

Note sul linguaggio ed ambiente statistico

Dr. Luca Scrucca
Dipartimento di Economia, Finanza e Statistica
Sezione di Statistica
Università degli Studi di Perugia
luca@stat.unipg.it

Questo documento fornisce un'introduzione a R (www.r-project.org), un linguaggio ed un ambiente per il calcolo statistico e la rappresentazione grafica. R è un'implementazione Open Source e distribuito sotto licenza GPL del linguaggio S sviluppato presso i Bell Laboratories della AT&T da John Chambers et al.

R fornisce un'ampia gamma di metodi statistici di base: statistiche descrittive e grafici esplorativi, modelli lineari, lineari generalizzati e modelli non lineari, inferenza statistica parametrica e nonparametrica, analisi di serie storiche, metodi di analisi multivariata, ecc. Accanto a questi, ulteriori metodi sviluppati nei più diversi ambiti applicativi e teorici sono messi continuamente a disposizione dalla comunità di sviluppatori-ricercatori-utenti di R (<http://cran.r-project.org/src/contrib/PACKAGES.html>). Inoltre, data la sua natura di linguaggio di programmazione, nuove tecniche e metodi possono essere implementati in R da ogni utente in base alle proprie esigenze.

Infine, R è disponibile per i principali sistemi operativi, quali Unix/Linux, MS Windows ed Apple OSX.

Indice

1	Introduzione	2
2	R documentazione e help in linea	3
3	Oggetti e tipologie di dati	3
4	Vettori e matrici	4
5	Funzioni per l'input/output di base	8
6	Distribuzioni statistiche	8
6.1	Esempi	9
7	Importare ed esportare dati	10
7.1	Alcune note sull'utilizzo e la costruzione delle path per file esterni	12
8	Funzioni	13
8.1	Esempi	14
9	Analisi univariate	16
9.1	Statistiche di sintesi	16
9.2	Rappresentazioni grafiche	17
10	Tabelle di contigenza	20
11	Modelli di regressione lineare	22
11.1	Regressione semplice	22
11.2	Regressione multipla	26
11.3	ANOVA ad un fattore	29
11.4	ANOVA a due fattori	31
11.5	ANCOVA	34

1 Introduzione

```
> R # inizia sessione R

R : Copyright 2003, The R Development Core Team
Version 1.7.0 (2003-04-16)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

> # prompt dei comandi
> x <- c(0.77, 1.74, 0.81, 1.20, 1.95, 1.20, 0.47, 1.43, 3.37, 2.20, # si inseriscono i dati
+       3.00, 3.09, 1.51, 2.10, 0.52, 1.62, 1.31, 0.32, 0.59, 0.81, # assegnandoli alla
+       2.81, 1.87, 1.18, 1.35, 4.75, 2.48, 0.96, 1.89, 0.90, 2.05) # variabile x
> x
[1] 0.77 1.74 0.81 1.20 1.95 1.20 0.47 1.43 3.37 2.20 3.00 3.09 1.51 2.10 0.52
[16] 1.62 1.31 0.32 0.59 0.81 2.81 1.87 1.18 1.35 4.75 2.48 0.96 1.89 0.90 2.05
> mean(x)
[1] 1.675
> var(x)
[1] 1.001233
> sqrt(var(x))
[1] 1.000616
> summary(x)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
  0.32  0.915  1.47 1.675  2.088 4.75
> median(x)
[1] 1.47
> range(x)
[1] 0.32 4.75
> quantile(x,0.25)
 25%
 0.915
> par(mfrow=c(2,2)) # suddivide la finestra grafica in una matrice 2x2
> hist(x) # disegna un istogramma
> hist(x,8,probability=TRUE) # disegna un istogramma con numero di bins = 8 e
# su una scala di probabilità
> boxplot(x) # disegna un boxplot
> qqnorm(x) # disegna un Q-Q plot per controllare normalità
> qqline(x) # e vi aggiunge una linea di riferimento
> par(mfrow=c(1,1)) # ripristina il numero di plots in un grafico
> q() # termina sessione
Save workspace image? [y/n/c]
```

Al termine della sessione viene richiesto se si desidera salvare su file il *workspace*, cioè lo stato attuale dell'ambiente con tutti gli oggetti (funzioni, variabili, ecc.) definiti. Qualora venga salvato, al successivo riavvio di R il workspace così salvato verrà ripristinato.

2 R documentazione e help in linea

Ogni installazione di R contiene ottima ed abbondante documentazione. I seguenti manuali vengono forniti in formato pdf e html:

- An Introduction to R (manuale ufficiale, 100 pagg. circa)
- Writing R Extensions
- The R Language Definition
- R Data Import/Export
- R Installation and Administration

Inoltre, è disponibile un help in linea consultabile con la funzione `help()`, mentre `help.search()` consente di effettuare ricerche per parole chiavi nella documentazione in linea. Ad esempio:

```
> help(mean) # richiama help per la funzione mean
> help.search("regression") # cerca la stringa "regression" tra la documentazione disponibile
> apropos("glm") # restituisce tutte le funzioni che contengono glm nel nome
```

La documentazione in linea è anche disponibile in formato html, consultabile attraverso un browser già installato. Per richiamare l'help in formato html occorre digitare:

```
> help.start()
```

Con questo comando si aprirà un browser web, da cui è possibile accedere ai manuali sopra menzionati in formato html, ed alla sezione **Search Engine & Keywords**. Quest'ultima consente di consultare la documentazione organizzata per keywords oppure per campi definiti di ricerca ¹.

3 Oggetti e tipologie di dati

In R ogni elemento è un oggetto ed ogni oggetto appartiene ad una classe. Le principali tipologie di oggetti sono:

- **character**: stringhe o sequenze di caratteri
> a <- c("tizio", "caio", "sempronio")
- **numeric**: numeri reali, che in R sono espressi in precisione doppia
> b <- c(1.5, 3, 4, 8, 10)
- **integer**: numeri interi, eventualmente con segno
> c <- 1:3
- **logical**: sono elementi che assumono solo i valori TRUE (vero) o FALSE (falso) e NA (not available)
> d <- c(TRUE, FALSE, TRUE)
- **complex**: i numeri complessi (vedi `help(complex)`)

Ciascun oggetto può essere strutturato in *vettori* (come si è fatto sopra), in *scalari* (vettori di un solo elemento), *matrici* a due dimensioni, oppure *array* *k*-dimensionali. Esiste poi anche la struttura della *lista*, la quale può contenere al suo interno diverse tipologie di dati. Ad es.

¹N.B.: è richiesto un browser con Java installato

```

> l <- list(stringhe=a, numeri=b, interi=c, d)
> l
$stringhe
[1] "tizio"      "caio"      "sempronio"

$numeri
[1] 1.5 3.0 4.0 8.0 10.0

$interi
[1] 1 2 3

[[4]]
[1] TRUE FALSE TRUE

```

Si può accedere a ciascun elemento della lista come segue:

```

> l$interi
[1] 1 2 3
> l[[3]]
[1] 1 2 3
> l[3]
 $interi
[1] 1 2 3

```

Per visualizzare in maniera compatta la struttura di un oggetto in R è utile il comando `str()`:

```

> str(l)
List of 4
 $ stringhe: chr [1:3] "tizio" "caio" "sempronio"
 $ numeri  : num [1:5] 1.5 3 4 8 10
 $ interi  : int [1:3] 1 2 3
 $         : logi [1:3] TRUE FALSE TRUE

```

4 Vettori e matrici

La più semplice struttura di dati disponibile in R sono i vettori, includendo tra questi anche gli scalari che sono trattati come vettori di lunghezza unitaria. Quindi:

```

> a <- 2 # scalare
> a
[1] 2
> class(a)
[1] "numeric"
> str(a)
 num 2

> b <- c(1,3,5,2) # vettore
> b
[1] 1 3 5 2
> class(b)
[1] "numeric"
> str(b)
 num [1:4] 1 3 5 2

```

Il comando `c()` consente di concatenare gli elementi che seguono, ciascuno separato da una `,`. Ogni elemento, o sottoinsieme di elementi, di un vettore può essere selezionato indicando tra parentesi `[...]` la posizione che occupa:

```

> b[3]
[1] 5

```

```

> b[c(1,3)]
[1] 1 5
> b[-2]
[1] 1 5 2
> b[1] <- 100
> b
[1] 100 3 5 2

```

Sui vettori è possibile effettuare operazioni matematiche, come ad es.:

```

> b-1
[1] 99 2 4 1
> b*2
[1] 200 6 10 4
> log(b)
[1] 4.6051702 1.0986123 1.6094379 0.6931472
> c <- c(2,3,1,2)
> b*c # prodotto elemento per elemento
[1] 200 9 5 4
> b %*% c # prodotto scalare  $b^T c$  (inner product)
[1,] 218
> b %*% t(c) #  $bc^T$  (outer product)
[1,] [2,] [3,] [4,]
[1,] 200 300 100 200
[2,] 6 9 3 6
[3,] 10 15 5 10
[4,] 4 6 2 4

```

L'ultima operazione crea una matrice, ovvero una serie di vettori concatenati per riga o per colonna. Per creare direttamente una matrice si può usare la funzione `matrix()`:

```

> x <- matrix(1:20, nrow=4, ncol=5, byrow=TRUE) # genera una matrice 4x5
> x
[1,] [2,] [3,] [4,] [5,]
[1,] 1 5 9 13 17
[2,] 2 6 10 14 18
[3,] 3 7 11 15 19
[4,] 4 8 12 16 20
> class(x)
[1] "matrix"
> str(x)
int [1:4, 1:5] 1 6 11 16 2 7 12 17 3 8 ...
> x[1,] # seleziona la prima riga
[1] 1 5 9 13 17
> x[,4] # seleziona la quarta colonna
[1] 13 14 15 16
> x[4,5] # seleziona l'elemento nell'ultima cella in basso a destra
[1] 20
> dim(x) # dimensione della matrice x
[1] 4 5
> y <- cbind(x[,1],x[,5]) # genera una matrice 4x2 selezionando le colonne 1 e 5 di x
> y # e concatenandole per colonna; l'analogo comando rbind()
# può essere utilizzato per concatenare le righe
[1,] [2,]
[1,] 1 17
[2,] 2 18
[3,] 3 19
[4,] 4 20

```

Alcune delle operazioni tra matrici disponibili:

```

> y + cbind(x[,2],x[,3])
  [,1] [,2]
[1,]   6  26
[2,]   8  28
[3,]  10  30
[4,]  12  32
> y - cbind(x[,2],x[,3])
  [,1] [,2]
[1,]  -4   8
[2,]  -4   8
[3,]  -4   8
[4,]  -4   8
> y * cbind(x[,2],x[,3])
  [,1] [,2]
[1,]   5 153
[2,]  12 180
[3,]  21 209
[4,]  32 240
> y / cbind(x[,2],x[,3])
  [,1] [,2]
[1,] 0.2000000 1.888889
[2,] 0.3333333 1.800000
[3,] 0.4285714 1.727273
[4,] 0.5000000 1.666667

```

Tutte le precedenti operazioni si sono applicate a matrici della stessa dimensione, in quanto si trattava di operazioni elemento per elemento. Il prodotto matriciale, invece, richiede che il numero delle colonne della prima matrice sia uguale al numero delle righe della seconda matrice, cioè se $\dim(A)=(r \ c)$ e $\dim(B)=(c \ s)$, allora $\dim(A\%*%B)=(r \ s)$, dove con $\%*\%$ si indica in R il prodotto matriciale. Ad es.:

```

> dim(y)
[1] 4 2
> t(y) # la funzione t() traspone y
  [,1] [,2] [,3] [,4]
[1,]   1   6  11  16
[2,]   5  10  15  20
> t(y) %*% cbind(x[,2],x[,3]) # prodotto matriciale
  [,1] [,2]
[1,]  70  110
[2,] 486  782

```

In R esiste una vasta libreria di funzioni di Algebra Lineare che si applicano a matrici o a vettori, come ad esempio:

