in alto a destra
- **Environment** (oggetti presenti nel workspace) & **History** (storico comandi eseguiti)

Hands on: Operatori relazionali e logici

Operatori relazionali

```
3 == 3 # uguale
[1] TRUE

3 != 3 # diverso
[1] FALSE

x >= 3 # maggiore o uguale
[1] TRUE

5 %in% c(3, 5, 8) # inclusione
[1] TRUE
```

Operatori logici

```
x <- TRUE
y <- !x # negazione
y
[1] FALSE

x & (5 < 2) # congiunzione
[1] FALSE

x | (5 < 2) # disgiunzione inclusiva
[1] TRUE
```

Esercizi sugli operatori relazionali e logici:

Source: <https://psicostat.github.io/Introductions2R/first-comands.html#esercizi>

1. Definisci una proposizione per valutare la seguente condizione: “x è un numero compreso tra -4 e -2 oppure è un numero compreso tra 2 e 4”
2. Definisci due relazioni false e due vere che ti permettano di valutare i risultati di tutti i possibili incroci che puoi ottenere con gli operatori logici & e |
3. Esegui le seguenti operazioni $4 \wedge 3 \in c(2,3,4)$ e $4 * 3 \in c(2,3,4)$. Cosa osservi nell’ordine di esecuzione degli operatori?

Oggetti e funzioni

- **Oggetti:** identificano dei valori salvati nel workspace (**Environment**); i valori vengono assegnati agli oggetti con il simbolo `<-` (minore e meno); per richiamare un oggetto è sufficiente scrivere il suo nome

```
pippo_32 <- 2 # assegno valore a oggetto
pippo_32 # stampo oggetto
[1] 2
pippo_32 <- pippo_32 + 1 # aggiorno oggetto
pippo_32
[1] 3
```

- **Funzioni:** etichette associate a sequenze di comandi programmati per restituire uno specifico output (chiamato **valore**) sulla base di uno o più input (chiamati **argomenti**); il nome della funzione è sempre seguito dalle parentesi tonde, entro le quali si impostano gli argomenti (spesso ci sono dei valori di default)

```
sqrt(x = 9) # radice quadrata dell'argomento x
[1] 3
seq(from = 1, to = 5) # sequenza numerica dal valore 'from' al valore 'to'
[1] 1 2 3 4 5
```


Tipi (classi) di oggetti

Logical (logico)

```
x <- TRUE
x <- T
class(x)
[1] "logical"
```

Numeric (numeri)

```
x <- 1.4
class(x)
[1] "numeric"
```

Integer (numeri interi)

```
as.integer(x)
[1] 1
```

Character (stringa di testo)

```
x <- "Mi   piace R"
x
[1] "Mi   piace R"
```

Vector (vettore): serie di valori con la stessa classe (es. numeric) combinati con la funzione `c()` (combine)

```
x <- c(1, 10.5, 3, 2)
x + 1
[1] 2.0 11.5 4.0 3.0
sqrt(x)
[1] 1.000000 3.240370 1.732051 1.414214
y <- c("mi","piace", "R")
```

Matrix (matrice): tabella `nrow * ncol`

```
x <- matrix(1:12, nrow = 3, ncol = 4)
rownames(x) <- y # nomi di riga
x
```

	[,1]	[,2]	[,3]	[,4]
mi	1	4	7	10
piace	2	5	8	11
R	3	6	9	12

Get started

R objects

R workspace

Graphics

LM

LMER & SEM

Risorse

oooooooooooooooo

●oooooooooooooooo

oooooooooooooooo

oooooooooooooooo

oooooooooooooooo

o

oooo

R objects

Tipi (classi) di oggetti: vector

Un vettore (*vector*) è una **sequenza di elementi dello stesso tipo** (classe), che può essere creata con la funzione `c()` (*combine*) o con altre funzioni.

```
x <- c(1, 10.5, 3, 2) # creo vettore numerico
y <- 1:10 # un altro vettore numerico
(z <- rep(c(TRUE, FALSE), each = 2)) # vettore logico
[1] TRUE TRUE FALSE FALSE
as.character(x) # converte la classe del vettore da numeric a character
[1] "1" "10.5" "3" "2"
as.numeric(z) # da logical a numeric (FALSE = 0, TRUE = 1)
[1] 1 1 0 0
```

Applicando un'operazione o una funzione al vettore, questa viene applicata a tutti i suoi elementi.

```
y*2 # moltiplica tutti i valori per 2
[1] 2 4 6 8 10 12 14 16 18 20
round(sqrt(y), 2) # rad. quadrata dei valori di y, arrotondati a 2 decimali
[1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Tipi (classi) di oggetti: vector

Alcune funzioni restituiscono un unico valore a partire da un vettore di valori.

```
length(y) # numero di elementi nel vettore
[1] 10
```

Ad esempio, per calcolare delle **statistiche descrittive su vettori numerici**:

```
sum(y) # somma gli elementi nel vettore
[1] 55

max(y) # valore massimo
[1] 10

mean(y) # media
[1] 5.5

median(y) # mediana
[1] 5.5

var(y) # varianza
[1] 9.166667

sd(y) # deviazione standard
[1] 3.02765
```

Tipi (classi) di oggetti: vector

Le **parentesi quadre** [] permettono di **selezionare uno o più elementi** del vettore.

```
pippo32 <- c("uno", "due", "tre", "quattro", "cinque") # vettore di caratteri
pippo32[3] # seleziono il terzo valore di pippo32
[1] "tre"
pippo32[3:4] # terzo e quarto valore del vettore pippo32
[1] "tre"      "quattro"
pippo32[c(4, 2)] # quarto e secondo valore (non scrivere pippo32[4,3] !)
[1] "quattro" "due"
```

Ad esempio, si possono selezionare gli **elementi che rispettano certe condizioni**, usando gli operatori logici e relazionali:

```
y[y <= 3 | y > 8] # valori di y minori o uguali a 3 o maggiori di 8
[1] 1 2 3 9 10
pippo32[pippo32 != "due"] # valori di pippo32 diversi da "due"
[1] "uno"      "tre"      "quattro" "cinque"
pippo32[substr(pippo32, 2, 2) == "u"] # valori con lettera "u" in 2a posizione
[1] "due"      "quattro"
```

Tipi (classi) di oggetti: vector

La funzione `which()` restituisce la **posizione** degli elementi per cui è vera una condizione specificata.

