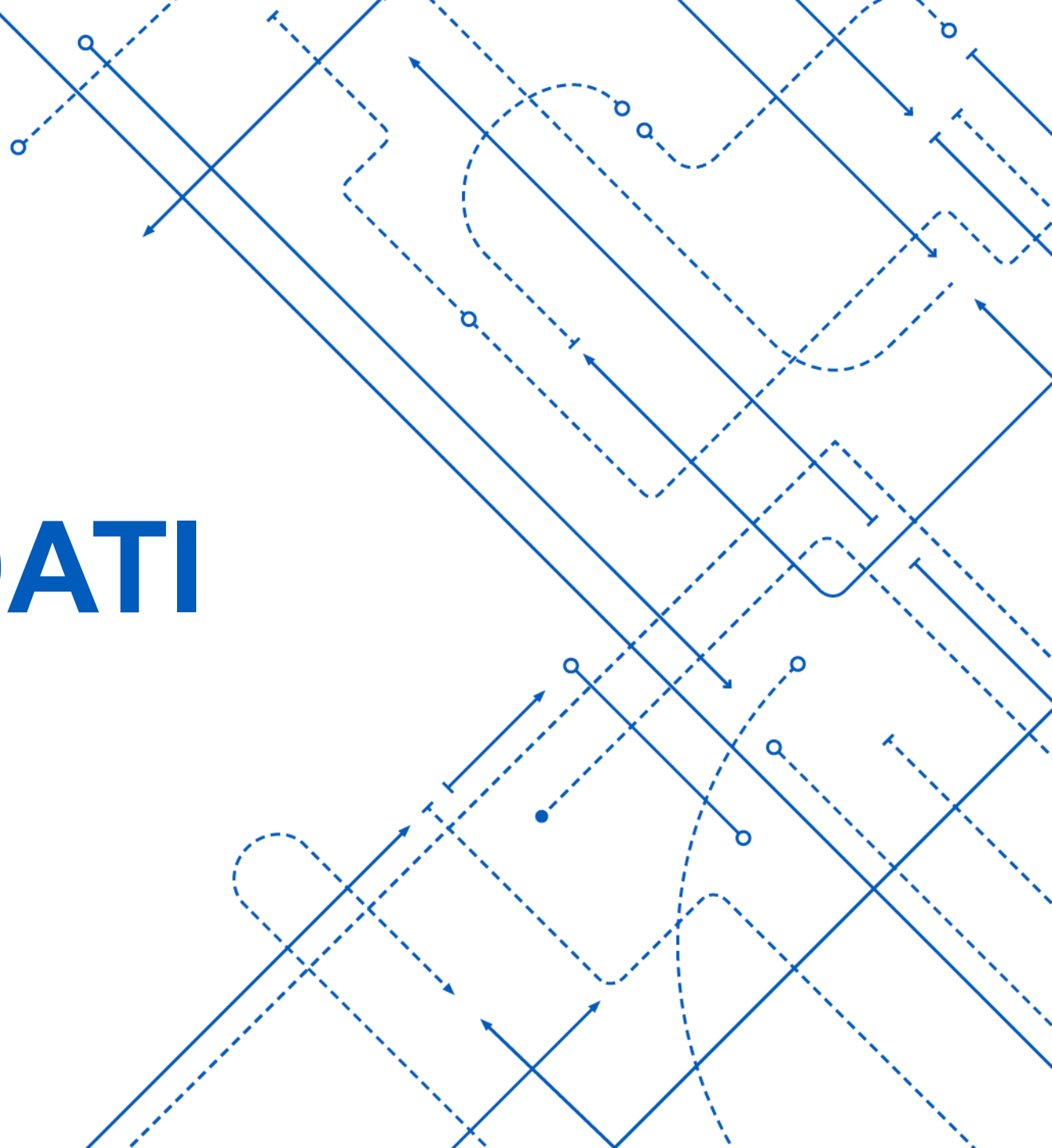
STATISTICA E ANALISI DEI DATI

Strutture Dati

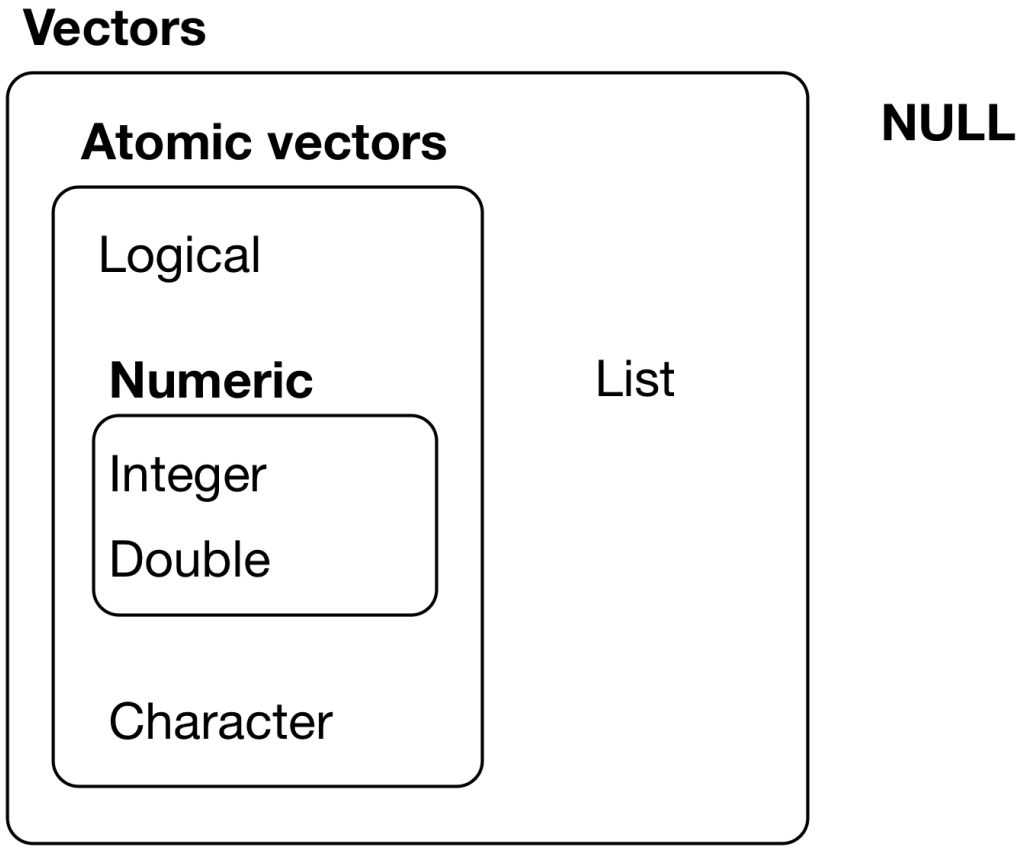
Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026



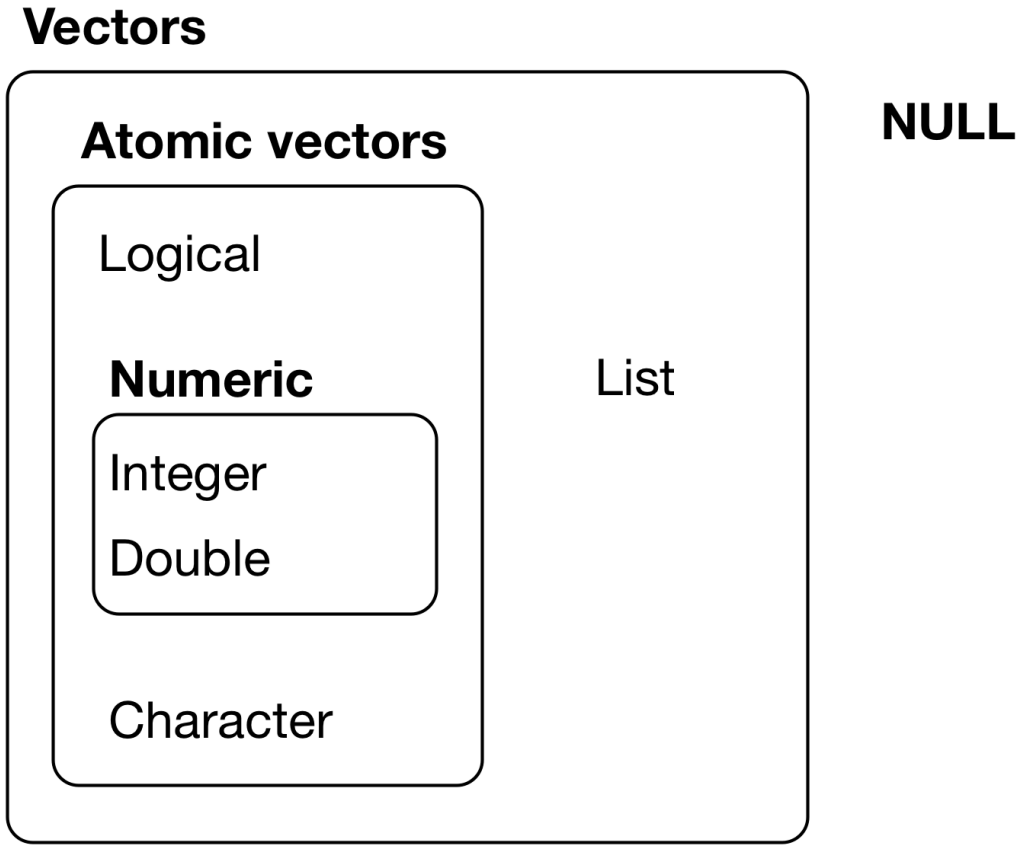
VETTORI

- Ci sono due tipi di vettori:
  1. Vettori **Atomic**
     - I vettori interi e double sono conosciuti collettivamente come vettori **numerici**
  2. **Liste**, che a volte sono chiamate vettori ricorsivi perché le liste possono contenere altre liste
- La differenza principale tra vettori atomici e liste è che i vettori atomici sono **omogenei**, mentre le liste possono essere **eterogenee**



VETTORI ATOMICI

- Il termine **atomico** significa che **ogni elemento del vettore è un singolo "atomo" indivisibile** di quel tipo:
  - un numero
  - una stringa
  - un valore logico
- Quindi:
  - un **vettore atomico** = sequenza di atomi dello stesso tipo
  - ogni elemento non può contenere altre strutture complesse



VETTORI ATOMICI

- Vettori **atomic**, di cui esistono sei tipi: **logical**, **integer**, **double**, **character**, **complex** e **raw**

Tipo (class)	Descrizione	Esempio
logical	Valori logici `TRUE` / `FALSE`	`c(TRUE, FALSE, TRUE)`
integer	Numeri interi (si usa il suffisso `L` per crearli esplicitamente)	`c(1L, 2L, 3L)`
double o **numeric**	Numeri reali / decimali	`c(1.5, 2.7, 3.0)`
character	Stringhe di testo	`c("a", "b", "c")`
complex	Numeri complessi	`c(1+2i, 3-4i)`
raw	Dati grezzi in byte	`as.raw(c(0, 255))`

```
> as.raw(c(0, 255))
[1] 00 ff
```

NULL VALUES

- Ogni tipo di vettore atomico ha il suo valore mancante:

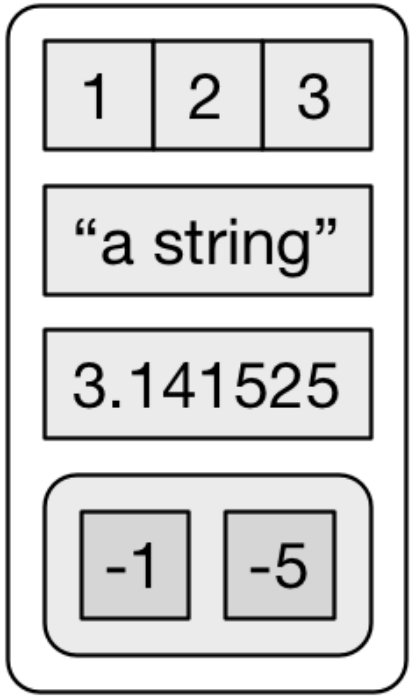
Tipo vettore atomico	Valore mancante associato	Esempio
logical	`NA` (logico)	`c(TRUE, NA, FALSE)`
integer	`NA_integer_`	`c(1L, 2L, NA_integer_)`
double/numeric	`NA_real_`	`c(1.5, NA_real_, 3.2)`
character	`NA_character_`	`c("a", NA_character_, "c")`
complex	`NA_complex_`	`c(1+2i, NA_complex_, 3+4i)`
raw	non esiste un `NA_raw_` (raw non ha NA)	_

```
> NA              # logical
[1] NA
> NA_integer_     # integer
[1] NA
```

```
> NA_real_        # double
[1] NA
> NA_character_ # character
[1] NA
```

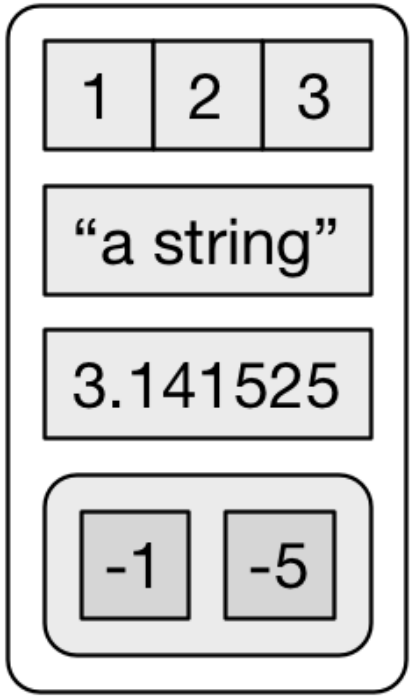
LISTE

- Una lista è una struttura dati in R che può contenere elementi eterogenei, ossia dati di diversi tipi (numeri, stringhe, vettori, liste, matrici, ecc.)
  - A differenza di vettori e array, gli elementi di una lista non devono essere dello stesso tipo o dimensione



