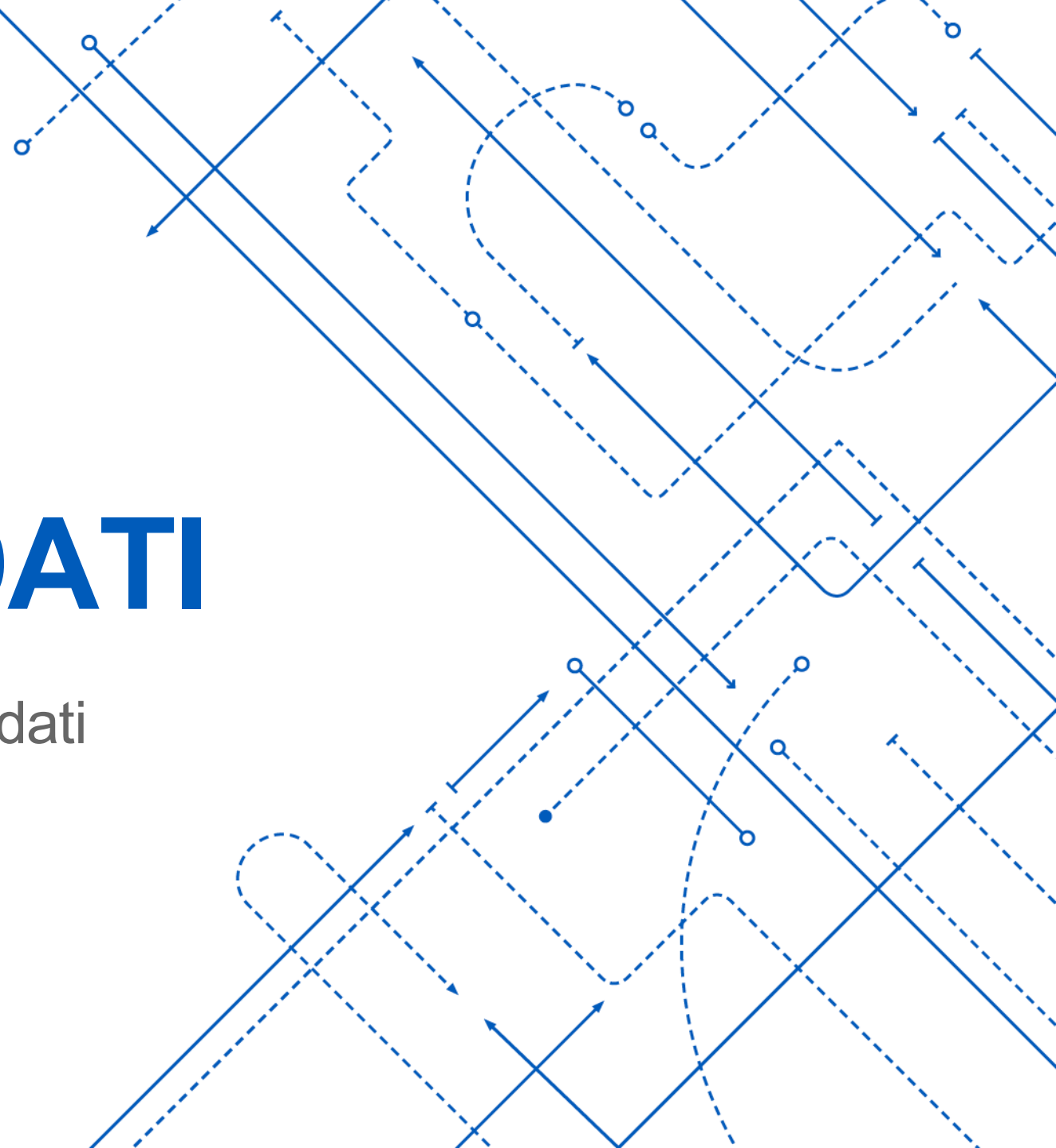
STATISTICA E ANALISI DEI DATI

Capitolo 1 – Funzioni e Strutture dati

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

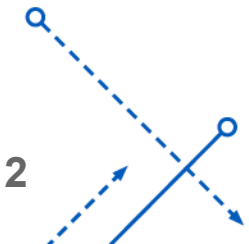
a.a. 2025-2026



FUNZIONI IN R

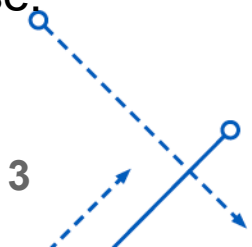
Breve recap: Funzioni

- Una funzione è un **blocco di codice** riutilizzabile e che esegue una specifica operazione.
- Per poter parlare di funzione è necessario che siano definiti almeno **input** e **output** della stessa per poterla poi considerare una black-box:
  - **Curiosità**: Il «*come vengono implementati i meccanismi di input e output*» dipende dal linguaggio di programmazione:
    - ad esempio, in C l'output può essere implementato *anche* sfruttando il passaggio di *argomenti per riferimento (i puntatori!)* anziché per valore;
- In **R**, una funzione può essere definita in diversi modi, poiché R stesso è **multi-paradigma**.
- In **R**, una funzione **accetta zero o più input** e restituisce **almeno un output** (che è NULL nel caso la funzione sia vuota).



PROGRAMMAZIONE FUNZIONALE

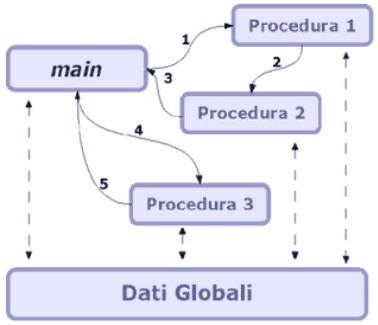
- **Cos'è la Programmazione Funzionale?**
  - La **programmazione funzionale** è un paradigma di programmazione che si basa sull'uso di **funzioni come entità di prima classe**
  - In R, le funzioni possono essere:
    - Passate come argomenti ad altre funzioni
    - Restituite da altre funzioni
    - Memorizzate in variabili
- Caratteristiche della Programmazione Funzionale:
  - Funzioni come Entità di Prima Classe: Le funzioni possono essere manipolate come dati.
  - Immutabilità: Le variabili non vengono modificate, ma si creano nuove variabili.
  - Funzioni Pure: Funzioni senza effetti collaterali, cioè il risultato dipende solo dagli input.
  - Composizione di Funzioni: Funzioni combinate insieme per formare operazioni più complesse.



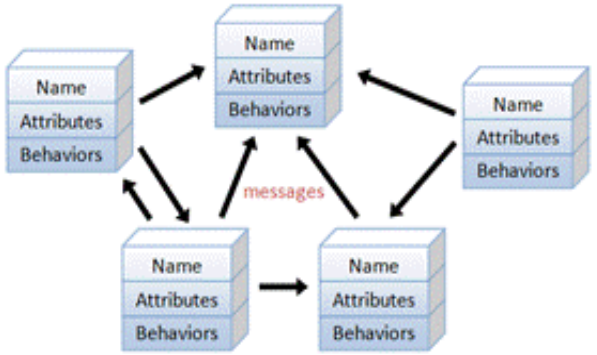
Funzioni in R

Flashback: In che modo si programma R (e non in R!)

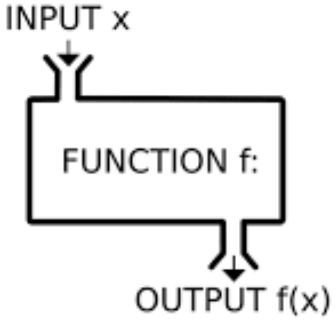
- R dispone offre un *linguaggio di interazione multi-paradigma e mette a disposizione caratteristiche funzionali, OOP e di* ***programmazione funzionale****.*



Procedurale



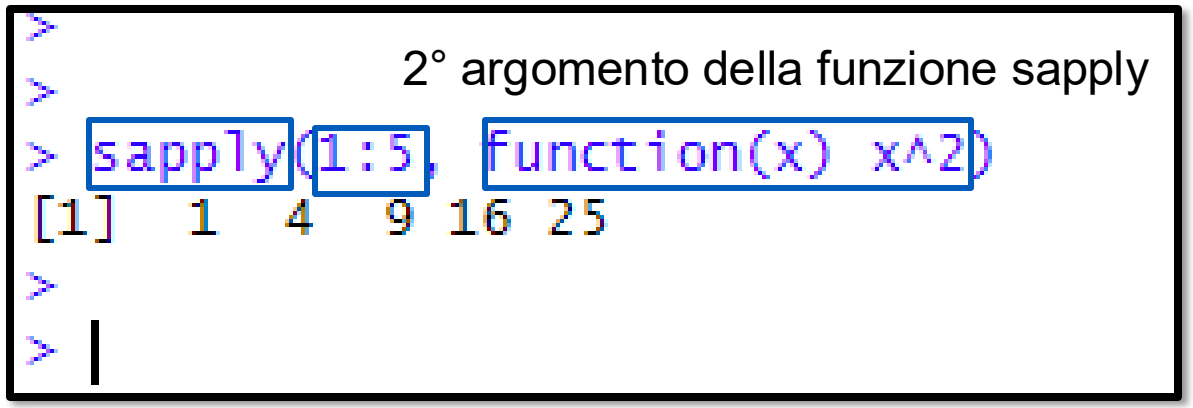
OOP



Funzionale

FUNZIONI ANONIME

- R è contestualmente *fortemente* orientato alla **programmazione funzionale**:
  - Supporta le **funzioni anonime**;
  - Le funzioni sono entità di prima classe *(possono essere annidate, assegnate a variabili,* ***passate come argomenti*** *o ancora restituite da altre funzioni):*
    - Alcune funzioni **consentono di applicare funzioni** su **intere strutture di dati** (come vettori o liste) senza la necessità di espliciti cicli.