```
substr(pippo32, 2, 2) == "u" # testa l'equivalenza per ogni elemento
[1] FALSE TRUE FALSE TRUE FALSE
which(substr(pippo32, 2, 2) == "u") # elementi con equivalenza TRUE
[1] 2 4
pippo32[substr(pippo32,2,2)=="u"] == pippo32[which(substr(pippo32,2,2)=="u")]
[1] TRUE TRUE
```

Per **cambiare** uno o più elementi di un vettore, usa il simbolo `<-`

```
pippo32[1] <- "UNO!"
```

Per **eliminare** uno o più elementi da un vettore, usa il simbolo `-`

```
pippo32[-c(2, 4)]
[1] "UNO!" "tre" "cinque"
```

Hands on: **Esercizio 1.5** (Moodle > ADcom2122 > Materiale Didattico > Materiale esercizi)

Tipi (classi) di oggetti: factor

Un fattore (*factor*) è un tipo speciale di vettore usato in R per lavorare con le **variabili categoriali** (nominali o ordinali). I valori possibili assunti dal fattore sono chiamati **livelli** (*levels*), di default ordinati in ordine crescente (numerico o alfabetico).

```
as.factor(pippo32) # da character a factor
[1] UNO!    due     tre     quattro cinque
Levels: cinque due quattro tre UNO!

# summary() mostra un sommario della variabile
summary(pippo32)

      Length      Class      Mode
      5 character character

# per i fattori, mostra la freq. di ogni livello
summary(as.factor(pippo32)) # equivale a table()

cinque    due quattro    tre    UNO!
      1      1      1      1      1
```

```
(y <- rep(c(2,4,6),3))
[1] 2 4 6 2 4 6 2 4 6

as.factor(y) # da numeric a factor
[1] 2 4 6 2 4 6 2 4 6
Levels: 2 4 6

summary(y)

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2      2      4      4      6      6

summary(as.factor(y))

2 4 6
3 3 3
```

Tipi (classi) di oggetti: factor

Creare un fattore con la funzione `factor()`:

```
factor(x = c("C",rep("A",3),c("B","A","C")))

[1] C A A B A C
Levels: A B C

(x <- factor(x = c("C","A","B","A"), # vettore
             levels = c("C","A","B"))) # livelli

[1] C A B A
Levels: C A B

levels(x) # levels() stampa i nomi dei livelli

[1] "C" "A" "B"

factor(x, levels=c("B","A","C")) # cambia ordine

[1] C A B A
Levels: B A C

levels(x) <- c("Uno","Due","Tre") # cambia nomi
x

[1] Uno Due Tre Due
Levels: Uno Due Tre
```

Quando si lavora con **variabili ordinali**, è possibile specificare l'ordine dei livelli impostando l'argomento `ordered = TRUE`. Viene così generato un **fattore ordinato**, sempre seguendo l'ordine definito da `levels`.

```
# fattore non ordinato (default)

factor(x = c("Maria","Mauro","Teresa","Carlo"))

[1] Maria  Mauro  Teresa  Carlo

Levels: Carlo Maria Mauro Teresa

# fattore ordinato (default: ordine alfabetico)

factor(x = c("Maria","Mauro","Teresa","Carlo"),
       ordered = TRUE)

[1] Maria  Mauro  Teresa  Carlo

Levels: Carlo < Maria < Mauro < Teresa
```

Hands on: **Esercizio 1.8**

Tipi (classi) di oggetti: matrix

Una matrice (*matrix*) è una **struttura bidimensionale** ($nrow \times ncol$) di **elementi dello stesso tipo**, che può essere creata con la funzione:

```
matrix(data, nrow = , ncol = , byrow = FALSE)
```

```
(x <- matrix(1:12, nrow = 3, ncol = 4))

      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

matrix(1:12, nrow = 3, ncol = 4, byrow = TRUE)

      [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12

matrix(c("Mar", "Mau", "Ter", "Car"), nrow = 2)

      [,1] [,2]
[1,] "Mar" "Ter"
[2,] "Mau" "Car"
```

Per **selezionare uno o più elementi** di una matrice, usamo ancora una volta le **parentesi quadre**, ma questa volta con la sintassi `nome_matrice[num_riga, num_colonna]`

```
x[1,2] # prima riga, seconda colonna
[1] 4

x[2,1] # seconda riga, prima colonna
[1] 2

x[1:3,2] # righe 1-3, seconda colonna
[1] 4 5 6

x[1,] # prima riga, tutte le colonne
[1] 1 4 7 10

x[,2] # seconda colonna, tutte le righe
[1] 4 5 6
```

Tipi (classi) di oggetti: matrix

```
x
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Unire due matrici:

```
cbind(x,matrix(rep(3,6),nrow=3)) # per colonna
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    7   10    3    3
[2,]    2    5    8   11    3    3
[3,]    3    6    9   12    3    3

rbind(x,matrix(rep(3,4),ncol=4)) # per riga
      [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]    3    3    3    3
```

Nomi e numero di righe e colonne:

```
rownames(x) <- c("a","b","c") # nomi righe
colnames(x) <- 1:4 # nomi colonne

x
      1 2 3 4
a 1 4 7 10
b 2 5 8 11
c 3 6 9 12

c(nrow(x),ncol(x)) # numero righe e col. = dim(x)
[1] 3 4

t(x) # matrice trasposta (inverte righe e col.)
      a b c
1 1 2 3
2 4 5 6
3 7 8 9
4 10 11 12
```

Hands on: **Esercizio 1.6**

Strutture di dati: dataframe

Un dataframe è una **struttura bidimensionale di elementi di diverso tipo**

(es. numeric, character e factor), che può essere creata con la funzione:

```
data.frame(nome_variabile1 = c(...), nome_variabile2 = c(...), ...)
```

```
(x <- data.frame(Num = 1:4,
                 Char = c("a","b","c","d"),
                 Logi = rep(c(TRUE,FALSE),2)))
```

```
Num Char Logi
```

```
1  1    a TRUE
```

```
2  2    b FALSE
```

```
3  3    c TRUE
```

```
4  4    d FALSE
```

```
str(x) # struttura del dataframe
```

```
'data.frame':  4 obs. of  3 variables:
```

```
$ Num : int  1 2 3 4
```

```
$ Char: chr  "a" "b" "c" "d"
```

```
$ Logi: logi  TRUE FALSE TRUE FALSE
```

Mentre il comando `str(nome_df)` restituisce la struttura di un dataframe, `summary(nome_df)` stampa un sommario per ogni variabile.

```
summary(x)
```

	Num	Char	Logi
Min.	:1.00	Length:4	Mode :logical
1st Qu.:	1.75	Class :character	FALSE:2
Median :	2.50	Mode :character	TRUE :2
Mean	:2.50		
3rd Qu.:	3.25		
Max.	:4.00		

Strutture di dati: dataframe

La **manipolazione di un dataframe** è molto simile a quella già vista per le matrici.

