Testing Psicologico

Lezione 1

Filippo Gambarota

@Università di Padova

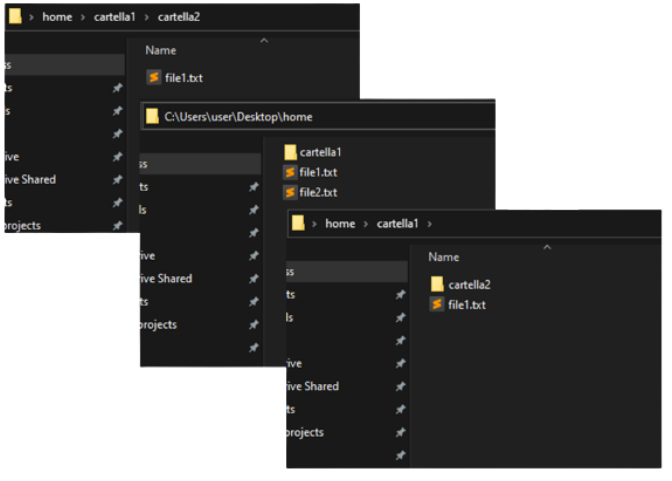
24/10/2022

Organizzazione R ed R Studio

Working Directory e Percorsi

Il nostro computer è composto da file e cartelle organizzati in modo **gerarchico** tra loro

```
📦home
 ┣ 📂cartella1
 ┃ ┣ 📂cartella2
 ┃ ┃ ┗ 📜file1.txt
 ┃ ┗ 📜file1.txt
 ┣ 📜file1.txt
 ┗ 📜file2.txt
```



Working Directory e Percorsi

Nel momento in cui usiamo **R**, lui si colloca automaticamente in un dato percorso:

```
getwd()
```

```
## [1] "/home/filippogambarota/Documents/teaching-projects/didattica-testing-psicologico/slides/lezione1"
```

Noi possiamo modificare il collocamento di R usando il comando `setwd()`

```
setwd("cartella/sub-cartella/...")
```

Importare una funzione

In R tutto (vettore, dataframe, lista, etc.) è un oggetto, anche le funzioni. Per caricare una funzione salvata in un file `.R` possiamo usare il comando `source(file)`. Il file verrà caricato e tutto il codice lanciato. Se qualche oggetto o funzione è stato creato sarà disponibile globalmente:

```
source("../../R/rsummary.R")
ls()
```

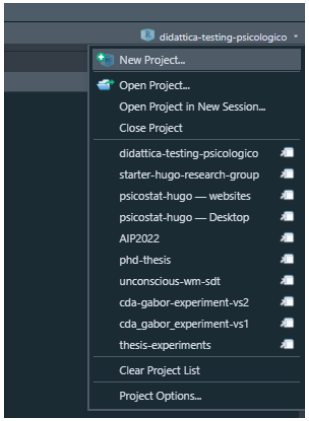
```
## [1] "all_rmd" "html"    "i"
```

Extra: R Projects

Extra: R Projects

Gli R Projects sono una funzionalità di R Studio e permettono di impostare automaticamente la **working directory** nella cartella dove è contenuto il file `*.Rproj`. In questo modo, ogni volta che R Studio viene aperto caricando un R Project, tutti i percorsi sono relativi alla *root* del progetto.

- Video Tutorial sui percorsi e R Projects



Strutture dati

Strutture dati

Le strutture dati sono modalità tramite cui un linguaggio di programmazione **organizza** tipologia e **struttura** dei vari tipi possibili di dato. Il vettore e la matrice sono delle strutture dati.

Strutture dati

Le strutture dati sono modalità tramite cui un linguaggio di programmazione **organizza** tipologia e **struttura** dei vari tipi possibili di dato. Il vettore e la matrice sono delle strutture dati.

Aspetti principali di una struttura dati:

- presenza di **vincoli** (e.g., il vettore può essere solo numerico o di stringhe)
- presenza di **metodi** (i.e., funzioni) per **accedere**, **estrarre** e **modificare** i dati

