Introduzione alla statistica descrittiva con R

Dispensa per il corso “Tecniche di Analisi di Dati I”, Corso di Laurea Specialistica in “Psicobiologia e Neuroscienze Cognitive”, Università di Parma.

terza versione

0. Indice

1. Introduzione

Questa prima sezione vi spiega come scaricare i programmi che vi servono e vi guida attraverso un primo esempio. Lo scopo è farsi un'idea generale di come utilizzeremo il software R per calcolare statistiche descrittive univariate e bivariate e per la loro presentazione grafica.

1.1 Contenuti e filosofia

Lo scopo di questa dispensa è mettervi in grado di lavorare su dati in R in modo da poter mettere in pratica la teoria che discuteremo durante le lezioni. Ci sono molte maniere di calcolare statistiche descrittive, a partire da programmi statistici come SPSS, oppure da fogli di calcolo come Excel, o utilizzando una calcolatrice. Qui imparerete a usare l'ambiente R. I vantaggi di R rispetto alle alternative sono spiegati nella sezione successiva.

La dispensa è pensata per essere visualizzata direttamente dallo schermo del vostro computer. La stampa non è vietata ma la visualizzazione sarà non ottimale sia perchè perderete i colori sia perché il layout di testo e immagini non è ottimizzato per la stampa. Inoltre, la dispensa non è fatta per essere letta ma è pensata come una guida alla vostra interazione con l'ambiente di R. Non dovete solo leggere, dovete fare. Infine, un vantaggio non da poco se leggete il documento con un lettore pdf è che potete sempre usare il find del lettore per cercare un argomento specifico. Per qualsiasi chiarimento potete comunque sempre rivolgervi al docente. Per chi lo desidera può essere utile scaricare anche il Manuale "An Introduction to R"[1] che può integrare il materiale presentato in questa dispensa. Inoltre sono disponibili in rete, sia in inglese sia in italiano, molti altri manuali o dispense per corsi universitari. Molto utile è anche il motore di ricerca R-seek (funziona come Google), disponibile su http://www.rseek.org/. Vi incoraggio ad esplorare queste risorse.

Questo è un documento informale e potrebbe contenere errori. Inviare commenti o segnalazioni di errori a nicola.bruno@unipr.it. Ogni suggerimento è il benvenuto.

1.2 Il software

Googlare "R". Dalla pagina "The R Project for Statistical Computing", selezionare download R. Scegliere uno dei CRAN Mirrors italiani e scaricare la versione appropriata per il proprio sistema operativo (Mac OS X, Windows, o Linux). R è un ambiente di sviluppo per l'analisi e la grafica statistica

[1] Venables, W.N., Smith, D.M., and the R Core Team (2012). *An introduction to R, Version 2.15.2*. R Core Team, http://R.com.

distribuito sotto licenza GNU GPL. I vantaggi di R sono molteplici. Innanzi tutto, R è gratuito mentre i maggiori pacchetti statistici sono molto costosi. R è un ambiente progettato in maniera da rispondere ai vostri comandi in maniera "intelligente", adattando la risposta al tipo di dati attualmente in esame. Inoltre R è molto ben documentato anche se imparare ad utilizzare bene la documentazione richiede un certo sforzo iniziale. R è anche molto completo. In R è possibile fare tutto quello che fanno gli altri pacchetti commerciali, in molti casi meglio di come lo fareste con quei pacchetti. Un altro vantaggio è che la comunità degli utilizzatori di R è molto vasta e collaborativa. Grazie a ciò, si rendono continuamente disponibili nuovi pacchetti per l'analisi, ed è relativamente facile trovare in rete esempi e soluzioni preconfezionate sui problemi che ci si trova ad affrontare. Altro vantaggio importante è che nell'ambiente R sono disponibili applicazioni per la grafica estremamente potenti, che vi consentiranno sia di esplorare i vostri dati sia di produrre grafici di qualità per la stampa o la videopresentazione. Infine, R non è solo un programma statistico ma un vero e proprio ambiente di sviluppo. Per chi ha già qualche esperienza di programmazione, questo facilita moltissimo l'apprendimento. Per chi non ce l'ha, costituisce una prima introduzione alla programmazione di alto livello, che faciliterà l'apprendimento di altri ambienti. In generale, la capacità di lavorare nell'ambiente R costituirà un aspetto qualificante del vostro CV dopo la laurea.

1.3 Un esempio

Una volta scaricato il software lanciare R. Dopo il prompt (">", o quello che compare sul vostro sistema), scrivere

```
> assign("v", 1)
```

assign è una funzione di R, mentre v è un oggetto. la funzione assegna il valore 1 all'oggetto chiamato "v" nello spazio di lavoro (workspace). Di solito in R l'assegnazione viene scritta in forma più compatta usando una freccia:

```
>v <- 1
```

Ora scrivere

```
> v
```

viene stampato l'oggetto chiamato v:

```
[1] 1
```

Ora scrivere

```
> help(assign)
```

Compare la documentazione relativa ad assign. Per ogni funzione di R è possibile visualizzare la documentazione relativa. Notate che è possibile specificare una serie di opzioni oltre al nome e al valore. Non preoccupatevi se ancora non capite tutti i dettagli. La maniera più efficiente di imparare ad usare R è per prove ed errori, quando avrete l'esigenza di ottenere un certo risultato. Ora scrivere

```
> v <- c(18,20,20,23,23,24,24,24,25,25,25,26,27,27,27,28,30)
```

In questo caso il valore assegnato è una serie di numeri invece che un numero singolo. L'ogetto è stato creato dalla funzione c, consultate l'help in linea per capire come funziona. Ora scrivete

```
>V
```

R risponde con un mesaggio di errore. R differenzia fra caratteri minuscoli e maiuscoli, quindi V e v non sono la stessa cosa. Ora scrivete

```
> v
```

R risponde stampando v

```
[1] 18 20 20 23 23 24 24 24 25 25 25 26 27 27 27 28 30
```

a questo punto avete un oggetto chiamato v nello spazio di lavoro. Infatti se scrivete

```
> objects()
```

R risponde elencando gli oggetti presenti:

```
[1] "v"
```

A questo punto possiamo porci il problema di presentare questi dati usando alcune funzioni di R. Supponiamo che i numeri siano voti ad una sessione d'esame del prof. Bruno. Vogliamo farci un'idea di come sono questi voti. Una maniera di farlo è scrivere

```
> median(v)
```

R risponde

[1] 25

il voto tipico è 25. Scrivere

> min(v)

[1] 18

> max(v)

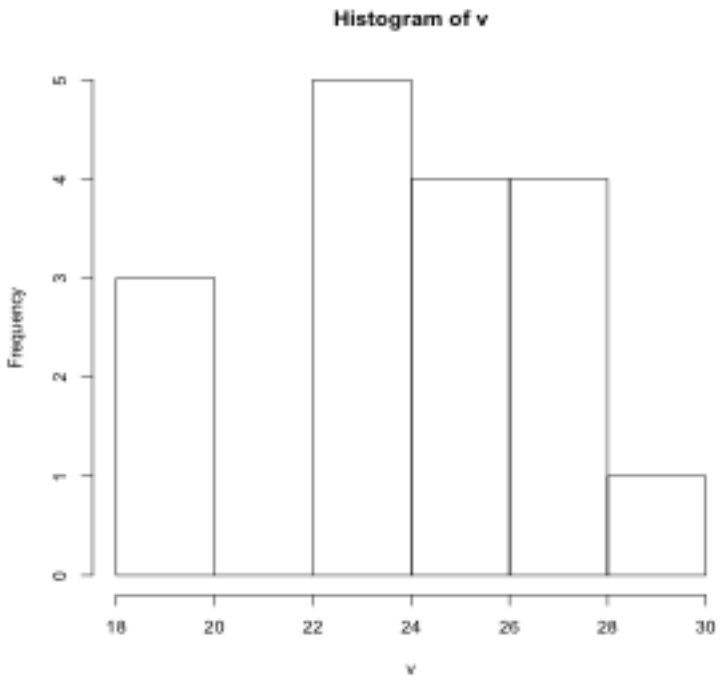
[1] 30

> length(v)

[1] 17

ci sono 17 voti, il minimo è 18 e il massimo è 30. Per studiare la distribuzione possiamo fare un istogramma. Scrivere

> hist(v)

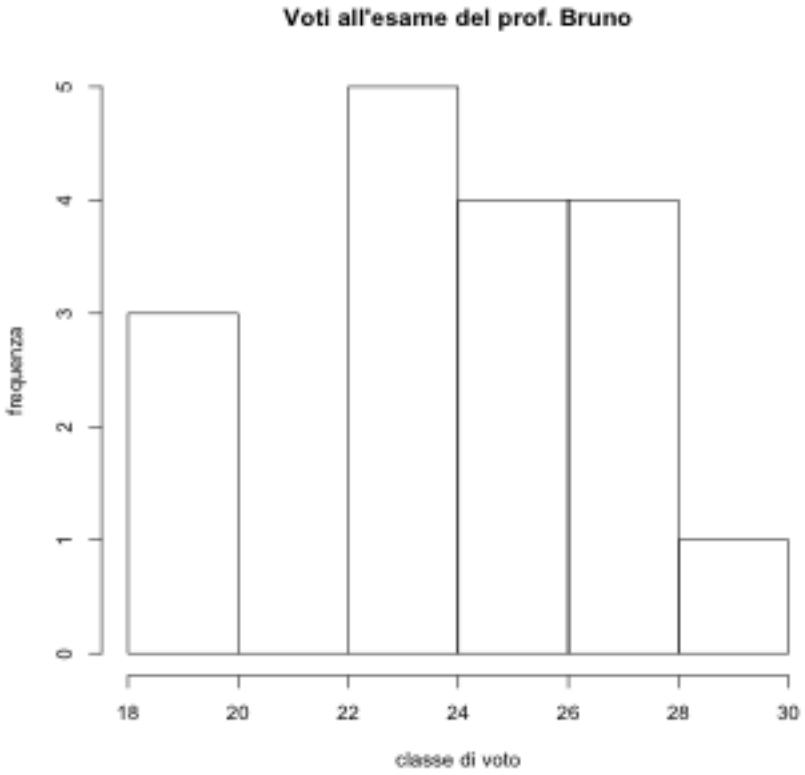


Notate che di default viene usata la convenzione del limite destro: nell'intervallo fra 18 e 20 ci sono i voti minori uguali a 20 (sono tre, un 18 e due 20), nell'intervallo fra 22 e 24 ci sono i voti compresi fra 24 e 23, ma

non quelli pari a 22, e così via.

Il titolo del grafico e le etichette degli assi sono poco informativi. Per migliorare il grafico aggiungiamo alcune opzioni. Scrivete

>hist(v, main="Voti all'esame del prof. Bruno", ylab="frequenza", xlab="classe di voto")



utilizzando le opzioni è possibile modificare molte altre caratteristiche dell'istogramma. Ad esempio, usare la convenzione del limite sinistro, o modificare i colori utilizzati. Studiate l'help in linea e provate a fare queste modifiche.

Una volta completato il grafico, l'ultima cosa da fare è salvarlo. Nell'ambiente OS X del Mac, per fare questo basta selezionare save o save as nel menu File del programma. Di default R lo salva in formato pdf, che è il formato che vi consiglio di usare. Il file in formato pdf può essere importato direttamente nel vostro word processor e modificato nelle dimensioni senza perdere qualità grafica.

1.4 Alcuni concetti utili

In R lavorerete principalmente con funzioni e oggetti.

Una funzione è un qualsiasi comando seguito da parentesi. All'interno delle parentesi vanno scritti gli argomenti della funzione, separati da virgole quando passate alla funzione più di un argomento. Anche se non passate alcun argomento è necessario aggiungere le parentesi. Pensate alle funzioni come alle istruzioni con cui dite ad R cosa deve fare, agli argomenti come le informazioni di cui R ha bisogno per farlo. In generale, non è necessario passare ad R tutti gli argomenti potenzialmente disponibili (e che potete vedere specificati nell'help in linea), perché per la maggior parte di questi R capisce a partire dai dati che valore dare a quegli argomenti o utilizza valori di default sensati.

Un oggetto è qualsiasi cosa a cui possiamo dare un nome. Gli oggetti si distinguono a seconda della loro classe (class) e del loro modo (mode). La classe definisce il tipo di oggetto. Le classi con cui lavoreremo sono: vector (vettore), matrix (matrice), factor (fattore) e data frame. Vedremo più avanti la differenza. Esiste anche una classe denominata lista (list) ma non ce ne occuperemo se non occasionalmente. Il modo di un oggetto definisce che tipo di dati sono contenuti nell'oggetto. I modi con cui lavoreremo sono numeric (numerico), logical (logico), e character (carattere). La maggior parte delle funzioni di R accetta come argomento oggetti di diverse classi e modi. In tal caso, il comportamento della funzione dipende dalle caratteristiche dell'oggetto a cui viene applicata. In molti casi, non occorre preoccuparsi di questo perché il comportamento di R di solito è quello che uno si aspetterebbe. Basta abituarsi a controllare sempre il risultato, visualizzando l'oggetto che viene creato, senza spaventarsi per gli eventuali messaggi di errore che è relativamente facile imparare ad interpretare.

È possibile usare R inserendo i comandi dopo il prompt, uno alla volta, come abbiamo fatto nell'esempio della sezione precedente. In alternativa, è possibile creare un file che contiene una sequenza di comandi (un "codice" o un "programma" di R) ed eseguirli inserendo il nome del file come argomento della funzione source(). Questo approccio è preferibile perchè consente di salvare in un file le diverse fasi del proprio lavoro, dalla lettura del file di dati, alla eventuale riorganizzazione dei dati stessi se in un formato non adatto al tipo di analisi che vogliamo fare, fino all'analisi vera e propria, procedendo un passo alla volta e correggendo eventuali errori. Vedremo a breve esempi anche di questa maniera di procedere.

2. Operazioni su dati in R

Possiamo usare R per fare calcoli aritmetici, calcoli su vettori e matrici, o per trovare il valore di operazioni logiche, operazioni su insiemi, o per lavorare su stringhe alfanumeriche. Vediamo ora le principali funzioni per questo. Se avete già qualche pratica di programmazione probabilmente potete anche saltare questa sezione o sarà sufficiente un'occhiata veloce. Se non avete pratica vi suggerisco invece di spendere un po' di tempo per familiarizzarvi con queste funzioni, che vi torneranno utili in seguito.

2.1 Operazioni su numeri

Scrivete

```
>x <- 4
>y <- 3
```

ora scrivete

```
>x+y
```

R risponde

```
[1] 7
```

ora scrivete

```
>x-y
```

```
[1] 1
```

```
>x*y
```

```
[1] 12
```

```
>x/y
```

```
[1] 1.333333
```

R vi consente di fare somme, sottrazioni, moltiplicazioni e divisioni. Per elevare a potenza scrivete

```
>x^y
```

```
[1] 64
```

per la radice quadrata

```
>sqrt(x)

[1] 2

>sqrt(y)

[1] 1.732051
```

dato che la radice n-esima di x = x elevato alla 1/n, la radice quadrata può essere calcolata anche come

```
>x^(1/2)

[1] 2
```

la radice cubica (e, generalizzando, n-esima) come

```
>x^(1/3)

[1] 1.587401
```

infatti

```
> 1.587401^3

[1] 4
```

attenzione alle parentesi che definiscono l'ordine delle operazioni. Scrivete

```
>x^1/3

[1] 1.333333.
```

R ha elevato x alla 1 e poi diviso per 3 invece che elevare alla potenza frazionaria. Un altro esempio:

```
> z <- x^y
> z
```

[1] 64

>z^(1/3)

[1] 4

Ora scrivete

>sqrt(x) == x^(1/2)

R risponde

[1] TRUE

Il simbolo == è l'operatore logico "uguale a". Un operatore logico restituisce solo due possibili risultati, TRUE e FALSE. Per esempio

>sqrt(x) != x^(1/2)

[1] FALSE

perché il simbolo != è l'operatore logico "diverso da". Gli operatori logici che potete utilizzare sono

<	"minore di"
>	"maggiore di"
<=	"minore di o uguale a"
>=	"maggiore di o uguale a"
==	"uguale a"
!=	"diverso da"
&	AND
I	OR
!	NOT

se alcuni non vi sono familiari non preoccupatevi. Se e quando avrete l'esigenza di usarli non avrete difficoltà a capire cosa significano.

2.2 Vettori

Scrivete

```
> v <- 1:6
>v
```

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

oppure

```
> v <- c(2,4,8,12)
>v

[1] 2 4 8 12
```

per accedere a elementi del vettore è possibile usare gli indici ponendoli in parentesi quadre:

```
> v[1]

[1] 2

> v[3]

[1] 8

> v[1:3]

[1] 2 4 8

> v[-4]

[1] 2 4 8
```

nell'ultimo esempio il segno meno indica che voglio stampare tutto il vettore, meno il quarto elemento. Per questo motivo il comando da lo stesso risultato di quello in cui richiedo gli elementi da 1 a 3 del vettore. Usando il comando c posso anche scrivere:

```
> v[-c(2,4)]

[1] 2 8
```

notate che gli argomenti di c sono gli indici del vettore, non i valori corrispondenti.

Ora scrivete

```
> v*2
```

```
[1] 4 8 16 24
```

moltiplicare il vettore per una costante (uno scalare) moltiplica ogni elemento del vettore per lo scalare. In generale, qualsiasi operazione su un vettore viene effettuata su tutti i suoi elementi. Ad esempio

```
> v + 10

[1] 12 14 18 22
```

ora scrivete

```
> v1 <- 1:3
> v2 <- 4:6
> v1 + v2

[1]  5 7 9
```

la somma di due vettori è il vettore che contiene le somme di ogni coppia di elementi. In generale

```
> v1 * v2

[1]  4 10 18

> v2 / v1

[1] 4.0 2.5 2.0
```

e così via. Per questo tipo di operazioni su vettori è bene che i vettori siano della stessa lunghezza. Se il più lungo è un multiplo del più corto, quest'ultimo viene "riciclato" fino ad arrivare alla lunghezza del più lungo. Se non lo è, R da un messaggio di errore.

Per un esempio più interessante, proviamo a creare un programma. Aprire un nuovo documento nell'editor di R (sul mio computer, icona con un foglio a sinistra dell'icona per stampare). Inserire i comandi sotto. Salvare con il nome voti.R.

```
# vettori con in punteggi di tre prove in itinere
it1 <- c(11, 27, 27.5, 19, 20, 17.5, 23.5, 23, 13, 20)
it2 <- c(8, 18, 15, 12,9, 18.5, 14.5, 14.5, 14.5, 15)
it3 <- c(11.5, 22, 14.5, 18, 22, 22  ,24,28, 25, 16.5)
media <- (it1 + it2 + it3)/3 # dei punteggi per ogni studente
```

Notare l'uso di # (segnala un commento, R ignora quello che segue). Eseguiamo il file:

> source('voti.R')

a questo punto possiamo visualizzare il risultato, stampando il vettore media:

> media

Un trucco utile: sul mio computer (e presumo anche sul vostro), per eseguire un programma basta trascinare l'icona del file nel workspace e dare invio. Questo è molto comodo perchè in questa maniera R individua automaticamente il path del file che vogliamo lanciare.

Un altro strumento utile per lavorare con vettori sono le funzioni per generare sequenze regolari, seq() e rep().

La sintassi di seq è seq(from = 1, to, by = 1, length) . Il primo argomento indica il valore iniziale della sequenza, che di default è 1. Il secondo il valore finale. Il terzo indica l'incremento tra un valore e il valore successivo, anche questo di default pari a 1. L'ultimo è la lunghezza della sequenza. Non tutti gli argomenti vanno specificati contemporaneamente. Ad esempio

> seq(from = 0, to = 50, by = 10)

genera questa sequenza di sei numeri:

[1] 0 10 20 30 40 50

la quale può essere generata anche in questo modo:

> seq(from = 0, to = 50, length = 6)

[1] 0 10 20 30 40 50

Gli argomenti possono essere specificati in ordine arbitrario, se i loro nomi sono specificati fra le parentesi. Se vengono passati i soli valori degli argomenti, invece, occorre che siano nell'ordine prefissato. Quindi avrei potuto scrivere

> seq(0, 50, 10)

ma se scrivo

```
> seq(0, 50, 6)
```

R genera questa sequenza perchè attribuisce il valore 6 all'argomento by:

```
[1]  0  6 12 18 24 30 36 42 48
```

Una sequenza in cui i valori sono incrementati di uno si può ottenere anche in questa maniera:

```
> 1:20

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

La funzione rep() ha la sintassi rep(x, times). Il primo argomento è l'oggetto su cui applicare la funzione. Il secondo specifica quante volte rep() deve ripetere l'oggetto. Ad esempio

```
> rep(x = 18:30, times = 3)

 [1] 18 19 20 21 22 23 24 25 26 27 28 29 30 18 19 20 21 22 23 24 25
26 27 28 29 30 18 19 20 21
[31] 22 23 24 25 26 27 28 29 30

> o <- seq(0, 1, 0.05)
> rep(o, 2)

 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60
0.65 0.70 0.75 0.80 0.85
[19] 0.90 0.95 1.00 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45
0.50 0.55 0.60 0.65 0.70
[37] 0.75 0.80 0.85 0.90 0.95 1.00
```

2.3 Matrici

Una matrice è un insieme di dati organizzati in righe e colonne. Un vettore è una matrice con una sola colonna. Scrivete

```
> m <- matrix(1:6)
> m

     [,1]
[1,]    1
```

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

Ora scrivete

```
> m <- matrix(1:6, nrow=2)
> m

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

abbiamo specificato che la matrice deve avere 2 righe. Possiamo anche specificare le colonne:

```
> m <- matrix(1:6, ncol = 2)
> m

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

in entrambi i casi la matrice viene costruita riempiendo una colonna alla volta. Per riempire la matrice una riga alla volta, scrivere

```
> m <- matrix(1:6, ncol = 2, byrow = TRUE)
> m

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

Notare la differenza con il caso precedente. Come per i vettori, gli operatori di base come + o * possono essere utilizzati per le matrici. Ad esempio,

```
> m1 <- matrix(1:6, nrow=2)
> m2 <- matrix(1:6, nrow=2)
> m1 + m2
```

```
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    4    8   12
```

genera una matrice con le somme di ogni coppia di elementi delle due matrici di partenza.

```
> m1 * m2

     [,1] [,2] [,3]
[1,]    1    9   25
[2,]    4   16   36
```

genera una matrice con i prodotti. Ricordare che per fare questo occorre che le due matrici abbiano le stesse dimensioni. Per sapere le dimensioni di una matrice, ad esempio di m1, scrivere

```
> dim(m1)

[1] 2 3
```

Usando gli operatori di base posso moltiplicare tutti gli elementi di una matrice per uno scalare, o sommare una costante, e così via. Ad esempio

```
> s <- 10
< m1 * s

     [,1] [,2] [,3]
[1,]   10   30   50
[2,]   20   40   60
```

moltiplica tutti gli elementi della matrice per dieci. Infine, è possibile effettuare operazioni proprie del calcolo matriciale in senso stretto, come il prodotto fra matrici, l'inversa, la trasposizione, e così via. Per ora non ce ne preoccupiamo.

