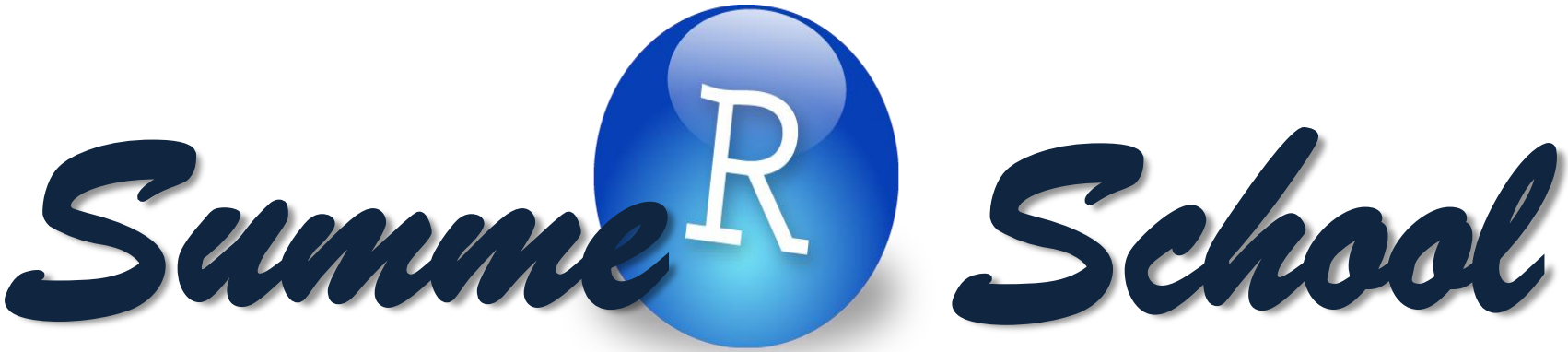


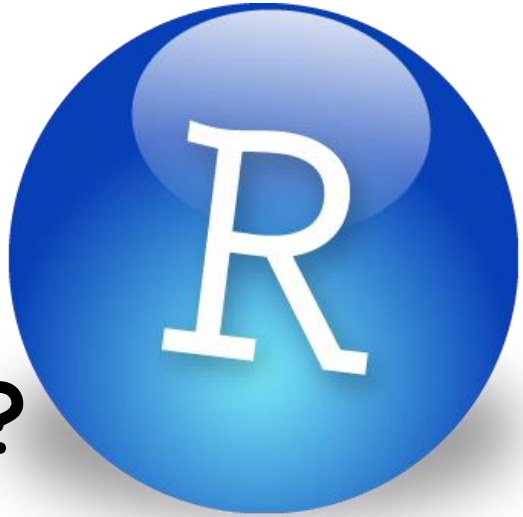
Davide Gentilini
Antonino Oliveri

Pavia 4 Giugno 2018

LEZIONE 1

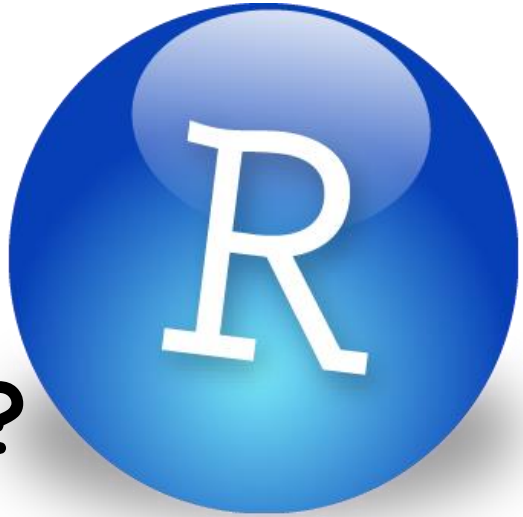
INTRODUZIONE AD R

Cosa è R ?



Piuttosto che definire R come un software statistico, esso deve essere definito come un AMBIENTE, ovvero un insieme di macro, librerie, oggetti che possono essere utilizzati per la gestione, l'analisi dei dati e la produzione di grafici

Perché R ?



- E' un linguaggio di scripting
- Non necessita di compilazione
- Adatto ad uso interattivo
- Funzioni ad alto livello
- Multipiattaforma
- Gratuito (derivato da S-Plus)
- Open source
- Specializzato per uso statistico
- Pensato per descrivere modelli anche molto complessi
- E’ un linguaggio object-oriented

Pro e Contro

- Alto livello implica:
  - Funzioni potenti, ma solo per quello che R "sa fare".
  - Poco potente e flessibile per il resto (meglio linguaggi di programmazione come C++, Java, etc.)
  - Ottimo per applicazioni statistiche di piccole e medie dimensioni

- La licenza open source prevede che il software venga fornito sia nel formato compilato (eseguibile) che in quello sorgente

Un po' di Storia..

Dagli anni settanta sono state sviluppate tecniche statistiche che richiedono un notevole supporto computazionale al fine di essere fruibili.
Negli anni novanta i Bell Laboratories hanno deciso di sviluppare un nuovo ambiente per l'analisi statistica in grado di permettere, oltre l'applicazione delle metodologie conosciute, anche la sperimentazione di nuovi modelli ed idee statistiche. Nasce quindi il linguaggio S

Come si ottiene R ?

R è gratuito e si può scaricare

https://cran.r-project.org/

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (Friday 2017-04-21, You Stupid Darkness) R-3.4.0.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

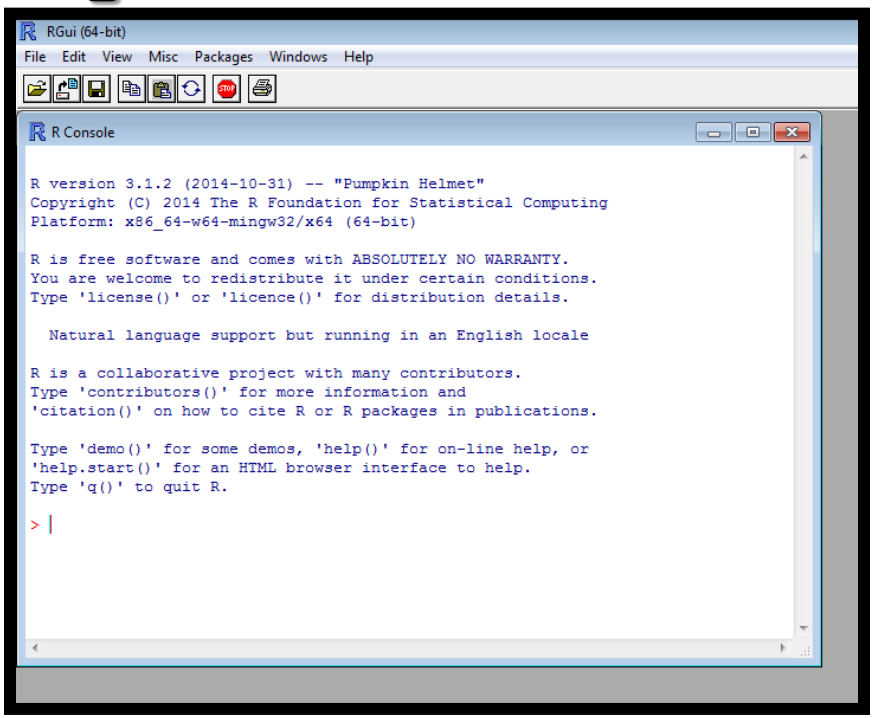
Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

