Matematica e Statistica con R

Federico Comoglio e Maurizio Rinaldi

4 febbraio 2016

Indice

Capitolo 1

Strutture di dati

Premettiamo che per visualizzare un oggetto di R si può usare il comando `print` o il comando `cat` che fornisce spesso un risultato migliore. `str` visualizza la struttura di un oggetto mentre `head` o `tail` ne visualizzano l'inizio o la fine.

1.1 Le stringhe

Una stringa di testo è una collezione di caratteri; in genere, una stringa è resa riconoscibile dall'essere racchiusa tra virgolette.

1.1.1 Operare con le stringhe

Oltre alle virgolette, vi sono numerosi altri caratteri speciali che possono apparire in una stringa. I più comuni sono "`\t`" per `TAB`, " `\n`" per una nuova linea e "`\`" per un singolo *backslash*. Quest'ultimo carattere è un carattere di *escape* e consente una lettura diversa di quanto lo segue. Per esempio

```
> cat("\"sin\"")
```

```
"sin"
```

```
> nchar("\"sin\"")
```

```
[1] 5
```

```
> cat("\\")
```

```
\
```

```
> cat("ora a capo\nsono a capo?")
```

```
ora a capo
sono a capo?
```

```
> cat("ora spazio\triprendo")
```

```
ora spazio        riprendo
```

La funzione `nchar`, che conta il numero di caratteri di una stringa, non includerà quindi il carattere di *escape* nel totale dei caratteri. Ad esempio:

```
> "Tab\t"
[1] "Tab\t"
> cat("Tab\t")
Tab
> nchar("Tab\t")
[1] 4
```

Succede spesso di dover lavorare in modo automatico con stringhe di testo, anche nello scrivere indirizzi di rete o cartelle di lavoro. In R diversi comandi consentono la generazione, manipolazione e stampa di una o più stringhe di testo[1]. Consideriamo inizialmente una singola frase.

```
> x="lavorare con le stringhe"
```

Possiamo verificarne la classe e determinare il numero di caratteri di `x`

```
> class(x)
[1] "character"
> nchar(x)
[1] 24
```

e anche considerare sottostringhe

```
> substr(x,3,8)
[1] "vorare"
```

o abbreviazioni ottenibili con il comando `abbreviate`

```
> abbreviate("Mario Rossi",4)
Mario Rossi
     "MrRs"
```

Certi oggetti possono essere convertiti a stringhe: per esempio il numero 2 può essere visto come una stringa e riconvertito a numero.

[1] Per un uso più specifico si può consultare il pacchetto `biostrings`.

```
>  i=2;toString(i)
```

```
[1] "2"
```

```
> as.numeric(toString(i))
```

```
[1] 2
```

Alcune stringhe molto frequenti sono le lettere dell'alfabeto, maiuscole o minuscole

```
> letters[1:10]
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
> LETTERS[1:10]
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

o i mesi dell'anno (per esempio abbreviati in inglese)

```
> month.abb
```

```
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep"
[10] "Oct" "Nov" "Dec"
```

Le stringhe possono poi essere "incollate" con il comando

```
> paste("a","b",sep="")
```

dove `sep` indica il separatore usato. E tutto insieme

```
> for (i in 1:5) cat(paste("a",toString(i),"\t",sep=""))
```

```
a1      a2      a3      a4      a5
```

Il comando può anche essere utilizzato su vettori. Per esempio

```
> paste(letters[1:10],1:10,sep="")
```

```
 [1] "a1"  "b2"  "c3"  "d4"  "e5"  "f6"  "g7"  "h8"  "i9"
[10] "j10"
```

La *recycling rule* continua a valere

```
> paste(letters[1:3],1:10,sep="")
```

```
 [1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"
[10] "a10"
```

```
> paste(letters[1:3],1:12,sep="")
```

```
 [1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"
[10] "a10" "b11" "c12"
```

e giocando con `rep` si possono ottenere diverse combinazioni.

```
> paste(rep(letters[1:3],each=5),1:15,sep="")
```

```
 [1] "a1"  "a2"  "a3"  "a4"  "a5"  "b6"  "b7"  "b8"  "b9"
[10] "b10" "c11" "c12" "c13" "c14" "c15"
```

```
> paste(rep(letters[1:3],ntimes=5),1:15,sep="")
```

```
 [1] "a1"  "b2"  "c3"  "a4"  "b5"  "c6"  "a7"  "b8"  "c9"
[10] "a10" "b11" "c12" "a13" "b14" "c15"
```

Con l'opzione `collapse="x"` le stringhe vengono unite con separatore la stringa "x".

```
> paste(c("X", "Y"), 1:4, sep = "-", collapse = "--")
```

```
[1] "X-1--Y-2--X-3--Y-4"
```

Si noti il separatore - dell'operazione `paste` e - - dell'operazione `collapse`.

1. Inserisci il tuo cognome in una variabile 'cognome' ed il tuo nome in una variabile 'nome'. Crea una terza variabile 'nomecognome' che contenga entrambi separati da un TAB. Stampa a console la scritta "Good job" seguita dal valore di nomecognome.
2. Creare un elenco che contenga mesi e anno dal 2001 al 2010 nel seguente formato "tre lettere iniziali del mese-anno".
3. Costruire una tabella che contenga tutte le parole di 2 lettere.
4. Si consideri

   ```
   > paste(letters[1:7],1:7,sep="=")
   ```

   Estendere la corrispondenza a tutto l'alfabeto.
5. Creare un elenco in cui a ciascun mese corrisponda il suo numero (a partire da gennaio).
6. Creare un elenco con nomi i mesi e valori il numero di giorni di ciascun mese.
7. Scrivere un elenco di 5 persone con le relative date di nascita nel formato anno-mese-giorno.

1.2 Matrici

Assegnati $n \times m$ ingressi possiamo costruire una matrice (ossia una tabella) con n righe e m colonne. Occorre solo riempire la matrice per righe o per colonne. Ad esempio:

```
> a<-matrix(letters[1:12],nrow=3,ncol=4)
> a
```

```
     [,1] [,2] [,3] [,4]
[1,] "a"  "d"  "g"  "j"
[2,] "b"  "e"  "h"  "k"
[3,] "c"  "f"  "i"  "l"
```

```
> class(a)
```

```
[1] "matrix"
```

Se i parametri hanno natura diversa vengono resi uniformi

```
> a<-matrix(c(1:6,letters[1:6]),nrow=3,ncol=4);a
```

```
     [,1] [,2] [,3] [,4]
[1,] "1"  "4"  "a"  "d"
[2,] "2"  "5"  "b"  "e"
[3,] "3"  "6"  "c"  "f"
```

Con il parametro `byrow=T` il riempimento avviene per righe, anzichè per colonne.

```
>  a<-matrix(1:12,nrow=3,ncol=4,byrow=T)
>  a
```

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

Se i numeri sono insufficienti vengono *riciclati*

```
> a<-matrix(1:4,nrow=3,ncol=4)
> a
```

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

in modo pacifico se sono un sottomultiplo della dimensione della matrice o con qualche `warning` altrimenti.

```
> a<-matrix(2,nrow=3,ncol=4)
> a
```

Si possono anche definire gli ingressi attraverso opportune funzioni

```
> for(j in (1:4)) for(i in (1:3))  a[i,j]<-i^2+j
```

Per assegnare nomi alle righe e alle colonne:

$$\texttt{colnames}(\textit{matrice}) = \texttt{c}(\texttt{" }\textit{nome}_1\texttt{"}, \texttt{"}\textit{nome}_2\texttt{"}, \ldots, \texttt{"}\textit{nome}_n\texttt{"})$$

$$\texttt{rownames}(\textit{matrice}) = \texttt{c}(\texttt{"}\textit{nome}_1\texttt{"}, \texttt{"}\textit{nome}_2\texttt{"}, \ldots, \texttt{"}\textit{nome}_n\texttt{"})$$

```
> colnames(a)=c("c1","c2","c3")
> rownames(a)=c("r1","r2","r3")
> a
```

Aggiungere righe o colonne

Per aggiungere una o più righe (o colonne) ad una matrice si possono usare i comandi (`rbind` e `cbind`)

```
> dim(a)
```

```
[1] 3 4
```

```
> rbind(a,letters[1:ncol(a)])
```

```
     [,1] [,2] [,3] [,4]
[1,] "1"  "4"  "3"  "2"
[2,] "2"  "1"  "4"  "3"
[3,] "3"  "2"  "1"  "4"
[4,] "a"  "b"  "c"  "d"
```

```
> cbind(a,letters[1:nrow(a)])
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,] "1"  "4"  "3"  "2"  "a"
[2,] "2"  "1"  "4"  "3"  "b"
[3,] "3"  "2"  "1"  "4"  "c"
```

Possiamo anche effettuare semplici operazioni, come somma degli elementi delle righe o delle colonne

```
> colSums(a)
```

```
[1] 6 7 8 9
```

```
> rowSums(a)
```

```
[1] 10 10 10
```

1.2.1 Operazioni con le matrici

Trasposizione

Per invertire righe e colonne di una matrice ossia per ottenere il trasposto di una matrice si usa il comando `t(matrice)`

```
> t(a)
```

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

Prodotto

Per la moltiplicazione di matrici (definita per ingressi numerici) si usa il simbolo **%*%**.

```
> b<-matrix(2,nrow=3,ncol=3)
> for(j in (1:3))
+ for(i in (1:3)) b[i,j]<-i+j+i^2
> b
```

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    7    8    9
[3,]   13   14   15
```

Non è infatti possibile moltiplicare una matrice 3x4 con una 3x3. Possiamo però calcolare

```
> b%*%a
```

```
     [,1] [,2] [,3] [,4]
[1,]   26   26   30   38
[2,]   50   54   62   74
[3,]   86   96  110  128
```

Determinante

Il determinante di una matrice quadrata si ottiene con il comando

$$\texttt{det}(\textit{matrice}) \tag{1.1}$$

```
> det(b)
```

```
[1] 1.887379e-14
```

Si noti che se eseguendo i calcoli a mano si trovano in alcuni casi risultati diversi da quelli di R. Per esempio la matrice in esame ha determinante 0 e 0 ne è anche un autovalore.

1. Creare una matrice 3×2 che abbia come ingressi i primi 6 numeri pari. Estendere la matrice aggiungendo due colonne contenenti i primi 6 numeri dispari. Calcolare e stampare la somma per riga e la somma per colonna. Modifica la matrice cambiando di segno la prima riga. Moltiplicare la matrice per 4.

1.3 I *dataframe*

I *dataframe* (in R `data.frame`) costituiscono in R la classe di oggetti fondamentali per la collezione di dati per una susseguente analisi statistica. Un *dataframe* è una collezione di vettori aventi egual lunghezza e allineati verticalmente. Un dataframe è diverso da una matrice in quanto le colonne sono vettori eventualmente di tipi diversi. Il comando generale per costruire *dataframe* a partire da vettori o liste è `data.frame`. Esso richiede come parametri i nomi dei vettori (colonna) da affiancare nella tabella. In generale si scrive:

$$\texttt{data.frame}(\mathit{vettore}_1, \mathit{vettore}_2, \ldots, \mathit{vettore}_n)$$

dove tutti i vettori hanno la stessa lunghezza. Si noti la asimmetria (rispetto ad una matrice) nel ruolo di righe e colonne. Le colonne sono omogenee, lo stesso non si può dire per le righe. Le colonne sono le variabili analizzate, le righe le unità statistiche. Anche i vari comandi che vedremo rispettano tale differenza. Il data.frame classico da cui partiamo è `iris`

Per avere una stampa abbreviata

```
> head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
> tail(iris)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width
145          6.7         3.3          5.7         2.5
146          6.7         3.0          5.2         2.3
147          6.3         2.5          5.0         1.9
148          6.5         3.0          5.2         2.0
149          6.2         3.4          5.4         2.3
150          5.9         3.0          5.1         1.8
      Species
145 virginica
146 virginica
147 virginica
148 virginica
149 virginica
150 virginica
```

Il comando `str` consente una visualizzazione parziale che ci fornisce la struttura.

```
'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Per esempio possiamo considerare il dataframe `d` definito come segue

```
> L3 <- LETTERS[1:3]
> d <- data.frame(cbind(x=1, y=1:10,
+ fac=sample(L3, 10, replace=TRUE)),
+ stringsAsFactors=TRUE)
> d
```

```
   x  y fac
1  1  1   A
2  1  2   B
3  1  3   C
4  1  4   B
5  1  5   C
6  1  6   B
7  1  7   B
8  1  8   B
9  1  9   B
10 1 10   B
```

Si noti che anche in questo caso si usa la *recycling rule*

```
> data(crabs)
> dim(crabs)
[1] 201   8
```

In quanto segue lavoreremo con il seguente *dataframe* che rappresenta i risultati di un'indagine svolta sugli studenti che nell'Anno Accademico 2007/2008 frequentavano il primo anno del corso di Laurea di Farmacia della Facoltà di Farmacia del Piemonte Orientale. Potete scaricarlo dal sito e caricarlo in R con il comando

```
> data(farmacia)
```

Il comando `colnames` consente di visualizzare o assegnare il nome alle colonne. Per esempio

```
> colnames(farmacia)
[1] "Sex"  "W"    "H"    "Eyes" "Hair" "Sh"   "hM"
[8] "hF"
```

Le varie colonne hanno dei nomi di facile interpretazione. Si noti anche che a fianco di variabili numeriche (`W` e `H`, peso-altezza ad esempio) sono presenti variabili *nominali* quali sesso (`Sex`) e colore degli occhi (`Eyes`). Per associare i nomi alle colonne alle varie colonne dobbiamo eseguire una operazione di collegamento con il comando `attach`,

```
> attach(farmacia)
```

A questo punto digitando il nome delle colonne appare il contenuto della colonna

```
> Sex
 [1] F M M M F M F F M F F F F F M F F F M M M F F F F
[26] F M F M M F F M M F F F F M F M M F F F F F F F M
[51] M M F F M
Levels: F M
```

Le variabili nominali sono caratterizzate dal fatto che i loro valori (livelli, `levels`) non hanno significato numerico, anche se possono essere codificati con dei numeri. Ad esempio il sesso di una persona è una variabile nominale con due possibili valori, che sono stati indicati qui con la convenzione `"F"` per le femmine e `"M"` per i maschi. Se volessimo eliminare i livelli di una variabile nominale potremmo scrivere

```
> Sex=as.vector(Sex)
> Sex
 [1] "F" "M" "M" "M" "F" "M" "F" "F" "M" "F" "F" "F"
[13] "F" "F" "M" "F" "F" "F" "M" "M" "M" "F" "F" "F"
[25] "F" "F" "M" "F" "M" "M" "F" "F" "M" "M" "F" "F"
[37] "F" "F" "M" "F" "M" "M" "F" "F" "F" "F" "F" "F"
[49] "F" "M" "M" "M" "F" "F" "M"
```

```
> class(Sex)
```

```
[1] "character"
```

Possiamo anche considerare il processo inverso e cambiare una variabile priva di livelli in una nominale

$$\texttt{factor}(\textit{variabile}) \rightarrow \textit{variabile}$$

Per definire i suoi livelli (ad esempio n) scriveremo:

$$\texttt{levels}(\textit{variabile}) \leftarrow \texttt{c}(\textit{nome}_1, \textit{nome}_2, \ldots, \textit{nome}_n)$$

Per rendere la variabilie `Sex` nominale con nomi dei livelli `F` e `M`) scriveremo:

```
> Sex=factor(Sex)
> Sex
```

```
 [1] F M M M F M F F M F F F F F M F F F M M M F F F F
[26] F M F M M F F M M F F F F M F M M F F F F F F F M
[51] M M F F M
Levels: F M
```

Con il comando `detach` si può eliminare l'associazione creata tra colonne e nomi delle colonne. Consideriamo ora un *dataset* simile raccolto dagli studenti di Biotecnologie dello stesso anno

```
> sito="http://www.pharm.unipmn.it/rinaldi"
> nomefile="/file_corso/datibiotec.csv"
> biotec=read.table(paste(sito,nomefile,sep=""),sep=";",dec=",",header=T)
```

Scrivendo

```
> class(biotec)
```

```
[1] "data.frame"
```

vediamo che anche `biotec` è un *dataframe*. Inoltre confrontando i nomi delle colonne di `farmacia` e di `biotec` possiamo verificare che sono essenzialmente uguali a meno di traduzione e eventuale abbreviazione. Possiamo creare un *dataframe* unico che raggruppi `biotec` e `farmacia`. Per farlo vorremmo incollare un *dataframe* sopra all'altro. A tal fine occorre uniformare i nomi delle colonne scrivendo per esempio

```
> colnames(biotec)=colnames(farmacia)
> studenti=rbind(farmacia,biotec)
> head(studenti)
```