Il comando matrix può essere utilizzato per creare tabelle con nomi per righe e colonne. Scrivere

```
> m <- matrix(1:6, nrow = 2, byrow = TRUE)
> righe <- c("a", "b")
> colonne <- c("prim", "sec", "ter")
> dimnames(m) <- list(righe, colonne)
> m
```

```
  prim sec ter
a    1   3   5s
b    2   4   6
```

a questo punto le etichette funzionano esattamente come gli indici

> m[”a”, ”sec”]

[1] 3

Vediamo ora come svolgere calcoli su righe o colonne. Create il programma conting.R inserendo questi comandi:

```
m <- matrix(1:6, nrow = 2, byrow = TRUE)
righe <- c("a", "b")
colonne <- c("prim", "sec",  "ter")
dimnames(m) <- list(righe, colonne)
tc <- apply(m, 2, sum) #totali colonne
mm <- rbind(m, tc)
tr <- apply(mm, 1, sum) #totali righe
mmm <- cbind(mm, tr)
```

Notate l’uso della funzione apply(), che applica una funzione ad una matrice o a un array[2]. La sua sintassi è apply(x, margin, fun), dove x è una matrice, margin specifica se la funzione va applicata sulle righe (margin = 1), colonne, (margin = 2), o righe e colonne (margin = c(1, 2)), e fun è la funzione da applicare, in questo caso sum(). Ora scrivete

> source(conting.R)
>mmm

R mostra la tabella di contingenza

```
   prim sec ter tr
a     1   2   3  6
b     4   5   6 15
tc    5   7   9 21
```

la funzione apply() con argomento 2 ha calcolato la somma per colonna della matrice m, con 1 ha calcolato la somma per riga. il comando cbind ha aggiunto una colonna con i totali di riga, rbind ha aggiunto una riga con i totali di colonna e con il totale generale.

[2] Per generare matrici che hanno più di due dimensioni e quindi più di due indici, R utilizza la funzione array(). L’uso è analogo a matrix().

2.4 Data frame

Vettori, matrici e matrici multidimensionali usano dati omogenei (ad esempio, tutti numerici come negli esempi precedenti). Molto spesso, con dati psicologici lavoriamo con dati sia numerici sia categoriali organizzati in righe e colonne. L'organizzazione più frequente è il cosiddetto formato rettangolare standard, in cui ogni riga è un caso e ogni colonna una variabile. Per questo tipo di dati utlizzeremo i data frame. Un data frame è una matrice in cui le colonne possono essere numeriche o caratteri.

Per creare un data frame si utilizza la funzione data.frame(). Ad esempio per creare un data frame con nomi di soggetti e dati, potremmo scrivere

```
> ss <- c("qui", "quo", "qua", "pippo", "pertica", "palla")
> tr <- c(440, 650, 340, 450, 260, 610)
> d <- data.frame(c(ss, tr))
```

Tuttavia, nella pratica della ricerca, capita raramente di procedere in questa maniera. Il caso tipico invece è quello in cui i dati sono già salvati in un file di testo, ad esempio, a partire da un foglio elettronico come Excel. Per leggerli in un data frame si utilizza la funzione read.table().

Facciamo un esempio. Aprire un nuovo documento nell'editor di R, copiarvi i contenuti sotto, salvare con il nome "dati".

```
lt    nm
a     12
b     45
c     23
d     78
e     49
f     73
g     80
```

Per leggere il data frame scrivere

```
> dd <- read.table("dati.R", header = TRUE)
```

R riconosce la struttra di dati e crea un oggetto di classe data frame, chiamato dd. Notate l'argomento header, che serve a specificare che la prima riga contiene i nomi delle variabili. Provate a confrontare con il comportamento di R se si omette (equivalente a specificare header = FALSE).

Per sapere come è strutturato il data frame, posso usare

```
> str(dd)
```

R risponde

```
'data.frame':        7 obs. of  2 variables:
 $ lt: Factor w/ 7 levels "a","b","c","d",..: 1 2 3 4 5 6 7
 $ nm: int  12 45 23 78 49 73 80
```

ossia, dicendo che dd è un oggetto di classe data frame, con 7 casi (righe) di 2 variabili (colonne). Le variabili si chiamano lt, un fattore a 7 livelli, e nm, un vettore di numeri interi.

In alternativa, per verificare quali variabili sono in colonna, avrei potuto dare un'occhiata alle prime righe del data frame, ad esempio con

```
> head(dd, 2)
```

che mostra le prime due righe (più lo header) del data frame. Ovviamente se avessi passato un 7 invece che un 2 avrei visto l'intero data frame, ma in un caso realistico potrei avere file di dati di centinaia di casi, e non sarebbe pratico visionarli tutti. Naturalmente esiste anche una funzione tail(), che per chi lo non lo sapesse in inglese vuol dire coda e che funziona come uno si aspetterebbe.

A questo punto nel vostro workspace esiste un oggetto che si chiama dd. Sappiamo che contiene due variabili, lt e nm, ma quest'ultime non sono oggetti di R. Provate a scrivere

```
> lt
```

R risponde con un messaggio di errore.

Per accedere individualmente ai contenuti di una sola colonna del data frame, occorre riferirsi alla colonna con il nome del dataframe, il segno di dollaro, e il nome della colonna. Ad esempio:

```
> dd$lt
```

stampa i contenuti della prima colonna lt. Usando la stessa convenzione, posso anche aggiungere colonne al dataframe. Ad esempio, una colonna chiamata xx

```
> dd$xx <- c(1, 2, 3, 4, 5, 6, 7)
```

Infatti se ora scrivo

```
> dd
```

R mi mostra un data frame con tre colonne

```
  lt nm xx
1  a 12  1
2  b 45  2
3  c 23  3
4  d 78  4
5  e 49  5
6  f 73  6
7  g 80  7
```

e se chiedo

```
> str(dd)
```

```
'data.frame':        7 obs. of  3 variables:
 $ lt: Factor w/ 7 levels "a","b","c","d",..: 1 2 3 4 5 6 7
 $ nm: int  12 45 23 78 49 73 80
 $ xx: num  1 2 3 4 5 6 7
```

3. Statistiche descrittive univariate

In questa sezione prenderemo confidenza con le funzioni di R per calcolare statiche descrittive univariate. Come prima cosa, scaricare il file IQ.txt dal sito del docente e salvarlo sul proprio computer. Ora scrivere

```
> d <- read.table("IQ.txt", header = TRUE)
```

attenzione a specificare correttamente il path del file. Come già detto, trascinando l'icona del file fra le virgolette il path viene specificato direttamente da R.

Diamo un'occhiata al file.

```
> head(d)

  Gender FSIQ VIQ PIQ Weight Height MRI_Count
1 Female  133 132 124    118   64.5    816932
2   Male  140 150 124    151   72.5   1001121
3   Male  139 123 150    143   73.3   1038437
4   Male  133 129 128    172   68.8    965353
5 Female  137 132 134    147   65.0    951545
6 Female   99  90 110    146   69.0    928799
```

Il set di dati riporta, per ogni soggetto (righe), il genere, il QI complessivo nella scala Wechsler (full scale IQ, FSIQ), il QI nella sottoscala verbale, il QI nella sottoscala di prestazione, il peso in libbre, l'altezza in pollici, e una misura delle dimensioni del cervello. Quest'ultima misura è stata ricavata esaminando, per ogni partecipante, 18 immagini di risonanza magnetica e contando il numero di pixel indicanti presenza di materia grigia. I dati sono ricavati da una ricerca[3] il cui scopo era valutare se il punteggio al test di intelligenza dipende dalla grandezza del cervello.

Esaminiamo la struttura del set di dati:

```
> str(d)

'data.frame':        40 obs. of  7 variables:
 $ Gender   : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 1 1 1 2 2 ...
```

[3] Willerman, L., Schultz, R., Rutledge, J. N., and Bigler, E. (1991), In Vivo Brain Size and Intelligence, *Intelligence*, 15, 223-228. Nei dati originali l'altezza e il peso di alcuni soggetti non sono stati registrati (sono missing values). Per semplicità di trattazione nel database questi valori mancanti sono stati sostituiti con la media della variabile.

```
 $ FSIQ     : int  133 140 139 133 137 99 138 92 89 133 ...
 $ VIQ      : int  132 150 123 129 132 90 136 90 93 114 ...
 $ PIQ      : int  124 124 150 128 134 110 131 98 84 147 ...
 $ Weight   : int  118 151 143 172 147 146 138 175 134 172 ...
 $ Height   : num  64.5 72.5 73.3 68.8 65 69 64.5 66 66.3 68.8 ...
 $ MRI_Count: int  816932 1001121 1038437 965353 951545 928799
991305 854258 904858 955466 ...
```

Il data frame contiene 7 variabili, come sapevamo, misurate su 40 casi. La tipologia di dato è specificata dopo il nome: Gender è un fattore, i QI sono numeri interi, l'altezza è un numero reale e così via. in questa sezione ci concentreremo solo sulle variabili Gender e Height.

3.1 Distribuzioni di una variabile

La distribuzione di una variabile è la lista dei valori che la variabile può assumere, associati alle loro frequenze relative. Per studiare la distribuzione di una variabile R mette a disposizione diversi strumenti.

3.1.1 Distribuzione di una variabile nominale

Iniziamo dalla variabile Gender. Gender è un variabile nominale o categoriale (un fattore nella terminologia di R). Per avere un'idea della distribuzione possiamo scrivere

```
> summary(d$Gender)

Female   Male
    20     20
```

la tabella presenta le frequenze assolute associate ai due valori che la variabile può assumere. Lo stesso risultato si ottiene con la funzione table().

```
> table(d$Gender)

Female   Male
    20     20
```

Sia summary() sia table() sono funzioni generiche, che producono risultati diversi a seconda dell'argomento. In particolare, table() può essere usata anche per costruire tabelle di contingenza. Ad esempio, aggiungiamo al dataframe un'altra colonna con un'ulteriore classificazione fittizia

```
> d$add <- rep(1:2, 20)
```

la funzione rep() ha due argomenti, il vettore da ripetere (in questo caso, i numeri interi 1 e 2) e il numero di ripetizioni:

```
> head(d)

  Gender FSIQ VIQ PIQ Weight Height MRI_Count add
1 Female  133 132 124    118   64.5    816932   1
2   Male  140 150 124    151   72.5   1001121   2
3   Male  139 123 150    143   73.3   1038437   1
4   Male  133 129 128    172   68.8    965353   2
5 Female  137 132 134    147   65.0    951545   1
6 Female   99  90 110    146   69.0    928799   2
```

per ottenere la tabella a doppia entrata scrivere

```
> table(d$Gender, d$add)
```

R risponde:

```
            1  2
  Female 13  7
  Male      7 13
```

Vedremo a breve altre maniere di usare summary(). Tornando alla distribuzione di Gender, per ottenerla dobbiamo ancora convertire le frequenze assolute in frequenze relative. In questo caso è sufficiente utilizzare la funzione length(), che calcola il numero di elementi nel vettore:

```
> table(d$Gender)/length(d$Gender)
```

il risultato è la distribuzione di Gender, i valori della variabile associati alle frequenze relative:

```
Female   Male
   0.5    0.5
```

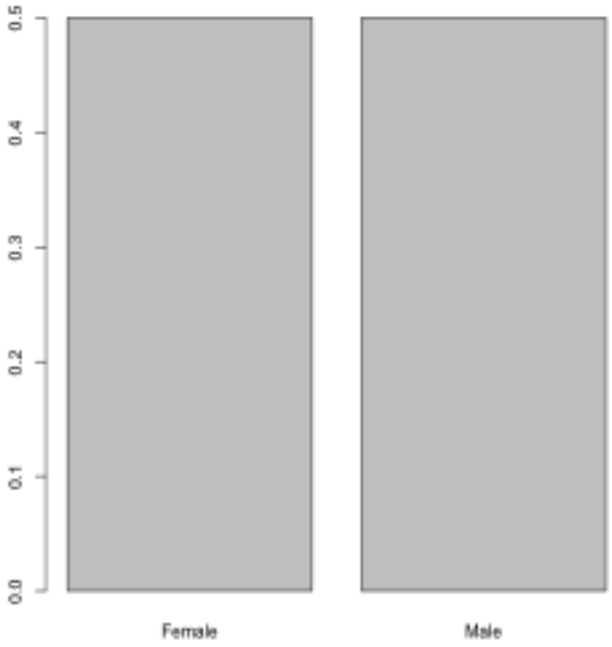
In alternativa, è possibile usare la funzione di R prop.table():

```
> prop.table(table(d$Gender))
```

che produce lo stesso risultato. Ma la maniera più efficiente di studiare la distribuzione di una variabile è fare un grafico. Normalmente, si inizia guardando il grafico e solo dopo, se necessario, si calcolano statistiche

esatte. Per mettere in grafico la distribuzione di Gender possiamo usare la funzione barplot():

```
> barplot(table(d$Gender)/length(d$Gender))
```



qui attenzione a non fare confusione: tecnicamente, il grafico è un istogramma della distribuzione di Gender. Infatti il grafico presenta le frequenze relative di R usando la convenzione per cui l'area totale = 1. In R esistono diverse maniere di costruire un istogramma. Ad esempio, con la funzione hist() che vedremo a breve. Questa funzione tuttavia è costruita per lavorare su vettori numerici, raggruppando i valori del vettore in classi. Provate ad esplorare i grafici ottenibili sostituendo hist() a barplot() nell'istruzione precedente. Per come è fatto R, per ottenere un istogramma relativo ad una variabile nominale è meglio utilizzare la funzione pensata per costruire un diagramma a barre.

In alternativa all'istogramma, la distribuzione di una variabile categoriale può essere visualizzata usando un diagramma a torta, sempre a partire dalla tabella:

```
> pie(table(d$Gender))
```

anche in questo caso, l'area totale corrisponde al totale dei casi.

3.1.2 Distribuzione di una variabile numerica

Per studiare la distribuzione di Height si procede più o meno come per Gender. R riconosce che la variabile in questo caso è numerica e modifica il suo comportamento nella maniera appropriata. Scrivere

```
> summary(d$Height)
```

R risponde con informazioni sulla distribuzione della variabile. Per la precisione, ci dice quali sono il minimo, il massimo, il primo, il secondo (la mediana) e il terzo quartile della distribuzione. Prendendo in esame il risultato ci facciamo un'idea abbastanza precisa della distribuzione. Ne conosciamo la gamma: l'altezza dei soggetti va da circa un metro e mezzo a quasi due metri. Sappiamo che la metà supera il metro e 73 cm, e che metà dei soggetti è compreso fra 1.68 m e 1.79 m.

Una maniera molto simile di riassumere la distribuzione è data dalla funzione fivenum(), che calcola il cosiddetto five-number-summary (riassunto basato su cinque numeri). I cinque numeri in questione sono il minimo e il massimo, la mediana, il lower hinge e l'upper hinge (letteralmente, il cardine inferiore e superiore, come i cardini a cui è appesa una porta). Gli hinge sono molto simili ai quartili, ma non sempre esattamente uguali: il lower hinge è la mediana del primo 50% dei dati, mentre l'upper hinge è la mediana del secondo 50%.

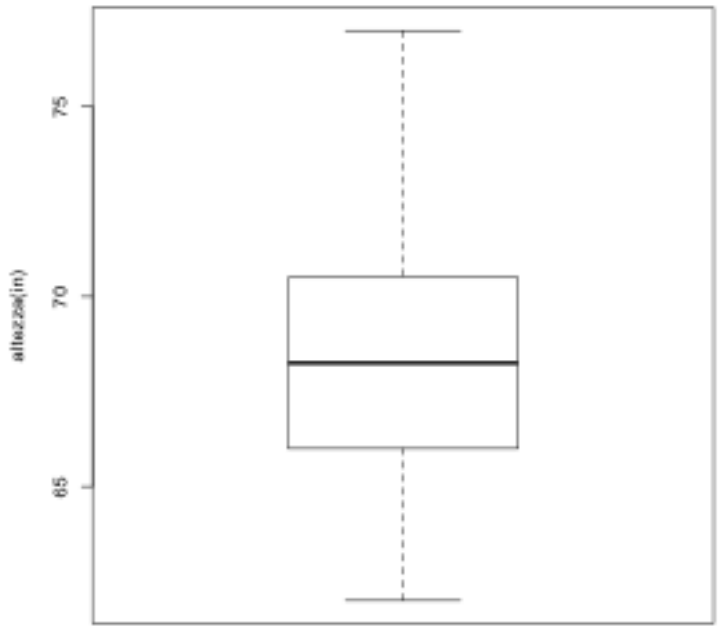
Il concetto di five-number-summary è stato inventato da John Tukey[4], una delle figure più influenti della statistica applicata moderna. Per visualizzare il five-number-summary con un grafico, Tukey ha anche inventato il boxplot.

In un boxplot, la "scatola" (box) centrale marca il 50% centrale della distribuzione (dal primo al terzo quartile), la linea in grassetto al centro rappresenta la mediana, e i due baffi esterni sono il minimo e il massimo della distribuzione.

Ci sono molte funzioni di R che usano il boxplot. Nel pacchetto di grafica standard, quello che viene caricato automaticamente all'installazione, la funzione si chiama appunto boxplot():

```
> boxplot(d$Height, ylab = "altezza(in)")
```

4 Tukey, J. W. (1977) *Exploratory data analsys.* Boston: Addison-Wesley.



La distribuzione sembra ragionevolmente simmetrica, anche se il baffo superiore è decisamente più alto di quello inferiore. Tuttavia dobbiamo notare che non abbiamo ancora una rappresentazione completa della distribuzione, ma un suo riassunto (appunto, usando cinque indici di posizione). Se ci interessano le frequenze relative di tutti i casi, possiamo, in analogia a quello che abbiamo fatto con Gender, usare table():

```
> table(d$Height)/length(d$Height)

   62  62.5    63  64.5    65    66  66.3  66.5    67    68  68.5
0.025 0.025 0.050 0.100 0.025 0.050 0.050 0.075 0.025 0.075 0.025
 68.8    69    70  70.5    72  72.5  73.3  73.5    74  75.5  76.5
0.050 0.100 0.050 0.050 0.025 0.025 0.025 0.025 0.025 0.050 0.025
   77
0.025
```

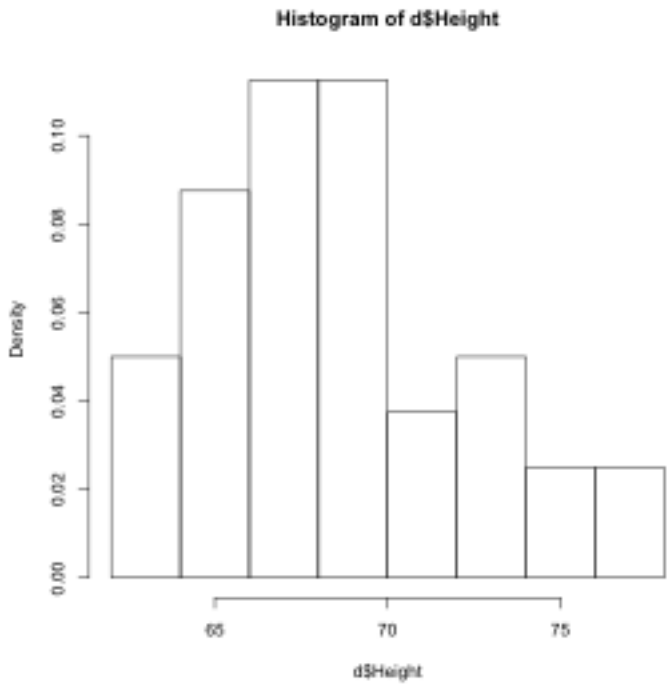
Ma da una tabella come questa non ci rendiamo veramente conto della forma della distribuzione. Anche se ci sono solo 40 casi, sono comunque già troppi per potersi fare un'idea ispezionando i numeri. Facciamo dunque quello che avremmo dovuto fare come prima cosa, costruire un istogramma.

```
> hist(d$Height, freq = 0)
```

In un istogramma, l'asse delle y non riporta la frequenza della variabile, ma la densità. Infatti l'istogramma rappresenta la distribuzione della variabile secondo la convenzione che l'area totale dell'istogramma rappresenta il 100% casi, ossia ha densità pari a 1. L'area di ogni singolo rettangolo rappresenta la densità della classe corrispondente. Le classi sono rappresentate sull'asse delle x.

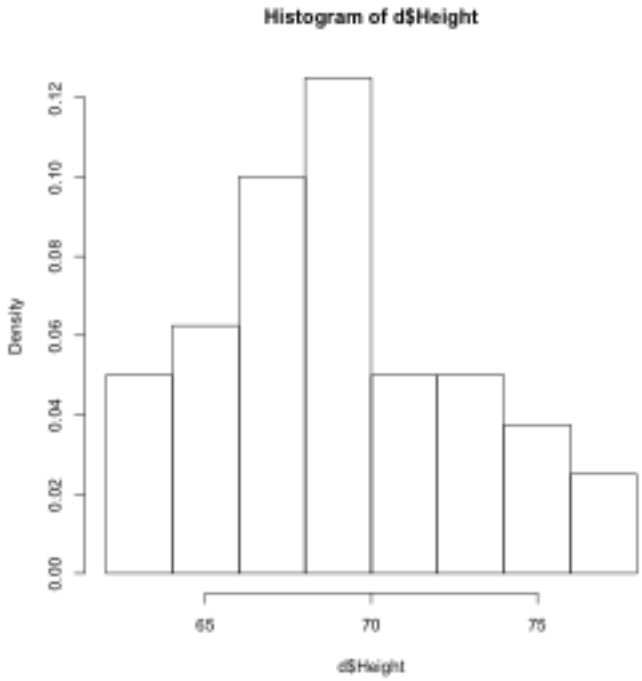
Per disegnare l'istogramma useremo la funzione hist(), che è uno degli strumenti grafici per gli istogrammi (per la precisione, quello nel pacchetto preinstallato chiamato graphics). Di default, hist usa le frequenze, non la densità, sull'asse delle y. Per ottenere un vero e proprio istogramma passeremo alla funzione l'argomento freq, specificando che non le vogliamo. In questo caso non fa una grande differenza (controllare), ma vedremo più avanti altri esempi in cui la fa.

```
> hist(d$Height, freq = FALSE)
```



Una domanda importante, studiando un istogramma di una variabile numerica, è quale convenzione venga usata per definire le le classi. Di default, R usa la convenzione del limite destro. Ossia, ogni intervallo di classe comprende il valore superiore, ma non quello inferiore (eccetto che per il valore minimo, che viene incluso nella prima classe).