- ✓ `diag` estrae o sostituisce gli elementi diagonali di una matrice, oppure costruisce una matrice diagonale;
- ✓ `det` calcola il determinante di una matrice;
- ✓ `solve` risolve un sistema lineare di equazioni (usata anche per invertire una matrice, vedi `help in linea`);

```

> X <- matrix(c(0.69, -0.04, 1.27, 0.52, -0.04, 0.19, -0.33, -0.12, 1.27,
               -0.33, 3.12, 1.30, 0.52, -0.12, 1.30, 0.58),
              nrow=4, ncol=4, byrow=TRUE)
> X
  [,1] [,2] [,3] [,4]
[1,] 0.69 -0.04 1.27 0.52
[2,] -0.04 0.19 -0.33 -0.12

```

```

[3,] 1.27 -0.33 3.12 1.30
[4,] 0.52 -0.12 1.30 0.58
> diag(X)
[1] 0.69 0.19 3.12 0.58
> diag(4)
      [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
> diag(rep(2,3))
      [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    2    0
[3,]    0    0    2
> det(X)
[1] 0.00184532
> Xinv <- solve(X)
      [,1]      [,2]      [,3]      [,4]
[1,] 9.534390 -6.384800 -6.720786 5.194763
[2,] -6.384800 11.073418 6.512692 -6.582056
[3,] -6.720786 6.512692 10.182516 -15.449895
[4,] 5.194763 -6.582056 -15.449895 30.334034
> Xinv %*% X
      [,1]      [,2]      [,3]      [,4]
[1,] 1.000000e+00 -2.566849e-16 4.316426e-15 1.977368e-15
[2,] -8.630249e-16 1.000000e+00 -2.157129e-15 -1.388863e-15
[3,] -3.685420e-15 4.110210e-16 1.000000e-00 -1.478852e-15
[4,] 2.101617e-15 -8.068633e-16 5.256212e-15 1.000000e+00
> round(Xinv %*% X, 4)
      [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1

```

- ✓ qr decomposizione QR di una matrice;
- ✓ svd decomposizione in valori singolari di una matrice;
- ✓ eigen decomposizione spettrale (autovalori e autovettori) di una matrice;
- ✓ chol decomposizione di Choleski;
- ✓ ginv inversa generalizzata di una matrice.

Per un quadro più ampio delle funzioni disponibili si veda l'help in linea relativo alle sezioni *Mathematics/Basic Arithmetic and Sorting*, *Mathematics/Matrices and Arrays* e *Mathematics/Linear Algebra*.

Molto utili sono anche le funzioni per l'ottimizzazione (vedi la sezione *Mathematics/Optimization* dell'help in linea) tra cui:

- ✓ `optimize` ottimizzazione unidimensionale;
- ✓ `nlm` minimizzazione di una funzione non lineare con un algoritmo di tipo Newton;
- ✓ `simplex{boot}` metodo del simplesso per problemi di programmazione lineare, disponibile nel package `boot`.

5 Funzioni per l'input/output di base

```
> write(x, file="output.dat", ncolumns=1)      # scrive sul file ASCII output.dat il contenuto dell'oggetto x
                                                # di x , formattato su una sola colonna
> source("comandi.R")                        # legge il file comandi.R ed esegue i comandi
                                                # in esso contenuti
> sink("output.txt")                          # reindirizza i risultati, che di solito appaiono a video,
...                                           # verso il file output.txt
...
> sink()                                      # chiude il precedente file di output, ripristinando
                                                # l'output a video
> history(50)                                # mostra gli ultimi 50 (il default è 25) comandi
                                                # digitati alla console
> savehistory()                               # salva tutti i comandi della sessione di lavoro sul file
                                                # .Rhistory (default)

> ls()                                        # elenca gli oggetti contenuti nel workspace corrente

> save.image()                                # salva il workspace corrente nel file ".RData" (de-
fault)
                                                # della directory corrente; uguale a q("yes")'
> save(x, y, file = "xy.Rdata")              # salva solo gli oggetti x e y nel file indicato
> load("xy.Rdata")                           # legge il file indicato e ripristina gli oggetti
                                                # precedentemente salvati
```

6 Distribuzioni statistiche

Per molte delle distribuzioni statistiche usate comunemente in R è possibile calcolare la funzione di densità, o di probabilità nel caso discreto, la funzione di ripartizione o di probabilità cumulata, i quantili, e generare campioni di una data ampiezza. Tra le distribuzioni disponibili: Beta, Binomial, Cauchy, Chi-Square, Exponential, F, Gamma, Geometric, Hypergeometric, Logistic, Lognormal, Negative Binomial, Normal (Gaussian), Poisson, Student's t, Uniform, Weibull.

Vediamo ora alcuni esempi che consentono di illustrare la sintassi dei comandi. Per le altre distribuzioni non trattate si veda la sezione Probability Distributions and Random Numbers nel help in linea.

✓ Normale

```
dnorm(x, mean=0, sd=1)  calcola densità in x
pnorm(q, mean=0, sd=1)  calcola probabilità cumulata fino a q
qnorm(p, mean=0, sd=1)  calcola quantile corrispondente alla probabilità p
rnorm(n, mean=0, sd=1)  genera un campione di dimensione n
```

Se i parametri opzionali non sono impostati si ottiene una distribuzione normale standardizzata, altrimenti `mean` corrisponde alla media e `sd` alla deviazione standard.

✓ Chi-quadrato

```
dchisq(x, df)  calcola densità in x
pchisq(q, df)  calcola probabilità cumulata fino a q
qchisq(p, df)  calcola quantile corrispondente alla probabilità p
rchisq(n, df)  genera un campione di dimensione n
```

Il parametro `df` corrisponde ai gradi di libertà della variabile chi-quadrato.

✓ Binomiale

```
dbinom(x, size, prob)  calcola densità in x
pbinom(q, size, prob)  calcola probabilità cumulata fino a q
qbinom(p, size, prob)  calcola quantile corrispondente alla probabilità p
rbinom(n, size, prob)  genera un campione di dimensione n
```

I parametri `size` e `prob` si riferiscono al numero di prove e alla probabilità di successo di ciascuna prova. La variabile casuale di Bernoulli è ottenuta come caso particolare specificando `size=1`.

6.1 Esempi

✓ **v.c. Normale Standardizzata**

Supponiamo si voglia rappresentare graficamente la densità di una v.c. normale standardizzata:

```
> x <- sort(rnorm(1000))          # si generano 1000 num. casuali da una normale standardizzata
                                   # e si ordinano in senso non decrescente
> f <- dnorm(x)                   # si calcolano le corrispondenti densità
> plot(x, f, type="l")
> rug(x)                           # aggiunge delle linee verticali in corrispondenza dei valori osservati
```

Si provi a ripetere l'esperimento cambiando la dimensione del campione. Si generino anche osservazioni da una distribuzione normale non standardizzata (suggerimento: vedi `help(rnorm)`).

✓ **approssimazione di una v.c. Chi-quadrato con una Normale**

Ricordando che se $y \sim \chi_k^2$ allora $E(y) = k$ e $\text{Var}(y) = 2k$, possiamo rappresentare graficamente una v.c. χ^2 e, contemporaneamente, una v.c. normale con stessa media e varianza:

```
> y <- sort(rchisq(1000,10))      # genera 1000 num.casuali da una v.c. chi-quadrato
                                   # con 10 gradi di libertà, e li ordina
> plot(y,dchisq(y,10), type="l")  # disegna la funzione di densità
> x <- seq(10-4*sqrt(20),10+4*sqrt(20),by=1) # crea una sequenza di valori tra  $10 \pm 4\sigma$  con passo 1
> lines(x,dnorm(x,10,sqrt(20)),col=2,lty=2) # aggiunge la funzione di densità per una normale di
                                   # media 10 e varianza 20
```

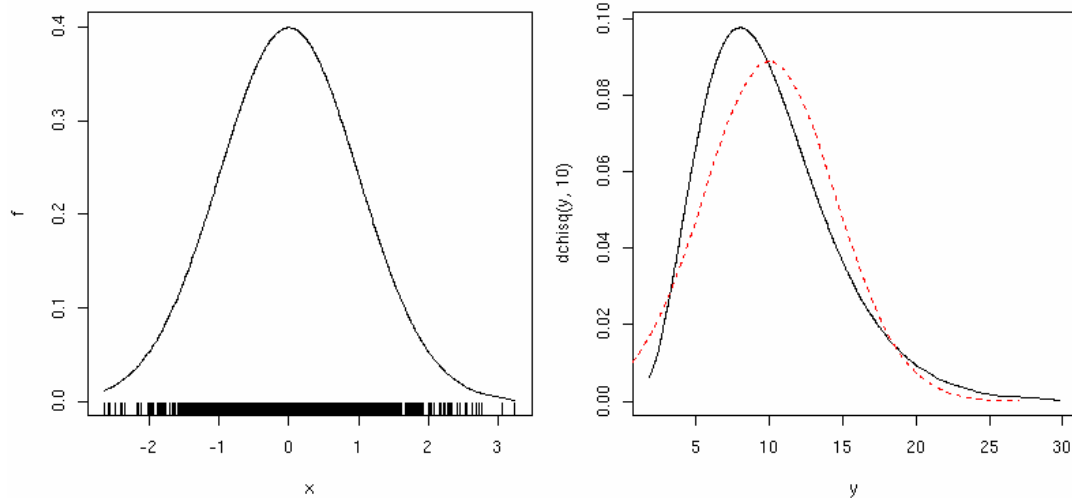
Si riprovi l'esperimento cambiando i gradi di libertà. In particolare, cosa succede se i gradi di libertà sono 100, 500 oppure 1000?

✓ **Confronto tra distribuzione teorica ed empirica**

Si supponga ora di voler comparare la distribuzione osservata di una v.c. binomiale, con la corrispondente distribuzione teorica:

```
> x <- rbinom(100,10,0.7)          # si generano 100 osservazioni da una binomiale(10,0.7)
> table(x)/length(x)             # tabula la distribuzione di frequenza
   3   4   5   6   7   8   9  10
0.01 0.02 0.08 0.25 0.27 0.23 0.12 0.02
> x1 <- seq(0, max(x)+2, by=1)     # si crea un vettore di valori interi 0,1,2,...
> hist(x, breaks=unique(x), xlim=range(x1), # si rappresenta l'istogramma in scala di probabilità
      probability=TRUE)
> lines(x1, dbinom(x1,10,0.7),     # quindi si compara con la probabilità teorica
      type="h", col=2, lwd=4)     # di ciascun evento
```

Provare a ripetere l'esperimento più volte. Cosa succede? E se si generano 1000 osservazioni e non 100? E se la probabilità di successo è pari a 0.5 oppure 0.1?



7 Importare ed esportare dati

Abbiamo già visto che la funzione `c()` consente di assegnare un insieme di valori ad una variabile. Si consideri ad esempio i valori giornalieri di chiusura di un portafoglio azionario: 45, 43, 46, 48, 51, 46, 50, 47, 46, 45.

```
> x <- c(45, 43, 46, 48, 51, 46, 50, 47, 46, 45)
```

Altri metodi esistono per inserire i dati, quali

```
> x <- scan()                # si inseriscono i dati separati da uno spazio
1: 45 43 46 48 51 46 50 47 46 45 # e l'immissione termina con doppio invio
11:
Read 10 items
```

oppure

```
> x <- NA                    # la variabile deve essere prima definita, quindi
> data.entry(x)              # viene aperta una finestra contenente un semplice foglio elettronico
```

Quando la mole di dati è sostanziosa l'immissione da tastiera non è agevole. In tali casi è usuale avere a disposizione i dati su file separati che dovranno essere letti e resi disponibili ad R.