```
  Sex  W    H     Eyes    Hair Sh   hM   hF
1   F 62 1.75  CASTANI CASTANI 40 1.77 1.78
2   M 64 1.84  CASTANI CASTANI 43 1.72 1.80
3   M 80 1.70  CASTANI CASTANI 44 1.65 1.73
4   M 80 1.75  CASTANI    NERI 44 1.66 1.78
5   F 50 1.70 NOCCIOLA    NERI 38 1.65 1.79
6   M 63 1.88  CASTANI  BIONDI 44 1.58 1.75
```

Si noti che il comando `rbind` incolla per riga, mentre l'analogo comando `cbind` incolla le colonne. Le intestazioni di riga di dati sono

```
> rownames(studenti)
```

```
 [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"
 [8] "8"   "9"   "10"  "11"  "12"  "13"  "14"
[15] "15"  "16"  "17"  "18"  "19"  "20"  "21"
[22] "22"  "23"  "24"  "25"  "26"  "27"  "28"
[29] "29"  "30"  "31"  "32"  "33"  "34"  "35"
[36] "36"  "37"  "38"  "39"  "40"  "41"  "42"
[43] "43"  "44"  "45"  "46"  "47"  "48"  "49"
[50] "50"  "51"  "52"  "53"  "54"  "55"  "110"
[57] "210" "310" "410" "56"  "61"  "71"  "81"
[64] "91"  "101" "111" "121" "131" "141" "151"
[71] "161" "171" "181" "191" "201" "211" "221"
[78] "231" "241" "251" "261" "271" "281" "291"
[85] "301" "311" "321" "331" "341" "351" "361"
[92] "371" "381" "391" "401" "411"
```

Per correggere la strana numerazione possiamo scrivere

```
> rownames(studenti)=seq(length=nrow(studenti))
```

Giunti a questo punto la tabella `dati` presenta ancora alcuni problemi; per esempio se scriviamo

```
> levels(studenti$Eyes)
```

```
 [1] "AZZURRI"  "CASTANI"  "MARRONI"  "NERI"
 [5] "NOCCIOLA" "VERDI"    "azzurri"  "castani"
 [9] "marroni"  "verdi"
```

```
> levels(studenti$Hair)
```

```
[1] "BIONDI"         "CASTANI"
[3] "NERI"           "biondi"
[5] "castani"        "castano chiaro"
[7] "castano scuro"  "neri"
```

Ci incuriosisce il dato con gli occhi neri. Verifichiamo:

```
> studenti[which(studenti$Eyes=="NERI"),]
```

```
   Sex  W   H Eyes Hair Sh   hM   hF
20   M 69 1.7 NERI NERI 41 1.55 1.75
```

Possiamo ritenere che sia un errore e che in realtà gli occhi siano marroni molto scuri. Risulta evidente che nel riportare i colori degli occhi si sono usate dizioni diverse per colori essenzialmente uguali, per esempio i livelli `"CASTANI"`, `"NOCCIOLA"`, `"MARRONI"` possono esser fatti confluire in un unico livello `"castani"` e possiamo rendere minuscoli i nomi degli altri livelli con il comando

```
> options(width=50)
> levels(studenti$Eyes)=c("azzurri","castani","castani", "castani", "castani","verdi","azzu
```

A questo punto

```
> levels(studenti$Eyes)
```

```
[1] "azzurri" "castani" "verdi"
```

Facciamo lo stesso con i capelli

```
> levels(studenti$Hair)=c("biondi","castani","neri", "biondi", "castani","castani","castani
> levels(studenti$Hair)
```

```
[1] "biondi"  "castani" "neri"
```

Selezione in base a criteri

Supponiamo di voler selezionare gli studenti con gli occhi castani. Basta scrivere

```
> subset(studenti,studenti$Eyes=="verdi")
```

```
   Sex  W    H  Eyes    Hair Sh   hM   hF
7    F 63 1.70 verdi castani 38 1.72 1.82
17   F 51 1.55 verdi castani 37 1.60 1.70
25   F 91 1.81 verdi  biondi 42 1.60 1.87
33   M 75 1.82 verdi castani 43 1.60 1.75
35   F 46 1.64 verdi castani 37 1.56 1.89
37   F 56 1.70 verdi castani 39 1.68 1.90
38   F 55 1.65 verdi castani 38 1.68 1.70
41   M 56 1.70 verdi castani 39 1.65 1.80
42   M 67 1.73 verdi castani 42 1.55 1.85
47   F 52 1.75 verdi  biondi 38 1.62 1.80
51   M 75 1.76 verdi castani 42 1.60 1.68
58   M 64 1.74 verdi castani 41 1.63 1.80
```

```
60   M 64 1.80 verdi castani 42 1.56 1.75
72   F 55 1.67 verdi castani 40 1.60 1.80
76   M 85 1.84 verdi castani 43 1.69 1.69
77   F 60 1.67 verdi castani 38 1.64 1.70
81   F 62 1.61 verdi castani 39 1.60 1.66
90   F 49 1.60 verdi castani 40 1.58 1.75
91   F 62 1.76 verdi  biondi 40 1.70 1.73
93   F 53 1.65 verdi castani 38 1.55 1.85
95   M 80 1.80 verdi castani 44 1.68 1.70
```

Se siamo invece interessati al colore dei capelli degli studenti con occhi castani

```
> subset(studenti,studenti$Eyes=="verdi",select="Hair")
```

```
      Hair
7  castani
17 castani
25  biondi
33 castani
35 castani
37 castani
38 castani
41 castani
42 castani
47  biondi
51 castani
58 castani
60 castani
72 castani
76 castani
77 castani
81 castani
90 castani
91  biondi
93 castani
95 castani
```

1.4 Gli *array*

Un *array* è una generalizzazione multidimensionale di una matrice. Gli *array* sono caratterizzati dal numero di dimensioni (se le dimensioni sono 2 un *array* si identifica con una matrice) e dal nome dei vari livelli

```
> array(LETTERS[1:24],
+ dim=c(2,3,4))
```

```
, , 1

     [,1] [,2] [,3]
[1,] "A"  "C"  "E"
[2,] "B"  "D"  "F"

, , 2

     [,1] [,2] [,3]
[1,] "G"  "I"  "K"
[2,] "H"  "J"  "L"

, , 3

     [,1] [,2] [,3]
[1,] "M"  "O"  "Q"
[2,] "N"  "P"  "R"

, , 4

     [,1] [,2] [,3]
[1,] "S"  "U"  "W"
[2,] "T"  "V"  "X"
```

```
> array(sample(1:100,24), dim=c(3,4,2),dimnames=list(LETTERS[1:3],LETTERS[11:14],letters[1:
> x[,,"b"]
```

```
   K  L  M  N
A 60 77 15  6
B 31 26 64 88
C 37 68 87 18
```

1.5 Liste

Una lista (in R `list`) è un vettore di oggetti. Gli oggetti possono avere un nome ed avere natura diversa fra di loro. Per esempio

```
> x=list(a=month.abb , b=array(rep(0,20), dim=c(4,5)),c="your name")
> x
```

```
$a
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul"
 [8] "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
$b
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0

$c
[1] "your name"
```

Possiamo annidare anche liste entro liste

```
> x=list(a=1:10,b=array(rep(0,20),dim=c(4,5)),
+ c="testo",d=list(g="h",r=1:10) )
> x
```

```
$a
 [1]  1  2  3  4  5  6  7  8  9 10

$b
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
[4,]    0    0    0    0    0

$c
[1] "testo"

$d
$d$g
[1] "h"

$d$r
 [1]  1  2  3  4  5  6  7  8  9 10
```

1.5.1 Funzioni applicate a oggetti

La funzione `apply`

Quando abbiamo una struttura di dati a più dimensioni ad esempio un *array*

```
> array(1:30,dim=c(2,5,3))->x
> x
```

```
, , 1

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

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]   11   13   15   17   19
[2,]   12   14   16   18   20

, , 3

     [,1] [,2] [,3] [,4] [,5]
[1,]   21   23   25   27   29
[2,]   22   24   26   28   30
```

possiamo applicare una funzione selezionando il livello escluso:

```
> apply(x,2,mean)
```

```
[1] 11.5 13.5 15.5 17.5 19.5
```

```
> apply(x,c(1,3),mean)
```

```
     [,1] [,2] [,3]
[1,]    5   15   25
[2,]    6   16   26
```

In un *dataframe* possiamo considerare diverse operazioni che coinvolgano due o più colonne. Per esempio possiamo considerare la tabella `farmacia` e proporci di calcolare la media del peso dei maschi e la media del peso delle femmine. Il comando `tapply` consente di applicare una funzione ad una colonna di una tabella ripartita in gruppi in accordo ad un criterio specificato da un'altra colonna (nominale) della stessa tabella

`tapply`(*vettore*, *criterio*, *funzione*)

L'esempio sopra riportato si risolve con la scrittura:

```
> tapply(studenti$W, studenti$Sex,mean)->Wmean
> Wmean
```

```
       F        M
58.19298 69.34615
```

Possiamo anche chiedere di calcolare altri indicatori sempre basandoci su questa ripartizione maschi/femmine. Ad esempio possiamo ottenere la deviazione standard(`sd`) di maschi e femmine:

```
> tapply(studenti$W, studenti$Sex,sd)->Wsd
> Wsd
```

```
       F        M
10.01149 10.61971
```

Il comando `table` applicato a diverse variabili nominali e le tabelle di contingenza

Consideriamo il seguente *dataframe* che riporta le ambizioni di un gruppo di scolari americani

```
> read.table("http://lib.stat.cmu.edu/DASL/Datafiles/PopularKids.html",
+ skip=39,header=T,nrow=478,sep="\t")->kidinterest
```

```
  Gender Grade Age  Race Urban.Rural School
1    boy     5  11 White       Rural    Elm
2    boy     5  10 White       Rural    Elm
3   girl     5  11 White       Rural    Elm
4   girl     5  11 White       Rural    Elm
5   girl     5  10 White       Rural    Elm
6   girl     5  11 White       Rural    Elm
    Goals Grades Sports Looks Money
1  Sports      1      2     4     3
2 Popular      2      1     4     3
3 Popular      4      3     1     2
4 Popular      2      3     4     1
5 Popular      4      2     1     3
6 Popular      4      2     1     3
```

Nella tabella le colonne che ci interessano al momento sono quelle che riguardano il sesso, gli obiettivi (scelti tra successo scolastico, capacità sportiva e popolarità) e la provenienza (colonne 1, 5 e 7). Nelle colonne dalla 8 alla 11 sono messi in ordine di importanza per il conseguimento della popolarità voti, sport, aspetto esteriore e denaro.

```
> colnames(kidinterest)
```

```
 [1] "Gender"      "Grade"       "Age"
 [4] "Race"        "Urban.Rural" "School"
 [7] "Goals"       "Grades"      "Sports"
[10] "Looks"       "Money"
```

```
> interessi=kidinterest[,c(1,5,7)]
> head(interessi)
```

```
  Gender Urban.Rural   Goals
1    boy       Rural  Sports
2    boy       Rural Popular
3   girl       Rural Popular
4   girl       Rural Popular
5   girl       Rural Popular
6   girl       Rural Popular
```

```
> table(interessi)
```

```
, , Goals = Grades

      Urban.Rural
Gender Rural Suburban Urban
  boy     21       51    45
  girl    36       36    58

, , Goals = Popular

      Urban.Rural
Gender Rural Suburban Urban
  boy     19       20    11
  girl    31       22    38

, , Goals = Sports

      Urban.Rural
Gender Rural Suburban Urban
  boy     26       18    16
  girl    16        4    10
```

Consideriamo per esempio le variabili provenienza e traguardi

```
> interessi2=kidinterest[,c(5,7)]
> tabella=table(interessi2)
> tabella
```

```
           Goals
Urban.Rural Grades Popular Sports
   Rural        57      50     42
   Suburban     87      42     22
   Urban       103      49     26
```

La tabella di contingenza delle due variabili (in questo caso a 3 livelli ciascuna) si realizza con il comando `table`. `table`(*(*lista)) applicato ad una singola variabile nominale

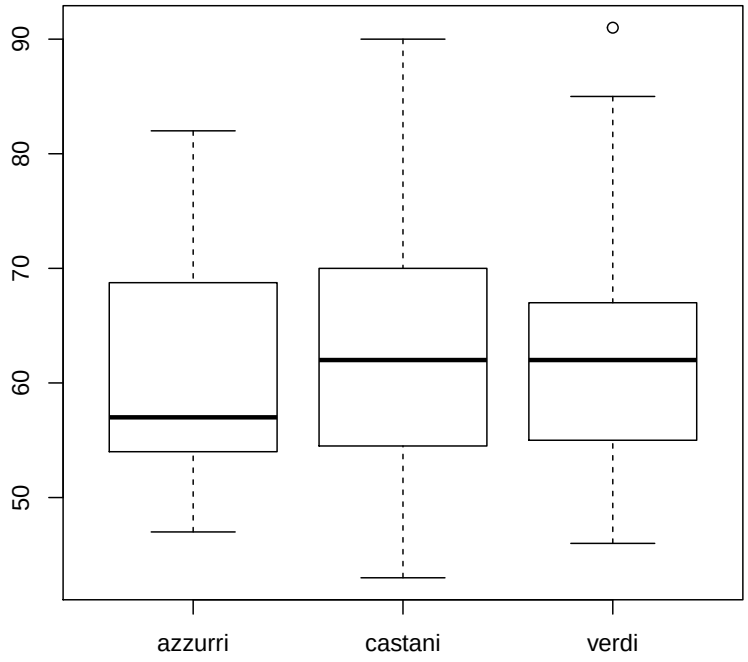
conta le frequenze di ciascuna classe, se le variabili sono n esamina tutte le combinazioni possibili (2^n se le variabili sono dicotomiche) e ne conta tutte le occorrenze

Allo stesso modo possiamo creare una tabella dei colori degli occhi e dei capelli

```
> tabellaEH=table(studenti$Eyes,studenti$Hair)
```

O ripartirne il peso in base al colore degli occhi

```
> stEyeW=split(studenti$W,studenti$Eyes)
> boxplot(stEyeW)
```



1. Costruire una funzione `inserisci` tale che il comando `inserisci`(x, *dove*, *cosa*) inserisca nel vettore x l'elemento *cosa* nella posizione *dove*. Per esempio

```
> y <-letters[1:10]
> y
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
> inserisci(x, 5, 0)
```

```
[1] "a" "b" "c" "d" "0" "e" "f" "g" "h" "i" "j"
```

Generalizzare al caso di inserimento di righe in *dataframe*.

2. Come si può verificare l'uguaglianza di 2 vettori includendo gli ingressi `NA`

   ```
   > NA==NA
   ```

   Aiuto: usare `is.na`

3. [1] Carichiamo il *dataset* `juul` dal pacchetto `ISwR`. Ricavare il sottoinsieme di dati che corrispondono a ragazze tra i 7 e i 14 anni. Plottare `igf1` versus età per ragazzi e ragazze separatamente. Conclusioni?

Capitolo 2

Operatori logici e selezione di elementi

Consideriamo la selezione di elementi di un oggetto di R, per esempio `crabs`. Selezioniamo per esempio tra i primi 50 i granchi con carapaci di lunghezza (sesta colonna) maggiore di 42:

```
> crabs[1:50,6]>42
```

```
 [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[16] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[31] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[46] NA NA NA NA NA
```

Ciascun elemento del vettore viene confrontato con la soglia scelta (42) e quando supera la soglia la risposta è `TRUE` altrimenti `FALSE`.

operatore	significato
>	maggiore
>=	maggiore o uguale
<	minore
<=	minore o uguale
==	uguale
!=	diverso
&	and (e)
|	or (o)

Gli ultimi due operatori combinano gli operatori precedenti. Per esempio

```
> crabs[crabs$sex == "F" | crabs$sp == "O",]
> dim(crabs[crabs$sex == "F" & crabs$sp == "O",])
```

2.0.1 I valori non assegnati

Capita frequentemente che un oggetto contenga valori non assegnati. Per esempio

```
> x=1;x[10]=4;x
 [1]  1 NA NA NA NA NA NA NA NA  4
```

```
> x[!is.na(x)]
[1] 1 4
```

```
> class(x)
[1] "numeric"
```

```
> class(unclass(x))
[1] "numeric"
```

```
> is.na(x)
 [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
 [8]  TRUE  TRUE FALSE
```

```
>  DF <- data.frame(x = c(1, 2, 3), y = c(0, 10, NA))
> na.omit(DF)
  x  y
1 1  0
2 2 10
```

```
> complete.cases(x)
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
 [8] FALSE FALSE  TRUE
```

Capitolo 3

Statistica con R

3.1 Variabili aleatorie

Una variabile aleatoria (*random variabile*) è una variabile i cui valori sono soggetti a variazioni casuali. Quando i valori possibili di una variabile aleatoria possono essere elencati parliamo di variabile aleatoria discreta. Quando i valori non possono essere elencati parliamo di variabile aleatoria continua.

3.2 Variabili aleatorie discrete

Le variabili aleatorie discrete che assumono un numero limitato di valori si dicono anche *finite*. I valori di una variabile aleatoria discreta possono essere numerici o nominali. Supponiamo di avere una variabile aleatoria che possa assumere un insieme di valori in un *alfabeto* assegnato costituito da lettere, parole o numeri. Per esempio un alfabeto può essere del tipo che segue

- (Femmina, Maschio)
- (A,C,T,G)
- (0,1)
- (Ottimo, Buono, Discreto, Sufficiente, Insufficiente)
- (Testa, Croce).
- I numeri interi

Per caratterizzare completamente una variabile aleatoria discreta oltre ai valori che questa può assumere occorre conoscere la probabilità di questi valori.

Per semplicità considereremo variabili aleatorie finite.
Come possiamo simulare variabili aventi valore nell'alfabeto assegnato? In effetti qualunque

comando di generazione su un computer non è perfettamente casuale; infatti la generazione avviene in effetti in modo pseudo-casuale e secondo un meccanismo che dipende dallo stato interno del computer codificato in una variabile indicata con `.Random.seed`. Se il *seme* iniziale è lo stesso i numeri generati saranno uguali. Spesso conviene che i calcoli (ad esempio a fine didattico) siano riproducibili. Ad esempio mettendo in una variabile `seme` il valore corrente di `.Random.seed` e richiamandolo o generandolo all'occorrenza. Un altro modo di procedere consiste nell'impostare il valore di `.Random.seed` attraverso il comando `set.seed` la cui sintassi è `set.seed`(n) dove n è un numero intero.