FUNZIONI

- Indipendentemente dalle funzioni presenti nei pacchetti, l'utente può scrivere una propria funzione seguendo delle regole di programmazione
- Per definire una funzione in R si usa il comando:

  function(lista parametri)

- Ad esempio:

  function(x) x+2

  - è la funzione che prende un oggetto x in input e restituisce x+2
    - La funzione in se non ha un nome, perché una funzione è un valore come un intero o un booleano (è appunto un tipo primitivo!)
- Se vogliamo assegnare una funzione ad un oggetto "funz", scriveremo

  funz <- **function**()

```
>function(x) x + 2
```

```
##
function(x)
x + 2
```

```
>funz = function(x) x + 2
>funz
```

```
##
function(x)
x + 2
```

```
>funz(10)
```

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

```
# c() definisce un vettore
>funz(c(3, 5))
```

```
##
[1] 5
7
```

```
# funzione anonima
>(function(x) x + 2)(c(3, 5))
```

```
##
[1] 5
7
```

FUNZIONI

- Una **funzione** in R è un blocco di codice che può essere richiamato e riutilizzato
- Le funzioni in R possono prendere **argomenti in input** e restituire **valori in output**

```r
#Funzioni
# Definire una funzione che somma due numeri
somma <- function(a, b) {
  return(a + b)
}

# Richiamare la funzione
risultato <- somma(5, 3)
risultato
```

```
8
```

FUNZIONI

- Una **funzione** in R è un blocco di codice che può essere richiamato e riutilizzato
- Le funzioni in R possono prendere **argomenti in input** e restituire **valori in output**
- Alcuni argomenti in input possono avere un valore di default come parametro

```r
#Funzioni
# Definire una funzione che somma due numeri
somma <- function(a, b) {
  return(a + b)
}

# Richiamare la funzione
risultato <- somma(5, 3)
risultato
```

```
8
```

```r
# Funzione con valore di default per un argomento
saluta <- function(nome = "mondo") {
  return(paste("Ciao", nome))
}

saluta()
saluta("Alice")
```

```
'Ciao mondo'
'Ciao Alice'
```

PROGRAMMAZIONE FUNZIONALE

- Passare una Funzione come Argomento:

```r
# Funzione Come Argomento
# Funzione che applica una funzione ad ogni elemento di un vettore
applica_funzione <- function(f, x) {
  sapply(x, f)
}

# Funzione anonima come argomento
v = 1:10
risultato_quadrato <- applica_funzione(function(x) x^2, v)
risultato_cubo <- applica_funzione(function(x) x^3, v)

#Stampa risultati
v
risultato_quadrato
risultato_cubo
```

1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10

1 · 4 · 9 · 16 · 25 · 36 · 49 · 64 · 81 · 100

1 · 8 · 27 · 64 · 125 · 216 · 343 · 512 · 729 · 1000

PROGRAMMAZIONE FUNZIONALE

- Passare una Funzione come Argomento:

```r
# Funzione Come Argomento
# Funzione che applica una funzione ad ogni elemento di un vettore
applica_funzione <- function(f, x) {
  sapply(x, f)
}

# Funzione anonima come argomento
v = 1:10
risultato_quadrato <- applica_funzione(function(x) x^2, v)
risultato_cubo <- applica_funzione(function(x) x^3, v)

#Stampa risultati
v
risultato_quadrato
risultato_cubo
```

1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10

1 · 4 · 9 · 16 · 25 · 36 · 49 · 64 · 81 · 100

1 · 8 · 27 · 64 · 125 · 216 · 343 · 512 · 729 · 1000

- Restituire una Funzione:

```r
# Funzione che restituisce un'altra funzione
crea_funzione <- function(moltiplicatore) {
  return(function(x) x * moltiplicatore)
}

# Creare una funzione che moltiplica per 3
moltiplica_per_3 <- crea_funzione(3)
moltiplica_per_3(5)
```

15

FUNZIONI IN R

La famiglia di funzioni apply

La famiglia di funzioni apply è progettata per facilitare l'applicazione di operazioni su insiemi di dati come matrici, array, liste e data frame:

- riducono l'uso esplicito dei cicli for;
- migliorano la leggibilità e la compattezza del codice.

FUNZIONI IN R

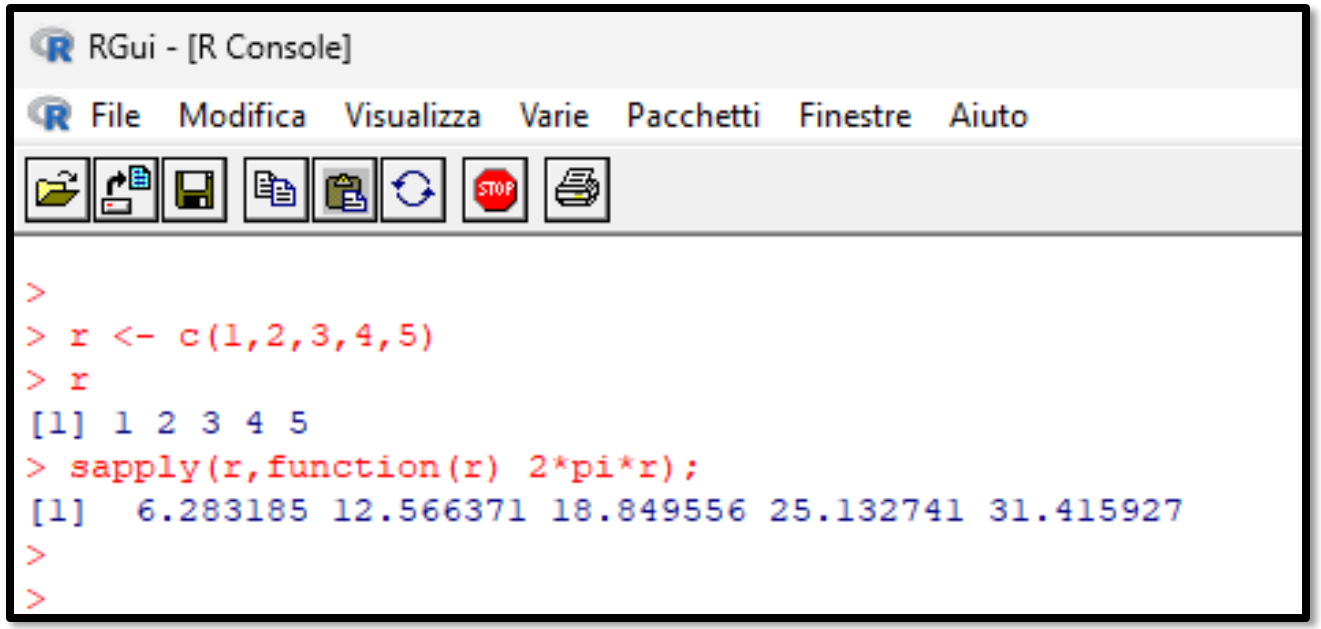
La famiglia di funzioni apply

- **apply**() viene utilizzata per applicare una funzione su righe o colonne di una matrice o array.
- **lapply**() viene utilizzata per applicare una funzione a ogni elemento di una lista o di un vettore e restituisce una lista.
- **sapply**() è simile a **lapply**(), ma semplifica il risultato in un vettore o una matrice (se possibile), invece di restituire sempre una lista.
- **vapply**() è una variante di **sapply**() che richiede che venga specificato il tipo di output atteso, rendendolo più sicuro rispetto a **sapply**().
- **mapply**() è una funzione multivariata che applica una funzione a più liste o vettori contemporaneamente.
- **tapply**() applica una funzione su un vettore in base a sottogruppi definiti da un fattore o una variabile categoriale.
- **rapply**() è una versione ricorsiva di **lapply**(), che permette di applicare una funzione agli elementi di una lista anche se essa contiene altre liste (annidate).

FUNZIONI IN R

La funzione sapply

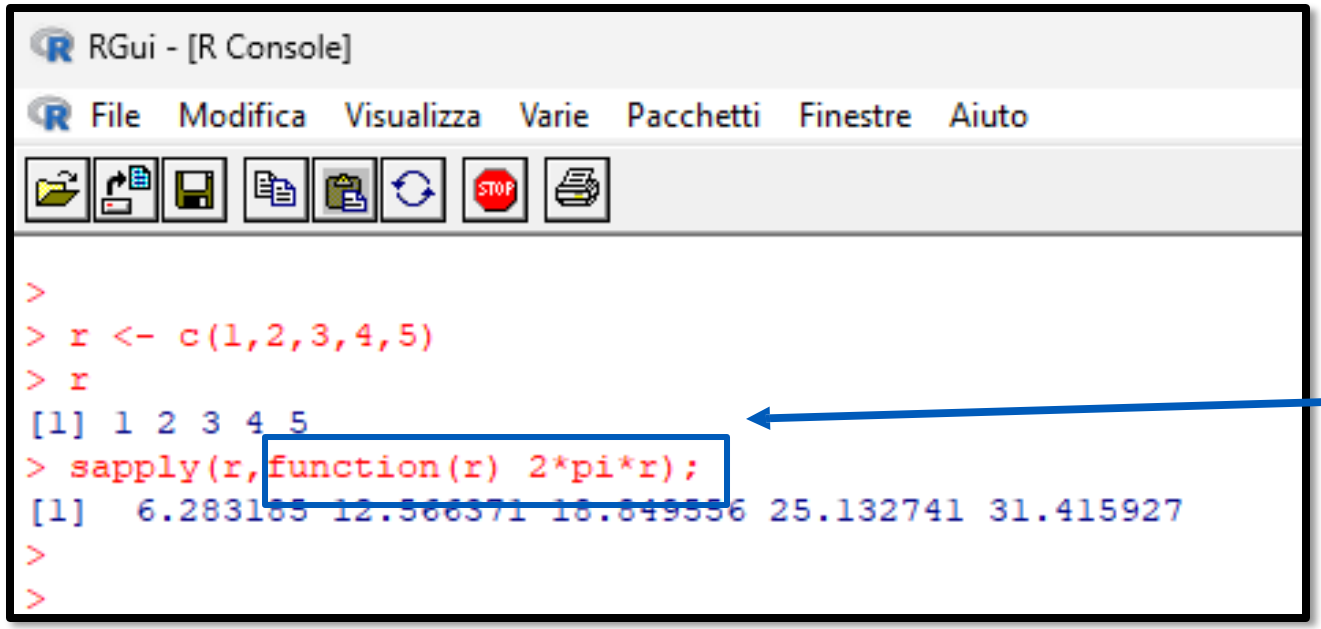
- La funzione **sapply** è definita come **sapply**(X, FUN, ..., simplify = TRUE)
- In **R** si utilizza la funzione sapply per applicare una ***funzione FUN*** a tutti gli elementi di una sequenza, lista o di un **vettore X**.