LISTE

- Una lista è una struttura dati in R che può contenere elementi eterogenei, ossia dati di diversi tipi (numeri, stringhe, vettori, liste, matrici, ecc.)
  - A differenza di vettori e array, gli elementi di una lista non devono essere dello stesso tipo o dimensione



Caratteristica	Lista	Vettore	Array
Tipologia di dati	Può contenere dati di **diversi tipi**	Dati di un **solo tipo**	Dati di un **solo tipo**
Dimensione	Può avere elementi di **dimensioni diverse**	Tutti gli elementi hanno la **stessa dimensione**	Tutti gli elementi hanno la **stessa dimensione**
Struttura	**Struttura flessibile**	Struttura **monodimensionale**	Struttura **multidimensionale**

LISTE

- A differenza dei vettori, degli array e delle matrici, in R una lista è un oggetto che consiste in una raccolta di altri oggetti, che possono essere anche differenti tra loro, detti componenti della lista.
  - Ad esempio, i risultati creati dalla funzione *eigen()* si presentano come una lista costituita da un oggetto (primo componente) di tipo vettore, che contiene gli **autovalori**, e un oggetto (secondo componente) di tipo matrice, che contiene i corrispondenti **autovettori**

LISTE

- A differenza dei vettori, degli array e delle matrici, in R una lista è un oggetto che consiste in una raccolta di altri oggetti, che possono essere anche differenti tra loro, detti componenti della lista.
  - Ad esempio, i risultati creati dalla funzione *eigen()* si presentano come una lista costituita da un oggetto (primo componente) di tipo vettore, che contiene gli **autovalori**, e un oggetto (secondo componente) di tipo matrice, che contiene i corrispondenti **autovettori**

- Esempio:
  - **Accesso per nome**: mia_lista$numero accede direttamente all'elemento con il nome numero.
  - **Accesso per indice**: mia_lista[[2]] accede al secondo elemento della lista, in questo caso la stringa "R è Forte"

```r
# Creazione di una lista con elementi eterogenei
mia_lista <- list(
  numero = 42,
  stringa = "R è Forte",
  vettore = c(1, 2, 3),
  matrice = matrix(1:4, nrow = 2)
)

# Accesso agli elementi della lista
print(mia_lista$numero)     # Accesso tramite nome
print(mia_lista[[2]])       # Accesso tramite indice
```

```
[1] 42
[1] "R è Forte"
```

Manipolazione Di Liste

- Per aggiungere un elemento alla fine di una lista di *n−1* componenti basta utilizzare l'istruzione di assegnazione
- Ciò crea una lista di *n+1* componenti, con i componenti in posizione successiva a quello che inserito che vengono automaticamente spostati di una posizione

```r
# Creare una lista
mia_lista <- list(numero = 42, stringa = "R è potente")
mia_lista
```

$numero 42

$stringa 'R è potente'

```r
# Inserire un nuovo elemento
mia_lista$nuovo_elemento <- c(1, 2, 3)
mia_lista
```

$numero 42

$stringa 'R è potente'

$nuovo_elemento 1 · 2 · 3

MANIPOLAZIONE DI LISTE

- Per aggiungere un elemento alla fine di una lista di *n−1* componenti basta utilizzare l'istruzione di assegnazione
- Ciò crea una lista di *n+1* componenti, con i componenti in posizione successiva a quello che inserito che vengono automaticamente spostati di una posizione
- Per aggiungere un elemento in una posizione *j* di una lista con *n* componenti, basta utilizzare la funzione append() specificando:
  - La lista
  - L'elemento da aggiungere
  - La posizione *j*

```r
# Creare una lista
mia_lista <- list(numero = 42, stringa = "R è potente")
mia_lista
```

$numero 42
$stringa 'R è potente'

```r
# Inserire un nuovo elemento
mia_lista$nuovo_elemento <- c(1, 2, 3)
mia_lista
```

$numero 42
$stringa 'R è potente'
$nuovo_elemento 1 · 2 · 3

```r
# Inserimento di un nuovo elemento "nuovo" in posizione 2
mia_lista <- append(mia_lista, "nuovo Elemento", after = 1)

# Visualizzazione della lista aggiornata
mia_lista
```

$numero 42
[[2]] 'nuovo Elemento'
$stringa 'R è potente'
$nuovo_elemento 1 · 2 · 3

Manipolazione Di Liste

- Per cancellare il componente *j*-esimo di una lista basta invece utilizzare l'istruzione di assegnazione, passandogli l'oggetto NULL
- Ciò crea una lista di *n-1* componenti, con i componenti in posizione successiva a quello che inserito che vengono automaticamente spostati di una posizione indietro

```
# Inserimento di un nuovo elemento "nuovo" in posizione 2
mia_lista <- append(mia_lista, "nuovo Elemento", after = 1)

# Visualizzazione della lista aggiornata
mia_lista
```

$numero	42
[[2]]	'nuovo Elemento'
$stringa	'R è potente'
$nuovo_elemento	1 · 2 · 3

```
# Cancellare un elemento
mia_lista$nuovo_elemento <- NULL
mia_lista
```

$numero	100
[[2]]	'nuovo Elemento'
$stringa	'R è potente'

MANIPOLAZIONE DI LISTE

- Per cancellare il componente *j*-esimo di una lista basta invece utilizzare l'istruzione di assegnazione, passandogli l'oggetto NULL
- Ciò crea una lista di *n-1* componenti, con i componenti in posizione successiva a quello che inserito che vengono automaticamente spostati di una posizione indietro
- Per cancellare un elemento in una posizione *j* di una lista con *n* componenti, basta utilizzare lo slicing specificando:
  - La posizione *j*

```
# Inserimento di un nuovo elemento "nuovo" in posizione 2
mia_lista <- append(mia_lista, "nuovo Elemento", after = 1)

# Visualizzazione della lista aggiornata
mia_lista
```

$numero	42
[[2]]	'nuovo Elemento'
$stringa	'R è potente'
$nuovo_elemento	1 · 2 · 3

```
# Cancellare un elemento
mia_lista$nuovo_elemento <- NULL
mia_lista
```

$numero	100
[[2]]	'nuovo Elemento'
$stringa	'R è potente'

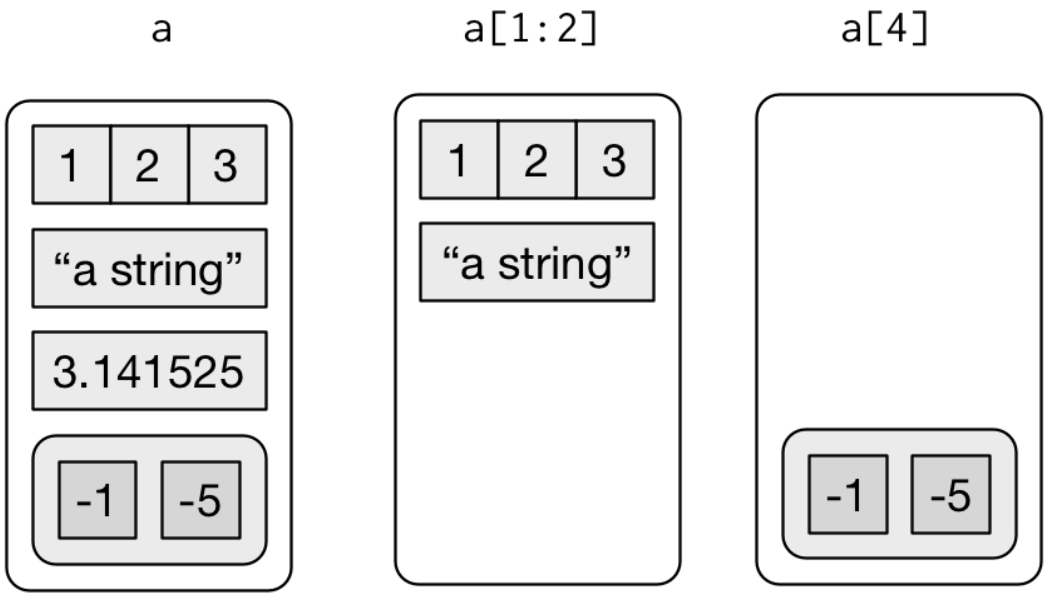
```
# Cancellare un elemento in posizione 2
mia_lista <- mia_lista[-3]
mia_lista
```

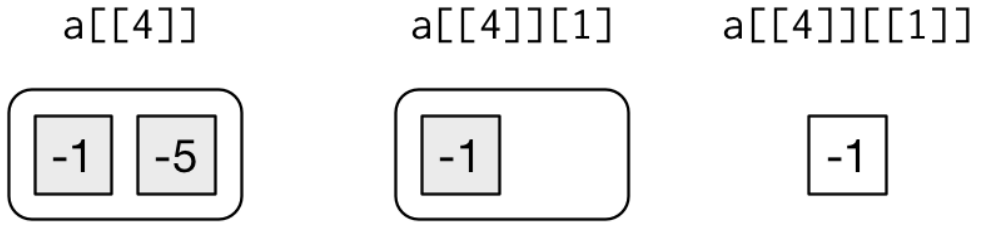
$numero	42
[[2]]	'nuovo Elemento'

ACCESSO ALLE LISTE

```
> lista <- list(a = 1:3, b = "a string", c = pi, d = list(-1, -5))
> lista
$a
[1] 1 2 3

$b
[1] "a string"

$c
[1] 3.141593

$d
$d[[1]]
[1] -1

$d[[2]]
[1] -5
```

- La distinzione tra `[` e `[[` è davvero importante per le liste, perché `[[` si addentra nella lista mentre `[` restituisce una nuova lista più piccola





ACCESSO ALLE LISTE (NELLA VITA REALE)

- Consideriamo una saliera `x`
  - `x[1]` è una saliera di pepe contenente un **singolo (primo)** pacchetto di pepe
  - `x[2]` avrebbe lo stesso aspetto, ma conterrebbe il **secondo** pacchetto
  - `x[1:2]` sarebbe una saliera per il pepe contenente **due** pacchetti di pepe

`x`

`x[1]`

ACCESSO ALLE LISTE (NELLA VITA REALE)

- Consideriamo una saliera `x`
  - `x[[1]]` è un **singolo (primo)** pacchetto di pepe
  - Se voleste ottenere il contenuto del pacchetto peppe `x[[1]][[1]]`