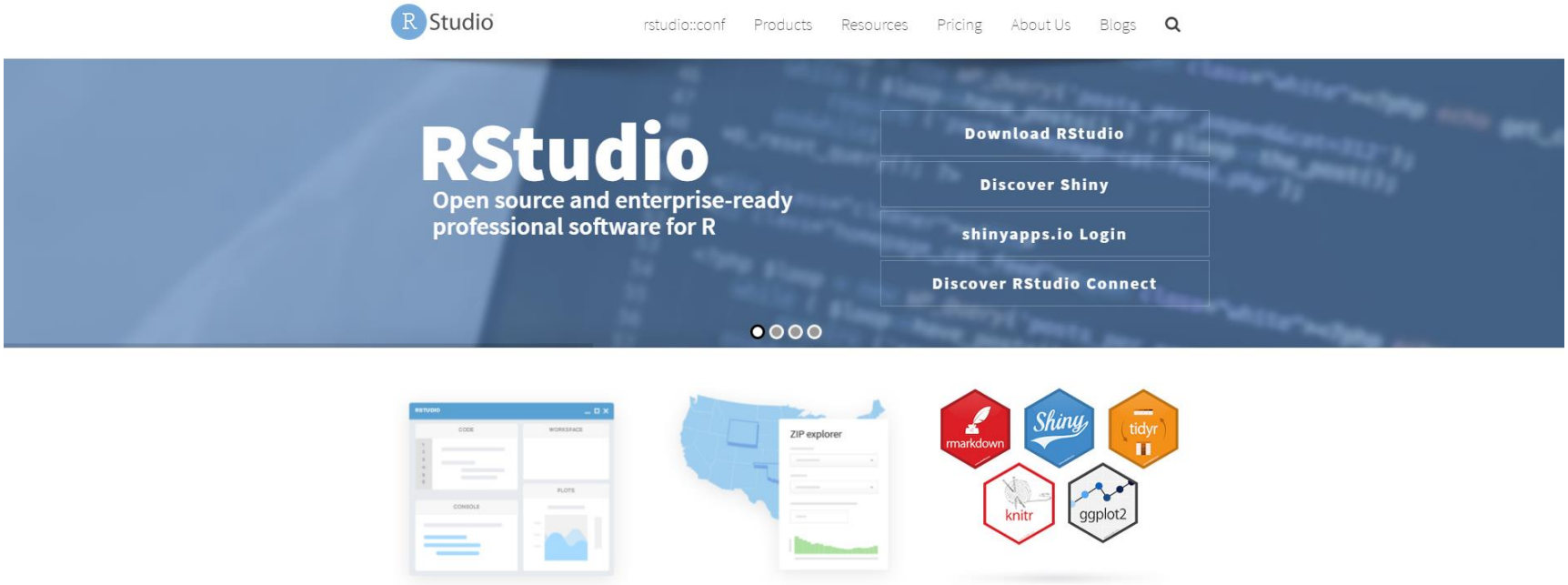
Come si presenta R ?

R è stato pensato essere utilizzato tramite riga di comando.
Esistono anche interfacce grafiche, ma sono limitate alle semplici operazioni ricorrenti (leggere files, etc.)
Sotto windows usiamo la R-console
Sotto unix usiamo la shell.

Come si presenta R ?



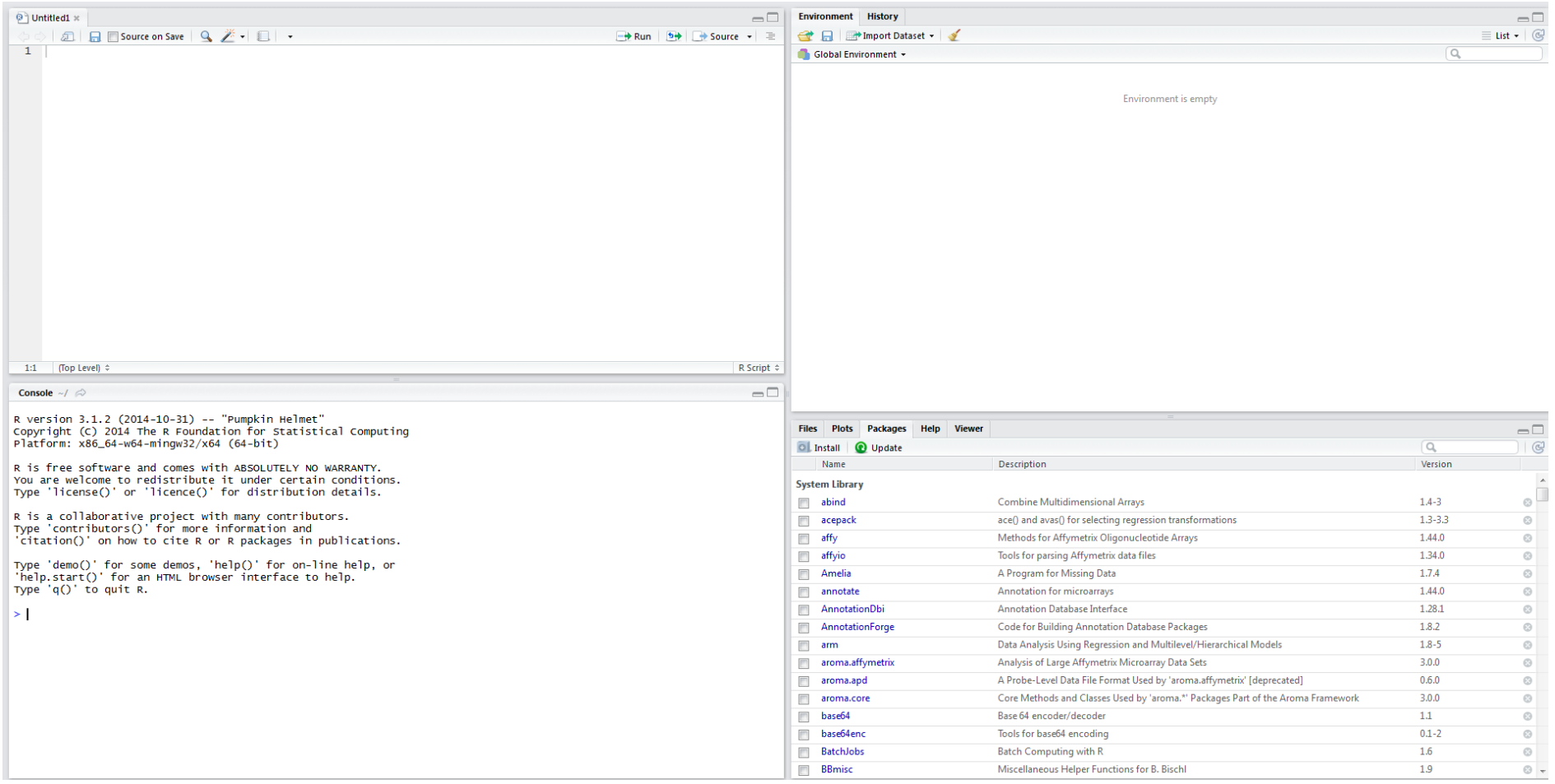
Interfaccia Grafica minimale con alcune funzioni nel menu'



RStudio rappresenta una evoluzione di R con numerose utilità implementate e una interfaccia più sviluppata

https://www.rstudio.com/products/rstudio/download/

Come si presenta R ?