FUNZIONI IN R

La funzione sapply

- La funzione **sapply** è definita come **sapply**(X, FUN, ..., simplify = TRUE)
- In **R** si utilizza la funzione sapply per applicare una ***funzione FUN*** a tutti gli elementi di una sequenza, lista o di un **vettore X**.



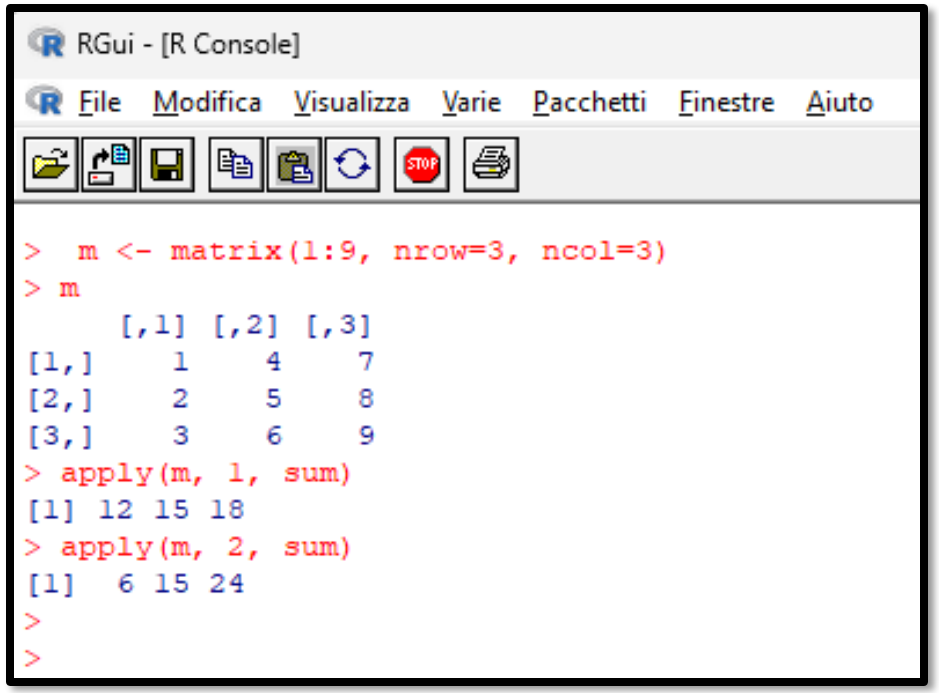
Attenzione:
Ricordate a cosa servono i (…) ?
Nella sapply vengono usati nel caso sia necessario passare parametri opzionali alla funzione FUN

Attenzione:
Che tipo di funzione è questa?

FUNZIONI IN R

La funzione apply

- La funzione **apply** è definita come **apply**(X, MARGIN, FUN, ...) **in cui** X è una matrice o un array, **MARGIN** specifica se applicare la funzione per righe (MARGIN=1) o per colonne (MARGIN=2) mentre **FUN**: è la funzione da applicare.



FUNZIONI IN R

La funzione apply /2

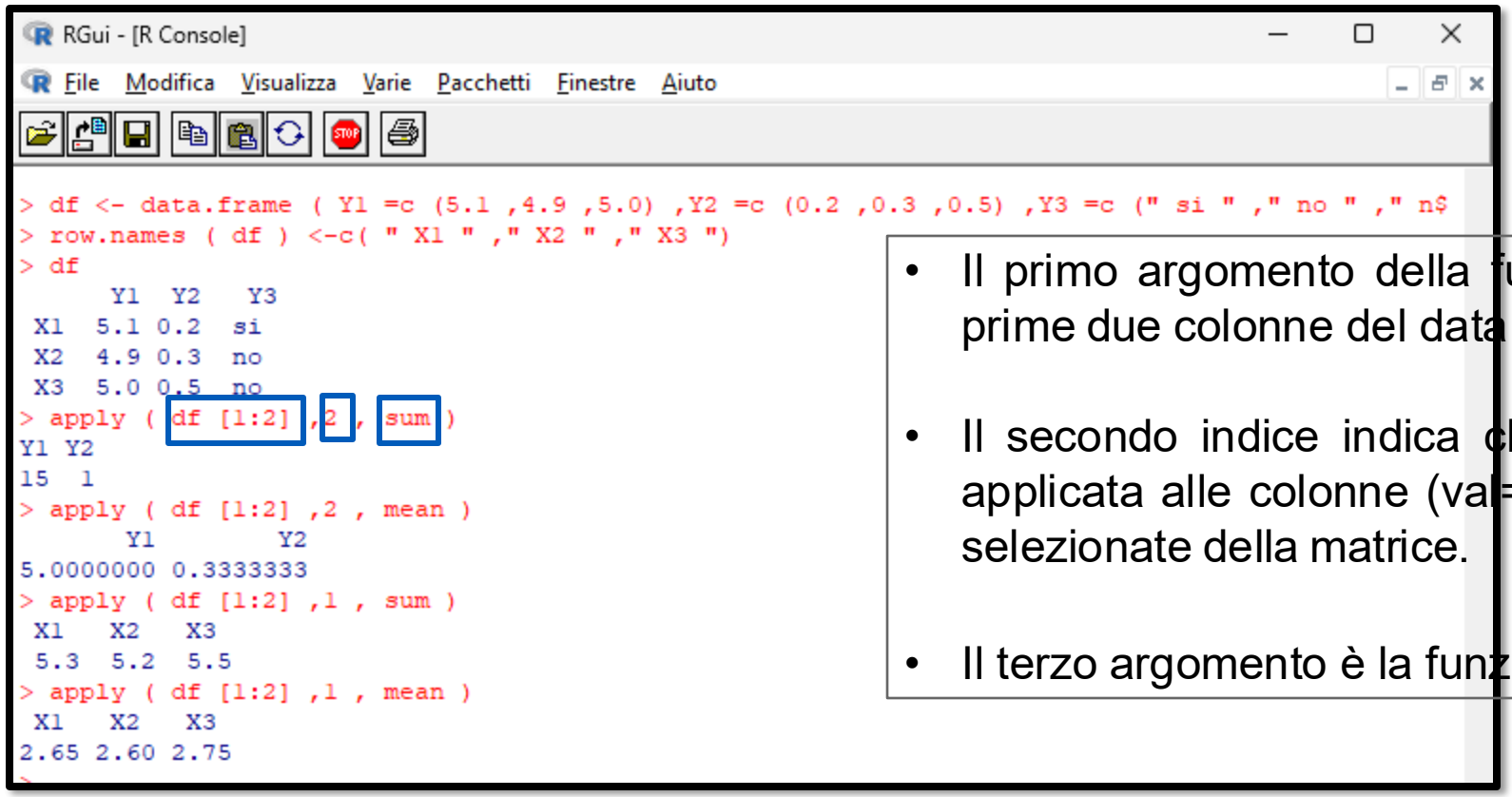
- Applichiamo la funzione **apply**() ad una matrice calcolando il valore minimo delle righe ed il valore massimo delle colonne.

```
> a <- matrix (c(5, 2, 7, 1, 2, 8, 4, 5, 6), nrow=3, ncol=3)
> rownames(a)<-c("X1","X2","X3")
> colnames(a)<-c("Y1","Y2","Y3")
> a # visualizza la matrice a
   Y1 Y2 Y3
X1  5  1  4
X2  2  2  5
X3  7  8  6
>
> apply(a,1,min)
X1 X2 X3
 1  2  6
 >
> apply(a,2,max)
Y1 Y2 Y3
 7  8  6
```

FUNZIONI IN R

La funzione apply /3

- Calcoliamo ora la somma e la media aritmetica degli elementi delle prime due colonne del seguente data frame:



- Il primo argomento della funzione apply() seleziona le prime due colonne del data frame df
- Il secondo indice indica che la funzione deve essere applicata alle colonne (val=2) oppure alle righe (val=1) selezionate della matrice.
- Il terzo argomento è la funzione da utilizzare.

FUNZIONI IN R

Da ricordare

- **apply()** viene utilizzata principalmente per applicare una funzione a una matrice o a un array multidimensionale.

- **sapply()** è una versione semplificata di lapply che restituisce un vettore o una matrice se possibile. È utilizzata principalmente per applicare una funzione a ciascun elemento di una lista o di un vettore.

APPLY E LAPPLY

- apply(): Applica una funzione su righe o colonne di una matrice o un array
- È utile quando si desidera eseguire calcoli su ogni riga o colonna di una matrice senza usare cicli espliciti

apply(X, MARGIN, FUN, ...)

  - X: Matrice o array.
  - MARGIN:
    - 1 per le righe
    - 2 per le colonne
  - FUN: Funzione che si desidera applicare (ad esempio, sum(), mean(), ecc.)