Per leggere dati da file R fornisce numerose funzioni, tra le quali `read.table()` e `read.csv()`. La funzione

```
read.table(file, header = FALSE, sep = "", dec = ".", ...)
```

legge il contenuto di un file dati in formato ASCII, dove ogni colonna si riferisce ad una variabile ed ogni riga ad un'unità statistica. Restituisce un oggetto di class `data.frame`, ovvero un oggetto di R con caratteristiche simili ad una matrice (vedi `help(data.frame)`). Argomento obbligatorio è `file`, una stringa che fornisce il nome del file da leggere (comprensivo della path se il file è presente in una directory diversa da quella corrente). I diversi campi possono essere separati dal carattere definito in `sep`, che per default è impostato ad almeno uno spazio o tabulazione. L'argomento opzionale `header` se specificato uguale a `TRUE` consente di inserire una prima riga con le labels per le variabili in colonna.

Esempio: inflazione USA, 1948-1989 Il file di dati `inflation.txt` contiene una serie di dati annuali (fonte. Johnston J. and DiNardo J. (1997), *Econometric Methods*, fourth ed., McGraw-Hill, New York). Le prime righe del file `inflation.txt` sono le seguenti:

```

CPI INF UNR UNR1 Year
24.1 8.07175 3.8 NA 1948
23.8 -1.24482 5.9 3.8 1949
24.1 1.26051 5.3 5.9 1950
...
124 4.81826 5.3 5.5 1989

```

Ogni riga di questo file si riferisce agli anni dal 1948 al 1989, mentre in ogni colonna compaiono le variabili. Quindi, i dati sono disposti in una matrice ($n \times p$), dove $n = 42$ è il numero delle osservazioni, mentre $p = 5$ è il numero delle variabili. La prima riga viene utilizzata come “header” e contiene i nomi delle variabili:

```

CPI = Consumer price index
INF = Inflation rate derived as: (CPIt-1 - CPIt-1)/CPIt-1*100
UNR = Unemployment rate for civilian workers, aged 16 and over
UNR1 = One-year lag unemployment rate
Year = year

```

La lettura in R di questo file avviene come segue:

```

> data = read.table("inflation.txt", header=TRUE)           # legge i dati dal file inflation.txt
> str(data)                                                # e li assegna al data frame data
'data.frame':  42 obs. of  5 variables:
 $ CPI : num  24.1 23.8 24.1 26 26.5 26.7 26.9 26.8 27.2 28.1 ...
 $ INF : num   8.07 -1.24  1.26  7.88  1.92 ...
 $ UNR : num   3.8 5.9 5.3 3.3 3 2.9 5.5 4.4 4.1 4.3 ...
 $ UNR1: num   NA 3.8 5.9 5.3 3.3 3 2.9 5.5 4.4 4.1 ...
 $ Year: int  1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 ...
> attach(data)                                           # crea una variabile per ogni vettore (colonna) del
                                                         # data frame data
> CPI
 [1] 24.1  23.8  24.1  26.0  26.5  26.7  26.9  26.8  27.2  28.1  28.9  29.1
[13] 29.6  29.9  30.2  30.6  31.0  31.5  32.4  33.4  34.8  36.7  38.8  40.5
[25] 41.8  44.4  49.3  53.8  56.9  60.6  65.2  72.6  82.4  90.9  96.5  99.6
[37] 103.9 107.6 109.6 113.6 118.3 124.0
> detach(data)                                           # rimuove le variabili create dal corrispondente
                                                         # comando attach

```

Una seconda funzione che può essere utilizzata per leggere i dati è

```
read.csv(file, header = TRUE, sep = ",", dec=".", ...)
```

la quale legge file dati in formato `csv`. Tale formato prevede che ogni colonna sia separata da un separatore (default `sep = ","`) e che la prima riga contenga il nome delle variabili. Il separatore dei decimali è `dec="."` per default. Il formato `csv` si può ottenere facilmente da tutti i fogli elettronici, compreso MS Excel.

I valori di default vanno bene per il formato `csv` “classico”. Tuttavia la versione in lingua italiana di MS Excel salva i file `csv` in maniera leggermente diversa. Ovvero, utilizza come separatore decimale la virgola e come separatore di colonne il punto e virgola. Perciò dovremo cambiare i default degli argomenti `sep = ";"`, `dec=","`, oppure utilizzare la funzione già predisposta `read.csv2` (vedi help in linea per maggiori dettagli).

Esempio: inflazione USA, 1948-1989 (continua) Riprendendo l’esempio precedente, il file dati `inflation.csv` può essere importato in R come segue:

```
> data = read.csv("inflation.csv")
```

Per esportare un data frame o una matrice su di un file in R è disponibile la funzione

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

Infine, la libreria `foreign` contiene tutta una serie di funzioni utili per importare file da software statistici quali Minitab, S, SAS, SPSS, Stata.

```
> library(foreign)
> library(help="foreign")
```

7.1 Alcune note sull'utilizzo e la costruzione delle path per file esterni

In R ogni operazione su file avviene a partire dalla directory corrente di lavoro, la quale è visibile con il comando:

```
> getwd()
[1] "C:\\Programmi\\GnuR\\rw2001"
```

Se si vuol leggere un file dati, ad esempio `dati.txt` con il comando `read.table("dati.txt")`, l'operazione sarà eseguita correttamente solo se il file si trova nella directory corrente. Se, invece, si trova in un'altra directory (o su supporto diverso, come un Floppy o CD-ROM) si dovrà indicare il percorso completo da seguire (*path*).

Quindi, se si deve leggere o scrivere un file che si trova su una directory diversa da quella corrente o di lavoro, si può agire in due modi:

- 1) cambiare la directory corrente di lavoro con il comando `setwd(<directory>)`;
- 2) specificare espressamente nel comando di lettura o scrittura la path completa del file.

Nei sistemi Linux/Unix le path sono specificate con il simbolo / (slash), mentre nei sistemi MS Windows con il simbolo \ (backslash). In questi ultimi, se il file `dati.txt` è contenuto nella directory `c:\Documenti\Stat`, la path da usare in R è `c:\\Documenti\\Stat` oppure `c:/Documenti/Stat`. Il comando per leggere il file sarà pertanto:

```
> read.table("c:\\Documenti\\Stat\\dati.txt")
oppure
> read.table("c:/Documenti/Stat/dati.txt")
```

Da notare che sotto MS Windows i nomi dei file e dei percorsi (path) possono essere scritti indifferentemente in maiuscolo o minuscolo (ma questo non vale nei sistemi Linux/Unix), mentre se la path contiene uno o più spazi questi devono essere sempre preceduti dal simbolo \. Per esempio `c:\Miei Documenti\Stat` diventa `"c:\\Miei\ Documenti\\Stat"`.

Nelle ultime versioni di R per MS Windows c'è un'ulteriore possibilità: la funzione `choose.files()` consente di selezionare agevolmente un file utilizzando la nota finestra di selezione (GUI) di MS Windows. Quindi, il seguente comando:

```
> read.table(choose.files(), header=TRUE)
```

può essere utilizzato per leggere un file selezionandolo dalla finestra di dialogo che viene proposta.

8 Funzioni

Lo schema generale per la definizione di una funzione è il seguente:

```
<nome_funzione> <- function(<argomenti>)  
{  
  <corpo_della_funzione>  
  <valore_restituito_dalla_funzione>  
}
```

Il `<nome_funzione>` può essere un qualsiasi nome, contenente caratteri alfanumerici, compreso il “.”. Se già esiste una funzione con lo stesso nome, questa ultima verrà sostituita dalla nuova funzione definita.

Gli `<argomenti>` che vengono forniti alla funzione possono essere di vario tipo. Ad es., `function(x)` definisce come singolo argomento obbligatorio `x`. Se si vuole renderlo opzionale si deve assegnare un valore all'argomento, ad es. `function(x=0)`. Una funzione può avere uno o più argomenti, alcuni obbligatori altri opzionali. A questo livello non viene effettuato alcun controllo di congruenza. Ad esempio, se una funzione calcola la somma di due valori ed è definita come

```
mysum <- function(a=0, b) a+b
```

ci si attende che sia `a` che `b` siano valori numerici. Tuttavia, se richiamata come `mysum(1,"pippo")` andrà in errore nel momento di effettuare la somma. Mentre `mysum(1,2)` fornirà la risposta corretta, cioè 3. Essendo l'argomento `a` opzionale, potremmo voler richiamare la funzione specificando solo il secondo argomento `mysum(b=2)` che ritornerà il valore 2. Infine, una funzione può anche non contenere argomenti, o prevedere per ciascuno un valore di default così che, in entrambi i casi, può essere richiamata semplicemente come `<nome_funzione>()`.

Il `<corpo_della_funzione>` contiene tutte le espressioni (comandi) in R che definiscono ciò che la funzione effettivamente esegue. Gli argomenti e le variabili che sono create all'interno della funzione sono **variabili locali**, nel senso che al termine della funzione il loro valore non viene mantenuto. Nell'esecuzione dei comandi all'interno della funzione le variabili locali hanno precedenza sulle **variabili globali** definite a livello di workspace, mentre tutte le variabili non definite localmente devono essere definite a livello di workspace altrimenti si produrrà un errore.

```
myfun <- function(a=0)                                # definisco una nuova funzione con un solo argomento a  
{                                                    # mentre b è una variabile utilizzata nel corpo della funzione  
  if (a==0)  
    { a+b }  
  else  
    { b = a * 10  
      a+b }  
}  
> myfun()                                            # b non è definita nè localmente nè globalmente  
Error in myfun() : Object "b" not found  
> myfun(1)                                          # b è definita localmente  
[1] 11  
> b <- 2                                            # b è ora definita globalmente  
> myfun()                                          # ora la funzione non va in errore, ma utilizza il  
[1] 2                                              # valore di b definito globalmente  
> myfun(1)                                          # il risultato è lo stesso di prima perché variabili definite  
[1] 11                                              # localmente hanno la priorità su quelle globali
```

In realtà le cose possono essere anche più complesse, in quanto si possono definire degli *environment* all'interno dei quali la funzione può essere valutata. Per default tutte le funzioni vengono valutate nell'environment principale che è il workspace.

Il `<valore_restituito_dalla_funzione>` è il risultato dell'ultima espressione valutata. È possibile specificare esplicitamente cosa una funzione debba restituire attraverso il comando `return`: ad es. `return(x)` restituisce il contenuto dell'oggetto `x`. Per default una funzione restituisce il risultato dell'ultima espressione valutata.