WORK SPACE

`> getwd()`	Capire la directory in cui si sta lavorando
`> setwd(/directory/)`	Cambiare la directory
`> ls()`	Lista oggetti nel workspace
`> history()`	Mostra ultimi 25 comandi
`> history(max.show=Inf)`	Mostra tutti i comandi
`> savehistory(file="nome.file")`	Salva la history
`> loadhistory(file="nome.file")`	Carica la history
`> save.image()`	Salva area di lavoro (.Rdata)
`> save(lista_oggetti,"NOME.RData")`	Salva oggetti particolari(.Rdata)
`> load("NOME.RData")`	Carica un oggetto .RData
`> dir.create("/directory/")`	Crea una cartella
`> rm(oggetto)`	Cancella oggetto dall'area di lavoro
`> rm(list=ls())`	Cancella tutti gli oggetti dall'area di lavoro

I PACCHETTI

R consente di integrare una serie di funzioni e pacchetti già pronti e avvalersi in questo modo del lavoro di altri

```
> install.packages("NOME PACCHETTO")        Installa pacchetto da repository
> library(NOME PACCHETTO)                   Carica pacchetto installato
> install.packages("/PATH/PACCHETTO.zip")   Installa da file locale .zip
> search()                                  Mostra pacchetti caricati
```

E' possibile utilizzare il menù per compiere queste operazioni

I PACCHETTI

Il programma presenta una serie di pacchetti aggiuntivi che possono aumentare notevolmente le potenzialità di R. Ne esistono dio tre tipi:

- Pacchetti automaticamente installati e avviati
- Pacchetti automaticamente installati ma non avviati
- Pacchetti reperibili in rete

Per conoscere l'elenco dei pacchetti instalati si usa il comando

```
library()
```

Se vogliamo caricare un pacchetto in particolare usiamo il comando

```
library(NOME-PACCHETTO)
```

Sintassi di un comando R

Variabile/oggetto <- comando(par1, par2, …)

- **Il simbolo <- è usato al posto dell'uguale (=)**
- **R supporta il segno = ma ne sconsiglia l'uso**
- **E' disponibile anche il comando assign**
- **Se non specifichiamo la variabile destinazione il risultato viene tenuto nella variabile .Last.value**
- **Premendo i tasti freccia su e freccia giu possiamo navigare nella command history (la lista dei comandi eseguiti precedentemente)**

LE VARIABILI

Visualizzare il contenuto di una variabile

- Per visualizzare il contenuto di una variabile basta digitarne il nome
- Per visualizzare tutte le variabili esistenti si può usare il comando

```
objects()
ls()
```

Assegnare il contenuto di una variabile

```
X<-1
X<-"CASA"
```

Vedere la struttura di una variabile

```
str(x)
class(x)
```

Pulizia del Work Space

- Ogni assegnazione in R sovrascrive il contenuto della variabile di destinazione.
- Gli oggetti possono essere rimossi con il comando rm(oggettoDaCancellare)
- Rimozioni multiple rm(pippo,pluto, x1)
- con rm(list=ls()) svuotiamo il workspace

GLI OPERATORI

Gli operatori matematici

- **Addizione** +
- **Sottrazione** -
- **Moltiplicazione** *
- **Divisione** /
- **Elevamento a potenza** ^

Addizione

- **Esempio:**

```
> 1+2
[1] 3
> x
[1] 1 2 3 4 5 6 7
> y
[1] -3.2 -2.2 -1.2 -0.2  0.8  1.8  2.8
> x+y
[1] -2.2 -0.2  1.8  3.8  5.8  7.8  9.8
```

Divisione

- **Esempio:**

```
> 21/7
[1] 3
> 2/0
[1] Inf
> -1/0
[1] -Inf
> 0/0
[1] NaN
> # NaN = Not a Number
> x
[1] 1 2 3 4 5 6 7
> y
[1] -3.2 -2.2 -1.2 -0.2  0.8  1.8  2.8
> y/x
[1] -3.20 -1.10 -0.40 -0.05  0.16  0.30  0.40
```

GLI OPERATORI

Gli operatori Relazionali

- **Maggiore** >
- **Minore** <
- **Uguale** ==
- **Maggiore o Uguale** =>
- **Minore o Uguale** =<
- **Diverso** !=

Gli operatori Logici

- **AND** &
- **OR** |

GLI OPERATORI

AND

- **Esempio:**

```
> 1&5
[1] TRUE
> x
[1] 0.11 1.20 2.30 4.50 0.00
> x&3
[1]  TRUE  TRUE  TRUE  TRUE FALSE
```

OR

- **Esempio:**

```
> 5|0
[1] TRUE
> x
[1] 0.11 1.20 2.30 4.50 0.00
> x|3
[1] TRUE TRUE TRUE TRUE TRUE
```

GLI OGGETTI

Gli oggetti base usati dal programma R sono i seguenti:

- Vettori
- Matrici
- Array
- Liste
- Fattori
- Serie storiche
- Data frame

I VETTORI

Cosa sono i vettori?

I vettori in R possono essere di varie tipologie:

- numerico
- logico
- complesso
- caratteri

a secondo della tipologia di elementi che li compongono.

I VETTORI

Creazione di un vettore

Creazione di un vettore con la funzione scan

```
scan()
```

```
scan(what="character")
```

La funzione consente di inserire i valori a terminale uno alla volta e procedendo premendo invio. Terminata la lista di elementi si chiude la funzione premendo due volte invio dopo l'ultimo elemento. Nella seconda opzione per creare un vettore non numerico

I VETTORI

Creazione di un vettore

Creazione di un vettore con la funzione c

Con il comando c(elemento1,elemento2,.....) viene creato un vettore tramite la concatenazione degli elementi separati da virgole.