Strutture dati

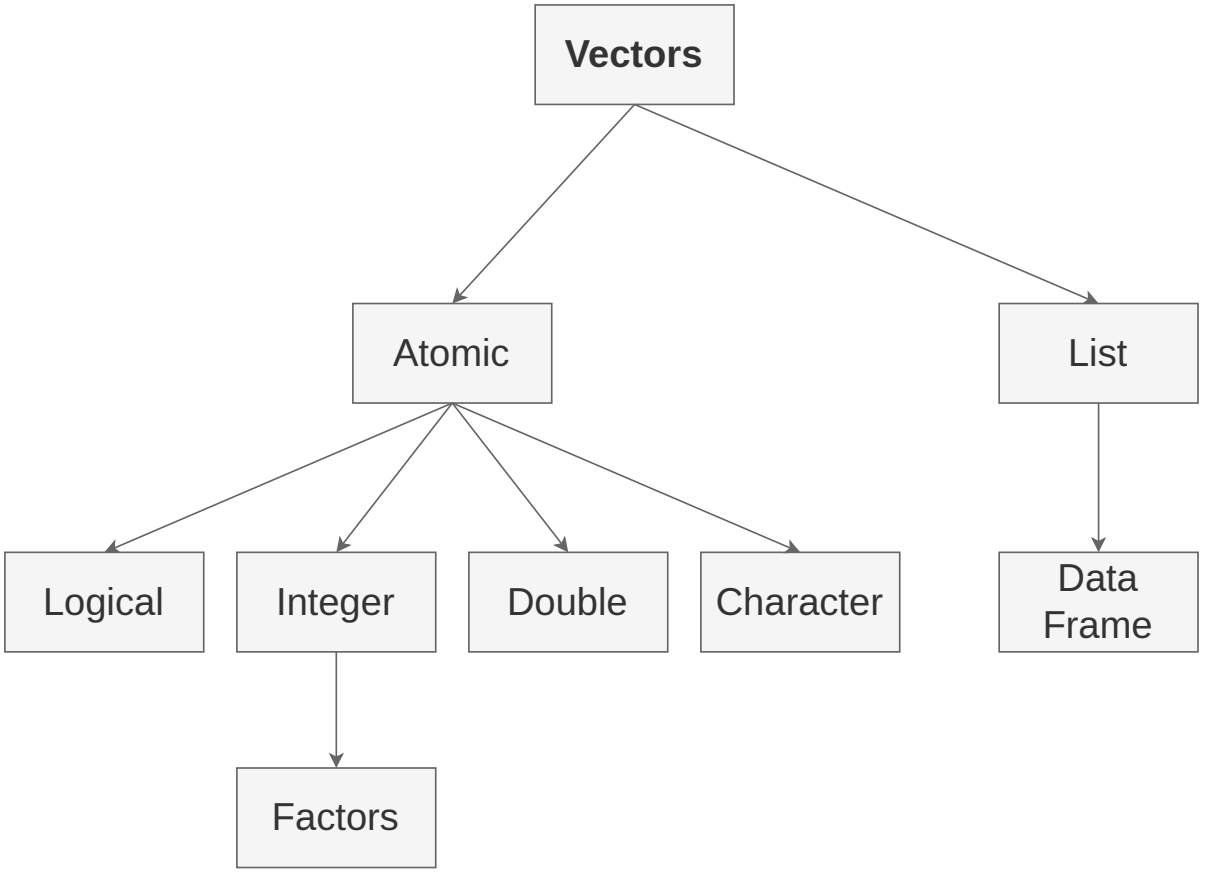
Esiste una struttura dati che abbiamo sicuramente usato. Quale? 🤔

Strutture dati

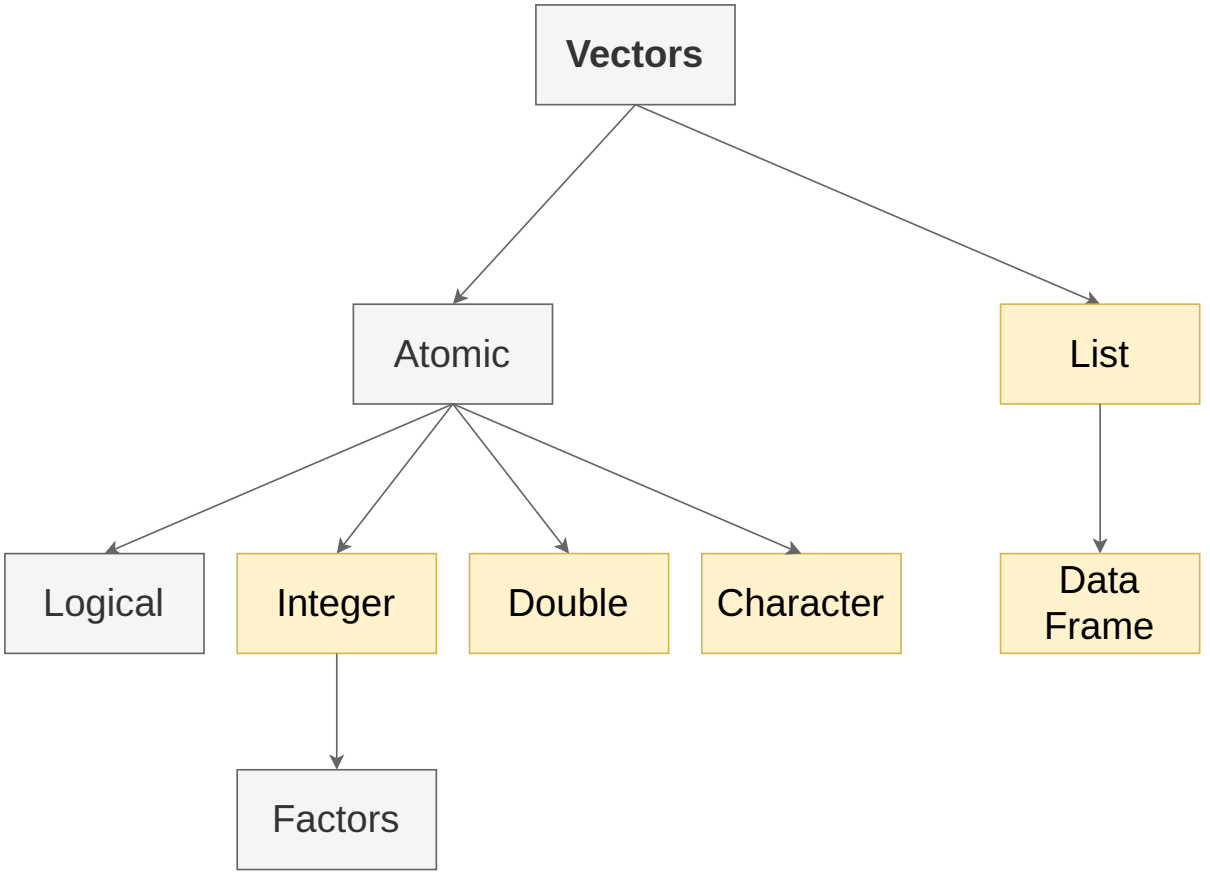
Esiste una struttura dati che abbiamo sicuramente usato. Quale? 🤔

