



Laboratorio R associato all'insegnamento  
di Analisi dei Dati  
Cdl: Comunicazione Digitale

Orario Lezioni: Lunedì 13:30-15:30

*Marco Notaro*

# L' Ambiente R

- R is a **free** open-source software environment for statistical computing and graphics
- It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS
- To **download R**, please choose your preferred CRAN mirror.
- CRAN (Comprehensive R Archive Network) is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.
- R è un *programma interpretato*. I programmi interpretati vengono tradotti ed eseguiti immediatamente riga per riga (l' interprete simula una macchina astratta, no distinzione netta fra compile-time e run-time).

# Let's start with some simple commands...

## •Avviare R

Linux/Mac: R from command line

Windows: open R from GUI (***G**raphical **U**ser **I**nterface*)

## •Uscire da R

`q()`

## •Chiedere aiuto ad R

`help()` # chiede l'help generale

`help(nomecomando)` # chiede l'help del comando

`?nomecomando` # chiede l'help del comando

`help.start()` # lancia l'help in un browser

## •Manipolazione workspace

```
ls() # mostra il contenuto del workspace
ls(pattern="V") #mostra gli oggetti che contengono "V" nel nome

rm(i) # elimina la variabile "i"
rm(list=ls(pat="V")) # rimuoviamo tutte le variabili che
                    # contengono "V" nel nome
rm(list=ls()) # cancelliamo tutto il workspace

save(...,file="prova.rda") # salviamo il workspace in un
                           # file chiamato prova.rda

load("prova.rda") # ricarichiamo i dati salvati in precedenza

setwd("c:/Users/Marco/Desktop/") # spostarsi in una directory
                                 # (windows usage)
```

# Strutture dati

Nei linguaggi di programmazione ad alto livello non si ha accesso diretto alla memoria, ma ad una sua astrazione (**struttura dati**)

Le principali strutture dati fornite da R sono:

1. Vettori
2. Array e Matrici
3. Fattori
4. Liste
5. Data frame

# Vettori

*Rappresentano sequenze ordinate di dati omogenei*  
(sequenze ordinate di numeri o di caratteri).

*Esempio:*

```
> c(1,4,5) # crea un vettore di interi  
> c("A","B","C") # crea un vettore di  
caratteri  
> c("gatto", "topo", "12") # crea un  
vettore di stringhe
```

La *funzione* `c`(arg1, arg2, arg3, arg4) combina i suoi  
argomenti in vettore (`c` sta per “**concatenation**”)

# Variabili ed assegnamenti -1

Un vettore può essere *assegnato* ad una *variabile*.

## Es.1

```
> X <- c(1,4,5) # il vettore <1 4 5> è assegnato alla variabile
> X
> X
[1] 1 4 5
```

La variabile X rappresenta ora il vettore <1 4 5>: si può pensare come un “contenitore” del vettore stesso

## Es. 2

```
> X <- c(4,7) # il vettore <4 7> è assegnato alla variabile X
> X
[1] 4 7
```

Un nuovo assegnamento cancella il contenuto precedente

## Es.3

```
> Y <- 100 # vettore formato da 1 solo elemento
> Y
[1] 100
```

# Variabili ed assegnamenti -2

Altri modi per l' assegnamento:

Es.4

```
> c(1,4,5) -> x           # notare il verso della freccia  
>x = c(1,4,5)
```

I valori di una variabile possono essere assegnati ad altre variabili:

Es.5

```
> y <- 2  
> z <- y  
> z  
[1] 2
```

Non possono essere assegnati valori ad una costante:

Es.6

```
> 2 <- x  
Error in 2 <- x : invalid (do_set) left-hand side to  
assignment
```



## Concatenazione di vettori

I vettori possono essere concatenati attraverso l'operatore **c** di concatenazione:

```
> x <- c(1, 2, 3)
```

```
> y <- c(4, 5, 6)
```

```
> z <- c(x, y)
```

```
> z
```

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

```
> w <- c(z, x, 9, y)
```

```
> w
```

```
[1] 1 2 3 4 5 6 1 2 3 9 4 5 6
```

# Tipi elementari di vettori

I vettori sono sequenze ordinate i cui elementi possono essere di 3 tipi base:

- **Numerici**: numeri interi o in virgola mobile (floating point)
- **Caratteri**: singoli caratteri o stringhe (sequenze) di caratteri
- **Logici**: TRUE o FALSE

# Operazioni con vettori aritmetici

Le operazioni usuali dell'aritmetica vengono eseguite sui vettori **elemento per elemento**:

## Addizione e sottrazione

```
> x <- c(1,2,3)
```

```
> y <- c(4,5,6)
```

```
> z <- x + y
```

```
> z
```

```
[1] 5 7 9
```

```
> d <- y - x
```

```
> d
```

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

## Moltiplicazione e divisione

```
> x <- c(1,2,3)
```

```
> y <- c(4,5,6)
```

```
> p <- x * y
```

```
> p
```

```
[1] 4 10 18
```

```
> q <- y / x
```

```
> q
```

```
[1] 4.0 2.5 2.0
```

# Funzioni matematiche

Sono disponibili diversi operatori e funzioni matematiche (che operano sempre elemento per elemento). Ad esempio:

```
> X <- c(1,2,3)
> X^3
> log(x)
[1] 1 8 27
[1] 0.0000000 0.6931472 1.0986123
> exp(x)
[1] 2.718282 7.389056 20.085537
> sin(x)
[1] 0.8414710 0.9092974 0.1411200
> sqrt(x)
[1] 1.000000 1.414214 1.732051
```