```
> set.seed(3)
```

A questo punto possiamo simulare le variabili richieste usando la struttura

$$\texttt{sample}(\textit{alfabeto}, n) \tag{3.1}$$

Se l'alfabeto consiste di tutte le lettere minuscole dell'alfabeto ordinario e ne vogliamo selezionare $n = 8$ (in modo che ciascun uscita abbia la stessa probabilità) basta scrivere

```
> sample(letters,8)
[1] "e" "u" "j" "h" "n" "m" "c" "f"
```

Se invece l'alfabeto consiste delle basi del DNA

```
> alfabeto=c("A","C","G","T")
> sample(alfabeto,2)
[1] "G" "C"
```

Notiamo che

```
>  sample(alfabeto)
[1] "G" "C" "T" "A"
```

restituisce una permutazione dell'alfabeto, mentre chiedendo un campione di lunghezza superiore alla lunghezza dell'alfabeto otteniamo un messaggio di errore. Possiamo però immaginare di re-immettere la lettera estratta nell'urna dopo ogni estrazione. In questo caso non c'è limite alla sequenza generata. Per esempio

```
> alfabeto=c("testa","croce")
> sample(alfabeto,5,replace=T)
[1] "croce" "croce" "testa" "croce" "croce"
```

Il precursore del dado era chiamato astragalo ed era giocato nell'antica Grecia e nell'antica Roma [**?**]. Gli astragali sono dei piccoli ossicini di forma irregolare ed hanno 6 facce ma atterranno in modo stabile solo su 4 di esse numerate 1, 3, 4 e 6 con probabilità all'incirca 0.4 per il 3 e il 4 e di 0.1 per l'1 e il 6. In altre parole l'astragalo è descritto dalla tabella

valore	probabilità
1	0.1
3	0.4
4	0.4
6	0.1

Il tiro più gettonato all'epoca era l'uscita di 4 facce diverse nel lancio di 4 astragali e si chiamava *Venus*. Il lancio considerato peggiore sul singolo lancio era l'1 chiamato cane o avvoltoio. Per simulare un astragalo su un computer

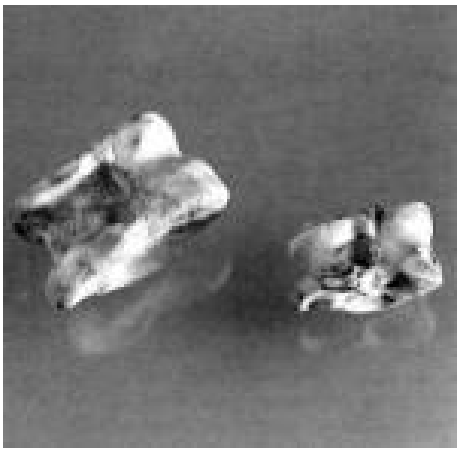


Figura 3.1: Astragalo.

```
> sample(c(1,3,4,6),4,replace=T,prob=c(0.1,0.4,0.4,0.1))
```

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

Torniamo ora ai classici dadi a 6 facce. Supponiamo di lanciare 100 volte un dado equo a 6 facce e di registrare in `x` le uscite rilevate

```
> set.seed(3)
> dadi100<-sample(1:6,100,replace=T)
> dadi100
```

```
  [1] 2 5 3 2 4 4 1 2 4 4 4 4 4 4 6 5 1 5 6 2 2 1 1 1 2 5 4 6
 [29] 4 5 3 3 2 3 2 3 6 2 4 2 2 5 2 4 3 2 1 1 2 5 2 2 6 6 6 6
 [57] 3 2 1 2 5 1 5 1 5 2 5 4 3 1 5 5 6 6 4 4 1 1 5 5 5 4 3 1
 [85] 6 6 2 3 4 6 1 2 3 5 6 2 2 2 2 5
```

Volendo invece simulare una combinazione da giocare al SuperEnalotto possiamo scrivere

```
> (x<-sample(1:90,6,replace=T))
```

```
[1] 69 62 19 65 55 31
```

I numeri usciti sono stati salvati in una variabile `x`, per poter effettuare la ricerca di indicatori statistici. Il comando che consente di ordinare una lista o un vettore è `sort`, esso può essere usato in associazione al nome di una variabile o di una lista, ossia:

$$\texttt{sort}(\textit{variabile/lista}) \tag{3.2}$$

Volendo ordinare i numeri precedentemente ricavati scriveremo

```
> sort(x)
[1] 19 31 55 62 65 69
```

3.3 Statistica descrittiva: singola variabile

3.3.1 Indicatori statistici

- Media.

  La media di una serie di numeri si ottiene con la funzione `mean` scrivendo: `mean`(*variabile*). Ad esempio, lavorando con la lunghezza del sepalo di 150 piante di iris

  ```
  > x=iris[,1]
  > mean(x)
  [1] 5.843333
  ```

- Varianza campionaria

  Si ottiene con la funzione predefinita di espressione: `var`(*variabile*). Possiamo calcolare la varianza come

  ```
  > var(x)
  [1] 0.6856935
  ```

- Deviazione Standard campionaria.

  Non è altro che la radice della varianza. Si ottiene con la funzione predefinita di espressione: `sd`(*variabile*). Sempre basandosi sull'esempio precedente scriveremo

  ```
  > sd(x)
  ```

```
[1] 0.8280661
```

- Quantili. La notazione standard è semplicemente: `quantile`(*variabile*) che determina i quartili e ci fornisce in uscita la statistica dei 5 numeri

```
> quantile(x)
```

```
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

Volendo ricavare i decili dovremo scrivere:

$$\texttt{quantile}(\textit{variabile}, \texttt{seq(0,1,by=0.1)})$$

in quanto vogliamo dividere l'intervallo $[0, 1]$ a passo 0.1

Nell'esempio:

```
> quantile(x,seq(0,1,by=0.1))
```

```
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
4.30 4.80 5.00 5.27 5.60 5.80 6.10 6.30 6.52 6.90 7.90
```

Si noti che `quantile` ammette 9 varianti specificabili con l'opzione `type` $= n$ dove n va da 1 a 9. Per esempio

```
> quantile(x,type=2)
```

```
  0%  25%  50%  75% 100%
 4.3  5.1  5.8  6.4  7.9
```

Sui dati in esame le 9 varianti coincidono. La convenzione da noi adottata corrisponde al numero 2

Per quanto riguarda gli indicatori statistici nel caso di dati ripetuti basta notare che se la lista x contiene i valori e la lista f le frequenze assolute il comando

$$\textit{rep}(x, f)$$

costruisce un'unica lista dei dati inclusiva delle ripetizioni. Per esempio

```
> x=1:6
> f=c(9,7,9,7,8,10)
> (dati=rep(x,f))
```

```
 [1] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 4 4 4 4 4
[31] 4 4 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6
```

Ovviamente senza bisogno di visualizzare `dati` possiamo calcolarne tutti gli indicatori statistici. Il comando

```
> cumsum(f)
```

```
[1]  9 16 25 32 40 50
```

restituisce le frequenze cumulate, dalle quali si possono ricavare facilmente la mediana i quantili.

3.3.2 Raggruppamenti in classi

Consideriamo la rilevazione della temperatura media giornaliera di Milano nel mese di Gennaio 2016. Scegliamo il mese

```
> stringa="Milano/2016/Gennaio?format=csv"
```

```
> sito="http://www.ilmeteo.it/portale/archivio-meteo/"
> indirizzo=paste(sito,stringa,sep="")
> meteo=read.table(indirizzo,sep=";",header=T)
```

```
> meteo
```

```
   LOCALITA      DATA TMEDIA..C TMIN..C TMAX..C PUNTORUGIADA..C
1    Milano  1/1/2016         1      -2       4               1
2    Milano  2/1/2016         1       0       2               1
3    Milano  3/1/2016         1       0       3               1
4    Milano  4/1/2016         2       1       3               1
5    Milano  5/1/2016         3       2       5               2
6    Milano  6/1/2016         5       3       8               2
7    Milano  7/1/2016         3      -1       6               2
8    Milano  8/1/2016         2      -1       5               2
9    Milano  9/1/2016         5       3       5               4
10   Milano 10/1/2016         5       4       7               5
11   Milano 11/1/2016         6       4       8               6
12   Milano 12/1/2016         5       0      12               3
13   Milano 13/1/2016         7       1      12               0
14   Milano 14/1/2016         2      -2       5               1
```

```
15   Milano 15/1/2016         5       1      11               1
16   Milano 16/1/2016         5      -3      11               0
17   Milano 17/1/2016         6       2       8               0
18   Milano 18/1/2016         0      -5       5               0
19   Milano 19/1/2016        -1      -5       4               0
20   Milano 20/1/2016         0      -4       4               0
21   Milano 21/1/2016         0      -5       5               0
22   Milano 22/1/2016         0      -5       6               0
23   Milano 23/1/2016         2      -3       8               0
24   Milano 24/1/2016         2      -4       8               0
25   Milano 25/1/2016         4      -2      13               2
26   Milano 26/1/2016         7      -1      15               5
27   Milano 27/1/2016         7       1      12               5
28   Milano 28/1/2016         9       7      12               6
29   Milano 29/1/2016        10       6      15               6
30   Milano 30/1/2016         8       7       9               6
31   Milano 31/1/2016         8       4      13               7
   UMIDITA.. VISIBILITA.km VENTOMEDIA.km.h VENTOMAX.km.h
1         97             2               6            11
2         97             2               5             9
3         96             3               7            11
4         93             4               7            11
5         89             5               6            11
6         85             5               8            13
7         88             8               5            11
8         89             7               7            17
9         95             2               6            11
10        95             3               5            11
11        95             3               8            13
12        80             6               7            20
13        46            31              11            19
14        74            10               7            15
15        54             9              11            22
16        43            13              10            24
17        22            18              17            28
18        40            26               4            11
19        55            14               4             9
20        63            10               6            13
21        69             7               6            11
22        75             7               4             7
23        73             6               6            15
24        75             5               6            11
25        77             5               5            13
26        81             5               5            11
27        85             4               4             7
```

```
28        84             4                5             9
29        81             5                5            11
30        91             2                4             9
31        92             2                8            17
   RAFFICA.km.h PRESSIONESLM.mb PRESSIONEMEDIA.mb PIOGGIA.mm
1             0            1026                 0          0
2             0            1019                 0          0
3             0            1010                 0          0
4             0            1000                 0          0
5             0            1001                 0          0
6             0            1001                 0          0
7             0            1004                 0          0
8             0            1009                 0          0
9             0            1008                 0          0
10            0            1004                 0          0
11            0             997                 0          0
12            0            1002                 0          0
13            0            1014                 0          0
14            0            1014                 0          0
15           35            1010                 0          0
16            0            1014                 0          0
17           44            1017                 0          0
18            0            1020                 0          0
19            0            1017                 0          0
20            0            1018                 0          0
21            0            1023                 0          0
22            0            1032                 0          0
23            0            1033                 0          0
24            0            1034                 0          0
25            0            1032                 0          0
26            0            1030                 0          0
27            0            1029                 0          0
28            0            1028                 0          0
29            0            1030                 0          0
30            0            1027                 0          0
31            0            1016                 0          0
              FENOMENI
1               nebbia
2  pioggia neve nebbia
3          neve nebbia
4         pioggia neve
5              pioggia
6               nebbia
7               nebbia
8               nebbia
```

```
9              pioggia
10      pioggia nebbia
11      pioggia nebbia
12               nebbia
13
14
15
16
17
18
19
20
21
22
23
24
25               nebbia
26               nebbia
27               nebbia
28             pioggia
29      pioggia nebbia
30      pioggia nebbia
31               nebbia
```

A questo punto selezioniamo la colonna della temperatura media

```
> meteo[,3]->Milano;
> Milano
 [1]  1  1  1  2  3  5  3  2  5  5  6  5  7  2  5  5  6  0 -1  0
[21]  0  0  2  2  4  7  7  9 10  8  8
```

```
> quantile(Milano)
  0%  25%  50%  75% 100%
-1.0  1.5  4.0  6.0 10.0
```

L'ultimo comando in particolare ci fornisce minimo e massimo dei dati. Possiamo esaminare la serie temporale dei dati con i comandi

```
> plot(Milano,type="l",xlab=paste(m,anno, "a milano"),ylab="temperatura media")
```

ottenendo la figura 3.2

Raggruppiamo ora i dati in classi comprese tra due estremi che comprendano certamente tutti i dati, per esempio -2 e 10, decidendo di applicare un passo di 2 e vedere come si distribuiscono. Il comando `cut` associa a ciascun dato la classe di appartenenza selezionata in base ai punti di taglio.

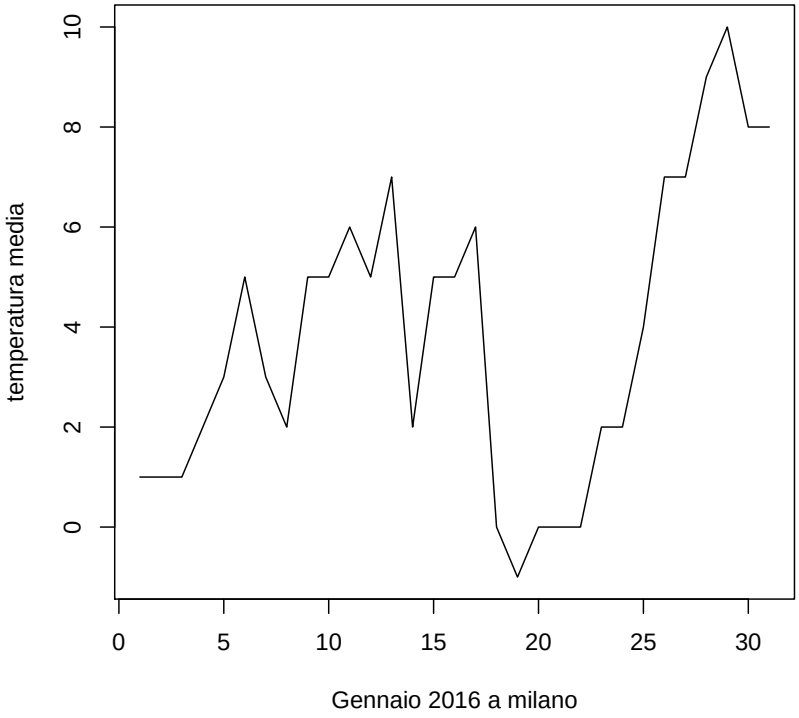


Figura 3.2: Andamento della temperatura a Gennaio 2016 a Milano.

```
> tagli=c(-2,0,2,4,6,10)
> cut(Milano,breaks=tagli)
```

```
 [1] (0,2]  (0,2]  (0,2]  (0,2]  (2,4]  (4,6]  (2,4]  (0,2]
 [9] (4,6]  (4,6]  (4,6]  (4,6]  (6,10] (0,2]  (4,6]  (4,6]
[17] (4,6]  (-2,0] (-2,0] (-2,0] (-2,0] (-2,0] (0,2]  (0,2]
[25] (2,4]  (6,10] (6,10] (6,10] (6,10] (6,10] (6,10]
Levels: (-2,0] (0,2] (2,4] (4,6] (6,10]
```

Il comando `table` conta i dati di ciascuna classe

```
> table(cut(Milano,breaks=tagli))
```

```
(-2,0]  (0,2]  (2,4]  (4,6] (6,10]
     5      8      3      8      7
```