Selezione elementi e unione di due dataframe:

```
x[2, 2:3] # seconda riga, colonne 2 e 3
```

	Char	Logi
2	b	FALSE

```
cbind(x,data.frame(new=4:1)) # unione per colonna
```

	Num	Char	Logi	new
1	1	a	TRUE	4
2	2	b	FALSE	3
3	3	c	TRUE	2
4	4	d	FALSE	1

```
rbind(x[1:3,],data.frame(Num=10, # per riga
                          Char="z",Logi=FALSE))
```

	Num	Char	Logi
1	1	a	TRUE
2	2	b	FALSE
3	3	c	TRUE
4	10	z	FALSE

Nomi e numero di righe e colonne:

```
rownames(x) # default = 1:nrow(x)
```

```
[1] "1" "2" "3" "4"
```

```
colnames(x)[2] # nome colonna 2
```

```
[1] "Char"
```

```
nrow(x)
```

```
[1] 4
```

```
ncol(x)
```

```
[1] 3
```

Trasporre un dataframe:

```
t(x)
```

	[,1]	[,2]	[,3]	[,4]
Num	"1"	"2"	"3"	"4"
Char	"a"	"b"	"c"	"d"
Logi	"TRUE"	"FALSE"	"TRUE"	"FALSE"

Strutture di dati: dataframe



Per selezionare una colonna (vettore) di un dataframe, si può usare il simbolo \$, con la sintassi `nome_dataframe$nome_colonna`:

```
x$Char # seleziono colonna Char
[1] "a" "b" "c" "d"

x$Char[2] # secondo elemento della colonna Char
[1] "b"

x$Char[2] == x[2,2] # due comandi equivalenti
[1] TRUE

x$Char <- NULL # elimino colonna Char

x[x$Num < 3,] # seleziono casi con Num < 3

  Num  Logi
1    1  TRUE
2    2 FALSE

# stesso risultato con subset(x, Num < 3)
```

Modo alternativo di selezionare le colonne:

```
x[, "Logi"]

[1] TRUE FALSE TRUE FALSE

x[1:2, c("Num", "Logi")]

  Num  Logi
1    1  TRUE
2    2 FALSE
```

“Testa” e “coda” di un dataframe:

```
head(x, n = 2) # prime due righe

  Num  Logi
1    1  TRUE
2    2 FALSE

tail(x, 1) # ultima riga

  Num  Logi
4    4 FALSE
```

Hands on: **Esercizio 1.9**

Strutture di dati: list

Una lista (*list*) è una **raccolta di oggetti** che possono avere **diversa classe** (es. vector, matrice e dataframe) e **diversa lunghezza** (al contrario di matrici e dataframe). Si tratta della struttura più complessa eversatile di R, e si crea con la funzione

```
list(nome_oggetto1 = ..., nome_oggetto2 = ..., ...)
```

```
x <- list(Num = 1:4,
          Matr = matrix(1:12, nrow=3),
          df = x,
          lst = list(1:3,2:3))

str(x) # struttura del dataframe

List of 4

 $ Num : int [1:4] 1 2 3 4
 $ Matr: int [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
 $ df  :'data.frame':  4 obs. of  2 variables:
 ..$ Num : int [1:4] 1 2 3 4
 ..$ Logi: logi [1:4] TRUE FALSE TRUE FALSE
 $ lst :List of 2
 ..$ : int [1:3] 1 2 3
 ..$ : int [1:2] 2 3
```

Per **selezionare gli elementi** di una lista, usiamo sempre le **parentesi quadre**

```
x[1] # singole = crea sotto-lista
$Num
[1] 1 2 3 4

class(x[1])
[1] "list"

x[[1]] # doppie = estrae l'oggetto
[1] 1 2 3 4

class(x[[1]])
[1] "integer"

x[[3]][2,1]
[1] 2
```

R workspace

Funzioni e pacchetti

Molte cose in R si fanno usando delle funzioni, composte dai seguenti elementi: **nome**, **parentesi tonde**, **argomenti** (nomeArgomento = valoreArgomento oppure senza nome, in base alla posizione di default), valore restituito (*value*)

```
sqrt(x = c(1,2,3))
[1] 1.000000 1.414214 1.732051

sqrt(c(1,2,3))
[1] 1.000000 1.414214 1.732051
```

R Help system: Per conoscere i dettagli (argomenti) di qualsiasi funzione, basta inserire il simbolo ? seguito dal nome della funzione

```
?sqrt
```

R packages: Per ottenere funzioni aggiuntive rispetto a quelle dei pacchetti di base, è necessario installare e aprire il relativo pacchetto (**package**)

```
install.packages("nome_pacchetto") # installare un pacchetto

library(nome_pacchetto) # aprire un pacchetto

nome_pacchetto::nome_funzione() # usare funzione senza aprire il pacchetto
```


Oggetti, funzioni e workspace

Quando assegniamo un valore ad una variabile, questa viene registrata nel **workspace**: il posto che contiene tutti gli oggetti e le funzioni definiti dall'utente (sezione Environment di RStudio).

La funzione `ls()` stampa i nomi di tutti gli oggetti e le funzioni presenti nel workspace, mentre la funzione `rm()` rimuove l'oggetto specificato tra parentesi:

```
x <- 1 # assegno valore a x
y <- 2 # assegno valore a y
ls() # mostra tutti gli oggetti nel workspace
[1] "x" "y"
rm(y) # rimuove l'oggetto y
ls()
[1] "x"
```

Combinando le due funzioni, il comando `rm(list=ls())` svuota il workspace, eliminando tutti gli oggetti e le funzioni (molto utile **all'inizio di ogni script!**)

```
rm(list = ls())
```

Oggetti e funzioni già inclusi in Base R

Alcuni pacchetti di base (oggetti e funzioni) sono **già installati in R**. Questi non richiedono l'apertura di nessun pacchetto e non compaiono nel workspace.

```
rm(list=ls()) # svuoto il workspace
head(sleep,4) # dataset sleep dal pacchetto datasets
  extra group ID
1   0.7      1  1
2  -1.6      1  2
3  -0.2      1  3
4  -1.2      1  4
mean(sleep$extra) # funzione mean() dal pacchetto base
[1] 1.54
letters[2] # costanti dal pacchetto base
[1] "b"
ls() # il workspace è vuoto!
character(0)
```

Trucchetto con RStudio: ogni volta che inizi un nuovo progetto (es. analisi tesi, report progetto), crea un nuovo **R project** (.Rproj) dal menu **File > New R Project**, selezionando una directory esistente o creandone una nuova. Così quella sarà già impostata come la WD per tutti i file associati al progetto.