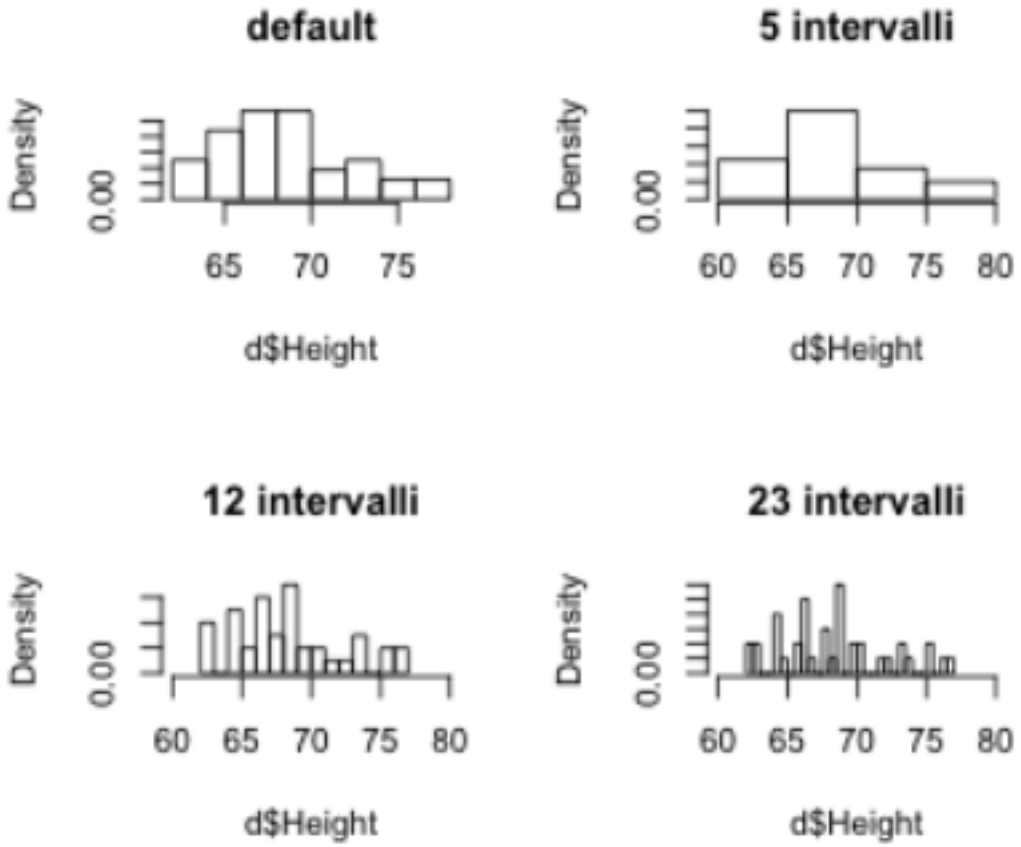
```
> hist(d$Height, freq = FALSE, right = FALSE)
```



notate che l'istogramma ha la stessa forma complessiva, leggermente asimmetrica, ma è cambiato nei dettagli. R ha costruito gli intervalli di classe in maniera diversa, e la rappresentazione è cambiata leggermente.

Un'altra cosa da notare è che R ha suddiviso le nostre 40 osservazioni in 8 intervalli di classe. Se osservate la tabella che abbiamo costruito poco fa vi potete rendere conto che i valori possibili sono in realtà 23. Raggruppandoli in otto classi, R ha "riassunto" la distribuzione, omettendo alcuni dettagli. Proviamo a scrivere un frammento di programma per controllare cosa succede modificando il numero di classi. Aprite l'editor, inserite i comandi sotto, e salvate col nome fourhist.R:

```
par(mfrow=c(2,2))
hist(d$Height, freq = 0, main = "default")
hist(d$Height, freq = 0, breaks = 5, xlim = c(60,80), main = "5
intervalli")
hist(d$Height, freq = 0, breaks = 12, xlim = c(60,80), main =
"12 intervalli")
hist(d$Height, freq = 0, breaks = 23, xlim = c(60,80), main =
"23 intervalli")
```



Notate che abbiamo utilizzato la funzione par() per dividere la finestra di output in quattro parti uguali, e l'argomento xlim per modificare i limiti dell'asse x che altrimenti non comprendevano tutti i valori possibili. Notate anche che R non ha suddiviso esattamente le distribuzioni nel numero di classi che abbiamo specificato. R valuta se l'argomento che abbiamo pssato è ragionevole, e cerca comunque di suddividere la distribuzione in modo da avere classi rappresentative. I titoli dei grafici dovrebbero essere "circa" 5 classi, "circa" 12 classi, e così via. Per ora non approfondiamo questi due aspetti, che riprenderemo nella sezione finale sulla grafica in R.

Di deafult, la funzione hist() di R usa la cosiddetta "regola di Sturges"[5]. La maggior parte dei pacchetti statistici utilizza questo algoritmo per calcolare il numero di classi ottimale quando viene costruito un istogramma. La validità generale dell'algoritmo tuttavia è stata messa in dubbio dalla ricerca statistica più recente, soprattutto quando i dati sono poco numerosi. R consente di scegliere anche altre regole, ma in generale è preferibile un approccio per prove ed errori. Quando studiamo un istogramma, è sempre una buona idea provare a modificare il numero di classi e confrontare i risultati per identificare quello che sembra dare la rappresentazione più

[5] Sturges, H. (1926) The choice of a class-interval. *J. Amer. Statist. Assoc.,* 21, 65–66.

fedele sia della forma complessiva sia dei dettagli che ci interessano. Di solito se la numerosità del campione è abbastanza alta non occorre preoccuparsene.

3.2 Misure di tendenza centrale

Calcolare le misure di tendenza centrale per una distribuzione (ovviamente, numerica) è relativamente semplice. Abbiamo già visto che la funzione summary() include la mediana, oppure possiamo scrivere direttamente

```
> median(d$Height)

[1] 68.25
```

Per trovare la media aritmetica, scrivere

```
> mean(d$Height)

[1] 68.5375
```

la funzione mean() naturalmente è equivalente a scrivere

```
> sum(d$Height)/length(d$Height)
```

Per quanto riguarda gli altri tipi di media, R non ha una funzione per calcolare la media quadratica, ma basta ricordare che la media quadratica è la radice quadrata della media aritmetica dei dati al quadrato:

```
> sqrt(mean(d$Height^2))

[1] 68.64804
```

Perlomeno nei package preistallati, R non ha neanche funzioni per la media geometrica e la media armonica, ma anche in questo caso il calcolo non è difficile.

Per la media geometrica, basta ricordare che la media geometrica è la radice n-esima del prodotto degli n dati:

```
> prod(d$Height)^(1/length(d$Height))

[1] 68.42843
```

in alternativa, possiamo anche sfruttare il fatto che la media geometrica è

l'antilogaritmo dellla media aritmetica dei logaritmi dei dati:

```
> exp(mean(log(d$Height)))

[1] 68.42843
```

Per la media armonica, basta ricordare che la media armonica è il reciproco della media dei reciproci dei dati:

```
> 1/mean(1/d$Height)

[1] 68.32095
```

Infine, se proprio non volete fare i calcoli, è possibile installare il pacchetto psych che include le funzioni geometric.mean() e harmonic.mean().

Per trovare la moda di un vettore numerico, infine, è sufficiente cercare il valore o i valori massimi delle frequenze calcolate da table(), sfruttando la flessibilità con cui R consente di specificare gli indici di un vettore:

```
> t <- table(d$Height)
> t[t==max(t)]

64.5   69
  4    4
```

notate l'uso dell'operatore logico ==, che identifica i casi in cui il valore di t è uguale a max(t). In questo caso, la distribuzione ha tecnicamente due mode.

3.2 Misure di dispersione

Anche calcolare le misure di dispersione per una distribuzione (ovviamente, numerica) è semplice.

La gamma è il più piccolo intervallo che contiene tutti i dati, ossia la distanza fra il massimo e il minimo della distribuzione. Quindi in R la gamma è data da

```
> max(d$Height) - min(d$Height)

[1] 15
```

Volendo controllare il risultato, Il massimo e il minimo sono visualizzabili usando summary() o direttamente con la funzione range():

```
> range(d$Height)

[1] 62 77
```

Analogamente per la gamma interquartile

```
> quantile(d$Height, .75) - quantile(d$Height, .25)

75%
4.5
```

che può essere calcolata anche con la funzione IQR()

```
> > IQR(d$Height)

[1] 4.5
```

Per calcolare la deviazione standard e la varianza,

```
> sd(d$Height)

[1] 3.943816

> var(d$Height)

[1] 15.55369
```

consultando l'help in linea, possiamo verificare che R calcola la deviazione standard campionaria, ponendo al denominatore n -1. Infatti

```
> sqrt(sum((d$Height - m)^2)/(length(d$Height)-1))

[1] 3.943816
```

produce lo stesso risultato di sd(), cosa che avremmo potuto verificare anche usando un operatore logico:

```
> sd(d$Height) == sqrt(sum((d$Height - m)^2)/(length(d$Height)-1))

[1] TRUE
```

Infine, calcoliamo la deviazione mediana assoluta (median absolute deviation), l'indice di dispersione più robusto in caso di distribuzioni

fortemente asimmetriche:

```
> mad(d$Height)
[1] 3.33585
```

4. Cambio di scala e trasformazione dei dati

In alcuni casi, dopo una prima esplorazione della distribuzione di una variabile emergono aspetti che suggeriscono l'opportunità di modificare il dato per renderlo più facilmente interpretabile o analizzabile. Qui esaminiamo una modifica banale ma talvolta molto utile, quello in cui viene modificata la scala dei dati, e una modifica molto più delicata, quello in cui viene effettuata una vera e propria trasformazione della distribuzione.

4.1 Cambio dell'unità di misura

Nelle prime sezioni di questa parte utilizzeremo ancora il database IQ.txt. Se non avete chiuso o avete salvato la sessione di R dovreste averlo ancora nel workspace, altrimenti dovete caricarlo nuovamente.

Se i dati sono una quantità, hanno un'unità di misura. Non è detto che questa sia sempre quella più adatta. Ad esempio, nel database IQ.tx l'atezza è espressa in pollici e il peso in libbre. Dato che siamo in Europa, sarebbe opportuno esprimere queste quantità in metri e kilogrammi. Un metro = 39.37 pollici e un kilogrammo = 2.205 libbre. Pertanto il cambio di unità di misura si fa in questo modo:

```
> hm <- d$Height/39.37
> wk <- d$Weight/2.205
```

Tenete presente che cambiando l'unità di misura abbiamo cambiato anche la media e le altre statiche descrittive, anche se il significato del numero è lo stesso. Ad esempio

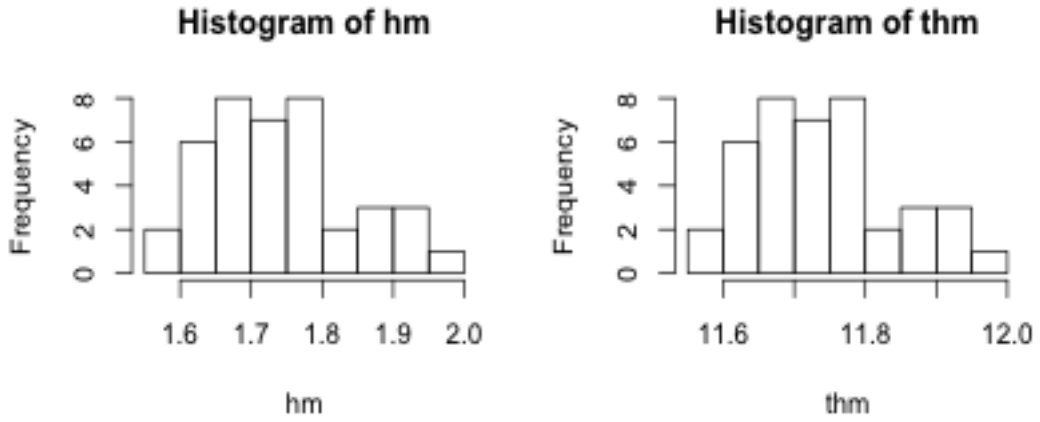
```
> mean(hm)

[1] 1.740856
```

4.2 Traslazione dei dati

In alcuni casi è utile spostare i dati in avanti o indietro sull'asse. Questa traslazione corrisponde a sommare o sottrarre una costante dal vettore. Ad esempio:

```
> thm <- hm + 10
> par(mfrow = c(1,2))
> hist(hm)
> hist(thm)
```

La traslazione sposta l'istogramma, sull'asse delle x, in avanti di 10 metri:



In questo modo abbiamo cambiato il valore della media e di tutti gli indici di posizione (si sono spostati in avanti di 10 metri):

```
> mean(thm) == mean(hm) + 10

[1] TRUE
```

ma non abbiamo cambiato la forma della distribuzione, come si vede dagli istogrammi. Infatti tutti gli indici di dispersione rimangono uguali a prima:

```
> round(sd(thm, 5)) == round(sd(hm, 5))

[1] TRUE
```

notate che in questo caso abbiamo dovuto arrotondare le deviazioni standard, perché l'operatore logico è sensibile anche a minimi errori di approssimazione. Provate a controllare cosa succede se non usiamo round().

4.3 Standardizzazione

Un cambio di scala molto utile è la standardizzazione, o trasformazione in unità standard o punti z. Standardizzare una variabile significa cambiarne la scala in modo che l'unità sia uguale alla devizione standard, e la media sia uguale a 0. Per ottenere questo risultato bisogna sottrarre ogni dato dalla media e dividerlo per la deviazione standard:

```
hz <- (d$Height - mean(d$Height))/sd(d$Height)
```

R ha una funzione che fa questo direttamente:

```
> hz1 <- scale(d$Height)
```

mettiamo i due risultati su due righe in modo per verificare che il risultato sia lo stesso:

```
> head(cbind(hz, hz1[, 1]))
```

```
            hz
[1,] -1.0237546 -1.0237546
[2,]  1.0047375  1.0047375
[3,]  1.2075867  1.2075867
[4,]  0.0665599  0.0665599
[5,] -0.8969739 -0.8969739
[6,]  0.1172722  0.1172722
```

notate la maniera leggermente diversa di accedere al vettore. Mentre hz è semplicemente un secondo vettore, hz1 è un oggetto di classe list. Per i fini di questo corso non ci preoccuperemo di come accedere agli elementi di una lista.

La standardizzazione mette in evidenza la posizione di ogni dato rispetto alla distribuzione. Ad esempio, consideriamo il 40-esimo elemento di Height

```
> d$Height[40]
```

```
[1] 75.5
```

```
> hz[40]
```

```
[1] 1.765422
```

una persona alta 75.5 pollici è più alta della media di quasi due deviazioni standard, vale a dire, circa 8 pollici. Analogamente per il primo elemento:

```
> d$Height[1]
```

```
[1] 64.5
```

```
> hz[1]
```

```
[1] -1.023755
```

una persona alta 64.5 pollici è più bassa della media di circa una deviazione standard.

La standardizzazione è un'operazione molto utile quando vogliamo confrontare fra loro gruppi di dati raccolti in condizioni diverse. Un esempio tipico sono i voti agli esami. Lo stesso voto (ad esempio, 25) potrebbe rappresentare una prestazione molto buona in un esame difficile (ad esempio, se la media era 21 e la deviazione standard era 2, chi ha preso 25 è ben due deviazioni standard sopra la media), ma una prestazione scadente se l'esame era molto facile (ad esempio, se la media era 28). La standardizzazione rende i voti confrontabili, perché li esprime in unità di deviazione standard e come posizione rispetto alla media.

Utilizzando una matrice e la già vista (sezione 2.3) funzione apply() possiamo riprendere l'esempio del calcolo dei voti. Supponete che in un esame il voto finale sia la media di tre prove in itinere. Prima di attribuire i voti occorre renderle confrontabili. Scrivete ed eseguite il programma votiveri.R riportato sotto. Notare che questo programma è necessario che il computer sia collegato alla rete. Il programma leggerà il file direttamente dal sito dell'Università di Parma.

```
# lettura del file da www2.unipr.it
inp <- scan("http://www2.unipr.it/~brunic22/esame.txt",
                skip = 1,list(i1=0, i2=0, i3=0))
# matrice, ogni colonna è un parziale, ogni riga uno studente
m <- matrix(c(inp$i1, inp$i2, inp$i3), ncol=3)
zm <- scale(m)                  # cambio scala, 0=media, 1 = ds
mns <- apply(zm, 1, mean)       # medie per riga
# dalle medie (in unità standard) ai voti (18-30)
voti<-18+round((mns+(abs(min(mns))))/(max(mns)-min(mns))*12)
# output grafico
oldpar <- par(mfcol = c(1, 2))
boxplot(  m, names = c("1", "2", "3"),
          main= "prove in itinere",
          ylab= "punteggio",
          col= "grey"     )
hist(voti,
     breaks = 14,
     main = "risultati finali esame",
     xlab = "voto in 30esimi",
     ylab = "frequenza",
     col="darkred")
par(oldpar)
```

Studiando il programma avrete notato l'uso della funzione par(), che

abbiamo già visto. Si tratta di una funzione generica che controlla i parametri grafici utilizzati dai comandi successivi, in questo caso, la suddivisione della finestra grafica in "celle" in cui stampare i grafici. La funzione par() ha due argomenti con cui è possibile controllare questo aspetto, mfcol e mfrow. Guardate l'help in linea per la differenza. Dato che par() modifica i parametri grafici globali, è buona pratica una volta prodotto il grafico recuperare i parametri iniziali. Per fare questo il metodo standard sfrutta il fatto che ogni chiamata a par() produce un oggetto contente i vecchi parametri. Normalmente questo oggetto è "silente", nel senso che non viene stampato e non gli viene attribuito un nome. Ma se vogliamo possiamo scriverlo in un oggetto normale, in questo caso chiamato oldpar, che possiamo usare alla fine del programma in una seconda chiamata per rimettere tutto a posto.

4.4 Trasformazioni

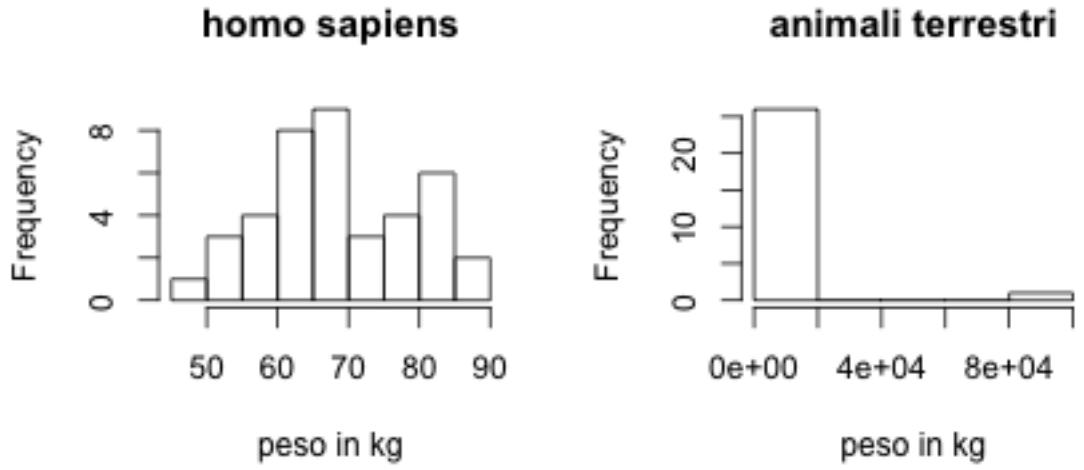
In alcuni casi, prima di analizzare i dati può essere utile modificare la forma stessa della loro distribuzione. In questo caso si parla di trasformazioni vere e proprie, anche se in senso tecnico un cambio di scala è anch'esso una trasformazione, e una trasformazione vera e propria comporta anche un cambio di scala. Ad esempio, la standardizzazione viene spesso chiamata trasformazione in punti z, e la trasformazione logaritimica, di cui ci occuperemo ora, è a tutti gli effetti un cambio della scala dei dati. In questa sezione useremo ancora il database IQ.txt ma ne aggiungeremo un'altro. Scaricate dal sito del docente il file BW.txt. Quindi scrivete

```
> dd <- read.table("BW.txt", header = TRUE)
```

Il database contiene tre variabili con 27 osservazioni. La variabile animal è un fattore e consiste nel nome di un animale terrestre, vivente (p.e. mucca) o estinto (p.e. diplodoco, un dinosauro del Giurassico). La variabile body è il peso del corpo in kg. La variabile brain è il peso del cervello in grammi. I dati provengono da un corso di statistica della University of Illinois - Urbana Champaign[6]. Per ora ci occuperemo solo del peso del corpo. Scrivete:

```
> oldpar <- par(mfrow = c(1,2))
> hist(d$Weight, main = "homo sapiens", xlab = "peso in kg")
> hist(dd$body, main = "animali terrestri", xlab = "peso in kg")
> par(oldpar)
```

[6] Rousseeuw, P.J. & Leroy, A.M. (1987) *Robust Regression and Outlier Detection.* Wiley, p. 57.