	A	B	C	D	E
1	id	nome	professione		
2	1	Filippo	studente		
3	2	Andrea	lavoratore		
4	3	Francesco	studente		
5	4	Franco	studente		
6	5	Luca	lavoratore		
7	6	Filippo	studente		
8	7	Andrea	studente		
9	8	Francesco	lavoratore		
10	9	Franco	studente		
11	10	Luca	studente		
12	11	Filippo	lavoratore		
13	12	Andrea	studente		
14	13	Francesco	studente		
15	14	Franco	lavoratore		
16	15	Luca	studente		
17					
18					

Strutture dati in R



Strutture dati in R



Vettori

Dubbi/Domande? 🤔

Esercizi

1.Create il seguente **vettore**:

$V = (2, 3.5, 5, 6.5, 8, 9.5)$

2.Create il seguente **vettore di caratteri**:

$V = (x, x, x, y, y, z, z, z, z, z, z)$

3.Create la seguente **matrice**:

$$\begin{bmatrix} 3 & 5 & 11 \\ 2 & 99 & 4 \\ 2 & 55 & 100 \\ 1 & 0 & 3 \end{bmatrix}$$

4.Data la matrice 3:

- accedere al numero di dimensioni
- accedere alla terza colonna
- accedere agli elementi $x_1 = [3, 1]$ e $x_2 = [4, 2]$

Soluzioni

```
seq(2, 10, 1.5)
```

```
## [1] 2.0 3.5 5.0 6.5 8.0 9.5
```

```
rep(c("x", "y", "z"), c(3, 2, 6))
```

```
##  [1] "x" "x" "x" "y" "y" "z" "z" "z" "z" "z" "z"
```

```
mat <- matrix(data = c(3,5,11,2,99,4,2,55,100,1,0,3),
              nrow = 4,
              ncol = 3,
              byrow = TRUE)
dim(mat)
```

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

```
mat[3, 1]
```

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

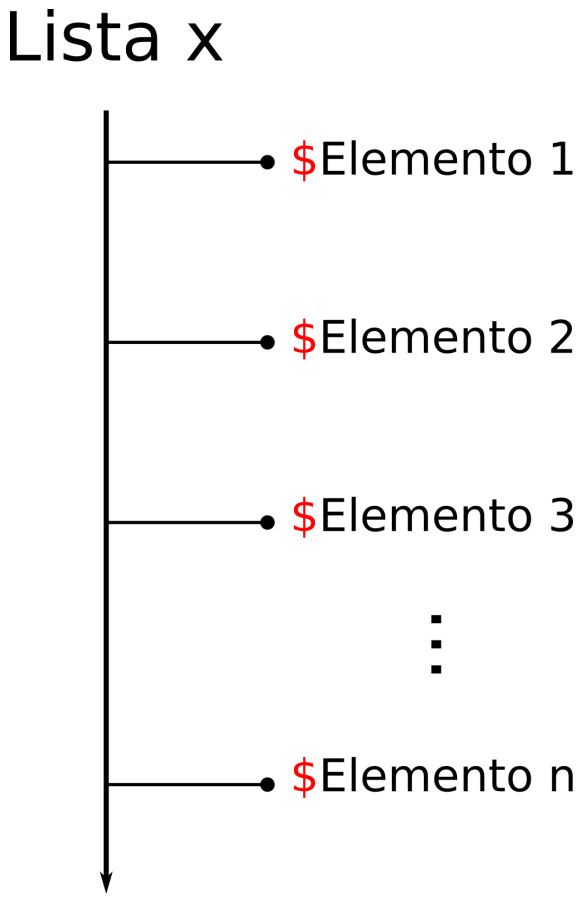
```
mat[5, 2]
```

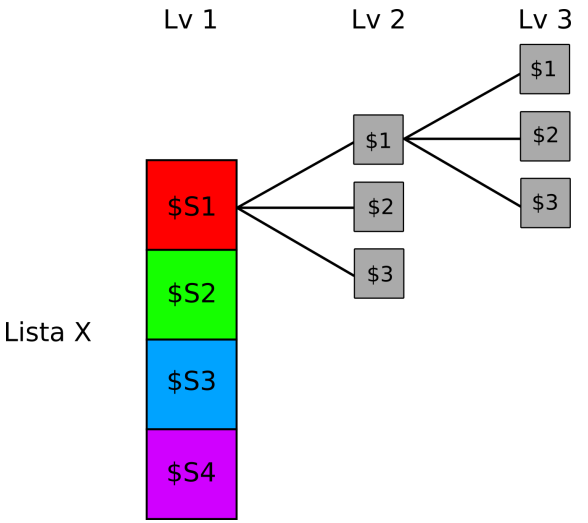
```
## Error in mat[5, 2]: subscript out of bounds
```

Liste

Liste

In R la lista è la struttura dati più versatile (meno strutturata 😄) e utile.