Si noti che la suddivisione in classi prevede intervalli aperti a sinistra e chiusi a destra. Per suddividere in modo che gli intervalli siano chiusi a sinistra e aperti a destra si specifica il parametro `right=FALSE`. Possiamo anche usare il comando `seq` per specificare i tagli.

$$\texttt{table(cut(}\textit{variabile}\texttt{, breaks=seq(}\textit{estremo inf}\texttt{,}$$
$$\textit{estremo sup}\texttt{, by} = \textit{passo}\texttt{), right=TRUE))}$$

o in modo più generale

$$\texttt{table(cut(}\ \textit{variabile}\texttt{,}$$
$$\texttt{breaks=c(}\textit{estremo}\ \ \textit{inferiore}\texttt{,}\ldots\texttt{,}\textit{estremo superiore}\texttt{))}$$

estremamente utile in quanto consente di raggruppare i dati in classi non necessariamente di ugual ampiezza.

```
> table(cut(Milano,breaks=c(-3,1,3,4,5,6,8,10)))
```

```
(-3,1]  (1,3]  (3,4]  (4,5]  (5,6]  (6,8] (8,10]
     8      7      1      6      2      5      2
```

Volendo raggruppare in classi i dati delle precedenti uscite del dado possiamo scrivere

```
> table(cut(dadi100,breaks=0:6))
```

```
(0,1] (1,2] (2,3] (3,4] (4,5] (5,6]
   15    25    11    17    18    14
```

Se scegliamo di chiudere a sinistra gli intervalli dobbiamo però includere il 7 altrimenti il valore 6 non risutlerebbe incluso.

```
> table(cut(dadi100,breaks= 1:7,right=FALSE))
```

```
[1,2) [2,3) [3,4) [4,5) [5,6) [6,7)
   15    25    11    17    18    14
```

3.3.3 Areogrammi

Il comando generico per generare un istogramma è:

$$\texttt{hist}(\textit{variabile})$$

che segue però la struttura del comando `cut`. L'ampiezza di ciascuna classe salvo diversamente indicato è costante e decisa da R. È possibile variare tale condizione definendo una lista con i punti di taglio (*cutoff*) delle classi volute:

$$\texttt{hist}(\textit{variabile}, \texttt{c}(\textit{valore}_1, \textit{valore}_2, \ldots)) \tag{3.3}$$

Per esempio se `dadi100` rappresenta le solite 100 uscite del lancio del dado, il comando

```
> par(mfrow=c(1,2))
> hist(dadi100,breaks=seq(0.5,6.5,1),col="red")
> hist(dadi100,freq=FALSE,breaks=seq(0.5,6.5,1),col="blue")
```

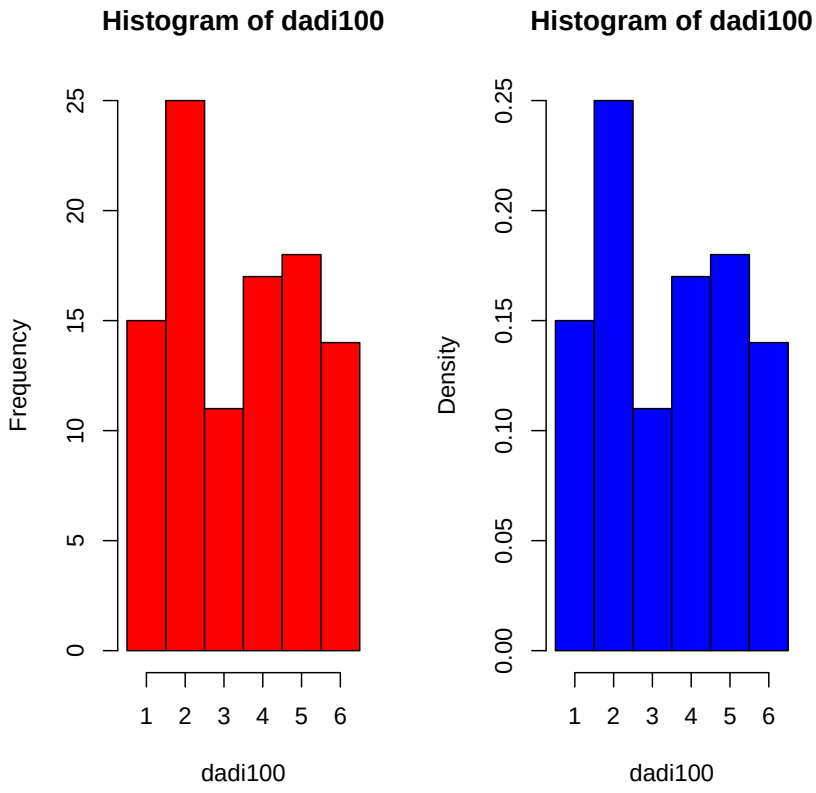


Figura 3.3: Diagramma a colonne e areogramma per il lancio di un dado.

genera l'istogramma (in rosso, a sinistra Figura 3.2) con le frequenze assolute delle classi in ordinata. La sequenza dei punti di taglio è stata scelta in modo che i numeri interi da 1 a 6 siano al centro delle classi corrispondenti. Se invece volessimo creare un areogramma (ossia avere un tracciato per cui le aree siano pari alle frequenze relative) a partire dalle stesse uscite dovremo imporre il parametro `freq=FALSE` otterremo il pannello a destra (in blu) della figura (3.2). Avendo scelto classi di ampiezza costante i 2 grafici differiscono semplicemente per un cambio di scala sull'asse y.

In modo simile possiamo tracciare un areogramma dei dati nella variabile `milano`

```
> par(mfrow=c(1,2))
> hist(Milano, col="green",freq=FALSE,right=FALSE,
+ main="Cutoff automatici")
```

lasciando R libero di scegliere i punti di taglio (pannelli a sinistra della figura 3.4) o scegliendoli a nostra volta (pannelli a destra della stessa figura 3.4)

```
> hist(Milano,col="red",freq=FALSE,
+ breaks=unique(as.vector(quantile(Milano,seq(0,1,by=1/6)))),
+ main="Cutoff personalizzati")
```

Si noti la stabilità degli areogrammi rispetto ai cambi nella suddivisione.

3.3.4 Generazione di boxplot

Il `boxplot` è una rappresentazione grafica immediata della statistica dei 5 numeri e simultaneamente ci segnala eventuali punti discordanti o anomali, *outlier*. Il comando generico è:

$$\texttt{boxplot}(\textit{variabile}) \tag{3.4}$$

prendendo il vettore x contenente i risultati di 100 lanci otteniamo la figura 3.5 da cui si evince che il valore massimo dei dati è 6, il minimo è 1 e non ci sono punti anomali, per cui non vi sono dati anomali, altrimenti evidenziati da un pallino. Si legge inoltre il valore di mediana (4) primo quartile (2) e terzo quartile (5).

3.3.5 Creazione di grafici a torta

Il comando `pie` consente, partendo da una tabella, di tracciare il diagramma a torta per una variabile nominale raggruppata in classi. Il comando è

$$\texttt{pie(table(}\textit{variabile}\texttt{))}$$

ad esempio (facendo riferimento ai precedenti dati):

```
> pie(table(dadi100))
```

fornisce in uscita la Figura 3.6

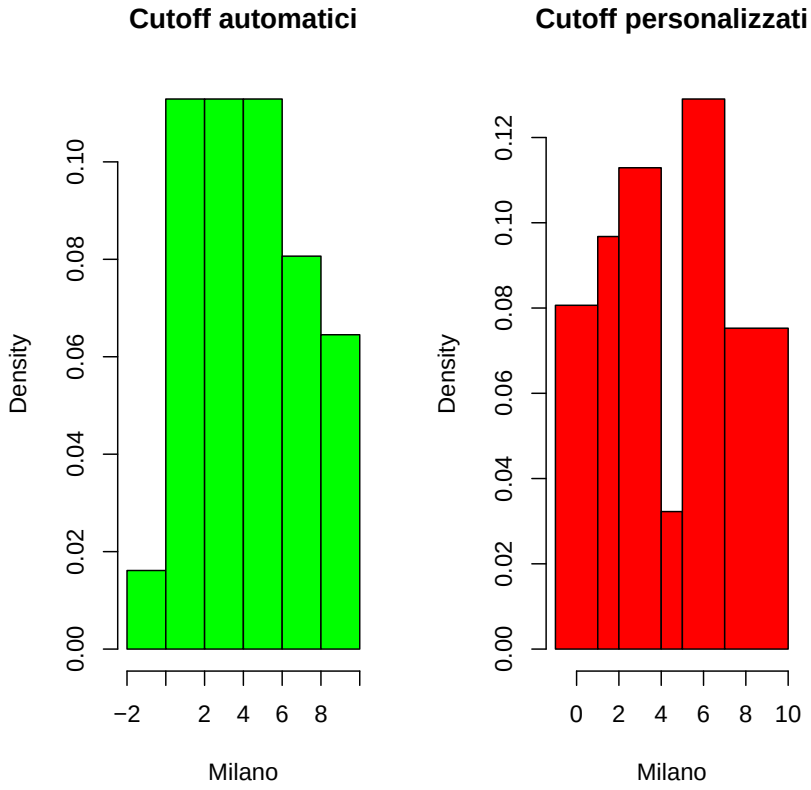


Figura 3.4: Areogramma dei dati della temperatura. Scelta automatica dei punti di taglio.

```
> boxplot(dadi100)
```

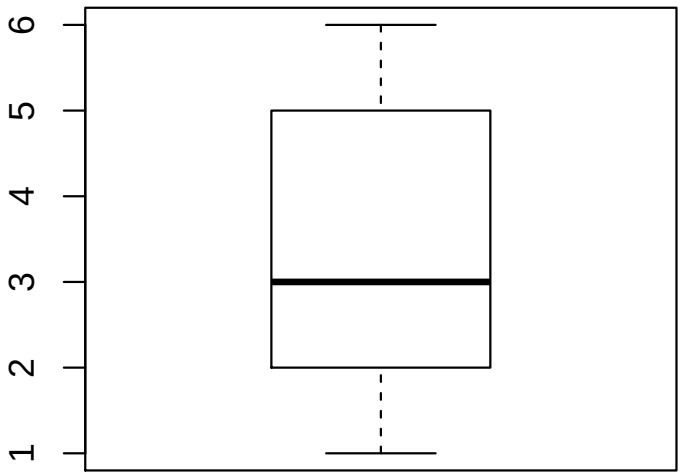


Figura 3.5: Boxplot dei risultati del lancio di un dado

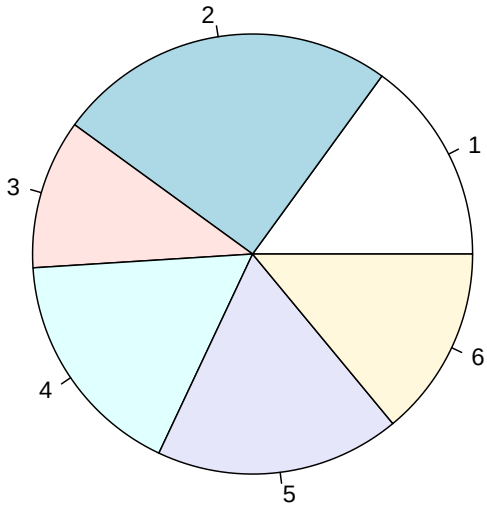


Figura 3.6: Diagramma a torta per il lancio di un dado equo.

Costruire una matrice contenente le coordinate di 50 punti nel rettangolo $[0,4] \times [0,2]$ in due dimensioni (generate utilizzando il generatore di numeri pseudocasuali). Produrre un grafico con due pannelli, dove il primo pannello è uno scatter-plot

3.4 Variabili doppie e rette di regressione

Supponiamo di misurare la concentrazione di acido lattico muscolare durante uno sforzo di 10 minuti,

```
> x<-tempo<-c(1,2,3,4,5,6,7,8,9,10)
> y<-concentrazione<-c(0.3,0.65,0.7,0.8,0.95,1.05,1.3,1.7,1.9,
+ 2.5)
```

Per analizzare questi dati conviene preliminarmente tracciarne un diagramma a dispersione. Possiamo inoltre determinare il coefficiente di correlazione lineare

```
> cor(x,y)
```

```
[1] 0.9620456
```

Per definire un modello di relazione lineare occorre usare il comando `lm` (*linear model*). Nella sua generica forma il comando è espresso come[1]

$$\texttt{lm}(y \sim x)$$

Otteniamo i valori di pendenza e intercetta.

Possiamo tracciare la retta di regressione con il comando `abline`.

```
> plot(x,y,pch=19,col="red")
> abline(lm(y~x),col="blue")
```

Per determinare la retta di regressione sulle y dobbiamo invertire x e y.

```
> (modellox=lm(x~y))
```

```
Call:
lm(formula = x ~ y)

Coefficients:
(Intercept)            y
     0.3516       4.3446
```

[1] Per digitare la tilde $\sim$ su Mac premere ALT 5 su PC invece il tasto Alt Gr (attivazione del codice ASCII) e sul tastierino numerico digitare il numero 126. Lavorando su un portatile il tastierino numerico è spesso incorporato nella tastiera con colorazione blu dei tasti.

```
> plot(x,y)
```

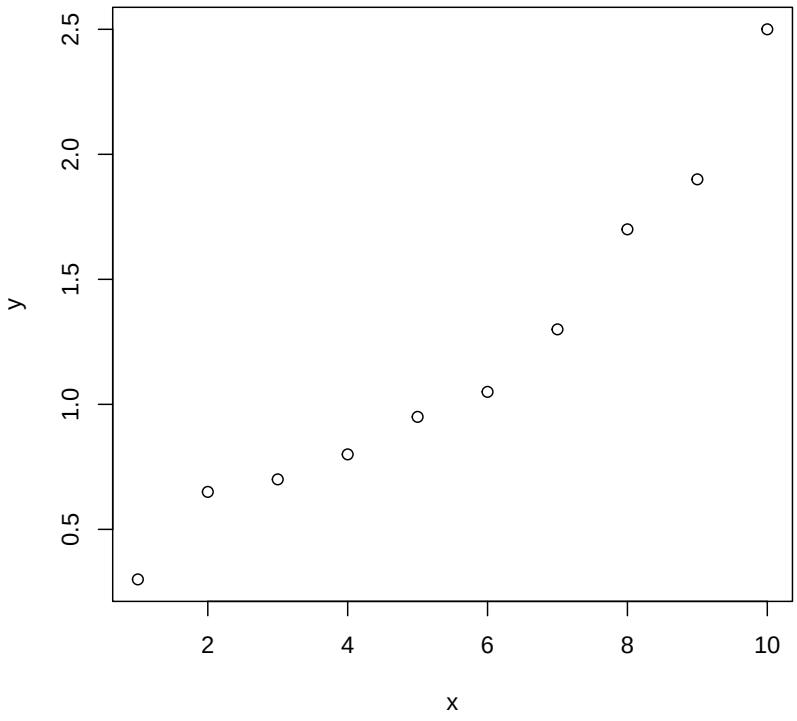


Figura 3.7: Diagramma a dispersione tempo/concentrazione.

```
> coeff=modellox$coefficients
> (a=1/coeff[2])
```

```
        y
0.2301707
```

```
> (b=-coeff[1]/coeff[2])
```

```
(Intercept)
-0.08093883
```

```
> abline(b,a,col="green")
```

In tal modo otteniamo il grafico 3.8.

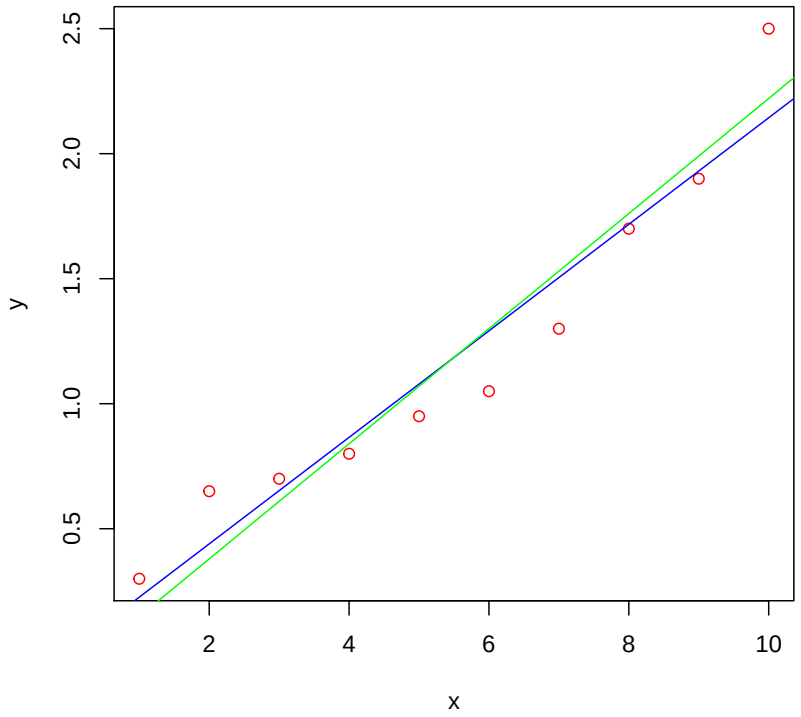


Figura 3.8: Rette di regressione. In blu R_x, in verde R_y.