Caricare ed esportare un dataset

Il primo passo di ogni analisi dei dati con R è quello di caricare una struttura di dati (registrati con un certo metodo e salvati con un certo formato) nel workspace.

Per caricare un dataset che si trova nella working directory, è necessario usare una specifica funzione in base al formato del file.

```
# file .RData
load(file = "data/questionarioStudenti.RData") # import
save(qs, file = "data/questionarioStudenti.RData") # export
```

R può importare ed esportare dati salvati con molti formati diversi, alcuni dei quali richiedono l'installazione di pacchetti aggiuntivi.

```
# file .CSV (comma separated values)
qs <- read.csv(file = "data/questionarioStudenti.csv") # import
write.csv(x = qs, "data/questionarioStudenti.csv", row.names = FALSE) # export

# file .SAV (da SPSS)
library(foreign)
qs <- read.spss("data/questionarioStudenti.sav", to.data.frame=TRUE)
```

Caricare ed esportare un dataset

Un modo veloce (ma poco riproducibile!) per caricare un file su R è usare la funzione `file.choose()` al posto del nome del file. Questo consente di caricare un file senza nemmeno dover impostare la WD.

```
qs <- read.csv(file = file.choose())
```

Se hai dubbi su quale funzione usare per leggere un file, puoi usare il menu punta-e-clicca di RStudio: **File > Import Dataset**

Hands on:

- Apri o crea un file `.xlsx` sul tuo PC, salvalo in formato `.csv` (comma separated values) e importalo su R
- Prova ad importare direttamente il file `.xlsx` (se hai dubbi, cerca su Google “*how to read xlsx file with R*”)
- In entrambi i casi, osserva la classe e la struttura dell’oggetto importato

Hands on: Questionario incontri facoltativi

1. Scarica i file `questionarioStudenti.RData` e `questionarioStudenti.csv` da Github:
<https://github.com/Luca-Menghini/eseRcitazioni/tree/main/data> (Premere su **File** > **Raw** > **Download** o tasto **dx** > **Save as**), salva il file in una cartella e imposta quella cartella come working directory.
2. Importa entrambi i file su RStudio e confronta i due oggetti. Di che classe sono? E le variabili di che classe sono?
3. Usa la funzione `describe()` del pacchetto `psych` (forse va prima installato!) per calcolare le statistiche descrittive della variabile `numVar` e usa la funzione `hist()` per visualizzare il grafico di densità ad istogrammi (prova a cambiare il valore dell'argomento `breaks`).
4. Usa la funzione `table()` per produrre la tabella di frequenza della variabile Q02 (“*Quale giorno preferiresti per gli incontri facoltativi?*”)
5. Ora fai la stessa cosa, ma considerando solo chi ha risposto “Sì” alla domanda Q01 (“*Parteciperai a tutti gli incontri?*”)
6. Ora incrocia le frequenze delle variabili Q02 e Q03 (“*Riusciresti a partecipare anche se fossero nel giorno che NON hai scelto?*”)

Statistiche descrittive (univariate)

```
x <- c(1,1,1,2,8,9) # creo vettore numerico
```

```
c(mean(x),median(x)) # media & mediana
```

```
[1] 3.666667 1.500000
```

```
as.numeric(which.max(table(x))) # moda
```

```
[1] 1
```

```
c(var(x),sd(x)) # varianza & dev. standard
```

```
[1] 14.266667 3.777124
```

```
quantile(x,probs=0.90) # 90° percentile
```

```
90%
```

```
8.5
```

```
quantile(x,probs=c(0.25,0.50,0.75,1)) # quartili
```

```
25% 50% 75% 100%
```

```
1.0 1.5 6.5 9.0
```

```
round(rank(x)/length(x),2) # ranghi percentili
```

```
[1] 0.33 0.33 0.33 0.67 0.83 1.00
```

```
table(x) # frequenze assolute
```

```
x
```

```
1 2 8 9
```

```
3 1 1 1
```

```
round(table(x)/length(x),2) # freq relative
```

```
x
```

```
1 2 8 9
```

```
0.50 0.17 0.17 0.17
```

```
cumsum(x) # somma cumulata
```

```
[1] 1 2 3 5 13 22
```

```
cumsum(table(x)) # freq cumulate assolute
```

```
1 2 8 9
```

```
3 4 5 6
```

```
round(cumsum(table(x)/length(x),2) # freq cumulate relative
```

```
1 2 8 9
```

```
0.50 0.67 0.83 1.00
```

Statistiche descrittive (bivariate)

```
y <- -x - 1 # valori inversamente prop. a x
z <- round(rnorm(n=length(x)),1) # valori a caso
(df <- data.frame(x,y,z)) # nuovo dataframe
```

```
  x   y   z
1 1  -2 -0.5
2 1  -2 -0.6
3 1  -2 -1.4
4 2  -3  0.4
5 8  -9  0.0
6 9 -10  0.3
```

Correlazione e covarianza

```
cov(x,y) # covarianza tra x e y
[1] -14.26667

cor(x,y) # correlazione
[1] -1
```

,

Quando ho più di due variabili:

```
cov(df) # matrice di covarianza
```

```
      x      y      z
x 14.26667 -14.26667  1.520
y -14.26667  14.26667 -1.520
z  1.52000  -1.52000  0.456
```

```
cor(df) # matrice di correlazione
```

```
      x      y      z
x 1.0000000 -1.0000000  0.5959364
y -1.0000000  1.0000000 -0.5959364
z  0.5959364 -0.5959364  1.0000000
```


Esercizi

- **Esercizio 1.10***
- **Esercizi 1.11-1.13***
- **Esercizio 1.26****

*I dataset sono inclusi nella cartella `dati.esercizi.zip` su Moodle

(`ADcom2122 > Materiale didattico > MateRiale eseRcizi`), ma si possono scaricare anche da github.com/Luca-Menghini/eseRcitazioni/tree/main/data

**Per scaricare il pacchetto `ADati` del prof. Pastore, esegui i seguenti comandi collegandoti ad una rete wifi:

```
install.packages("devtools") # installo pacchetto devtools
library(devtools) # apro pacchetto devtools
install_github("https://github.com/masspastore/ADati") # installo ADati da github
library(ADati) # apro pacchetto ADati
```