`x`

`x[1]`

`x[[1]]`

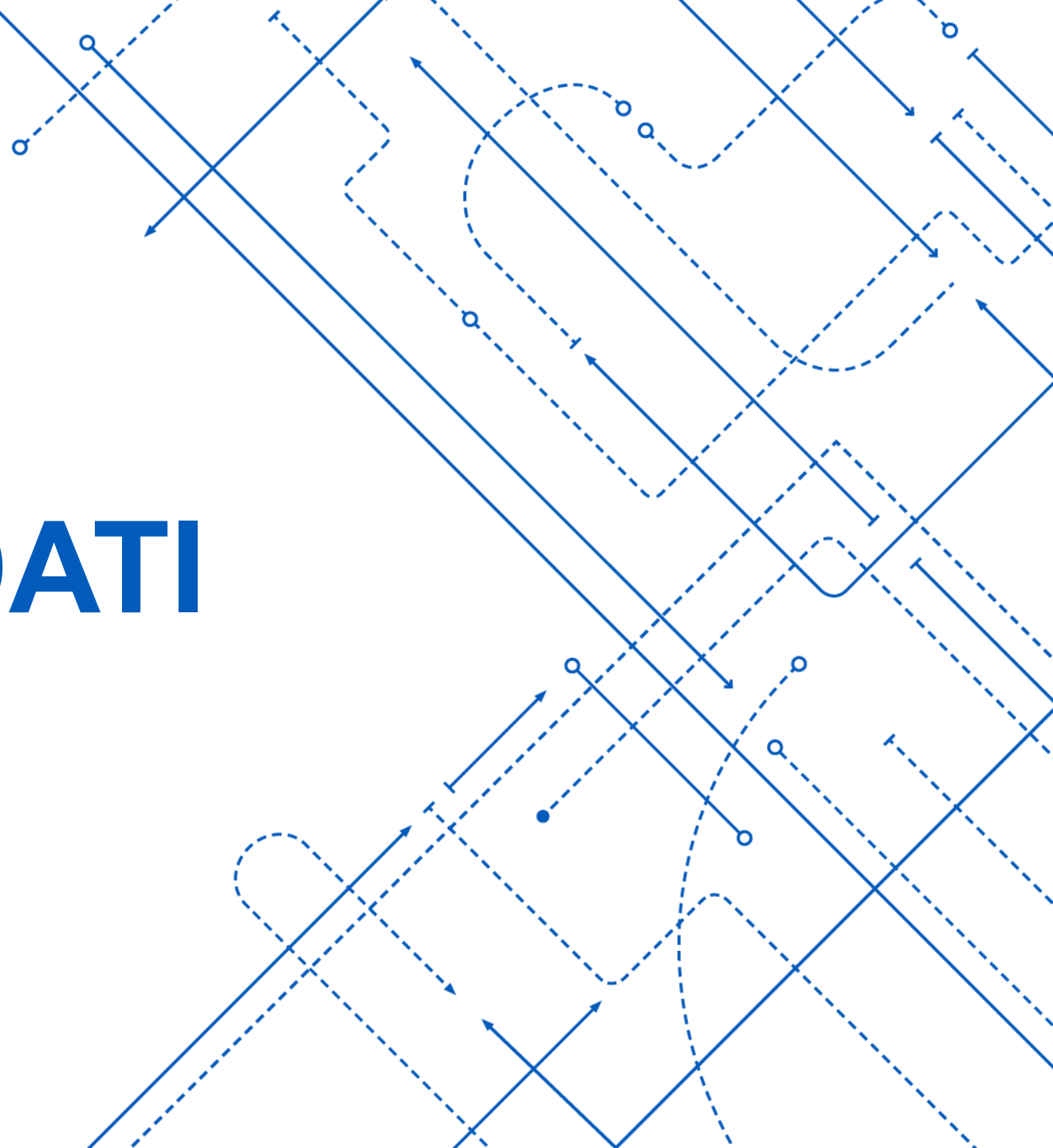
`x[[1]][[1]]`

STATISTICA E ANALISI DEI DATI

DataFrame

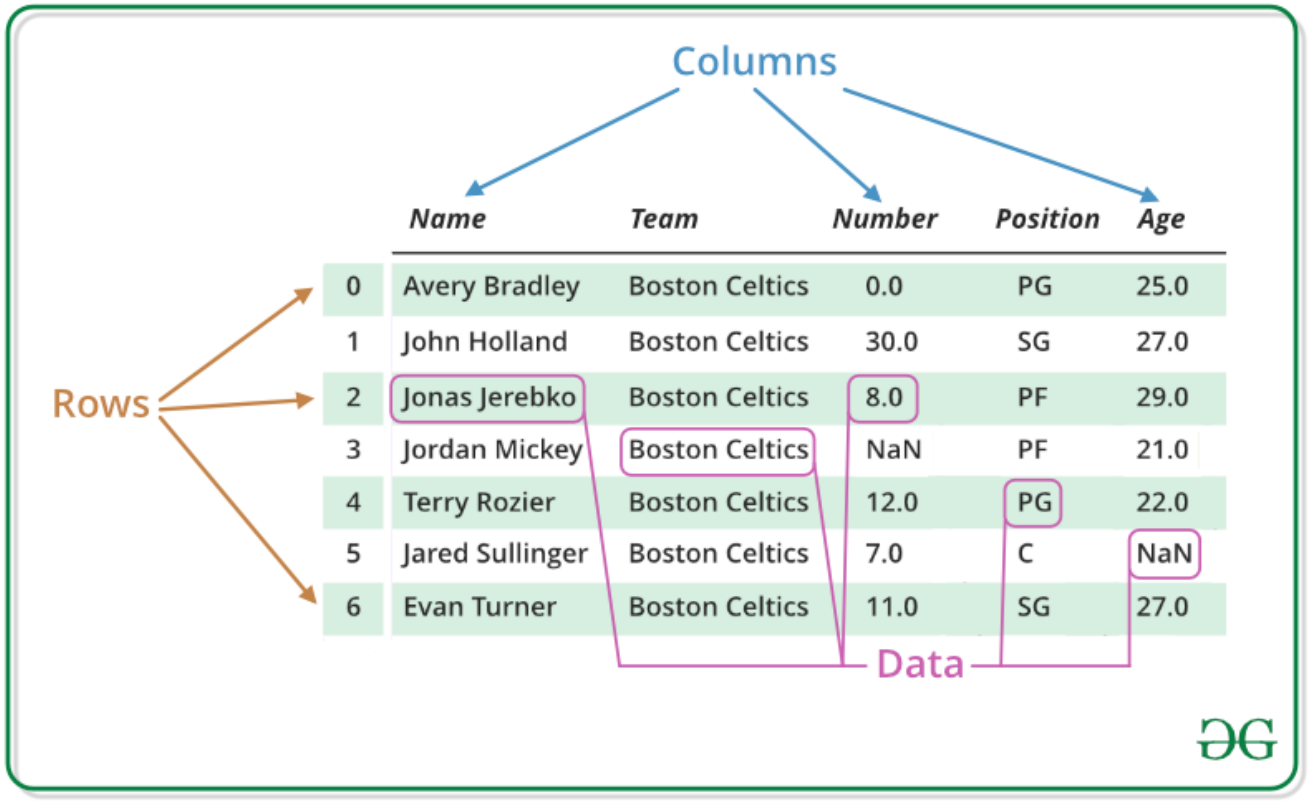
Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026



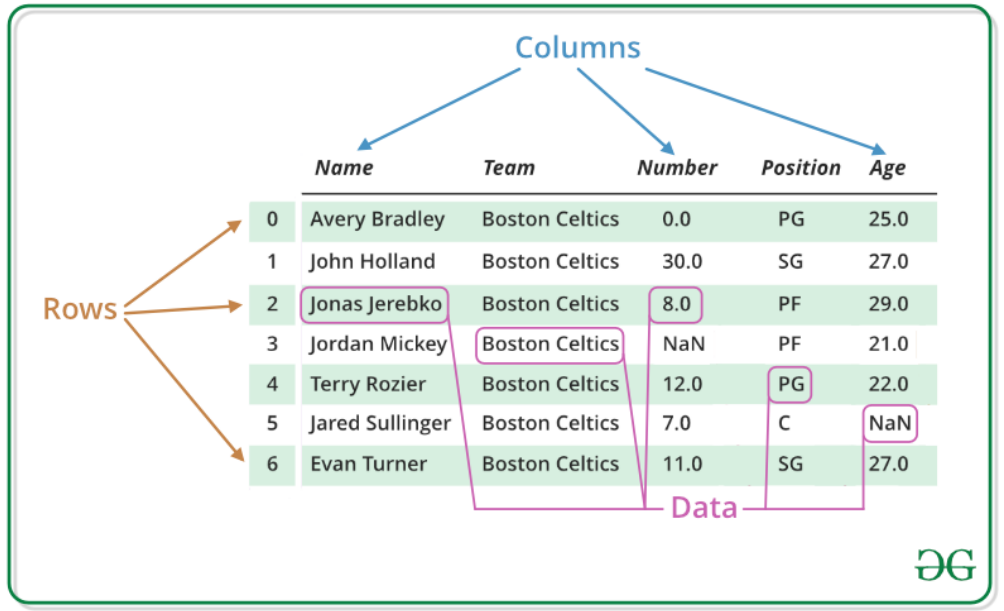
DataFrame

- Il dataframe è (probabilmente) l'oggetto più utilizzato di tutto l'ambiente R, almeno in una sua ottica di gestione e analisi dei dati
- Un dataframe è una matrice di dati in cui ad ogni riga corrisponde una osservazione (unità statistica) e ad ogni colonna una variabile statistica, e nell'ambiente viene trattato come una lista
- Ogni elemento di tale lista rappresenta una variabile statistica, per cui length() restituisce il numero delle variabili, mentre names() i rispettivi nomi
- Tra le diverse opzioni disponibili, è possibile costruire un dataframe direttamente con la funzione data.frame()



DATAFRAME

- In R un data frame è un oggetto di tipo lista che si presenta in forma di tabella (matrice di dati) ed è costituito da righe e colonne.
  - Ogni riga del data frame individua un'osservazione e ad ogni colonna corrisponde una variabile
- Come per le matrici, le colonne di un data frame hanno tutte la stessa lunghezza e i valori contenuti in ogni singola colonna sono omogenei (numeri, caratteri, valori logici, . . .).
  - Invece, a differenza delle matrici, un data frame può avere colonne di tipo diverso
- Le righe e le colonne possono avere delle etichette costituite da nomi o da valori numerici
  - Il numero di colonne è individuato da *length(nomeDataFrame)*, mentre *dim(nomeDataFrame)* fornisce un vettore contenente il numero di righe e di colonne del data frame



DataFrame

- Il comando *names(nomeDataFrame)* restituisce i nomi delle etichette delle colonne
  - I nomi delle righe (indici di riga) possono essere modificati attraverso il comando:

    rownames(nomeDataFrame) <- c("riga1", "riga2", "riga3")
  - I nomi delle colonne (feature/attributi) possono essere modificati attraverso il comando:

    colnames(nomeDataFrame) <- c("colonna1", "colonna2", "colonna3")

```
# Creare un dataframe di esempio
df <- data.frame(
  x = c(1, 2, 3),
  y = c(4, 5, 6),
  z = c(7, 8, 9)
)

df
```

A data.frame: 3 × 3

x	y	z
<dbl>	<dbl>	<dbl>
1	4	7
2	5	8
3	6	9

```
# Modificare i nomi delle righe
row.names(df) <- c("riga1", "riga2", "riga3")
colnames(df) <- c("colonna1", "colonna2", "colonna3")

# Visualizzare i nuovi nomi delle righe
df
```

A data.frame: 3 × 3

	colonna1	colonna2	colonna3
	<dbl>	<dbl>	<dbl>
riga1	1	4	7
riga2	2	5	8
riga3	3	6	9

DataFrame

```
# crea un datafame
> df <- data.frame(a = 1:3, sesso = c("F", "M", "F"))
> df
```

```
##   a sesso
## 1 1      F
## 2 2      M
## 3 3      F
```

```
> length(df)  # n. variabili
```

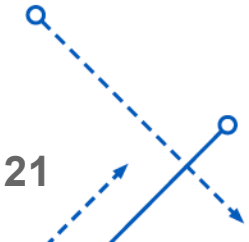
```
## [1] 2
```

```
> names(df)  # nome delle variabili
```

```
## [1] "a"     "sesso"
```

```
> dim(df)  # restituisce la dimensione (n. osservazioni e n. variabili)
```

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



DataFrame

```
> str(df)  # restituisce informazioni circa la struttura di df
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ a    : int  1 2 3
##  $ sesso: chr  "F" "M" "F"
```

- Possiamo aggiungere altre variabili (colonne) al dataframe assegnando direttamente un nome alla variabile

```
> df$eta <- c(21, 19, 20)  # aggiunge una variabile di nome 'eta'
> df
```

```
##   a sesso eta
## 1 1      F  21
## 2 2      M  19
## 3 3      F  20
```

SUBSETTING DI DATAFRAME

- Il dataframe creato è definito da 3 casi e 3 variabili (due numeriche ed una categoriale) che possono essere selezionate in modi diversi utilizzando il nome della variabile:
  - df$eta
  - df[,“eta”]
- Utilizzando il numero di colonna che la variabile occupa:
  - df[,2]
- Il risultato è comunque un vettore
- Nelle applicazioni è utile selezionare soltanto una parte del dataframe iniziale, ad es. solo alcune variabili o soltanto alcuni casi che soddisfano determinati criteri

FILTERING AND SELECTION

```
> # selezione dell'età delle sole femmine
> df[df$sesso == "F", "eta"]
```

```
## [1] 21 20
```

```
> # selezione delle sole femmine con eta<=20 (si fa
> # notare l'uso dell'operatore logico '&' affinché
> # siano soddisfatte entrambe le condizioni)
> df[df$sesso == "F" & df$eta <= 20, ]
```

```
##   a sesso eta
## 3 3     F  20
```

```
> # selezione dei valori di 'sesso' con eta<20 o
> # eta>20 (si fa notare l'uso dell'operatore
> # logico '|' affinché siano soddisfatte
> # alternativamente le due condizioni)
> df[df$eta < 20 | df$eta > 20, "sesso"]
```

```
## [1] "F" "M"
```

FILTERING AND SELECTION

Alternativamente è possibile selezionare sottoinsiemi di dati utilizzando la funzione subset(***data***,***condizione***,***selezione***)

```
> # selezione dei casi con età <=20
> subset(df, eta <= 20)

##   a sesso eta
## 2 2      M  19
## 3 3      F  20

> # selezione dei casi con 19 e 21 anni (si fa
> # notare l'uso dell'operatore logico %in% che
> # permette di specificare una lista di valori
> # come clausole di ricerca in un'unica query)
> subset(df, eta %in% c(19, 21))
##   a sesso eta
## 1 1      F  21
## 2 2      M  19
```

DATAFRAME

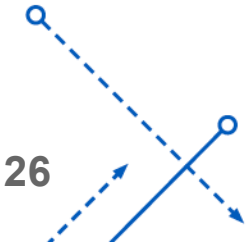
- La funzione data.frame() vista in precedenza non esaurisce le diverse possibilità per affrontare il problema dell'inserimento e gestione di dati
- Ad es. la funzione as.data.frame() può essere utilizzata per forzare una matrice di dati ad un dataframe

```
> # Richiamiamo la matrice X1
> str(X1)
```

```
##  int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
```

```
> df1 <- as.data.frame(X1)
> str(df1)
```

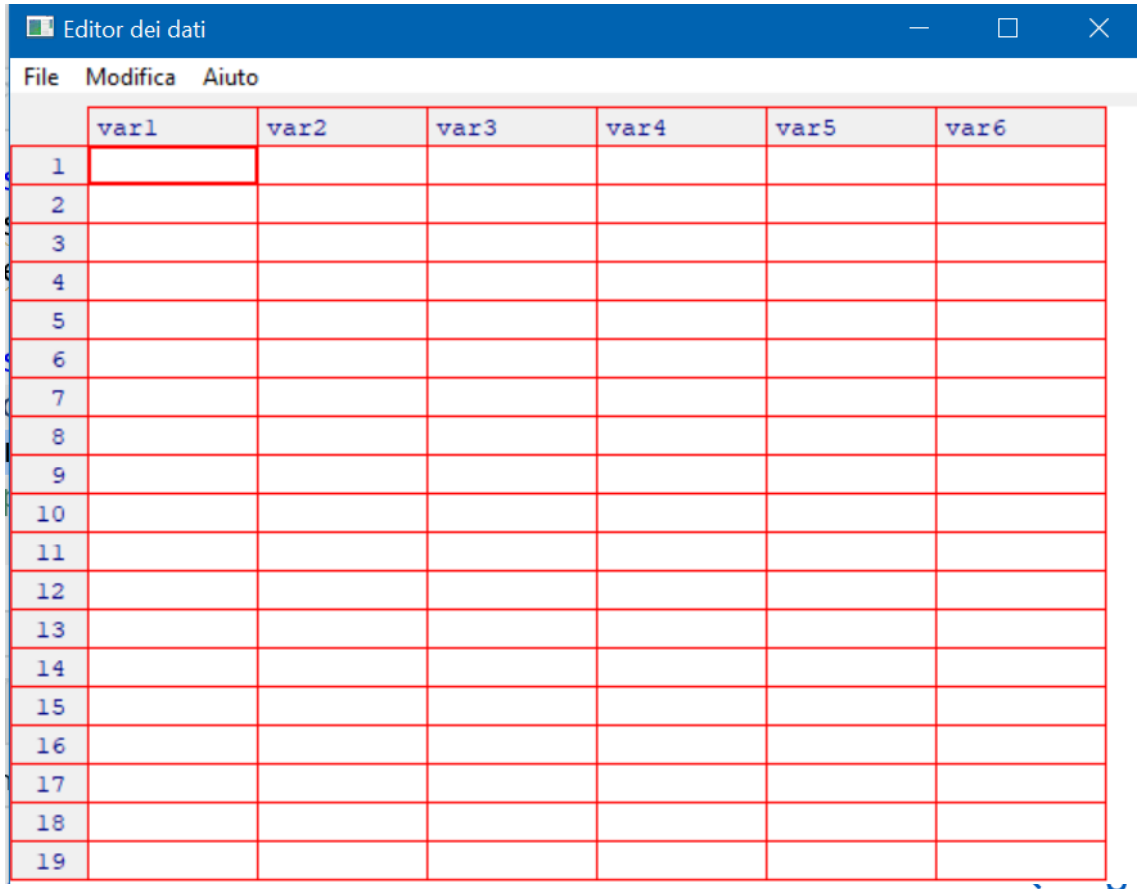
```
## 'data.frame':    4 obs. of  3 variables:
##  $ V1: int  1 2 3 4
##  $ V2: int  5 6 7 8
##  $ V3: int  9 10 1 2
```