## Altre funzioni applicate a vettori numerici

```
> x <- (1,2,3)
> mean(x)
[1] 2
> var(x)
[1] 1
> max(x)
[1] 3
> min(x)
[1] 1
> range(x)
[1] 1 3
> sum(x)
[1] 6
> prod(x)
[1] 6
```

```
> y <- rnorm(10) # generazione ~N(0,1)
> y
[1] -2.0196896 -0.4782142 -2.0803635 -
0.8054892 -1.6530595 -1.5180686 0.3310005
-0.5082732 0.2394145 -1.2659536
> sort(y) #ordinamento
[1] -2.0803635 -2.0196896 -1.6530595 -
1.5180686 -1.2659536 -0.8054892 -0.5082732
-0.4782142 0.2394145 0.3310005
> order(y) #ordinamento indici
[1] 3 1 5 6 10 4 8 2 9 7
```

# Generazione di sequenze regolari

R dispone di diversi comandi per generare automaticamente sequenze di numeri:

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

```
> 5:1  
[1] 5 4 3 2 1
```

```
> seq(from=1, to=5, by=0.5)  
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Un'altra funzione per generare repliche di vettori è **rep()**:

```
> rep( 1:2, times=4)  
[1] 1 2 1 2 1 2 1 2
```

# Vettori ed operatori logici

Sono vettori i cui elementi possono assumere valore **TRUE** o **FALSE**.

## Operatori logici:

<, <=, >, >=, == (uguaglianza), != (disuguaglianza)

Es:

```
> x <- 3:8
```

```
> x > 5
```

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

```
> x <= 5
```

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

```
> x == 5
```

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

```
> x != 5
```

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

```
> x != c(5,6) # vale la "regola del riciclo"!
```

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

# La “regola del riciclo” in R

Se si sommano due vettori di diversa lunghezza in R, il vettore più corto viene ripetuto tante volte fino a raggiungere la lunghezza del vettore di maggior lunghezza.

Es:

```
> x <- c(2,4)
> y <- c(3,5,7,9)
> x+y      #il vettore x viene ripetuto due volte regola del riciclo)
[1]  5  9  9 13
```

La somma precedente equivale cioè a:

```
> xx <- c(x,x)    # duplicazione del vettore x
> xx+y
[1]  5  9  9 13
```

La regola vale in generale in R, anche per altre strutture dati e per altre operazioni ..... Sperimentate!!!



# Dati mancanti

- In molte situazioni reali i componenti di un vettore possono essere “non noti” o comunque non disponibili.
- In questi casi R riserva il valore speciale **NA** (“Not Available”).
- In generale qualunque operazione che coinvolga valori NA ha come risultato NA.

**Es:**

```
> x <- c(1:4,NA)
```

```
> x + 2
```

```
[1] 3 4 5 6 NA
```

Si noti che NA non è un valore ma un “marcatore” di una quantità non disponibile:

```
> x == NA
```

```
[1] NA NA NA NA NA
```

Per individuare quali elementi siano effettivamente NA in un vettore si deve usare la funzione **is.na()**:

```
> is.na(x)
```

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

# Vettori: selezione e accesso a sottoinsiemi di elementi

Esistono diverse modalità di accesso a singoli elementi o a sottoinsiemi di elementi di un vettore. In generale la selezione e l'accesso avviene tramite l' **operatore []** (parentesi quadre) : sottoinsiemi di elementi di un vettore sono selezionati collegando al nome del vettore un vettore di indici in parentesi quadre. Esistono *almeno 4 modalità di selezione/accesso*:

- Vettori di **indici interi positivi**
- Vettore di **indici interi negativi**
- Vettore di **indici logici**
- Vettori di **indici a caratteri**

# Selezione ed accesso tramite vettori di indici interi positivi

Gli elementi di un vettore  $x$  sono selezionati tramite un vettore positivi  $y$  di indici positivi racchiuso fra parentesi quadre :  $x[y]$  i corrispondenti elementi sono selezionati e concatenati.

```
> x <- 5:10
> x
[1] 5 6 7 8 9 10
> length(x)
[1] 6
> x[1]
[1] 5
> x[5]
[1] 9
> x[7]
[1] NA
> x[2:4]
[1] 6 7 8
> x[c(1,4,6)]
[1] 5 8 10
```

# Selezione ed accesso tramite vettori di indici interi negativi

Sono selezionati gli elementi di un vettore  $x$  che devono **essere esclusi** tramite un vettore  $y$  di indici negativi racchiuso fra parentesi quadre:  $x[y]$

```
> y <- rep(c("G", "A", "T", "T"),
times=3)
> y
 [1] "G" "A" "T" "T" "G" "A" "T" "T"      :
"G" "A" "T" "T"
> z <- y[-(1:5)]
> z
 [1] "A" "T" "T" "G" "A" "T" "T"
> z <- y[-length(y)]
> z
 [1] "G" "A" "T" "T" "G" "A" "T" "T"
"G" "A" "T"
```

# Selezione ed accesso tramite vettori indice logici

Il vettore indice deve essere della stessa lunghezza del vettore i cui elementi devono essere selezionati. Sono selezionati gli elementi corrispondenti a TRUE nel vettore degli indici ed omessi quelli corrispondenti a FALSE.

Es:

```
> x <- c(1:5,NA,NA)
> x
[1] 1 2 3 4 5 NA NA
> i <- c(rep(T,times=3),rep(F,times=4))
>
> i
[1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE
> x[i]
[1] 1 2 3
> x[!is.na(x)]
[1] 1 2 3 4 5
> x[!is.na(x) & x > 2]
[1] 3 4 5
```

# Selezione ed accesso tramite vettori di indici a caratteri

E' applicabile quando un vettore possiede un attributo **names** per identificare le sue componenti. In questo caso un sottovettore del vettore names può essere utilizzato per selezionare le componenti

Es:

```
> campione <- c(45,210,5.12,73.22,0.82)
> campione
[1] 45.00 210.00 5.12 73.22 0.82
> names(campione) <- letters[1:5]
> campione
      a      b      c      d      e
45.00 210.00 5.12 73.22 0.82
> sel.cam.nam <- campione[c("a","e")]
> sel.cam.nam
      a      e
45.00 0.82
```

# Selezione ed accesso mediante comando

## *which*

```
x <- -3:8
which(x<2) # indici degli elementi che verificano
x<2
which((x >= -1) & (x < 5)) #x >= -1 e x < 5
which((x < -2) | (x > 1)) #x < -2 o x > 1
indici<-which((x >= -1) & (x < 5))

# mettiamo gli indici in un vettore
x[indici] # estraiamo da "x" i valori che ci interessano
x[-indici] # e i loro "complementari"
```

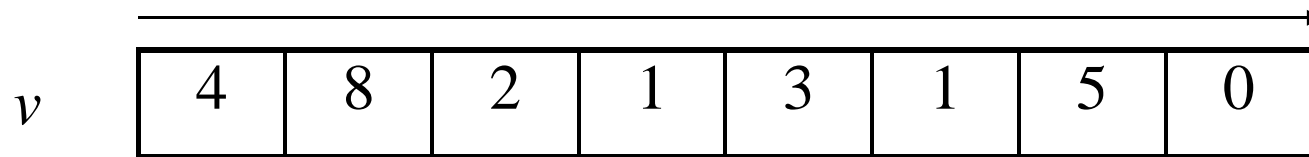
# Esercizi

1. Generare un vettore contenente i primi 100 interi positivi: Calcolare la media, la varianza e la deviazione standard dei suoi elementi (funzioni `mean`, `var`, `sd`)
2. Ripetere il precedente esercizio con un vettore di 100 numeri random estratti da una distribuzione normale standard (si veda la funzione `rnorm`)
3. (a) Costruire una sequenza  $s$  costituita da 3 ripetizioni in sequenza della stringa "CGCT".  
(b) Estrarre dalla sequenza ottenuta una sottosequenza  $sub$  in cui compaiano tutti gli elementi di  $s$  eccetto la lettera C.  
(c) Aggiungere in coda alla sequenza ottenuta 3 valori NA.  
(d) Riottenere la sequenza  $sub$  in (b) tramite la funzione `is.na()`
4. Generare un vettore  $vet$  di 100 elementi casuali estratti secondo la distribuzione uniforme in  $[0,1]$  (vedi `runif`). Estrarre da  $vet$  un vettore  $subvet$  i cui elementi abbiano valore  $v > 0.2$  e valore  $v < 0.6$ . Estrarre da  $subvet$  gli elementi di indice pari ed assegnarli al vettore  $w$ . Trasformare  $w$  in modo che i suoi elementi siano normalizzati rispetto al loro valore medio ed alla deviazione standard.



# Array e matrici come generalizzazioni multidimensionali di vettori (1)

- I vettori sono sequenze ordinate di elementi omogenei. Un vettore  $v$  è rappresentabile tramite una *struttura unidimensionale*:



- Essendo strutture unidimensionali è possibile accedere o modificare un elemento di un vettore utilizzando un *unico indice*:

```
> v[2]
```

```
8
```

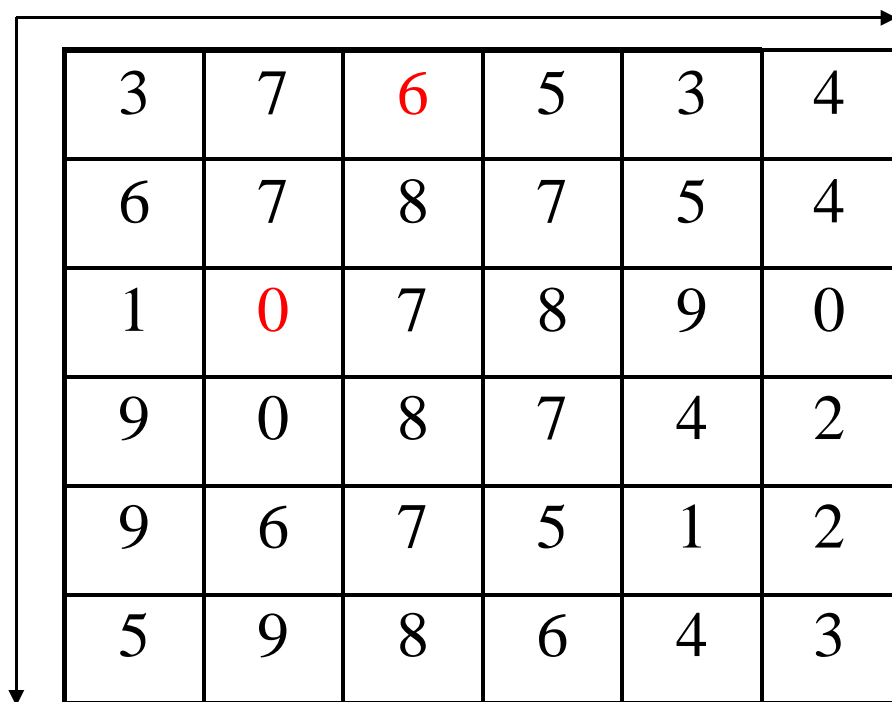
```
> v[1] <- 7
```

```
> v[1]
```

```
7
```

## Array e matrici come generalizzazioni multidimensionali di vettori (2)

- In R è possibile rappresentare *estensioni bidimensionali* di vettori (**matrici**)



3	7	6	5	3	4
6	7	8	7	5	4
1	0	7	8	9	0
9	0	8	7	4	2
9	6	7	5	1	2
5	9	8	6	4	3

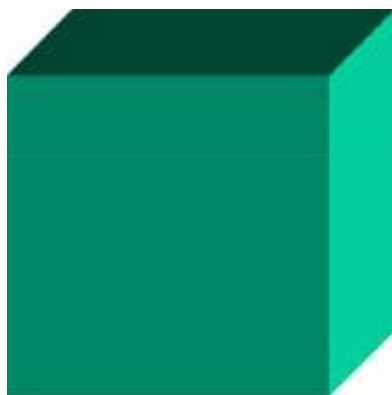
- Essendo strutture bidimensionali, è necessaria una *coppia di indici* per **accedere** o **modificare** un elemento di una matrice :

```
> m[1,3]
# seleziona el. I riga e III colonna
> 6
> m[3,2] <- 1
```

# Array e matrici come generalizzazioni multidimensionali di vettori (3)

- In generale in R è possibile rappresentare *estensioni multidimensionali* di vettori (**array**):

Es: array  
tridimensionale



Per accedere ad un elemento sono necessari 3 indici.

Es:

> a[1,3,4]

- In R si possono costruire array di dimensione arbitraria (limitatamente alle disponibilità di memoria)
- Le **matrici** sono a tutti gli effetti **array bidimensionali**
- Sugli array sono applicabili le medesime operazioni di accesso e modifica (estese a più dimensioni) utilizzabili per i vettori

# Costruzione di matrici (1)

Le matrici possono essere costruite tramite la funzione **matrix** a partire da un vettore esistente

```
> x <- 1:24
> m <- matrix(x,nrow=6) # genera una matrice con 6 righe
#utilizzando gli elementi del vettore x
> m #si noti l' assegnamento dei valori "per colonne"
      [,1] [,2] [,3] [,4]
[1,]    1    7   13   19
[2,]    2    8   14   20
[3,]    3    9   15   21
[4,]    4   10   16   22
[5,]    5   11   17   23
[6,]    6   12   18   24
> length(m)
[1] 24
> dim(m)
[1] 6 4
```

L' attributo dim mostra che la matrice *m* è un array bidimensionale 6 X 4

# Costruzione di matrici (2)

La funzione **matrix** può avere anche altri argomenti:

```
> x <- 1:12
> m <- matrix (x, ncol=4) # si può specificare il n. delle colonne
> m
```

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

```
> m <- matrix (x, ncol=4, byrow=TRUE) # inserimento el. per righe
> m
```

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

```
> m <- matrix (x, ncol=5) # anche per le matrici si applica la “regola del riciclo”
```

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

Warning message:

In matrix(x, ncol = 5) :

data length [12] is not a sub-multiple or multiple of the number of columns [5]

# Le funzioni cbind e rbind

Le matrici possono essere costruite anche tramite le funzioni **cbind** ed **rbind**.

**cbind** forma matrici legando insieme vettori o matrici “orizzontalmente”:

```
> x <- 1:3
> y <- 4:6
> m <- cbind(x,y)
>
```

	m	x	y
[1,]		1	4
[2,]		2	5
[3,]		3	6

**rbind** forma matrici legando insieme vettori o matrici “verticalmente”:

```
> x <- 1:3
> y <- 4:6 rbind(x,y)
> m <-
>
```

	m [,1]	[,2]	[,3]
x	1	2	3
y	4	5	6

Se i vettori costituenti non sono della stessa lunghezza si applica la regola

# Operazioni con le matrici (1)

## *Somma e prodotto con costanti*

```
> x <- 1:12
> A <- matrix(x,nrow=3)
> A + 2
      [,1] [,2] [,3] [,4]
[1,]    3    6    9   12
[2,]    4    7   10   13
[3,]    5    8   11   14
> B <- A * 2
> B
      [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
```

## *Somma e prodotto “elemento per elemento”*

```
> A+B
      [,1] [,2] [,3] [,4]
[1,]    3   12   21   30
[2,]    6   15   24   33
[3,]    9   18   27   36
> A*B
      [,1] [,2] [,3] [,4]
[1,]    2   32   98  200
[2,]    8   50  128  242
[3,]   18   72  162  288
```

# Operazioni con le matrici (2)

Prodotto di matrici “righe X  
colonne”: Operatore %\*%

Si ricordi che A e B sono matrici  
3X4:

```
> A%*%B # il numero delle  
# colonne di A deve essere  
# uguale al numero di righe  
# B!
```

Error in A %\*% B : non-  
conformable arguments