I bambini di Kalama (Egitto). Ancora retta di regressione

Da DASL [2] possiamo scaricare un *dataset* in cui i ricercatori hanno misurato le altezze (cm) dai 18 ai 29 mesi di vita, di 161 bambini di Kalama, un villaggio egiziano. Le altezze sono state mediate tra i bambini per fornire un singolo valore mese per mese.

```
> age=18:29
> height=c(76.1,77,78.1,78.2,78.8,79.7,79.9,81.1,81.2,81.8,82.8,83.5)
```

Possiamo quindi costruire il `data.frame`

```
> village=data.frame(age=age,height=height)
```

Ora diamo una prima occhiata ai dati: L'andamento è lineare. Determiniamo la retta di

```
> plot(age,height)
```

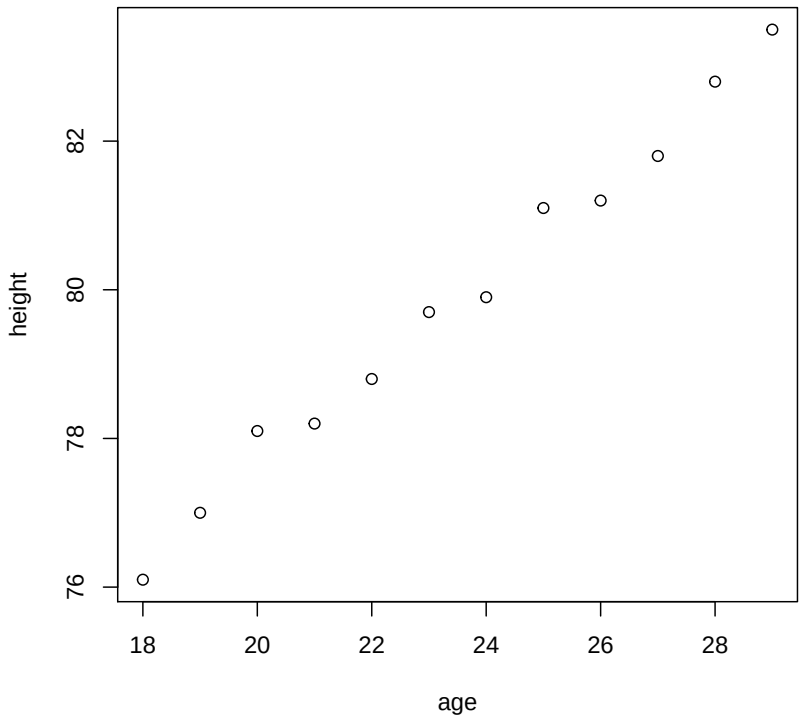


Figura 3.9: Crescita dei bambini di Kalama

regressione per predire l'altezza media nota l'età in mesi.

```
> (modello=lm(height~age))
```

```
Call:
lm(formula = height ~ age)

Coefficients:
```

```
(Intercept)          age
     64.928        0.635
```

La retta di regressione cercata ha formula:

$$h(\texttt{age}) = 64.93 + 0.63\,\texttt{age}$$

Possiamo ora utilizzare R come semplice calcolatore per predire l'altezza a 27.5 mesi di età: oppure, è più efficiente utilizzare direttamente il *dataframe* e la funzione `predict`:

```
> predict(modello,data.frame(age=27.5))
```

```
       1
82.38986
```

fornendo in input i parametri della retta ed un preciso valore della variabile indipendente (richiamata col proprio nome). Molti comandi di R sono in grado di manipolare *dataframe* lavorando direttamente sulla struttura. Per esempio, il comando plot di un `dataframe` in due colonne, esegue in automatico il grafico della seconda colonna (variabile dipendente) vs prima colonna (variabile indipendente). Possiamo ottenere il modello lineare visto nel caso precedente, passando `village` direttamente al comando:

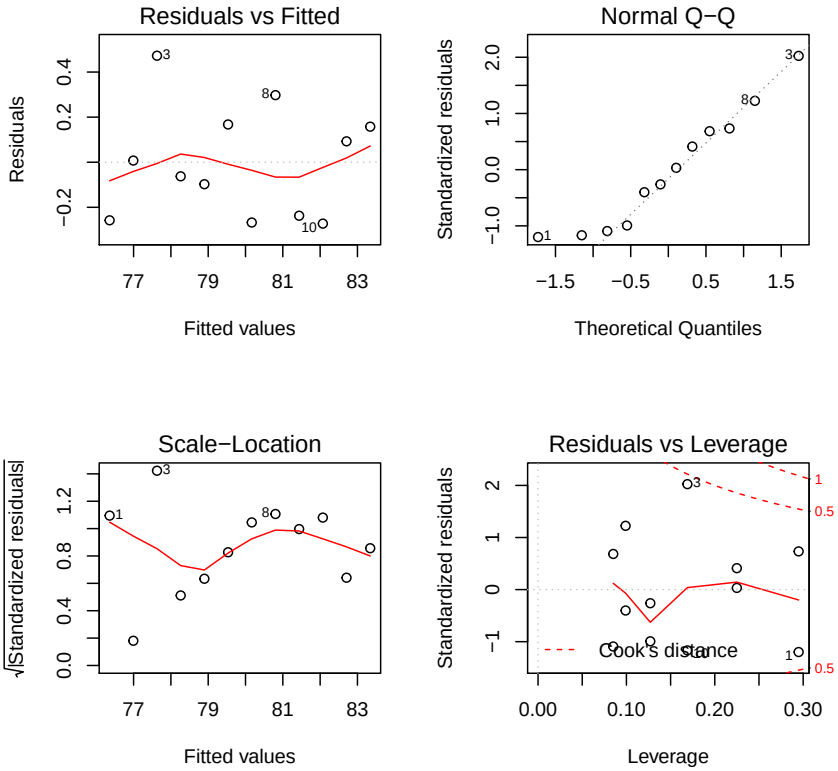
```
> modello=lm(height~age,data=village)
> modello
```

```
Call:
lm(formula = height ~ age, data = village)

Coefficients:
(Intercept)          age
     64.928        0.635
```

con la formula $lm(\tilde{y}\tilde{x}, data = dataset)$. Inoltre possiammo considerare il plot di un oggetto `lm` che fornisce una serie di rappresentazioni grafiche

```
$mfrow
[1] 1 1
```



3.5 Modelli potenza

Consideriamo ora il seguente *dataset* di mammiferi in cui le 2 variabili rappresentano le dimensioni del corpo e del cervello.

```
> library(MASS)
> mammals
```

	body	brain
Arctic fox	3.385	44.50
Owl monkey	0.480	15.50
Mountain beaver	1.350	8.10
Cow	465.000	423.00
Grey wolf	36.330	119.50
Goat	27.660	115.00
Roe deer	14.830	98.20
Guinea pig	1.040	5.50
Verbet	4.190	58.00
Chinchilla	0.425	6.40
Ground squirrel	0.101	4.00
Arctic ground squirrel	0.920	5.70

African giant pouched rat	1.000	6.60
Lesser short-tailed shrew	0.005	0.14
Star-nosed mole	0.060	1.00
Nine-banded armadillo	3.500	10.80
Tree hyrax	2.000	12.30
N.A. opossum	1.700	6.30
Asian elephant	2547.000	4603.00
Big brown bat	0.023	0.30
Donkey	187.100	419.00
Horse	521.000	655.00
European hedgehog	0.785	3.50
Patas monkey	10.000	115.00
Cat	3.300	25.60
Galago	0.200	5.00
Genet	1.410	17.50
Giraffe	529.000	680.00
Gorilla	207.000	406.00
Grey seal	85.000	325.00
Rock hyrax-a	0.750	12.30
Human	62.000	1320.00
African elephant	6654.000	5712.00
Water opossum	3.500	3.90
Rhesus monkey	6.800	179.00
Kangaroo	35.000	56.00
Yellow-bellied marmot	4.050	17.00
Golden hamster	0.120	1.00
Mouse	0.023	0.40
Little brown bat	0.010	0.25
Slow loris	1.400	12.50
Okapi	250.000	490.00
Rabbit	2.500	12.10
Sheep	55.500	175.00
Jaguar	100.000	157.00
Chimpanzee	52.160	440.00
Baboon	10.550	179.50
Desert hedgehog	0.550	2.40
Giant armadillo	60.000	81.00
Rock hyrax-b	3.600	21.00
Raccoon	4.288	39.20
Rat	0.280	1.90
E. American mole	0.075	1.20
Mole rat	0.122	3.00
Musk shrew	0.048	0.33
Pig	192.000	180.00
Echidna	3.000	25.00

Brazilian tapir	160.000	169.00
Tenrec	0.900	2.60
Phalanger	1.620	11.40
Tree shrew	0.104	2.50
Red fox	4.235	50.40

Per prima cosa tracciamo il grafico dei punti in scala non trasformata e, visto la compresenza di dati molto prossimi all'origine e di dati molto distanti in scala logaritmica (sia le x che le y vengono trasformate prendendone i logaritmi)

```
> par(mfrow=c(1,2))
> plot(mammals)
> plot(mammals,log="xy")
```

come in Figura 3.10. Visti i risultati ottenuti usando la scala logaritmica tracciamo anche

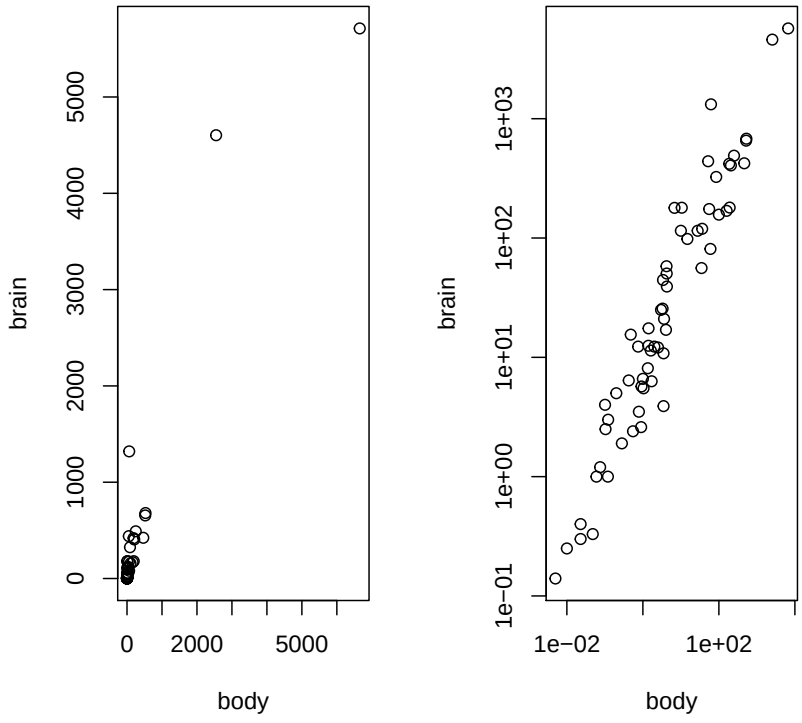


Figura 3.10: Diagramma a dispersione massa corporea/massa del cervello in scala normale ed in scala logaritmica.

la corrispondente retta di regressione

```
> plot(log(mammals$brain)~log(mammals$body),col="BLUE",pch=19,type="p")
> abline(lm(log(mammals$brain)~ log(mammals$body)),col="red",lwd=3);
> uomo=which(rownames(mammals)=="Human")
> text(log(mammals[uomo ,1]),log(mammals[uomo ,2]),rownames(mammals)[uomo])
```

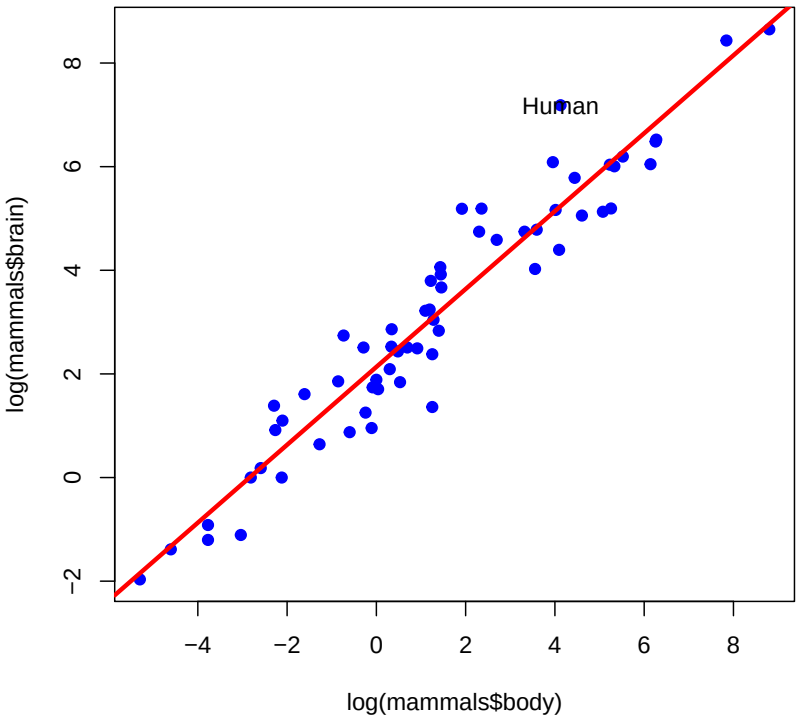


Figura 3.11: Retta di regressione. Dimensione del corpo e del cervello. Si noti la posizione dell'uomo.

Si noti il comando `text(x,y, testo)` dove *x* e *y* e *testo* sono vettori di arbitraria lunghezza contenenti ascisse, ordinate e testo da inserire.

3.6 Distribuzioni in R

I nomi delle principali distribuzioni in R sono

`norm`	normale
`t`	Student
`chisq`	chi quadro
`f`	Fisher
`binom`	binomiale

A questi nomi possiamo aggiungere diversi prefissi

`d`	densità
`p`	primitiva
`q`	quantile
`r`	random

per caratterizzare diversi aspetti.

3.6.1 Distribuzione normale

La funzione `dnorm`

Come appena visto R indica con il nome `dnorm`, la densità normale o gaussiana. Essa accetta come parametri sia la media μ che la deviazione standard σ come è possibile verificare con il comando `formals` che ci fornisce gli argomenti di una funzione e gli eventuali valori preassegnati.

```
> formals(dnorm)
```

```
$x


$mean
[1] 0

$sd
[1] 1

$log
[1] FALSE
```

Se i parametri sono omessi `dnorm` rappresenta la densità normale standard con $\mu = 0$ e $\sigma = 1$. Il grafico (3.12) della gaussiana tra due estremi, ad esempio -2.5 e 2.5 si ottiene con il solito comando

```
> curve(dnorm,-2.5,2.5)
```

Per visualizzare una gaussiana non standard, ad esempio una gaussiana con media $\mu = 1$ e deviazione standard $\sigma = 1.5$, tra -3 e 3. scriveremo invece

```
> curve(dnorm(x,mean=1,sd=1.5),-3,3)
```

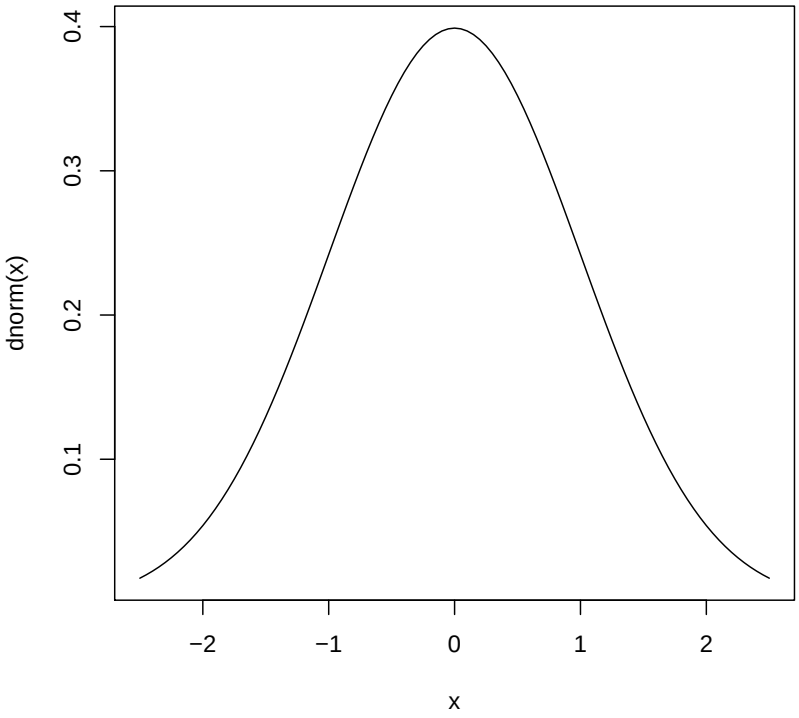


Figura 3.12: Grafico della normale standard nell'intervallo $[-2.5, 2.5]$.

