Statistica base

Paolo Bosetti (per TSM)

Indice

1 Numeri casuali e Distribuzioni

R dispone di una completa serie di funzioni per la gestione di numeri casuali, dalla generazione al calcolo della distribuzione. Le funzioni hanno nomi costituiti secondo questo schema: `<r|d|p|q><dist_name>()`, dove `dist_name` è un nome breve per la corrispondente distribuzione (ad es. `norm` per la *normale*) e il prefisso sta per:

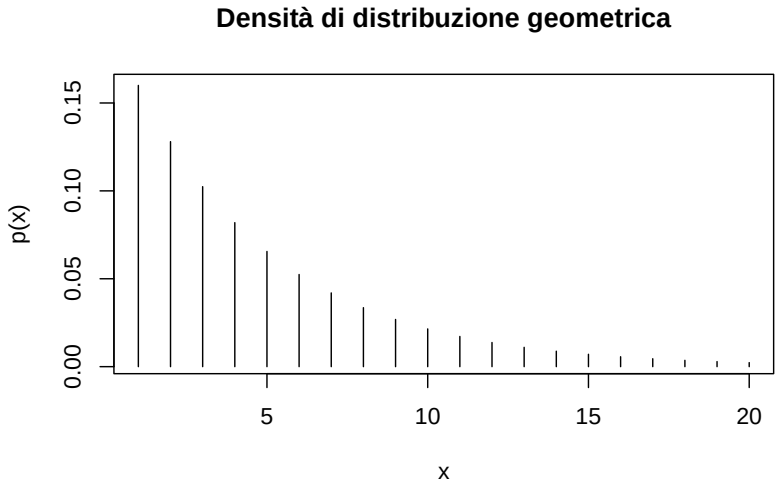
- `r` (random): genera numeri casuali
- `d` (density): funzione densità di distribuzione (*Probability Density Function*, PDF)
- `p` (probability): funzione di probabilità cumulata (*Cumulative Distribution Function*, CDF)
- `q` (quantile): funzione quantile, inversa della CDF

1.1 Distribuzioni di probabilità discrete

Le distribuzioni discrete hanno valore solo sui numeri interi. Le più comuni sono `geom` (geometrica), `binom` (binomiale), `pois` (Poisson) I grafici vengoni generalmente riportati con linee verticali, usando l'opzione `typ="h"` nei comandi di plot:

```
x <- 1:20
plot(dgeom(x, prob=0.2),
     typ="h",
     xlab="x",
```

```
     ylab="p(x)",
     main="Densità di distribuzione geometrica")
```



La CDF è invece preferibile plottarla a step, opzione `typ="s"`. Per chiarezza, confrontare il grafico otenuto con `typ="S"` (maiuscolo).

Inoltre, ricordarsi che la CDF della variabile casuale X può riportare la coda alta (upper tail):

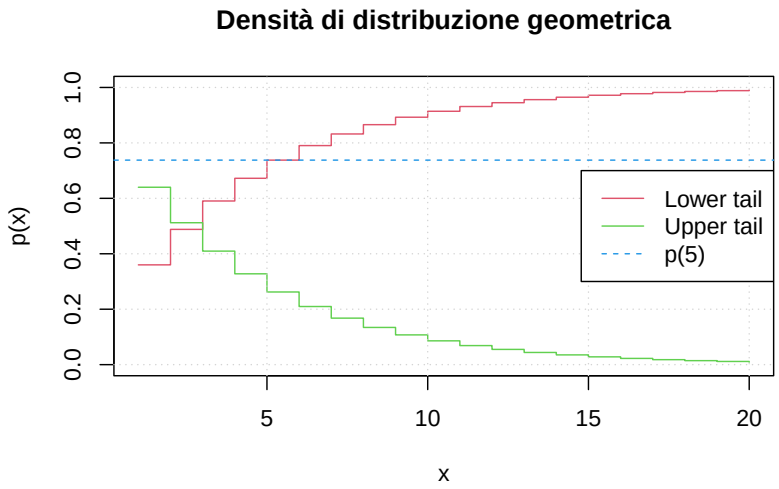
$$F_{X,U}(x) = \mathrm{P}(X \leq x) = \sum_{x_i \leq x} p(x_i)$$