Stiamo esaminando le distribuzioni di frequenza di due campioni di pesi corporei. Il primo proviene da IQ.txt, e consiste nel peso di 40 quaranta individui adulti (uomini e donne). Il secondo rappresenta la distribuzione del peso corporeo di 27 animali terrestri diversi. La variabile è la stessa ma le distribuzioni sono drammaticamente diverse. Quella relativa alla specie homo è abbastanza simmetrica, anche se presenta due picchi, un picco principale attorno a circa 65 kg ed un secondo picco a circa 80 (su questo torneremo). La distribuzione degli animali terrestri è estremamente asimmetrica. Nella scala del grafico, la maggior parte degli animali è concentrata all'inizio dove ci sono i pesi relativamente bassi, ma ci sono alcuni animali con un peso molto grande nella coda a destra della distribuzione. Questo dipende dal fatto che i pesi corporei degli animali terrestri hanno un campo di variazione molto ampio, comprendente diversi ordini di grandezza. La variazione è così grande che R ha dovuto usare, per l'asse delle x, la notazione scientifica in cui i numeri sono espressi come potenze di 10: 1e+02 ad esempio vuol dire 1 moltiplicato per 10 alla seconda; 8e+03 vuol dire 8 moltiplicato per 10 alla terza; e così via. Proviamo a verificarlo usando R:

```
e > 2e2

[1] 200

> 2 * 10^2

[1] 200

> 2e2 == 2 * 10^2

[1] TRUE
```

Quando una quantità copre diversi ordini di grandezza quasi sempre la sua distribuzione è fortemente asimmetrica con coda sinistra. Nella ricerca psicobiologica questo avviene abbastanza di frequente. Pensate ad esempio alla distribuzione dei tempi di risposta in un compito cognitivo. Non è difficile immaginare che questi tempi siano tipicamente di alcune centinaia di millisecondi, ma possano arrivare in alcuni casi anche ad alcuni secondi e, in casi più rari, anche a decine di secondi. Ci dobbiamo aspettare che la distribuzione comprenda almeno due e probabilmente anche tre ordini di grandezza. In questi casi, l'asimmetria della distribuzione rende problematico l'utilizzo delle tecniche statistiche standard, a molti livelli. Per quanto riguarda l'uso delle statistiche descrittive, ad esempio, implica che la media aritmetica non è rappresentativa della tendenza centrale della distribuzione, ossia non rappresenta una caso tipico, a causa dell'influenza dei casi estremi sul suo calcolo. Ad esempio, troviamo la media del peso degli animali terrestri:

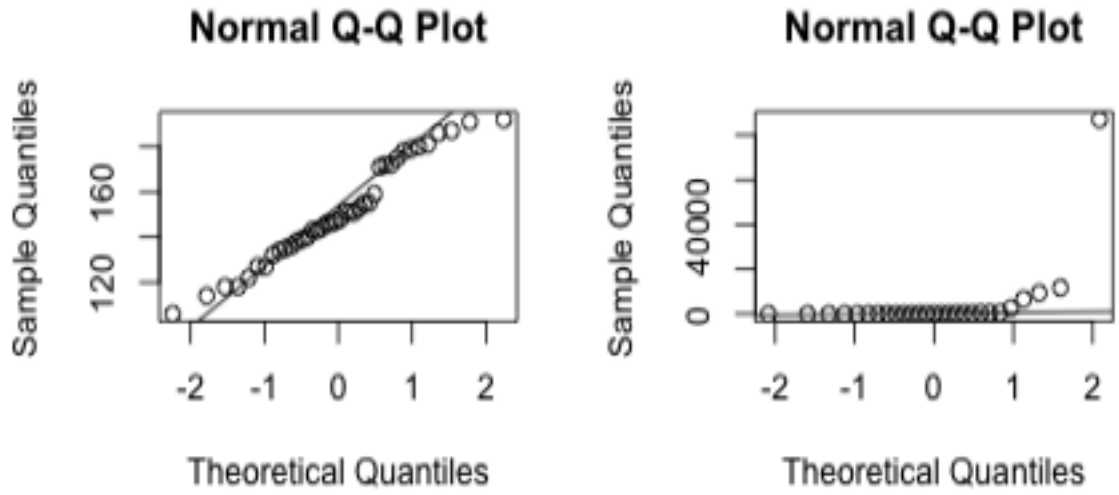
```
> mean(dd$body)

[1] 4436.889
```

la media è più di 4000 kilogrammi. Ma questo è un peso tutt'altro che comune. Confrontiamo con la mediana:

```
> median(dd$body)

[1] 55.5
```

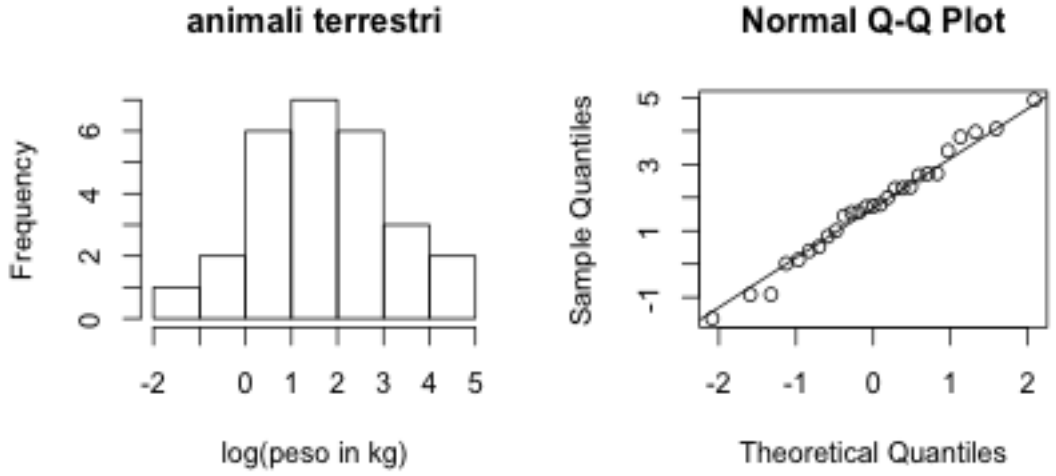
la metà degli animali considerati pesa meno di 55.5 kg. Il problema è particolarmente serio perchè le tecniche inferenziali standard della statistica assumono invece che i dati siano distribuiti in modo simmetrico e unimodale, abbiano cioè quella che viene chiamata la distribuzione normale o gaussiana. R offre diversi strumenti per valutare lo scostamento di una distribuzione da una normale teorica. Uno di questi è la funzione qqnorm(), che crea un grafico dei quantili della distribuzione in funzione dei quantili teorici normali (qq sta per quantile-quantile). Vediamola all'opera:

```
> qqnorm(d$Weight)
> qqline(d$Weight)
> qqnorm(dd$body)
> qqline(dd$body)
```



I grafici evidenziamo molto bene che la distribuzione campionaria è ragionevolmente vicina ai quantili normali teorici (vedi a retta in diagonale) per il campione di 40 homo sapiens, ma diversissima da questi nel caso dei 27 animali terrestri. Almeno in un caso come questo, fortunatamente, esiste una soluzione. Quando l'asimmetria di una distribuzione dipende dal fatto che una variabile si estende su diversi ordini di grandezza, di solito trasformare la variabile nel suo logaritmo rende la distribuzione simmetrica e molto simile alla normale. Proviamo a verificarlo con R:

```
> oldpar <- par(mfrow = c(1,2))
> hist(log10(dd$body), main = "animali terrestri", xlab = "log(peso in kg)")
> qqnorm(log10(dd$body))
> qqline(log10(dd$body))
> par(oldpar)
```



Appare evidente che lo studio della distribuzione e il calcolo di statische descrittive risulta più interessante se fatto su log(peso) piuttosto che sul peso originale. Questo pone tuttavia alcui problemi. Ad esempio, se voglio riassumere la tendenza centrale della distribuzione, potrei avere la tentazione di calcolare

```
> mean(log10(dd$body))

[1] 1.718994
```

il risultato è certamente al centro della distribuzione simmetrica, come si vede dall'istogramma. Ma avendo perso l'unità di misura di peso, la sua interpretazione non è chiara. Qual è un peso tipico in questo campione di animali? Tipico in che senso? Per recuperare l'unità di misura, posso calcolare l'antilogaritmo

```
> 10^1.718994

[1] 52.35932
```

che è di nuovo un numero con una unità di misura: tenuto conto della distribuzione logaritmica, il peso tipico sarebe circa 52 kg, che non è troppo diverso dalla mediana. Questo peso tipico tuttavia non è la media aritmetica della variabile, ma la sua media geometrica. Infatti, come abbiamo già calcolato in precedenza

```
> mean(dd$body)

[1] 4436.889
```

mentre (notate il problema di approssimazione al quinto decimale, il calcolo della radice n-esima non è banale)

```
> prod(dd$body)^(1/length(dd$body))

[1] 52.35934
```

Sono disponibili molti altri tipi di trasformazione, che possono essere appropriati nel caso di distribuzioni asimmetriche o in presenza di dati con certe caratteristiche. In generale, il tema delle trasformazioni è un tema delicato, perché pone il problema della interpretazione delle analisi sui dati trasformati. In questo corso ci limitiamo quindi solo al caso trattato, la cui interpretazione è chiara: per dati come questi, la tendenza centrale è bene

riassunta dalla media geometrica, e anche per gli altri indici di posizione è opportuno operare nella stessa maniera, passando dal logaritmo alla misura originale. Per chi desiderasse approfondire il tema, rimando a un lavoro classico del già incontrato John Tukey[7].

7 Tukey, J. W. (1957) On the comparative anatomy of transformations. *Ann. Math. Statist.,* 28, 602-632.

5. Statistiche descrittive bivariate

In questa sezione vedremo come usare R per descrivere relazioni fra coppie di variabili. Esamineremo i tre casi possibili: la relazione fra due variabili nominali o categoriali, quella fra due variabili numeriche, e quella (tipica della ricerca psicologica) della relazione fra una variabile numerica e una categoria. Vedremo sia come misurare la forza dell'associazione, sia come fittare un modello che consenta di prevedere y in funzione di x.

5.1 Due variabili nominali

5.1.1. Distribuzione bivariata di categorie

Per studiare la distribuzione congiunta di due variabili nominali o categoriali possiamo esaminare una tabella di contingenza. Un evergreen della statistica applicata è l'associazione fra colore degli occhi e colore dei capelli. Creiamo una tabella con dati fittizi:

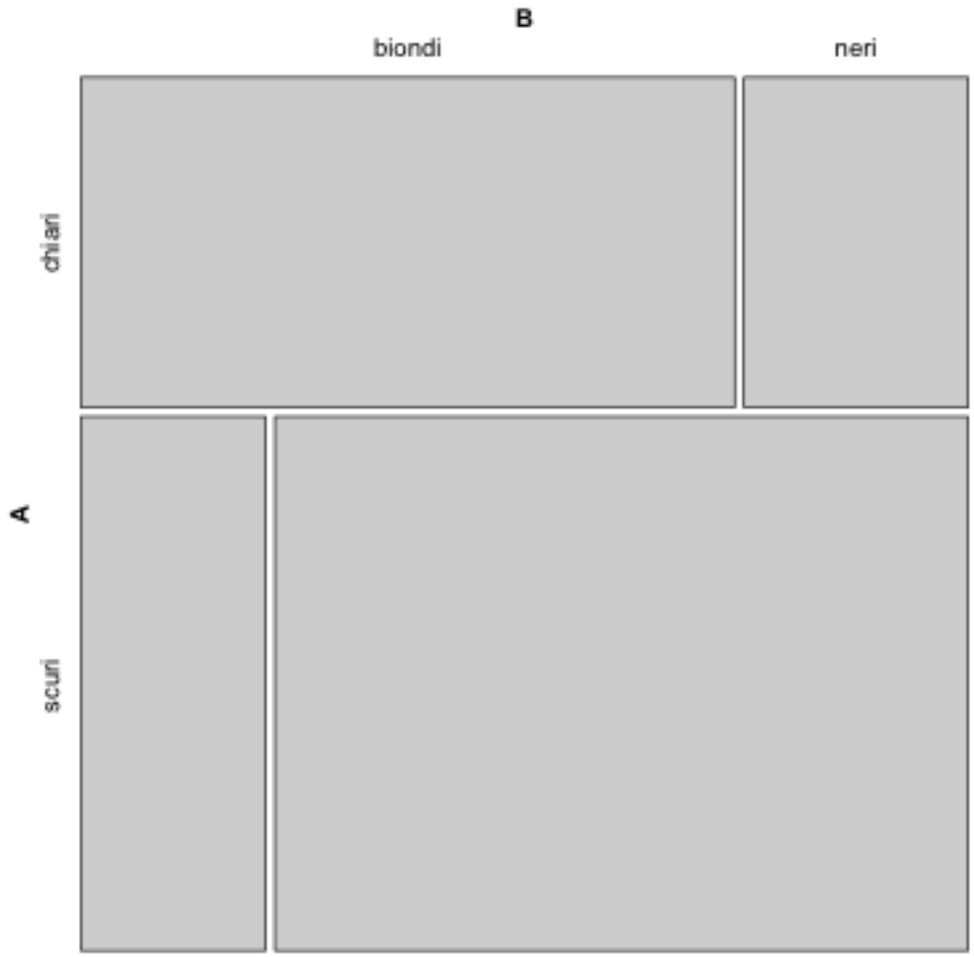
```
> eyehair <- matrix(c(35, 12, 16, 60), nrow = 2, byrow = TRUE)
> eye <- c("chiari", "scuri")
> hair <- c("biondi", "neri")
> dimnames(eyehair) <- list(eye, hair)
> eyehair

       biondi  neri
chiari     35   12
scuri      16   60
```

ci sono più persone con gli occhi scuri rispetto a chiari, e con capelli neri rispetto a biondi. Ma fra quelli che hanno capelli neri, è proporzionalmente più facile osservare occhi scuri, mentre fra quelli che hanno capelli biondi è più facile osservare occhi chiari.

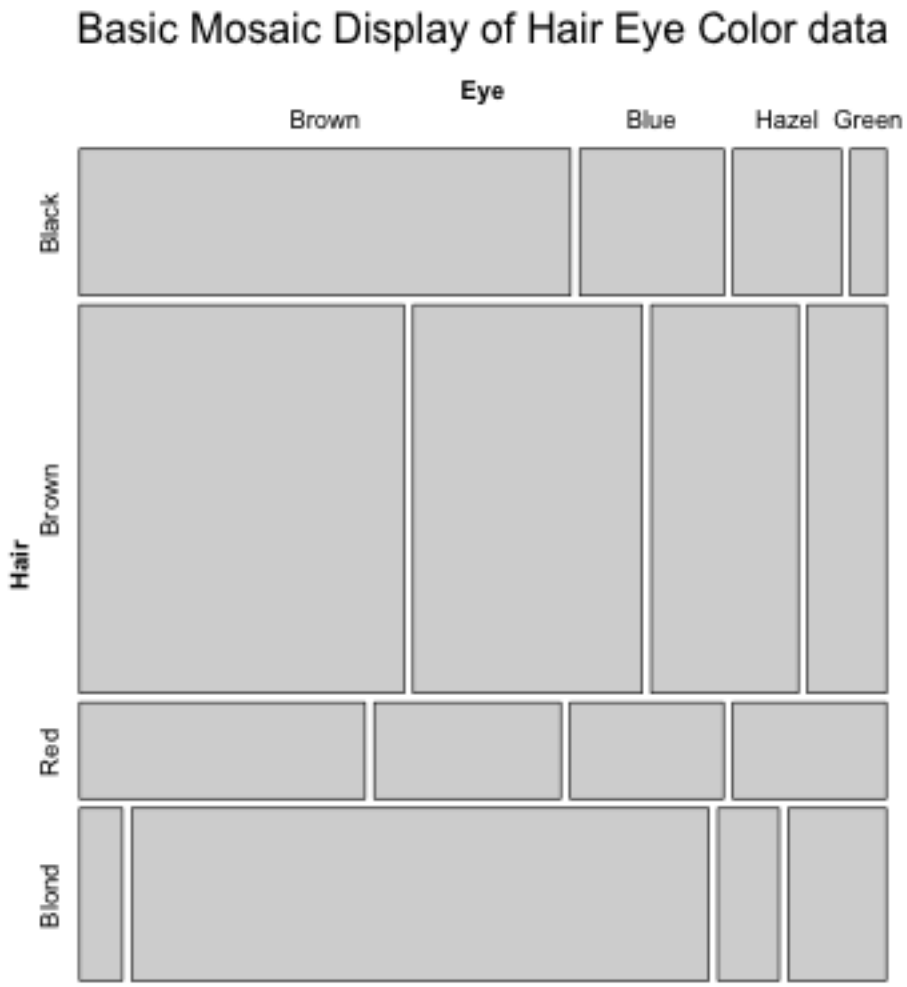
Ci potremmo chiedere se esiste, per la distribuzione bivariata di categorie, uno strumento grafico che consenta la visualizzazione in analogia al caso univariato. La risposta è positiva, si tratta del cosiddetto mosaic plot (grafico a mosaico, vedi[8]). Ad esempio, per la tabella di contingenza eyehair il mosaic plot è quello riportato sotto. Possiamo interpretare un grafico a mosaico come se fosse l'intersezione di due grafici a barre, uno in verticale ed uno in orizzontale. Le aree relative rappresentano la frequenza relativa all'interno di ogni cella della marice. Per creare un mosaic plot usando la grafica standard di R, possiamo usare la funzione mosaicplot():

[8] Friendly, M. (2000) *Visualizing Categorical Data*. SAS Institute, Cary, NC.

```
> mosaicplot(eyehair)
```



Usando uno dei pacchetti non standard di R, che si chiama vcd (visualization of categorical data), è possibile creare anche tabelle più complesse. Ad esempio, di seguito viene presentata una tabella in cui colore degli occhi e dei capelli sono distribuiti in maniera più realistica. Utilizzando ancora altre strategie è possibile mettere in grafico la relazione fra tre o più variabili, nonché calcolare direttamente diversi indici di associazione. Non entriamo nei dettagli, ma a chi fosse interessato ad approfondirlo suggerisco di scaricare vcd e provare a studiare la documentazione.



5.1.2 Associazione fra variabili categoriali

Per misurare l'associazione fra le variabili in una tabella 2 x 2 come quella creata nella sezione precedente, usiamo la statistica V di Cramér che ha valore 0 nel caso di perfetta indipendenza e valore 1 nel caso di perfetta associazione. Il V di Cramér misura l'associazione utilizzando gli scarti al quadrato fra le frequenze osservate e le frequenze attese, espressi come proporzione delle frequenze attese. Come abbiamo visto a lezione, la somma di questi scarti è la statistica chi-quadrato, mentre V è la radice quadrata di chi-quadrato, diviso per il numero totale di osservazioni moltiplicato per il numero di righe o di colonne (scegliere il più piccolo se sono diversi), meno uno. Per calcolare V devo quindi trovare innanzi tutto chi-quadrato, in questa maniera:

```
> chisq <- chisq.test(eyehair)
> chisq
```

```
        Pearson's Chi-squared test with Yates' continuity correction

data:  eyehair
X-squared = 31.9735, df = 1, p-value = 1.563e-08
```

La funzione chisq.test() non si limita a calcolare il valore di chi-quadrato ma calcola anche una probabilità. Per la precisione, si tratta della probabilità di osservare un valore come quello trovato, o maggiore, se è vera l'ipotesi nulla secondo cui non vi è associazione fra le due variabili. La probabilità osservata è estremamente piccola, il che può venire interpretato come una prova a sfavore della correttezza dell'ipotesi nulla, ossia di un risultato statisticamente significativo. La logica e l'interpretazione dei cosiddetti test di significatività sono oggetto del vostro secondo corso di analisi di dati e non verranno qui approfonditi, ma useremo occasionalmente funzioni come chisq.test() per il calcolo di statistiche che ci tornano utili, come nel caso del calcolo di V.

Avendo trovato chi-quadrato, per calcolare V dobbiamo trovare il numero totale di casi

```
> tot <- sum(eyehair)
> tot

[1] 123
```

e poi per calcolare V potremmo anche procedere con un taglia e incolla:

```
> sqrt(31.9735/123 * (2-1))

0.5098498
```

Ovviamente la moltiplicazione per 1 non era necessaria dato che la tabella è 2 x 2, e il taglia e incolla non va bene se volessimo scrivere la nostra funzione generale per V. Una soluzione più elegante, e più didattica rispetto al nostro scopo di prendere confidenza con R, consiste nel leggere il valore della statistica dentro all'oggetto creato dalla funzione chisq.test(). Diamogli un'occhiata:

```
> str(chisq)

List of 9
 $ statistic: Named num 32
  ..- attr(*, "names")= chr "X-squared"
 $ parameter: Named int 1
```

```
  ..- attr(*, "names")= chr "df"
 $ p.value  : num 1.56e-08
 $ method   : chr "Pearson's Chi-squared test with Yates' continuity
correction"
 $ data.name: chr "eyehair"
 $ observed : num [1:2, 1:2] 35 16 12 60
 $ expected : num [1:2, 1:2] 19.5 31.5 27.5 44.5
 $ residuals: num [1:2, 1:2] 3.51 -2.76 -2.96 2.33
 $ stdres   : num [1:2, 1:2] 5.84 -5.84 -5.84 5.84
 - attr(*, "class")= chr "htest"
```

l'oggetto è una lista, i cui elementi sono accessibili usando i nomi preceduti dal dollaro, in analogia a come abbiamo fatto per le colonne dei data.frame. Ad esempio, per il primo elemento (che è quello che ci interessa) posso scrivere

```
> chisq$statistic

X-squared
 31.97346
```

e poi basta scrivere

```
> V <- sqrt(chisq$statistic[[1]]/tot)
> V

[1] 0.5098498
```

Un'ultima cosa: fate caso alla notazione con la doppia parentesi per estrarre il valore del chi-quadrato senza includere anche il nome. Per capire meglio come funzionano le liste, è interessante provare a verificare cosa R mette nell'oggetto phi se si omette l'indice con la doppia parentesi, scrivendo

```
> phi <- sqrt(chisq$statistic/tot)
```

In ogni caso, il valore di V è circa 0.5, che indica un'associazione piuttosto forte fra le due variabili.

Naturalmente una variabile categoriale può avere anche più di due categorie. In effetti, il caso della matrice 2 x2 è un caso particolare, in cui la misura di associazione viene anche chiamata phi o phi quadrato di Cramér, mentre il nome V si usa per la statistica generale. Ad esempio, creiamo velocemente una matrice eyehair2 che ha 3 righe x 3 colonne. Non ci preoccupiamo di dare i nomi alle variabili per velocizzare l'operazione:

```
> eyehair2 <- matrix(c(35, 12, 11, 30, 16, 60, 80, 20, 15), nrow = 3,
byrow = TRUE)

> eyehair2

    [,1] [,2] [,3]
[1,]   35   12   11
[2,]   30   16   60
[3,]   80   20   15
```

per trovare V

```
> chisq <- chisq.test(eyehair2)
> k <- min(dim(eyehair2))
> n <- sum(eyehair2)
> V <- sqrt(chisq$statistic[[1]]/(n * (k -1)))
> V

[1] 0.3197222
```

Come conclusione di questa sezione, vi ricordo un paio di cose utili. Primo, una tabella a doppia entrata per variabili categoriali può essere ottenuta anche a partire dai dati in formato rettangolare standard, ossia in R da un data frame. In questo caso, la funzione da usare è table() che abbiamo già visto in azione nella sezione 3.1.1. Se studiate l'help relativo alla funzione chisq.test, vedrete però che la funzione accetta anche coppie di vettori da un data frame, e ne sa calcolare la tabella di contingenza. Secondo, nel caso voleste esaminare la tabella della frequenze attese, anche questa viene calcolata automaticamente da chisq.test. La trovate studiando l'oggetto prodotto dalla funzione. Ad esempio, nell'ultimo oggetto che abbiamo creato per eyehair2, e che avevamo chiamato di nuovo chisq:

```
> str(chisq)
List of 9
 $ statistic: Named num 57
  ..- attr(*, "names")= chr "X-squared"
 $ parameter: Named int 4
  ..- attr(*, "names")= chr "df"
 $ p.value  : num 1.21e-11
 $ method   : chr "Pearson's Chi-squared test"
 $ data.name: chr "eyehair2"
 $ observed : num [1:3, 1:3] 35 30 80 12 16 20 11 60 15
 $ expected : num [1:3, 1:3] 30.14 55.09 59.77 9.98 18.24 ...
```

```
 $ residuals: num [1:3, 1:3] 0.885 -3.38 2.617 0.64 -0.524 ...
 $ stdres   : num [1:3, 1:3] 1.434 -6.194 4.926 0.79 -0.731 ...
 - attr(*, "class")= chr "htest"
```

sono disponibilli una matrice chisq$observed, che contiene la matrice delle frequenze osservate che avevamo dato in input:

```
> chisq$observed
     [,1] [,2] [,3]
[1,]   35   12   11
[2,]   30   16   60
[3,]   80   20   15
```

e anche la matrice chisq$expected, con le frequenze attese calcolate come abbiamo visto a lezione:

```
> chisq$expected
     [,1]     [,2]     [,3]
[1,] 30.14337  9.978495 17.87814
[2,] 55.08961 18.236559 32.67384
[3,] 59.76703 19.784946 35.44803
```

5.2 Due variabili numeriche

In questa sezione ci occupiamo di come descrivere distribuzioni bivariate di variabili numeriche. Prenderemo in considerazione sia scale numeriche vere e proprie sia scale ordinali.