3.6.2 La funzione `pnorm`

La funzione `pnorm`(x) è la antiderivata di `dnorm` calcolata come segue

$$\texttt{pnorm}(x) = \int_{-\infty}^{x} \texttt{dnorm}(s)ds$$

Ovviamente

$$\int_{a}^{b} \texttt{dnorm}(x)dx = \texttt{pnorm}(b) - \texttt{pnorm}(a)$$

e per avere l'area sottesa tra 3 e 5 basta scrivere:

```
> pnorm(5)-pnorm(3)
```

```
[1] 0.001349611
```

Per ottenere il valore dell'area tra 0 e x bisogna allora sottrarre `pnorm`(0)=0.5 all'area fornita dalla funzione. Per cui possiamo scrivere:

```
> pnorm(1)-0.5
```

```
[1] 0.3413447
```

3.6.3 La funzione `qnorm` e la tabella della densità di Gauss

La funzione `qnorm` rappresenta la funzione inversa di `pnorm`.

$$\texttt{qnorm}(A) = x \Leftrightarrow A = \int_{-\infty}^{x} \texttt{dnorm}(s)ds$$

come illustrato nella figura 3.13.

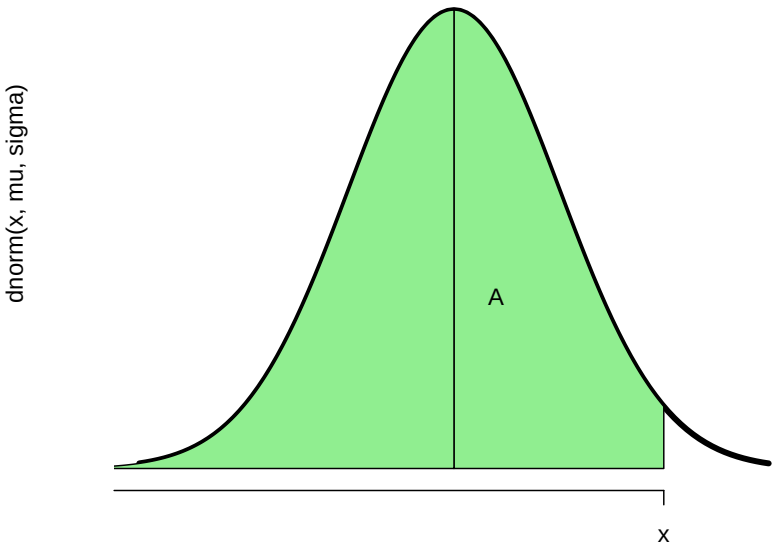


Figura 3.13: $x = \texttt{qnorm}(A)$

Vogliamo costruire una funzione, diciamo U tale che assegnato un valore di area A fornisca l'ascissa $x = U(A)$ come in figura 3.14 in modo che l'area tra - x e x sia esattamente pari ad A.

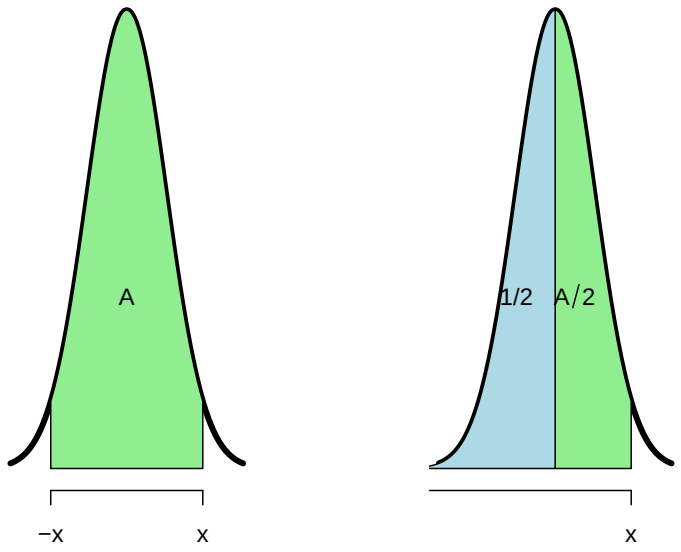


Figura 3.14: $x = U(A) = \texttt{qnorm}(1/2 + A/2)$

Dalla stessa figura si evince che la funzione che riproduce la tabella è

```
> U <-function (A) qnorm (1/2 + A/2)
```

Questa funzione fornisce fissato il livello di fiducia l'ascissa x tale che l'intervallo simmetrico $[-x, x]$ racchiuda un'area pari al lvello di fiducia. Per esempio

```
> U(0.95)
[1] 1.959964
```

3.6.4 La funzione `rnorm`

È possibile generare dei valori standardizzati casuali (media uguale a 0, deviazione standard pari a 1) che seguono la distribuzione normale standard. Basta semplicemente definire il numero di valori desiderati. Il comando nella sua espressione generale è:

$$\texttt{rnorm}(n, \texttt{mean} = \textit{valore}_1, \texttt{sd} = \textit{valore}_2) \tag{3.5}$$

Nel caso in cui volessimo una lista di 20 valori di una variabile normale con media assegnata 5 e deviazione standard 1 scriveremo :

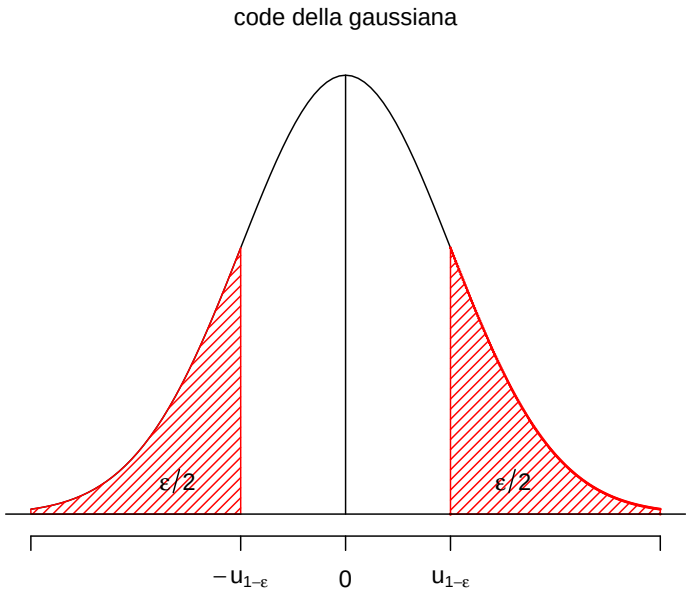


Figura 3.15: Code della distribuzione normale

```
> rnorm(20,mean=5,sd=1)
```

3.6.5 La distribuzione t di Student

In R la distribuzione di Student è indicata con la lettera `t`. Come per le altre densità si possono considerare le funzioni

dt	densità
pt	primitiva
qt	quantili
rt	generatore random

Il grafico della distribuzione di Student ad un certo numero `df` di gradi di libertà si ottiene con il comando

`curve(dt(x, df)`, a, b)

Tracciamo ad esempio un grafico tra -2 e 2 per una distribuzione a 10 gradi di libertà (vedi figura (3.16)):

```
> curve(dt(x,10),-2,2)
```

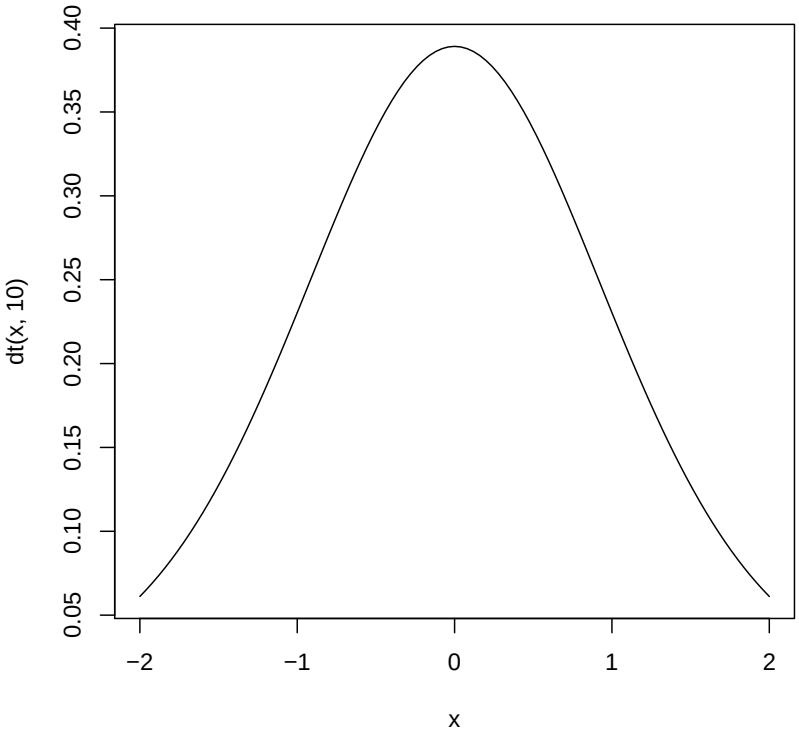


Figura 3.16: Grafico della distribuzione di Student a 10 gradi di libertà.

Ricordiamo che la distribuzione di Student si usa in particolare nei casi in cui la deviazione standard della popolazione σ non è conosciuta e viene rimpiazzata dalla deviazione standard campionaria S, calcolata con un numero N di dati e quindi con $N-1$ gradi di libertà. Quando però il numero di dati si avvicina a 30 la curva di Student è praticamente sovrapposta a quella della distribuzione normale, come mostra il grafico (3.17):

```
> curve(dnorm(x),-2,2,col=3)
> curve(dt(x,2),-2,2,col=1,add=T)
> curve(dt(x,25),-2,2,col=2,add=T)
> legend("topleft", c("df=2","df=25","normale"),pch=15,col=1:3);
```

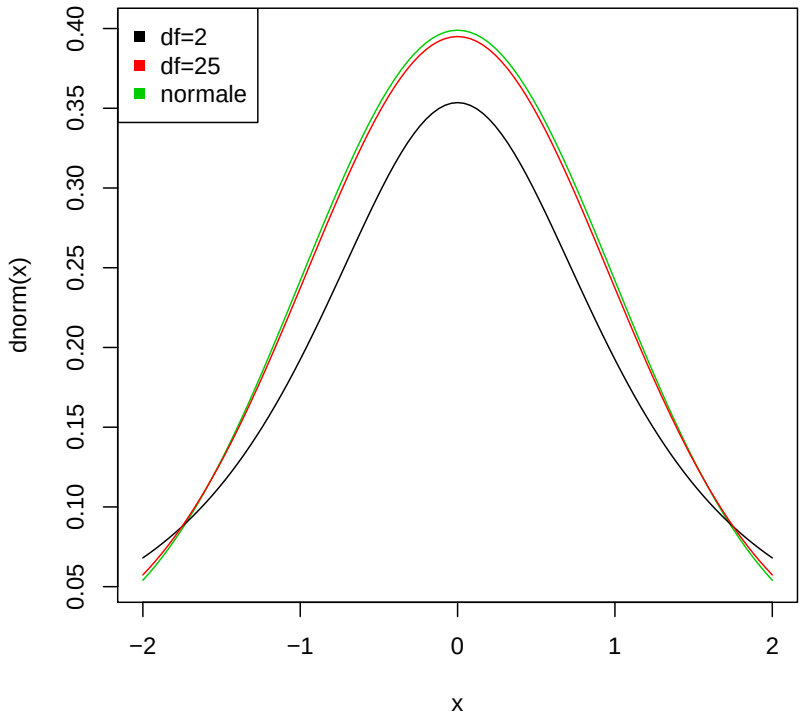


Figura 3.17: Grafico della distribuzione di Student a 10 gradi di libertà.

3.6.6 Intervalli di confidenza e test di Student (dati non appaiati)

La funzione di R che esegue il test di Student nelle sue diverse forme è `t.test`. Nella sua forma più semplice

```
> x=1:20;  t.test(x)
        One Sample t-test

data:  x
t = 7.9373, df = 19, p-value = 1.884e-07
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
```

```
  7.731189 13.268811
sample estimates:
mean of x
     10.5
```

In assenza di ipotesi R calcola il consuntivo

$$t = \frac{M_N(X) - \mu}{S_X}\sqrt{N}$$

assumendo che sia $\mu = 0$. Possiamo anche eseguire specificare l'ipotesi sul valore di μ:

```
> t.test(x,mu=7)
```

```
        One Sample t-test

data:  x
t = 2.6458, df = 19, p-value = 0.01595
alternative hypothesis: true mean is not equal to 7
95 percent confidence interval:
  7.731189 13.268811
sample estimates:
mean of x
     10.5
```

Possiamo infine specificare l'ipotesi alternativa. Per esempio se l'ipotesi alternativa è `"less"` il risultato del test cambia completamente.

```
> t.test(x,mu=7, alternative="less")
```

```
        One Sample t-test

data:  x
t = 2.6458, df = 19, p-value = 0.992
alternative hypothesis: true mean is less than 7
95 percent confidence interval:
     -Inf 12.78743
sample estimates:
mean of x
     10.5
```

In pratica ci viene fornito come p-value il valore dell'area sottesa dalla distribuzione di Student da $-\infty$ al valore di t se l'ipotesi alternativa è `"less"` e il valore dell'area sottesa dalla distribuzione di Student dal valore di t a $+\infty$ se l'ipotesi alternativa è `"greater"`

3.6.7 Test di Student per dati appaiati

Il test di Student per dati appaiati non è altro che un test di Student sulla differenza di 2 liste di dati di ugual lunghezza. Consideriamo ad esempio il confronto di 2 tecniche di misura applicate agli stessi campioni

```
> x<-c(1.46,2.22,2.84,1.97,1.13,2.35)
> y<-c(1.42,2.38,2.67,1.8,1.09,2.25)
```

Possiamo calcolare la differenza `x-y` ed applicare il test di Student oppure ottenere lo stesso risultato specificando l'opzione `paired=TRUE`

```
>  t.test(x,y,paired=TRUE)
```

```
        Paired t-test

data:  x and y
t = 1.2, df = 5, p-value = 0.2839
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.06852909  0.18852909
sample estimates:
mean of the differences
                   0.06
```

Il consuntivo `t` cade entro la regione di accettazione del test. È possibile specificare il livello di fiducia da utilizzare per il test di Student come:

$$\texttt{conf.level} = \textit{numero}$$

Il comando completo di tutti i parametri è quindi:

$$\texttt{t.test}(\textit{dati}_1, \textit{dati}_2, \texttt{paired=TRUE,conf.level} = \textit{valore})$$

Ad esempio eseguiamo un t-test per dati appaiati, tra $x = (1, 2, 3, 4)$ e $y = (3, 2, 4, 5)$ con *confidence level* di 0.85. Scriveremo

```
> t.test(1:4,5:2,paired=TRUE,conf.level=0.85)
```

Il consuntivo t cade fuori dalla regione di accettazione proposta.