Liste

```
list(elemento1, elemento2, elemento3) # lista normale
list(nome1 = elemento2, nome2 = elemento2, nome3 = elemento3) # lista named
```

```
el1 <- runif(100)
el2 <- rep(letters[1:10], 3)
el3 <- iris
my_list <- list(vec1 = el1, vec2 = el2, data = el3)

names(my_list)
```

```
## [1] "vec1" "vec2" "data"
```

```
length(my_list)
```

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

```
str(my_list)
```

```
## List of 3
##  $ vec1: num [1:100] 0.332 0.631 0.207 0.634 0.999 ...
##  $ vec2: chr [1:30] "a" "b" "c" "d" ...
##  $ data:'data.frame':    150 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Liste

Accediamo/modifichiamo gli elementi della lista:

```
my_list$vec1 # con il dollaro + nome
```

```
##  [1] 0.3321608 0.6306689 0.2070681 0.6339305 0.9991772 0.3501283 0.9017213 0.3235232 0.4325908
## [10] 0.7022708
##  [ reached getOption("max.print") -- omitted 90 entries ]
```

```
my_list[1] # con la parentesi quadra
```

```
## $vec1
##  [1] 0.3321608 0.6306689 0.2070681 0.6339305 0.9991772 0.3501283 0.9017213 0.3235232 0.4325908
## [10] 0.7022708
##  [ reached getOption("max.print") -- omitted 90 entries ]
```

```
my_list[[1]] # con la doppia parentesi quadra
```

```
##  [1] 0.3321608 0.6306689 0.2070681 0.6339305 0.9991772 0.3501283 0.9017213 0.3235232 0.4325908
## [10] 0.7022708
##  [ reached getOption("max.print") -- omitted 90 entries ]
```

```
my_list[[1]] <- nuovoelemento # sovrascrivo il primo elemento
my_list[[4]] <- nuovoelemento # aggiungo un elemento
my_list[[length(my_list) +  1]] <- nuovoelemento # più raffinato
my_list <- append(my_list, list(nuovoelemento)) # usando la funzione append
my_list <- c(my_list, list(nome = nuovoelemento)) # usando la funzione c
```

Esercizi

1. Create una lista (usando dei nomi che volete) che contenga
   - una sequenza di 10 numeri partendo da 3 e incrementando di 1.33
   - le lettere dell'alfabeto (vedi `letters`) ripetute tutte 2 volte
   - il dataset `iris`
   - 100 numeri campionati da una distribuzione normale standard (vedi `rnorm`) $\mu = 0$ e $\sigma = 1$
2. Accedete al secondo elemento della lista
3. Aggiungete un quinto elemento con 10 numeri campionati da una distribuzione normale con $\mu = 10$ e $\sigma = 0$
4. Sostituite il terzo elemento con un'altra lista formata da un vettore numerico con i numeri da 1 a 30 e le prime 10 lettere dell'alfabeto

Esercizi - Soluzioni

```
my_list <- list(
  sequenza = seq(3, by = 1.33, length.out = 10),
  lettere = rep(letters, 2),
  iris = iris,
  normale01 = rnorm(100, mean = 0, sd = 1)
)

my_list[[2]]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
##  [ reached getOption("max.print") -- omitted 42 entries ]
```

```
my_list$lettere # se conosco il nome
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
##  [ reached getOption("max.print") -- omitted 42 entries ]
```

```
my_list <- c(my_list, list(new_normale = rnorm(10, 10, 0)))
my_list[[3]] <- list(1:30, letters[1:10])
my_list[[3]]
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##  [ reached getOption("max.print") -- omitted 20 entries ]
##
## [[2]]
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

Dataframe

Dubbi/Domande? 🤔

Dataframe

Dataframe

- In R il dataframe è la struttura dati più utilizzata. Permette di organizzare dati, fare statistiche descrittive, fare analisi (come regressioni, t-test, etc.) e molte altre cose

Dataframe

- In R il dataframe è la struttura dati più utilizzata. Permette di organizzare dati, fare statistiche descrittive, fare analisi (come regressioni, t-test, etc.) e molte altre cose
- E' un tipo particolare di **lista** dove la lunghezza di ogni elemento è fissa (vincolo) portando ad una **struttura rettangolare**

Dataframe

- In R il dataframe è la struttura dati più utilizzata. Permette di organizzare dati, fare statistiche descrittive, fare analisi (come regressioni, t-test, etc.) e molte altre cose
- E' un tipo particolare di **lista** dove la lunghezza di ogni elemento è fissa (vincolo) portando ad una **struttura rettangolare**
- E' la *traduzione* in codice del foglio di calcolo Excel

Dataframe

Ci sono diversi dataframe già presenti in R come oggetti. Vediamo quello più semplice ovvero `iris`:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
##  [ reached 'max' / getOption("max.print") -- omitted 4 rows ]
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
class(iris)
```