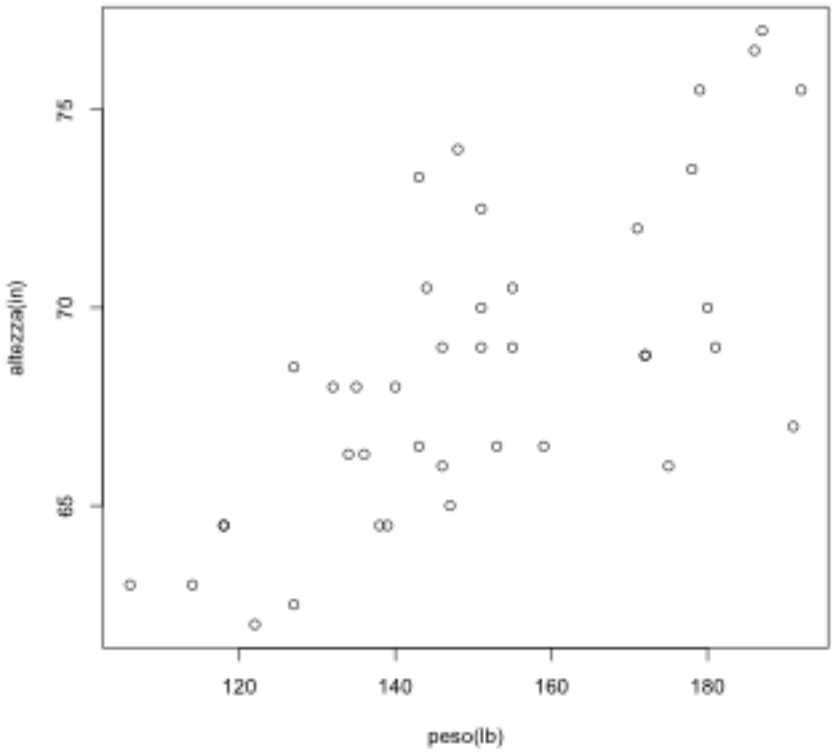
5.2.1 Distribuzione bivariata di variabili numeriche

Per questa sezione useremo nuovamente il dataset IQ. Usando objects() controllate se avete ancora il data frame chiamato d nel workspace. Se c'è non occorre fare nulla. In caso contrario occorre leggere di nuovo i dati:

```
> d <- read.table("IQ.txt", header = TRUE)
```

Per studiare la distribuzione bivariata di variabili numeriche lo strumento più utile è il diagramma di dispersione (scatterplot). Creiamo uno scatterplot dell'altezza in funzione del peso.

```
> plot(d$Weight, d$Height, xlab = "peso(lb)", ylab = "altezza(in)")
```



Per evidenziare la struttura dei dati in uno scatterplot è utile individuare il punto delle medie e tracciare la retta delle deviazioni standard[9], come abbiamo visto a lezione.

Vediamo come aggiungere al grafico le rette che individuano il punto delle medie. Per non appesantire il grafico tracciamo rette tratteggiate usando l'opzione lty (line type). La funzione da usare è segments, che ha la sintassi segments(x1, y1, x2, y2), dove x1, y1 sono le coordinate del punto di partenza e x2, y2 le coordinate del punto finale del segmento:

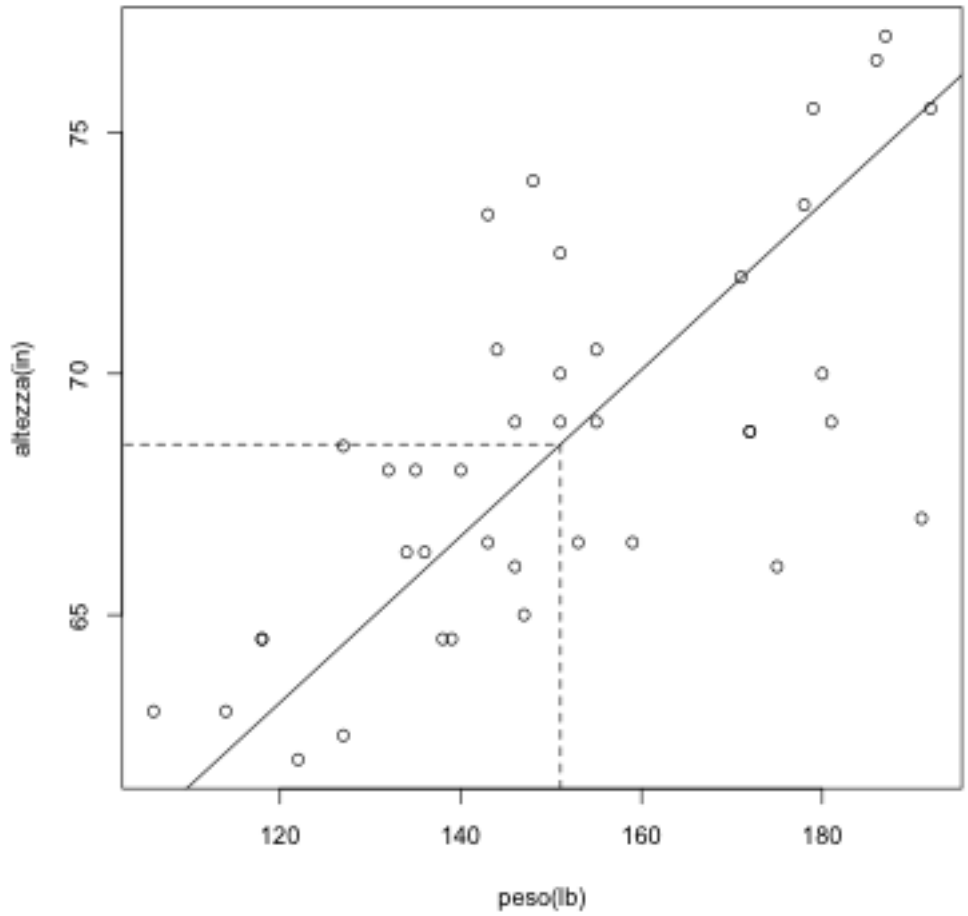
```
> segments (mean(d$Weight), mean(d$Height), mean(d$Weight),
min(d$Height) - 5, lty = "dashed")
> segments (min(d$Weight) - 5, mean(d$Height), mean(d$Weight),
mean(d$Height), lty = "dashed")
```

Infine, aggiungiamo la retta delle deviazioni standard. Per farla estendere su tutto il grafico, la tracciamo partendo da un punto 3 deviazioni standard sotto la media fino a un punto 3 deviazioni standard sopra la media, sia per x sia per y:

```
> x1 <- mean(d$Weight) - 3 * sd(d$Weight)
```

[9] Freedman, D., Pisani, R., Purves, R. (1998). *Statistica.* McGraw-Hill.

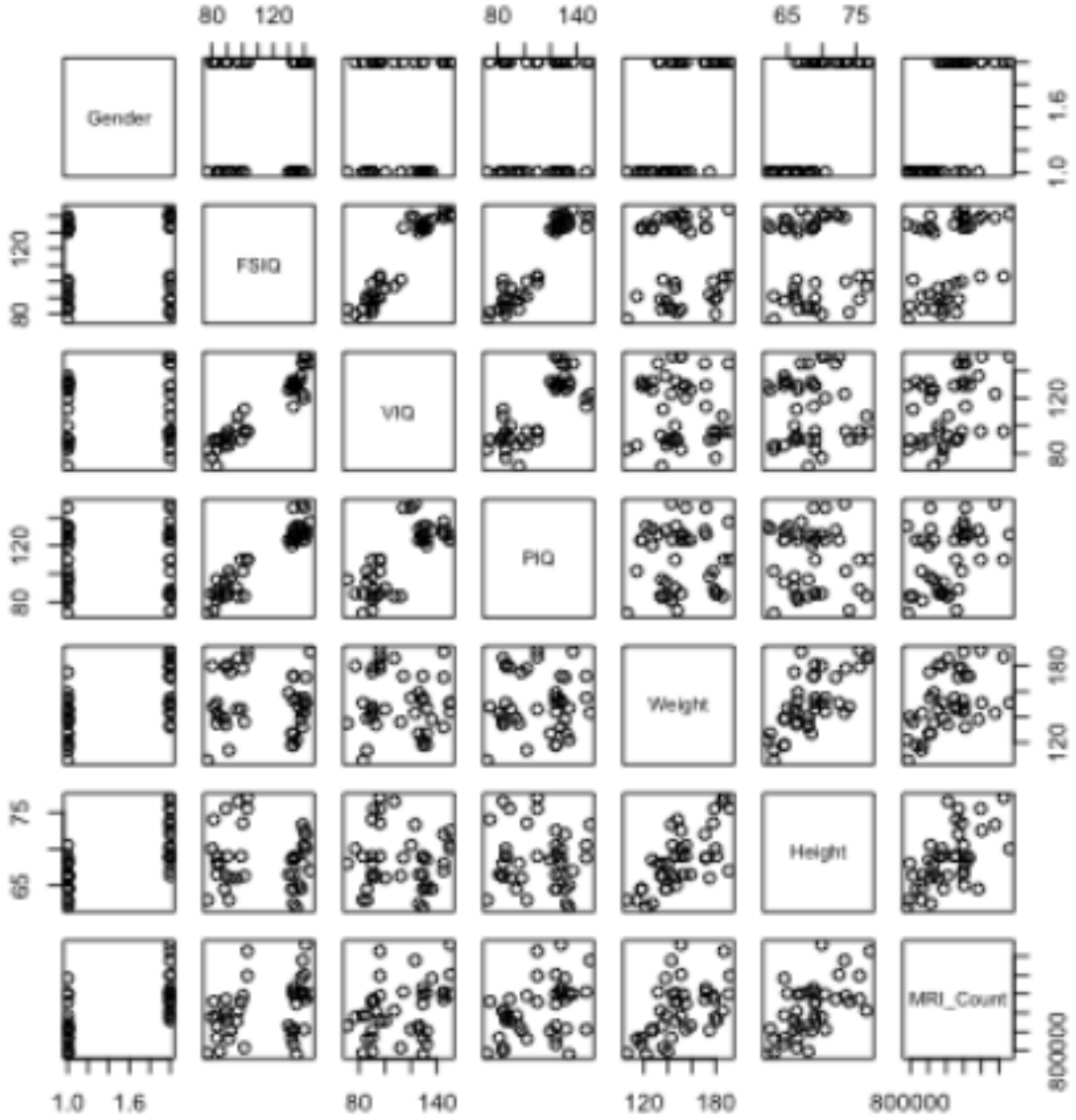
```
> x2 <- mean(d$Weight) + 3 * sd(d$Weight)
> y1 <- mean(d$Height) - 3 * sd(d$Height)
> y2 <- mean(d$Height) + 3 * sd(d$Height)
> segments (x1, y1, x2, y2)
```



Il data set contiene molte altre variabili di cui ci potrebbe interessare la distribuzione bivariata. Per dare un'occhiata rapida usando la funzione pairs() possiamo creare una matrice di diagrammi di dispersione, con tutte le coppie possibili. La funzione pairs() è un'ottima maniera di visualizzare la struttura complessiva di un set di dati, facendosi un'idea delle relazioni fra le diverse variabili. Scriviamo

```
> pairs(d)
```

Il risultato dovrebbe essere:



Esaminiamo le distribuzioni bivariate. In alcune, i dati sono bene raggruppati attorno alla diagonale, evidenziando una relazione monotonica e approssimativamente lineare fra le due variabili. In altre, i dati non si raggruppano per nulla, occupando tutto lo spazio del diagramma. In questo caso non sembra esserci una relazione fra le due variabili. In altre ancora, i dati si separano in due raggruppamenti separati. Nel caso dei diagrammi che coinvolgono la variabile categoriale Gender, il motivo è banale. La funzione pairs() ha trasformato Gender, un fattore, in due numeri (se guardate l'asse, sono 1 e 2). Il diagramma di dispersione non ha molto senso in questi casi. In altri casi il motivo non è così ovvio. Ad esempio, il diagramma di dispersione che plotta l'IQ full-scale (FSIQ) in funzione delle dimensioni del cervello (MRI_Count) mostra due raggruppamenti molto bene separati rispetto

all'ordinata. Per capire il motivo bisogna leggere l'articolo: i ricercatori hanno selezionato soggetti con un IQ full scale maggiore di 130 o minore di 103, quindi i dati nell'intervallo 103-130 mancano.

L'esame della distribuzione bivariata di variabili numeriche è importante anche perché ci consente di studiare non solo la presenza o meno di una associazione più o meno forte fra le due variabili, ma anche la natura di tale associazione. In particolare, è sempre interessante valutare se l'associazione è lineare e se può dunque essere sensatamente misurata con statistiche che misurano l'associazione lineare. Ad esempio, scaricate il file OIL.txt dal sito docente e create un data frame:

```
> od <- read.table("OIL.txt", header = TRUE)
```

il data frame contiene una serie temporale, la produzione di petrolio in milioni di barili in funzione del tempo:

```
> head(od, 15)
```

```
   Year milbarrels
1  1880         30
2  1890         77
3  1900        149
4  1905        215
5  1910        328
6  1915        432
7  1920        689
8  1925       1069
9  1930       1412
10 1935       1655
11 1940       2150
12 1945       2595
13 1950       3803
14 1955       5626
15 1960       7674
```

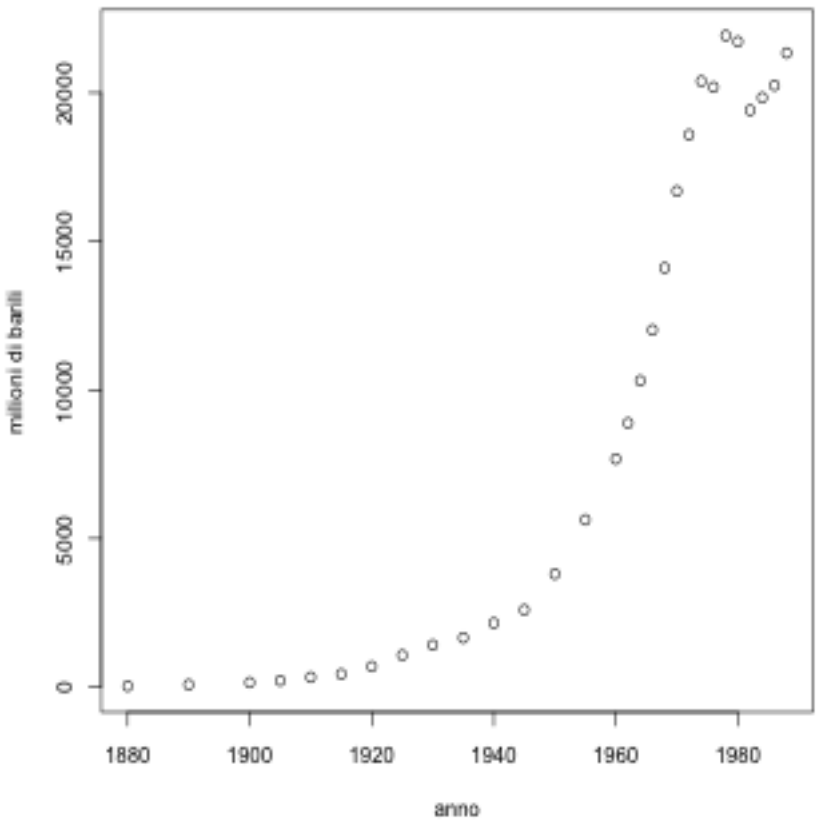
le annate prese in considerazione partono dal 1880 e arrivano al 1988

```
> range(od$Year)
```

```
[1] 1880 1988
```

esaminiamo la serie temporale:

```
> plot(od$Year, od$milbarrels, ylab = "milioni di barili", xlab = "anno")
```



il diagramma è chiaramente non lineare, la crescita nella produzione sembra approssimativamente esponenziale. Inoltre negli anni '70 la produzione per ben due volte ha smesso di crescere, per poi riprendere la crescita dal 1980. Le due variabili sono associate ma l'associazione non può essere misurata con una singola statistica nè con una misura di associazione lineare come quelle considerate nella prossima sezione.

5.2.2 Associazione fra due variabili numeriche

Il grado di associazione lineare fra due variabili numeriche è misurato dal coefficiente di correlazione di Pearson. Proviamo a calcolarlo per la relazione fra peso e altezza nel data frame d (file IQ.txt):

```
> cor(d$Weight, d$Height)

[1] 0.6900076
```

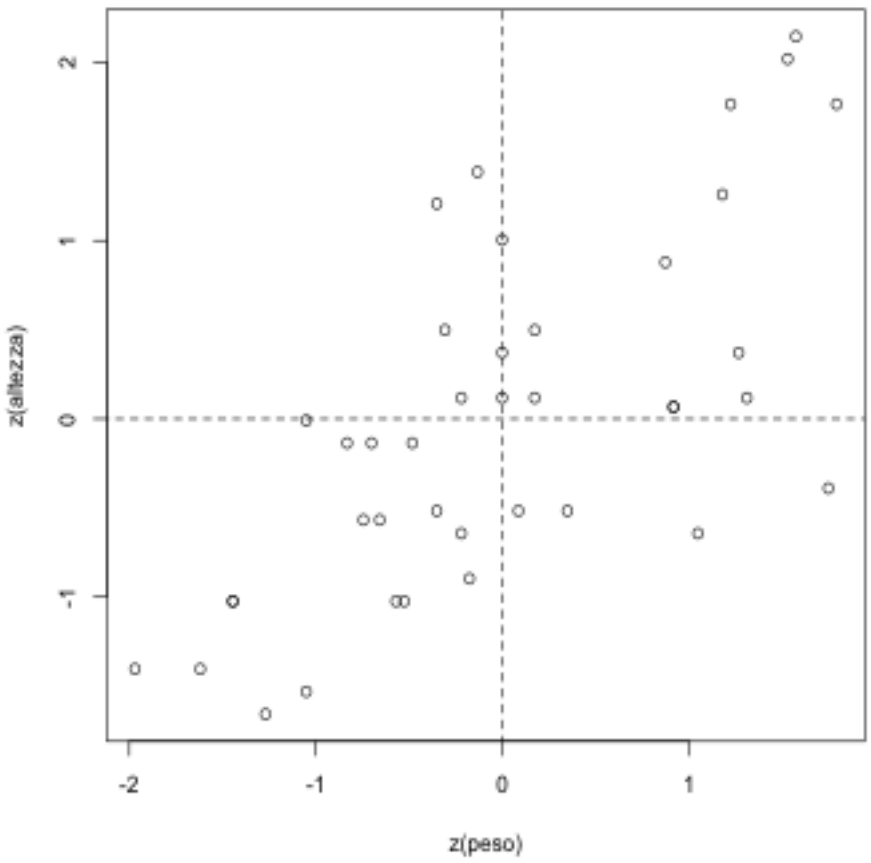
che ovviamente sono correlate positivamente. Il coefficiente di correlazione dovrebbe essere uguale alla covarianza standardizzata di y e x. Proviamo a

controllare:

```
> var(d$Weight, d$Height)/(sd(d$Weight) * sd(d$Height))

[1] 0.6900076
```

A lezione abbiamo anche definito il coefficiente correlazione come la media dei prodotti di z(x) z(y). Creiamo il diagramma di dispersione per i dati standardizzati:

```
> plot(scale(d$Weight), scale(d$Height), xlab = "z(peso)", ylab =
"z(altezza)")
> segments(-3, 0, 3, 0, lty = "dashed")
> segments(0, -3, 0, 3, lty = "dashed")
```



notate che il punto delle medie ha adesso coordinate 0, 0 e che i dati sono espressi in punti z. La maggior parte dei prodotti sarà positiva perché i punti cadono nel quadrante (+, +) e (-, -). Calcoliamo la media dei prodotti:

```
> mean(scale(d$Weight) * scale(d$Height))
```

```
[1] 0.6727574
```

notate che il risultato assomiglia, ma non è esattamente uguale a quello di cor(). Il coefficiente di correlazione campionario calcolato da R è esattamente

```
> (sum(scale(d$Weight) * scale(d$Height)))/(length(d$Weight)-1)

[1] 0.6900076
```

La funzione cor() può anche calcolare una correlazione per ranghi. Per far questo occorre specificare un "metodo". Ad esempio, riprendiamo dalla sezione precedente l'esempio della relazione fra produzione di petrolio e anno. Esaminiamo i soli dati fino al 1973:

```
> od1 <- subset(od, od$Year < 1973)
> cor(od1$Year, od1$milbarrels)

[1] 0.8385266
```

il risultato sembra indicare un alto grado di associazione. Tuttavia se guardiamo il diagramma nella sezione precedente possiamo notare che la associazione è perfetta. Ma non è una associazione lineare. In un caso come questo, in cui vi è una relazione nonlineare monotonica, un coefficiente di correlazione per ranghi rappresenta una misura di associazione migliore. Infatti

```
> cor(od1$Year, od1$milbarrels, method = "spearman")

[1] 1
```

La correlazione per ranghi con il metodo spearman non è altro che la correlazione calcolata sui ranghi invece che sui dati originari. Proviamo a verificarlo ad esempio per la correlazione fra peso e altezza nel data frame d:

```
> rw <- rank(d$Weight)
> rh <- rank(d$Height)
> round(cor(d$Weight, d$Height, method = "spearman")) == round(cor(rw, rh))

[1] TRUE
```

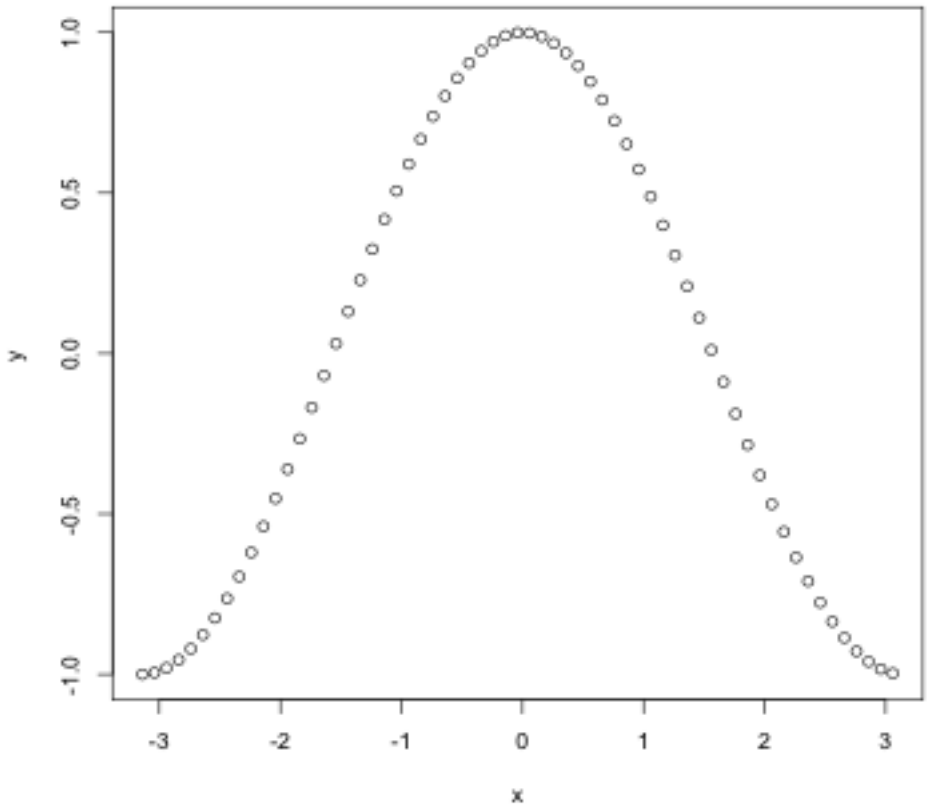
Un metodo alternativo per la correlazione per ranghi è “kendall”. Se non specificate un metodo, cor() usa il metodo di default che è “pearson”.