3.7 Test χ^2 di indipendenza

Consideriamo il seguente *dataframe* che riporta le ambizioni di un gruppo di scolari americani

```
> data(kidinterest)
> str(kidinterest)
```

```
'data.frame':        478 obs. of  11 variables:
 $ Gender     : Factor w/ 2 levels "boy","girl": 1 1 2 2 2 2 2 2 2 2 ...
 $ Grade      : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Age        : int  11 10 11 11 10 11 10 10 10 10 ...
 $ Race       : Factor w/ 2 levels "Other","White": 2 2 2 2 2 2 2 2 2 2 ...
 $ Urban.Rural: Factor w/ 3 levels "Rural","Suburban",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ School     : Factor w/ 9 levels "Brentwood Elementary",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ Goals      : Factor w/ 3 levels "Grades","Popular",..: 3 2 2 2 2 2 2 1 3 3 ...
 $ Grades     : int  1 2 4 2 4 4 3 3 3 4 ...
 $ Sports     : int  2 1 3 3 2 2 4 4 2 3 ...
 $ Looks      : int  4 4 1 4 1 1 1 2 1 2 ...
 $ Money      : int  3 3 2 1 3 3 2 1 4 1 ...
```

Nella tabella le colonne che ci interessano al momento sono quelle che riguardano il sesso, gli obiettivi (scelti tra successo scolastico, capacità sportiva e popolarità) e la provenienza (colonne 1, 5 e 7). Nelle colonne dalla 8 alla 11 sono messi in ordine di importanza per il conseguimento della popolarità voti, sport, aspetto esteriore e denaro.

```
> interessi=kidinterest[,c(1,5,7)]
> head(interessi)
```

```
  Gender Urban.Rural   Goals
1    boy       Rural  Sports
2    boy       Rural Popular
3   girl       Rural Popular
4   girl       Rural Popular
5   girl       Rural Popular
6   girl       Rural Popular
```

```
> table(interessi)
```

```
, , Goals = Grades

      Urban.Rural
Gender Rural Suburban Urban
  boy     21       51    45
  girl    36       36    58

, , Goals = Popular

      Urban.Rural
Gender Rural Suburban Urban
  boy     19       20    11
  girl    31       22    38
```

```
, , Goals = Sports

      Urban.Rural
Gender Rural Suburban Urban
  boy     26       18    16
  girl    16        4    10
```

Consideriamo per esempio le variabili provenienza e traguardi

```
> interessi2=kidinterest[,c(5,7)]
> tabella=table(interessi2)
> tabella
```

```
           Goals
Urban.Rural Grades Popular Sports
   Rural        57      50     42
   Suburban     87      42     22
   Urban       103      49     26
```

Il test χ^2 di indipendenza consente di verificare se due variabili sono indipendenti. Se consideriamo le due variabili precedenti sesso e interessi. R dispone del comando `chisq.test`, dalla sintassi generale:

`chisq.test`(*tabella*)

Nell'esempio

```
> data(studenti)
> str(studenti)
```

```
'data.frame':        96 obs. of  9 variables:
 $ X   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Sex : Factor w/ 2 levels "F","M": 2 2 2 1 2 1 2 2 1 1 ...
 $ W   : num  86 53 64 61 64 51 78 55 59 52 ...
 $ H   : num  1.9 1.76 1.74 1.64 1.8 1.68 1.78 1.68 1.68 1.67 ...
 $ Eyes: Factor w/ 3 levels "azzurri","castani",..: 2 1 3 2 3 2 2 2 2 2 ...
 $ Hair: Factor w/ 3 levels "biondi","castani",..: 3 2 2 2 2 2 2 3 2 2 ...
 $ Sh  : int  48 42 41 39 42 38 42 40 38 39 ...
 $ hM  : num  1.58 1.7 1.63 1.65 1.56 1.6 1.65 1.56 1.58 1.65 ...
 $ hF  : num  1.82 1.6 1.8 1.78 1.75 1.75 1.75 1.7 1.83 1.7 ...
```

```
> tabellaEH=table(studenti$Eyes,studenti$Hair)
> chisq.test(tabellaEH)
```

```
        Pearson's Chi-squared test

data:  tabellaEH
X-squared = 5.9614, df = 4, p-value = 0.202
```

L'intervallo di accettazione dell'ipotesi (che ricordiamo è l'indipendenza) al 95% di fiducia e 1 gradi di libertà è $[0, 3.841]$, il consuntivo cade dentro, per cui l'ipotesi è accettata. Per eliminare la correzione di Pearson si utilizza il parametro `correct=FALSE`. Ad esempio scriveremo:

```
> chisq.test(tabellaEH,correct=FALSE)
```

3.7.1 Test χ^2 di adeguamento

Consideriamo una variabile aleatoria discreta con frequenza assoluta delle uscite racchiuse in una lista `data`. Ci si pone il problema di stabilire se tali frequenze sono compatibili con le probabilità (riportate nella lista p).

```
> data<-c(2,3,4,5,6,7,8,9,10,11)
> prob<-c(5,20,5,10,5,15,5,10,10,15)
> sum(prob)
> chisq.test(data,p=prob,rescale.p=TRUE)
```

Si è usata qui la scelta `rescale.p=TRUE` in quanto la somma delle probabilità non era 1. L'uscita del test riporta il valore del consuntivo χ^2 i gradi di libertà ed il valore p.

3.8 Distribuzione Binomiale

Il coefficiente binomiale è definito come

$$\texttt{choose}(n, m) = \binom{n}{m} = \frac{n!}{m! \times (n-m)!}$$

Ad esempio

```
> choose(6,3)
[1] 20
```

La distribuzione binomiale in R ha la sintassi

dbinom(*successi*, *prove*, *probabilità successo*)

e fornisce la probabilità di ottenere nel corso di un certo numero di prove il numero di successi indicato. Ad esempio, nel lancio di un dado 10 volte, vogliamo determinare la probabilità che esca *esattamente* due volte il numero 4:

```
> dbinom(2,10,1/6)
[1] 0.29071
```

La probabilità è circa del 29%.

conta i caratteri del tuo nome

vettore che contenga il quadrato dei primi 8 numeri pari

vettore che contenga la radice cubica dei primi 10 numeri naturali.

inserire in una matrice le coordinate 2D dei punti di un pentagono. Plot dei punti, come punti, come linee e come linee tratteggiate. Più arduo: riempimento della superficie

derivata di $x^3 + x^2 + 2x + 1$, plot della funzione e plot della derivata come due pannelli distinti in un grafico unico. hint: par(mfrow)

Tabelle delle distribuzioni statistiche

Per generare la tabella delle aree sottese dalla distribuzione normale da 0 ad x si deve per prima cosa tenere conto del fatto che `pnorm` è la cumulativa ad una sola coda. Si sceglie l'intervallo di tabulazione, il numero di colonne ed il numero di cifre.

```
> start=0.01
> stop=3.00
> step=0.01;
> nc=6;
> cifre=5;
> correzione<-function(x)  round(10^cifre* (pnorm(x)-0.5))/10^cifre
>  tabnormale<-cbind(matrix(correzione(seq(start,stop,by=step)),nc=nc),
+  matrix(seq(start,stop,by=step),nc=nc))
> as.vector(t(matrix(c(nc+1:nc,1:nc),nc=2)))->ordinecol
> colnames( tabnormale)= rep(c( "P=A","x"),each=6)
> rownames( tabnormale)=rep("",nrow( tabnormale))
```

In modo simile per generare la tabella della distribuzione t di Student si selezionano i livelli di fiducia di interesse e i gradi di libertà

```
> gradi=c(1:40,50,60,70,80,90,100,150,200,Inf)
> fiducia=c(0.8,0.85,0.9,0.95,0.98,0.99,0.999);
> ncol=length(fiducia);
> nrow=length(gradi);
> cifre=5;
> tstud<-function(x,gradi,cifre)
+ round(10^cifre*qt((1+x)/2,gradi))/10^cifre
> tabstudent=matrix(0,ncol=ncol,nrow=nrow)
>  for (i in 1:length(fiducia))
+  tabstudent[,i]= tstud(fiducia[i],gradi,5)
```

```
>  rownames(tabstudent)=gradi
> colnames(tabstudent)=fiducia
```

Per la distribuzione χ^2 si procede esattamente come sopra

```
> gradi=c(1:40,50,60,70,80,90,100,150,200)
> fiducia=c(0.8,0.85,0.9,0.95,0.98,0.99,0.999);
> ncol=length(fiducia);
> nrow=length(gradi);
> cifre=4;
> chiqua<-function(x,gradi,cifre)
+ round(10^cifre*qchisq(x,gradi))/10^cifre
> tabchi=matrix(0,ncol=ncol,nrow=nrow)
> for (i in 1:length(fiducia))
+ tabchi[,i]=chiqua(fiducia[i],gradi,5)
> rownames(tabchi)=gradi
> colnames(tabchi)=fiducia
```

Per la distribuzione di Fisher occorre specificare il numero di gradi di libertà del numeratore e del denominatore e fissare i valori di significatività (0.05 e 0.01)

```
> gradinum=1:9
> gradiden=c(1:40,50,60,70,80,90,100,150,200)
> fiducia=c(0.95,0.99);
> ncol=length(gradinum);
> nrow=length(gradiden);
> cifre=3;
> fisher95<-function(gradinum,gradiden,cifre)
+ round(10^cifre*qf(fiducia[1],gradinum,gradiden))/10^cifre
> fisher095=matrix(0,ncol=ncol,nrow=nrow);
> for (i in 1:length(gradinum))
+ fisher095[,i]= fisher95(gradinum[i],gradiden,cifre)
>  rownames(fisher095)=gradiden
> colnames(fisher095)=gradinum
> fisher099=fisher095;
> fisher99<-function(gradinum,gradiden,cifre)
+ round(10^cifre*qf(fiducia
+ [2],gradinum,gradiden))/10^cifre;for (i in 1:length(gradinum))
> fisher099[,i]= fisher99(gradinum[i],gradiden,cifre)
```

Aree A della distribuzione normale da 0 ad x

x	P=A	x	P=A	x	P=A	x	P=A	x	P=A	x	P=A
0.01	0.00399	0.51	0.19497	1.01	0.34375	1.51	0.43448	2.01	0.47778	2.51	0.49396
0.02	0.00798	0.52	0.19847	1.02	0.34614	1.52	0.43574	2.02	0.47831	2.52	0.49413
0.03	0.01197	0.53	0.20194	1.03	0.34849	1.53	0.43699	2.03	0.47882	2.53	0.49430
0.04	0.01595	0.54	0.20540	1.04	0.35083	1.54	0.43822	2.04	0.47932	2.54	0.49446
0.05	0.01994	0.55	0.20884	1.05	0.35314	1.55	0.43943	2.05	0.47982	2.55	0.49461
0.06	0.02392	0.56	0.21226	1.06	0.35543	1.56	0.44062	2.06	0.48030	2.56	0.49477
0.07	0.02790	0.57	0.21566	1.07	0.35769	1.57	0.44179	2.07	0.48077	2.57	0.49492
0.08	0.03188	0.58	0.21904	1.08	0.35993	1.58	0.44295	2.08	0.48124	2.58	0.49506
0.09	0.03586	0.59	0.22240	1.09	0.36214	1.59	0.44408	2.09	0.48169	2.59	0.49520
0.10	0.03983	0.60	0.22575	1.10	0.36433	1.60	0.44520	2.10	0.48214	2.60	0.49534
0.11	0.04380	0.61	0.22907	1.11	0.36650	1.61	0.44630	2.11	0.48257	2.61	0.49547
0.12	0.04776	0.62	0.23237	1.12	0.36864	1.62	0.44738	2.12	0.48300	2.62	0.49560
0.13	0.05172	0.63	0.23565	1.13	0.37076	1.63	0.44845	2.13	0.48341	2.63	0.49573
0.14	0.05567	0.64	0.23891	1.14	0.37286	1.64	0.44950	2.14	0.48382	2.64	0.49585
0.15	0.05962	0.65	0.24215	1.15	0.37493	1.65	0.45053	2.15	0.48422	2.65	0.49598
0.16	0.06356	0.66	0.24537	1.16	0.37698	1.66	0.45154	2.16	0.48461	2.66	0.49609
0.17	0.06749	0.67	0.24857	1.17	0.37900	1.67	0.45254	2.17	0.48500	2.67	0.49621
0.18	0.07142	0.68	0.25175	1.18	0.38100	1.68	0.45352	2.18	0.48537	2.68	0.49632
0.19	0.07535	0.69	0.25490	1.19	0.38298	1.69	0.45449	2.19	0.48574	2.69	0.49643
0.20	0.07926	0.70	0.25804	1.20	0.38493	1.70	0.45543	2.20	0.48610	2.70	0.49653
0.21	0.08317	0.71	0.26115	1.21	0.38686	1.71	0.45637	2.21	0.48645	2.71	0.49664
0.22	0.08706	0.72	0.26424	1.22	0.38877	1.72	0.45728	2.22	0.48679	2.72	0.49674
0.23	0.09095	0.73	0.26730	1.23	0.39065	1.73	0.45818	2.23	0.48713	2.73	0.49683
0.24	0.09483	0.74	0.27035	1.24	0.39251	1.74	0.45907	2.24	0.48745	2.74	0.49693
0.25	0.09871	0.75	0.27337	1.25	0.39435	1.75	0.45994	2.25	0.48778	2.75	0.49702
0.26	0.10257	0.76	0.27637	1.26	0.39617	1.76	0.46080	2.26	0.48809	2.76	0.49711
0.27	0.10642	0.77	0.27935	1.27	0.39796	1.77	0.46164	2.27	0.48840	2.77	0.49720
0.28	0.11026	0.78	0.28230	1.28	0.39973	1.78	0.46246	2.28	0.48870	2.78	0.49728
0.29	0.11409	0.79	0.28524	1.29	0.40147	1.79	0.46327	2.29	0.48899	2.79	0.49736
0.30	0.11791	0.80	0.28814	1.30	0.40320	1.80	0.46407	2.30	0.48928	2.80	0.49744
0.31	0.12172	0.81	0.29103	1.31	0.40490	1.81	0.46485	2.31	0.48956	2.81	0.49752
0.32	0.12552	0.82	0.29389	1.32	0.40658	1.82	0.46562	2.32	0.48983	2.82	0.49760
0.33	0.12930	0.83	0.29673	1.33	0.40824	1.83	0.46638	2.33	0.49010	2.83	0.49767
0.34	0.13307	0.84	0.29955	1.34	0.40988	1.84	0.46712	2.34	0.49036	2.84	0.49774
0.35	0.13683	0.85	0.30234	1.35	0.41149	1.85	0.46784	2.35	0.49061	2.85	0.49781
0.36	0.14058	0.86	0.30511	1.36	0.41309	1.86	0.46856	2.36	0.49086	2.86	0.49788
0.37	0.14431	0.87	0.30785	1.37	0.41466	1.87	0.46926	2.37	0.49111	2.87	0.49795
0.38	0.14803	0.88	0.31057	1.38	0.41621	1.88	0.46995	2.38	0.49134	2.88	0.49801
0.39	0.15173	0.89	0.31327	1.39	0.41774	1.89	0.47062	2.39	0.49158	2.89	0.49807
0.40	0.15542	0.90	0.31594	1.40	0.41924	1.90	0.47128	2.40	0.49180	2.90	0.49813
0.41	0.15910	0.91	0.31859	1.41	0.42073	1.91	0.47193	2.41	0.49202	2.91	0.49819
0.42	0.16276	0.92	0.32121	1.42	0.42220	1.92	0.47257	2.42	0.49224	2.92	0.49825
0.43	0.16640	0.93	0.32381	1.43	0.42364	1.93	0.47320	2.43	0.49245	2.93	0.49831
0.44	0.17003	0.94	0.32639	1.44	0.42507	1.94	0.47381	2.44	0.49266	2.94	0.49836

0.45	0.17364	0.95	0.32894	1.45	0.42647	1.95	0.47441	2.45	0.49286	2.95	0.49841
0.46	0.17724	0.96	0.33147	1.46	0.42785	1.96	0.47500	2.46	0.49305	2.96	0.49846
0.47	0.18082	0.97	0.33398	1.47	0.42922	1.97	0.47558	2.47	0.49324	2.97	0.49851
0.48	0.18439	0.98	0.33646	1.48	0.43056	1.98	0.47615	2.48	0.49343	2.98	0.49856
0.49	0.18793	0.99	0.33891	1.49	0.43189	1.99	0.47670	2.49	0.49361	2.99	0.49861
0.50	0.19146	1.00	0.34134	1.50	0.43319	2.00	0.47725	2.50	0.49379	3.00	0.49865

Distribuzione di Student

```
> tabstudent
```

	0.8	0.85	0.9	0.95	0.98	0.99	0.999
1	3.07768	4.16530	6.31375	12.70620	31.82052	63.65674	636.61925
2	1.88562	2.28193	2.91999	4.30265	6.96456	9.92484	31.59905
3	1.63774	1.92432	2.35336	3.18245	4.54070	5.84091	12.92398
4	1.53321	1.77819	2.13185	2.77645	3.74695	4.60409	8.61030
5	1.47588	1.69936	2.01505	2.57058	3.36493	4.03214	6.86883
6	1.43976	1.65017	1.94318	2.44691	3.14267	3.70743	5.95882
7	1.41492	1.61659	1.89458	2.36462	2.99795	3.49948	5.40788
8	1.39682	1.59222	1.85955	2.30600	2.89646	3.35539	5.04131
9	1.38303	1.57374	1.83311	2.26216	2.82144	3.24984	4.78091
10	1.37218	1.55924	1.81246	2.22814	2.76377	3.16927	4.58689
11	1.36343	1.54756	1.79588	2.20099	2.71808	3.10581	4.43698
12	1.35622	1.53796	1.78229	2.17881	2.68100	3.05454	4.31779
13	1.35017	1.52992	1.77093	2.16037	2.65031	3.01228	4.22083
14	1.34503	1.52310	1.76131	2.14479	2.62449	2.97684	4.14045
15	1.34061	1.51723	1.75305	2.13145	2.60248	2.94671	4.07277
16	1.33676	1.51213	1.74588	2.11991	2.58349	2.92078	4.01500
17	1.33338	1.50766	1.73961	2.10982	2.56693	2.89823	3.96513
18	1.33039	1.50371	1.73406	2.10092	2.55238	2.87844	3.92165
19	1.32773	1.50019	1.72913	2.09302	2.53948	2.86093	3.88341
20	1.32534	1.49704	1.72472	2.08596	2.52798	2.84534	3.84952
21	1.32319	1.49419	1.72074	2.07961	2.51765	2.83136	3.81928
22	1.32124	1.49162	1.71714	2.07387	2.50832	2.81876	3.79213
23	1.31946	1.48928	1.71387	2.06866	2.49987	2.80734	3.76763
24	1.31784	1.48714	1.71088	2.06390	2.49216	2.79694	3.74540
25	1.31635	1.48517	1.70814	2.05954	2.48511	2.78744	3.72514
26	1.31497	1.48336	1.70562	2.05553	2.47863	2.77871	3.70661
27	1.31370	1.48169	1.70329	2.05183	2.47266	2.77068	3.68959
28	1.31253	1.48014	1.70113	2.04841	2.46714	2.76326	3.67391
29	1.31143	1.47870	1.69913	2.04523	2.46202	2.75639	3.65941
30	1.31042	1.47736	1.69726	2.04227	2.45726	2.75000	3.64596
31	1.30946	1.47611	1.69552	2.03951	2.45282	2.74404	3.63346
32	1.30857	1.47494	1.69389	2.03693	2.44868	2.73848	3.62180
33	1.30774	1.47384	1.69236	2.03452	2.44479	2.73328	3.61091
34	1.30695	1.47281	1.69092	2.03224	2.44115	2.72839	3.60072
35	1.30621	1.47184	1.68957	2.03011	2.43772	2.72381	3.59115
36	1.30551	1.47092	1.68830	2.02809	2.43449	2.71948	3.58215
37	1.30485	1.47005	1.68709	2.02619	2.43145	2.71541	3.57367
38	1.30423	1.46923	1.68595	2.02439	2.42857	2.71156	3.56568
39	1.30364	1.46846	1.68488	2.02269	2.42584	2.70791	3.55812
40	1.30308	1.46772	1.68385	2.02108	2.42326	2.70446	3.55097
50	1.29871	1.46199	1.67591	2.00856	2.40327	2.67779	3.49601
60	1.29582	1.45820	1.67065	2.00030	2.39012	2.66028	3.46020
70	1.29376	1.45550	1.66691	1.99444	2.38081	2.64790	3.43501