Gli elementi non possono essere contemporaneamente caratteri e numeri. Ad esempio con:

```
x<-c(1.5,2,2.5)
```

Si possono aggiungere nuovi elementi ad un vettore precedentemente creato. Ad esempio con:

```
x<-c(x,3)
```

Per ottenere un vettore di stringhe

```
x<-c("questo","`e","un esempio")
```

I VETTORI

Creazione di un vettore

Creazione di un vettore con la funzione seq()

Con il comando seq () possiamo creare un vettore costituito da una sequenza di numeri. Devono essere specificati 3 elementi per la funzione seq:

1- numero di partenza, 2- ultimo numero, 3- incremento

```
X<-seq(2,20,5)
```

2,7,12,17

Creazione di un vettore con la funzione rep()

Con il comando rep () possiamo creare un vettore costituito da una ripetizione di numeri. Devono essere specificati 2 elementi per la funzione rep:

1- numero da ripetere, 2- numero ripetizioni

```
X<-seq(2,5)
```

2,2,2,2,2

I VETTORI

Attributi di un vettore

```
length(X)                  # Misuro quanti elementi contiene
mode(X)                    # Verifico il modo del vettore
names(X)                   # Visualizzo i nomi del vettore
names(X)<-c("A", "B"…)     # Assegno nomi al vettore
```

```
X[n]                       # visualizzo elemento n
X[n1:n2]                   # visualizzo da n1 a n2
X[c(n1,n2,n3)]             # visualizzo n1 n2 e n3
X[-(n1:n2)]                # visualizzo tranne da n1 a n2
X[X>n]                     # visualizzo elementi > n
X[X>n1 | X<n2]             # visualizzo elementi > n1 o <n2
X[X>n1 & X<n2]             # visualizzo elementi > n1 e <n2
X["a"]                     # visualizzo elementi = a
```

I VETTORI

Ordinare un vettore

- `sort(X)`
- `sort(X,decreasig=T)`

Funzioni elementari di un vettore

```
mean(X)       # calcola media del vettore
min(X)        # calcola minimo del vettore
max(X)        # calcola massimo del vettore
median(X)     # calcola mediana del vettore
range(X)      # calcola range del vettore
quantile(X)   # calcola quantili del vettore
sum(X)        # calcola sommatoria del vettore
sd(X)         # calcola dev standard del vettore
var(X)        # calcola varianza del vettore
cumsum(X)     # calcola somma progressiva del vettore
```

I VETTORI

Funzioni elementari di un vettore

```
union(X,Y)          # unisce due vettori X e Y
intersect(X,Y)      # interseca due vettori X e Y
setdiff(X,Y)        # trova differenze tra due vettori X e Y
is.element(X,Y)     # verifica se X è presente in Y
X%in%Y              # verifica se X è presente in Y
which(X,Y)          # trova posizione in cui c'è X in Y
ave(X,FUN=funz.)    # applica una funzione ad ogni elemento di X
rev(X)              # inverte gli elmenti del vettore
round(X)            # arrotonda gli elementi del vettore
head(X)             # visualizza i primi sei elementi del vettore
```

I FATTORI

I dati qualitativi in R sono gestiti da oggetti chiamati factors

```
as.factor(X)  # rendo elementi del vettore come fattori
```

```
eta <- c("giovane", "anziano", "adulto", "adulto")
facEta <- as.factor(eta)
```

- E'possibile assegnare un ordine ai livelli di un fattore:

```
ordered(facEta, levels=c("giovane", "adulto", "anziano")
```

- E' possibile ricodificare i livelli di un fattore:

```
levels(facEta)<-c(1,2,3)
```

LE MATRICI

Una matrice è una tabella ordinata di elementi dello stesso tipo. Ciascun elemento è univocamente localizzato tramite una coppia di numeri interi: l'indice di riga e quello di colonna

La funzione **matrix()** permette la creazione di una matrice:

```
mtx<-matrix(1:25,5)      # costruisce una matrice di 5 colonne
```

Gli elementi sono disposti per colonne, a meno di non specificare il parametro ,byrow=TRUE.

E' possibile costruire una matrice combinando vettori con le funzioni **cbind()** e **rbind()**

```
mtx<-cbind(V1,V2,V3)     #da una matrice di 3 colonne
mtx<-rbind(V1,V2,V3)     #da una matrice di 3 righe
```

LE MATRICI

Altri modi per costruire matrici

- ```
  emptyMTX<-matrix(,nrow=3,ncol=5)
  emptyMTX[]<-12:21
  ```
- ```
  mtx<-matrix(scan(),5,6)
  ```
- ```
  X<-1:10
  dim(X)<-c(5,2)
  ```
  ```

MATRICI

Funzioni elementari di una matrice

- `dim(mtx)` # indica dimensioni matrice
- `head(mtx)` # visualizza prime sei righe matrice
- `mtx[,1]` # richiama prima colonna matrice
- `mtx[1,]` # richiama prima colonna matrice
- `mtx[1:3,1:5]` # richiama prime tre righe e 5 colonne
- `mtx[c(1,3),c(1,5)]` # richiama riga 1 e 3, colonne 1 e 5
- `t(mtx)` # traspone la matrice

- `ave(X,FUN=funz.)` # applica una funzione ad ogni elemento di X
- `rev(X)` # inverte gli elmenti del vettore
- `round(X)` # arrotonda gli elementi del vettore
- `head(X)` # visualizza i primi sei elementi del vettore

GLI ARRAY

Gli array sono costituiti da dati raggrupati in tabelle ad entrata multipla ordinate in funzione dei dati che contengono. Sono utili perché consentono l'analisi congiunta di più di un'unità

- Per costruire un array si utilizza la funzione **array()**:

```
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

- Le dimensioni indicano numero righe colonne ed elementi della lista

```
Titanic
   , , Age = Child, Survived = No

       Sex
   Class  Male Female
   1st     0      0
   2nd     0      0
   3rd    35     17
   Crew    0      0

   , , Age = Adult, Survived = No

       Sex
   Class  Male Female
   1st   118      4
   2nd   154     13
   3rd   387     89
   Crew  670      3

   , , Age = Child, Survived = Yes

       Sex
   Class  Male Female
   1st     5      1
   2nd    11     13
   3rd    13     14
   Crew    0      0

   , , Age = Adult, Survived = Yes
```

Notiamo che l'array Titanic presenta le variabili

- class
- sex
- age
- survived

aventi le seguenti modalità:

- class = 1 2 3 4
- sex = m f
- age = child adult
- survived = no Yes

LE LISTE

Le liste rappresentano un insieme ordinato di oggetti (componenti).