DATAFRAME

- I dati possono essere passati ad R anche attraverso l'inserimento manualmente utilizzando un foglio elettronico
- Si può utilizzare il comando **fix**()

```
> X <- data.frame()
> fix(X)  # apre un Editor di dati
> # che permette l'inserimento dei dati cella per
+ # cella
```



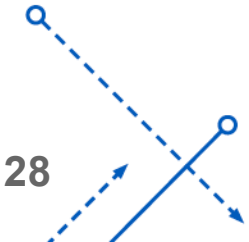
CARICAMENTO DI FILE

- Un'altra possibilità (probabilmente la più frequente nelle applicazioni), è quella di importare i dati da un file ASCII (.txt, .asc o .csv)
- A tale scopo viene utilizzata la funzione **read.table**( ) o **read.delim**( )

```
> stud <- read.table(file.choose(), header = TRUE, sep = "\t",
                     na.strings = "NA", dec = ",")
```

```
> str(stud)
```

```
##  'data.frame':    20 obs. of  7 variables:
##  $ id     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Sex    : chr  "Female" "Male" "Male" "Male" ...
##  $ Age    : num  18.2 17.6 16.9 20.3 23.7 ...
##  $ Height: num  173 178 NA 160 165 ...
##  $ WHnd   : chr  "Right" "Left" "Right" "Right" ...
##  $ Pulse : int  92 104 87 NA 35 64 83 74 72 90 ...
##  $ Smoke : chr  "Never" "Regul" "Occas" "Never" ...
```



CARICAMENTO DI FILE

- In questo caso, il file *"Iris.txt"* viene importato e automaticamente convertito nel dataframe *stud*
- Alcuni argomenti della funzione sono
  - Il path (o percorso del file)
    - può essere specificato e scritto come negli ambienti Unix (o Linux) con "/" (ad es. *"C:/Data/Iris.txt"*)
    - il percorso viene dinamicamente scelto attraverso la funzione file.choose() che apre la finestra di navigazione
      - header=TRUE specifica che la prima linea del file contiene le intestazioni delle colonne
      - sep="\t" indica che i diversi campi sono separati da un tab (altre opzioni posso essere ad es. ";",",")
      - na.strings="NA" è utile se nel file sono presenti valori mancanti, in questo caso individuati con NA
      - dec="," specifica il tipo di carattere utilizzato nel file per separare i decimali, in questo caso la virgola

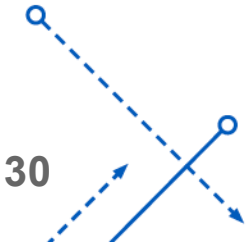
CARICAMENTO DI FILE

Se il file di input è in formato ".csv" si utilizza la funzione read.csv() con la stessa logica vista in precedenza

```
> salary <- read.csv(file.choose(), header = TRUE, sep = ",")
```

```
> str(salary)
```

```
## 'data.frame':    8 obs. of  5 variables:
##  $ id         : int  1 2 3 4 5 6 7 8
##  $ name       : chr  "Rick" "Dan" "Michelle" "Ryan" ...
##  $ salary     : num  623 515 611 729 843 ...
##   $ start_date: chr  "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
##  $ dept       : chr  "IT" "Operations" "IT" "HR" ...
```



L'AMBIENTE INTEGRATO R

Principali funzioni su data frame

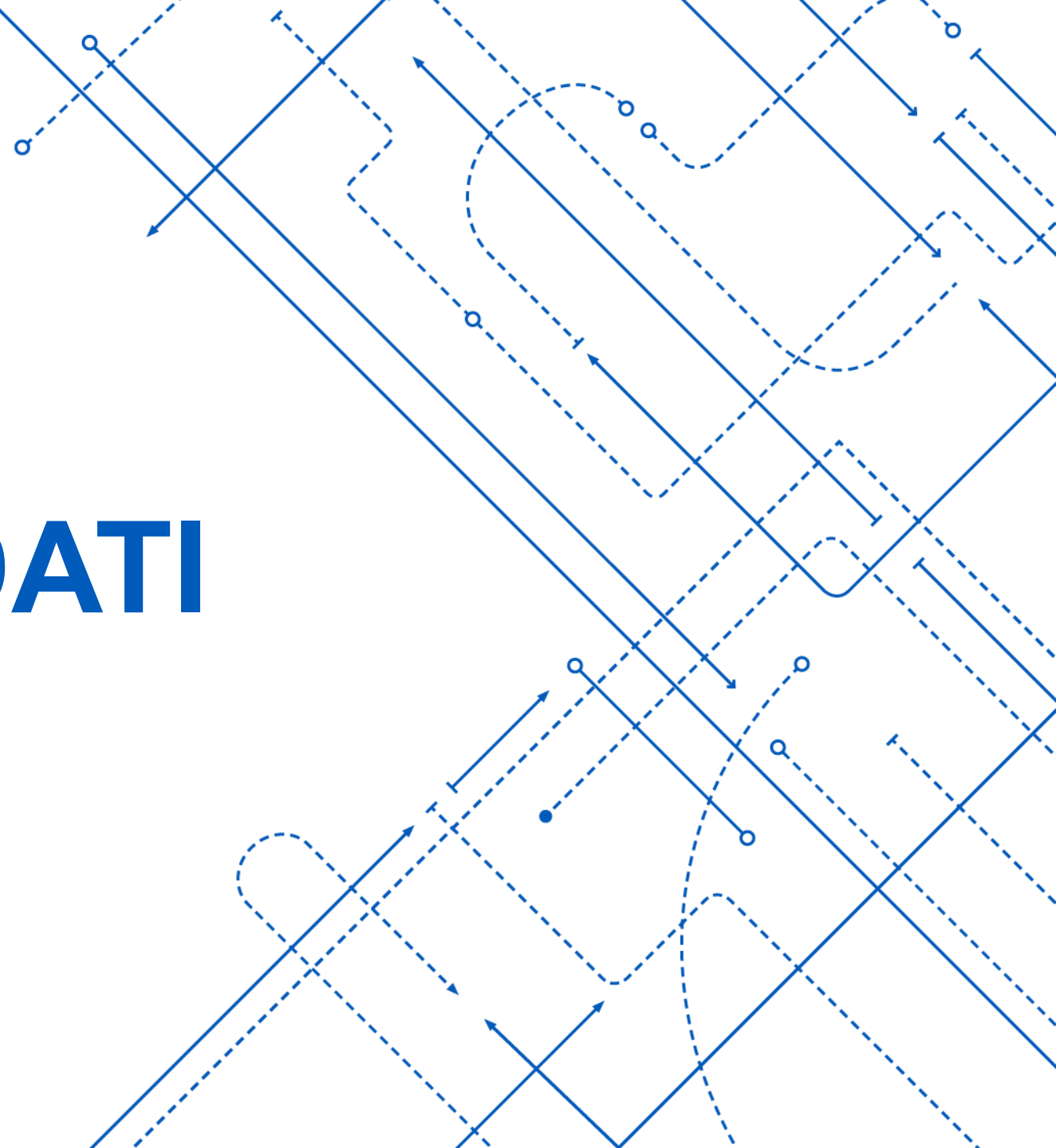
- **Numero di colonne del dataframe df**
  `> length(df)`
- **Numero di righe e colonne (osservazioni e variabili) del data frame df**
  `> dim(df)`
- **Nomi delle etichette delle colonne del data frame df**
  `> names(df)`
- **Assegna/modifica le etichette delle righe del data frame df**
  `> row.names(df)`
- **Individua l'elemento in posizione (i, j) nel data frame df**
  `> df[i,j]`
- **Seleziona nella k-esima colonna gli elementi da i a j**
  `> df[i:j, k]`
- **Seleziona una parte del data frame df avente le righe da i a j e le colonne da k a r**
  `> df[i : j, k : r]`
- **Seleziona la parte del data frame df costituita dalla i-esima riga**
  `> df[i,]`
- **Seleziona la parte del data frame df costituita dalla j-esima colonna**
  `> df[,j]`

STATISTICA E ANALISI DEI DATI

Fattori ed Intervalli

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026



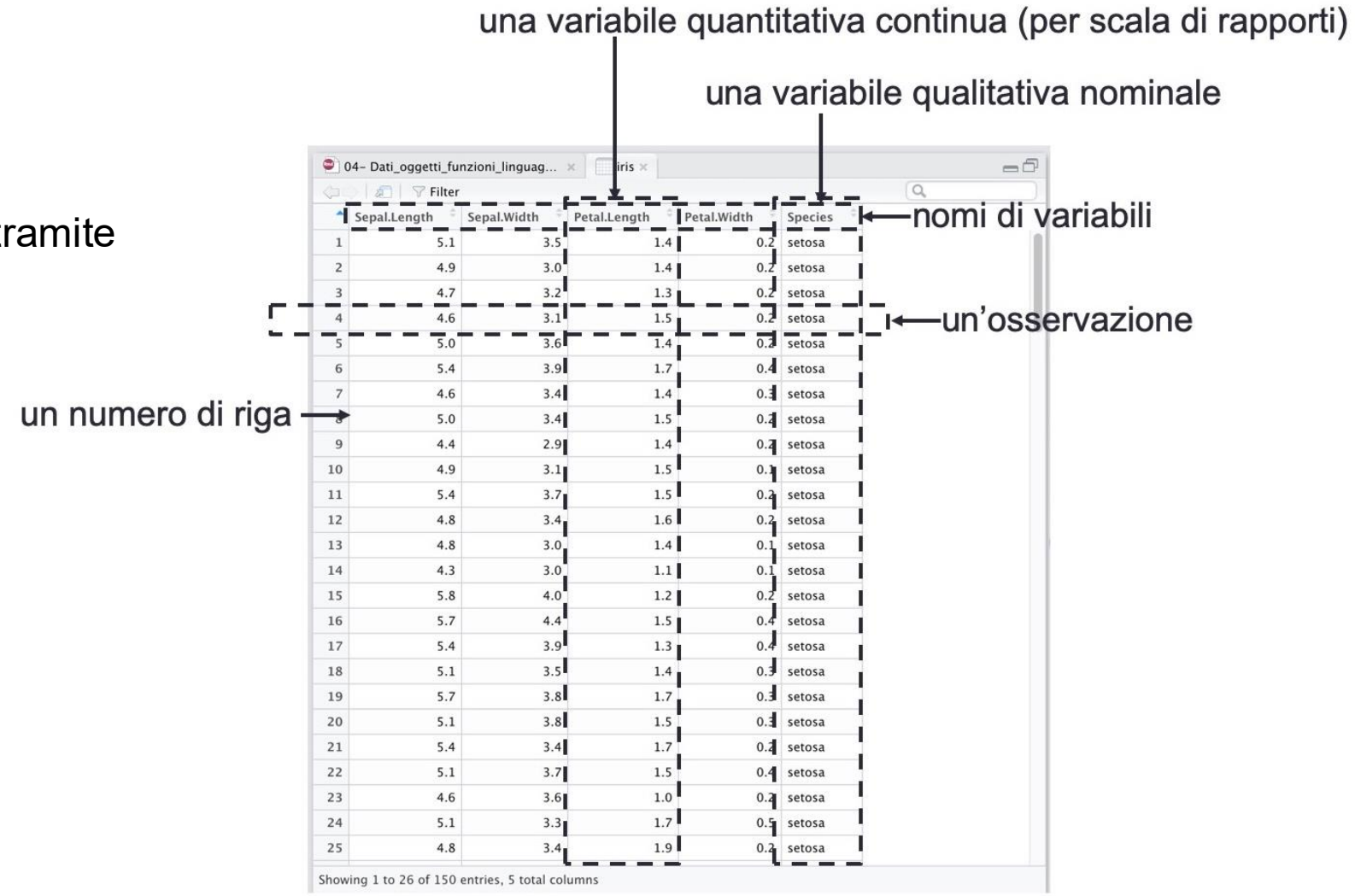
VARIABILI QUANTITATIVE E QUALITATIVE

QUANTITATIVI

I dati quantitativi sono rappresentati tramite valori numerici.