e la coda bassa (lower tail):

$$F_{X,L}(x) = \mathrm{P}(X > x) = \sum_{x_i > x} p(x_i)$$

Per default, R considera la *lower tail* (`lower.tail=TRUE`):

```
plot(pgeom(x, prob=0.2),
     typ="s",
     xlab="x",
     ylab="p(x)",
     ylim=c(0,1),
     main="Densità di distribuzione geometrica",
     col="2")
lines(pgeom(x, prob=0.2, lower.tail=F),
     typ="s",
     xlab="x",
     ylab="p(x)",
     ylim=c(0,1),
     main="Densità di distribuzione geometrica",
     col=3)
grid()
abline(h=pgeom(5, prob=0.2), lty=2, col=4)
legend("right",
       legend=c("Lower tail", "Upper tail", "p(5)"),
       lty=c(1, 1, 2),
       col=2:4,
       bg="white")
```



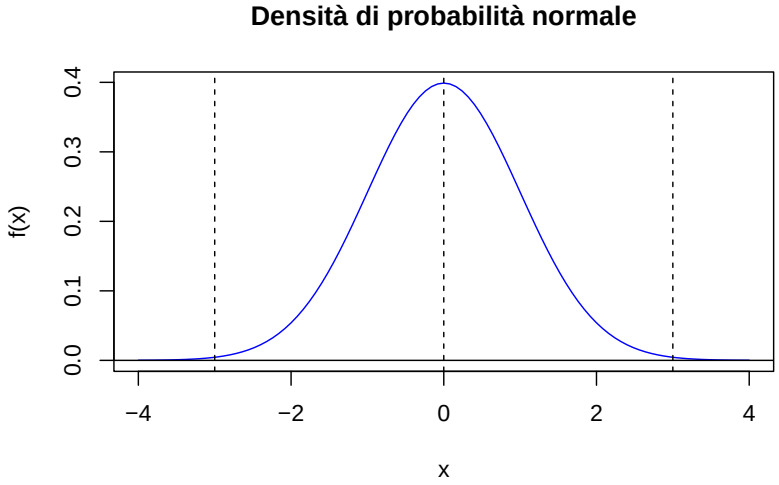
1.2 Distribuzioni di probabilità continue

Poco cambia rispetto alle distribuzioni discrete, salvo l'ovvia differenza che le funzioni hanno valore sui reali e che la CDF è definita come:

$$F_{X,U}(x) = \mathrm{P}(X \leq x) = \int_{-\infty}^{x} f(\xi)\mathrm{d}\xi$$

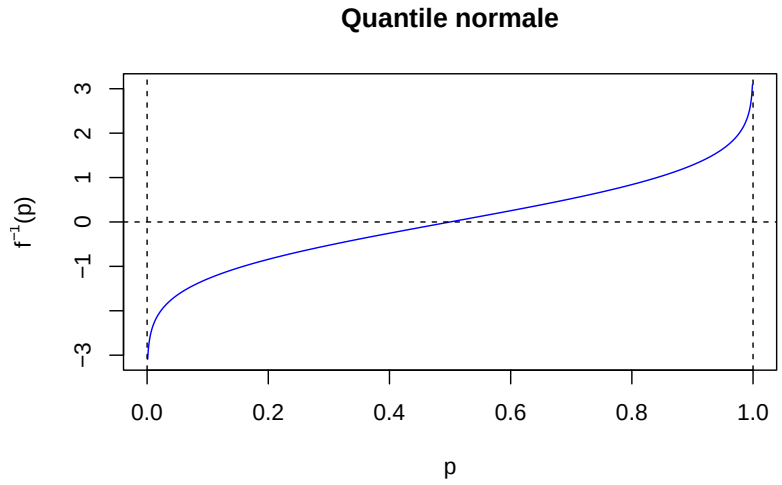
Inoltre, essendo la funzione continua posso creare il grafico con la funzione `curve()`:

```
curve(dnorm(x), from=-4, to=4, ylab="f(x)",
      main="Densità di probabilità normale",
      col="blue")
abline(v=c(-3, 0, 3), lty=2)
abline(h=0)
```