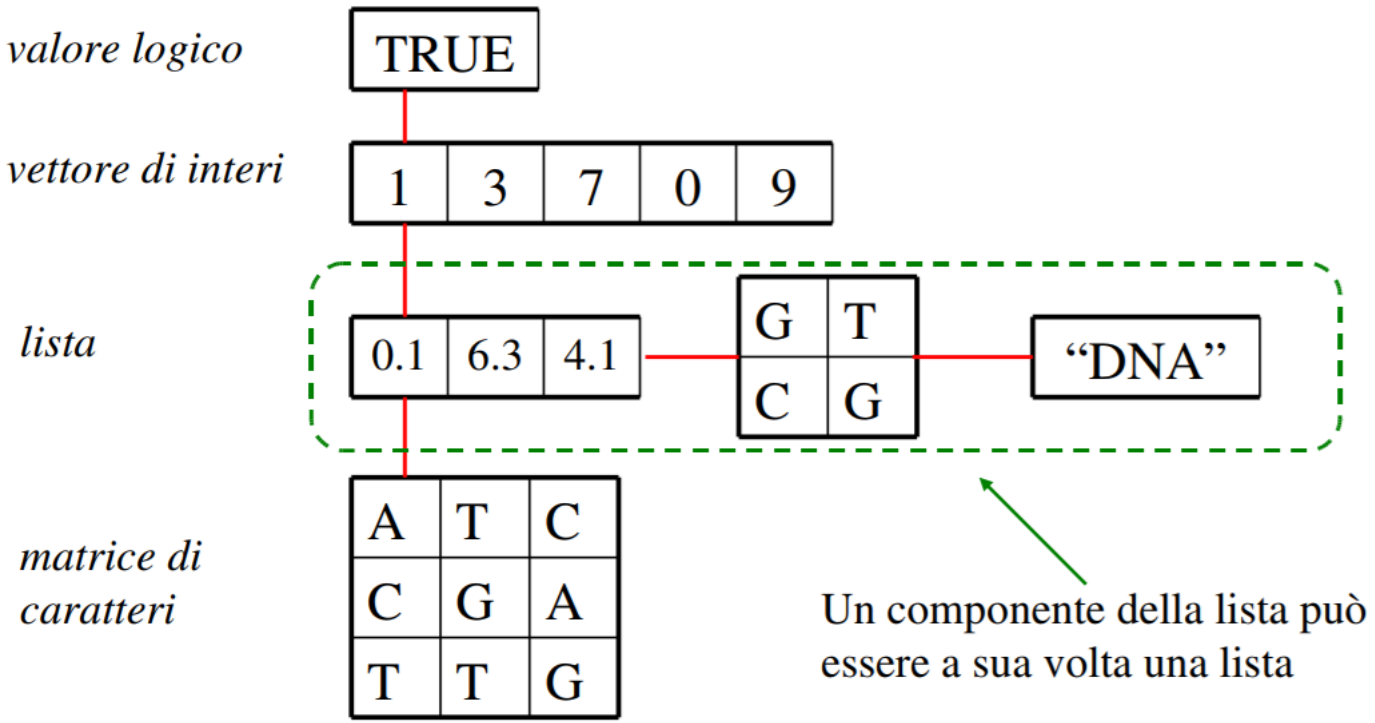
Le componenti possono non essere dello stesso tipo o modo. Quindi le liste rappresentano insiemi di oggetti eterogenei.

I componenti possono essere ad es: un vettore numerico, un valore logico, una matrice, un array di caratteri, u na 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

LE LISTE

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



LE LISTE

- Per costruire un array si utilizza la funzione **list()**:

```
li <- list(TRUE, c(1,3,7,0,9))
li<-list(val=TRUE,vec=c(1,3,7,0,9)) # aggiungo nomi agli elementi
```

Accesso alle componenti di una lista
Esistono 3 modalità di accesso alle componenti di una lista:

1.Accesso tramite indice numerico

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

```
li [[1]]
```

2.Accesso tramite il nome delle componenti

E' possibile accedere anche ai singoli elementi delle componenti:

```
li[[2]][2]
```

```
li$vec[4]
```

3.Accesso tramite indice "a caratteri"

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

```
v<"vec";  li[[v]]
```

LE LISTE

Gli operatori [[]] e []

L' accesso alle componenti tramite gli operatori [[]] e [] produce risultati sostanzialmente differenti.

[[]] è l' operatore che seleziona l' oggetto contenuto nel la 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)

DATAFRAME

I dataframe sono oggetti simili alle matrici ma che contengono colonne con ogeetti di varia natura. Tipicamente vettori, fattori o vettori logici.

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)

- Per costruire un dataframe si utilizza la funzione `data.frame()`:

- `data.frame(vector1,vector2,..vectorn)`

In questo caso vengono combinati in colonna

cbind ed rbind sono comunemente usati come per le matrici

DATAFRAME

- Per estrarre gli elementi di un dataframe si utilizzano i comadi visti per le matrici
  - Per vedere la struttura di un dataframe e i modi in cui si presentano le variabili si utilizza la funzione

```
str(DATAFRAME)
```

  - Per rendere ogni colonna un vettore si utilizza il comando

```
attach(DATAFRAME)
```

```
detach(DATAFRAME)
```

  - In alternativa le variabili di un dataframe si visualizzano come

```
DATAFRAME$Var1
```

DATAFRAME

Altre funzioni

```
dim(DATAFRAME)          # visualizza dimensioni DF
ncol(DATAFRAME)         # visualizza n colonne DF
nrow(DATAFRAME)         # visualizza n righe DF
head(DATAFRAME)         # visualizza 6 righe DF
rowSums(DATAFRAME)      # sommatoria di riga DF
colSums(DATAFRAME)      # sommatoria di colonna DF
colMeans(DATAFRAME)     # media di colonna DF
rowMeans(DATAFRAME)     # media di riga DF
fix(DATABASE)           # visualizzo tutto il DF
na.omit (DATAFRAME)     # rimuove righe con dati mancanti
```

FUNZIONI SPECIALI

MERGE combina due dataframes basandosi su corrispondenze

```
R<-merge(DF1,DF2,by.x="NOME_COL",by.y="NOME_COL",all=T)
```

SUBSET filtra un dataframe

```
subset(airquality,Ozone<20, select = c(Ozone, Temp))
```

APPLY applica una funzione alle righe o alle colonne di un DF

```
apply(DATAFRAME,1,mean) alle righe
apply(DATAFRAME,2,mean) alle colonne
```

LAPPLY applica una funzione alle agli elementi di una lista

```
lapply(list,1,fun)
```

TAPPLY e BY applica una funzione ad una variabile condizionatamente ad un indice

```
tapply(DATAFRAME$VETTORE,DATAFRAME$FATTORE,fun)
by((DATAFRAME$VETTORE,DATAFRAME$FATTORE,fun)
```

FUNZIONI SPECIALI

SPLIT possiamo dividere oggetti secondo un fattore specificato

```
list<-split(DF1,DF$FACTOR)
```

SUMMARY descrive l'intero dataframe

```
summary(DATAFRAME)
```

TABULATE produce un vettore con frequenze assolute di un fattore

```
tabulate(DATAFRAME)
table(DATAFRAME)
```

FIVENUM calcola min quartili, mediana e max di un vettore

```
fivenum(DATAFRAME)
```

UNIQUE seleziona elementi unici a partire da un vettore

```
unique (DATAFRAME$VETTORE)
DATAFRAME[unique(DATAFRAME$VETTORE),]
```

Creare funzioni personalizzate con R

Una delle cose che rendono R un sistema per l'analisi dei dati tanto potente e versatile è il suo essere a tutti gli effetti un linguaggio di programmazione.
Quando siamo davanti a un problema al quale gli sviluppatori di R non hanno pensato (capita più spesso di quanto si possa credere), possiamo vestire noi stessi i panni dello sviluppatore e creare delle nuove funzioni R.

Per creare una nuova funzione dobbiamo:

1. Creare un nuovo oggetto (*fun* ad esempio) che sarà il risultato delle operazioni eseguite tramite il comando **`function()`**.