Get started

oooooooooooooooo

R objects

oooooooooooooooo

R workspace

oooooooooooooo

Graphics

●oooooooooooooooo

LM

oooooooooooooooo

LMER & SEM

o

Risorse

oooo

Graphics

Graphics in R: Principali funzioni

Funzioni di alto livello (stanno in piedi da sole)

```
# funzione base (dipende dalla classe dell'oggetto)
plot() # scatter plot (grafico a dispersione)
```

```
# distribuzione
```

```
boxplot() # boxplot (diagramma a scatola e a baffi)
qqnorm() # quantile-quantile plot
```

```
# frequenze
```

```
barplot() # grafico a barre (variabili categoriali)
hist() # istogramma (variabili continue)
pie() # pie chart
```

```
# interazioni
```

```
interaction.plot()
```

Funzioni di basso livello (aggiungono elementi alle prime)

```
points() # aggiunge punti
```

```
lines() # aggiunge linee
```

```
text() # aggiunge del testo
```

```
# retta di regressione lineare
```

```
abline(a = ..., b = ...)
```

```
# elementi più complessi
```

```
rect() # aggiunge rettangoli
```

```
polygon() # aggiunge poligoni
```

```
# altre caratteristiche del grafico
```

```
axis() # per modificare gli assi
```

```
legend() # per aggiungere una legenda
```

Parametri grafici

Esegui il comando `?par` per vedere tutti i parametri grafici modificabili.

Alcuni possono essere impostati usando gli argomenti delle funzioni grafiche:

```
# dimensioni

cex = 2 # grandezza testo e simboli (moltiplicatore)

lwd = 0.5 # spessore linee


# colore e forma

col = "red" # (vedi ?colors)

lty = 2 # tipo di linee (1=solid, 2=dashed, ...)

pch = 19 # forma dei punti (vedi ?points)


# titoli

main = "Titolo del grafico"

xlab = "Titolo asse x"
```

Altri devono essere impostati all'interno della funzione `par()`, che va eseguita prima di generare il grafico:

```
# margini (bottom, left, top, right)

mai # dimensione margini (in ince)

mar # dimensione margini (in linee di testo)


# struttura griglie di grafici

mfrow=c(nrow, ncol)

# es. due grafici uno di fianco all'altro

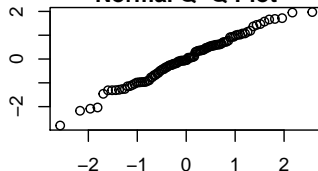
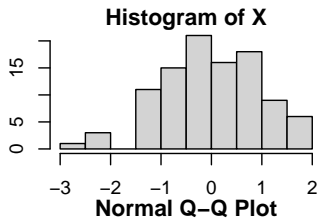
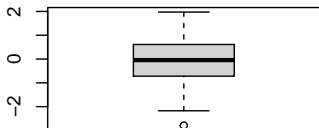
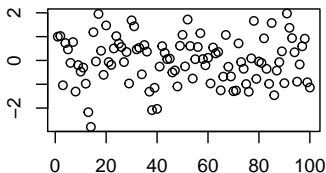
mfrow=c(2,1)

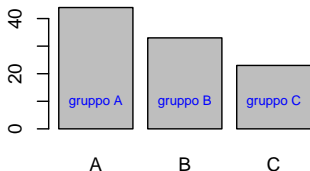
# es. due grafici uno sotto l'altro

mfrow=c(1,2)
```

Distribuzioni univariate: variabili continue

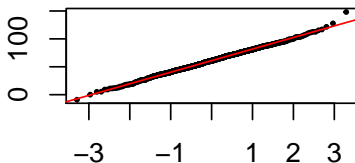
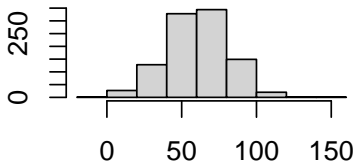
```
X <- rnorm(n=100, mean=0, sd=1) # genero valori casuali dalla distribuzione normale standard
par(mfrow = c(2,2), mai = c(0.3,0.5,0.2,0.5)) # voglio 4 grafici in una sola finestra
plot(X) # scatter plot: grafico a dispersione (variabilità)
hist(X) # istogramma: distribuzione di frequenze per classi
boxplot(X) # box plot: diagramma a scatola (1° e 3° quartile) e a baffi (+/- 1.5 IQR)
qqnorm(X) # Q-Q plot: distribuzione cumulata di X vs. distribuzione cumulata normale)
```



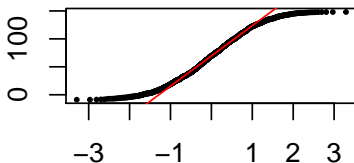
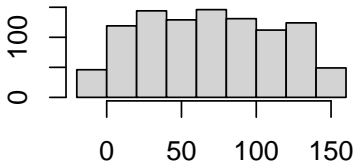


Valutare la normalità di una distribuzione

```
X <- rnorm(n = 1000, mean = 60, sd = 20) # norm
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4))
hist(X,main="",xlab="",ylab="") # istogramma
qqnorm(X,cex=0.5,main="",pch=20) # Q-Q plot
qqline(X,col="red") # aggiungo linea distr. norm.
```



```
Y <- runif(n=1000,min=min(X),max=max(X)) # unif
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4))
hist(Y,main="",xlab="",ylab="") # istogramma
qqnorm(Y,cex=0.5,main="",pch=20) # Q-Q plot
qqline(Y,col="red") # aggiungo linea distr. norm.
```



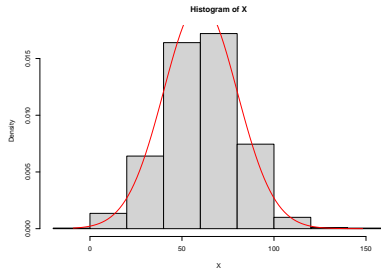
Valutare la normalità di una distribuzione

```
# Distribuzione empirica
```

```
hist(X, freq = FALSE)
```

```
# Distribuzione teorica (attesa)
```

```
curve(dnorm(x, mean = 60, sd = 20),  
      from = min(X), to = max(X),  
      col = "red", lwd = 2,  
      add = TRUE)
```