Per realizzare il grafico della funzione quantile è necessario ricordarsi che essa è l'inversa della CDF e, quindi, è definita solo nell'intervallo $(0, 1)$ e va all'infinito agli estremi:

```
curve(qnorm(x), from=0.001, to=0.999, n=1000,
      ylab=TeX("f^{-1}(p)"),
      xlab="p",
      col="blue",
      main="Quantile normale")
abline(h=0, lty=2)
abline(v=c(-1, 0, 1), lty=2)
```



Si noti la funzione `TeX` della librera `latex2exp` per inserire formule nelle etichette dei grafici ($f^{-1}(p)$).

Le funzioni che cominciano con `r` sono utili per *generare* vettori di numeri casuali. Per ottenere sempre la setessa sequenza pseudo-casuale si può impostare un seme:

```
set.seed(123)
x <- rnorm(100)
x[1:5]
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

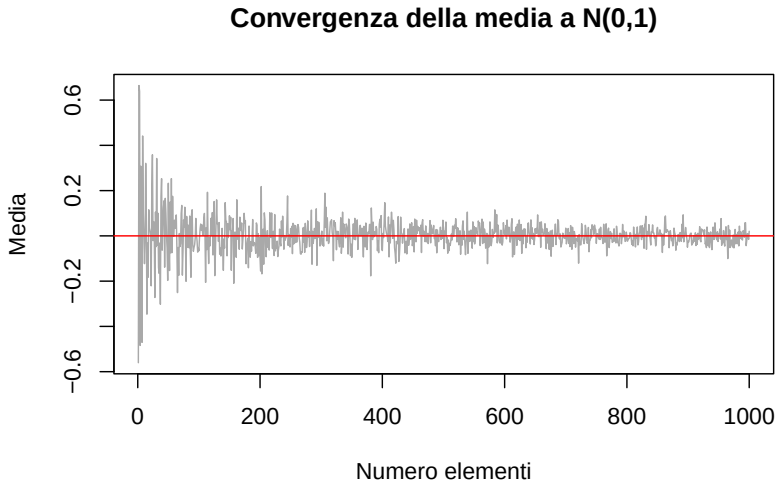
```
cat(paste("Media:", mean(x)),
    paste("Mediana:", median(x)),
    paste("Deviazione standard:", sd(x)),
    sep="\n")
```