Infine, nel caso di una associazione nonlineare e non monotonica (ad esempio, una sigmoide o un diagramma di dispersione a forma di U rovesciata), non esiste di norma una singola misura di associazione fra le due variabili. Occorre utilizzare altre strategie per caratterizzare l’associazione, di solito fittando una funzione ai dati. Vediamo un esempio:

```
> x <- seq(-3.14, 3.14, by = 0.1)
> y <- cos(x)
> cor(x, y)

[1] 0.03098811
```

la correlazione lineare fra x e y è (praticamente) zero. Ma le due variabili sono perfettamente associate:

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



Per misurare l’associazione in presenza di associazione nonlineare e non

monotonica, consultare un esperto.

5.2.3 Regressione lineare

Quando due variabili numeriche hanno una distribuzione bivariata compatibile con una associazione lineare (il diagramma di dispersione ha approssimativamente la forma di un pallone da rugby), la distribuzione può essere riassunta usando la retta che fitta meglio i dati, ossia fittando un modello lineare. Il modello può essere utilizzato per prevedere il valore di y, noto il valore di x. La pendenza della retta può essere intrepretata come la media bivariata, che riassume la tendenza positiva o negativa dell'associazione lineare. La media quadratica dei residui (l'errore root-mean-square) può essere interpretata come una misura di dispersione bivariata, che corrisponde all'errore di previsione del modello. Il fit può essere basato su criteri diversi, noi useremo quello dei minimi quadrati (least squares).

Per fittare un modello lineare, in R è possibile usare la funzione lm(), che produce un oggetto con i risultati della modellizzazione. Vediamola in azione:

```
> rgm <- lm(d$Height ~ d$Weight)
> summary(rgm)

Call:
lm(formula = d$Height ~ d$Weight)

Residuals:
   Min     1Q Median     3Q    Max
-6.2914 -2.2402 -0.1047  1.7359  5.8254

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 50.56320    3.09261  16.350  < 2e-16 ***
d$Weight     0.11900    0.02025   5.877  8.41e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.892 on 38 degrees of freedom
Multiple R-squared: 0.4761,     Adjusted R-squared: 0.4623
F-statistic: 34.53 on 1 and 38 DF,  p-value: 8.407e-07
```

L'oggetto rgm contiene una serie di informazioni per valutare il fit oltre ai parametri della retta. Vediamoli in dettaglio. La prima parte riporta la formula di R usata per il fit. In questo caso, la formula si legge: Height in

funzione di Weight. Questo è il primo esempio di specificazione di modello per mezzo di una formula che vediamo in questa dispensa. Moltissime funzioni di R possono accettare come argomento una formula, il che si rivela estremamente utile in molti casi. Vedremo a breve altri esempi, anche con funzioni grafiche. La seconda parte riassume la distribuzione dei residui, che sono gli scarti dei valori osservati da quelli previsti dal modello. La terza parte presenta i parametri stimati, ossia l'intercetta della retta (intercept) e la pendenza, che è il numero subito a destra della variabile usata come predittore (d$Weight). La tabella dei coefficienti include anche l'errore standard del parametro stimato, e un test, basato sulla distribuzione t di Student, dell'ipotesi nulla che il parametro sia nullo. In questo caso entrambi i parametri sono statisticamente significativi (la probabilità di osservarli, se l'ipotesi nulla è vera, è < 0.05). Infine, l'ultima parte dell'oggetto riporta alcuni indicatori relativi alla bontà del fit o, come si dice talvolta in italiano, dell'adattamento. Quello che R chiama Residual standard error è l'errore root-mean-square. R-quadrato (Mutiple R-squared) è il coefficiente di correlazione al quadrato, la proporzione di varianza nei dati "spiegata" dal modello, e la statistica F alla fine si riferisce a un test sul modello nel suo complesso. Approfondirete il significato di questi aspetti della modellizzazione nel prossimo corso di Analisi dei Dati.

I parametri della retta di regressione possono essere calcolati anche a partire dai dati, utilizzando le formule che abbiamo discusso a lezione. La pendenza della retta y.x è la correlazione di x e y moltiplicata per il rapporto fra le rispettive deviazioni standard:

```
> b <- cor(d$Height, d$Weight) * sd(d$Height)/sd(d$Weight)
> b

[1] 0.1189957
```

La pendenza della retta y.x è anche la covarianza di x e y divisa per la deviazione standard di x:

```
> b <- cov(d$Height, d$Weight)/var(d$Weight)
> b

[1] 0.1189957
```

Notate che in entrambi i casi otteniamo la stessa stima indicata nel modello, a meno di una trascurabile differenza dovuta all'approssimazione. Trovata la pendenza per calcolare l'intercetta basta sostituire nella formula le medie di x e y ed esprimere in funzione dell'intercetta:

```
> a <- mean(d$Height) - b * mean(d$Weight)
> a

[1] 50.5632
```

Possiamo calcolare anche R-quadrato:

```
> cor(d$Height, d$Weight)^2

[1] 0.4761104
```

Infine, per calcolare l'errore di previsione o root-mean-square dobbiamo prima calcolare i valori previsti di y, per convenzione indicati con la lettera "y" con sopra il segno "^" - in inglese "y-hat" o y con il cappello. Per trovarli usiamo i parametri della retta

```
> yhat <- a + b * d$Weight
```

Con questi calcoliamo i corrispondenti residui e possiamo valutarne la distribuzione con il five-number-summary. Notate che sono gli stessi indici di posizione nell'oggetto rgm:

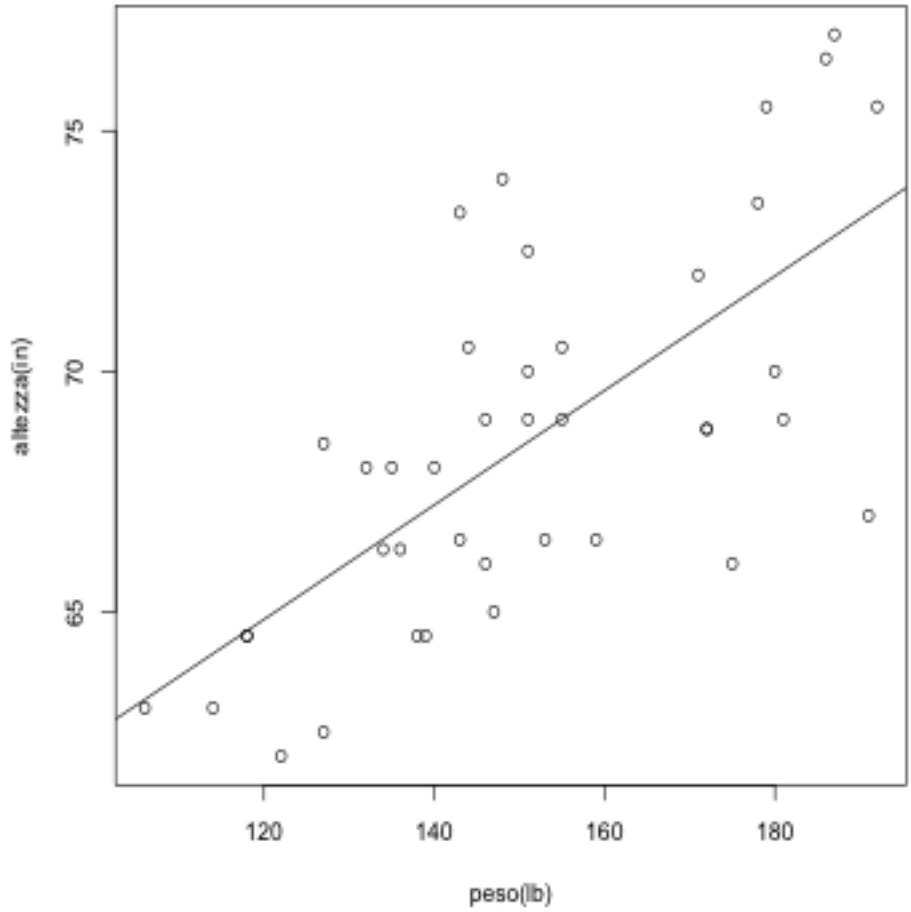
```
> res <- d$Height - yhat
> fivenum(res)

[1] -6.2913794 -2.2500011 -0.1046911  1.7424669  5.8254370
```

A questo punto, per trovare l'errore root-mean-square della regressione, basta calcolare la media quadratica dei residui, tenendo presente che la stima campionaria è priva di errore sistematico se al denominatore metto n -2 (i gradi di libertà del modello, indicati con degrees of freedom o DF in rgm). Anche in questo caso notate che il risultato è proprio quello che R chiama Residual standard error:

```
> rmse <- sqrt(sum(res^2)/(length(res)-2))
> rmse

[1] 2.891859
```

In forma grafica, la regressione lineare viene presentata sovrapponendo la retta fittata al diagramma di dispersione. La sintassi di R consente di farlo rapidamente sfruttando la funzione abline(), a cui è possibile passare i parametri a e b direttamente da lm():

```
> plot(d$Weight, d$Height, xlab = "peso(lb)", ylab = "altezza(in)")
> abline(lm(d$Height ~ d$Weight))
```



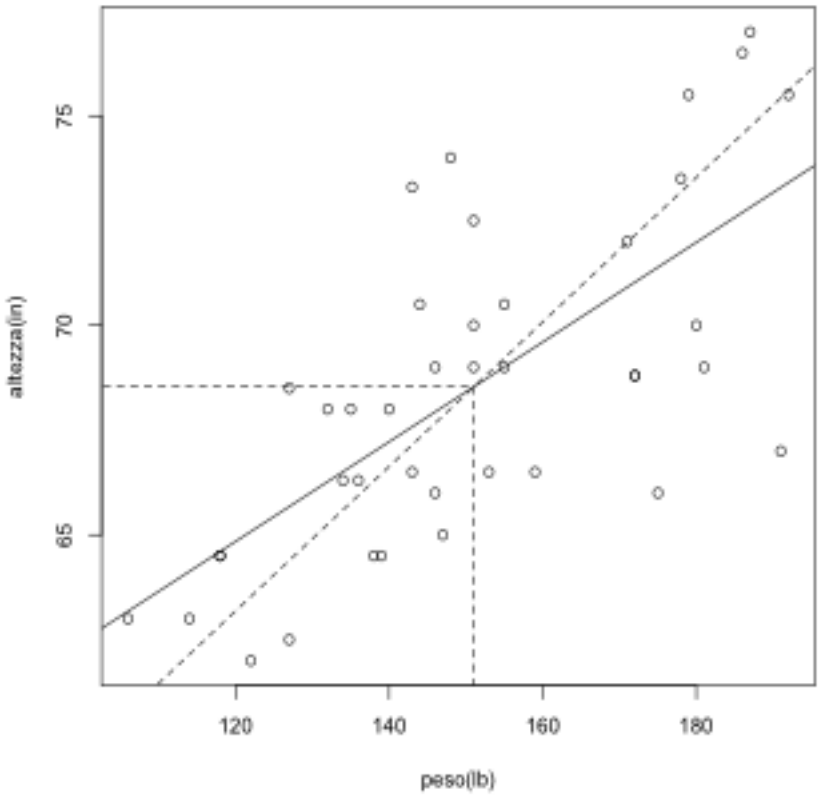
Per visualizzare in modo completo la distribuzione bivariata, includendo dei riferimenti sia per il modello che prevede y in funzione di x (regressione) sia per la visualizzazione dell'associazione lineare fra le due variabili, possiamo sovrapporre al grafico anche il punto delle medie e la retta delle deviazioni standard, come abbiamo fatto in precedenza. Per far questo proviamo a costruirci la nostra funzione di R (a questo punto dovreste essere eccitati, la prima funzione creata da voi!). Aprite l'editor e scrivete:

```
bivd <- function(x,y, xname, yname){
mx <- mean(x)
my <- mean(y)
sdx <- sd(x)
sdy <- sd(y)
plot(y ~ x, xlab = xname, ylab = yname)
segments (mx, my, mx, min(y) - 5, lty = "dashed")
```

```
segments (min(x) - 5, my, mx, my, lty = "dashed")
x1 <- mx - 3 * sdx
x2 <- mx + 3 * sdx
y1 <- my - 3 * sdy
y2 <- my + 3 * sdy
segments (x1, y1, x2, y2, lty ="dashed")
abline(lm(y ~ x))
}
```

La funzione bivd prende in input quattro parametri: due vettori, di cui volete fare il diagramma di dispersione, e due stringhe alfanumeriche, che servono per etichettare correttamente gli assi a seconda di cosa passate alla funzione. Con questa roba fa un po' di conti e disegna il diagramma. Il bello è che la funzione lo farà per qualsiasi coppia di variabili. Salvate la funzione e caricatela nel workspace come fareste con un programma. R non fa niente, in apparenza, ma in realtà la funzione è adesso nel vostro workspace e la potete usare esattamente come usate le altre funzioni predefinite. Quindi possiamo scrivere

```
> bivd(d$Weight, d$Height, "peso(lb)", "altezza(in)")
```



Notate la differenza fra la retta delle deviazioni standard (tratteggiata) e la

retta di regressione (continua). La prima fa da riferimento per valutare la forza dell'associazione lineare: tanto più i dati sono raggruppati attorno a questa retta, maggiore il coefficiente di correlazione. La seconda indica i valori previsti di y in base al modello lineare. Tanto più i dati sono raggruppati attorno a questa seconda retta, tanto minore l'errore di previsione. A questo punto potrebbe essere interessante aggiungere al diagramma anche la retta di regressione del peso in funzione dell'altezza. Dove si collocherà in questo diagramma di dispersione? Vi lascio quest'ultimo punto come esercizio.

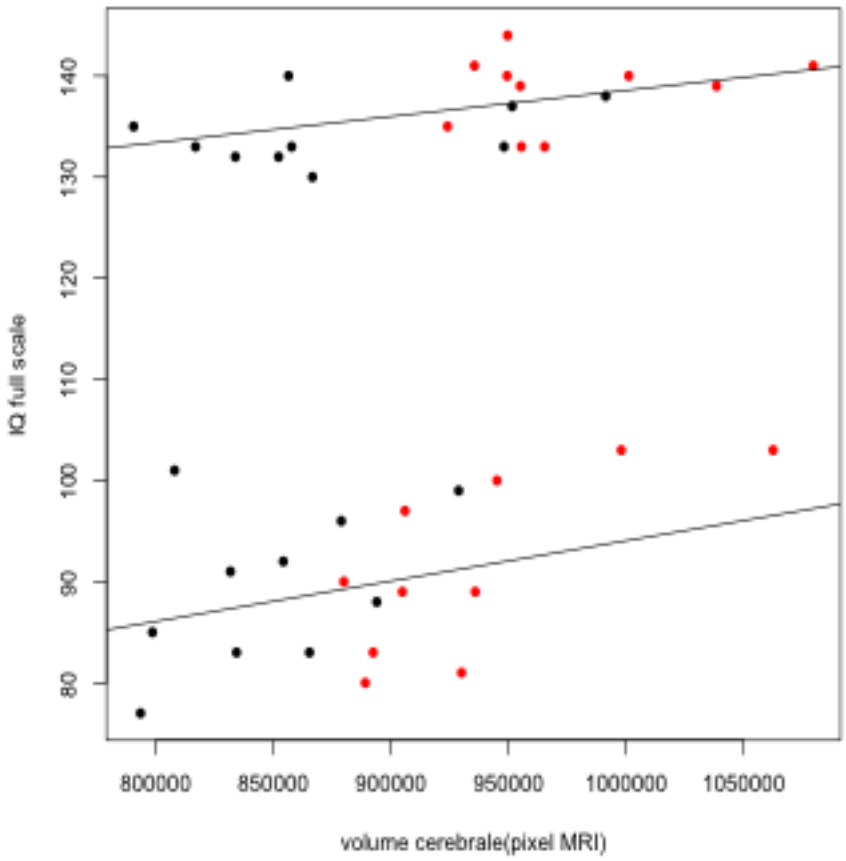
Il fit di un modello lineare ha senso se i dati sono distribuiti in maniera appropriata. Nel caso del dataset IQ, eravamo interessati a valutare se il punteggio al test di intelligenza dipende dalle dimensioni del cervello. La maniera naturale di rispondere a questa domanda è esaminare la distribuzione bivariata del punteggio, ad esempio, il QI full scale, in funzione del conteggio di pixel nelle immagini MRI. Ma come abbiamo visto in precedenza (sezione 5.2.1), il diagramma di dispersione per FSIQ in funzione di MRI_Count aveva dei problemi. I dati erano separati in due gruppi perché nell'esperimento sono stati studiati solo partecipanti con IQ superiore a 130 o inferiore a 103. La soluzione consiste nel fittare un modello lineare separatamente per i due gruppi. Vediamo come farlo. Per aggiungere anche un'altra informazione, possiamo colorare diversamente i punti relativi a uomini e donne. Useremo l'argomento pch per dire ad R che vogliamo cerchi pieni come simboli, e col per dirgli di colorare i cerchi in base al valore del fattore Gender. Se questi argomenti vi sono oscuri, per ora non preoccupatevi. Approfondiremo l'uso di questi aspetti della grafica, e in particolare del controllo dei colori, in una delle sezioni successive.

```
> plot(d$FSIQ ~ d$MRI_Count, xlab = "volume cerebrale(pixel MRI)", ylab =
"IQ full scale", pch =16, col = d$Gender)
```

Per fittare due modelli lineari separati, dobbiamo suddividere il dataset in due parti usando la funzione subset(). Fatto questo, disegniamo le rette di regressione per ogni sottoinsieme dei dati:

```
> d1 <- subset(d, d$FSIQ < 103)
> d2 <- subset(d, d$FSIQ > 130)
> abline(lm(d1$FSIQ ~ d1$MRI_Count))
> abline(lm(d2$FSIQ ~ d2$MRI_Count))
```

Il risultato dovrebbe essere :



Le due regressioni suggeriscono che, sia per i partecipanti con QI basso sia per quelli con QI alto, il QI tende a crescere all'aumentare della grandezza del cervello, in maniera simile per donne e uomini. Ma crescere di quanto? Se osserviamo la retta, possiamo vedere che, nella gamma di dimensioni cerebrali studiate, la differenza fra i cervelli più piccoli e quelli più grandi è circa di 5-6 punti per il gruppo con QI alto, e di circa 10 punti per quelli con QI basso. Presumibilmente, non si tratta di differenze che hanno grande rilevanza pratica. Possiamo provare ad approfondire la questione esaminando i due tassi di crescita del QI in funzione del volume, ossia le pendenze delle due rette. Abbiamo già visto che lm() produce un oggetto che contiene tutta una serie di informazioni sul fit del modello. Usando questo oggetto (che è una lista) potremmo accedere direttamente ai coefficienti. R ci semplifica la vita con la funzione coef():

```
> rgm1 <- lm(d1$FSIQ ~ d1$MRI_Count)
> rgm2 <- lm(d2$FSIQ ~ d2$MRI_Count)


> coef(rgm1)
```

```
 (Intercept)         d1$MRI_Count
5.411671e+01     3.994072e-05

> coef(rgm2)

 (Intercept)         d2$MRI_Count
1.127339e+02     2.583554e-05
```

Per eliminare la notazione scientifica e rendere i numeri più facilmente comprensibili, possiamo usare round(). Ad esempio, per il primo sottoinsieme abbiamo:

```
> round(coef(rgm1), 5)

 (Intercept)         d1$MRI_Count
    54.11671          0.00004
```

il tasso di crescita del QI in funzione del volume cerebrale è 4 decimillesimi di punto per pixel. Per capire cosa questo significhi veramente, dovremmo sapere come convertire il numero pixel in una misura di volume vera e propria, ad esempio, centrimetri al cubo. In ogni caso, non sembra un tasso molto impressionante. Se esaminate in dettaglio gli oggetti rgm1 e rgm2, noterete che comunque entrambi i modelli sono statisticamente signficativi, il che ha un certo interesse dal punto di vista teorico. Ma la significatività statistica non va confusa con la significatività pratica.