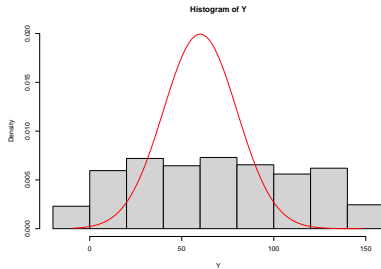


```
# Distribuzione empirica
```

```
hist(Y, freq = FALSE, ylim = c(0,0.02))
```

```
# Distribuzione teorica (attesa)
```

```
curve(dnorm(x, mean = 60, sd = 20),  
      from = min(Y), to = max(Y),  
      col = "red", lwd = 2,  
      add = TRUE)
```



Valutare la normalità di una distribuzione

```
library(moments) # asimmetria e curtosi
c(skewness(X), kurtosis(X)) # (buono se ~ 0)
[1] 0.01828131 3.21099011
# test Kolmogorov-Smirnov (H0: X è normale)
ks.test(X, y="pnorm", mean=mean(X), sd=sd(X))
```

One-sample Kolmogorov-Smirnov test

```
data: X
D = 0.015544, p-value = 0.9691
alternative hypothesis: two-sided
# test Shapiro-Wilk (H0: X è normale)
shapiro.test(X)
```

Shapiro-Wilk normality test

```
data: X
W = 0.99852, p-value = 0.5663
```

```
# asimmetria e curtosi
c(skewness(Y), kurtosis(Y)) # (buono se ~ 0)
[1] 0.03571885 1.86771627
# test Kolmogorov-Smirnov (H0: Y è normale)
ks.test(Y, y="pnorm", mean=mean(Y), sd=sd(Y))
```

One-sample Kolmogorov-Smirnov test

```
data: Y
D = 0.062657, p-value = 0.0007781
alternative hypothesis: two-sided
# test Shapiro-Wilk (H0: Y è normale)
shapiro.test(Y)
```

Shapiro-Wilk normality test

```
data: Y
W = 0.96151, p-value = 1.409e-15
```

Distribuzioni bivariate: variabili continue

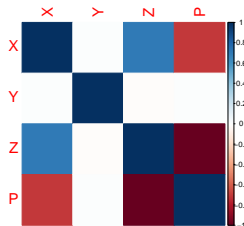
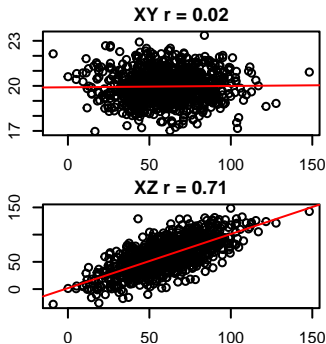
```
Y <- rnorm(1000, mean=20, sd=1) # Y (indipendente da X)
Z <- X + rnorm(1000, sd = 20) # Z (proporzionale a X)
# Scatter plot
par(mfrow=c(2,1),mai=rep(0.3,4),mar=rep(1.8,4),cex=.5)
plot(X, Y, main = paste("XY r =",round(cor(X, Y),2)))
abline(lm(Y~X),col="red") # retta di regr. lineare
plot(X, Z, main = paste("XZ r =",round(cor(X, Z),2)))
abline(lm(Z~X),col="red") # tilde (~) con Alt + 1-2-6
```

```
df <- data.frame(X, Y, Z, P = -Z)
```

```
# Correlation plot
```

```
library(corrplot)
```

```
corrplot(cor(df),method="color",
          tl.cex=2,cl.cex=1,
          mar = c(0,0,0,15))
```



Distribuzioni bivariate: variabili categoriali

creo data.frame con sesso (M/F) e livello di educazione (primaria, medie, superiori, università)

```
df <- data.frame(sex = sample(x = rep(c("F","M"),50), size = 100),
  edu = sample(x = rep(c("prim","med","sup","uni"),25), size = 100))
```

```
table(df) # frequenze incrociate
```

```
edu
```

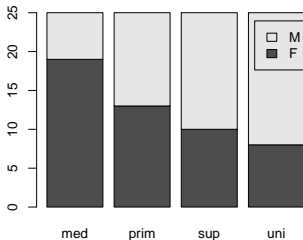
```
sex med prim sup uni
```

```
F 19 13 10 8
```

```
M 6 12 15 17
```

```
par(mai=rep(0.4,4))
```

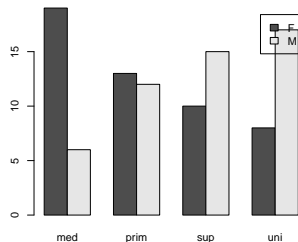
```
barplot(table(df),legend.text = TRUE) # bar plot
```



versione con 'gruppi affiancati'

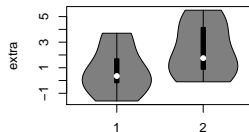
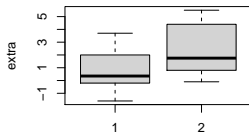
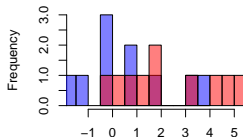
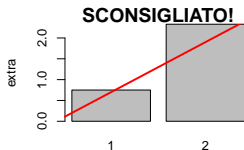
```
par(mai=rep(0.4,4))
```

```
barplot(table(df),legend.text = TRUE,
  beside = TRUE)
```



Distribuzioni bivariate: continue & categoriali

```
par(mfrow = c(2,2), mai = c(0.5,1,0.2,0.5)) # nota: uso il dataset sleep (già incluso in R base)
barplot(tapply(sleep$extra, sleep$group, FUN=mean), # grafico a barre e le medie (SCONSIGLIATO!)
        main="SCONSIGLIATO!", cex.main=1.5, ylab = "extra"); abline(a=0,b=1,col="red",lwd=2)
hist(sleep[sleep$group==1,"extra"], breaks=10, col = rgb(0,0,1,alpha=0.5),
      xlim = c(min(sleep$extra), max(sleep$extra)), xlab="", main="") # istogramma per gruppo
hist(sleep[sleep$group==2,"extra"], breaks=10, col = rgb(1,0,0,alpha=0.5), add = TRUE)
boxplot(extra ~ group, data = sleep) # box plot per gruppo
library(vioplplot); vioplplot(extra ~ group, data = sleep) # grafico a violino: densità per gruppo
```



Salvare un grafico

```
# 1. apro nuovo file con formato PNG
png(filename = "grafico.png", # titolo del grafico
     width = 350, height = 350) # larghezza e altezza

# 2. creo il grafico
plot(sleep$group, sleep$extra,
     xlab = "group", ylab = "extra", col = "#2E9FDF")

# 3. chiudo il file
dev.off()
pdf
2
```

Formati alternativi (vedi ?png)

```
png() # Portable Network Graphics

bpm() # bitmap

jpeg() # Joint Photographic Experts Group

tiff() # Tagged Image File Format
```