APPLY E LAPPLY

- apply(): Applica una funzione su righe o colonne di una matrice o un array

```r
# Creare una matrice
matrice <- matrix(1:9, nrow = 3, byrow = TRUE)
matrice
```

A matrix:

3 × 3 of

type int

1	2	3
4	5	6
7	8	9

```r
# Applicare la funzione sum() su ogni riga
somme_righe <- apply(matrice, 1, sum)  # Somma gli elementi di ogni riga
somme_righe
```

6 · 15 · 24

APPLY E LAPPLY

- apply(): Applica una funzione su righe o colonne di una matrice o un array

```r
# Creare una matrice
matrice <- matrix(1:9, nrow = 3, byrow = TRUE)
matrice
```

A matrix:

3 × 3 of

type int

1	2	3
4	5	6
7	8	9

```r
# Applicare la funzione sum() su ogni riga
somme_righe <- apply(matrice, 1, sum)  # Somma gli elementi di ogni riga
somme_righe
```

6 · 15 · 24

```r
# Applicare la funzione mean() su ogni colonna
media_colonne <- apply(matrice, 2, mean)  # Calcola la media di ogni colonna
print(media_colonne)
```

```
[1] 4 5 6
```

APPLY E LAPPLY

- lapply(): è usata per applicare una funzione a ciascun elemento di una lista o di un vettore
- È utile per iterare su liste di qualsiasi tipo, non limitandosi a liste numeriche, ma anche su liste che contengono vettori, matrici o altri oggetti

lapply(X, FUN, ...)

  - X: Lista o Vettore.
  - FUN: Funzione che si desidera applicare (ad esempio, sum(), mean(), ecc.)

- L'output è sempre una **Lista**

APPLY E LAPPLY

- lapply(): è usata per applicare una funzione a ciascun elemento di una lista o di un vettore
- È utile per iterare su liste di qualsiasi tipo, non limitandosi a liste numeriche, ma anche su liste che contengono vettori, matrici o altri oggetti

lapply(X, FUN, ...)

  - X: Lista o Vettore.
  - FUN: Funzione che si desidera applicare (ad esempio, sum(), mean(), ecc.)

- L'output è sempre una **Lista**

```r
# Creare una lista
lista <- list(a = 1:3, b = 4:6, c = 7:9)
lista
```

```
$a
1 · 2 · 3

$b
4 · 5 · 6

$c
7 · 8 · 9
```

```r
# Applicare una funzione che moltiplica per 2 ciascun elemento della lista
risultato <- lapply(lista, function(x) x * 2)
risultato
```

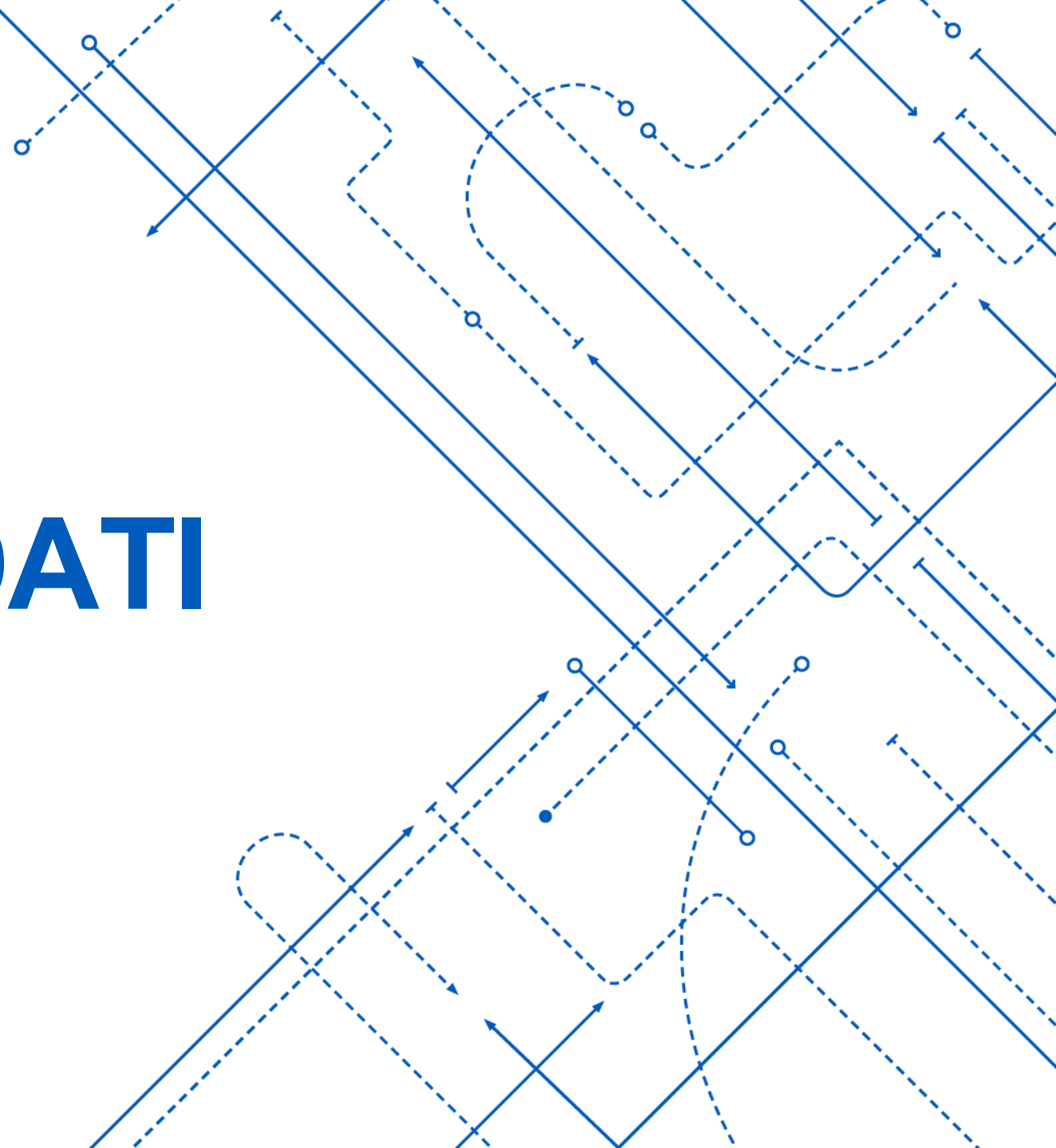
```
$a
2 · 4 · 6

$b
8 · 10 · 12

$c
14 · 16 · 18
```

STATISTICA E ANALISI DEI DATI

Vettori

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2024-2025



VETTORI

Vettori

In R i vettori sono oggetti che mantengono al loro interno un insieme indicizzato di elementi **tutti dello stesso tipo**

Classi di vettori

- **character**
- **numeric**
- **integer**
- **logical**
- **complex**

VETTORI

Vettori

In R i vettori sono oggetti che mantengono al loro interno un insieme indicizzato di elementi **tutti dello stesso tipo**

Classi di vettori

- **character**
- **numeric**
- **integer**
- **logical**
- **complex**

Inizializzare vettore

[19]:
```
# Vettore di numeri interi
vettore_interi <- c(1, 2, 3, 4, 5)

# Vettore di numeri decimali
vettore_decimali <- c(1.1, 2.2, 3.3, 4.4)

print(vettore_interi)
print(vettore_decimali)
```

```
[1] 1 2 3 4 5
[1] 1.1 2.2 3.3 4.4
```

[20]:
```
# Vettore di stringhe
vettore_stringhe <- c("R", "è", "un", "ottimo", "linguaggio")

print(vettore_stringhe)
```

```
[1] "R"          "è"          "un"         "ottimo"     "linguaggio"
```

VETTORI

Vettori

In R i vettori sono oggetti che mantengono al loro interno un insieme indicizzato di elementi **tutti dello stesso tipo**

Classi di vettori

- **character**
- **numeric**
- **integer**
- **logical**
- **complex**

[21]:
```
# Vettore logico
vettore_logico <- c(TRUE, FALSE, TRUE, FALSE)

print(vettore_logico)
```

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

[25]:
```
vettore <- c(1, "R", 2, "un", "ottimo", 3)
vettore
```

'1' · 'R' · '2' · 'un' · 'ottimo' · '3'

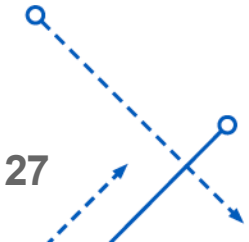
[27]:
```
vettore[1]
```

'1'

[26]:
```
typeof(vettore[1])
```

'character'

I vettori contenti sia stringhe che numeri, considerano i numeri come tipo character



OPERAZIONI CON I VETTORI

Come aggiungere elementi ad un vettore

I valori possono essere inseriti in un vettore in diversi modi, ad esempio attraverso:

- L'operatore di sequenza

```
> z<-8:12
> z # visualizza il vettore z
[1]  8  9 10 11 12
>
> class(z)
[1] "integer"
```

OPERAZIONI CON I VETTORI

Come aggiungere elementi ad un vettore

I valori possono essere inseriti in un vettore in diversi modi, ad esempio attraverso:

- L'operatore di sequenza
- L'operatore di concatenazione

```
> z<-c(8.5,9.4,10.3,11.5,12.6)
> z # visualizza il vettore z
[1]  8.5  9.4 10.3 11.5 12.6
>
> class(z)
[1] "numeric"
```

OPERAZIONI CON I VETTORI

Come aggiungere elementi ad un vettore

I valori possono essere inseriti in un vettore in diversi modi, ad esempio attraverso:

- L'operatore di sequenza
- L'operatore di concatenazione
- La funzione seq()

```
> z<-seq(8,16,2)
> z # visualizza il vettore z
[1]  8 10 12 14 16
```