```
> A%*%t(B) # t(B) indica la  
# trasposta di B
```

	[,1]	[,2]	[,3]
[1,]	332	376	420
[2,]	376	428	480
[3,]	420	480	540

Prodotto di matrici per vettori:

```
> z <- 1:4
```

```
> A %*% z
```

```
 [,1]
```

```
[1,] 70
```

```
[2,] 80
```

```
[3,] 90
```

```
> z %*% A
```

Errore in z %\*% A : gli argomenti  
non sono compatibili

```
> z %*% t(A)
```

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

```
[1,] 70 80 90
```

```
>
```



# Accesso agli elementi di array e matrici

Le regole di accesso per array e matrici seguono quelle già viste per i vettori, considerando però l'esistenza di più indici e quindi la possibilità di utilizzare un vettore per ogni dimensione:

- Vettori di **indici interi positivi**
- Vettore di **indici interi negativi**
- Vettore di **indici logici**
- Vettori di **indici a caratteri**

Si utilizza quindi un vettore di indici *per ogni dimensione* dell'array

## Esempi di accesso agli elementi di una matrice

```
> m <- matrix (1:12,nrow=2)
> m
  [,1] [,2] [,3] [,4] [,5] [,6]
[1,]  1  3  5  7  9 11
[2,]  2  4  6  8 10 12
> m[1,2]
[1] 3
> m [1,3:4] # el. I riga,col. 3 e 4
[1] 5 7
> m [1:2,4:6] # I e II riga, col.
# dalla IV alla VI
  [,1] [,2] [,3]
[1,]  7  9 11
[2,]  8 10 12
> m[1,] #tutti gli el. della I riga
[1] 1 3 5 7 9 11
> m[,3] #tutti gli el. III col.
[1] 5 6
> m[,c(1,2,6)]
  [,1] [,2] [,3]
[1,]  1  3 11
[2,]  2  4 12
```

```
> m[,-4] # esclusione el. IV
# colonna
  [,1] [,2] [,3] [,4] [,5]
[1,]  1  3  5  9 11
[2,]  2  4  6 10 12
> m[m>4] # el.>4 (si ottiene un
# vettore)
[1] 5 6 7 8 9 10 11 12
> dimnames(m) <- list(c("r1","r2"),
paste("c",1:6,sep="")) # viene
# dato un nome alle componenti
> m
  c1 c2 c3 c4 c5 c6
r1  1  3  5  7  9 11
r2  2  4  6  8 10 12
> m["r1","c2"] # vettori indice
# a caratteri
[1] 3
> m["r1",]
c1 c2 c3 c4 c5 c6
1  3  5  7  9 11
```

# Esercizi

1. Costruire una matrice 10X10 composta da numeri casuali in almeno 2 modi diversi utilizzando le funzioni `matrix` ed `array`.
2. Costruire 2 matrici di caratteri a piacere  $x$  e  $y$ , la prima di dimensione 3X4, la seconda di dimensione 5X3. Modificare con un unico assegnamento la matrice  $x$  in modo da sostituire le sue 2 prime colonne con le ultime 2 righe di  $y$ .
3. Costruire due matrici  $M$  ed  $N$  che abbiano entrambe 5 colonne e numero di righe differenti (e.g.,  $M$  3x5,  $N$  4x5) . Costruire, se possibile, tramite `rbind` una matrice di 5 colonne che abbia come righe le righe di entrambe le matrici. Utilizzando  $M$  ed  $N$ , è possibile costruire una matrice tramite `cbind`?
4. Date due matrici quadrate  $A$  e  $B$  di dimensione 3X3 costituite da numeri casuali, calcolarne il prodotto elemento per elemento e “righe per colonne”. Scelto un vettore  $b$  di 3 elementi, risolvere il sistema lineare  $Ax=b$  che ne deriva.

# Fattori

- Strutture dati per rappresentare valori che *possono assumere solo valori discreti*
- In R i diversi valori che possono assumere i dati discreti sono definiti come *livelli*
- Es: dati qualitativi, o dati ordinali.
- Spesso si usano per “etichettare” dati

# Costruzione di fattori

```
> trt <- factor(rep(c("Control", "Treated"), c(3,4)))
# I fattori si possono costruire con factor utilizzando un vettore esistente
# fac <- c(rep("Control", 3), rep("Treated", 4))
# trt <- factor(fac)
> trt
[1] Control Control Control Treated Treated Treated
     Treated
Levels: Control Treated

> levels(trt) # visualizza i livelli del fattore
[1] "Control" "Treated"

> str(trt) # visualizza in modo succinto la struttura
# di un oggetto R
Factor w/ 2 levels "Control"," Treated": 1 1 1 2 2 2 2