5.2.4 Smoothers

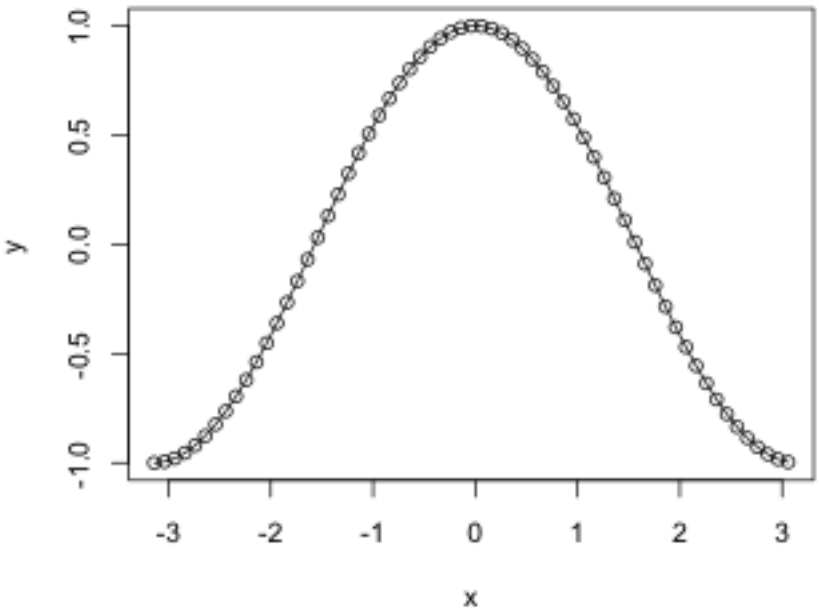
In alternativa alla regressione lineare, è possibile descrivere la relazione fra due variabili usando algoritmi non parametrici detti smoothers.
L'applicazione di uno smoother su dati bivariati consiste nel cercare una curva che catturi la forma della relazione globale fra y e x, eliminando le variazioni di dettaglio che si suppongono dovute a rumore. Non si assume a priori che la curva corrisponda ad una particolare funzione. Si applica invece un algoritmo per farla emergere direttamente dai dati. Questo approccio è particolarmente utile quando la relazione è nonlineare e soprattutto quando è nonlineare e non monotonica.

R dispone di molte funzioni per il calcolo di smoother. In questa sezione ne vedremo alcuni, allo scopo di farci un'idea di quali sono più comuni e di come lavorano.

Uno smoother semplice e robusto è runmed(), che è basato sul calcolo delle

mediana all'interno di sottoinsiemi successivi dei dati. Possiamo usarlo ad esempio sullo pseudo-scatterplot che avevamo creato con la funzione coseno:

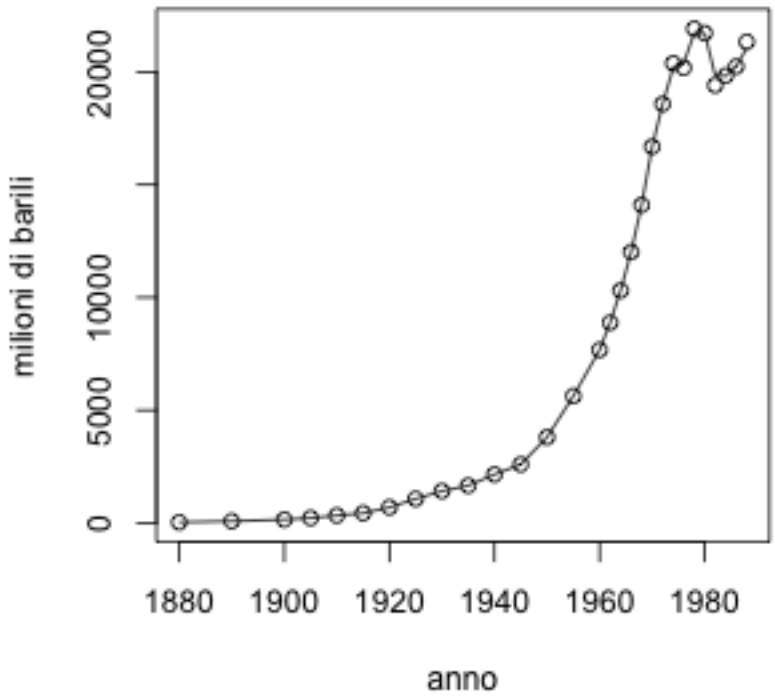
```
> x <- seq(-3.14, 3.14, by = 0.1)
> y <- cos(x)
> plot(x, y)
> lines(x, runmed(y, k = 3))
```



l'argomento k controlla l'ampiezza del sottoinsieme in cui viene calcolata la mediana, e per estendere la curva a tutto il range dei dati vengono utilizzate altre procedure che possiamo trascurare. In alternativa è possibile usare anche smooth(), che funziona in modo simile.

La funzione runmed() di solito funziona bene su serie temporali. Ne avevamo vista una nel data set OIL.txt. Il risultato cattura bene la crescita monotonica fino alla fine degli anni '70 e anche l'alterazione in questa crescita attorno al 1980:

```
> od <- read.table("~/Desktop/R/dispense/esempi_corso/OIL.txt", header =
TRUE)
> plot(od$Year, od$milbarrels, ylab = "milioni di barili", xlab = "anno")
> lines(od$Year, runmed(od$milbarrels, k = 3))
```



Per applicare runmed() o smooth() è necessario che y sia nell'ordine in cui i dati si posizionano sull'asse x. Nel primo esempio questo era implicito nella maniera in cui abbiamo creato i vettori. Ma non è necessariamente vero con dati reali in formato rettangolare. Prendiamo ad esempio le variabili peso e altezza dal dataset IQ.txt.

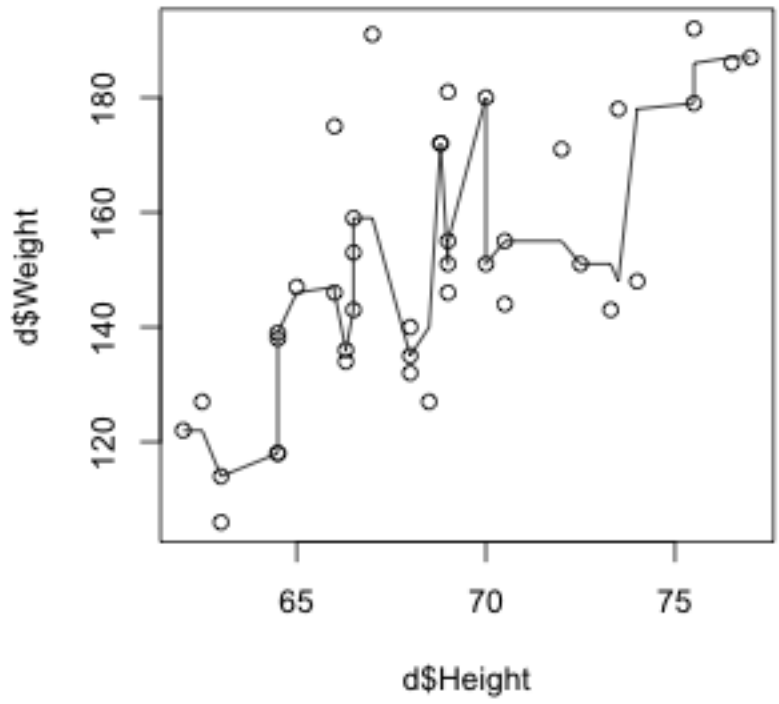
```
> d <- read.table("~/Desktop/R/dispense/esempi_corso/IQ.txt", header =
TRUE)
> plot(d$Weight ~ d$Height)
```

ora provate ad applicare runmed(), sostituendo Weight e Height a x e y nell'esempio precedente per disegnare lo smooth usando lines(). Chiaramentente qualcosa non ha funzionato. Creiamo un nuovo data frame, con i dati ordinati in base a Weight e poi in base a Height:

```
> sorted <- d[order(d$Height, d$Weight), ]
```

controllate il contenuto di sorted per vedere cosa è successo. Adesso possiamo scrivere:

```
> plot(d$Weight ~ d$Height)
> lines(sorted$Height, runmed(sorted$Weight, k = 3))
```



in questo caso ha funzionato, ma c'è un altro problema. Lo smoother cattura troppi dettagli della variazione di y in funzione di x. Il risultato non è un andamento globale ma riflette ancora piccole variazioni locali. Per rimediare possiamo provare a modificare l'argomento k, che definisce l'ampiezza della "finestra" all'interno della quale viene calcolata la mediana. Notate che lo smoother si modifica ma non riesce a produrre una curva continua. In questi casi può essere utile provare ad usare uno smoother basato su altri algoritmi. Vediamone alcuni. Fate caso alla differenza nella sintassi rispetto a runmed() o smooth(). Questi algortimi lavorano sulla distribuzione bivariata, non solo sul vettore y.

```
> plot(d$Weight ~ d$Height)
> lines(smooth.spline(d$Height, d$Weight))
```

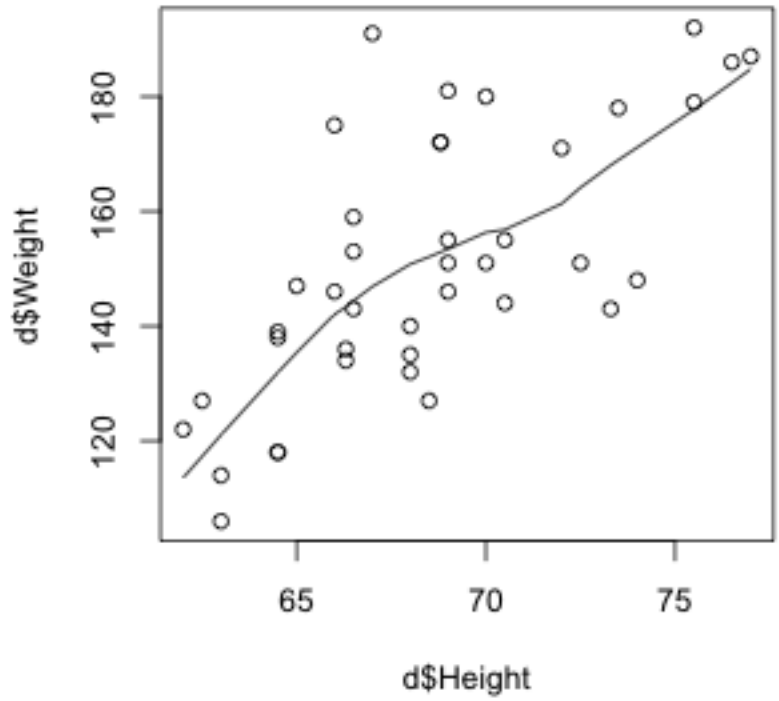
La funzione smooth.spline() è basato sul concetto di spline, uno strumento che veniva usato per il disegno tecnico consistente in un pezzo di metallo flessibile da usare come guida per tracciare curve (non conosco la traduzione italiana, anche se suppongo esista). Per usare lo spline si selezionavano dei punti del diagramma, detti nodi, e il metallo veniva curvato in modo da definire una curva continua fra due di questi punti. Procedendo per coppie successive di nodi il risultato era una curva continua. La funzione di R fa la stessa cosa, ma il processo è molto sensibile alla

selezione dei punti e come avete visto non funziona bene in un diagramma come questo. Provate a usarlo invece sulla funzione coseno che abbiamo studiato in precedenza.

```
> plot(d$Weight ~ d$Height)
> lines(ksmooth(d$Height, d$Weight))
```

La funzione ksmooth() funziona calcolando la media pesata di tutti i valori di y i cui valori di x sono prossimi al valore di x del punto che va messo nel grafico. La funzione che definisce i pesi viene chiamata kernel, e il numero di punti da usare nel calcolo delle medie viene controllato da un argomento chiamato bandwidth. Se studiate la documentazione online potete vedere qual è il valore di default e potete provare a controllare cosa succede se viene modificato.

```
> plot(d$Weight ~ d$Height)
> lines(lowess(d$Height, d$Weight))
```



Quest'ultimo esempio usa la funzione lowess(), che è basata su un algoritmo chiamato LOcally WEighted regresSSion[10]. La strategia in questo caso

[10] Cleveland, W. S. (1981) LOWESS: A program for smoothing scatterplots by robust locally weighted regression. *The American Statistician*, 35, 54.

consiste nel fittare una curva a sottoinsiemi successivi dei dati e usare i valori della curva per tracciare lo smoother. Questo approccio funziona piuttosto bene per questi dati. L'argomento f, che ha valore di default 0.6667, controlla quanto sono grandi i sottoinsiemi, per cui al crescere di f la curva diventa sempre più continua. Ad esempio, provate a settare f = 2. La curva diventa praticamente una retta, che è quello che vorremmo in questo caso. Sappiamo, perchè l'abbiamo studiata nella sezione sulla regressione lineare, che la relazione fra peso e altezza è descritta in maniera adeguata da una retta.

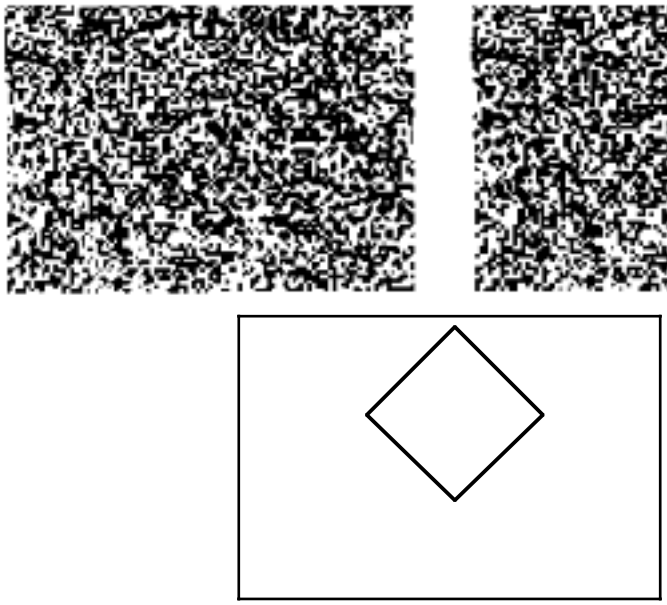
Il vantaggio di utilizzare uno smoother non sta nei casi come questo, che abbiamo analizzato a scopo didattico, ma nelle situazioni in cui la forma della relazione non è nota a priori ma l'approccio non parametrico può contribuire a farla emergere. Spesso si procede per prove ed errori fino a trovare la strategia che funziona meglio. L'interpretazione della curva ottenuta naturalmente dipende dalla teoria su cui è basato il lavoro di ricerca.

5.3 Una categoria e una variabile numerica

In quest'ultima sezione ci occupiamo di come descrivere la relazione fra una variabile categoriale ed una variabile numerica. Si tratta di un caso comunissimo nella ricerca psicologica, in cui una qualche variabile di prestazione, come il tempo di risposta o una misura ottenuta da registrazioni psicofisiologiche, viene esaminata confrontando gruppi diversi di partecipanti o partecipanti cui sono stati sommistrati trattamenti sperimentali diversi. Utilizzeremo due nuovi set di dati.

Il primo set di dati che esamineremo proviene da una classica ricerca sulla percezione di forme visive in stereogrammi di punti casuali (Random-Dot Stereograms)[11]. Uno stereogramma di punti casuali è uno stimolo molto usato in psicofisica, costituito da una coppia di immagini presentate separatamente ai due occhi mediante uno stereoscopio. Le singole immagini contengono solo punti disposti a caso, per cui in visione monoculare si vede solo rumore visivo. In visione binoculare, il cervello scopre delle corrispondenze nei punti delle due immagini, e grazie al processo di fusione binoculare fa emergere forme prima invisibili. Una tipica coppia di immagini ha un aspetto simile ai due rettangoli nella riga superiore dell'immagine qui sotto. Se siete capaci di fondere in visione libera, senza l'ausilio dello stereoscopio (qualche persona lo sa fare naturalmente, tutti possono imparare a farlo con l'aiuto di un esperto di visione), avrete visto emergere la forma disegnata nel rettangolo sotto al centro.

[11] Frisby, J. P. & Clatworthy, J.L. (1975). Learning to see complex random-dot steregrams, *Perception*, 4, pp. 173-178.



Scaricate il file RDS.txt dal sito del docente ed esaminatene il contenuto usando un qualsiasi programma di videoscrittura. I dati riportano il tempo necessario per arrivare alla fusione e alla visione di queste forme, in secondi (la variabile Ts). I partecipanti tentavano di fondere gli stereogrammi in due condizioni diverse (variabile group). In una condizione (NV), i partecipanti non ricevevano informazione visiva sulla forma dell'oggetto "nascosto" nello stereogramma. In un'altra condizione (VV), vedevano un disegno dell'oggetto prima di tentare di fondere. Esaminare l'associazione fra le due variabili è come chiedersi se il tempo di fusione dipende dal fatto di avere visto precedentemente la forma, ossia qual'è l'effetto del gruppo sul tempo.

Proviamo a procedere come farebbe un ricercatore: scriviamo e salviamo un programma per studiare la distribuzione e calcolare indicatori statistici di associazione. L'analisi è tutta contenuta nel programma di R riportato sotto. Copiate e incollate nel vostro computer e prendetevi qualche minuto per studiare cosa fa.

```
# lettura dei dati, attenzione al path sul vostro computer
RDS <- read.table("RDS.txt", header = TRUE)
par(ask = TRUE)  # grafici uno alla volta please
# che aspetto ha la distribuzione bivariata?
boxplot(RDS$Ts ~ RDS$group, xlab = "Group", ylab = "Time(s)")
# quanto è grave l'asimmetria?
par(mfrow = c(2,2))  # quattro grafici su una pagina please
hist(RDS$Ts[RDS$group == "NV"], freq = FALSE, main = "no
preview", xlab ="time(s)")
```

```
hist(RDS$Ts[RDS$group == "VV"], freq = FALSE, main = "preview",
xlab ="time(s)")
qqnorm(RDS$Ts[RDS$group == "NV"])
qqline(RDS$Ts[RDS$group == "NV"])
qqnorm(RDS$Ts[RDS$group == "VV"])
qqline(RDS$Ts[RDS$group == "VV"])
# occorre una trasformazione
par(mfrow = c(1,1)) # solo il boxplot sulla pagina please
boxplot(log10(RDS$Ts) ~ RDS$group, xlab = "Group", ylab =
"log(Time)")
# sembra un po' meglio ma controlliamo
par(mfrow = c(2,2))  # quattro grafici su una pagina please
hist(log10(RDS$Ts)[RDS$group == "NV"], freq = FALSE, main = "no
preview", xlab ="log(time))")
hist(log10(RDS$Ts)[RDS$group == "VV"], freq = FALSE, main =
"preview", xlab ="log(time))")
qqnorm(log10(RDS$Ts)[RDS$group == "NV"])
qqline(log10(RDS$Ts)[RDS$group == "NV"])
qqnorm(log10(RDS$Ts)[RDS$group == "VV"])
qqline(log10(RDS$Ts)[RDS$group == "VV"])
# rimettiamo a posto i parametri grafici
par(mfrow = c(1,1), ask = FALSE)
# quanto è forte l'associazione?
# qualche preliminare per semplificare il codice
t1 <- RDS$Ts[RDS$group == "NV"]
t2 <- RDS$Ts[RDS$group == "VV"]
lt1 <- log10(RDS$Ts)[RDS$group == "NV"]
lt2 <- log10(RDS$Ts)[RDS$group == "VV"]
n1 <- length(t1)
n2 <- length(t2)
# differenze fra le medie
mdif <- (mean(t2) - mean(t1))
lmdif <- (mean(lt2) - mean(lt1))  # in unità log10
# d di Cohen
d <- mdif/sqrt(((n1-1)*var(t1)+(n2-1)*var(t2))/(n1+n2-2))
ld <- lmdif/sqrt(((n1-1)*var(lt1)+(n2-1)*var(lt2))/(n1+n2-2))
# pendenza della retta di regressione = differenza fra le medie
RDS$dc <- c(rep(0, 43), rep(1, 35))
mod <- lm(RDS$Ts ~ RDS$dc)
cf <- coef(mod)
lmod <- lm(log10(RDS$Ts) ~ RDS$dc)
lcf <- coef(lmod)
```

La prima istruzione del programma vi è già familiare. Abbiamo letto i dati e

creato un data frame in R. Possiamo usare str() e head() per verificare come R ha letto le due colonne. Come era prevedibile, Ts è un vettore numerico mentre group è un fattore con due livelli.

La prima cosa da fare è usare dei grafici per studiare la distribuzione bivariata. Dato che faremo parecchi grafici, è una buona idea settare par() per comandare la presentazione da tastiera. Fatto questo, possiamo dare una prima occhiata al tempo nei due gruppi usando boxplot(). Notate l'uso della notazione con formula. Le due distribuzioni sono molto asimmetriche, come è tipico quando i dati sono tempi di risposta. Ci sono addirittura diversi punti che R ha posizionato sopra il massimo. Si tratta di dati che R ha identificato come outlier, ossia come valori estremi o anomali. Il criterio di default per la classificazione come outlier è che il dato sia più di 1.5 volte la gamma interquartile a partire dal terzo (sopra) o primo (sotto) quartile della distribuzione. La presenza di outlier in una distribuzione non è necessariamente motivo di preoccupazione. Tuttavia il fatto che questi siano solo sopra, e non sotto, il boxplot è un segno certo di asimmetria della distribuzione.

Quando è grave la deviazione da una distribuzione simmetrica e unimodale? Per farcene un'idea possiamo ispezionare gli istogrammi di Ts nei due gruppi e confrontarli con le aspettative della distribuzione normale usando qqnorm(). Per vedere i quattro grafici tutti assieme usiamo par() settando mfrow in modo appropriato. Nella seconda schermata vediamo che i due istogrammi sono asimmetrici e che i quantili delle distribuzioni empiriche sono diversi dalle aspettative teoriche basate sulla distribuzione normale. Per ridurre l'asimmetria proviamo la trasformazione logaritmica. Usiamo log10(), il logaritmo in base 10, per facilitare l'interpretazione del cambio di scala. Usando istogrammi e diagrammi quantile-quantile vediamo che la situazione è molto migliorata. Possiamo usare i dati trasformati per le successive analisi. A questo punto abbiamo finito di usare i grafici e rimettiamo par() sui valori di default.

L'ultima parte del programma calcola alcuni indicatori della relazione fra le due variabili. Volendo potremmo inserire nel programma anche dei comandi per stamparli ma è semplice anche chiederli dopo il prompt. Il primo e il più ovvio indicatore è la differenza fra le medie. Nelle unità originali, questa è

```
> mdif

[1] -3.009036
```

Nella condizione VV c'è un risparmio di ben tre secondi nel tempo di fusione rispetto alla condizione NV. Le medie aritmetiche sono però inadatte a

riassumere questi dati, a causa dell'asimmetria. In unità logaritmiche, la differenza è

```
> lmdif

[1] -0.1869884
```

La differenza è sempre negativa (risparmio in VV rispetto a NV) perché la trasformazione è monotonica: non ha modificato i ranghi dei dati, ma solo la scala. Questa differenza può essere interpretata riportando le medie alla scala originale con una elevazione a potenza della base del logaritmo:

```
> 10^mean(lt2) - 10^mean(lt1)

[1] -2.159266
```

con cui abbiamo trovato la differenza fra le medie geometriche delle distribuzioni. La differenza si è ridotta, perché la media aritmetica del gruppo NV era fortemente influenzata dai valori estremi (soprattutto uno) osservati in quel gruppo. Rimane comunque più di due secondi.