OPERAZIONI CON I VETTORI

Come aggiungere elementi ad un vettore

I valori possono essere inseriti in un vettore in diversi modi, ad esempio attraverso:

- L'operatore di sequenza
- L'operatore di concatenazione
- La funzione seq()
- La funzione rep()

```
> x<-rep(3,times=5)
> x # visualizza il vettore x
[1] 3 3 3 3 3
>
> y<-rep(c(4,3),times=5)
> y # visualizza il vettore y
 [1] 4 3 4 3 4 3 4 3 4 3
>
> z<-rep(c(4,3),times=5,each=2)
> z # visualizza il vettore z
 [1] 4 4 3 3 4 4 3 3 4 4 3 3 4 4 3 3 4 4 3 3
```

OPERAZIONI CON I VETTORI

Come aggiungere elementi ad un vettore

I valori possono essere inseriti in un vettore in diversi modi, ad esempio attraverso:

- L'operatore di sequenza
- L'operatore di concatenazione
- La funzione seq()
- La funzione rep()
- La funzione scan()

```
z = scan(text = "1 2 3")
z
```

```
1 · 2 · 3
```

```
z[2]
```

```
2
```

OPERAZIONI CON I VETTORI

Ordinare elementi in un vettore

L'ordinamento degli elementi all'interno di un vettore può essere effettuato tramite l'ausilio della funzione sort(), nel seguente modo:

```
> z<-c(8,15,16,2,4,8,2,3,10,9)
> sort(z, decreasing=FALSE)
 [1]  2  2  3  4  8  8  9 10 15 16
 >
> sort(z, decreasing=TRUE)
 [1] 16 15 10  9  8  8  4  3  2  2
```

OPERAZIONI CON I VETTORI

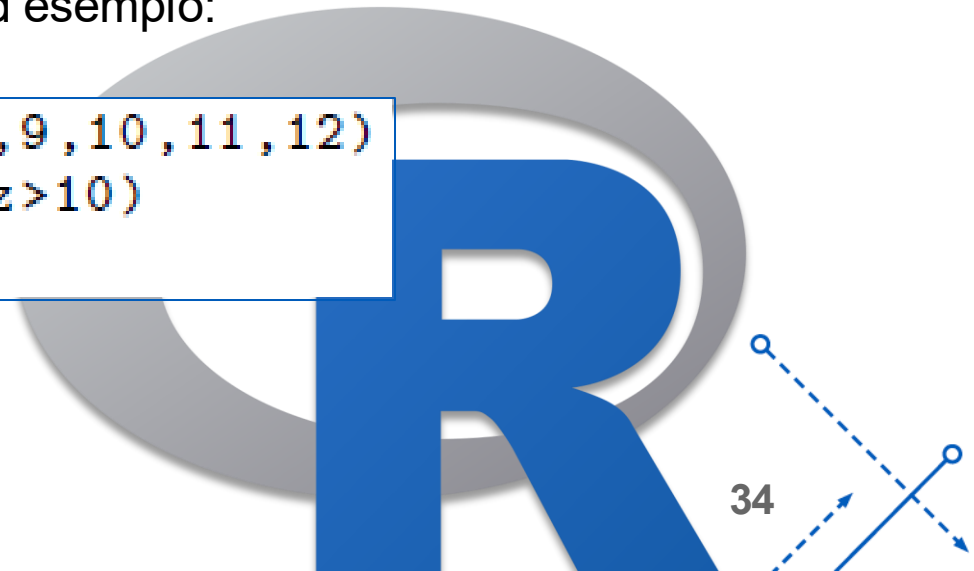
Ordinare e ricercare elementi in un vettore

L'ordinamento degli elementi all'interno di un vettore può essere effettuato tramite l'ausilio della funzione sort(), nel seguente modo:

```
> z<-c(8,15,16,2,4,8,2,3,10,9)
> sort(z, decreasing=FALSE)
 [1]  2  2  3  4  8  8  9 10 15 16
 >
> sort(z, decreasing=TRUE)
 [1] 16 15 10  9  8  8  4  3  2  2
```

È possibile conoscere le posizione assunte dagli elementi di un vettore che soddisfano una particolare condizione utilizzando la funzione which(), la quale richiede come argomento un vettore di tipo logico, ad esempio:

```
> z<-c(8,9,10,11,12)
> which(z>10)
[1] 4 5
```



OPERAZIONI CON I VETTORI

- I vettori presenti nella stessa espressione possono essere di lunghezza differente. In questo caso, il valore dell'espressione è un vettore con la stessa lunghezza del vettore più lungo presente in quell'espressione.
- I vettori con meno elementi sono riconsiderati tante volte fino ad arrivare alla stessa lunghezza del vettore più lungo. Ad esempio:

```
> y<-c(1,2)
> z<-c(8,9,10,11,12)
> y*z
[1]  8 18 10 22 12
Warning message:
In y * z : longer object length is not a multiple of shorter object
    length
>
> y+z
[1]  9 11 11 13 13
Warning message:
In y + z : longer object length is not a multiple of shorter object
    length
```

OPERAZIONI CON I VETTORI

- I vettori presenti nella stessa espressione possono essere di lunghezza differente. In questo caso, il valore dell'espressione è un vettore con la stessa lunghezza del vettore più lungo presente in quell'espressione.
- I vettori con meno elementi sono riconsiderati tante volte fino ad arrivare alla stessa lunghezza del vettore più lungo. Ad esempio:

```
> y<-c(1,2)
> z<-c(8,9,10,11,12)
> y*z
[1]  8 18 10 22 12
Warning message:
In y * z : longer object length is not a multiple of shorter object
    length
>
> y+z
[1]  9 11 11 13 13
Warning message:
In y + z : longer object length is not a multiple of shorter object
    length
```

SLICING

Accedere ad elementi del vettore

- Se si desidera accedere all'elemento i-esimo del vettore x occorre utilizzare x[i], mentre con x[−i] si crea un nuovo vettore contenente tutti gli elementi di x escluso l'elemento i-esimo.

```
> x = seq(1:10)
> x[3]
[1] 3
> x[-3]
[1]  1  2  4  5  6  7  8  9 10
```

SLICING

Accedere ad elementi del vettore

- Se si desidera accedere all'elemento i-esimo del vettore x occorre utilizzare x[i], mentre con x[−i] si crea un nuovo vettore contenente tutti gli elementi di x escluso l'elemento i-esimo.

- Invece, con x[i : j] si visualizzano gli elementi del vettore x dalla posizione i alla posizione j.

```
> x = seq(1:10)
> x[3]
[1] 3
> x[-3]
[1]  1  2  4  5  6  7  8  9 10
> x[2:6]
```

SLICING

Accedere ad elementi del vettore

- Se si desidera accedere all'elemento i-esimo del vettore x occorre utilizzare x[i], mentre con x[−i] si crea un nuovo vettore contenente tutti gli elementi di x escluso l'elemento i-esimo.

- Invece, con x[i : j] si visualizzano gli elementi del vettore x dalla posizione i alla posizione j.

- Infine con x[−(i : j)] si crea un nuovo vettore contenente tutti gli elementi di x esclusi gli elementi dalla posizione i alla posizione j.

```
> x = seq(1:10)
> x[3]
[1] 3
> x[-3]
[1]  1  2  4  5  6  7  8  9 10
> x[2:6]
[1] 2 3 4 5 6
> x[-(2:6)]
[1]  1  7  8  9 10
```

Funzioni matematiche che operano sui vettori

- **Numero di elementi del in x**
  > length(x)
- **Massimo valore contenuto in x**
  > max(x)
- **Minimo valore contenuto in x**
  > min(x)
- **Somma di tutti gli elementi di x**
  > sum(x)
- **Prodotto di tutti gli elementi di x**
  > prod(x)
- **Vettore con le differenze esistenti tra i valori di x**
  > diff(x)
- **Vettore con le somme parziali degli elementi di x**
  > cumsum(x)
- **Vettore con i prodotti parziali degli elementi di x**
  > cumprod(x)
- **Vettore con i massimi parziali degli elementi di x**
  > cummax(x)
- **Vettore con i massimi parziali degli elementi di x**
  > cummin(x)