> summary(trt) # la funzione summary fornisce una
# tabella della frequenze dei due livelli Control e #
Treated
Control Treated
3         4
```

# Esercizi

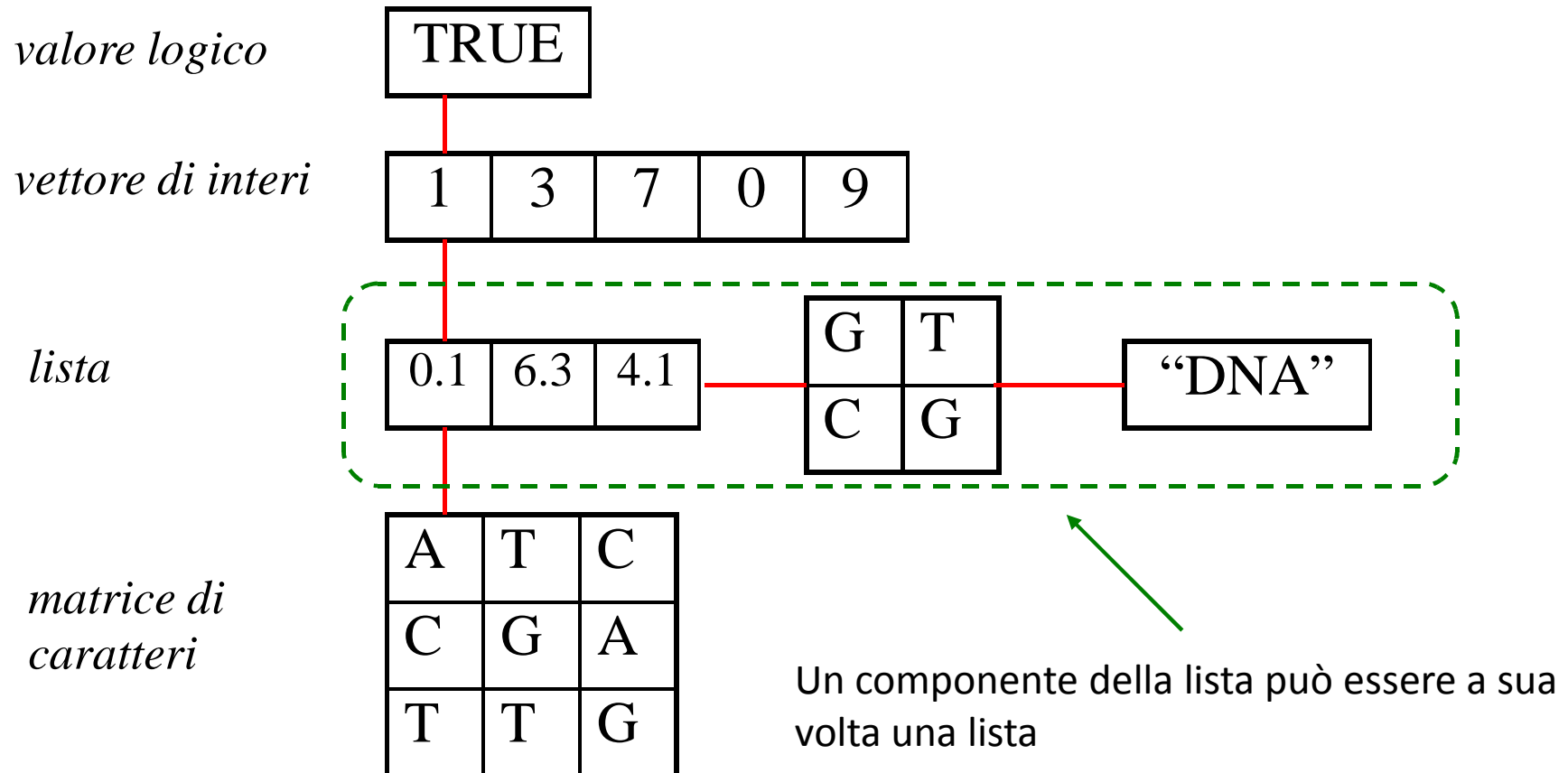
1. Costruisci un fattore di 27 elementi (stringhe), caratterizzato da 3 diversi livelli
2. Costruisci un fattore ordinato di 15 elementi di livelli basso < medio < alto.  
Genera un corrispondente vettore di misurazioni di lunghezza 15 e calcola il valor medio e la deviazione standard (funzione sd) di ogni livello tramite la funzione tapply.

# Lista come insieme ordinato di oggetti eterogenei (1)

- Le liste rappresentano un *insieme ordinato di oggetti* (componenti)
- Le componenti possono non essere dello stesso tipo o modo.  
Quindi le liste rappresentano *insiemi di oggetti eterogenei*.
- I componenti possono essere ad es: un vettore numerico, un valore logico, una matrice, un array di caratteri, una funzione o anche un' altra lista.
- La lista è quindi una *struttura dati ricorsiva*, poichè una sua componente può essere a sua volta una lista (e la lista componente può avere come componente un' altra lista).

# Lista come insieme ordinato di oggetti eterogenei (2)

Una lista composta da oggetti eterogenei: un valore logico, un vettore di interi, un'altra lista ed una matrice di caratteri





# Costruzione di una lista

Per costruire le liste si usa la funzione **list**:

I componenti delle liste sono sempre **numerati**:

```
> li <- list(TRUE, c(1,3,7,0,9))
> li
[[1]]
[1] TRUE
[[2]]
[1] 1 3 7 0 9
```

E' però possibile assegnare alle componenti un **nome**:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9))
> li
$val
[1] TRUE
$vector
[1] 1 3 7 0 9
```

# Accesso alle componenti di una lista

Esistono 3 modalità di accesso alle componenti di una lista:

1. Accesso tramite *indice numerico*
2. Accesso tramite *il nome delle componenti*
3. Accesso tramite *indice "a caratteri"*

# Accesso tramite indice numerico

I componenti delle liste sono numerati ed è possibile accedere ad essi tramite indice numerico racchiuso fra doppie parentesi quadre.