2. Definire gli argomenti da passare in input alla funzione.

3. Definire il corpo della funzione, ovvero tutte quelle operazioni da svolgere a partire dagli argomenti in input.

4. Restituire un valore in output.

Creare funzioni personalizzate con R

```
Nome_della_funzione <- function(argomenti)
{
```

- serie di operazioni da fare con gli argomenti
- valore della funzione

```
                                    }
```

Per esempio, per generare n numeri casuali da una distribuzione normale standard, farne l'istogramma e calcolare la media, si può definire la nuova funzione:

```
normistmed <- function(quantinevuoi){
              temp <- rnorm(quantinevuoi)
              hist(temp)
              mean(temp)
              }
 normistmed(100)
```

Creare funzioni personalizzate con R


```
correlazione<- function(x,y) {
mx <- mean(x);
my <- mean(y);
vx <- sd(x);
vy <- sd(y);
g <- (sum((x-mx)*(y-
my))/length(x))/(vx*vy);
return(g)
}
```

Cicli for

Il ciclo for() permette di eseguire una determinata operazione, oppure una serie di istruzioni, per un numero prefissato di volte. Vi si accede con
for (variabile in vettore) { istruzioni }.

```
somma<-0
for (i in 1:10){

somma<-somma+i
}
somma
```

la "i" serve unicamente come puntatore(temporaneo) per passare # tutti gli oggetti,uno alla volta, specificati dopo l'"in"

```
num<-NULL #inizializzo un vettore con zero elementi (vuoto)
for (i in 1:10){ num<-c(num, i) }
num
```

```
num<-0 #inizializzo un vettore con zero elementi (vuoto)
for (i in 1:10){ num[i]<-i }
num
```

IMPORTARE I DATI IN R

Per importare dei dati su R è necessario come prima cosa analizzare il tipo di file che vogliamo importare.
Dobbiamo verificare in particolare tre elementi:

- l'estensione del file
- se la prima riga contiene i nomi delle variabili o no,
- cosa separa gli elementi l'uno dall'altro (virgole, punti e virgola, tabulazioni, eccetera)

Le funzioni base per importare dati su R sono in particolare tre, e possono essere utilizzate per importare dei dati in formati quali .csv, .tsv o .txt.

Si tratta delle funzioni
read.table(), read.csv() e read.delim().

Queste tre funzioni hanno una struttura molto simile, ed è composta dai seguenti argomenti:

IMPORTARE I DATI IN R

- **il file**: è il file che vogliamo importare su R e può essere identificato col semplice nome se si trova nella directory di lavoro che abbiamo scelto per quel particolare progetto, oppure con l'intero indirizzo se si trova su una cartella esterna

- **l'header**: l'argomento header serve a specificare se nella prima riga dei nostri dati sono presenti i nomi delle variabili del file o no. In caso affermativo, è settato su TRUE

- **l'argomento sep** indica il separatore che divide i dati. Può essere ad esempio una virgola, o un punto e virgola. Scriveremo quindi

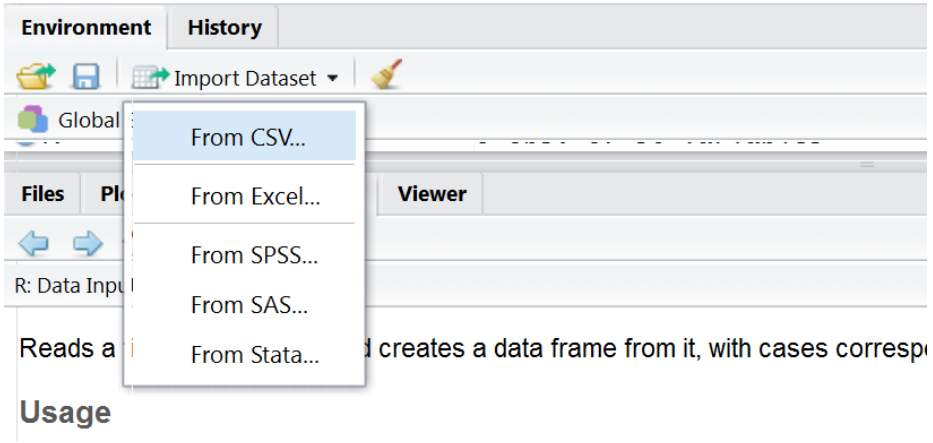
sep = "," oppure **sep = ";"** oppure **sep = "\t"** con il separatore tra virgolette.

IMPORTARE I DATI IN R

```
DATI<- read.table("database.txt", header=TRUE,
sep="\t", dec=".", na.string="NA", skip=0)
```

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

- **In Rstudio è possibile caricare i dati utilizzando interfaccia grafica**



ESPORTARE I DATI IN R

```
write.table(DATAFRAME,"database.txt",rownames=FALSE,
quote=FALSE, sep="\t")
```

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")
```

Alcuni dataset implementati in R

Package	Item	Title	Rows	Cols	csv	doc
datasets	AirPassengers	Monthly Airline Passenger Numbers 1949-1960	144	2	CSV	DOC
datasets	BJsales	Sales Data with Leading Indicator	150	2	CSV	DOC
datasets	BOD	Biochemical Oxygen Demand	6	2	CSV	DOC
datasets	CO2	Carbon Dioxide Uptake in Grass Plants	237	2	CSV	DOC
datasets	Formaldehyde	Determination of Formaldehyde	6	2	CSV	DOC
datasets	HairEyeColor	Hair and Eye Color of Statistics Students	4	4	CSV	DOC
datasets	InsectSprays	Effectiveness of Insect Sprays	72	2	CSV	DOC
datasets	JohnsonJohnson	Quarterly Earnings per Johnson & Johnson Share	84	2	CSV	DOC
datasets	LakeHuron	Level of Lake Huron 1875-1972	98	2	CSV	DOC
datasets	LifeCycleSavings	Intercountry Life-Cycle Savings Data	50	5	CSV	DOC
datasets	Nile	Flow of the River Nile	100	2	CSV	DOC
datasets	OrchardSprays	Potency of Orchard Sprays	64	4	CSV	DOC
datasets	PlantGrowth	Results from an Experiment on Plant Growth	30	2	CSV	DOC
datasets	Puromycin	Reaction Velocity of an Enzymatic Reaction	23	3	CSV	DOC
datasets	Titanic	Survival of passengers on the Titanic	1313	6	CSV	DOC
datasets	ToothGrowth	The Effect of Vitamin C on Tooth Growth in Guinea Pigs	60	3	CSV	DOC
datasets	UCBAdmissions	Student Admissions at UC Berkeley	2	2	CSV	DOC
datasets	UKDriverDeaths	Road Casualties in Great Britain 1969-84	192	2	CSV	DOC
datasets	UKgas	UK Quarterly Gas Consumption	108	2	CSV	DOC
datasets	USAccDeaths	Accidental Deaths in the US 1973-1978	72	2	CSV	DOC
datasets	USArrests	Violent Crime Rates by US State	50	4	CSV	DOC
datasets	USJudgeRatings	Lawyers' Ratings of State Judges in the US Superior Court	43	12	CSV	DOC
datasets	USPersonalExpenditure	Personal Expenditure Data	5	5	CSV	DOC
datasets	VADeaths	Death Rates in Virginia (1940)	5	4	CSV	DOC
datasets	WWWusage	Internet Usage per Minute	100	2	CSV	DOC
datasets	WorldPhones	The World's Telephones	7	7	CSV	DOC
datasets	airmiles	Passenger Miles on Commercial US Airlines, 1937-1960	24	2	CSV	DOC
datasets	airquality	New York Air Quality Measurements	153	6	CSV	DOC
datasets	anscombe	Anscombe's Quartet of 'Identical' Simple Linear Regressions	11	8	CSV	DOC
datasets	attenu	The Joyner-Boore Attenuation Data	182	5	CSV	DOC
datasets	attitude	The Chatterjee-Price Attitude Data	30	7	CSV	DOC
datasets	austres	Quarterly Time Series of the Number of Australian Residents	89	2	CSV	DOC

https://vincentarelbundock.github.io/Rdatasets/datasets.html

RICODIFICA DELLE VARIANTI

E' necessario attribuite e correggere il modo in cui si presentano le variabili una volta caricato un dataset.

In particolare attribuire valore categorico alle variabili categoriche, numerico alle numeriche.

Questo è possibile sovrascrivendo le variabili in questo modo:

```
DATAFRAME$VAR1<-as.factor(DATAFRAME$VAR1)
DATAFRAME$VAR2<-as.numeric(DATAFRAME$VAR2)
```

RICODIFICA DELLE VARIANTI

A volte si rende necessario ricodificare delle varianti.

RICODIFICO LA VARIABILE IN CATEGORIE

```
mtcars$mpg.cat <- ifelse(mtcars$mpg > 20,
c("UP"), c("DOWN"))
```

Altro esempio:

```
attach(mtcars)
mtcars$mpg.cat[mpg > 20] <- "UP"
mtcars$mpg.cat[mpg <= 20] <- "DOWN"
detach(mtcars)
```

RINOMINO VARIABILI

```
library(reshape)
mtcars <- rename(mtcars, c(mpg="PIPPO"))
```

I DATI MANCANTI

I dati mancanti possono essere un problema non tanto banale quando si analizza un set di dati e la loro contabilità non è in genere così semplice.

In R, i valori mancanti sono rappresentati dal simbolo NA (non disponibile). I valori impossibili (ad esempio, dividendo per zero) sono rappresentati dal simbolo NaN (non un numero). A differenza di SAS, R utilizza lo stesso simbolo per i dati personali e numerici.

TESTIAMO PER VALORI MANCANTI

```
is.na(mtcars)          # returns TRUE of x is missing
complete.cases(mtcars) # returns TRUE of complete observations
```

Si possono creare dataframe di dati completi o non completi

```
COMPLETI <- mtcars[complete.cases(mtcars),]
INCOMPLETI <- mtcars[!complete.cases(mtcars),]
COMPLETI<- na.omit(mtcars)
```

IMPUTAZIONE

```
library(Hmisc)
DF <- data.frame(age = c(10, 20, NA, 40), sex = c('male','female'))
DF<-data.frame(DF,DF,DF)
```

Imputazione usando la media

```
DATA_IMP<-impute(age, mean))
```

Imputazione usando la numeri casuali

```
DF$imputed_age2 <- with(DF, impute(age, 'random'))
```

Imputazione usando la mediana

```
with(DF, impute(age, median))
```

IMPUTAZIONE

DATI MANCANTI

CREO DATASET DATI MANCANTI DA AIRQUALITY

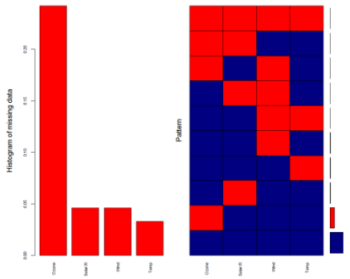
```
data <- airquality
data[4:10,3] <- rep(NA,7)
data[1:5,4] <- NA
data <- data[-c(5,6)]
summary(data)
```

```
library(mice)
md.pattern(data)
```

```
    Temp Solar.R Wind Ozone
104    1       1    1     1  0
 34    1       1    1     0  1
  4    1       0    1     1  1
  3    1       1    0     1  1
  3    0       1    1     1  1
  1    1       0    1     0  2
  1    1       1    0     0  2
  1    1       0    0     1  2
  1    0       1    0     1  2
  1    0       0    0     0  4
       5       7    7    37 56
```

104 campioni completi
34 campioni completi tranne ozono

```
library(VIM)
aggr_plot <- aggr(data, col=c('navyblue','red'), numbers=TRUE,
sortVars=TRUE, labels=names(data), cex.axis=.7, gap=3,
ylab=c("Histogram of missing data","Pattern"))
```



IMPUTAZIONE

```
library(mice)
md.pattern(data)

tempData <- mice(data,m=5,meth='pmm',seed=500)
summary(tempData)
```

CONSIDERAZIONI SUI PARAMETRI

m=5 sono I dataset imputati

meth='pmm' si riferisce al metodo di imputazione usato (media)

Altri metodi possono essere usati:

```
> methods(mice)
 [1] mice.impute.2l.norm      mice.impute.2l.pan       mice.impute.2lonly.mean
 [4] mice.impute.2lonly.norm  mice.impute.2lonly.pmm   mice.impute.cart
 [7] mice.impute.fastpmm      mice.impute.lda          mice.impute.logreg
[10] mice.impute.logreg.boot  mice.impute.mean         mice.impute.norm
[13] mice.impute.norm.boot    mice.impute.norm.nob     mice.impute.norm.predict
[16] mice.impute.passive      mice.impute.pmm          mice.impute.polr
[19] mice.impute.polyreg      mice.impute.quadratic    mice.impute.rf
[22] mice.impute.ri           mice.impute.sample       mice.mids
[25] mice.theme
```

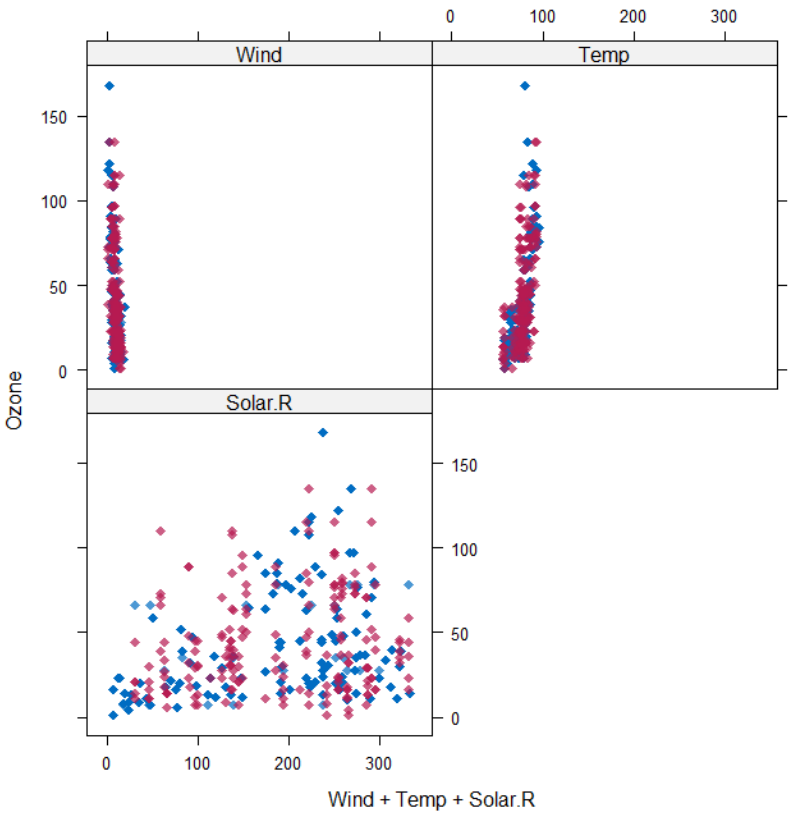
```
completedData <- complete(tempData,1)
```

si visualizza il primo dataset imputato

IMPUTAZIONE

Si può ispezionare la distribuzione dei dati veri e quelli imputati