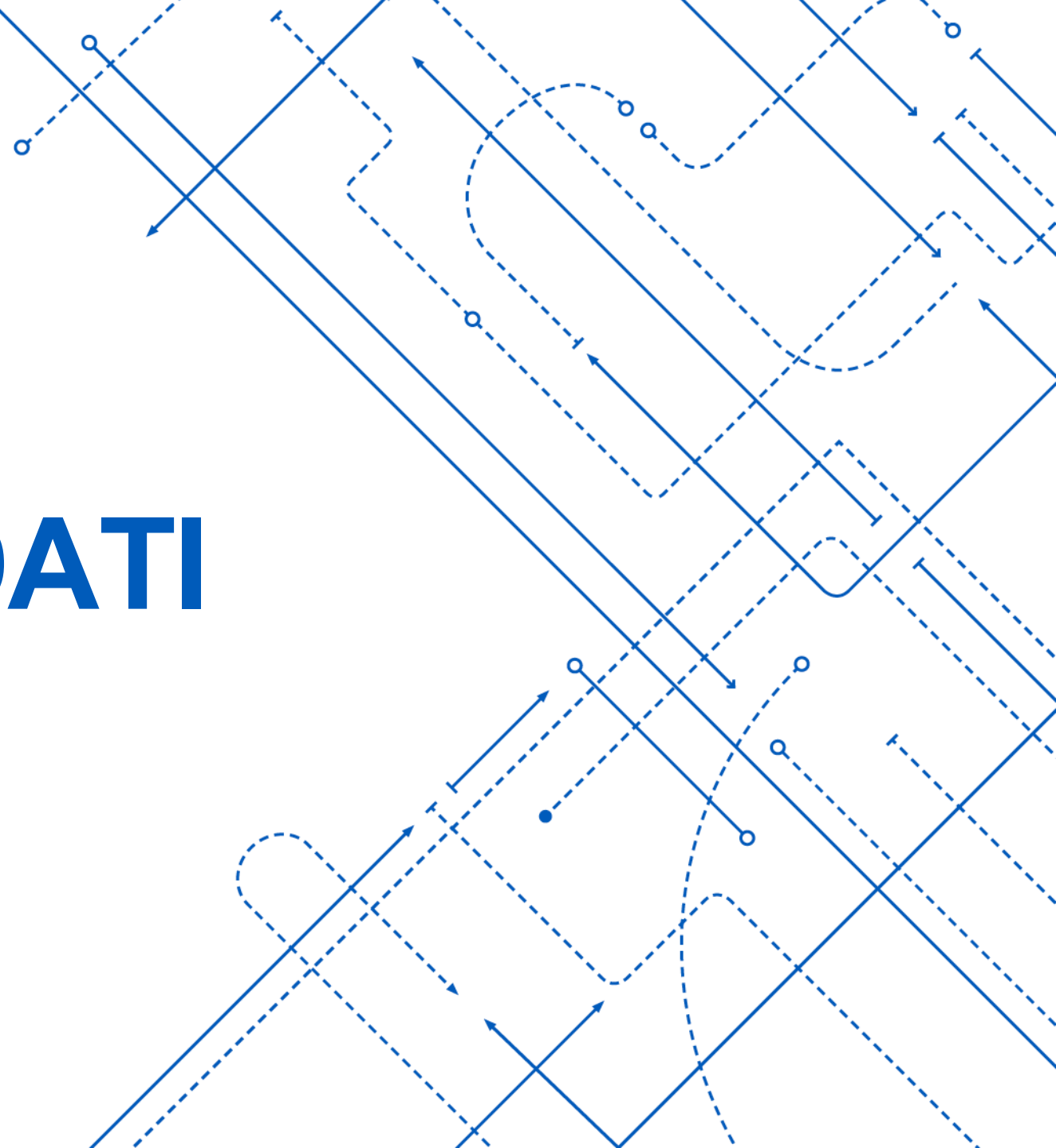
- **Media aritmetica dei valori presenti di x**
  > mean(x)
- **Mediana dei valori presenti di x**
  > median(x)
- **Vettore contenente il minimo ed il massimo di x**
  > range(x)
- **Vettore che contiene i valori distinti di x**
  > unique(x)
- **Vettore che contiene i quantili di x**
  > quantile(x)
- **Varianza campionaria di x**
  > var(x)
- **Deviazione standard campionaria di x**
  > sd(x)
- **Correlazione campionaria dei tra x e y**
  > cor(x,y)
- **Ordina gli elementi di x in ordine crescente**
  > sort(x)
- **Rovescia gli elementi del vettore x**
  > rev(x)
- **Ordina gli elementi di x in ordine decrescente**
  > rev(sort(x))

STATISTICA E ANALISI DEI DATI

Array e Matrici

Dott. Stefano Cirillo
Dott. Luigi Di Biasi

a.a. 2024-2025



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**

L'AMBIENTE INTEGRATO R

Array

In R gli array sono oggetti che mantengono al loro interno un insieme indicizzato di elementi **tutti dello stesso tipo**

Classi di array

- **character**
- **numeric**
- **integer**
- **logical**
- **complex**

R fornisce ben tre metodi per «scoprire» il tipo di dato contenuto in un array:

- class()
- typeof()
- mode()

L'AMBIENTE INTEGRATO R

Tipi di dati in un Array

R fornisce ben tre metodi per «scoprire» il tipo di dato contenuto in un array.

- class restituisce il tipo di dati contenuto in un array «dal punto di vista di R»

- typeof restituisce il tipo di dati contenuto in un array «dal punto di vista OOP»

- character
- numeric
- integer
- logical
- complex

```
> X <- c(24 ,26 ,30 ,25 ,29 ,27 ,20 ,29 ,27 ,28 ,18 ,21 ,26 ,30 ,28);
> class(X)
[1] "numeric"
> typeof(X)
[1] "double"
>
```

ARRAY VS VETTORI

- Nella pratica array e vettori possono risultare molto simili
  - Molto spesso gli array vengono indicati come “vector structure”
- Semplicemente un array apre alla possibilità di specificare un ulteriore **parametro che ne definisce la dimensione**
- In particolare, in un array tridimensionale *a*, *a[i, j, k]* è l’elemento nella posizione (i, j, k) dell’array
  - Un array tridimensionale *a* di dimensione $n \times m \times r$ è visto come una sovrapposizione di *r* array bidimensionali di dimensione $n \times m$

ARRAY VS VETTORI

- **Dimensioni**: I vettori sono sempre unidimensionali, mentre gli array possono avere più dimensioni
- **Creazione**: I vettori si creano con c(), mentre gli array con array() specificando le dimensioni
- **Uso pratico**: I vettori sono ideali per serie di dati semplici, mentre gli array sono utili per rappresentare tabelle o matrici di dati multidimensionali

DICHIARARE UN ARRAY

```
> a <- array (1:24 , dim =c (4 ,3 ,2) )
> a #visualizza l'array a
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24

> dim(a)
[1] 4 3 2
>
> length(a)
[1] 24
```

```
> v <- c("casa", "albero")
> v
[1] "casa"   "albero"
> a <- array(a, dim=c(1,2))
> a
     [,1]   [,2]
[1,] "casa" "albero"
> v2 <- c(v, "biscotto", "lettera")
> v2
[1] "casa"     "albero"   "biscotto" "lettera"
> a2 <- array(v2, dim=c(1,4))
> a
     [,1]   [,2]
[1,] "casa" "albero"
> a2
     [,1]   [,2]     [,3]       [,4]
[1,] "casa" "albero" "biscotto" "lettera"
> a3 <- array(v2, dim=c(2,2))
> a3
     [,1]     [,2]
[1,] "casa"   "biscotto"
[2,] "albero" "lettera"
```

Da Array a Matrici

- Gli array sono utili quando è necessario fornire differenti informazioni per identificare un elemento.
- Un esempio tipico di array contiene le temperature minime e massime, il giorno, il mese e l'anno di registrazione in diverse città (con tutti i valori espressi da numeri).
- Una matrice è un array bidimensionale di elementi univocamente determinati da una coppia di numeri interi, che costituiscono l'indice di riga e di colonna.
- Un array bidimensionale può essere inizializzato nel seguente modo:

```
> a <- array(1 : h, dim = c(n, m))
```

oppure equivalentemente

```
> a <- matrix(1 : h, nrow = n, ncol = m)
```

RIEMPIRE LE MATRICI

- Se si vuole costruire una matrice *n* × *m* con tutti elementi nulli si può utilizzare il comando **matrix()** nel seguente modo:

```
> a <- matrix(0, nrow = 2, ncol = 2)
> a
     [,1] [,2]
[1,]    0    0
[2,]    0    0
> a <- matrix(1, nrow = 2, ncol = 2)
> a
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> a <- matrix(, nrow = 2, ncol = 2)
> a
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

- Se si desidera accedere all'elemento (i, j) della matrice a occorre utilizzare *a[i, j]*, mentre con a[, j] si selezionano gli elementi della colonna j-esima e infine con a[i,] si selezionano gli elementi della riga i-esima.

FUNZIONI CBIND E RBIND

Riempire le matrici

- Le funzioni cbind() e rbind() permettono di creare opportune matrici componendo vettori di uguale lunghezza e matrici delle stesse dimensioni.
- La prima funzione usa i vettori per creare le colonne

```
a = cbind(c(1,2,5),4:6, matrix(7:12, nrow=3, ncol=2))
a
```

A matrix: 3 × 4

of type dbl

1	4	7	10
2	5	8	11
5	6	9	12

FUNZIONI CBIND E RBIND

Riempire le matrici

- Le funzioni cbind() e rbind() permettono di creare opportune matrici componendo vettori di uguale lunghezza e matrici delle stesse dimensioni.
- La prima funzione usa i vettori per creare le colonne

```
a = cbind(c(1,2,5),4:6, matrix(7:12, nrow=3, ncol=2))
a
```

A matrix: 3 × 4

of type dbl

1	4	7	10
2	5	8	11
5	6	9	12

FUNZIONI CBIND E RBIND

Riempire le matrici

- Le funzioni cbind() e rbind() permettono di creare opportune matrici componendo vettori di uguale lunghezza e matrici delle stesse dimensioni.
- La prima funzione usa i vettori per creare le colonne mentre la seconda per creare le righe.

```
b = rbind(c(1,2,5),4:6, matrix(7:12, nrow=2, ncol=3))
b
```

A matrix: 4 × 3 of type dbl

1	2	5
4	5	6
7	9	11
8	10	12

FUNZIONI CBIND E RBIND

Riempire le matrici

- Le funzioni cbind() e rbind() permettono di creare opportune matrici componendo vettori di uguale lunghezza e matrici delle stesse dimensioni.
- La prima funzione usa i vettori per creare le colonne mentre la seconda per creare le righe.