```
## [1] "data.frame"
```

Dataframe

Per accedere al dataframe usiamo un mix tra funzioni per le matrici (da cui prende la struttura rettangolare) e liste (da cui prende la flessibilità del tipo di dato):

```
iris$Sepal.Length # prima colonna/elemento
```

```
##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
##  [ reached getOption("max.print") -- omitted 140 entries ]
```

```
iris[[1]] # prima colonna/elemento
```

```
##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
##  [ reached getOption("max.print") -- omitted 140 entries ]
```

```
iris[, 1] # prima colonna
```

```
##  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
##  [ reached getOption("max.print") -- omitted 140 entries ]
```

Extra: Importare dati

Importare dati

- La maggior parte delle analisi dati prevede di importare partendo da formati diversi (`xlsx`, `csv`, `sav`, `txt`, etc.) un dataset.
- Importare i dati è tutt'altro che banale e richiede una comprensione di come i vari formati codificano le informazioni fondamentali, in particolare la delimitazione dei valori
- `csv` ad esempio significa **c**omma **d**elimited **v**alues dove i valori sono delimitati da una virgola. R deve sapere il tipo di file e il delimitatore per leggere correttamente i dati

Per approfondire questo documento è una buona introduzione

Esempio

Esempio

```r
# importiamo i dati
dat <- read.csv("../../data/pazienti.csv", sep = ",", header = TRUE, fileEncoding="UTF-8-BOM")

str(dat) # struttura
```

```
## 'data.frame':    30 obs. of  6 variables:
##  $ sogg      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ regione   : chr  "Veneto" "Piemonte" "Lombardia" "Piemonte" ...
##  $ cl.sociale: chr  "Bassa" "Media" "Media" "Bassa" ...
##  $ ansia     : num  5.1 5.5 3.8 4.5 5.4 0.7 4.6 6 4.5 5.8 ...
##  $ eta       : int  58 48 21 57 39 52 42 48 53 32 ...
##  $ disturbo  : chr  "schizofrenia" "nevrosi" "schizofrenia" "nevrosi" ...
```

```r
nrow(dat) # numero di righe (osservazioni)
```

```
## [1] 30
```

```r
ncol(dat) # numero di colonne (variabili)
```

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

```r
colnames(dat) # nomi delle colonne (variabili)
```

```
## [1] "sogg"       "regione"    "cl.sociale" "ansia"      "eta"        "disturbo"
```

Esempio

Lavorare in un dataframe segue la stessa logica di un foglio excel. Possiamo **filtrare** le righe e/o colonne in funzione di determinate *condizioni*:

```
# seleziono solo i pazienti con nevrosi e tutte le colonne
dat[dat$disturbo == "nevrosi", ]
```

```
##   sogg  regione cl.sociale ansia eta disturbo
## 2    2 Piemonte      Media   5.5  48  nevrosi
##  [ reached 'max' / getOption("max.print") -- omitted 11 rows ]
```

```
# seleziono solo i pazienti con età maggiore di 30
dat[dat$eta > 30, ]
```

```
##   sogg regione cl.sociale ansia eta     disturbo
## 1    1  Veneto      Bassa   5.1  58 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 27 rows ]
```

```
# seleziono i pazienti con ansia maggiore di 3 E provenienti dal veneto
dat[dat$ansia > 3 & dat$regione == "Veneto", ]
```

```
##   sogg regione cl.sociale ansia eta     disturbo
## 1    1  Veneto      Bassa   5.1  58 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 2 rows ]
```

Esercizi

1. Importa il dataframe `pazienti_sc.csv` (attenzione al *separatore*)
2. Estrai la struttura, il numero di colonne/righe
3. Estrai le righe 1, 10, 15, e 30
4. Estrai le righe da 1 a 15 e la 1 e 4 colonna
5. Estrai le osservazioni di pazienti provenienti dalla Liguria O dal Piemonte di classe sociale Alta e disturbi NON fobici
6. Estrai le osservazioni con età compresa tra 20 e 45 anni

Soluzioni

```
dat <- read.csv("../../data/pazienti_sc.csv", sep = ";", header = TRUE, fileEncoding="UTF-8-BOM") # import

# struttura, righe e colonne
str(dat)
```

```
## 'data.frame':    30 obs. of  6 variables:
##  $ sogg      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ regione   : chr  "Veneto" "Piemonte" "Lombardia" "Piemonte" ...
##  $ cl.sociale: chr  "Bassa" "Media" "Media" "Bassa" ...
##  $ ansia     : num  5.1 5.5 3.8 4.5 5.4 0.7 4.6 6 4.5 5.8 ...
##  $ eta       : int  58 48 21 57 39 52 42 48 53 32 ...
##  $ disturbo  : chr  "schizofrenia" "nevrosi" "schizofrenia" "nevrosi" ...
```

```
nrow(dat)
```

```
## [1] 30
```

```
ncol(dat)
```

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

Soluzioni