Esercizi

Introduzione a R:

1.22 (distribuzioni di probabilità)

1.26 (dataset, descrittive e grafici)*

Teoria dei campioni

2.2 (popolazione e distr. campionarie)

2.8 (variabili casuali e distr. campionarie)

2.14 (dataset, campioni e probabilità)**

Inferenza statistica

3.4 (dataset, grafici e test statistici)**

3.11 (grafici, test e potenza statistica)

*Per scaricare il pacchetto ADati del prof. Pastore:

```
install.packages("devtools")
library(devtools)
install_github("https://github.com/masspastore/ADati")
library(ADati)
```

*Per caricare un dataset dal pacchetto ADati:

```
data(gambling, # nome del dataset
     package = "ADati")
```

**I dataset sono inclusi nella cartella

dati.esercizi.zip su Moodle (ADcom2122 > Materiale didattico > Materiale esercizi), ma si possono scaricare anche da github.com/Luca-Menghini/eseRcitazioni/tree/main/data

Get started

R objects

R workspace

Graphics

LM

LMER & SEM

Risorse

oooooooooooooooo

oooooooooooooooo

oooooooooooooo

oooooooooooooooo

●ooooooooooooo

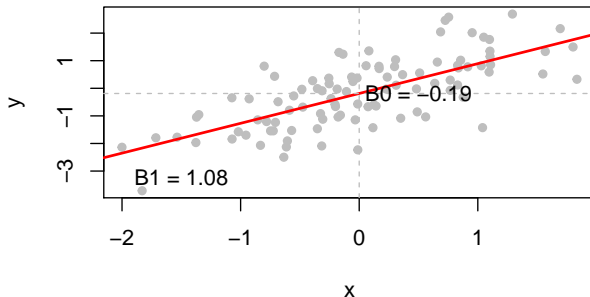
o

oooo

LM

Regressione lineare

- La **regressione** mira a stabilire se tra due variabili vi sia una relazione funzionale asimmetrica, in particolare a quantificare la misura in cui una variabile (chiamata **predittore**) agisca su un'altra (chiamata **dipendente** o **risposta**).
- La **regressione lineare** consiste nel determinare il legame tra due variabili attraverso una funzione lineare del tipo $Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$
- Tale funzione può essere rappresentata graficamente con una **retta** in cui β_0 rappresenta l'**intercetta** (valore assunto da Y quando $X = 0$) e β_1 il **coefficiente angolare**.



Specificare un modello di regressione lineare

Uso il dataset `gambling` dal pacchetto `ADati`. Voglio specificare dei modelli che predicano `frequency` (frequenza di gioco) sulla base di `gender` (genere) e `risk` (percezione del rischio).

```
data(gambling, package = "ADati")
```

In R, i modelli di regressione lineare si specificano con la funzione `lm()`, impostando gli argomenti `formula` (sintassi: `dipendente ~ predittore1 + predittore2 + ...`, dove il simbolo “~” è la **tilde**, che su windows si fa con `Alt + 126` del tastierino numerico) e `data` (il dataframe che include le variabili scritte nella formula)

- **Modello nullo:** il punteggio di `frequency` è predetto **soltanto dall'intercetta** b_0 , ovvero dal valore medio di `frequency` nel campione.

```
m0 <- lm(formula = frequency ~ 1, data = gambling)
```

```
coefficients(m0) # coefficiente stimato dal modello (intercetta)
```

```
(Intercept)
```

```
10.55997
```

```
mean(gambling$frequency) # media della variabile frequency
```

```
[1] 10.55997
```

Specificare un modello di regressione lineare

- **Modello di regressione lineare semplice:** il punteggio di **frequency** è predetto dall'intercetta b_0 (il valore atteso di **frequency** quando **gender** è uguale a "f") e dal coefficiente b_1 , che esprime la differenza 'media' $m - f$.

```
m1 <- lm(formula = frequency ~ gender, data = gambling)
```

```
coefficients(m1)
```

```
(Intercept)      genderm
```

```
  9.336685    3.042012
```

```
tapply(gambling$frequency, gambling$gender, mean) # media di frequency per f e m
```

```
      f      m
```

```
 9.336685 12.378697
```

(Intercept)	risk	genderm	risk:genderm
16.9206907	-1.0994674	5.9744077	-0.6087177

Valutare gli assunti del modello lineare

1. Indipendenza delle osservazioni: tutte le coppie di errori ϵ_i e ϵ_j sono indipendenti per ogni $i \neq j$. Questa non richiede particolari procedure, basta riflettere: le osservazioni nel mio dataset sono tra loro indipendenti?

2. Indipendenza dei predittori dall'errore: i residui (componente d'errore ϵ_i) del modello non sono associati ai valori dei predittori **gender** e **risk**.

Ma cosa sono i residui?

```
gambling$RESIDUI <- residuals(m3) # residui del modello con funzione residuals()
gambling$ATTESI <- fitted(m3) # valori di predetti dal modello con funzione fitted()
head(gambling[,c("frequency", "ATTESI", "RESIDUI")], 5) # RESIDUI = OSSERVATI - ATTESI
```

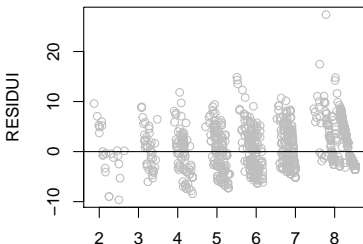
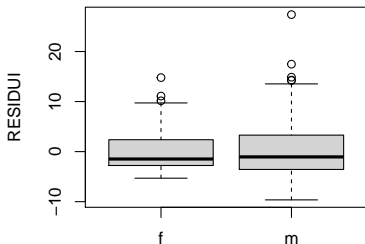
	frequency	ATTESI	RESIDUI
6	4.78	7.553228	-2.7732283
8	36.98	9.605418	27.3745820
10	4.78	7.553228	-2.7732283
18	11.16	11.335396	-0.1753962
19	5.62	7.828095	-2.2080952

Valutare gli assunti del modello lineare

2. Indipendenza dei predittori dall'errore: i residui (componente d'errore ϵ_i) del modello non sono associati ai valori dei predittori **gender** X_{1i} e **risk** X_{2i} . In questo caso, non sembrano esserci differenze di genere, né una relazione sostanziale con **risk**.