8.1 Esempi

Media ponderata Esempio di funzione per calcolare una media ponderata:

```
mean.w <- function(x, w=rep(1,length(x)) )
  { sum(x*w)/sum(w) }
```

Dati sulla produzione per ettaro di diverse coltivazioni e le corrispondenti superficie utilizzate (Forcina, 1996, ex. 8.3):

```
> produzione <- c(131, 60, 139, 127) # si inseriscono i dati della produzione per ettaro
> superficie <- c(16, 24, 78, 27) # e della superficie coltivata
> mean(produzione)
[1] 114.25 # media delle produzioni per ettaro
> mean.w(produzione, superficie)
[1] 122.8069 # produzione media per ettaro
> mean.w(produzione)
[1] 114.25 # assegnando a tutti lo stesso peso si ottiene la media semplice
```

La legge dei grandi numeri Vediamo ora come creare una funzione che illustra la ben nota legge dei grandi numeri nella sua versione più semplice. Si consideri $X_i \sim \text{Bernoulli}(p) \equiv \text{Bin}(1, p)$, e si definisca la v.c. somma cumulata $S_n = X_1 + \dots + X_n$. Questa esprime la frequenza assoluta, mentre la corrispondente frequenza relativa è data da $F_n = S_n/n$. È noto che $E(F_n) = p$ e $\text{Var}(F_n) = p(1-p)/n$. La legge (debole) dei grandi numeri garantisce che $F_n \xrightarrow{\mathcal{P}} p$.

```
demo.LLN <- function(p, n=1000) # la funzione accetta due argomenti:
{ # p (obbligatorio) e n (opzionale)
  x <- rep(NA,n) # crea un vettore vuoto
  plot(0, 0, type="n", ylim=c(0,1), xlim=c(0,n), # prepara un grafico vuoto
       xlab="n", ylab="freq.rel")
  abline(h=p, col=2, lwd=2) # aggiunge una linea in corrispondenza del
                             # valore a cui la serie dovrà convergere
  for (i in 1:n) # loop ... se n è grande sarà molto lento
    x[i] <- rbinom(1, size=1, prob=p)
  # x <- rbinom(n, size=1, prob=p) # in quanto iid sarebbe meglio usare Binomiale
  S <- cumsum(x) # somma cumulata
  F <- S/(1:n) # frequenza relativa
  lines(1:n, F) # disegna linee
  invisible(F) # restituisce la variabile F senza mostrare nulla.
} # Tuttavia il risultato della funzione può essere asse-
gnato.
> demo.LLN(p=0.5, n=1000)
```

Il teorema del limite centrale Si consideri la v.c. X con media μ e varianza σ^2 , ma la cui distribuzione di probabilità non è nota. Date n v.c. X_i iid, si definisca la v.c. $\bar{X}_n = \sum_{i=1}^n X_i/n$, allora per $n \rightarrow \infty$ si ha

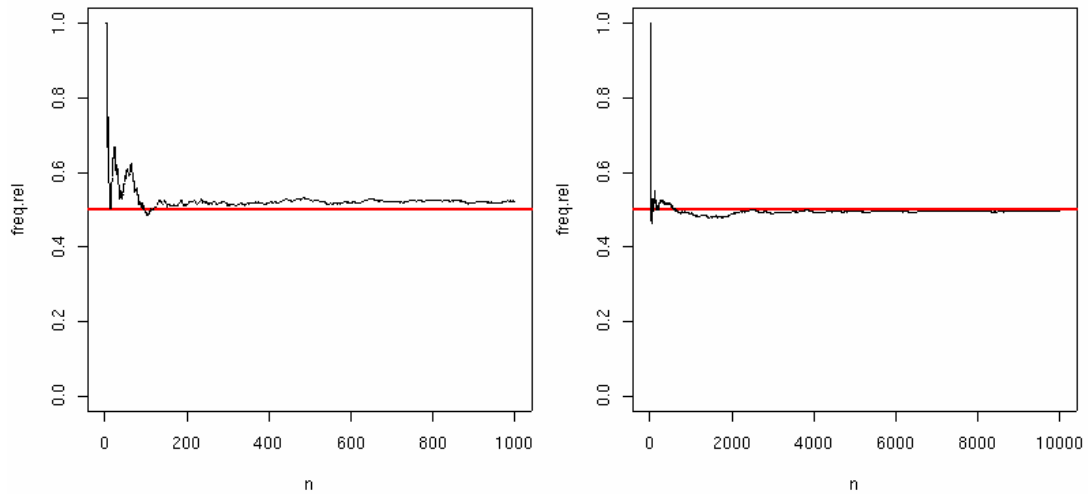
$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{\mathcal{D}} N(0, 1)$$

o equivalentemente

$$\bar{X}_n \xrightarrow{\mathcal{D}} N(\mu, \sigma^2/n)$$

L'approssimazione sarà esatta se $X \sim N(\mu, \sigma^2)$, mentre per un n fissato sarà tanto peggiore quanto più è lontana la distribuzione della v.c. X dalla normalità.

Costruiamo ora una funzione che esegue una simulazione per mostrare il teorema del limite centrale.



```

demo.CLT <- function(df, n, max.n=1000, m=1000)
{
  if (missing(n))
    n <- seq(50, max.n, by=50)
  xbar <- rep(NA, m)
  for (size in n)
    { for (i in 1:m)
      xbar[i] <- mean(rchisq(size,df))
      hist(xbar, probability=TRUE, breaks="scott",
           main=paste("chisq(",df,"), n=",size,", m=",i, sep=""))
      abline(v=mean(xbar), lwd=2, col="red")
    }
  summary(xbar)
}
> demo.CLT(5)
> sample.size <- c(seq(10,50,by=10), 100, 1000)
> demo.CLT(5, sample.size)

```

Il primo comando esegue la simulazione campionando da una distribuzione Chi-quadrato con 5 gradi di libertà. La simulazione per n molto grande può essere lenta, per cui per velocizzare la procedura si possono impostare i valori di n usati nella simulazione, come viene illustrato di seguito.

Provare a ripetere l'esperimento con valori diversi dei gradi di libertà. Cosa succede al crescere dei gradi di libertà? E al crescere di n ?

La stessa simulazione può essere modificata per il caso di una variabile casuale binomiale. Provare a riscrivere la funzione per tale situazione (N.B. ora i parametri sono due, il numero dei tentativi e la probabilità di successo).

9 Analisi univariate

9.1 Statistiche di sintesi

Alcune delle funzioni disponibili per il calcolo delle statistiche univariate di base sono le seguenti:

mean	Arithmetic Mean
var	Correlation, Variance and Covariance (Matrices)
sd	Standard Deviation
median	Median Value
max	Maxima
min	Minima
range	Range of Values
quantile	Sample Quantiles
fivenum	Tukey Five-Number Summaries
IQR	The Interquartile Range
mad	Median Absolute Deviation
rank	Sample Ranks
order	Ordering Permutation
sort	Sorting or Ordering Vectors

Per una descrizione delle singole funzioni si veda l'help in linea. Per ulteriori funzioni si veda la sezione `Statistics/univar` dell'help in linea.

Esempio: inflazione USA, 1948–1989 Riprendiamo il file di dati `inflation.csv` e calcoliamo alcune statistiche descrittive per la variabile `INF` che esprime la variazione percentuale annuale dell'indice dei prezzi al consumo:

```
> data = read.csv("inflation.csv")
> colnames(data)
[1] "CPI" "INF" "UNR" "UNR1" "Year"
> INF
# la variabile non è definita
Error: Object "INF" not found
> attach(data)
> INF
# ora invece è disponibile
[1] 8.071750 -1.244820 1.260510 7.883820 1.923080 0.754720 0.749059 -0.371749 1.492540
[10] 3.308820 2.846970 0.692044 1.718210 1.013510 1.003350 1.324500 1.307190 1.612900
[19] 2.857150 3.086420 4.191610 5.459770 5.722070 4.381450 3.209870 6.220100 11.036000
[28] 9.127790 5.762090 6.502630 7.590760 11.349700 13.498600 10.315500 6.160610 3.212430
[37] 4.317270 3.561110 1.858740 3.649640 4.137330 4.818260
> mean(INF)
[1] 4.223174
> median(INF)
[1] 3.434965
> sd(INF)
[1] 3.412673
> quantile(INF)
 0%      25%      50%      75%     100%
-1.244820 1.522630 3.434965 6.060980 13.498600
> quantile(INF, c(0, 0.10, 0.20, 0.80, 0.90, 1))
 0%      10%      20%      80%      90%     100%
-1.244820 0.779583 1.310652 6.446124 9.022186 13.498600
> min(INF)
[1] -1.24482
> max(INF)
[1] 13.4986
> IQR(INF)
[1] 4.53835
> summary(INF)
```

```

    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.245  1.523   3.435   4.223   6.061  13.500
> rank(INF)                posto occupato nella graduatoria da ciascun elemento
[1] 37  1  8 36 15  5  4  2 11 21 16  3 13  7  6 10  9 12 17 18 25 29 30 27 19 33 40 38 31 34 35 41
[33] 42 39 32 20 26 22 14 23 24 28
> order(INF)              indici ordinati (in senso crescente) degli elementi del vettore
[1]  2  8 12  7  6 15 14  3 17 16  9 18 13 39  5 11 19 20 25 36 10 38 40 41 21 37 24 42 22 23 29 35
[33] 26 30 31  4  1 28 34 27 32 33

```

9.2 Rappresentazioni grafiche

Le rappresentazioni grafiche di distribuzioni univariate completano l'analisi descrittive fornita dalle statistiche, ed inoltre forniscono ulteriori informazioni che altrimenti non sarebbero visibili.

Tra i grafici per distribuzioni univariate possiamo ricordare:

- **Istogrammi**

Un istogramma è una rappresentazione grafica che consente di approssimare la funzione di densità (o di probabilità nel caso discreto) di una distribuzione. La forma di un istogramma è determinata dalla distribuzione delle osservazioni e dal numero dei *bins* in cui è suddiviso il range delle osservazioni (assumendo un'ampiezza dei bins costante). Se l'area di ciascun rettangolo rappresenta la frazione delle osservazioni che appartengono al corrispondente intervallo, l'area totale di un istogramma sarà pari ad 1;

- **Boxplots**

Un boxplot è un grafico che evidenzia la distribuzione di una variabile attraverso i suoi quantili. Esso è costituito da un box al centro del grafico, il quale presenta come estremi il terzo quartile $Q_{0.75}$ e il primo quartile $Q_{0.25}$, mentre la linea al centro del box rappresenta il livello della mediana $Q_{0.5}$. Conseguentemente, il box contiene il 50% delle osservazioni. Inoltre si disegna un segmento verticale a partire dal lato inferiore del box fino al valore più piccolo che supera $Q_{0.25} - 1.5(Q_{0.75} - Q_{0.25})$. Mentre, un altro segmento verticale viene tracciato a partire dal lato superiore del box fino al valore più grande che non supera $Q_{0.75} + 1.5(Q_{0.75} - Q_{0.25})$. Le osservazioni non comprese nei due limiti precedenti sono segnalate nel grafico tramite un punto. Tali valori sono detti *outliers*, ovvero valori estremi o anomali.

- **Diagramma a barre**

Rappresentazione grafica della distribuzione di frequenza di una variabile qualitativa. Per ogni modalità della variabile, le barre (verticali di solito) si estendono per un'altezza pari alla frequenza (assoluta o relativa) corrispondente.

- **Diagrammi circolari**

Anche tali diagrammi (detti "torte") rappresentano graficamente la distribuzione di frequenza di una variabile qualitativa. In tal caso gli spicchi di ciascuna sezione circolare hanno un angolo che è proporzionale alla frequenza (assoluta o relativa) della modalità corrispondente.