```
dat[c(1, 10, 15, 30), ] # righe 1, 10, 15 e 30
```

```
##   sogg regione cl.sociale ansia eta     disturbo
## 1    1  Veneto      Bassa   5.1  58 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

```
dat[1:15, c(1, 4)] # righe 1:15 e colonna 1 e 4
```

```
##   sogg ansia
## 1    1   5.1
## 2    2   5.5
## 3    3   3.8
## 4    4   4.5
## 5    5   5.4
##  [ reached 'max' / getOption("max.print") -- omitted 10 rows ]
```

```
dat[(dat$regione == "Liguria" | dat$regione == "Piemonte") & dat$cl.sociale == "Alta" & dat$disturbo != "fobico", ] # pazien
```

```
##   sogg regione cl.sociale ansia eta disturbo
## 7    7 Liguria       Alta   4.6  42  nevrosi
##  [ reached 'max' / getOption("max.print") -- omitted 4 rows ]
```

```
dat[dat$eta > 20 & dat$eta < 45, ] # eta compresa tra 20 e 45
```

```
##   sogg   regione cl.sociale ansia eta     disturbo
## 3    3 Lombardia      Media   3.8  21 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 11 rows ]
```

Extra: Indicizzazione

Indicizzazione

Indicizzare una struttura dati è un'operazione fondamentale e complessa. Ma la logica sottostante è molto semplice. La sezione 10.2 del libro `Introduction2R` è un buon riferimento.

Il segreto è capire come funzionano le *operazioni logiche* ad esempio `dat$eta > 30` e come si concatenano tra loro

Indicizzazione logica

Indicizzare con la posizione è l'aspetto più semplice e intuitivo. E' possibile anche selezionare tramite valori `TRUE` e `FALSE`. L'idea è che se abbiamo un vettore di lunghezza *n* e un'altro vettore logico di lunghezza *n*, tutti gli elementi `TRUE` saranno selezionati:

```
my_vec <- 1:10
my_selection <- sample(rep(c(TRUE, FALSE), each = 5)) # random TRUE/FALSE
my_selection
```

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

```
my_vec[my_selection]
```

```
## [1] 1 3 4 5 9
```

Indicizzazione logica

Chiaramente non è pratico costruire a mano i vettori logici. Infatti possiamo usare delle *espressioni relazionali* per selezionare elementi:

```
my_vec <- 1:10
my_selection <- my_vec < 6
my_vec[my_selection]
```

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

```
my_vec[my_vec < 6] # in modo più compatto
```

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

Indicizzazione logica

Chiaramente possiamo usare **espressioni di qualsiasi complessità** perchè essenzialmente abbaimo bisogno di un vettore `TRUE/FALSE`:

```
my_vec <- 1:10
my_selection <- my_vec < 2 | my_vec > 8
my_vec[my_selection]
```

```
## [1]  1  9 10
```

```
my_vec[my_vec < 2 | my_vec > 8] # in modo più compatto
```

```
## [1]  1  9 10
```

Indicizzazione intera `which()`

La funzione `which()` è molto utile perchè restituisce la **posizione** associata ad una selezione logica:

```
my_vec <- rnorm(10)
which(my_vec < 0.5)
```

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

```
# Questo

my_vec[which(my_vec < 0.5)]
```

```
## [1] -0.47147678 -0.32360345 -1.40994804 -0.26032605  0.45108023 -0.74777312  0.04750194
## [8]  0.08075510
```

```
# e questo sono equivalenti

my_vec[my_vec < 0.5]
```

```
## [1] -0.47147678 -0.32360345 -1.40994804 -0.26032605  0.45108023 -0.74777312  0.04750194
## [8]  0.08075510
```

Indicizzazione dataframe

E' importante capire che a prescindere dalla complessità della struttura dati (vettore vs dataframe) quando selezioniamo delle righe/colonne non facciamo altro che *combinare operazioni logiche*, ottenere un vettore di `TRUE`/`FALSE` o di `interi` e con questo vettore indicare quali righe/colonne selezionare.

```
# età > 30 e regione veneto
my_sel_log <- dat$eta > 30 & dat$regione == "Veneto"
my_sel_log # vettore logico TRUE/FALSE
```

```
##  [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
##  [ reached getOption("max.print") -- omitted 20 entries ]
```

```
my_sel_int <- which(my_sel_log) # vettore di interi

dat[my_sel_log, ] # selezione logica
```

```
##   sogg regione cl.sociale ansia eta     disturbo
## 1    1  Veneto      Bassa   5.1  58 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

```
dat[my_sel_int, ] # selezione intera
```

```
##   sogg regione cl.sociale ansia eta     disturbo
## 1    1  Veneto      Bassa   5.1  58 schizofrenia
##  [ reached 'max' / getOption("max.print") -- omitted 3 rows ]
```

EDA - Exploratory Data Analysis

Nuove colonne

Possiamo aggiungere nuove informazioni (colonne) per aggiungere informazioni o modificare quelle esistenti:

```
# aggingiamo una colonna che indica alta o bassa ansia basandoci su un cut-off di 4
dat$ansia_cut <- ifelse(dat$ansia > 4, yes = "alta", no = "bassa")