3. Varianza costante (omogeneità delle varianze): la varianza dei residui (gli errori ϵ) è costante per qualunque valore di X . Qui vediamo che la varianza dei residui è leggermente (ma non esageratamente) maggiore nel gruppo dei maschi.

```
par(mfrow=c(1,2),mar=c(5,4,0,2)+0.1)
plot(RESIDUI ~ gender, data=gambling)
plot(RESIDUI ~ risk, data=gambling, col="gray")
abline(lm(RESIDUI ~ risk, data=gambling))
```



Valutare gli assunti del modello lineare

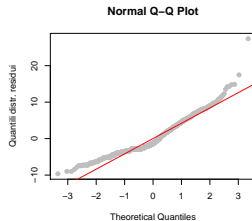
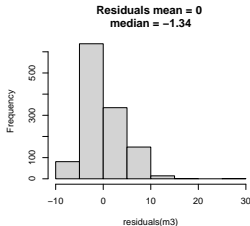
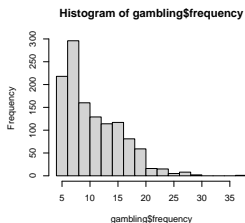
4. Linearità: il valore atteso dell'errore ϵ per un dato valore di X è zero.

In pratica, i residui devono avere media zero (come in questo caso).

5. Normalità: i residui (gli errori ϵ) sono distribuiti normalmente.

In questo caso non è un bellissimo fit, ma possiamo accettarlo...

```
par(mfrow=c(1,3))
hist(gambling$frequency) # se i residui non sono normali neanche la Y lo è
hist(residuals(m3),main=paste("Residuals mean =",round(mean(residuals(m3)),2),
                             "\nmedian =",round(median(residuals(m3)),2)))
qqnorm(residuals(m3), col="gray", pch=19, ylab = "Quantili distr. residui")
qqline(residuals(m3), col="red")
```



Confrontare diversi modelli

Likelihood ratio test: test del rapporto di verosomiglianza (probabilità dei dati osservati in funzione dei parametri stimati)

```
library(lmtest)
lrtest(m0,m1,m2,m3)
Likelihood ratio test
```

Model 1: frequency ~ 1

Model 2: frequency ~ gender

Model 3: frequency ~ risk + gender

Model 4: frequency ~ risk * gender

	#Df	LogLik	Df	Chisq	Pr(>Chisq)
1	2	-3687.3			
2	3	-3629.4	1	115.843	< 2.2e-16 ***
3	4	-3480.6	1	297.484	< 2.2e-16 ***
4	5	-3472.6	1	16.149	5.853e-05 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Criteri di informazione: penalizzano i modelli in base a specifici criteri (es. bassa verosomiglianza e parsimonia)

Akaike information criterion (AIC)

AIC(m0,m1,m2,m3)\$AIC # + basso meglio è
[1] 7378.599 7264.756 6969.272 6955.122

Akaike weights: da 0 (-) a 1 (+)

```
library(MuMIn)
```

```
Weights(AIC(m0,m1,m2,m3)) # Aw
```

```
model weights
```

[1] 0.000 0.000 0.001 0.999

Inferenza sui coefficienti di regressione

Seguendo l'approccio NHST, è possibile effettuare un test per valutare la significatività statistica dei coefficienti di regressione. In R, i test vengono effettuati in automatico quando si specifica il modello.

```
summary(m3)$coefficients # comando summary() mostra un sommario del modello
```

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	16.9206907	0.7416563	22.814733	3.337032e-96
risk	-1.0994674	0.1051711	-10.454087	1.484438e-24
genderm	5.9744077	1.0163116	5.878519	5.339754e-09
risk:genderm	-0.6087177	0.1512209	-4.025355	6.040343e-05

Coefficiente di determinazione R^2 :

```
# il modello spiega complessivamente il 29% della varianza in frequency
```

```
summary(m3)$r.squared
```

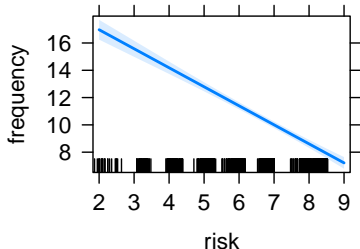
```
[1] 0.2965382
```


Inferenza sui coefficienti di regressione

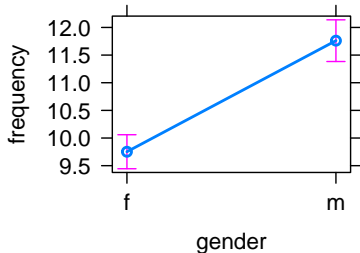
Per visualizzare gli effetti stimati, puoi usare il pacchetto `effects`

```
library(effects)  
plot(allEffects(m2))
```

risk effect plot



gender effect plot



Esercizi

Regressione lineare semplice

4.1 (v.casuali, grafici, correlazioni e test)

4.6 (dataset, grafici e coefficienti)**

4.14 (grafici, modelli e assunti)*

Regressione lineare multipla

5.1 (grafici, modelli, assunti ed effetti)*

5.4 (grafici, modelli e interazioni)**

*Per scaricare il pacchetto AData del prof. Pastore:

```
install.packages("devtools")
library(devtools)
install_github("https://github.com/masspastore/ADati")
library(ADati)
```

*Per caricare un dataset dal pacchetto AData:

```
data(gambling, # nome del dataset
     package = "ADati")
```

****I dataset sono inclusi nella cartella**

dati.esercizi.zip su Moodle (ADcom2122 > Materiale didattico > Materiale esercizi), ma si possono scaricare anche da github.com/LucaMenghini/esercitazioni/tree/main/data

LMER & SEM

Risorse

- R Core Team. *The R Manuals*. Manuali in formato pdf:
<https://cran.r-project.org/manuals.html> (in particolare *An Introduction to R (with many examples, R Data Import/Export)*)
- UniGlasgow - PsyTheacR. Corso interattivo analisi dati con R:
<https://psyteachr.github.io/>
- Dalgaard, P. (2008). *Introductory statistics with R*. New York: Springer.

Risorse: argomenti specifici

Graphics:

- Videolezioni di approfondimento sul pacchetto `ggplot2`: disponibili su Moodle
<https://elearning.unipd.it/scuolapsicologia/mod/page/view.php?id=150311>
- Murrell (2011) R Graphics (II edition). New York: Chapman and Hall/CRC. Esempi con codice disponibili al sito
<https://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html>

Contatti



dott. Luca Menghini, Ph.D.

Assegnista di ricerca, Dipartimento di Psicologia,
Università degli Studi di Bologna

luca.menghini3@unibo.it

[Linkedin](#) | [ResearchGate](#) | [GitHub](#) | [Twitter](#)