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9), m=matrix(1:12,nrow=2))
> li[[1]]
[1] TRUE
> li[[2]]
[1] 1 3 7 0 9
> li[[3]]
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
```

# Gli operatori “[[]]” e “[ ]”

- L' accesso alle componenti tramite gli operatori “[[]]” e “[ ]” produce risultati sostanzialmente differenti.
- “[[]]” è l' operatore che seleziona l' oggetto contenuto nella lista ( e l' eventuale nome associato all' oggetto non è incluso)
- “[ ]” è l' operatore che seleziona una sottolista (si riferisce quindi ad un elemento di modo “lista”, e l' eventuale nome associato all' oggetto viene incluso )

# Accesso tramite il nome delle componenti

Se le componenti hanno un nome è possibile accedere ad esse direttamente tramite il nome stesso

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9), m=matrix(1:12,nrow=2))
```

```
> li$val
```

```
[1] TRUE
```

```
> li$vector
```

```
[1] 1 3 7 0 9
```

Quindi `li$val` è equivalente a `li[[1]]` e `li$vector` a `li[[2]]`

Tramite la notazione `lista$nome` è possibile accedere anche ai singoli elementi delle componenti:

```
> li$vector[4]
```

```
[1] 0
```

```
> li$m[1,2]
```

```
[1] 3
```

# Accesso tramite indice “a caratteri”

Se le componenti hanno un nome è possibile accedere ad esse tramite indice “a caratteri”

Es:

```
> li <- list(val=TRUE, vector=c(1,3,7,0,9), m=matrix(1:12,nrow=2))
> li[["m"]]
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
```

Questa modalità può essere utile quando il nome della componente è memorizzato in un' altra variabile:

```
> v <- "vector"; li[[v]]
[1] 1 3 7 0 9
```

# Allungare e concatenare liste

Come qualsiasi altro oggetto accessibile tramite indici, le liste possono essere estese, aggiungendo specifiche componenti:

```
> li<-list(0.6798,  
paste(c(rep("A",4),rep("T",4)),  
collapse=""))  
> li[3] <- list(TRUE)  
> li  
[[1]]  
[1] 0.6798  
[[2]]  
[1] "AAAATTTT"  
[[3]]  
[1] TRUE
```

La concatenazione di liste è possibile tramite la funzione **c**:

```
> li1 <- list(TRUE,2)  
> li2 <- list("pippo")  
> li3<-list( c(1,2,3),  
c("T","A","C"))  
> li123 <- c(li1,li2,li3)  
> li123  
[[1]]  
[1] TRUE  
[[2]]  
[1] 2  
[[3]]  
[1] "pippo"  
[[4]]  
[1] 1 2 3  
[[5]]  
[1] "T" "A" "C"
```

# Esercizi

1. Costruire una lista *li* composta da una matrice numerica 4X4, da un vettore di caratteri con 32 elementi, dalla stringa "topo", e da un'ulteriore lista composta da un vettore di 10 elementi numerici e dal valore logico FALSE.
2. Si estragga dalla lista  

```
li<-list(m=matrix(rnorm(64),nrow=8),s=c(rep("T",3),rep("G",5)))
```

(a) la II colonna della matrice (b) le "G" del vettore *s*.  
Si aggiunga quindi alla lista un vettore composto da 10 numeri casuali.
3. Accedi in 3 modi diversi alla II componente della lista *li* dell'es.2
4. Spiega la differenza fra le due diverse modalità di accesso al primo elemento della lista *li*, *li[1]* e *li[[1]]*. Tramite la funzione *diag* si estragga la diagonale della matrice memorizzata nella lista *li*.



# Data frame come struttura per rappresentare insiemi di dati eterogenei (1)

- Un *data frame* può essere considerato come una matrice le cui colonne rappresentano dati eterogenei:

Dati:	val.num	val.car.	val. log.	val.num	val.car.
Dato1	3.45	ATTA	TRUE	0.45	CCAT
Dato2	5.67	TGAT	FALSE	0.91	TATT
Dato3	1.45	TATA	FALSE	3.78	CCCC
Dato4	4.56	TGAG	TRUE	8.03	GAGA
Dato5	8.09	CCCG	TRUE	8.09	AGAG
Dato6	3.11	CATG	TRUE	4.56	ATAT
Dato7	1.40	TGAG	FALSE	1.80	GCTA
Dato8	7.73	GGAC	TRUE	5.90	TGAT

- Formalmente è una lista di *classe* data.frame

# Data frame come struttura per rappresentare insiemi di dati eterogenei (2)

- Le colonne del data frame rappresentano *variabili* i cui modi ed attributi possono essere differenti (le matrici e gli array sono invece costituiti da elementi omogenei per modo ed attributo):

**Data frame**

Dati:	val.num	val.car.	val. log.	val.num	val.car.
Dato1	3.45	ATTA	TRUE	0.45	CCAT
Dato2	5.67	TGAT	FALSE	0.91	TATT
Dato3	1.45	TATA	FALSE	3.78	CCCC
Dato4	4.56	TGAG	TRUE	8.03	GAGA
Dato5	8.09	CCCG	TRUE	8.09	AGAG
Dato6	3.11	CATG	TRUE	4.56	ATAT
Dato7	1.40	TGAG	FALSE	1.80	GCTA
Dato8	7.73	GGAC	TRUE	5.90	TGAT