# convertiamo la classe sociale in un fattore ordinato (scala ordinale)
dat$cl.sociale <- factor(dat$cl.sociale, ordered = TRUE)

str(dat)
```

```
## 'data.frame':    30 obs. of  7 variables:
##  $ sogg      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ regione   : chr  "Veneto" "Piemonte" "Lombardia" "Piemonte" ...
##  $ cl.sociale: Ord.factor w/ 3 levels "Alta"<"Bassa"<..: 2 3 3 2 2 3 1 2 3 2 ...
##  $ ansia     : num  5.1 5.5 3.8 4.5 5.4 0.7 4.6 6 4.5 5.8 ...
##  $ eta       : int  58 48 21 57 39 52 42 48 53 32 ...
##  $ disturbo  : chr  "schizofrenia" "nevrosi" "schizofrenia" "nevrosi" ...
##  $ ansia_cut : chr  "alta" "alta" "bassa" "alta" ...
```

Esplorazione

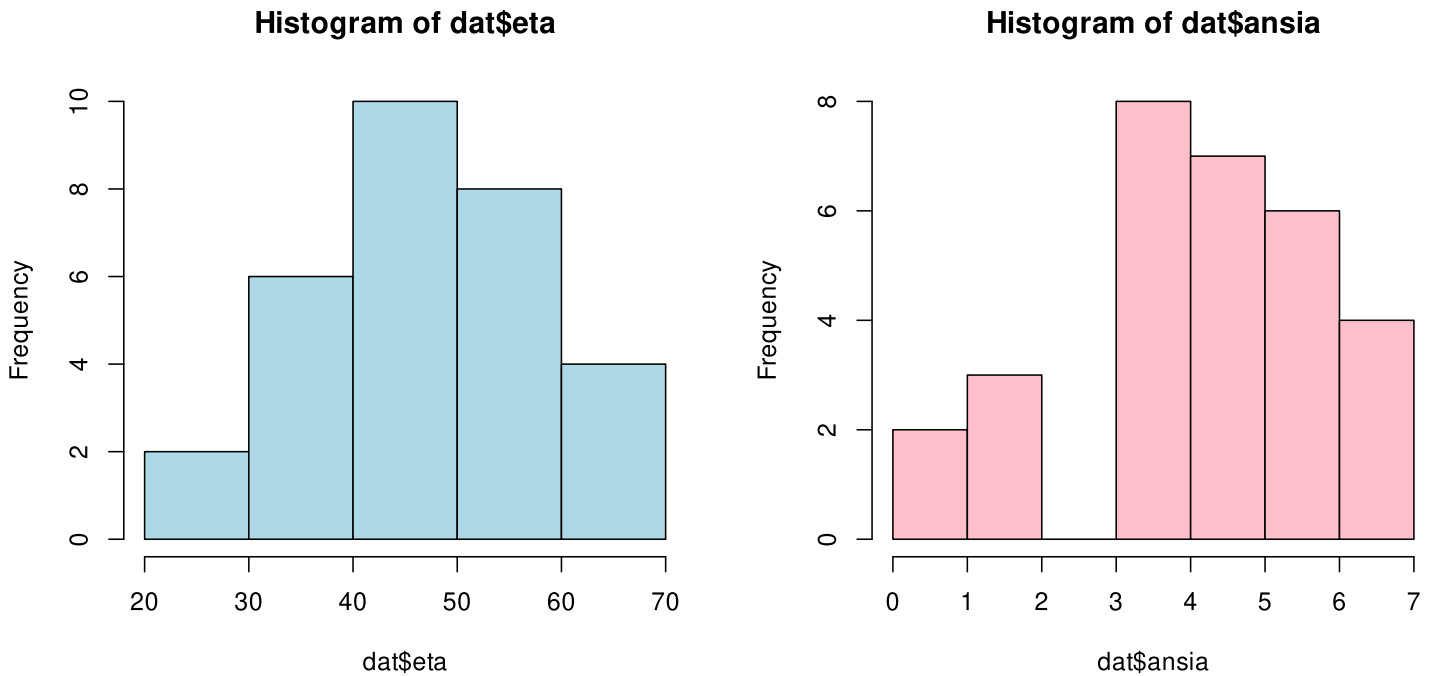
Ogni tipo di variabile è associata a determinate statistiche descrittive (e.g., *media* vs *frequenza*) e rappresentazioni grafiche (e.g., *barplot* vs *boxplot*).

- ha senso calcolare la media della variabile `disturbo`?
- ha senso calcolare le frequenze della variabile `ansia`?

Esplorazione

Facciamo un istogramma per le variabili numeriche:

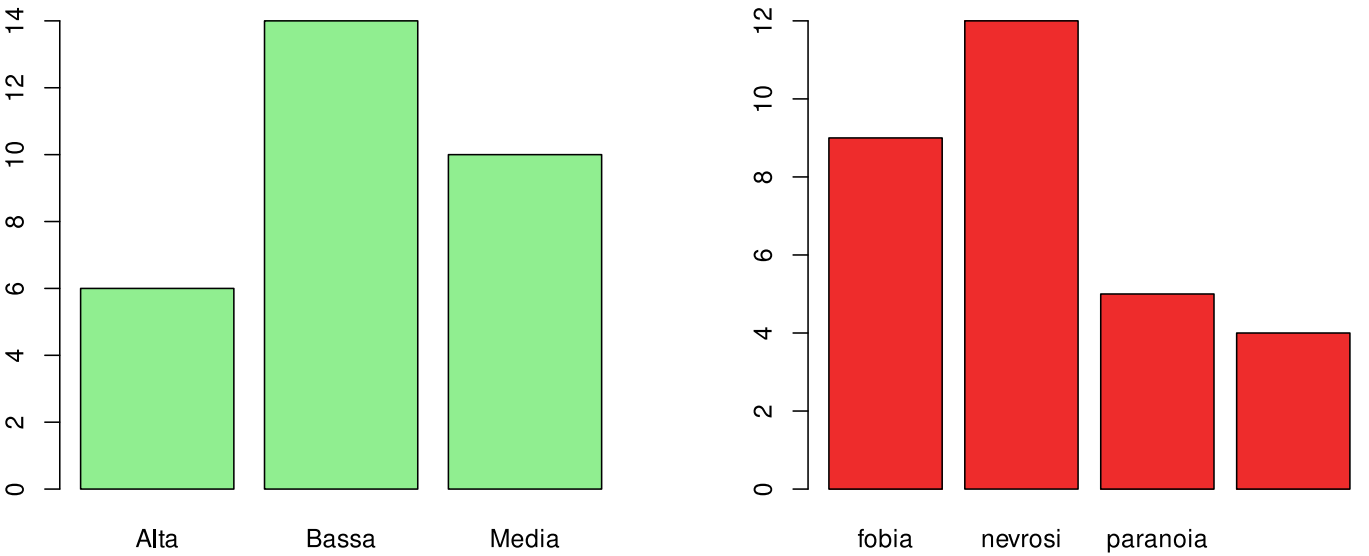
```
par(mfrow = c(1,2))
hist(dat$eta, col = "lightblue")
hist(dat$ansia, col = "pink")
```



Esplorazione

Facciamo un grafico a barre per le variabili categoriali/ordinali

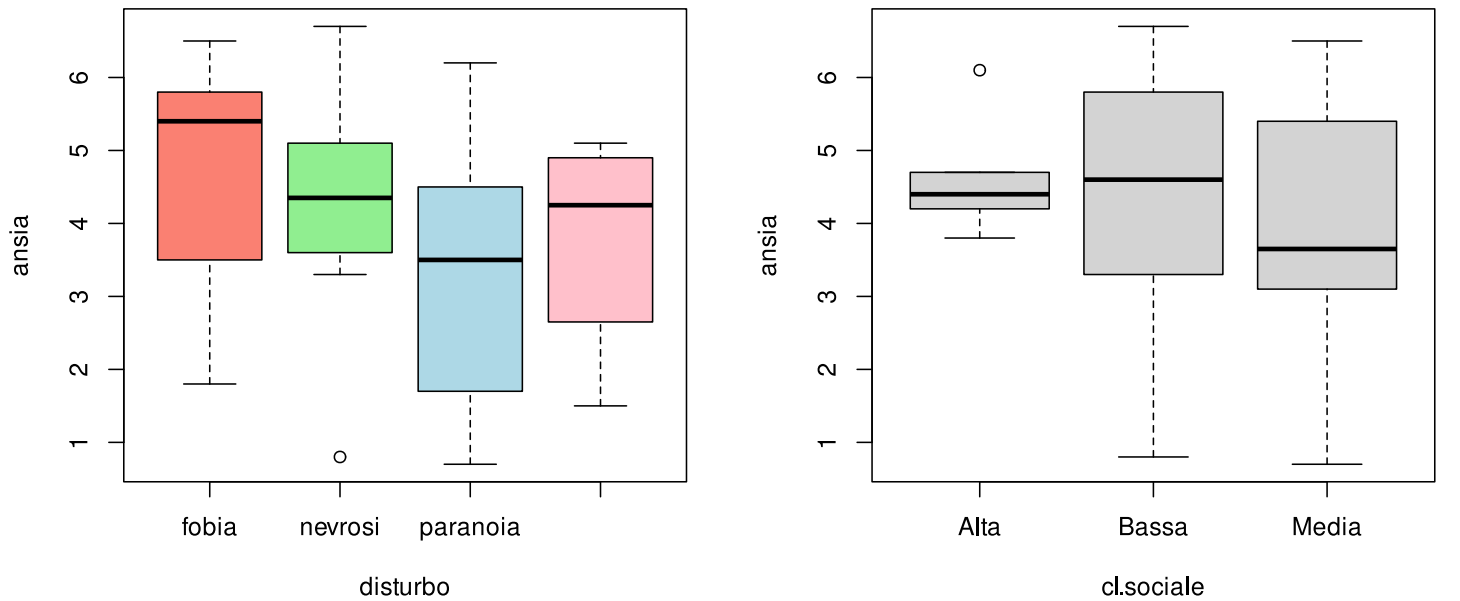
```
par(mfrow = c(1,2))
barplot(table(dat$cl.sociale), col = "lightgreen")
barplot(table(dat$disturbo), col = "firebrick2")
```



Explorazione - Grafici bi-variati

Possiamo vedere la distribuzione di una variabile numerica *in funzione* di una categoriale:

```
par(mfrow = c(1,2))
boxplot(ansia ~ disturbo, data = dat, col = c("salmon", "lightgreen", "lightblue", "pink"))
boxplot(ansia ~ cl.sociale, data = dat)
```



Esplorazione - Grafici bi-variati

Possiamo anche vedere la distribuzione di due variabili categoriali facendo un barplot ed una tabella di contingenza:

```
barplot(table(dat$cl.sociale, dat$disturbo), col = c("salmon", "lightgreen", "lightblue"))
legend("topright", legend=unique(dat$cl.sociale), pch=16, col = c("salmon", "lightgreen", "lightblue"))
```