```
## Media: 0.0904059086362066
## Mediana: 0.0617563090775401
## Deviazione standard: 0.912815879680979
```

Si noti come le funzioni `cat` e `paste` possono essere utilizzate per comporre *testo interpolato* (cioè testo che contiene i valori di espressioni valutate).

Possiamo studiare la *convergenza in distribuzione*:

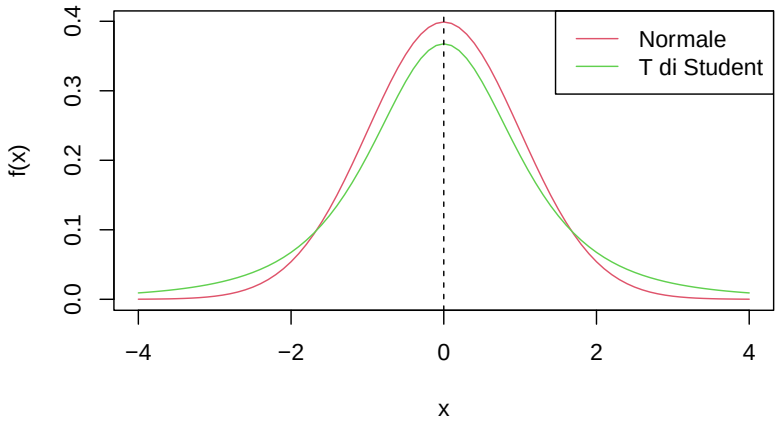
```
set.seed(123)
n <- 1:1000
plot(sapply(n, function(x) mean(rnorm(x))),
     typ="l",
     col="darkgrey",
     xlab="Numero elementi",
     ylab="Media",
     main="Convergenza della media a N(0,1)")
abline(h=0, col="red", lty=1)
```



Si noti che i dati sono stati generati non con un ciclo `for` ma con la funzione `sapply`: laddove possibile, le funzioni di mappatura sono sempre più veloci di un ciclo.

Vediamo ora come utilizzare i data frame per realizzare strutture dati più complesse:

```
df <- data.frame(x=seq(-4,4,0.1))
df$norm <- dnorm(df$x)
df$t <- dt(df$x, 3)
plot(norm~x, data=df, col=2, typ="l", ylab="f(x)")
lines(t~x, data=df, col=3)
legend("topright",
       legend=c("Normale", "T di Student"),
       lty=1,
       col=2:3)
abline(v=0, lty=2)
```



Si noti come, una volta creato, è possibile aggiungere nuove colonne ad un data frame con una semplice assegnazione mediante l'operatore `$`. Inoltre, la funzione `plot` è una *funzione generica*, che supporta cioè anche il metodo per la classe `formula`. In questo caso, la formula `norm~x` significa *colonna* ***`norm`*** *in funzione della colonna* ***`x`***.

2 Lettura e scrittura file

2.1 Scrivere file

In R scrivere dati su file è relativamente semplice. Ci sono sostanzialmente tre soluzioni:

1. *salvare* oggetti in formato proprietario R: `save()` (e l'opposto `load()`)
2. *scrivere* testo libero su file ASCII: `cat()`
3. *scrivere* dati tabulati ASCII: `write.table()` e `write.csv()`

La prima soluzione non permette lo scambio dati con altri software. La seconda soluzione è più flessibile, mentre la terza è più semplice.

In particolare, `cat()` e `write.table()` possono essere usate in sequenza per salvare una tabella anticipata da qualche riga di commento.

Per inciso, simili tabelle erano utilizzate per effetuare i T-test prima dell'avvento dei calcolatori.

```
file <- "t_values.txt"
n <- 1:120
p <- c(0.4, 0.25, 0.1, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001, 0.0005)
m <- t(sapply(n, function(x) round(qt(p, x, lower.tail=F), 3)))
rownames(m) <- as.character(n)
colnames(m) <- as.character(p)
cat(file=file, "# Quantili della distribuzione T\nDoF ")
write.table(m, "t_values.txt", quote = F, sep="\t", append=T)
head(m)
```

```
##     0.4  0.25   0.1  0.05  0.025   0.01  0.005  0.0025   0.001   5e-04
## 1 0.325 1.000 3.078 6.314 12.706 31.821 63.657 127.321 318.309 636.619
## 2 0.289 0.816 1.886 2.920  4.303  6.965  9.925  14.089  22.327  31.599
## 3 0.277 0.765 1.638 2.353  3.182  4.541  5.841   7.453  10.215  12.924
## 4 0.271 0.741 1.533 2.132  2.776  3.747  4.604   5.598   7.173   8.610
## 5 0.267 0.727 1.476 2.015  2.571  3.365  4.032   4.773   5.893   6.869
## 6 0.265 0.718 1.440 1.943  2.447  3.143  3.707   4.317   5.208   5.959
```

Come si vede, `cat()` oltre che per stampare stringhe in standard output può essere utilizzato per scrivere su file: è sufficiente passare il parametro `file`.

La matrice `m` può anche essere convertita in data frame per maggiore comodità, e esportata i ìn formato di interscambio CSV. Si noti comunque che `write.csv()` supporta in input sia matrici che data frame.