80	1.29222	1.45349	1.66412	1.99006	2.37387	2.63869	3.41634
90	1.29103	1.45192	1.66196	1.98667	2.36850	2.63157	3.40194
100	1.29007	1.45067	1.66023	1.98397	2.36422	2.62589	3.39049
150	1.28722	1.44694	1.65508	1.97591	2.35146	2.60900	3.35657
200	1.28580	1.44508	1.65251	1.97190	2.34514	2.60063	3.33984
Inf	1.28155	1.43953	1.64485	1.95996	2.32635	2.57583	3.29053

Distribuzione χ^2

```
> tabchi
```

	0.8	0.85	0.9	0.95	0.98	0.99	0.999
1	1.64237	2.07225	2.70554	3.84146	5.41189	6.63490	10.82757
2	3.21888	3.79424	4.60517	5.99146	7.82405	9.21034	13.81551
3	4.64163	5.31705	6.25139	7.81473	9.83741	11.34487	16.26624
4	5.98862	6.74488	7.77944	9.48773	11.66784	13.27670	18.46683
5	7.28928	8.11520	9.23636	11.07050	13.38822	15.08627	20.51501
6	8.55806	9.44610	10.64464	12.59159	15.03321	16.81189	22.45774
7	9.80325	10.74790	12.01704	14.06714	16.62242	18.47531	24.32189
8	11.03009	12.02707	13.36157	15.50731	18.16823	20.09024	26.12448
9	12.24215	13.28804	14.68366	16.91898	19.67902	21.66599	27.87716
10	13.44196	14.53394	15.98718	18.30704	21.16077	23.20925	29.58830
11	14.63142	15.76710	17.27501	19.67514	22.61794	24.72497	31.26413
12	15.81199	16.98931	18.54935	21.02607	24.05396	26.21697	32.90949
13	16.98480	18.20198	19.81193	22.36203	25.47151	27.68825	34.52818
14	18.15077	19.40624	21.06414	23.68479	26.87276	29.14124	36.12327
15	19.31066	20.60301	22.30713	24.99579	28.25950	30.57791	37.69730
16	20.46508	21.79306	23.54183	26.29623	29.63318	31.99993	39.25235
17	21.61456	22.97703	24.76904	27.58711	30.99505	33.40866	40.79022
18	22.75955	24.15547	25.98942	28.86930	32.34616	34.80531	42.31240
19	23.90042	25.32885	27.20357	30.14353	33.68743	36.19087	43.82020
20	25.03751	26.49758	28.41198	31.41043	35.01963	37.56623	45.31475
21	26.17110	27.66201	29.61509	32.67057	36.34345	38.93217	46.79704
22	27.30145	28.82245	30.81328	33.92444	37.65950	40.28936	48.26794
23	28.42879	29.97919	32.00690	35.17246	38.96831	41.63840	49.72823
24	29.55332	31.13246	33.19624	36.41503	40.27036	42.97982	51.17860
25	30.67520	32.28249	34.38159	37.65248	41.56607	44.31410	52.61966
26	31.79461	33.42947	35.56317	38.88514	42.85583	45.64168	54.05196
27	32.91169	34.57358	36.74122	40.11327	44.13999	46.96294	55.47602
28	34.02657	35.71499	37.91592	41.33714	45.41885	48.27824	56.89229
29	35.13936	36.85383	39.08747	42.55697	46.69270	49.58788	58.30117
30	36.25019	37.99025	40.25602	43.77297	47.96180	50.89218	59.70306
31	37.35914	39.12437	41.42174	44.98534	49.22640	52.19139	61.09831
32	38.46631	40.25630	42.58475	46.19426	50.48670	53.48577	62.48722
33	39.57179	41.38614	43.74518	47.39988	51.74292	54.77554	63.87010
34	40.67565	42.51399	44.90316	48.60237	52.99524	56.06091	65.24722

35	41.77796	43.63994	46.05879	49.80185	54.24383	57.34207	66.61883
36	42.87880	44.76407	47.21217	50.99846	55.48886	58.61921	67.98517
37	43.97822	45.88645	48.36341	52.19232	56.73047	59.89250	69.34645
38	45.07628	47.00717	49.51258	53.38354	57.96880	61.16209	70.70289
39	46.17303	48.12628	50.65977	54.57223	59.20398	62.42812	72.05466
40	47.26854	49.24385	51.80506	55.75848	60.43613	63.69074	73.40196
50	58.16380	60.34599	63.16712	67.50481	72.61325	76.15389	86.66082
60	68.97207	71.34110	74.39701	79.08194	84.57995	88.37942	99.60723
70	79.71465	82.25535	85.52704	90.53123	96.38754	100.42518	112.31693
80	90.40535	93.10575	96.57820	101.87947	108.06934	112.32879	124.83922
90	101.05372	103.90406	107.56501	113.14527	119.64846	124.11632	137.20835
100	111.66671	114.65882	118.49800	124.34211	131.14168	135.80672	149.44925
150	164.34919	167.96177	172.58121	179.58063	187.67850	193.20769	209.26460
200	216.60878	220.74413	226.02105	233.99427	243.18692	249.44512	267.54053

Tabella della distribuzione di Fisher 95%

```
> fisher095
```

	1	2	3	4	5	6	7	8	9
1	161.448	199.500	215.707	224.583	230.162	233.986	236.768	238.883	240.543
2	18.513	19.000	19.164	19.247	19.296	19.330	19.353	19.371	19.385
3	10.128	9.552	9.277	9.117	9.013	8.941	8.887	8.845	8.812
4	7.709	6.944	6.591	6.388	6.256	6.163	6.094	6.041	5.999
5	6.608	5.786	5.409	5.192	5.050	4.950	4.876	4.818	4.772
6	5.987	5.143	4.757	4.534	4.387	4.284	4.207	4.147	4.099
7	5.591	4.737	4.347	4.120	3.972	3.866	3.787	3.726	3.677
8	5.318	4.459	4.066	3.838	3.687	3.581	3.500	3.438	3.388
9	5.117	4.256	3.863	3.633	3.482	3.374	3.293	3.230	3.179
10	4.965	4.103	3.708	3.478	3.326	3.217	3.135	3.072	3.020
11	4.844	3.982	3.587	3.357	3.204	3.095	3.012	2.948	2.896
12	4.747	3.885	3.490	3.259	3.106	2.996	2.913	2.849	2.796
13	4.667	3.806	3.411	3.179	3.025	2.915	2.832	2.767	2.714
14	4.600	3.739	3.344	3.112	2.958	2.848	2.764	2.699	2.646
15	4.543	3.682	3.287	3.056	2.901	2.790	2.707	2.641	2.588
16	4.494	3.634	3.239	3.007	2.852	2.741	2.657	2.591	2.538
17	4.451	3.592	3.197	2.965	2.810	2.699	2.614	2.548	2.494
18	4.414	3.555	3.160	2.928	2.773	2.661	2.577	2.510	2.456
19	4.381	3.522	3.127	2.895	2.740	2.628	2.544	2.477	2.423
20	4.351	3.493	3.098	2.866	2.711	2.599	2.514	2.447	2.393
21	4.325	3.467	3.072	2.840	2.685	2.573	2.488	2.420	2.366
22	4.301	3.443	3.049	2.817	2.661	2.549	2.464	2.397	2.342
23	4.279	3.422	3.028	2.796	2.640	2.528	2.442	2.375	2.320
24	4.260	3.403	3.009	2.776	2.621	2.508	2.423	2.355	2.300
25	4.242	3.385	2.991	2.759	2.603	2.490	2.405	2.337	2.282
26	4.225	3.369	2.975	2.743	2.587	2.474	2.388	2.321	2.265

27	4.210	3.354	2.960	2.728	2.572	2.459	2.373	2.305	2.250
28	4.196	3.340	2.947	2.714	2.558	2.445	2.359	2.291	2.236
29	4.183	3.328	2.934	2.701	2.545	2.432	2.346	2.278	2.223
30	4.171	3.316	2.922	2.690	2.534	2.421	2.334	2.266	2.211
31	4.160	3.305	2.911	2.679	2.523	2.409	2.323	2.255	2.199
32	4.149	3.295	2.901	2.668	2.512	2.399	2.313	2.244	2.189
33	4.139	3.285	2.892	2.659	2.503	2.389	2.303	2.235	2.179
34	4.130	3.276	2.883	2.650	2.494	2.380	2.294	2.225	2.170
35	4.121	3.267	2.874	2.641	2.485	2.372	2.285	2.217	2.161
36	4.113	3.259	2.866	2.634	2.477	2.364	2.277	2.209	2.153
37	4.105	3.252	2.859	2.626	2.470	2.356	2.270	2.201	2.145
38	4.098	3.245	2.852	2.619	2.463	2.349	2.262	2.194	2.138
39	4.091	3.238	2.845	2.612	2.456	2.342	2.255	2.187	2.131
40	4.085	3.232	2.839	2.606	2.449	2.336	2.249	2.180	2.124
50	4.034	3.183	2.790	2.557	2.400	2.286	2.199	2.130	2.073
60	4.001	3.150	2.758	2.525	2.368	2.254	2.167	2.097	2.040
70	3.978	3.128	2.736	2.503	2.346	2.231	2.143	2.074	2.017
80	3.960	3.111	2.719	2.486	2.329	2.214	2.126	2.056	1.999
90	3.947	3.098	2.706	2.473	2.316	2.201	2.113	2.043	1.986
100	3.936	3.087	2.696	2.463	2.305	2.191	2.103	2.032	1.975
150	3.904	3.056	2.665	2.432	2.274	2.160	2.071	2.001	1.943
200	3.888	3.041	2.650	2.417	2.259	2.144	2.056	1.985	1.927

>

Tabella della distribuzione di Fisher 99%

```
> fisher099
```

	1	2	3	4	5	6	7	8
1	4052.181	4999.500	5403.352	5624.583	5763.650	5858.986	5928.356	5981.070
2	98.503	99.000	99.166	99.249	99.299	99.333	99.356	99.374
3	34.116	30.817	29.457	28.710	28.237	27.911	27.672	27.489
4	21.198	18.000	16.694	15.977	15.522	15.207	14.976	14.799
5	16.258	13.274	12.060	11.392	10.967	10.672	10.456	10.289
6	13.745	10.925	9.780	9.148	8.746	8.466	8.260	8.102
7	12.246	9.547	8.451	7.847	7.460	7.191	6.993	6.840
8	11.259	8.649	7.591	7.006	6.632	6.371	6.178	6.029
9	10.561	8.022	6.992	6.422	6.057	5.802	5.613	5.467
10	10.044	7.559	6.552	5.994	5.636	5.386	5.200	5.057
11	9.646	7.206	6.217	5.668	5.316	5.069	4.886	4.744
12	9.330	6.927	5.953	5.412	5.064	4.821	4.640	4.499
13	9.074	6.701	5.739	5.205	4.862	4.620	4.441	4.302
14	8.862	6.515	5.564	5.035	4.695	4.456	4.278	4.140
15	8.683	6.359	5.417	4.893	4.556	4.318	4.142	4.004
16	8.531	6.226	5.292	4.773	4.437	4.202	4.026	3.890
17	8.400	6.112	5.185	4.669	4.336	4.102	3.927	3.791
18	8.285	6.013	5.092	4.579	4.248	4.015	3.841	3.705
19	8.185	5.926	5.010	4.500	4.171	3.939	3.765	3.631
20	8.096	5.849	4.938	4.431	4.103	3.871	3.699	3.564
21	8.017	5.780	4.874	4.369	4.042	3.812	3.640	3.506
22	7.945	5.719	4.817	4.313	3.988	3.758	3.587	3.453
23	7.881	5.664	4.765	4.264	3.939	3.710	3.539	3.406
24	7.823	5.614	4.718	4.218	3.895	3.667	3.496	3.363
25	7.770	5.568	4.675	4.177	3.855	3.627	3.457	3.324
26	7.721	5.526	4.637	4.140	3.818	3.591	3.421	3.288
27	7.677	5.488	4.601	4.106	3.785	3.558	3.388	3.256
28	7.636	5.453	4.568	4.074	3.754	3.528	3.358	3.226
29	7.598	5.420	4.538	4.045	3.725	3.499	3.330	3.198
30	7.562	5.390	4.510	4.018	3.699	3.473	3.304	3.173
31	7.530	5.362	4.484	3.993	3.675	3.449	3.281	3.149
32	7.499	5.336	4.459	3.969	3.652	3.427	3.258	3.127
33	7.471	5.312	4.437	3.948	3.630	3.406	3.238	3.106
34	7.444	5.289	4.416	3.927	3.611	3.386	3.218	3.087
35	7.419	5.268	4.396	3.908	3.592	3.368	3.200	3.069
36	7.396	5.248	4.377	3.890	3.574	3.351	3.183	3.052
37	7.373	5.229	4.360	3.873	3.558	3.334	3.167	3.036
38	7.353	5.211	4.343	3.858	3.542	3.319	3.152	3.021
39	7.333	5.194	4.327	3.843	3.528	3.305	3.137	3.006
40	7.314	5.179	4.313	3.828	3.514	3.291	3.124	2.993
50	7.171	5.057	4.199	3.720	3.408	3.186	3.020	2.890
60	7.077	4.977	4.126	3.649	3.339	3.119	2.953	2.823
70	7.011	4.922	4.074	3.600	3.291	3.071	2.906	2.777

80	6.963	4.881	4.036	3.563	3.255	3.036	2.871	2.742
90	6.925	4.849	4.007	3.535	3.228	3.009	2.845	2.715
100	6.895	4.824	3.984	3.513	3.206	2.988	2.823	2.694
150	6.807	4.749	3.915	3.447	3.142	2.924	2.761	2.632
200	6.763	4.713	3.881	3.414	3.110	2.893	2.730	2.601

	9
1	6022.473
2	99.388
3	27.345
4	14.659
5	10.158
6	7.976
7	6.719
8	5.911
9	5.351
10	4.942
11	4.632
12	4.388
13	4.191
14	4.030
15	3.895
16	3.780
17	3.682
18	3.597
19	3.523
20	3.457
21	3.398
22	3.346
23	3.299
24	3.256
25	3.217
26	3.182
27	3.149
28	3.120
29	3.092
30	3.067
31	3.043
32	3.021
33	3.000
34	2.981
35	2.963
36	2.946
37	2.930
38	2.915
39	2.901
40	2.888
50	2.785

```
60     2.718
70     2.672
80     2.637
90     2.611
100    2.590
150    2.528
200    2.497
```

Per esempio, consideriamo la serie temporale (oggetto di classe `ts`) `discoveries` che mostra l'andamento annuale del numero di scoperte scientifiche. Rappresentiamo graficamente tale serie trascurando prima e ricordandoci poi della sua struttura addizionale

Indice analitico

Bibliografia

[1] Daalgaard, P: *Introductory Statistics with R.*

[2] Project, DASL: *The data and Stories Library.* `http://lib.stat.cmu.edu/DASL`, accessed on line 2015.

Indice analitico