QUALITATIVI

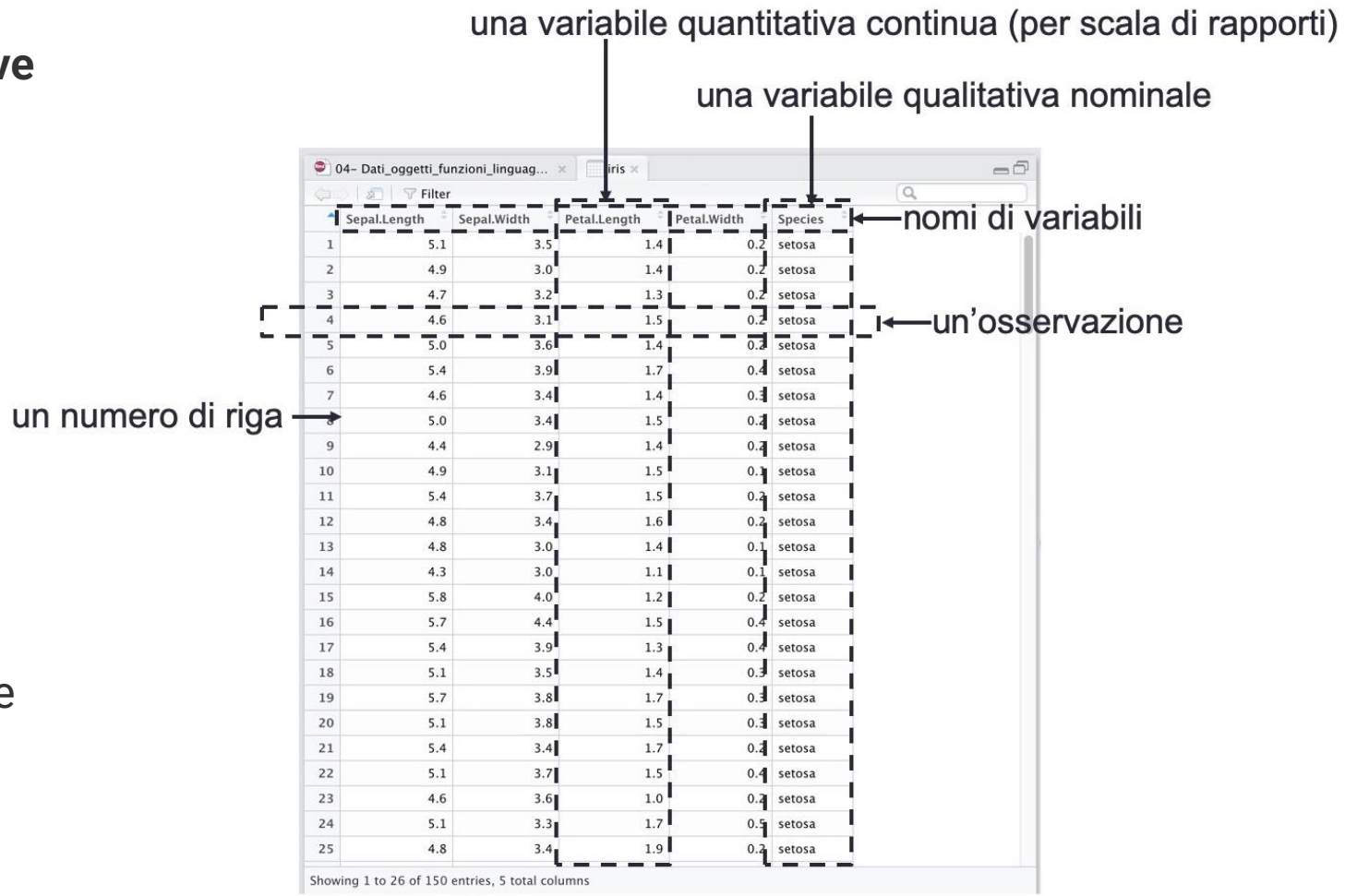
I dati qualitativi sono rappresentati attraverso stringhe di caratteri oppure mediante opportune classi (sottointervalli numerici).



	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
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa

VARIABILI QUANTITATIVE E QUALITATIVE

- *Iris* Dataset ha quattro **variabili quantitative continue**, per **scala di rapporti**:
  - Lunghezza e larghezza dei petali
  - Lunghezza e larghezza dei sepali
- una **variabile qualitativa nominale**:
  - il **nome** delle tre specie del genere *Iris* messe a confronto
  - Questa variabile può assumere un **numero finito di valori** o **categorie** che non hanno nessun ordine ovvio
  - In R questo tipo di variabile è chiamata "**fattore**"



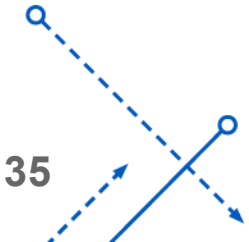
FATTORI

- In R, i **fattori** sono una struttura di dati utilizzata per rappresentare dati categoriali, cioè variabili che possono assumere un numero finito e fisso di valori distinti, chiamati livelli (levels)
  - I fattori sono particolarmente utili per gestire dati che rappresentano categorie o classi, come sesso, giorno della settimana, o risposte in un sondaggio.
- Possono rappresentare dati qualitativi, come:
  - Generi: 'Uomo', 'Donna', 'Non Binario', …
  - Colori: "Rosso", "Blu", "Verde", …

```r
# Esempio di creazione di un fattore
genere <- factor(c("Rosso", "Blu", "Verde", "Arancione"))
genere
```

```
[1] Rosso     Blu       Verde     Arancione
Levels: Arancione Blu Rosso Verde
```

- I fattori sono utili per:
  - Analisi statistiche che richiedono categorie distinte
  - Costruire modelli statistici con variabili categoriali
  - Visualizzazione grafica di dati categoriali (es. grafici a barre)



FATTORI

```
> individuo <-c( "bambino" ,"giovane" ,"adulto" ,"anziano" ,"bambino" ,"giovane")
> individuo
[1] "bambino" "giovane" "adulto"  "anziano" "bambino" "giovane"
> persona <- factor(individuo)
> persona
[1] bambino giovane adulto  anziano bambino giovane
Levels: adulto anziano bambino giovane
>
> str(individuo)
 chr [1:6] "bambino" "giovane" "adulto" "anziano" "bambino" "giovane"
> str(persona)
 Factor w/ 4 levels "adulto","anziano",..: 3 4 1 2 3 4
>
>
> persona1 <- ordered(individuo, levels=c("bambino", "giovane", "adulto", "anziano"))
> str(persona1)
 Ord.factor w/ 4 levels "bambino"<"giovane"<..: 1 2 3 4 1 2
> persona1
[1] bambino giovane adulto  anziano bambino giovane
Levels: bambino < giovane < adulto < anziano
>
> persona <- factor(individuo)
> ordered(persona, levels=c("bambino","giovane", "adulto", "anziano"))
[1] bambino giovane adulto  anziano bambino giovane
Levels: bambino < giovane < adulto < anziano
```

Metodo 1

FATTORI

```
> individuo <-c( "bambino" ,"giovane" ,"adulto" ,"anziano" ,"bambino" ,"giovane")
> individuo
[1] "bambino" "giovane" "adulto"  "anziano" "bambino" "giovane"
> persona <- factor(individuo)
> persona
[1] bambino giovane adulto  anziano bambino giovane
Levels: adulto anziano bambino giovane
>
> str(individuo)
 chr [1:6] "bambino" "giovane" "adulto" "anziano" "bambino" "giovane"
> str(persona)
 Factor w/ 4 levels "adulto","anziano",..: 3 4 1 2 3 4
>
>
> persona1 <- ordered(individuo, levels=c("bambino", "giovane", "adulto", "anziano"))
> str(persona1)
 Ord.factor w/ 4 levels "bambino"<"giovane"<..: 1 2 3 4 1 2
> persona1
[1] bambino giovane adulto  anziano bambino giovane
Levels: bambino < giovane < adulto < anziano
>
> persona <- factor(individuo)
> ordered(persona, levels=c("bambino","giovane", "adulto", "anziano"))
[1] bambino giovane adulto  anziano bambino giovane
Levels: bambino < giovane < adulto < anziano
```

Metodo 1

Metodo 2

FATTORI

- Perché usare i fattori?
  - **Analisi**: Molte funzioni in R per l'analisi statistica e la modellizzazione richiedono fattori per gestire correttamente le variabili categoriali.
  - **Definizione di Grafici**: Quando si creano grafici, i fattori assicurano che le categorie siano gestite correttamente e rappresentate con l'ordinamento desiderato.

- Per convertire un fattore in un vettore di caratteri:

  as.character(fattore)

- Per convertire un fattore in un vettore di numerico:

  as.numeric(fattore)

- I fattori sono strumenti potenti per gestire dati categoriali e ordinali in R, e vengono utilizzati per rappresentare variabili con un insieme fisso e finito di valori

```
grado <- factor(c("alto", "basso", "medio", "basso", "alto"),
                levels = c("basso", "medio", "alto"),
                ordered = TRUE)

print(grado)
```

```
[1] alto  basso medio basso alto
Levels: basso < medio < alto
```

```
as.character(grado)
```

'alto' · 'basso' · 'medio' · 'basso' · 'alto'

```
as.numeric(grado)
```

3 · 1 · 2 · 1 · 3

INTERVALLI - CUT

- In R si utilizza la funzione cut() per trasformare dati numerici in dati qualitativi mediante la loro collocazione in opportune classi:

  cut(X, breaks, labels = NULL, include.lowest = FALSE, **right** = TRUE)

  - X: il vettore numerico da suddividere
  - breaks: specifica il numero di intervalli o un vettore con i limiti degli intervalli
  - labels: etichette opzionali per i livelli.
  - Include.lowest: se TRUE, include il valore più basso nel primo intervallo.
  - **right**: se TRUE, l'intervallo include l'estremo destro.
- Nel parametro breaks si inseriscono gli estremi degli intervalli che sono aperti a sinistra e chiusi a destra.
- Se si desidera ottenere intervalli chiusi a sinistra e aperti a destra occorre specificare in cut() l'opzione aggiuntiva
  right = FALSE

INTERVALLI - CUT

- Nell'esempio seguente si considera un vettore numerico contenente i voti di 8 studenti e si considerano le seguenti classi: (17, 21], (21, 24], (24, 27], (27, 31]

```
> votiStudenti <- c(18,21,25,28,30,25,24,25)
> votiStudenti
[1] 18 21 25 28 30 25 24 25
>
> voti <- cut(votiStudenti, breaks=c(17,21,24,27,31))
> voti
[1] (17,21] (17,21] (24,27] (27,31] (27,31] (24,27] (21,24] (24,27]
Levels: (17,21] (21,24] (24,27] (27,31]
```

INTERVALLI - CUT

- Nell'esempio seguente si considera un vettore numerico contenente i voti di 8 studenti e si considerano le seguenti classi: (17, 21], (21, 24], (24, 27], (27, 31]

```
> votiStudenti <- c(18,21,25,28,30,25,24,25)
> votiStudenti
[1] 18 21 25 28 30 25 24 25
>
> voti <- cut(votiStudenti, breaks=c(17,21,24,27,31))
> voti
[1] (17,21] (17,21] (24,27] (27,31] (27,31] (24,27] (21,24] (24,27]
Levels: (17,21] (21,24] (24,27] (27,31]
```

- Nell'esempio seguente i valori numerici sono suddivisi in 4 categorie: "Giovane", "Adulto", "Mezza Età" e "Anziano".

```
# Esempio: Suddividere un vettore di età in categorie
eta <- c(15, 22, 37, 45, 52, 63, 80)
categorie_eta <- cut(eta, breaks = c(0, 18, 35, 50, 100),
                     labels = c("Giovane", "Adulto", "Mezza Età", "Anziano"))
categorie_eta
```

Giovane · Adulto · Mezza Età · Mezza Età · Anziano · Anziano · Anziano

▼ **Levels**:

'Giovane' · 'Adulto' · 'Mezza Età' · 'Anziano'

STATISTICA E ANALISI DEI DATI

Capitolo 1.10 – Gestione I/O Funzioni

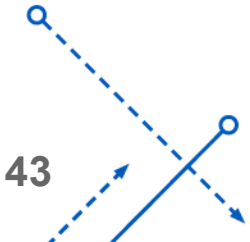
Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026

FUNZIONI IN R

Gestione dell'input delle funzioni in R

- Flashback: In **R**, una funzione **accetta zero o più * input** e restituisce **almeno un output** (che è NULL nel caso la funzione sia vuota).
  - Osserviamo meglio il formalismo che ci permette di definire funzioni in R:
  - `f <- function() { }` **Una funzione senza parametri di input.**
  - `f <- function(arg1) { }` **Una funzione con un parametro di input.**
  - `f <- function(arg1, arg2, ..., argN) { }` **Una funzione con N parametri di input**
  - **Ma c'è di più!** con «più» in R si intende anche «un numero indefinito» di parametri di input!