```
b = rbind(c(1,2,5),4:6, matrix(7:12, nrow=2, ncol=3))
b
```

A matrix: 4 × 3 of type dbl

1	2	5
4	5	6
7	9	11
8	10	12

FUNZIONE DIAG

Riempire le matrici

- Altre funzioni molto utilizzate nell'algebra matriciale sono:
  - *diag(v)*, con v vettore, usata per creare una matrice diagonale con gli elementi del vettore sulla diagonale e i restanti elementi nulli;

```
> v <- 1:10
> v
 [1]  1  2  3  4  5  6  7  8  9 10
> diag(v)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1    0    0    0    0    0    0    0    0     0
 [2,]    0    2    0    0    0    0    0    0    0     0
 [3,]    0    0    3    0    0    0    0    0    0     0
 [4,]    0    0    0    4    0    0    0    0    0     0
 [5,]    0    0    0    0    5    0    0    0    0     0
 [6,]    0    0    0    0    0    6    0    0    0     0
 [7,]    0    0    0    0    0    0    7    0    0     0
 [8,]    0    0    0    0    0    0    0    8    0     0
 [9,]    0    0    0    0    0    0    0    0    9     0
[10,]    0    0    0    0    0    0    0    0    0    10
```

FUNZIONE DIAG

Riempire le matrici

- Altre funzioni molto utilizzate nell'algebra matriciale sono:
  - *diag(v)*, con v vettore, usata per creare una matrice diagonale con gli elementi del vettore sulla diagonale e i restanti elementi nulli;
  - *diag(a)*, con a matrice, fornisce un vettore costituito dagli elementi della diagonale principale della matrice;

```
v = c(2,4,5,1,6,8,9)
v
```

```
2 · 4 · 5 · 1 · 6 · 8 · 9
```

```
diag(v)
```

A matrix: 7 × 7 of type dbl

2	0	0	0	0	0	0
0	4	0	0	0	0	0
0	0	5	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	6	0	0
0	0	0	0	0	8	0
0	0	0	0	0	0	9

FUNZIONE DIAG

Riempire le matrici

- Altre funzioni molto utilizzate nell'algebra matriciale sono:
  - *diag(v)*, con v vettore, usata per creare una matrice diagonale con gli elementi del vettore sulla diagonale e i restanti elementi nulli;
  - *diag(a)*, con a matrice, fornisce un vettore costituito dagli elementi della diagonale principale della matrice;
  - *diag(n)*, con n intero, fornisce la matrice identità $n \times n$.

```
diag(6)
```

A matrix: 6 × 6 of type dbl

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

ROWNAMES E COLNAMES

- È possibile cambiare i nomi assegnati alle intestazioni delle righe e delle colonne di una matrice:
- *rownames(a)* restituisce o imposta i nomi delle righe di una matrice o di un data frame
- *colnames(a)* restituisce o imposta i nomi delle colonne di una matrice o di un data frame

```r
# Creazione di una matrice
matrice <- matrix(1:9, nrow = 3, ncol = 3)

# Assegna nomi alle righe e colonne
rownames(matrice) <- c("Riga1", "Riga2", "Riga3")
colnames(matrice) <- c("Col1", "Col2", "Col3")

# Visualizzare la matrice
print(matrice)
```

```
      Col1 Col2 Col3
Riga1    1    4    7
Riga2    2    5    8
Riga3    3    6    9
```

OPERAZIONI MATRICIALI

- R permette di eseguire le usuali operazioni sui vettori e le matrici
- **R interpreta i suoi vettori come vettori colonna**
- Per effettuare la trasposizione di un vettore o di una matrice occorre utilizzare **la funzione t() di trasposizione**
  - Dall'esempio precedente, calcoliamo la matrice della trasposta:

```
# Visualizzare la matrice
print(matrice)
```

```
      Col1 Col2 Col3
Riga1    1    4    7
Riga2    2    5    8
Riga3    3    6    9
```

```
t(matrice)
```

A matrix: 3 × 3 of type int

	Riga1	Riga2	Riga3
Col1	1	2	3
Col2	4	5	6
Col3	7	8	9

Operazioni Matriciali

- Sui vettori e sulle matrici delle stesse dimensioni sono applicabili le usuali operazioni aritmetiche
  - Somma: +
  - Differenza: −
  - Prodotto: *
  - Ecc.

che agiscono elemento per elemento

```
# Visualizzare la matrice
print(matrice)
```

```
      Col1 Col2 Col3
Riga1    1    4    7
Riga2    2    5    8
Riga3    3    6    9
```

```
matrice+10
```

A matrix: 3 × 3 of type dbl

	Col1	Col2	Col3
Riga1	11	14	17
Riga2	12	15	18
Riga3	13	16	19

```
matrice-2
```

A matrix: 3 × 3 of type dbl

	Col1	Col2	Col3
Riga1	-1	2	5
Riga2	0	3	6
Riga3	1	4	7

```
matrice*0.5
```

A matrix: 3 × 3 of type dbl

	Col1	Col2	Col3
Riga1	0.5	2.0	3.5
Riga2	1.0	2.5	4.0
Riga3	1.5	3.0	4.5

Il Prodotto Matriciale

- R fornisce la possibilità di effettuare il prodotto matriciale tra due matrici *a* di dimensione $n \times m$ e *b* di dimensione $m \times r$ **utilizzando l'operatore %*%.**

```
# Visualizzare la matrice
print(matrice)
```

```
      Col1 Col2 Col3
Riga1    1    4    7
Riga2    2    5    8
Riga3    3    6    9
```

```
t(matrice)
```

A matrix: 3 × 3 of type int

	Riga1	Riga2	Riga3
Col1	1	2	3
Col2	4	5	6
Col3	7	8	9

```
#Prodotto Matriciale: Matrice x Trasposta
matrice %*% t(matrice)
```

A matrix: 3 × 3 of type dbl

	Riga1	Riga2	Riga3
Riga1	66	78	90
Riga2	78	93	108
Riga3	90	108	126

Determinante di Una Matrice

- Determinante di una matrice:
  - È un valore scalare associato a una matrice quadrata
  - Fornisce informazioni sulla matrice, ad esempio se è invertibile
  - Una matrice è invertibile solo se il suo determinante è diverso da zero
- È possibile calcolare il determinante di una matrice quadrata *a* con il comando ***det****(a)*,

```r
# Creazione di una matrice 4x4
matrice <- matrix(c(4, 7, 2, 6, 3, 4, 4, 5, 1, 10, 20, 14, 4, 4, 5, 1), nrow = 4)
rownames(matrice) <- c("Riga1", "Riga2", "Riga3", "Riga4")
colnames(matrice) <- c("Col1", "Col2", "Col3", "Col4")
matrice
```

A matrix: 4 × 4 of type dbl

	Col1	Col2	Col3	Col4
Riga1	4	3	1	4
Riga2	7	4	10	4
Riga3	2	4	20	5
Riga4	6	5	14	1

```r
# Calcolo del determinante
determinante <- det(matrice)
determinante
```

879

INVERSA DI UNA MATRICE

- **Matrice inversa**:
  - La matrice inversa di una matrice quadrata A è una matrice A^{-1} tale che:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

  Dove I è la matrice identità di ordine *n*

- In altre parole, è una matrice che, moltiplicata per la matrice originale, restituisce la matrice identità.
  - Una matrice quadrata A ha un'inversa A^{-1} se e solo se det(A) ≠ 0
- È possibile calcolare il determinante di una matrice quadrata *a* con il comando ***det(a)***
- Sia A una matrice quadrata invertibile, tale che det(A)≠0, allora la matrice inversa A^{-1} è uguale al prodotto tra l'inverso del determinante di A e la trasposta dei cofattori

$$A^{-1} = \frac{1}{det(A)} \cdot cof(A)^T$$

INVERSA DI UNA MATRICE

- Sia A una matrice quadrata invertibile, tale che det(A)≠0, allora la matrice inversa A^{-1} è uguale al prodotto tra l'inverso del determinante di A e la trasposta dei cofattori:

$$A^{-1} = \frac{1}{det(A)} \cdot cof(A)^T$$

- **Cofattori**:
  - Il **cofattore** di un elemento a_{ij} in una matrice A (nella posizione i-esima riga e j-esima colonna) è dato da:

$$C_{ij} = (-1)^{i+j} \cdot M_{ij}$$

  - dove:
    - $(-1)^{i+j}$ è il **segno** associato alla posizione (i,j), il che significa che se la somma i + j è pari, il segno è positivo (+), e se è dispari, il segno è negativo (−)
    - M_{ij} è il **minore** dell'elemento a_{ij}, ossia il determinante della matrice che si ottiene eliminando la riga i e la colonna j dalla matrice originale

INVERSA DI UNA MATRICE

- Per calcolare l'inversa di una matrice *a* quadrata non singolare (ossia con determinante diverso da zero) si usa il comando ***solve****(a):*

```
# Calcolo dell'inversa
inversa <- solve(matrice)