```
xyplot(tempData,Ozone ~
Wind+Temp+Solar.R,pch=18,cex=1)
```

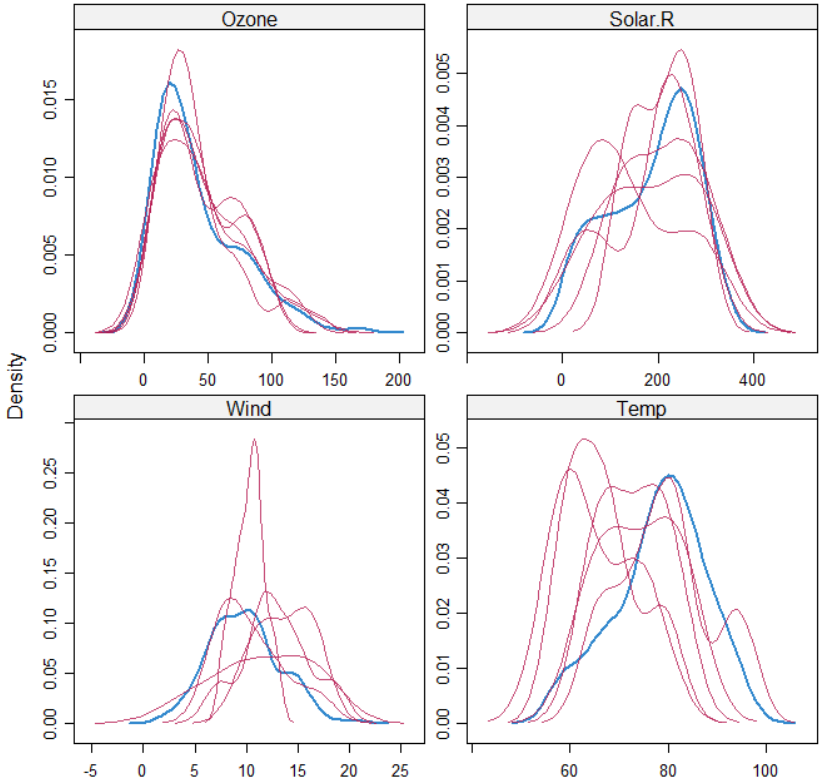


Ozono versus tutte le altre variabili in magenta i dati imputati

IMPUTAZIONE

Si analizza la distribuzione dei datasets imputati

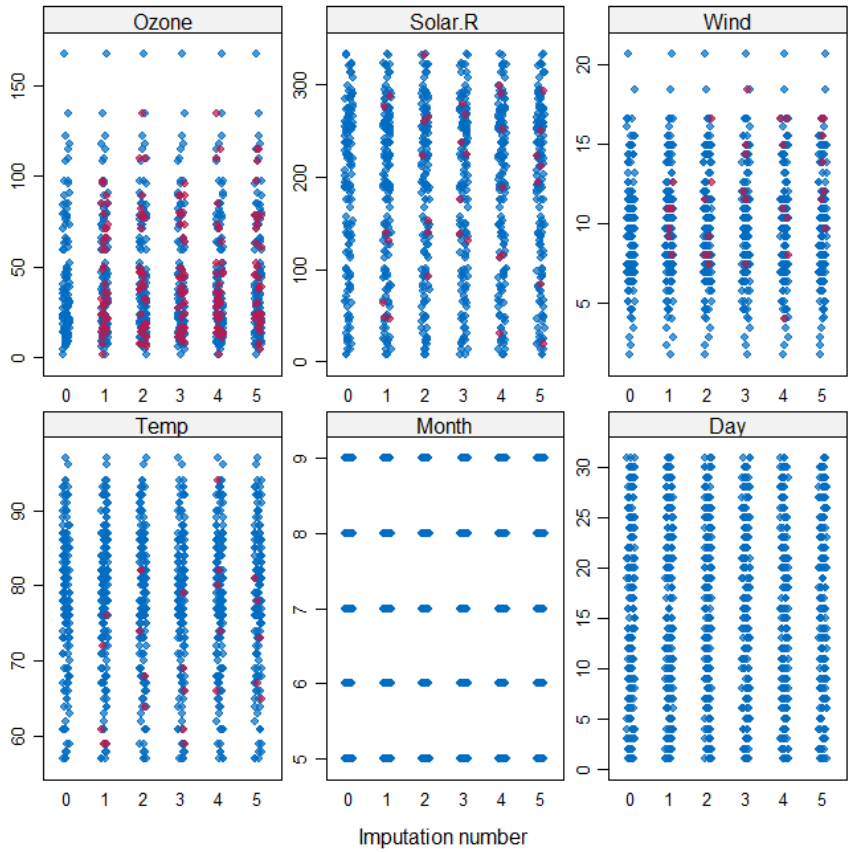
```
densityplot(tempData)
```



IMPUTAZIONE

Si analizza la distribuzione dei datasets imputati

```
stripplot(tempData, pch = 20, cex = 1.2)
```



IMPUTAZIONE

POOLING dei dataset imputati

```
modelFit1 <- with(tempData,lm(Temp~
Ozone+Solar.R+Wind))
summary(pool(modelFit1))
```

```
> summary(pool(modelFit1))
                  est          se         t        df     Pr(>|t|)        lo 95       hi 95
(Intercept) 71.944020696 2.847975542 25.261460 100.97912 0.000000e+00 66.294389152 77.59365224
Ozone        0.167893206 0.024225870  6.930327 106.10870 3.384755e-10  0.119863630  0.21592278
Solar.R      0.009922031 0.007260571  1.366563  81.39080 1.755255e-01 -0.004523177  0.02436724
Wind        -0.289099358 0.216968881 -1.332446  96.51328 1.858511e-01 -0.719749954  0.14155124
             nmis       fmi     lambda
(Intercept)    NA 0.1147021 0.09733977
Ozone          37 0.1053021 0.08859576
Solar.R         7 0.1538052 0.13326424
Wind            7 0.1230880 0.10510248
> |
```

ESERCIZIO N 1 https://urly.it/38gx	DATASET_1 https://urly.it/38g_
ESERCIZIO N 2 https://urly.it/38gy	DATASET_ESERCIZIO_1 https://urly.it/38ga
ESERCIZIO N 2 https://urly.it/38gz	DATASET_TIROIDE https://urly.it/38h0
SOLUZIONI https://urly.it/38g-	DATASET_CENTENARI https://urly.it/38h1