```
df <- as.data.frame(m)
write.csv(df, file="t_values_en.csv")
write.csv2(df, file="t_values_it.csv")
```

Si noti che `write.csv()` usa la virgola come separatore di campo e il punto come separatore dei decimali, mentre `write.csv2()` usa il punto e virgola come separatore di campo e la vorgola come separatore dei decimali. Quindi, `write.csv2()` è da usarsi se si intende importare il file creato, ad esempio, in versioni di Excel localizzate in Italiano o in lingue che usano la vorgola come separatore decimale.

2.2 Leggere da file

La lettura da file di testo libero può essere effettuata medante la funzione `scan()`. Tuttavia nella maggior parte dei casi è sufficiente leggere tabelle ASCII o csv. In questo caso si usano le funzioni `read.table()` o `read.csv()`/`read.csv2()`. Si noti che in questo caso la stringa che specifica il percorso di origine è un URI generico, quindi può essere sia un file locale che un percorso HTTP o HTTPS:

```
df <- read.table("http://repos.dii.unitn.it:8080/data/diet.dat", header=T)
str(df)
```

```
## 'data.frame':    24 obs. of  4 variables:
##  $ stdOrder: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ runOrder: int  2 10 11 20 4 8 9 12 14 15 ...
##  $ diet    : chr  "A" "A" "A" "A" ...
##  $ cTime   : int  60 59 63 62 65 66 67 63 64 71 ...
```

In particolare, l'opzione `header=T` specifica che i dati contengono i nomi delle colonne nella prima riga di intestazione.

3 Statistica descrittiva

3.1 Stimatori

È spesso utile descrivere un campione di numeri casuali mediante *indicatori* (come media, moda, mediana, deviazione standard) e mediante grafici. Tra i metodi grafici più utili ci sono gli istogrammi, di box-plot e i diagrammi quantile-quantile.

Vediamo gli stimatori più comuni:

```
v <- rnorm(10)
mean(v)
```

```
## [1] -0.2065041
```

```
median(v)
```

```
## [1] -0.1000487
```

```
var(v)
```

```
## [1] 0.6719288
```

```
sd(v)
```

```
## [1] 0.8197126
```

```
sd(v) == sqrt(var(v))
```

```
## [1] TRUE
```

```
quantile(v)
```

```
##         0%        25%        50%        75%       100%
## -1.7821402 -0.4729879 -0.1000487  0.3573861  0.9733320
```

Si noti la funzione `quantile()`: l'argomento opzionale `probs` è il vettore di probabilità per cui si vogliono i quantili (default a `seq(0, 1, 0.25)`).

Purtroppo R non fornisce una funzione per calcolare la moda (cioè il valore più frequente). É però facile costruirla:

```
set.seed(123)
(l <- sample(letters, replace = T)) # campionamento con reinserimento
```

```
##  [1] "o" "s" "n" "c" "j" "r" "v" "k" "e" "t" "n" "v" "y" "z" "e" "s" "y" "y" "i"
## [20] "c" "h" "z" "g" "j" "i" "s"
```

```
unique(l) # valori unici
```

```
##  [1] "o" "s" "n" "c" "j" "r" "v" "k" "e" "t" "y" "z" "i" "h" "g"
```

```
match(l, unique(l)) # indici dei valori unici che costruiscono l
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  3  7 11 12  9  2 11 11 13  4 14 12 15  5 13
## [26]  2
```