**Matrice (array bidimensionale)**

Dati:	val.num	val.num	val.num	val.num	val.num
Dato1	3.45	1.20	2.54	0.45	1.45
Dato2	5.67	2.95	5.44	0.91	0.95
Dato3	1.45	5.21	3.60	3.78	6.78
Dato4	4.56	8.00	2.12	8.03	8.73
Dato5	8.09	1.65	6.00	8.09	6.65
Dato6	3.11	2.58	3.98	4.56	3.56
Dato7	1.40	0.95	2.72	1.80	5.21
Dato8	7.73	8.82	4.43	5.90	3.18

- Un data frame può essere visualizzato come una matrice e si può accedere ai suoi elementi utilizzando indici (come per le matrici ordinarie)

# Componenti dei data frame

- Formalmente i **data frame** sono *liste di classe data.frame*
- I *componenti* (colonne) del data frame possono essere costituiti da:
  - Vettori (numerici, a caratteri, logici)
  - Fattori
  - Matrici numeriche
  - Liste
  - Altri data frame

# Costruzione dei data frame

I data frame sono costituiti tramite la funzione **data.frame**:

```
> x<-1:4
> y<-5:8
> z<-paste("A",1:4,sep="")
> da.fr<-data.frame(x,y,z)
> da.fr
  x y  z
1 1 5 A1
2 2 6 A2
3 3 7 A3
4 4 8 A4
```

```
> m1 <-matrix(1:12,nrow=2)
> v <- c("A","C")
> daf3<-data.frame(m1,v)
> daf3
  X1 X2 X3 X4 X5 X6 v
1  1  3  5  7  9 11 A
2  2  4  6  8 10 12 C
> v1<- c("A","C","G")
> daf4<-data.frame(m1,v1)
Error in data.frame(m1,v1)      :
arguments imply differing
numberof rows: 2, 3
```

## Accesso alle componenti ed agli elementi dei data frame

Esistono due modalità generali di accesso alle componenti ed agli elementi dei data frame:

1. I data frame sono liste, e quindi è possibile accedere ad essi secondo le *modalità di accesso tipiche delle liste* stesse.
2. Come classe data frame, sono definiti *operatori di accesso tramite vettori di indici*, simili a quelli utilizzati per le matrici e gli array.

# Accesso alle componenti dei data frame

Essendo liste, è possibile accedere alle componenti dei data frame secondo le modalità tipiche delle liste:

1. Accesso tramite indice numerico
2. Accesso tramite il nome delle componenti
3. Accesso tramite indice “a caratteri”

Es:

```
> x<-1:4; y<-5:8
```

```
> z<-paste("A",1:4,sep="")
```

```
> da.fr<-data.frame(x,y,z)
```

1. Accesso tramite indice numerico:

```
> da.fr[[1]]
```

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

```
> da.fr[1]
```

```
x
```

```
1 1
```

```
2 2
```

```
3 3
```

```
4 4
```

2. Accesso tramite il nome delle componenti:

```
> da.fr$x
```

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

3. Accesso tramite indice “a caratteri”:

```
> da.fr["x"]
```

```
x
```

```
1 1
```

```
2 2
```

```
3 3
```

```
4 4
```

```
> da.fr[["x"]]
```

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

# Accesso alle componenti tramite vettori di indici

Sono definiti operatori di accesso specifici per la classe *data.frame*: si tratta di vettori di indici con una semantica simile a quella delle matrici ordinarie:

Es:

```
> x<-1:4; y<-5:8
> z<-paste("A",1:4,sep=" ")
> df<-data.frame(x,y,z)
> df
  x y  z
1 1 5 A1
2 2 6 A2
3 3 7 A3
4 4 8 A4
```

```
> df[1,2]
[1] 5
> df[2,2:3]
  y  z
2 6 A2
> df[3,]
  x y  z
3 3 7 A3
> df[2:4,1:2]
  x y
2 2 6
3 3 7
4 4 8
```

# Estrazione “logica” di osservazioni da data frame

```
> v <- sample(rep(c("A", "T", "C", "G"), c(5, 5, 5, 5)))
> m <- matrix(runif(10), nrow=5)
> li <- list(m1=matrix(rnorm(30), nrow=5), m2=matrix(sample(25),
ncol=5), m3=matrix(runif(50), nrow=5))
> df <- data.frame(v, m, li)
```

# estrai da df solo le osservazioni la cui variabile m.3 > 0.56

```
> df[df$m3.1 > 0.56, ]
```

Equivalentemente si può usare la funzione *subset*:

```
> subset(df, m3.1 > 0.56)
```

Se si vogliono selezionare elementi da un insieme si può usare l'operatore *%in%*:

```
> subset(df, v %in% c("T", "A"))
```



# Esercizi

1. Costruire un data frame *da.fr* che abbia come componenti un vettore numerico casuale *v* di lunghezza 20, una matrice casuale *m* con 4 colonne ed una lista *i* cui componenti siano 3 matrici a piacere.
2. Costruire una lista che abbia come componenti 3 vettori a caratteri. Trasformare la lista in un data frame tramite la funzione *as.data.frame*. Quali sono le restrizioni che si devono applicare alle liste perchè siano dei data frame?
3. Selezionare dal data set *iris* le osservazioni relative alle specie “virginica” con `Petal.Length>5.890`.
4. Tramite la funzione *summary* ricavare informazioni statistiche di base sulla specie “versicolor” del data set *iris*.