FUNZIONI IN R

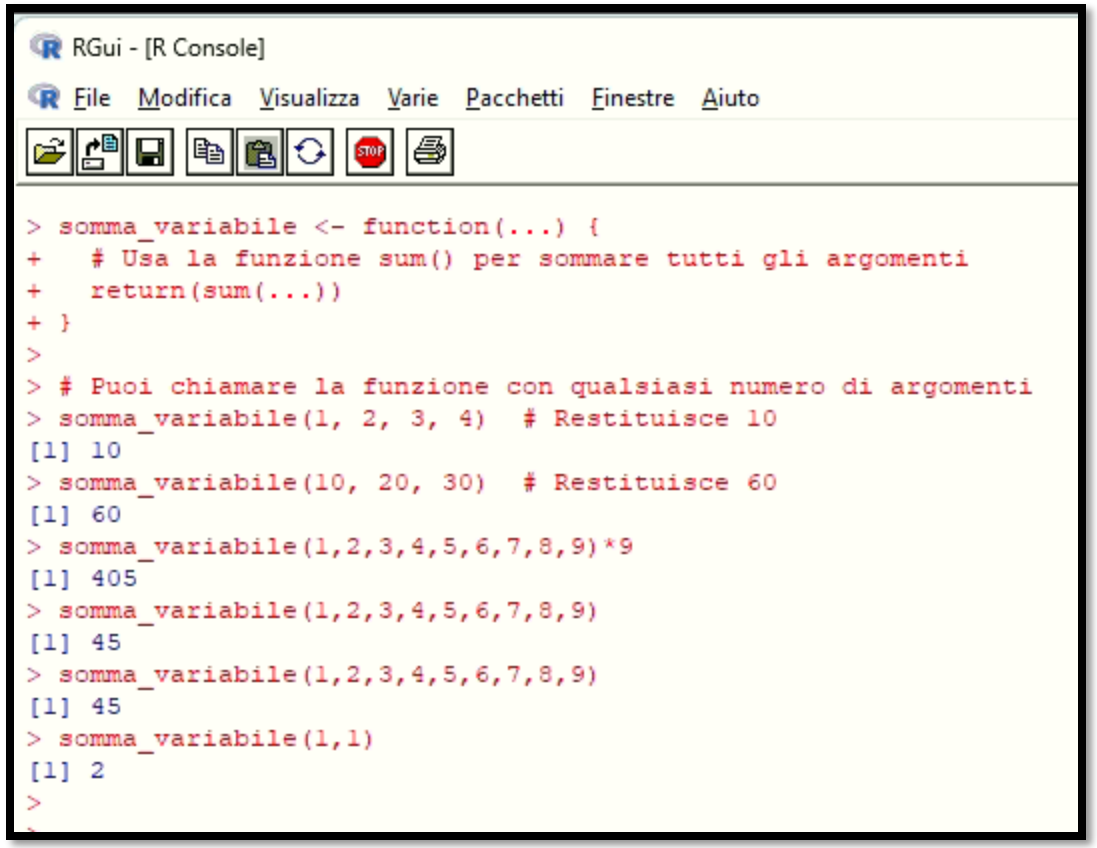
Funzioni con numero di parametri in input variabile

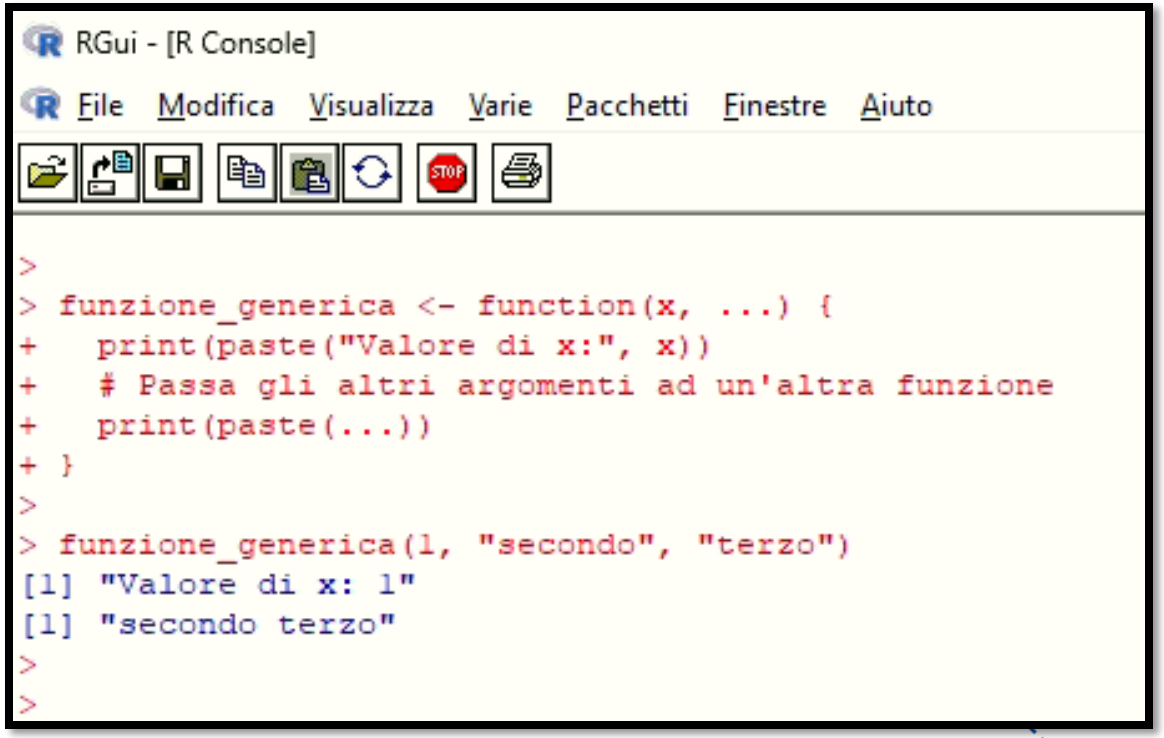
- Ci sono dei casi in cui potrebbe essere utile definire una funzione il cui numero di input vari in base al contesto o all'uso della funzione stessa.
- In **R**, è possibile definire una funzione che **accetta un numero di parametri non definito a priori**!
- La sintassi più comune per accettare un numero indefinito di argomenti è l'uso dei tre puntini ...
  - `f <- function(...) { }` **Una funzione con un numero di parametri variabile.**
- L'uso dei tre puntini (...) ci permette di passare un numero arbitrario di argomenti alla funzione.
- Gli argomenti passati con (...) possono essere successivamente manipolati o passati ad altre funzioni.

FUNZIONI IN R

Funzioni con numero di parametri in input variabile

- Alcuni esempi che ci permettono di apprezzare l'utilità di questo formalismo.

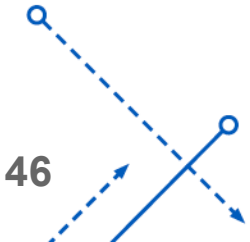




FUNZIONI IN R

Gestione dell'output di funzioni in R

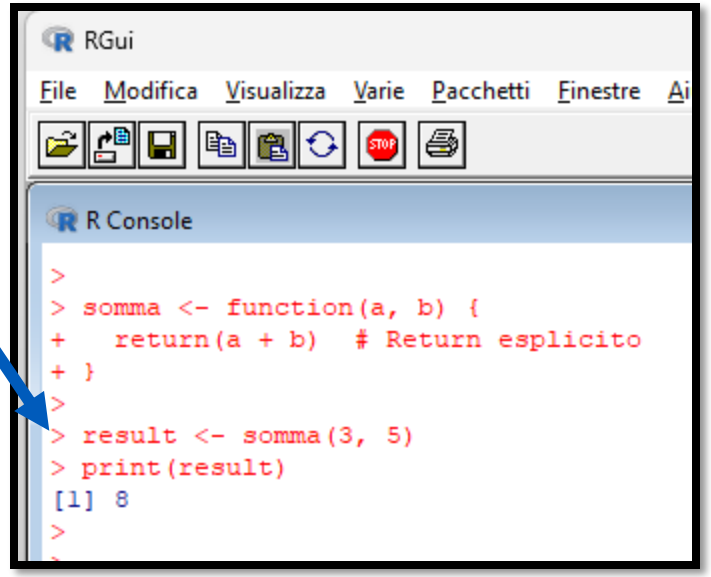
- In R le funzioni possono adottare un meccanismo di ***return*** implicito e uno esplicito:
  - se non è presente la keyword return, una funzione restituisce in output **il valore calcolato nell'ultima espressione** (return implicito);
  - **se è presente** l'istruzione `return(value)`, la funzione ritorna in output l'oggetto value che può essere un singolo oggetto oppure un insieme di oggetti.



FUNZIONI IN R

Gestione dell'output di funzioni in R

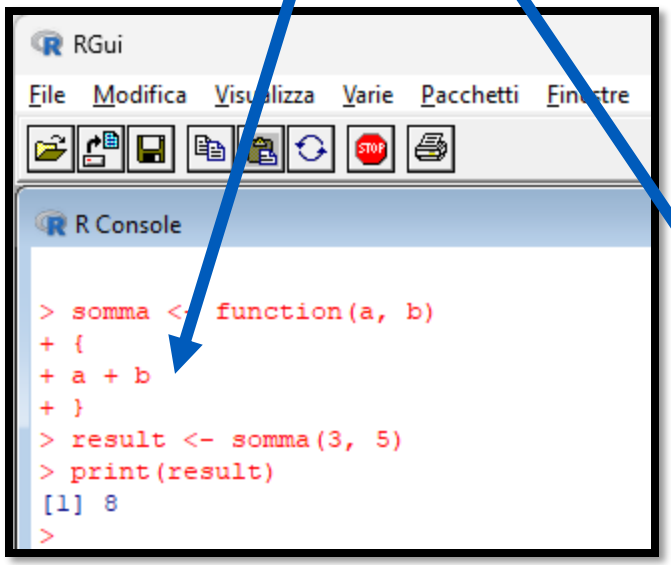
- In R le funzioni possono adottare un meccanismo di ***return*** implicito e uno esplicito:
  - se non è presente la keyword return, una funzione restituisce in output **il valore calcolato nell'ultima espressione** (return implicito);
  - **se è presente** l'istruzione `return(value)`, la funzione ritorna in output l'oggetto value che può essere un singolo oggetto oppure un insieme di oggetti.

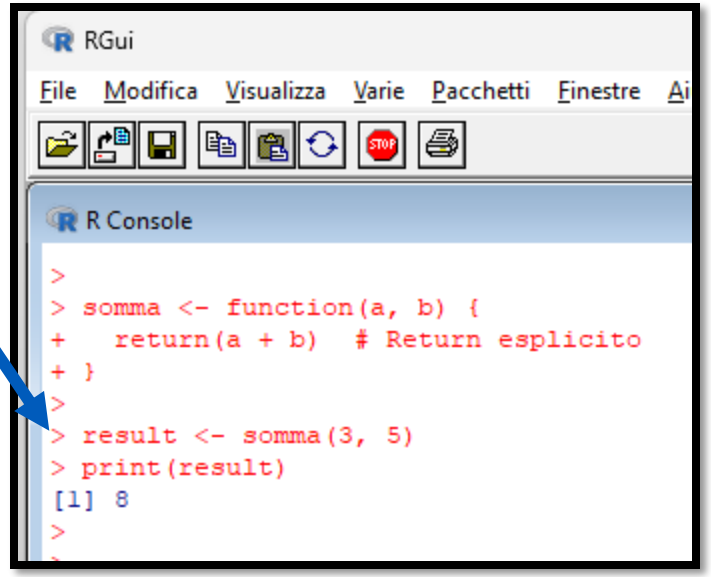


FUNZIONI IN R

Gestione dell'output di funzioni in R

- In R le funzioni possono adottare un meccanismo di ***return*** implicito e uno esplicito:
  - se non è presente la keyword return, una funzione restituisce in output **il valore calcolato nell'ultima espressione** (return implicito);
  - **se è presente** l'istruzione `return(value)`, la funzione ritorna in output l'oggetto value che può essere un singolo oggetto oppure un insieme di oggetti.

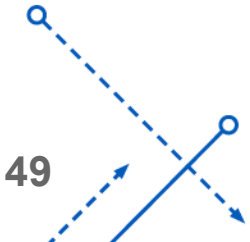




FUNZIONI IN R

Gestione di output multipli in R

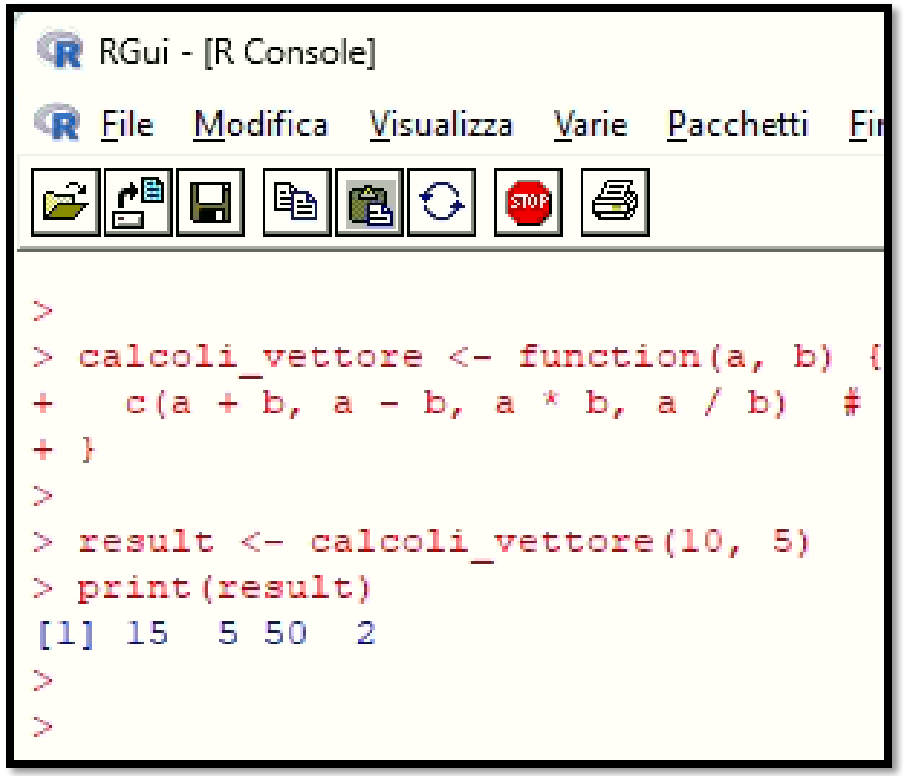
- In **R**, anche se per definizione una funzione deve restituire un singolo oggetto, esistono dei meccanismi che ci permettono di *restituire più valori contemporaneamente*.
  - Possiamo ad esempio racchiudere *più valori in strutture dati* come **liste**, **vettori**, **data frame (df)** o **matrici**.
  - Questi oggetti ci consentono di restituire output multipli in modo **organizzato** e accessibile.
    - **Curiosità:** Data la natura *anche* OOP di **R,** potremmo definire una classe «contenitore» e usarla per trasferire più oggetti in output.