```
tabulate(match(l, unique(l))) # conta le ripetizioni degli indici
```

```
##  [1] 1 3 2 2 2 1 2 1 2 1 3 2 2 1 1
```

```
which.max(tabulate(match(l, unique(l)))) # posizione del massimo
```

```
## [1] 2
```

```
unique(l)[which.max(tabulate(match(l, unique(l))))] # moda
```

```
## [1] "s"
```

```
mymode <- function(x) {
  xu <- unique(x)
  xu[which.max(tabulate(match(x, xu)))]
}
mymode(l)
```

```
## [1] "s"
```

Si noti che la funzione `mode()` già esiste e ritorna lo *storage mode* di un oggetto. Inoltre, si noti che `mymode()` restituisce *il primo* elemento più frequente, tralasciando eventuali parimerito. In genere, è opportuno ordinare il vettore in modo da restituire il più comune e più grande (o più piccolo) elemento:

```
mymode(sort(l, decreasing = T))
```

```
## [1] "y"
```

È frequente il caso in cui i dati in ingresso hanno valori mancanti, rappresentabili in R con la costante speciale `NA`. Gli stimatori statistici hanno l'opzione `na.rm` (default `FALSE`) che specifica se rimuovere o meno i valori mancanti (e quindi modificare la dimensione del vettore) prima di calcolare la stima:

```
set.seed(123)
v <- sample(10)
v[sample(10, size=2)] <- NA
v
```

```
##  [1]  3 10  2  8 NA  9  1  7  5 NA
```

```
mean(v) # nota: x + NA = NA, per ogni x
```

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

```
mean(v, na.rm=T)
```

```
## [1] 5.625
```

Le funzioni `na.fail()`, `na.omit()` sono d'aiuto a manipolare i casi di NA, e sono automaticamente invocate dalle funzioni che supportano la gestione dei NA. Spesso si decide di sostituire i NA con valori medi dei restanti elementi:

```
v[is.na(v)] <- mean(v, na.rm=T)
v
```

```
##  [1]  3.000 10.000  2.000  8.000  5.625  9.000  1.000  7.000  5.000  5.625
```

```
mean(v)
```

```
## [1] 5.625
```

Sono utili anche gli stimatori di covarianza:

$$\mathrm{COV}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

e correlazione:

$$\mathrm{CORR}(X,Y) = \frac{\mathrm{COV}(X,Y)}{\sigma_X \sigma_Y} \quad \in [-1,1]$$

In R:

```r
set.seed(123)
n <- 10
x1 <- rnorm(n, 3, 0.5)
x2 <- rnorm(n, 6, 1)
x3 <- x1 * 2 + rnorm(n, sd=0.1)
c(cov(x1, x2), cov(x1, x3))
```

```
## [1] 0.2859477 0.4368317
```

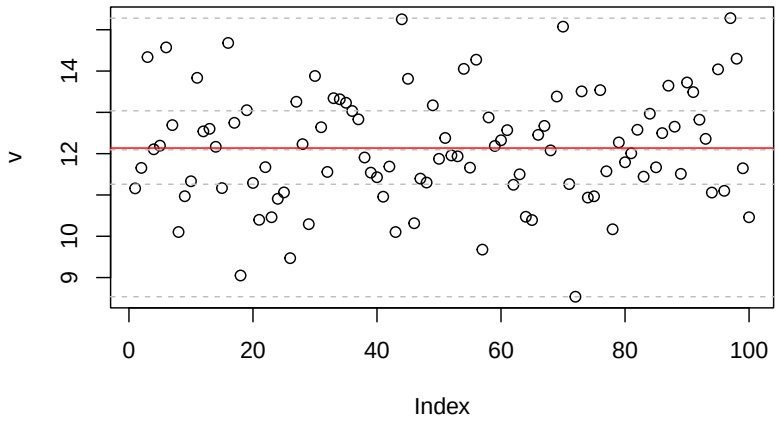
```r
c(cor(x1, x2), cor(x1, x3))
```

```
## [1] 0.5776151 0.9957156
```

3.2 Metodi grafici

È spesso utile rappresentare un vettore di dati casuali mediante metodi grafici. Possiamo utilizzare un diagramma a dispersione per visalizzare l'andamento ed evidenziare eventuali tendenze, e un istogramma per studiarne la distribuzione. La funzione kernel densty è inoltre una versione continua dell'istogramma, molto utile quando la dimensione del campione è molto grande.