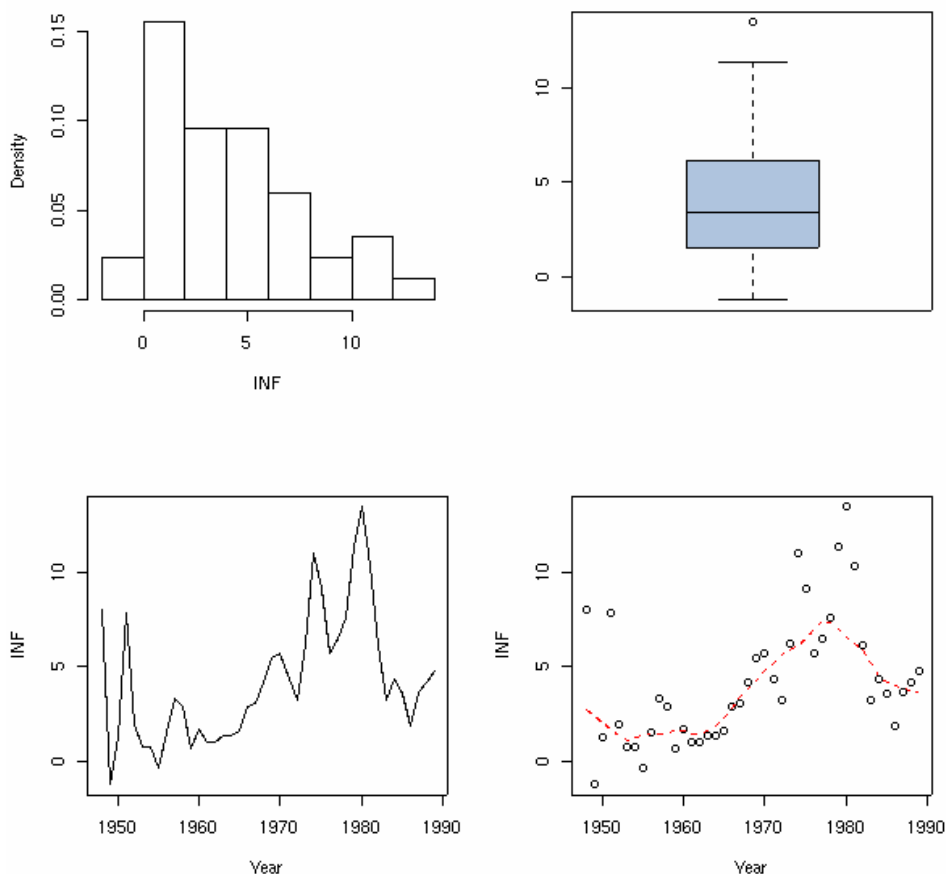
- **Diagrammi a punti**

In R le seguenti funzioni possono essere utilizzate per ottenere i precedenti grafici:

hist	Histograms
boxplot	Box Plots
barplot	Bar Plots
pie	Pie Charts
dotchart	Dot Plots

Esempio: inflazione USA, 1948–1989 Riprendiamo l'esempio sui dati inerenti l'inflazione. I valori di INF si riferiscono a diversi anni, per cui, oltre alle precedenti rappresentazioni grafiche univariate, una rappresentazione che mette in evidenza l'andamento temporale può ottenersi con un grafico della variabile INF vs Years.

```
> par(mfrow=c(2,2))
> hist(INF, probability=TRUE, main="")
> boxplot(INF, col="lightsteelblue")
> plot(Year, INF, type="l")
> plot(Year, INF)
> lines(lowess(Year, INF, f=0.3), col=2, lty=2)
> par(mfrow=c(1,1))
```



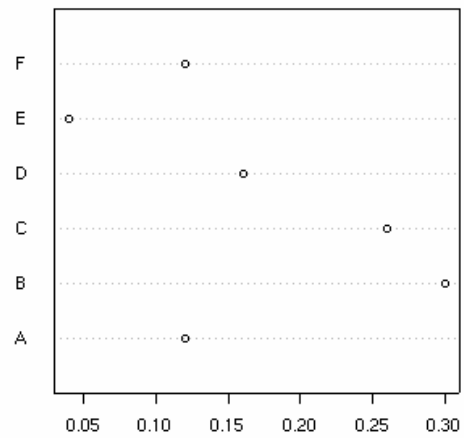
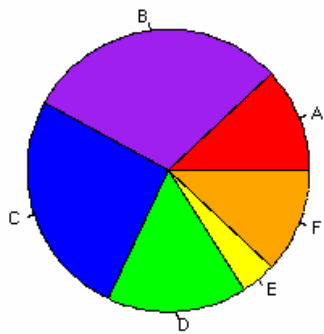
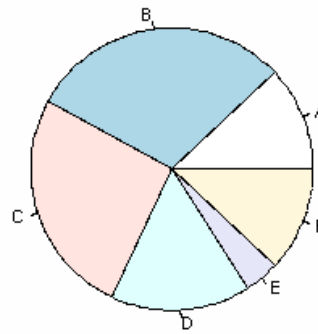
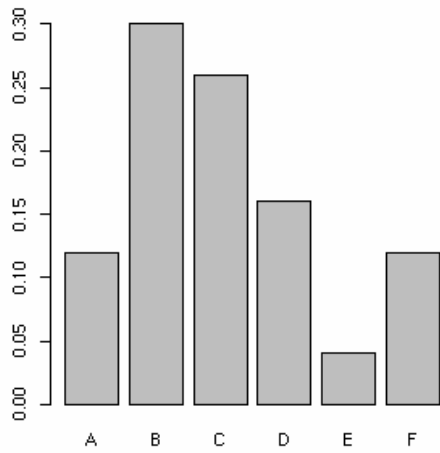
Esempio: distribuzione di frequenza delle vendite per tipo di prodotto Nel caso di distribuzione di frequenza per una variabile qualitativa possiamo (tra le innumerevoli possibilità) rappresentare graficamente la distribuzione come segue:

```
> vendite <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
> names(vendite) <- LETTERS[1:length(vendite)]
> vendite
  A   B   C   D   E   F
0.12 0.30 0.26 0.16 0.04 0.12
> par(mfrow=c(2,2), mar=c(3,3,2,1))
```

```

> barplot(vendite)
> pie(vendite)
> pie(vendite, col = c("red","purple","blue","green","yellow","orange"))
> dotchart(vendite)

```



10 Tabelle di contigenza

Si considerino i seguenti dati che riguardano alcune caratteristiche di 15 clienti che hanno acquistato un'automobile:

```
> Sesso = as.factor(c("M", "M", "F", "M", "F", "M", "F", "M", "M", "F", "M", "M", "M", "M", "F"))
> Eta = c(23, 34, 45, 18, 27, 32, 43, 61, 29, 27, 38, 40, 51, 26, 54)
> PrimaAuto = factor(c(1,0,0,1,1,0,0,0,1,1,0,1,0,1,0), labels = c("No", "Si"))

> table(Sesso) # tabelle marginali
Sesso
 F M
 5 10
> table(Eta)
Eta
18 23 26 27 29 32 34 38 40 43 45 51 61 # (poco utile se non si raggruppa in classi)
 1  1  1  2  1  1  1  1  1  1  1  1  1
> table(PrimaAuto)
PrimaAuto
No Si
 8  7

> tab = table(Sesso)
> tab/sum(tab) # tabella frequenze relative
Sesso
      F      M
0.3333333 0.6666667
> tab = table(Sesso, PrimaAuto) # tabella doppia di frequenza
      PrimaAuto
Sesso No Si
      F  3  2
      M  5  5
> margin.table(tab,1) # distribuzione marginale di riga
Sesso
 F M
 5 10
> margin.table(tab,2) # distribuzione marginale di colonna
PrimaAuto
No Si
 8  7
> sum(tab) # totale osservazioni della tabella
[1] 15

> apply(tab, 2, function(x) x/margin.table(tab,1)) # distribuzione condizionata PrimaAuto|Sesso
      PrimaAuto
      No Si
      F 0.6 0.4
      M 0.5 0.5

> t(apply(tab, 1, function(x) x/margin.table(tab,2))) # distribuzione condizionata Sesso|PrimaAuto
      PrimaAuto
Sesso No      Si
      F 0.375 0.2857143
      M 0.625 0.7142857

> help(cut) # funzione per creare classi
> ETA = cut(Eta, breaks = c(18, 25, 40, 60, 99), include.lowest = TRUE, right=TRUE)
 [1] [18,25] (25,40] (40,60] [18,25] (25,40] (25,40] (40,60] (60,99] (25,40]
[10] (25,40] (25,40] (25,40] (40,60] (25,40] (40,60]
Levels: [18,25] (25,40] (40,60] (60,99]
```

```
> table(ETA)
ETA
[18,25] (25,40] (40,60] (60,99]
      2      8      4      1
```

```
> table(Sesso, ETA)
      ETA
Sesso [18,25] (25,40] (40,60] (60,99]
  F      0      2      3      0
  M      2      6      1      1
```

tabella doppia di frequenza

```
> tab = table(Sesso, ETA, PrimaAuto)
, , PrimaAuto = No
```

tabella tripla di frequenza

```
      ETA
Sesso [18,25] (25,40] (40,60] (60,99]
  F      0      0      3      0
  M      0      3      1      1
```

```
, , PrimaAuto = Si
```

```
      ETA
Sesso [18,25] (25,40] (40,60] (60,99]
  F      0      2      0      0
  M      2      3      0      0
```

dimensioni tabella

```
> dim(tab)
[1] 2 4 2
```

distribuzione condizionata (Sesso,ETA) |PrimaAuto=No

```
> tab[, ,1]
      ETA
Sesso [18,25] (25,40] (40,60] (60,99]
  F      0      0      3      0
  M      0      3      1      1
```

```
> margin.table(tab,c(1,2))
```

distribuzione marginale (Sesso,ETA)

```
      ETA
Sesso [18,25] (25,40] (40,60] (60,99]
  F      0      2      3      0
  M      2      6      1      1
```

11 Modelli di regressione lineare

Un modello di regressione lineare con errori normali, indipendenti ed omoschedastici può essere scritto in notazione matriciale come:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad \boldsymbol{\epsilon} \sim N(0, \sigma^2 \mathbf{I})$$

dove \mathbf{y} è un vettore di n osservazioni, \mathbf{X} è una matrice ($n \times p$) di osservazioni su $(p-1)$ variabili esplicative (la prima colonna è di solito pari a tutti 1 per l'intercetta) e $\boldsymbol{\beta}$ un vettore di p coefficienti sconosciuti.

Lo stimatore dei minimi quadrati (ordinary least squares – OLS) dei coefficienti di regressione è dato dall'espressione

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

posto che $(\mathbf{X}^\top \mathbf{X})$ sia non singolare. Tale stimatore non dipende dalla assunzione di normalità della componente stocastica di errore, ma nel caso questa sia verificata esso è anche lo stimatore di massima verosimiglianza (maximum likelihood estimator – MLE).

A partire dalla stima di $\boldsymbol{\beta}$, si ottengono tutta una serie di statistiche, quali ad esempio:

$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$	residui (residuals)
$\hat{\sigma}^2 = \sum_{i=1}^n e_i^2 / (n - p) = \mathbf{e}^\top \mathbf{e} / (n - p)$	stima varianza dell'errore
$\hat{\mathbf{y}} = \mathbf{X} \hat{\boldsymbol{\beta}}$	valori stimati (fitted values)

Di seguito vengono proposti alcuni esempi di modelli lineari (regressione semplice e multipla, analisi della varianza ad una e due vie, analisi della covarianza), i quali sono tutti riconducibili alla forma sopra introdotta.

11.1 Regressione semplice

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i \quad (i = 1, \dots, n)$$

Esempio: relazione tra numero di settimane di esperienza e numero di pezzi scartati durante l'ultima settimana di produzione da parte di 12 lavoratori.

```
> x = c(7, 9, 6, 14, 8, 12, 10, 4, 2, 11, 1, 8) # Settimane di esperienza
> y = c(26, 20, 28, 16, 23, 18, 24, 26, 38, 22, 32, 25) # Numero di pezzi scartati
> X = cbind(1,x) # Matrice del disegno
      x
[1,] 1 7
[2,] 1 9
[3,] 1 6
...
[10,] 1 11
[11,] 1 1
[12,] 1 8
> n = nrow(X) # Numero osservazioni
> p = ncol(X) # Numero coef. da stimare
> XtXinv = solve( t(X) %*% X ) # Matrice  $(\mathbf{X}^\top \mathbf{X})^{-1}$ 
      x
      0.42773438 -0.044921875
x -0.04492188 0.005859375
> b = XtXinv %*% t(X) %*% y # Stime coef. regressione  $\hat{\boldsymbol{\beta}}$ 
      [,1]
      35.464844
      x -1.386719
> yfit = X %*% b # Valori Stimati  $\hat{\mathbf{y}}$ 
```

```

> e = y - yfit # Residui
> s2 = sum(e^2)/(n-p) # Stima di  $\sigma^2$ 
[1] 6.947656
> s2*XtXinv # Matrice  $\text{Var}(\hat{\beta}) = \hat{\sigma}^2(\mathbf{X}^\top \mathbf{X})^{-1}$ 
      x
      2.9717514 -0.31210175
x -0.3121017 0.04070892
> sqrt(diag(s2*XtXinv)) # Errore std stime dei coefficienti
      x
1.7238769 0.2017645
> R2 = 1 - sum(e^2)/sum((y-mean(y))^2) # Coef. di determinazione  $R^2$ 
[1] 0.8252894

> mod = lm(y ~ x) # Modello di regressione semplice
> summary(mod)

Call:
lm(formula = y ~ x)

Residuals:
    Min       1Q   Median       3Q      Max
-3.9180 -1.5479  0.0957  1.0889  5.3086

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  35.4648     1.7239  20.573 1.63e-09 ***
x            -1.3867     0.2018  -6.873 4.34e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.636 on 10 degrees of freedom
Multiple R-Squared:  0.8253,    Adjusted R-squared:  0.8078
F-statistic: 47.24 on 1 and 10 DF,  p-value: 4.335e-05

> anova(mod)
Analysis of Variance Table

Response: y
      Df Sum Sq Mean Sq F value    Pr(>F)
x         1  328.19   328.19  47.237 4.335e-05 ***
Residuals 10   69.48    6.95
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> class(mod)
[1] "lm"
> names(mod)
 [1] "coefficients" "residuals"      "effects"        "rank"
 [5] "fitted.values" "assign"         "qr"             "df.residual"
 [9] "xlevels"      "call"          "terms"         "model"
[13] "x"

> plot(x, y) # Grafico della var. risposta vs var. esplicativa
> abline(mod$coefficients) # Disegna retta di regressione

> b = mod$coefficients
(Intercept)      x
 35.464844   -1.386719
> df = mod$df.residual