FUNZIONI IN R

Restituire output multipli: Vettori

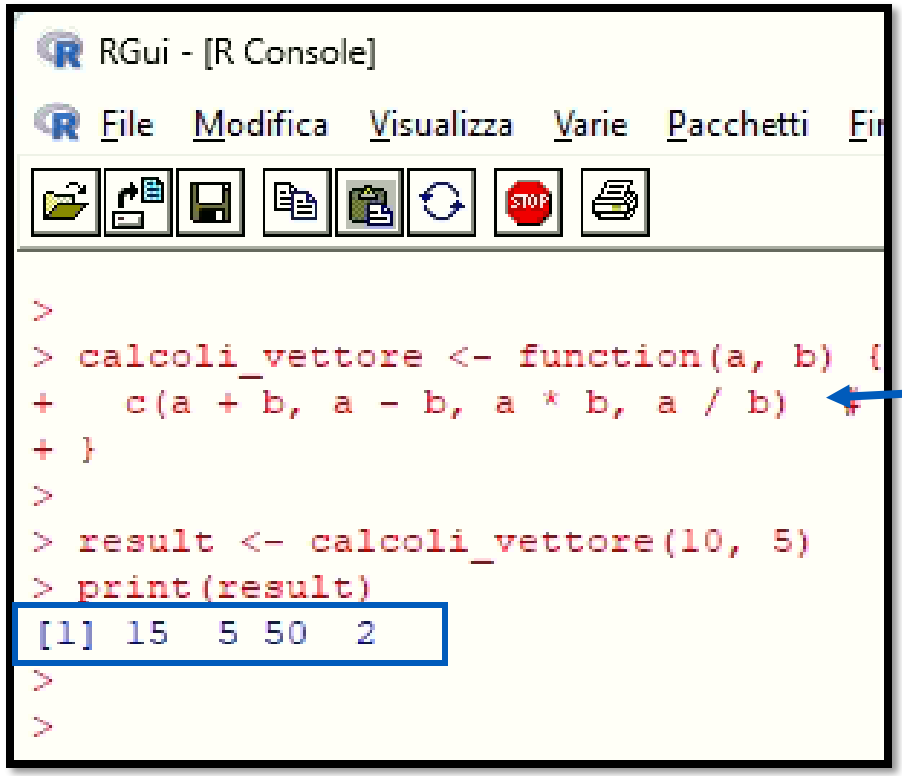
- In **R**, un oggetto di tipo **vettore può contenere solo elementi dello stesso tipo.** Usare i vettori è il modo più semplice per restituire output multipli.



FUNZIONI IN R

Restituire output multipli: Vettori

- In **R**, un oggetto di tipo **vettore può contenere solo elementi dello stesso tipo.** Usare i vettori è il modo più semplice per restituire output multipli.



Return implicito

FUNZIONI IN R

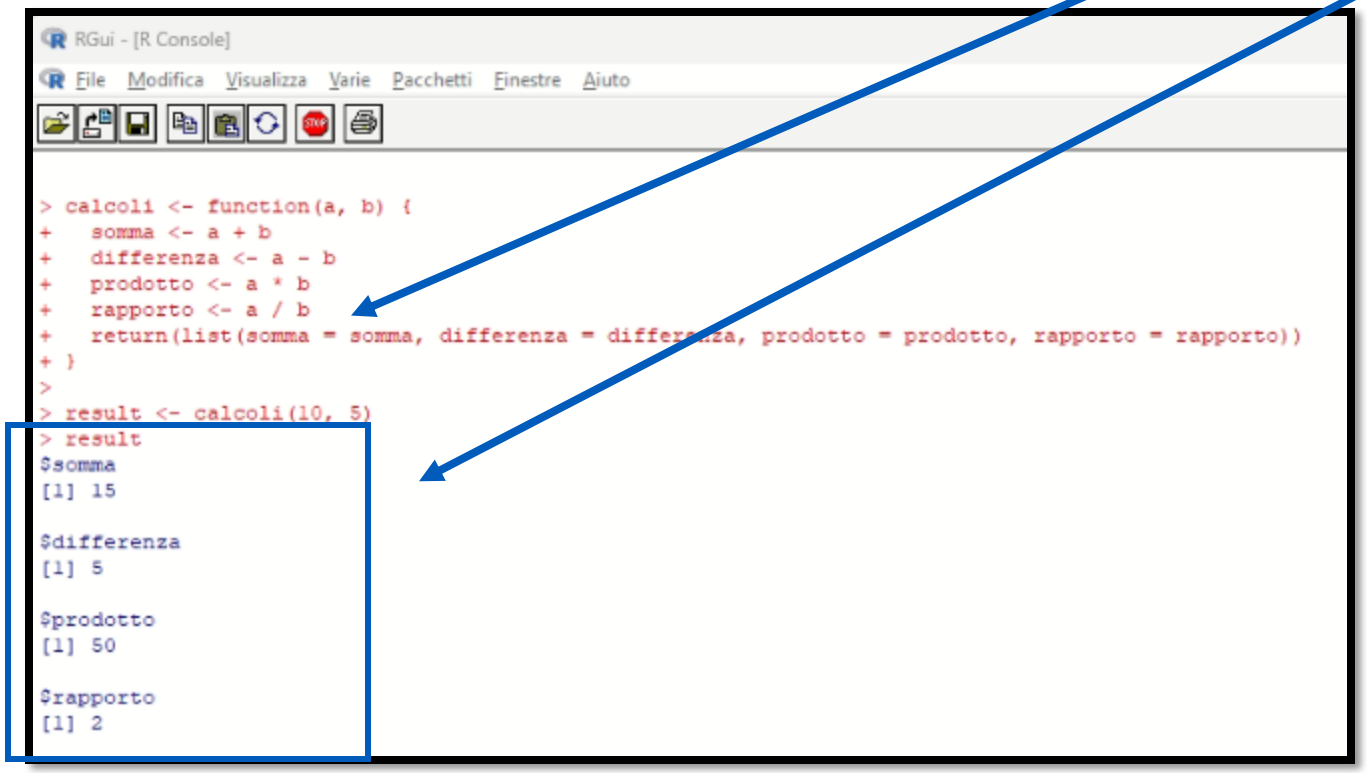
Restituire output multipli: Liste.

- In **R**, un oggetto di tipo **lista può contenere elementi (ognuno col suo nome) di qualsiasi tipo.** Possono anche essere utilizzate per «costruire» **oggetti complessi**.

FUNZIONI IN R

Restituire output multipli: Liste.

- In **R**, un oggetto di tipo **lista può contenere elementi (ognuno col suo nome) di qualsiasi tipo.** Possono anche essere utilizzate per «costruire» **oggetti complessi**.



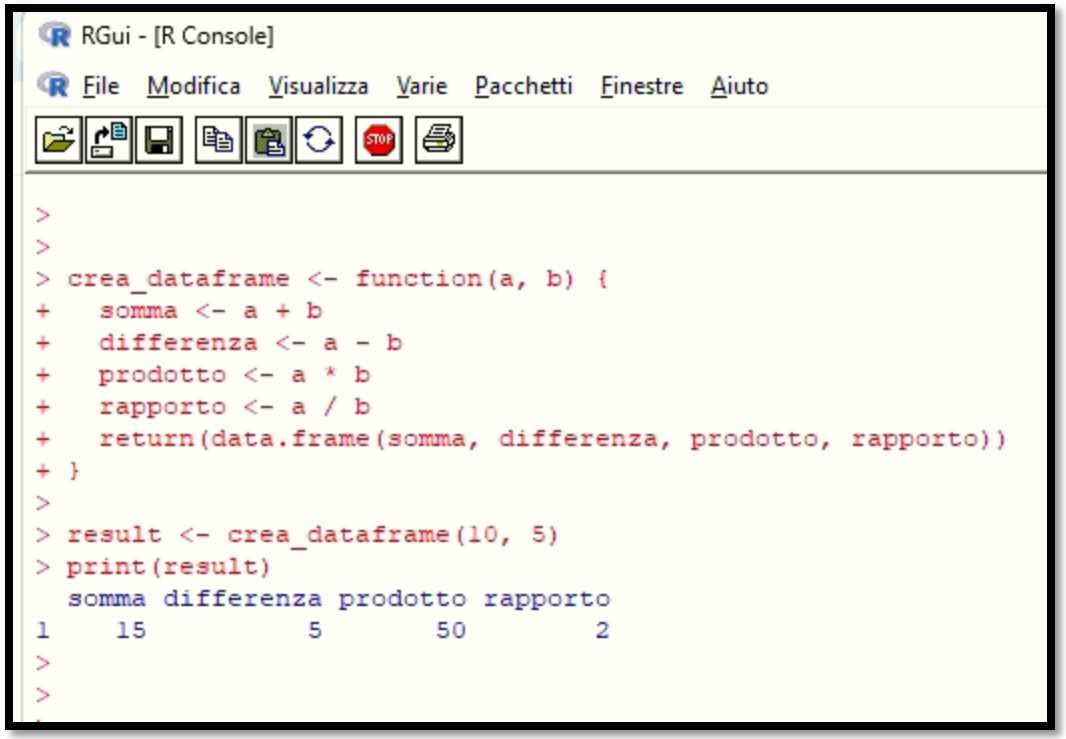
L'oggetto calcoli è una lista contenente quattro oggetti:

- calcoli$somma
- calcoli$differenza
- calcoli$prodotto
- calcoli$rapporto

FUNZIONI IN R

Restituire output multipli: Dataframe.

- In **R**, in un oggetto di tipo **Dataframe** le colonne *possono contenere dati di tipi differenti*. Questo tipo di oggetto può tornare utile quando si devono restituire risultati multipli che condividono una struttura comune (tabellare!)



STATISTICA E ANALISI DEI DATI

Capitoli 1.11/1.13 – Controllo del flusso in R

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026

CONTROLLO DEL FLUSSO IN R

Espressioni condizionali

In **R** esistono diversi costrutti e meccanismi che permettono di determinare l'ordine in cui vengono eseguite le istruzioni all'interno di un programma (la maggior parte sono i costrutti standard che già conoscete!):

- Condizionali (i classici if, then, else, switch);
- Operatori ed espressioni condizionali (and, or, not, …);
- Cicli di ripetizione (for, while, repeat, break, return).

CONTROLLO DEL FLUSSO IN R

Espressioni condizionali

- Nelle espressioni condizionali si può far uso di diversi operatori relazionali <, >, <=, >=, ==, ! = e logici NOT (!), AND (& e &&) e OR (| e ||).

Attenzione:

- Se si utilizzano gli operatori logici & e | il controllo **è eseguito termine a termine** tra gli elementi di due vettori e si ottiene un corrispondente vettore contenente i risultati logici via via ottenuti dal confronto degli elementi.

- Invece, **gli operatori logici && e || eseguono il controllo in sequenza** da sinistra a destra **esaminando così soltanto il primo elemento di ogni vettore.**

CONTROLLO DEL FLUSSO IN R

Espressioni condizionali

Il seguente esempio mostra che l'operatore & è applicato a tutti i singoli elementi del vettore mentre l'operatore && esegue il controllo soltanto sul **primo elemento**

```
> v<-c(-3,-2,-1,0,1,2,3)
>
> (v<=1)&(v>=-1)
[1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
>
> (v<=1)&&(v>=-1)
[1] FALSE
```

CONTROLLO DEL FLUSSO IN R

Strutture di selezione

- Le strutture di selezione messe a disposizione dal linguaggio R sono if e if . . . else.
- La sintassi usata in **R** per la prima struttura di selezione è:

```
if ( espressione condizionale ) {
   blocco di istruzioni da eseguire se l'espressione è vera
}
```

- Mentre per la seconda struttura di selezione è

```
if ( espressione condizionale ) {
  blocco di istruzioni da eseguire se l'espressione è vera
}
else {
   blocco di istruzioni da eseguire se l'espressione è false
}
```

CONTROLLO DEL FLUSSO IN R

Strutture di selezione / 2

- La parte di codice da eseguire dopo if e else **necessita di coppie di parentesi graffe nel caso siano presenti più istruzioni**.
- Le strutture di selezione possono avere altre strutture di selezione if e if . . . else annidate all'interno.
- Esiste una versione vettorizzata di if . . . else che ha la forma:

ifelse (espressione condizionale, a, b)

```
> v<-c(-3,-2,-1,0,1,2,3)
> a<-ifelse( (v<=1)&(v>=-1),1,0)
> a # visualizza a
[1] 0 0 1 1 1 0 0
```

con ifelse() viene valutata l'espressione condizionale per ogni elemento del vettore e si ottiene alla fine un vettore di 0 e 1 che **può essere assegnato al vettore a.**

STRUTTURE ITERATIVE

- Le strutture usate in R per realizzare costrutti iterativi sono simili a quelle disponibili in altri linguaggi di programmazione, ossia for, while e repeat.
- La sintassi della struttura for è la seguente:

```
for( variabile che assume valori in un insieme ) {
   corpo del ciclo
}
```

- Il corpo del ciclo può essere costituito da un'**unica istruzione** oppure da più istruzioni racchiuse tra una coppia di parentesi graffe
- Con il primo ed il secondo for i valori che la variabile n può assumere sono specificati da una sequenza di valori, mentre nel terzo caso sono gli elementi del vettore v.

```
> for(n in 1:4) print(n^2)
[1] 1
[1] 4
[1] 9
[1] 16
>
> for(n in seq(2,-4,by=-2)) print(n)
[1] 2
[1] 0
[1] -2
[1] -4
```

```
> for(v in c("sufficiente","buono","distinto","ottimo")) print(v)
[1] "sufficiente"
[1] "buono"
[1] "distinto"
[1] "ottimo"
```

STRUTTURE ITERATIVE

- Nell'esempio seguente usiamo la struttura iterativa for per scrivere una funzione che calcola il fattoriale di x, ossia valuta x!

```
> fact1<-function(x){
+ f<-1
+ if(x<2) return(1)
+ for(n in 2:x){
+       f<-f*n}
+ return (f)
+ }
>
> sapply(0:6,fact1)
[1]   1   1   2   6  24 120 720
```

- Nell'esempio è stata usata la funzione sapply(v, funz) che permette di applicare la funzione indicata nel parametro funz a tutti gli elementi di una sequenza o di un vettore v restituendo un vettore di valori

WHILE

- La sintassi della struttura while è:

```
> while( condizione ) {
    corpo del ciclo
    }
```

- che opera attivando l'esecuzione delle istruzioni indicate nel corpo del ciclo fino a quando la condizione espressa tra le parentesi tonde è verificata.