# Visualizzazione dei risultati
print(inversa)
```

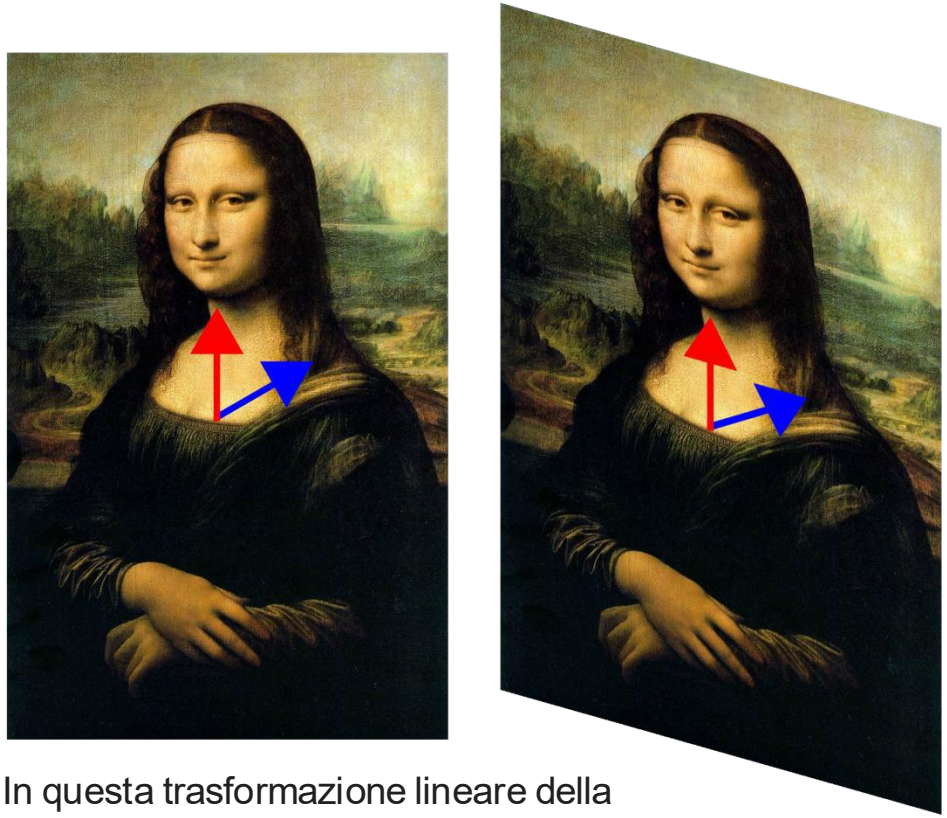
```
           Riga1       Riga2       Riga3       Riga4
Col1 -0.18885097  0.34698521 -0.11149033 -0.07508532
Col2  0.49829352 -0.61433447  0.01706485  0.37883959
Col3 -0.10352673  0.06370876  0.03526735 -0.01706485
Col4  0.09101251  0.09783845  0.08987486 -0.20477816
```

AUTOVALORI ED AUTOVETTORI

- R consente anche di determinare gli *autovalori* e gli *autovettori* di una matrice quadrata *A* di ordine n
- Autovalori (λ) e Autovettori (v) sono concetti fondamentali dell'algebra lineare:
  - Sia A una matrice quadrata n x n, v è chiamato *autovettore* se esiste un numero λ tale che valga la relazione:

$$A \cdot v = \lambda \cdot v$$

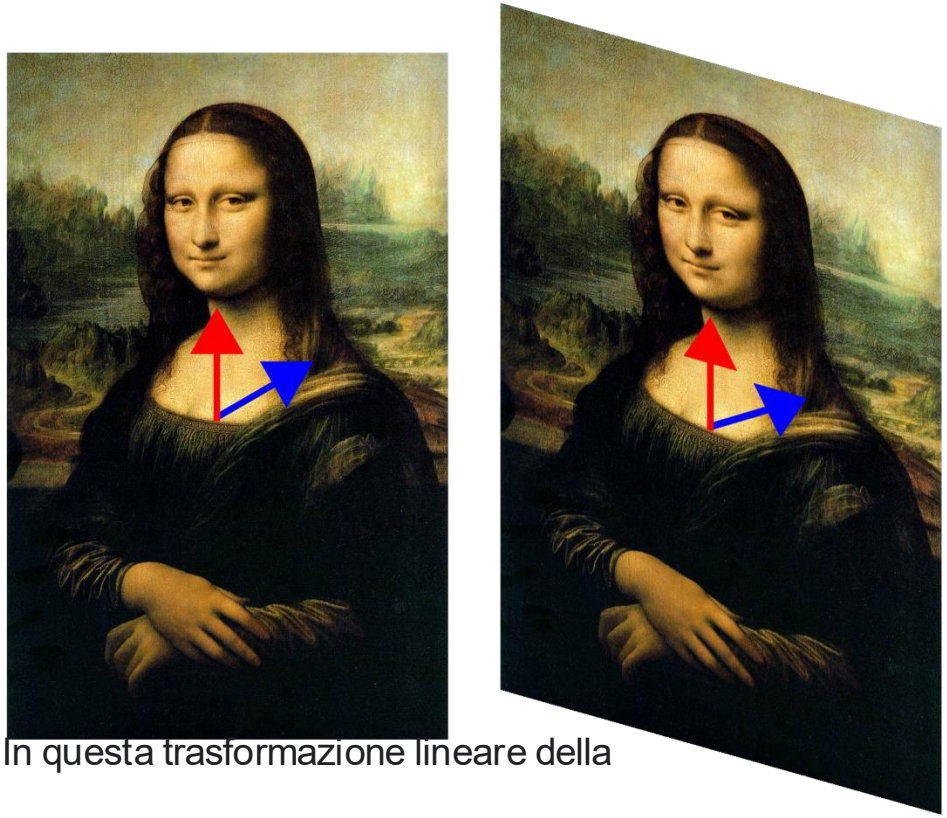
  - Ciò equivale a richiedere che $(A-\lambda I)v = 0$, dove I è la matrice identità ammette soluzioni non nulle se e solo se $\det(A-\lambda I)=0$
- Questo significa che, quando la matrice A agisce sull'autovettore v, il risultato è un vettore parallelo a v, scalato di un fattore λ



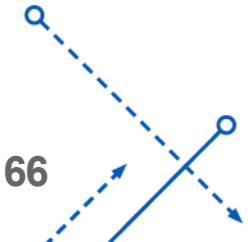
In questa trasformazione lineare della Gioconda l'immagine è modificata ma l'asse centrale verticale rimane fisso. Il vettore blu ha cambiato lievemente direzione, mentre quello rosso no. Quindi il vettore rosso è un autovettore della trasformazione e quello blu no. Inoltre, poiché il vettore rosso non è stato né allungato, né compresso, né ribaltato, il suo autovalore è 1. [Wikipedia]

AUTOVALORI ED AUTOVETTORI

- Gli autovettori rappresentano le **direzioni** lungo le quali una trasformazione lineare (rappresentata dalla matrice A) agisce semplicemente come una **dilatazione** o una **compressione**
- L'autovalore associato indica **quanto** viene dilatato o compresso l'autovettore nella sua direzione:
  - Se $\lambda>1$ la trasformazione **allunga** il vettore
  - Se $0<\lambda<1$, la trasformazione **accorcia** il vettore
  - Se $\lambda=1$, il vettore **resta invariato** in lunghezza
  - Se $\lambda=0$, il vettore viene **trasformato** nel vettore nullo
  - Se $\lambda<0$ il vettore viene **invertito** di direzione (oltre a essere eventualmente scalato)
- **Applicazioni**:
  - **Analisi delle componenti principali (PCA)**: In statistica, l'analisi delle componenti principali usa gli autovettori e gli autovalori della matrice di covarianza dei dati per ridurre la dimensionalità e trovare le direzioni principali di varianza
  - **Meccanica quantistica**: Gli autovalori e autovettori di operatori quantistici corrispondono agli stati osservabili e ai valori misurabili di grandezze fisiche (energia, momento, ecc.).
  - …



In questa trasformazione lineare della Gioconda l'immagine è modificata ma l'asse centrale verticale rimane fisso. Il vettore blu ha cambiato lievemente direzione, mentre quello rosso no. Quindi il vettore rosso è un autovettore della trasformazione e quello blu no. Inoltre, poiché il vettore rosso non è stato né allungato, né compresso, né ribaltato, il suo autovalore è 1. [Wikipedia]



AUTOVALORI ED AUTOVETTORI

- In R la funzione *eigen(A)* calcola gli autovalori e gli autovettori di una matrice *A*.
- Il risultato è una lista di due componenti:
  - un vettore di nome *values* contenente gli autovalori
  - una matrice di nome *vectors* che contiene i corrispondenti autovettori sulle sue colonne
- Questi due componenti possono essere usati in istruzioni di assegnazione utilizzando *eigen(A)$values* e *eigen(A)$vectors*, rispettivamente

```r
# Creazione della matrice 2x2
matrice <- matrix(c(4, 7, 2, 6), nrow = 2)

# Calcolo degli autovalori e autovettori
autovalori_autovettori <- eigen(matrice)

# Estrarre autovalori e autovettori
autovalori <- autovalori_autovettori$values
autovettori <- autovalori_autovettori$vectors

# Visualizzazione dei risultati
print(autovalori)
print(autovettori)
```

```
[1] 8.872983 1.127017
           [,1]       [,2]
[1,] -0.3796908 -0.5713345
[2,] -0.9251135  0.8207173
```

AUTOVALORI ED AUTOVETTORI

- Per verificare se gli autovalori e autovettori di una matrice sono stati calcolati correttamente, si può controllare se la relazione fondamentale $A \cdot v = \lambda \cdot v$ è soddisfatta per ciascun autovalore λ e il corrispondente autovettore v

```r
# Creazione di una matrice 2x2
matrice <- matrix(c(4, 7, 2, 6), nrow = 2)

# Calcolo degli autovalori e autovettori
autovalori_autovettori <- eigen(matrice)

# Estrazione degli autovalori e autovettori
autovalori <- autovalori_autovettori$values
autovettori <- autovalori_autovettori$vectors

# Verifica del primo autovalore e autovettore
A_v1 <- matrice %*% autovettori[, 1]  # A * v1
lambda1_v1 <- autovalori[1] * autovettori[, 1]  # λ1 * v1

# Verifica del secondo autovalore e autovettore
A_v2 <- matrice %*% autovettori[, 2]  # A * v2
lambda2_v2 <- autovalori[2] * autovettori[, 2]  # λ2 * v2
```

```r
# Visualizzazione dei risultati
cat("Verifica per il primo autovalore e autovettore:\n")
print(A_v1)
print(lambda1_v1)

cat("\nVerifica per il secondo autovalore e autovettore:\n")
print(A_v2)
print(lambda2_v2)
```

```
Verifica per il primo autovalore e autovettore:
          [,1]
[1,] -3.368990
[2,] -8.208516
[1] -3.368990 -8.208516

Verifica per il secondo autovalore e autovettore:
           [,1]
[1,] -0.6439035
[2,]  0.9249620
[1] -0.6439035  0.9249620
```

DOMANDE?