```

```

[1] 10
> s = sqrt(sum(mod$residuals^2)/df)
[1] 2.635841
> qt(0.995, df)
[1] 3.169273
> b[2] + qt(0.995, df) * s * sqrt(XtXinv[2,2])
x
-0.747272
> b[2] - qt(0.995, df) * s * sqrt(XtXinv[2,2])
-2.026166
> confint(mod, level=0.99) # Intervalli di confidenza per i coef.regressione
      0.5 %      99.5 %
(Intercept) 30.001408 40.928280
x           -2.026166 -0.747272

> x0 = c(1,3) # Valore di x per il quale vogliamo calcolare
              # il valore atteso e la previsione
> y0 = x0 %>% b # Stima della var.risposta
      [,1]
[1,] 31.30469

> se.y0.mean = s * sqrt(x0 %>% XtXinv %>% x0) # Errore std per il valore atteso
> c(y0 - qt(0.975, df) * se.y0.mean, # Intervallo conf. per il valore atteso
    y0 + qt(0.975, df) * se.y0.mean)
[1] 28.60733 34.00204

> se.y0 = s * sqrt(1 + x0 %>% XtXinv %>% x0) # Errore std per il valore previsto
> c(y0 - qt(0.975, df) * se.y0, # Intervallo conf. per il valore previsto
    y0 + qt(0.975, df) * se.y0)
[1] 24.84186 37.76751

> x0 = data.frame(x=3)
> predict(mod, newdata=x0, level=0.95, # Intervallo conf. per il valore atteso
          interval="confidence")
      fit      lwr      upr
[1,] 31.30469 28.60733 34.00204
> predict(mod, newdata=x0, level=0.95, # Intervallo conf. per il valore previsto
          interval="prediction")
      fit      lwr      upr
[1,] 31.30469 24.84186 37.76751

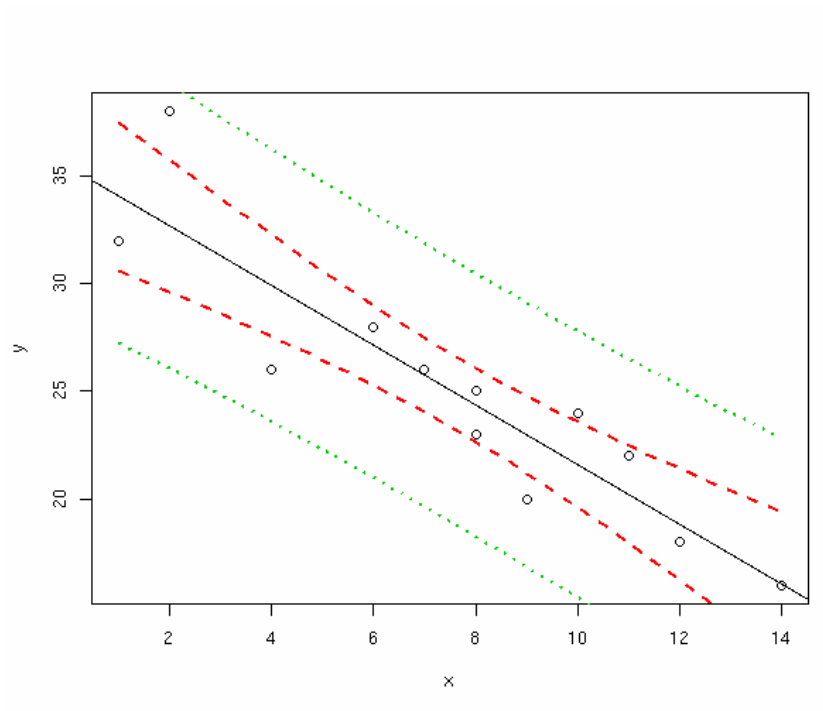
```

Possiamo ora disegnare un grafico di dispersione della y vs x con bande di confidenza puntuali per il valore atteso e previsto:

```

> x0 = data.frame(x=seq(min(x),max(x),length=50))
> Ey = predict(mod, newdata=x0, level=0.95, interval="confidence")
> Py = predict(mod, newdata=x0, level=0.95, interval="prediction")
> plot(x, y)
> abline(mod)
> lines(cbind(x0, Ey[,2]), lty=2, lwd=2, col=2)
> lines(cbind(x0, Ey[,3]), lty=2, lwd=2, col=2)
> lines(cbind(x0, Py[,2]), lty=3, lwd=2, col=3)
> lines(cbind(x0, Py[,3]), lty=3, lwd=2, col=3)

```

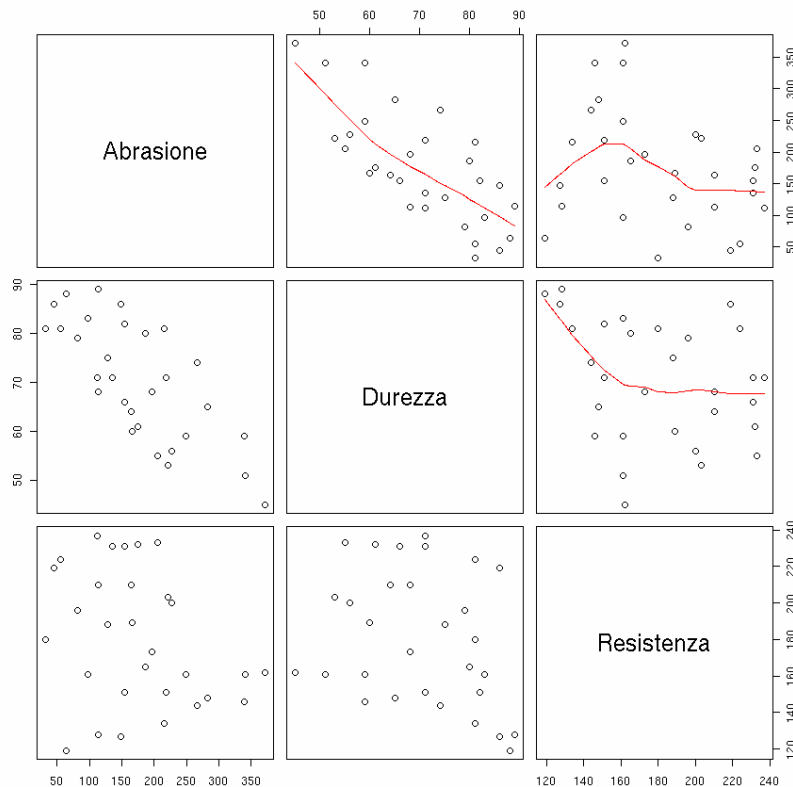


11.2 Regressione multipla

$$Y_i = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p + \epsilon_i \quad (i = 1, \dots, n)$$

Esempio: resistenza all'abrasione. Il file `abrasione.csv` contiene i dati inerenti le misure di resistenza all'abrasione di un certo tipo di plastica. 30 campioni di plastica sono stati sottoposti ad una prova di resistenza per un dato periodo di tempo. La variabile `abrasione` è una misura della perdita di abrasione (in grammi per ora). Le rimanenti variabili esplicative sono caratteristiche del processo produttivo che possono essere controllate. La variabile `durezza` è una misura della durezza della plastica (in gradi Shore), mentre `resistenza` è una misura della resistenza alla trazione dentro (dynes per cm^2).

```
> dati = read.csv("abrasione.csv")
> dati
  Abrasione Durezza Resistenza
1      372     45      162
2      206     55      233
3      175     61      232
...
30     148     86      127
> pairs(dati, upper.panel=panel.smooth) # scatterplot matrix
```



```
> attach(dati)
> mod = lm(Abrasione ~ Durezza + Resistenza) # modello di regressione multipla
> summary(mod)
```

Call:

```
lm(formula = Abrasione ~ Durezza + Resistenza)
```

```
Residuals:
```

```
      Min       1Q   Median       3Q      Max
-79.385 -14.608   3.816  19.755  65.981
```

```
Coefficients:
```

```
      Estimate Std. Error t value Pr(>|t|)
(Intercept)  885.1611    61.7516  14.334 3.84e-14
Durezza      -6.5708     0.5832 -11.267 1.03e-11
Resistenza   -1.3743     0.1943  -7.073 1.32e-07
```

```
Residual standard error: 36.49 on 27 degrees of freedom
Multiple R-Squared: 0.8402, Adjusted R-squared: 0.8284
F-statistic: 71 on 2 and 27 DF, p-value: 1.767e-11
```

```
> mod0 = lm(Abrasione ~ 1) # modello "nullo"
> anova(mod0, mod) # test sulla significatività dell'intera regressione
Analysis of Variance Table
```

```
Model 1: Abrasione ~ 1
Model 2: Abrasione ~ Durezza + Resistenza
  Res.Df  RSS Df Sum of Sq    F    Pr(>F)
1     29 225011
2     27 35950  2   189062 70.997 1.767e-11
```