- Un esempio di applicazione della struttura while è il seguente:

```
> n<-1 # condizione iniziale
> while(n<5){
+ print(n^2)
+ n<-n+1
+ }
[1] 1
[1] 4
[1] 9
[1] 16
```

```
> n<-1
> v<-vector(length=0) # oggetto iniziale
> while(n<5){
+ v[length(v)+1]<-n^2
+ n<-n+1
+ }
>
> v # visualizza il vettore v
[1]  1  4  9 16
```

REPEAT

- La sintassi della struttura repeat è la seguente:

  > **repeat** { corpo del ciclo }

- Se si utilizza questa struttura di iterazione, la terminazione della ripetizione del ciclo deve essere espressamente indicata all'interno delle istruzioni che costituiscono il corpo del ciclo.
- Il comando break può essere usato per uscire dal ciclo in cui questa istruzione è racchiusa ed è il solo modo per terminare un ciclo con repeat.

```
> fact3<-function(x){
+ f<-1
+ y<-x
+ repeat{
+ if(y<2) break
+ f<-f*y
+ y<-y-1}
+ return(f)
+ }
>
> sapply(0:6,fact3)
[1]   1   1   2   6  24 120 720
```

- R passa l'intera copia dell'oggetto alle funzioni invece di passare un riferimento all'oggetto, come accade in altri linguaggi di programmazione.
- In R è quindi sconsigliata la programmazione di cicli iterativi tramite una qualsiasi istruzione che fa usa esplicito di cicli a causa della **lentezza** con cui le iterazioni vengono realizzate dal sistema dovuta alla sequenza di copie e assegnazioni ripetute un numero elevato di volte

CONTROLLO DEL FLUSSO IN R

- **Quando è possibile, si consiglia l'uso di forme alternative per realizzare i cicli che si appoggiano su funzioni, messe a disposizione dal linguaggio R e che lavorano vettorialmente.**
- Ad esempio per calcolare x! per x = 1, 2, . . . , 6 basta utilizzare la funzione prodotto cumulativo **cumprod**() nel seguente modo:

```
> cumprod(1:6)
[1]   1   2   6  24 120 720
```

- Una funzione che calcola x! si può quindi così scrivere:

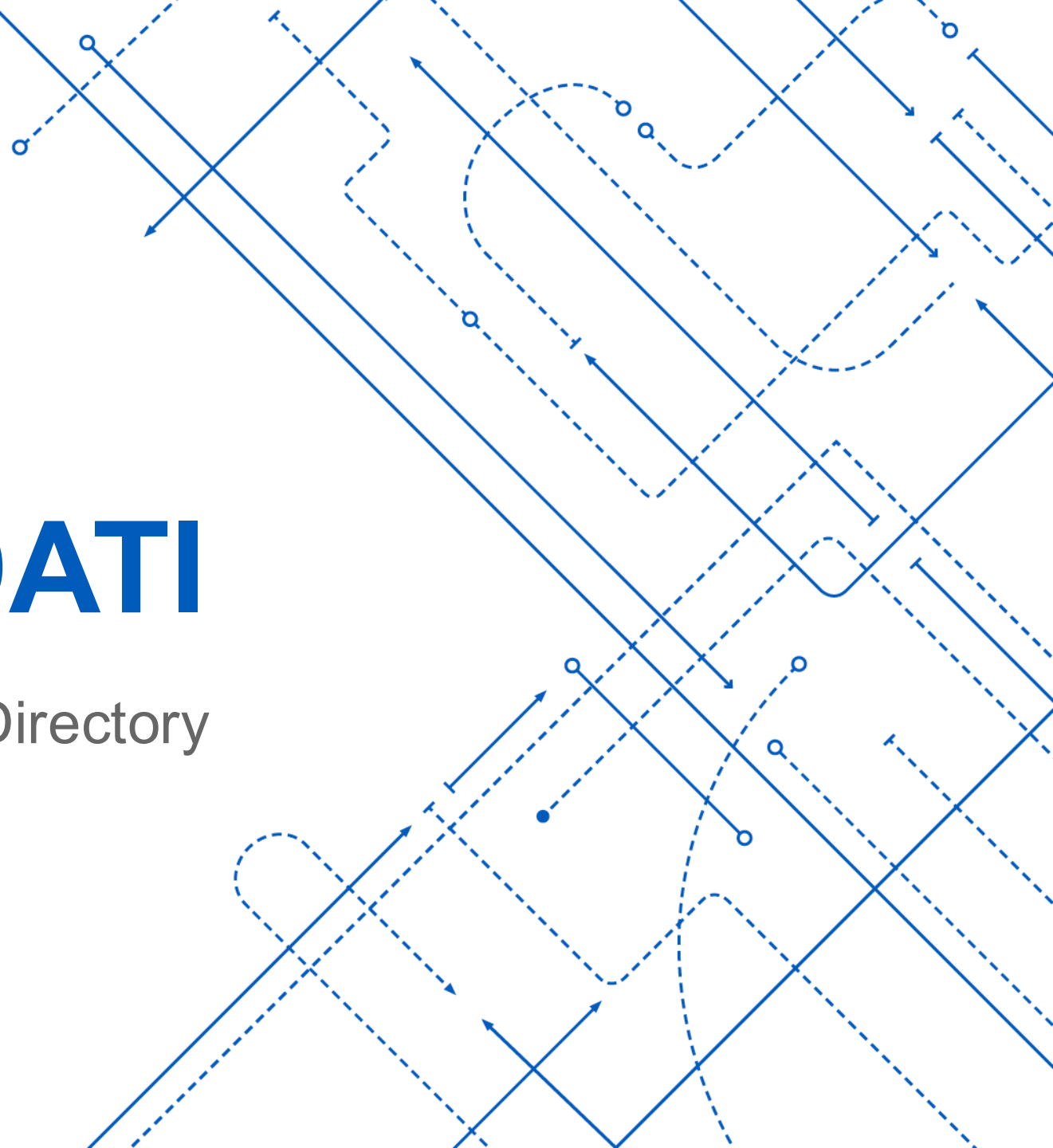
```
> fact4<-function(x) max( cumprod(1:x) )
>
> sapply(0:6,fact4)
[1]   1   1   2   6  24 120 720
```

STATISTICA E ANALISI DEI DATI

Capitolo 1.14 – Script, Output e Directory

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2025-2026



SCRIPTING

Definizione di nuovi script

Spesso potrebbe essere necessario scrivere procedure, funzioni e insiemi di comandi (o magari interi pacchetti o software) progettati dall'utente da utilizzare in diverse sessioni di R (o da distribuire online):

Pensate, per esempio, ad una sperimentazione statistica con la quale avete appena fatto una nuova scoperta sorprendente e che volete proporre per la pubblicazione:

→ La sperimentazione dovrà essere riproducibile e di conseguenza l'uso di uno script può aiutarvi a fornire direttamente tutti i passaggi da eseguire ai vostri peer-review.

In **R**, **A tal fine è necessario inserire queste procedure in un file di testo:**

- Nel menù è presente una voce che permette di aprire una finestra in cui scrivere le procedure che si desiderano.

SCRIPTING

Come salvare e ricaricare uno script

- Supponiamo che abbiate scoperto come calcolare il fattoriale in R:

```
> # funzione per il calcolo dei fattoriali
> fact4<-function(x) max( cumprod(1:x) )
```

- A questo punto, volete rendere disponibile il vostro lavoro ad altre persone.
- Per fare ciò, è sufficiente salvare tale file con un nome (ad esempio *nuoveScoperte*.R) nella directory corrente dell'utente.
- Un altro sviluppatore potrà utilizzare le vostre funzioni semplicemente installando il vostro pacchetto (*nuoveScoperte*.R) nella sua working directory e caricarlo in memoria utilizzando la funzione **source**().

```
> source("funzioniNuove.R")
> sapply(1 : 6, fact4)
[1] 1 2 6 24 120 720
```

SCRIPTING

Memorizzare l'output di una sessione

- **Attenzione** a non confondere con l'output di una sessione con il contenuto del vostro workspace (gli oggetti)
- Alcune volte si rende necessario poter memorizzare l'output di una sessione di R utilizzando la funzione **sink**(*nomefile.R*)
- Ad esempio, la sequenza di comandi

```
sink("outputFunzioniNuove.R")
fact4 <- function(x) max( cumprod(1:x) )
sapply(1 : 8, fact4)
sink()
```

- Crea un file *outputFunctioniNuove*.R nella directory corrente dell'utente in cui è inserito il testo

```
[1] 1  2  6  24  120  720  5040  40320
```

che può essere successivamente visualizzato.

- La funzione **sink**() serve per chiudere il file di output.

L'AMBIENTE INTEGRATO R

Output di sessione e Cronologia dei comandi (History)

- Il menù consente di salvare l'area di lavoro su un file di testo outputSessione.history contenente l'intero elenco dei comandi utilizzati in una sessione e che può essere di aiuto nella creazione degli script.
- Da un file outputSessione.history è anche possibile caricare le istruzioni, che potranno essere richiamate dalla console di R.
- In generale la directory può essere differente da quella in cui si trova l'eseguibile di R.
- È possibile verificare la directory corrente di lavoro attraverso la funzione getwd() e visionarne il contenuto attraverso la funzione dir() che mostra i nomi dei file e delle directory contenute all'interno della directory corrente.
- Poiché spesso si utilizzano differenti directory di lavoro, per spostarsi da una directory all'altra si utilizzano i comandi.

```
> wd < −"C : \\Magistrale\\SAD\\ProgettoSAD"
> setwd(wd)
```

L'AMBIENTE INTEGRATO R

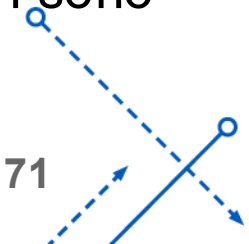
Importazione di dati in R

Quando eseguiamo analisi statistiche ad un certo punto avremo la necessità di importare dati che potrebbero essere resi disponibili in file creati da un qualsiasi altro programma.

In **R**, è disponibile la funzione read.table() che permette di:

- leggere dati tabulari da file di testo, come file CSV (Comma-Separated Values), TSV (Tab-Separated Values)
- Importare dati da file di testo che contengono **dati organizzati in righe e colonne.**

- La funzione readtable è definita come: `read.table(file, header = FALSE, sep = "", ...)`:
  - file: il percorso del file che si vuole leggere. Può essere un file locale o una URL.
  - header: un valore logico (TRUE o FALSE) che indica se la prima riga del file contiene i nomi delle colonne. Se header = TRUE, la prima riga sarà trattata come i nomi delle colonne. Il valore predefinito è FALSE.
  - sep: il separatore tra i valori all'interno del file. Ad esempio: sep = "," per file CSV (valori separati da virgola), sep = "\t" per file TSV (valori separati da tabulazione), Se i valori sono separati da spazi o altri caratteri, si specifica il separatore appropriato.



L'AMBIENTE INTEGRATO R

Importazione di dati in R

- La funzione readtable è definita come: `read.table(file, header = FALSE, sep = "", ...)`:
  - **file** è il percorso del file che si vuole leggere. Può essere un file locale o una URL;
  - **header** è un valore logico (TRUE o FALSE) che indica se la prima riga del file contiene i nomi delle colonne. Se header = TRUE, la prima riga sarà trattata come i nomi delle colonne. Il valore predefinito è FALSE;
  - **sep** è il separatore tra i valori all'interno del file (, = CSV) (\t = TSV).
  - **stringsAsFactors**: se TRUE (sconsigliato) importa i dati di tipo carattere e li converte automaticamente in fattori (categorici).
  - **colClasses**: permette di specificare le classi dei dati per ogni colonna, utile per migliorare l'efficienza del caricamento quando si conoscono in anticipo i tipi di dati.
  - **nrows**: indica quante righe leggere dal file. Utile per leggere solo una parte del file.

Attenzione a sep! Se i valori sono separati da spazi o altri caratteri, è necessario sepcificare il separatore appropriato.

Importazione di dati in R /2

- Se la prima riga del file contiene l'intestazione delle variabili, allora read.table() interpreterà la prima riga del file come una riga in cui sono contenuti i nomi delle variabili, assegnando ciascun nome alle variabili (colonne) del data frame.

- Un esempio:

```
tabella <- read.table(file = "Tabella.txt", header = TRUE, sep = " ", na.strings = "NA", dec = ".")
```

L'argomento header = TRUE specifica che nella prima riga del file Tabella.txt sono contenuti i nomi delle variabili.

L'argomento sep = " " indica che i diversi campi sono separati da uno spazio.

Inoltre l'argomento na.strings = "NA" è utile se nel file importato sono presenti valori mancanti individuati da NA ed infine l'argomento dec = "." specifica il tipo di carattere utilizzato nel file per separare i numeri decimali.

L'AMBIENTE INTEGRATO R

Importazione di dati in R /2

- Dopo l'importazione tramite read.table è possible trasformare la tabella può in una matrice attraverso l'istruzione:

```
dataMatrix <- as.matrix(tabella)
```

- Su questa matrice dei dati si può successivamente iniziare ad operare utilizzando il linguaggio R.

DOMANDE?