```r
set.seed(123)
n <- 100
v <- rnorm(n, 12, 1.5)
plot(v)
abline(h=quantile(v), col="gray", lty=2)
abline(h=mean(v), col="red")
```

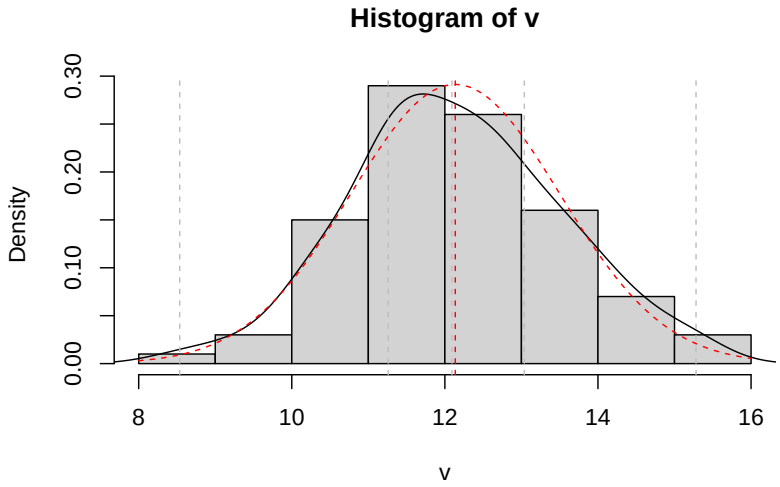


La serie non mostra tendenze o pattern (ovviamente!) e studiando i quantili osserviamo che la distribuzione appare leggermente gobab a sinistra, dato che la mediana è leggermente più bassa della media.

L'istogramma è creato dalla funzione `hist()`. Il numero di canne, o *bin*, in un istogramma è controllato dall'argomento `breaks`, che accetta o un vettore di punti di interruzione, o il nome dell'algoritmo (`"Sturges"`, `"Scott"`, `"FD"/"Freedman-Diaconis"`).

La versione continua dell'istogramma è ottenuta con la funzione `density()`, che è utile confrontare con la distribuzione di riferimento (in questo caso la normale):

```r
hist(v, freq=F) # freq=T riporta i conteggi invece delle frequenze
lines(density(v))
curve(dnorm(x, mean(v), sd(v)), col="red", lty=2, add=T)
abline(v=quantile(v), col="gray", lty=2)
abline(v=mean(v), col="red", lty=2)
```



La densità e l'istogramma confermano una leggera gobba a sinistra, anche se—come c'era da aspettarsi—il campione appare distribuito normalmente.

La *verifica di normalità* è un tema molto importante in statistica: generalmente si preferisce associare a tale verifica un *test statistico* che consenta di associare una probabilità di errore al risultato (come vedremo nel capitolo successivo). Tuttavia i metodi grafici risultano comunque utili a integrare i test. Ancora più utile dell'istogramma è il **diagramma quantile-quantile** (o *QQ-plot*), che confronta i quantili teorici con quelli campionari. Tanto più il grafico è allineato alla diagonale, tanto più la distribuzione del campione è simile a quella di riferimento (tipicamente la normale).

```
vu <- runif(length(v), 8, 15)
par(mfrow=c(1,2)) # grafici multipli su una riga, due colonne
qqnorm(v, main="Campione normale")
qqline(v, col="red")
qqnorm(vu, main="Campione uniforme")
qqline(vu, col="red")
```