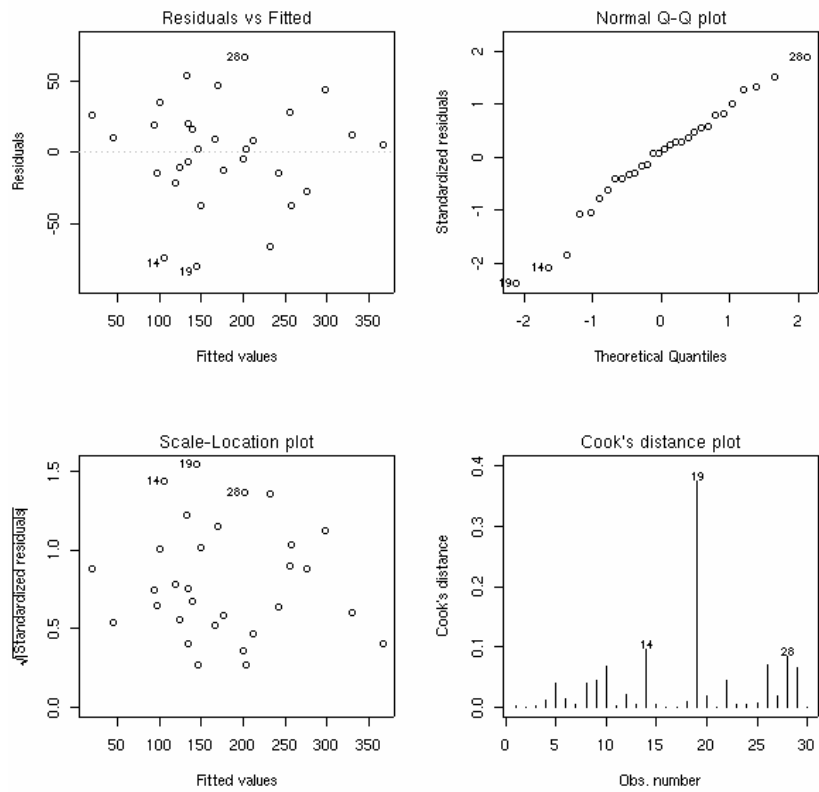
```
> anova(mod) # analisi sequenziale della varianza
Analysis of Variance Table
```

```
Response: Abrasione
      Df Sum Sq Mean Sq F value    Pr(>F)
Durezza  1 122455  122455  91.970 3.458e-10
Resistenza 1  66607   66607  50.025 1.325e-07
Residuals 27  35950    1331
```

```
> par(mfrow=c(2,2))
> plot(mod) # grafici diagnostici
```

```
> smod = summary(mod)
> names(smod)
 [1] "call"          "terms"          "residuals"      "coefficients"
 [5] "aliased"       "sigma"          "df"             "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
> smod$sigma # =  $\hat{\sigma}$ 
36.48934
> smod$cov.unscaled # =  $(\mathbf{X}^T \mathbf{X})^{-1}$ 
      (Intercept)      Durezza      Resistenza
(Intercept)  2.863941746 -2.254472e-02 -6.906905e-03
Durezza      -0.022544720  2.554376e-04  2.546717e-05
Resistenza   -0.006906905   2.546717e-05  2.835655e-05
```

```
> X = model.matrix(mod)
> t(X)%*%X
      (Intercept) Durezza Resistenza
(Intercept)      30    2108    5414
Durezza           2108 152422   376562
Resistenza        5414 376562  1015780
> XtXinv = solve(t(X)%*%X)
      (Intercept)      Durezza      Resistenza
```



```
(Intercept) 2.863941746 -2.254472e-02 -6.906905e-03
Durezza    -0.022544720  2.554376e-04  2.546717e-05
Resistenza  -0.006906905  2.546717e-05  2.835655e-05
> s2 = sum(mod$residuals^2)/mod$df.residual
> sqrt(s2)
36.48934
> s2 * XtXinv
      (Intercept)      Durezza      Resistenza
(Intercept) 3813.25821 -30.01766345 -9.19635018
Durezza     -30.01766  0.34010794  0.03390882
Resistenza   -9.19635  0.03390882  0.03775595
> sqrt(diag(s2 * XtXinv))
(Intercept)      Durezza      Resistenza
61.7515847  0.5831877  0.1943089
```

$$\# = \hat{\sigma}^2(\mathbf{X}^T \mathbf{X})^{-1}$$

```
> confint(mod, level = 0.95)
          2.5 %          97.5 %
(Intercept) 758.457323 1011.8648954
Durezza     -7.767432  -5.3742274
Resistenza  -1.773001  -0.9756228

> x0 = data.frame(Durezza = c(70, 70),
                  Resistenza = c(120, 180))
  Durezza Resistenza
1      70         120
2      70         180
> predict(mod, x0, interval="confidence", level=0.95)
      fit      lwr      upr
```

intervalli di confidenza per i coef. di regressione

data frame contenente i valori per i quali
si intende stimare la previsione

previsione valore medio

```

1 260.2856 232.4877 288.0835
2 177.8269 164.1513 191.5025
> predict(mod, x0, interval="prediction", level=0.95) # previsione valore singolo
      fit      lwr      upr
1 260.2856 180.4218 340.1494
2 177.8269 101.7182 253.9356

```

11.3 ANOVA ad un fattore

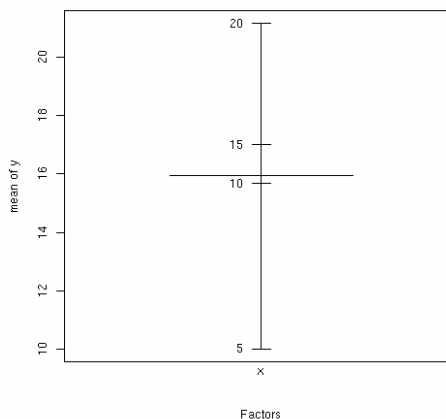
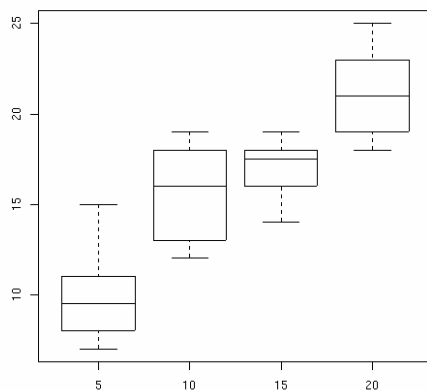
$$Y_{ij} = \mu + \alpha_i + \epsilon_{ij} \quad (i = 1, \dots, I; j = 1, \dots, n_i)$$

Esempio: resistenza alla trazione dei sacchetti di carta.

```

> x = rep(c(5,10,15,20), c(6,6,6,6)) # crea un vettore con i livelli del fattore
[1] 5 5 5 5 5 5 10 10 10 10 10 10 15 15 15 15 15 15 20 20 20 20 20 20
> x = as.factor(x) # trasforma vettore numerico in fattore qualitativo
[1] 5 5 5 5 5 5 10 10 10 10 10 10 15 15 15 15 15 15 20 20 20 20 20 20
Levels: 5 10 15 20
> class(x)
[1] "factor"
> levels(x)
[1] "5" "10" "15" "20"
> table(x)
 5 10 15 20
 6  6  6  6
> y = c(7, 8, 15, 11, 9, 10, 12, 17, 13, 18, 19, 15, # variabile risposta
      14, 18, 19, 17, 16, 18, 19, 25, 22, 23, 18, 20)
> mean(y)
[1] 15.95833
> tapply(y, x, mean) # medie y condizionate ai livelli di x
      5      10      15      20
10.00000 15.66667 17.00000 21.16667
> tapply(y, x, length) # num. osservazioni per ciascun livello di x
 5 10 15 20
 6  6  6  6
> boxplot(y ~ x)
> plot.design(y ~ x)

```



```

> mod = aov(y ~ x) # ANOVA 1 fattore
> summary(mod)
      Df Sum Sq Mean Sq F value    Pr(>F)
x           3  382.79   127.60   19.605 3.593e-06
Residuals  20  130.17     6.51

> model.tables(mod)
Tables of effects

x
      5      10      15      20
-5.958 -0.292  1.042  5.208

> model.tables(mod, type = "means")
Tables of means
Grand mean

15.95833

x
      5      10      15      20
10.000 15.667 17.000 21.167

> model.matrix(mod) # matrice del disegno X
  (Intercept) x10 x15 x20
1             1  0  0  0
2             1  0  0  0
3             1  0  0  0
4             1  0  0  0
5             1  0  0  0
6             1  0  0  0
7             1  1  0  0
8             1  1  0  0
9             1  1  0  0
10            1  1  0  0
11            1  1  0  0
12            1  1  0  0
13            1  0  1  0
14            1  0  1  0
15            1  0  1  0
16            1  0  1  0
17            1  0  1  0
18            1  0  1  0
19            1  0  0  1
20            1  0  0  1
21            1  0  0  1
22            1  0  0  1
23            1  0  0  1
24            1  0  0  1

> mod$coefficients
(Intercept)          x10          x15          x20
 10.000000    5.666667    7.000000   11.166667

> confint(mod, level=0.95)
      2.5 %  97.5 %
(Intercept) 7.827469 12.17253
x10         2.594244  8.73909
x15         3.927577 10.07242
x20         8.094244 14.23909

```

```

> x0 = data.frame(x = levels(x)) # valori medi previsti per i livelli di x
> predict(mod, newdata = x0, interval="confidence", level = 0.95)
      fit      lwr      upr
1 10.00000  7.827469 12.17253
2 15.66667 13.494136 17.83920
3 17.00000 14.827469 19.17253
4 21.16667 18.994136 23.33920

```

11.4 ANOVA a due fattori

$$Y_{ijk} = \mu + \alpha_i + \beta_j + (\alpha \cdot \beta)_{ij} + \epsilon_{ijl} \quad (i = 1, \dots, I; j = 1, \dots, J; k = 1, \dots, n_{ij})$$

Esempio: vernici di base per aerei.

```

> y = c(4.0, 4.5, 4.3, 5.4, 4.9, 5.6, 5.6, 4.9, 5.4, 5.8, 6.1, # variabile risposta
      6.3, 3.8, 3.7, 4.0, 5.5, 5.0, 5.0)
> metodo = rep(rep(c("a immersione", "a spruzzo"), c(3,3)), 3) # primo fattore
> metodo = as.factor(metodo)
> metodo
[1] a immersione a immersione a immersione a spruzzo a spruzzo
[6] a spruzzo a immersione a immersione a immersione a spruzzo
[11] a spruzzo a spruzzo a immersione a immersione a immersione
[16] a spruzzo a spruzzo a spruzzo
Levels: a immersione a spruzzo
> tipo = rep(c(1,2,3), c(6,6,6)) # secondo fattore
> tipo = as.factor(tipo)
> tipo
[1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3
Levels: 1 2 3
> data = data.frame(y, metodo, tipo)
      y      metodo tipo
1 4.0 a immersione 1
2 4.5 a immersione 1
3 4.3 a immersione 1
4 5.4 a spruzzo 1
5 4.9 a spruzzo 1
6 5.6 a spruzzo 1
7 5.6 a immersione 2
8 4.9 a immersione 2
9 5.4 a immersione 2
10 5.8 a spruzzo 2
11 6.1 a spruzzo 2
12 6.3 a spruzzo 2
13 3.8 a immersione 3
14 3.7 a immersione 3
15 4.0 a immersione 3
16 5.5 a spruzzo 3
17 5.0 a spruzzo 3
18 5.0 a spruzzo 3

> tapply(y, metodo, mean) # medie y condizionate ai livelli del fattore metodo
a immersione a spruzzo
 4.466667  5.511111

> tapply(y, tipo, mean) # medie y condizionate ai livelli del fattore tipo
 1 2 3
4.783333 5.683333 4.500000

> tapply(y, list(metodo, tipo), mean) # medie y condizionate ai livelli dei due fattori

```

```

      1      2      3
a immersione 4.266667 5.300000 3.833333
a spruzzo    5.300000 6.066667 5.166667

```

```
# media è massima per tipo="2" e metodo="a spruzzo"
```

```

> tapply(y, list(metodo, tipo), length)
      1 2 3
a immersione 3 3 3
a spruzzo    3 3 3
> replications(y ~ metodo * tipo, data = data)
      metodo      tipo metodo:tipo
      9         6         3

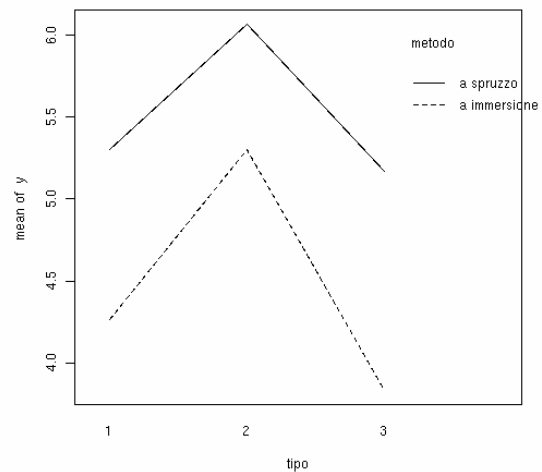
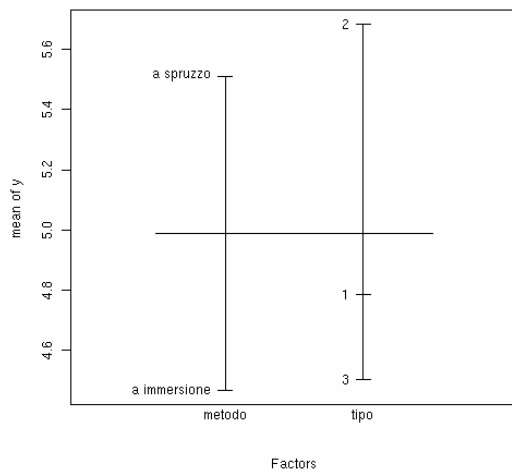
```

```
# num. replicazioni; disegno è bilanciato
```

```

> plot.design(y ~ metodo + tipo)
> interaction.plot(tipo, metodo, y)

```



```

> summary(aov(y ~ metodo))
      Df Sum Sq Mean Sq F value Pr(>F)
metodo  1  4.9089   4.9089  13.521 0.002039
Residuals 16  5.8089   0.3631

```

```
# ANOVA 1 fattore (metodo)
```

```

> summary(aov(y ~ tipo))
      Df Sum Sq Mean Sq F value Pr(>F)
tipo    2  4.5811   2.2906   5.5989 0.01527
Residuals 15  6.1367   0.4091

```

```
# ANOVA 1 fattore (tipo)
```

```

> mod1 = aov(y ~ metodo + tipo)
> summary(mod1)
      Df Sum Sq Mean Sq F value Pr(>F)
metodo  1  4.9089   4.9089  55.975 2.960e-06
tipo    2  4.5811   2.2906  26.119 1.884e-05
Residuals 14  1.2278   0.0877

```

```
# ANOVA 2 fattori senza interazione
```

```

> mod2 = aov(y ~ metodo*tipo)
> summary(mod2)
      Df Sum Sq Mean Sq F value Pr(>F)
metodo  1  4.9089   4.9089  59.7027 5.357e-06
tipo    2  4.5811   2.2906  27.8581 3.097e-05
metodo:tipo 2  0.2411   0.1206   1.4662  0.2693

```

```
# ANOVA 2 fattori con interazione
```

```
<-- interazione non significativa
```

```

Residuals  12 0.9867  0.0822

> model.tables(mod1)
Tables of effects

metodo
a immersione  a spruzzo
      -0.5222    0.5222
tipo
      1      2      3
-0.2056  0.6944 -0.4889

> model.tables(mod1, type="means")
Tables of means
Grand mean
4.988889

metodo
a immersione  a spruzzo
      4.467    5.511
tipo
      1      2      3
4.783 5.683 4.500

> confint(mod1, level=0.99)                                # intervalli di confidenza per i coefficienti del modello
              0.5 %    99.5 %
(Intercept)  3.8455399 4.6766824
metodoa spruzzo 0.6288732 1.4600157
tipo2         0.3910312 1.4089688
tipo3        -0.7923021 0.2256354

> x0 = data.frame(expand.grid(metodo=levels(metodo), tipo=levels(tipo)))
      metodo tipo
1 a immersione  1
2  a spruzzo    1
3 a immersione  2
4  a spruzzo    2
5 a immersione  3
6  a spruzzo    3
> predict(mod1, newdata = x0, interval="confidence", level = 0.95)
      fit      lwr      upr
1 4.261111 3.961696 4.560526
2 5.305556 5.006140 5.604971
3 5.161111 4.861696 5.460526
4 6.205556 5.906140 6.504971
5 3.977778 3.678363 4.277193
6 5.022222 4.722807 5.321637

> n = 3                                                    # intervalli di confidenza simultanei per
> r = nlevels(tipo)                                       # la media dei due fattori separatamente
> t = nlevels(metodo)
> s2 = sum(mod1$residuals^2)/(mod1$df.residual)
[1] 0.08769841

> ym.tipo = tapply(y, tipo, mean)                          # medie y per i livelli del fattore tipo
      1      2      3
4.783333 5.683333 4.500000
> F = qf(0.99, r-1, (r-1)*(t-1)+r*t*(n-1))

```

```

[1] 6.514884
> c = c(1,0,0) # contrasto
> c(c%ym.tipo - sqrt((r-1)*F*s2*sum(c^2)/(t*n)),
  c%ym.tipo + sqrt((r-1)*F*s2*sum(c^2)/(t*n)))
[1] 4.346929 5.219737 # intervallo di confidenza
> c = c(0,1,0) # contrasto
> c(c%ym.tipo - sqrt((r-1)*F*s2*sum(c^2)/(t*n)),
  c%ym.tipo + sqrt((r-1)*F*s2*sum(c^2)/(t*n)))
[1] 5.246929 6.119737 # intervallo di confidenza
> c = c(0,0,1) # contrasto
> c(c%ym.tipo - sqrt((r-1)*F*s2*sum(c^2)/(t*n)),
  c%ym.tipo + sqrt((r-1)*F*s2*sum(c^2)/(t*n)))
[1] 4.063596 4.936404 # intervallo di confidenza

> ym.metodo = tapply(y, metodo, mean) # medie y per i livelli del fattore metodo
a immersione a spruzzo
 4.466667 5.511111
> F = qf(0.99, t-1, (r-1)*(t-1)+r*t*(n-1))
[1] 8.861593
> c = c(1,0) # contrasto
> c(c%ym.metodo - sqrt((t-1)*F*s2*sum(c^2)/(r*n)),
  c%ym.metodo + sqrt((t-1)*F*s2*sum(c^2)/(r*n)))
[1] 4.172813 4.760520 # intervallo di confidenza
> c = c(0,1) # contrasto
> c(c%ym.metodo - sqrt((t-1)*F*s2*sum(c^2)/(r*n)),
  c%ym.metodo + sqrt((t-1)*F*s2*sum(c^2)/(r*n)))
[1] 5.217258 5.804964 # intervallo di confidenza

```

11.5 ANCOVA

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 Z_i + \beta_3 (X \cdot Z) + \epsilon_i \quad (i = 1, \dots, n)$$

```

> data <- read.table("salari.txt", header=TRUE)
  Y EXP NP EDU
1  58.8 4.49 0 3
2  34.8 2.92 0 1
3 163.7 29.54 42 3
4  70.0 9.92 0 3
...
64 60.5 2.10 0 3
65 104.8 19.81 24 3
> attach(data)
> mark = EDU
> EDU = as.factor(EDU)
> plot(EXP, Y, col=mark, pch=mark)
> for(j in 1:3)
+   mod = lm(Y ~ EXP, subset = (mark == j))
+   abline(mod, col=j)

> mod1 = lm(Y ~ EXP*EDU) # Modello ANCOVA con interazione tra fattore e
# variabile esplicativa
> summary(mod1)

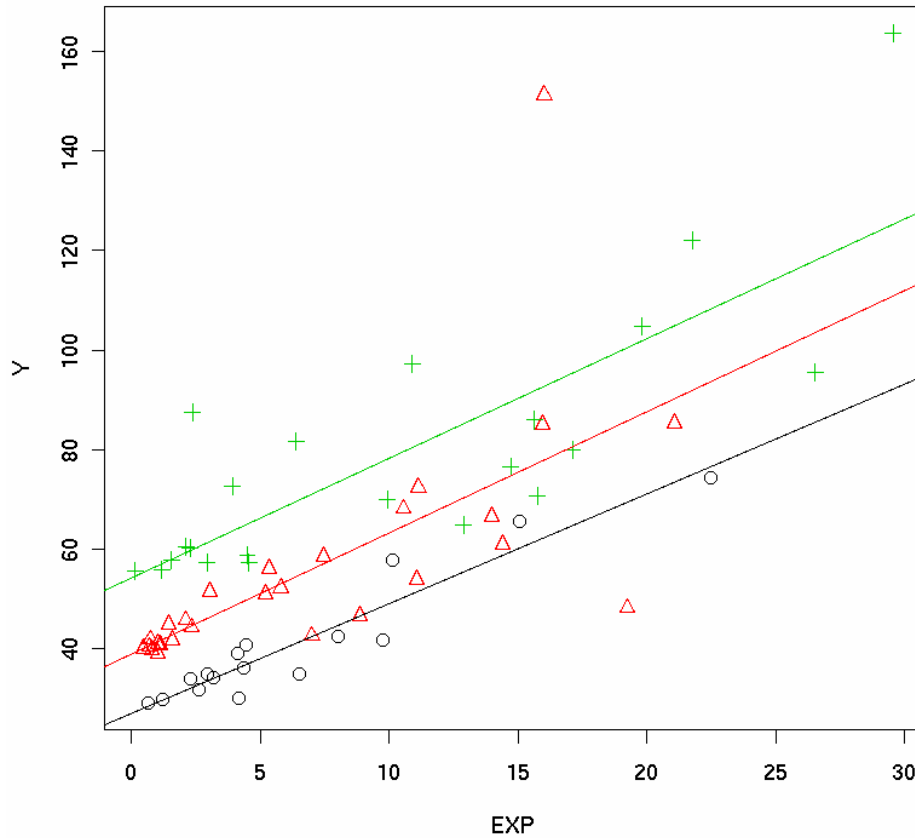
```

Call:

```
lm(formula = Y ~ EXP * EDU)
```

Residuals:

```
Min      1Q  Median      3Q      Max
```



-37.5129 -6.1580 -0.1960 2.9592 73.3403

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	26.9439	5.6380	4.779	1.21e-05
EXP	2.2063	0.6677	3.304	0.001623
EDU2	11.8696	7.0792	1.677	0.098895
EDU3	27.2045	7.4920	3.631	0.000592
EXP:EDU2	0.2326	0.8083	0.288	0.774534
EXP:EDU3	0.2012	0.7625	0.264	0.792774

Residual standard error: 14.83 on 59 degrees of freedom
 Multiple R-Squared: 0.713, Adjusted R-squared: 0.6886
 F-statistic: 29.31 on 5 and 59 DF, p-value: 8.048e-15

> anova(mod1)

Analysis of Variance Table

Response: Y

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
EXP	1	24851.9	24851.9	113.0524	2.457e-15
EDU	2	7342.7	3671.4	16.7012	1.788e-06
EXP:EDU	2	19.6	9.8	0.0447	0.9564
Residuals	59	12969.8	219.8		

```

> mod2 = lm(Y ~ EXP + EDU)                                # Modello ANCOVA senza interazione tra fattore e
                                                            # variabile esplicativa

> summary(mod2)

Call:
lm(formula = Y ~ EXP + EDU)

Residuals:
    Min       1Q   Median       3Q      Max
-36.8752  -5.7723  -0.4258   3.1572  73.8100

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   25.796     4.003   6.444 2.07e-08
EXP             2.387     0.259   9.214 3.70e-13
EDU2           13.383     4.607   2.905 0.00511
EDU3           28.566     4.901   5.828 2.27e-07

Residual standard error: 14.59 on 61 degrees of freedom
Multiple R-Squared: 0.7125,    Adjusted R-squared: 0.6984
F-statistic: 50.4 on 3 and 61 DF,  p-value: < 2.2e-16

> par(mfrow=c(2,2))                                       # Grafici diagnostici
> plot(mod2)                                              # ... quale problema è presente?

```

Riferimenti bibliografici

- Becker R.A., Chambers J.M. and Wilks A.R. (1988) *The New S Language*, Wadsworth & Brooks/Cole, Pacific Grove, CA.
- Chambers J.M.(1998), *Programming with Data*, Springer-Verlag, New York.
- Chambers J.M. and Hastie T. (1992) *Statistical Models in S*, Chapman and Hall, London.
- Dalgaard P. (2002) *Introductory Statistics with R*, Springer, New York.
- Fox J. (2002) *An R and S-Plus Companion to Applied Regression*, Sage Publications, Thousand Oaks, CA, USA.
- Everitt B. (1994) *A Handbook of Statistical Analyses Using S-PLUS*, Chapman and Hall, London.
- Iacus S., Masarotto G. (2003) *Laboratorio di Statistica con R*, McGraw-Hill, Milano.
- Ihaka R. and Gentleman R (1996), R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5, 299–314.
- Venables W.N. and Ripley B.D. (1999) *Modern Applied Statistics with S-PLUS*, Third Edition, Springer, New York.
- Venables W.N. and Ripley B.D. (2000) *S Programming*, Springer, New York.