Un altro indicatore di come il tempo dipende dal gruppo è l'effect size, che per la differenza fra due medie è il d di Cohen. Il programma lo calcola sia per i dati originali sia per i logaritmi:

```
> d

[1] -0.4415395

> ld

[1] -0.5279324
```

Notate che in unità logaritmiche il d di Cohen aumenta. Questo dipende dal fatto che d è una misura standardizzata, sensibile a una riduzione nella dispersione della distribuzione. Secondo le convenzioni proposte da Cohen, un effetto attorno a circa 0.5 può essere classificato come medio. Non va comunque dimenticato che, quando è possibile, è sempre meglio esprimere l'effetto in unità di misura che abbiano un significato pratico[12]. Per questo motivo, la differenza espressa in secondi è in questo caso preferibile. Ricordiamo, fra l'altro, che la differenza fra le medie corrisponde alla

[12] Wilikinson, L. & Task Force on Statistical Inference, APA Board of Scientific Affairs (1999) Statistical Methods in Psychology Journals: Guidelines and Explanations. *American Psychologist,* 54, 594-604.

pendenza della retta di regressione se riclassifichiamo i gruppi usando una codifica numerica in cui 0 = assenza della caratteristica, 1 = presenza (dummy coding). Nel programma, questo viene fatto creando una nuova variabile dm e fittando due modelli lineari, uno per i dati originali e uno per la loro trasformazione logaritmica. Vediamo i parametri e verifichiamo che le pendenze sono proprio uguali alla differenza fra le medie, mentre le intercette corrispondo alla media del gruppo codificato con lo zero.

```
> cf

(Intercept)      RDS$dc
   8.560465   -3.009036

> lcf

(Intercept)      RDS$dc
  0.7904207  -0.1869884
```

Nel caso della trasformazione logaritmica, possiamo verificare che il decremento associato all'aumento unitario di x è effettivamente, una volta riespresso in secondi, pari alla differenza fra le medie geometriche:

```
> 10^0.7904207 - 10^(0.7904207-0.1869884)

[1] 2.159266
```

Il secondo set di dati che esaminiamo proviene da uno dei pacchetti preinstallati di R, il package MASS. Per caricarlo è necessario scrivere:

```
> library(MASS)
> data(anorexia)
```

il dataset contiene i risultati di una ricerca su due trattamenti per l'anoressia, la terapia familiare e al terapia cognitivo-comportamentale. Esaminiamo la struttura del dataset:

```
> str(anorexia)

'data.frame':        72 obs. of  3 variables:
 $ Treat : Factor w/ 3 levels "CBT","Cont","FT": 2 2 2 2 2 2 2 2 2 2 ...
 $ Prewt : num  80.7 89.4 91.8 74 78.1 88.3 87.3 75.1 80.6 78.4 ...
 $ Postwt: num  80.2 80.1 86.4 86.3 76.1 78.1 75.1 86.7 73.5 84.6 ...
```

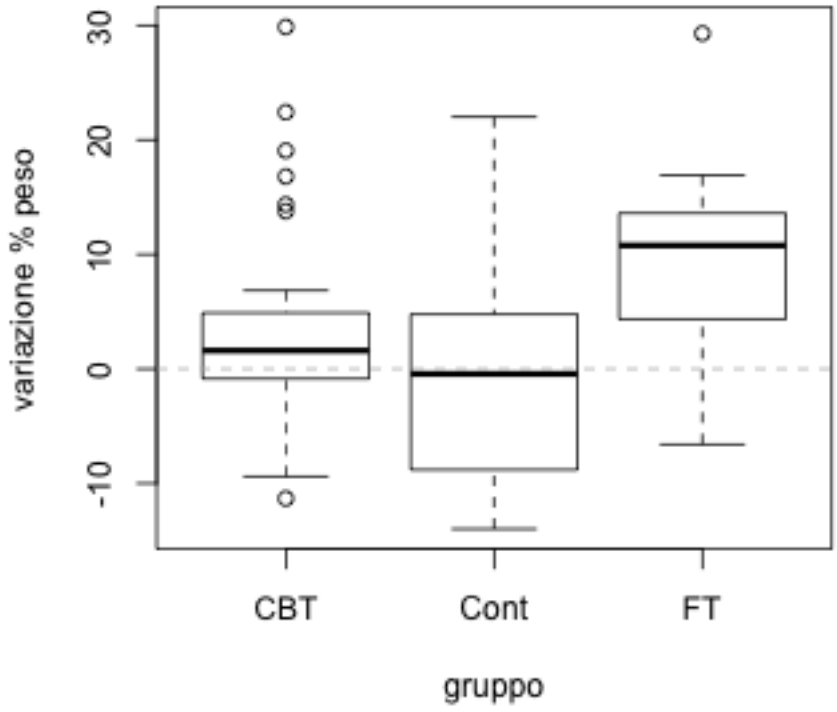
Il data frame contiene 72 osservazioni, relative al peso prima del

trattamento (Prewt) e dopo il trattamento (Postwt) in partecipanti anoressiche. Le partecipanti sono state assegnate a caso a uno di tre gruppi di trattamento (Treat): CBT (cognitive behavioral therapy, terapia cognitivo comportamentale), FT (family terapy, terapia familiare) e Cont (controllo). In questo caso l'informazione rilevante sta nella variazione prima e dopo il trattamento. Come prima cosa allora esprimiamo i dati come variazione percentuale:

```
> anorexia$difw <- 100 * (anorexia$Postwt -anorexia$Prewt)
                                          /anorexia$Prewt
```

quindi visualizziamo l'associazione fra variazione percentuale e gruppo:

```
> boxplot(difw ~ Treat, data = anorexia, ylab = "variazione % peso", xlab =
"gruppo")
> abline(h = 0, col = "dark grey", lty = "dashed")
```



le distribuzioni della variazione percentuale nei tre gruppi appare ragionevolmente simmetrica e indica che la terapia familiare produce mediamente un aumento di peso attorno al 10-15%, mentre la terapia cognitivo comportamentale è sostanzialmente simile al gruppo di controllo,

dove non si nota in media alcun aumento di peso. L'incremento proporzionale può essere considerato una misura della grandezza dell'effetto, in questo caso più significativo delle differenze standardizzate, quindi l'associazione fra le due variabili è ben rappresentato dalle medie dei tre gruppi.

Calcoliamo le medie usando la funzione tapply(), che applica una funzione a un vettore in base ai livelli di un fattore. La sua sintassi è tapply(x, index, fun) dove x è un vettore, index è il fattore, è fun la funzione da applicare:

```
> mns <- tapply(anorexia$difw, anorexia$Treat, mean)
> mns

      CBT       Cont             FT
3.72397187 -0.00655586  8.80129215
```

l'oggetto mns contiene le medie, che confermano l'andamento che avevamo visto nel boxplot. Lo stesso approccio può essere utilizzato per calcolare le deviazioni standard nei tre gruppi, o qualsiasi altra statistica.

6. Distribuzioni teoriche di probabilità

Per molte applicazioni statistiche è utile lavorare con distribuzioni teoriche di probabilità. R offre funzioni per lavorare con moltissime di queste. Qui vedremo in dettaglio come lavorare con la distribuzione normale, quindi vedremo come generalizzare il concetto alle altre distribuzioni.

6.1 Distribuzione normale o gaussiana

La funzione teorica di probabilità di una distribuzione normale è data dalla funzione dnorm(). Dato un vettore, dnorm() calcola l'altezza della curva normale (la densità di probabilità) per ogni elemento del vettore. La funzione ha sintassi dnorm(x, mean = 0, sd = 1), dove x è un qualsiasi vettore e mean e sd sono la media aritmetica e la deviazione standard del vettore. Se questi ultimi due argomenti vengono omessi, R assume la normale standard con media = 0 e sd = 1. Ad esempio, la densità della media in una normale standard è

```
> dnorm(0)

[1] 0.3989423
```

ma la densità di x = 0 in una normale con media 4 è

```
> dnorm(0, mean = 4)

[1] 0.0001338302
```

La densità del 95% percentile è

```
> dnorm(1.644854)

[1] 0.1031356
```

Naturalmente x può essere anche un vettore di numeri:

```
> dnorm(c(85, 100, 115), mean = 100, sd = 15)

[1] 0.01613138 0.02659615 0.01613138
```

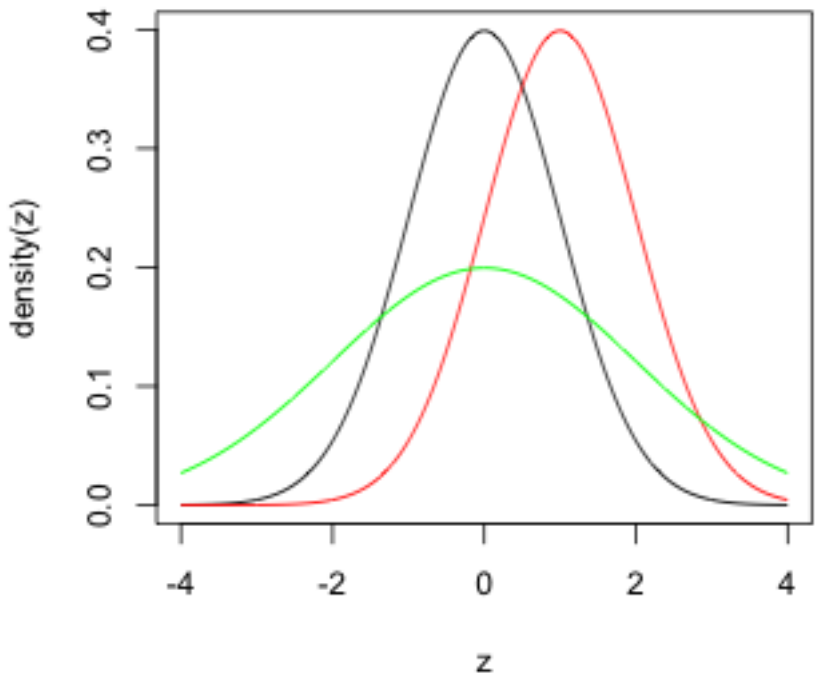
Attenzione a non confondere la densità con la probabilità. In una funzione continua, la probabilità è definita solo per intervalli, per quanto piccoli, di x, e non per singoli valori.

Ora creiamo un vettore e plottiamo la funzione:

```
> z <- seq(-4, 4, by = 0.01)
> plot(z, dnorm(z), type = "l", ylab = "density(z)")
```

Dovreste avere davanti la curva normale. Proviamo a sovrapporre in rosso una normale con media diversa e in verde una seconda normale con deviazione standard diversa.

```
> lines(z, dnorm(z, 1, 1), col = "red")
> lines(z, dnorm(z, 0, 2), col = "green")
```



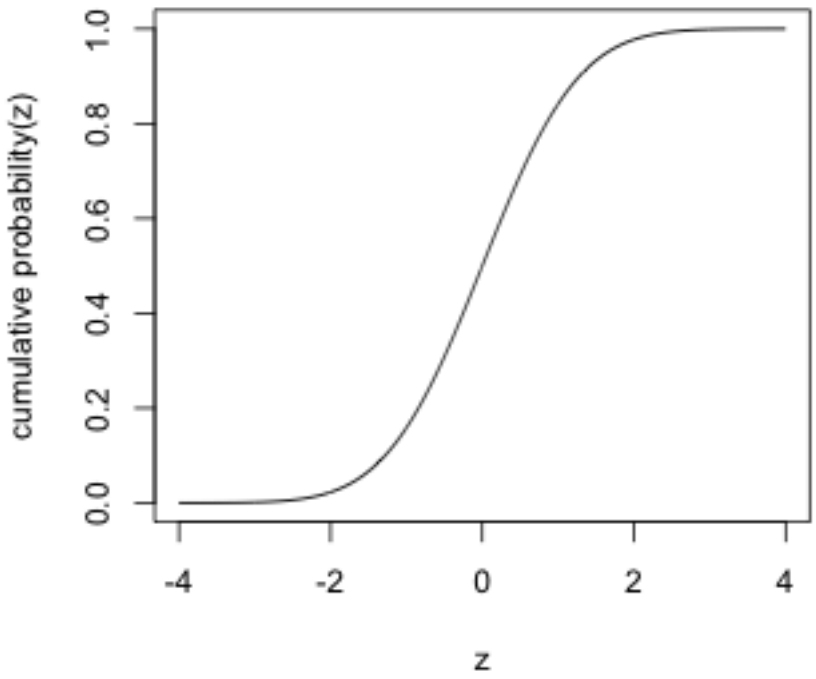
Con la funzione pnorm() otteniamo la curva della probabilità cumulativa della normale. La sintassi è uguale a quella di dnorm(), ma invece dell'altezza della curva ora calcoliamo l'area relativa (l'area totale = 1) sotto la curva dal valore dato di z fino a + infinito o - infinito. Di default R si basa sulla "coda" inferiore, ossia l'area da - infinito a z. Settando lower.tail = FALSE usa in la coda superiore. Ad esempio, quando z è la media la probabilità di osservare un valore pari a z o minore è sempre 0.5 perché la distribuzione è simmetrica.

```
> pnorm(0)
```

```
[1] 0.5

> pnorm(4)

[1] 0.9999683

> pnorm(4, mean = 4)

[1] 0.5

> pnorm(1000, mean = 1000)

[1] 0.5
```

Per disegnare la curva della probabilità cumulativa usiamo plot():

```
> plot(z, pnorm(z), ylab = "cumulative probability(z)", type = "l")
```

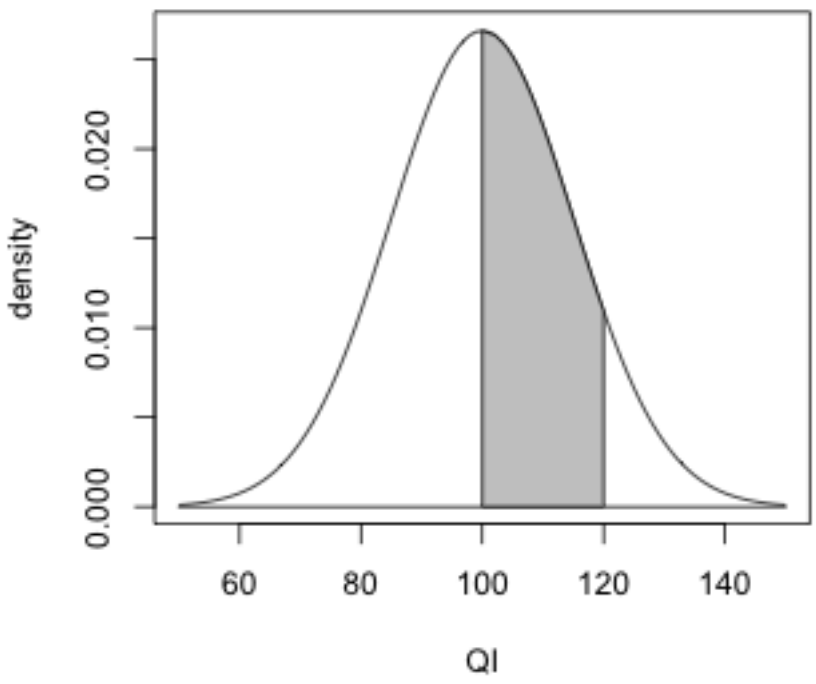


La funzione inversa di pnorm() è qnorm(). In questo caso, invece di calcolare la probabilità cumulativa a partire da un valore del vettore, si passa alla funzione una probabilità cumulativa e la funzione restituisce il valore del vettore che sottende a quell'area nella curva. Ad esempio, se usiamo la

normal standard qnorm() restituisce il punto zeta:

```
> qnorm(0.5)

[1] 0

> qnorm(0.95)

[1] 1.644854

> qnorm(0.5, mean = 100, sd = 15)

[1] 100
```

Usando queste funzioni è facile calcolare aree sotto la curva normale. Ad esempio, consideriamo la distribuzione del QI (per definizione con media 100 e deviazione standard 15). Supponiamo che ci interessi la probabilità di un QI compreso fra la media e 120, ossia che ci interessi di trovare l'area in grigio [13]:



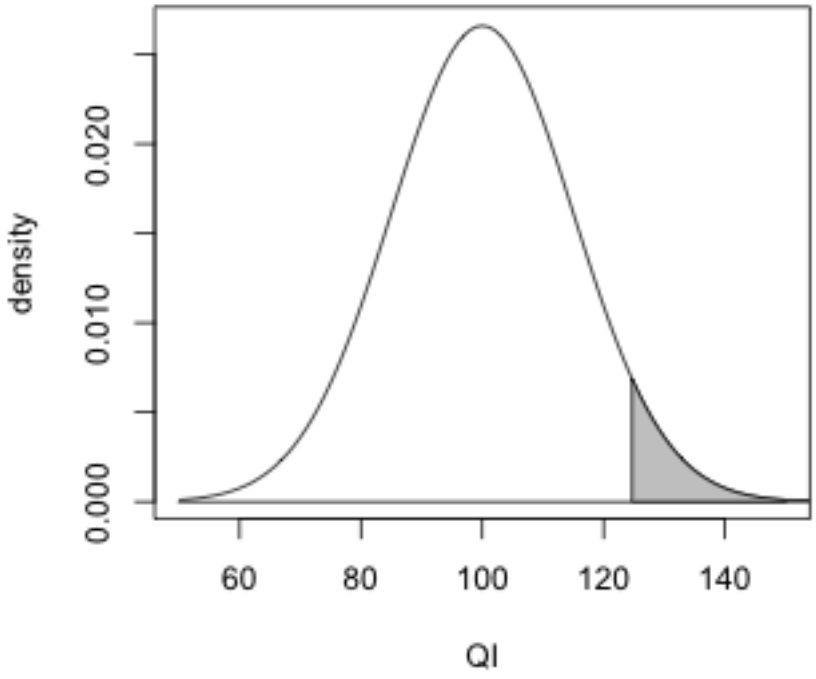
L'area relativa in questione non è altro che la probabilità cumulativa di un QI

[13] Per plottare l'area in grigio è stata usata la funzione polygon(), di cui vedremo maggiori dettagli nella sezione sulla grafica in R.

= 120 meno la probabilità cumulativa di un QI = 100, ossia l'area fra -infinito e 120 meno l'area fra -infinito e 100:

```
> pnorm(120, mean = 100, sd = 15) - pnorm(100, mean = 100, sd = 15)

[1] 0.4087888
```

Supponiamo ora che ci interessi l'area a partire da QI = 124.6728, vale a dire la parte in grigio nella coda di destra:



la troviamo con

```
> pnorm(200, 100, 15) - pnorm(124.6728, 100, 15)

[1] 0.05
```

Naturalmente, il valore non è scelto a caso. In punti z

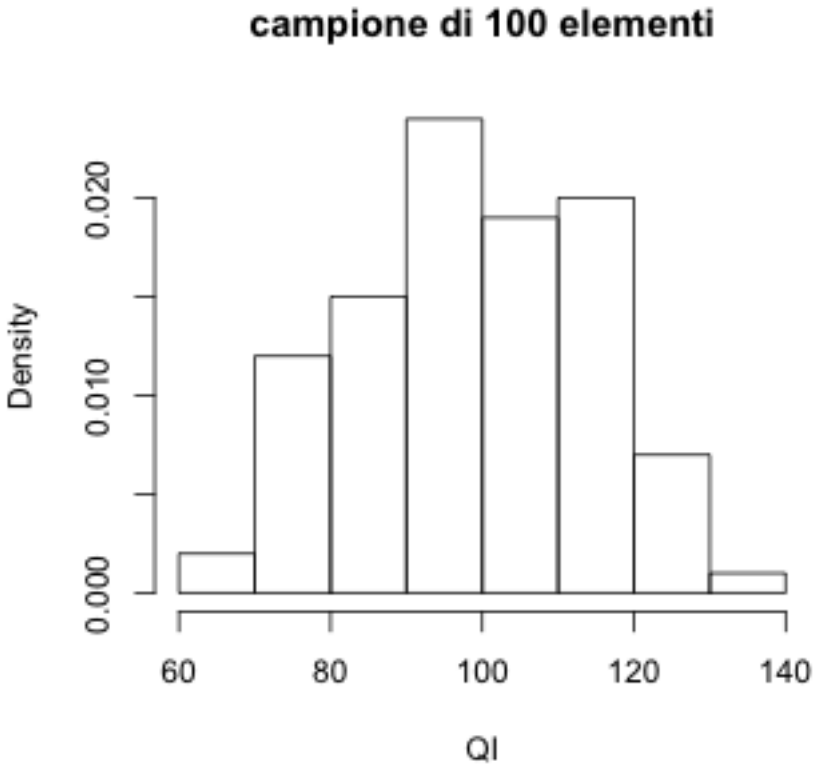
```
> scale(124.6728, 100, 15)
        [,1]
[1,] 1.644853
attr(,"scaled:center")
[1] 100
```

```
attr(,"scaled:scale")
[1] 15
```

e come sappiamo, il 90% della distribuzione teorica normale è compreso fra circa -1.645 volte la deviazione standard e 1.645 volte la deviazione standard.

Infine, una funzione molto utile è rnorm(). Questa funzione produce numeri casuali la cui distribuzione è normale. La sintassi è rnorm(n, mean, sd), dove n è la grandezza del campione, e mean, sd sono i parametri della distribuzione (come al solito, se li ometto usa la normale standard). Ad esempio, estraiamo un campione di 100 QI dalla normale teorica con media 100 e sd 15:

```
> c <- rnorm(100, 100, 15)
> hist(c, main = "campione di 100 elementi", freq = FALSE, xlab = "QI")
```



6.2 Distribuzione binomiale

R dispone di funzioni per lavorare su molte altre distribuzioni teoriche di probabilità. Sono facili da ricordare perché sono denominate tutte usando le stesse convenzioni che abbiamo visto per la normale. Ad esempio, per la distribuzione binomiale, dbinom() calcola la densità di probabilità, pbinom() la

funzione di probabilità cumulativa, qbinom() la sua inversa, e rbinom() produce numeri casuali la cui distribuzione è binomiale con i parametri specificati. L'unica differenza è che per la binomiale è necessario definire il numero di prove e la probabilità di successi. La sintassi diventa quindi, ad esempio, dbinom(x, size, prob).

Ad esempio, supponiamo che in un esame 30 domande prevedano risposte a scelta multipla con quattro alternative. Supponiamo che uno studente risponda a caso. Possiamo calcolare la probabilità di azzeccare 7 risposte con

```
> dbinom(7, 30, 1/4)

[1] 0.1662357
```

oppure 18 risposte con

```
> dbinom(18, 30, 1/4)

[1] 3.986919e-05
```

mentre se vogliamo la probabilità di non più di 10 risposte corrette, possiamo scrivere

```
> pbinom(10, 30, 1/4)

[1] 0.8942719
```

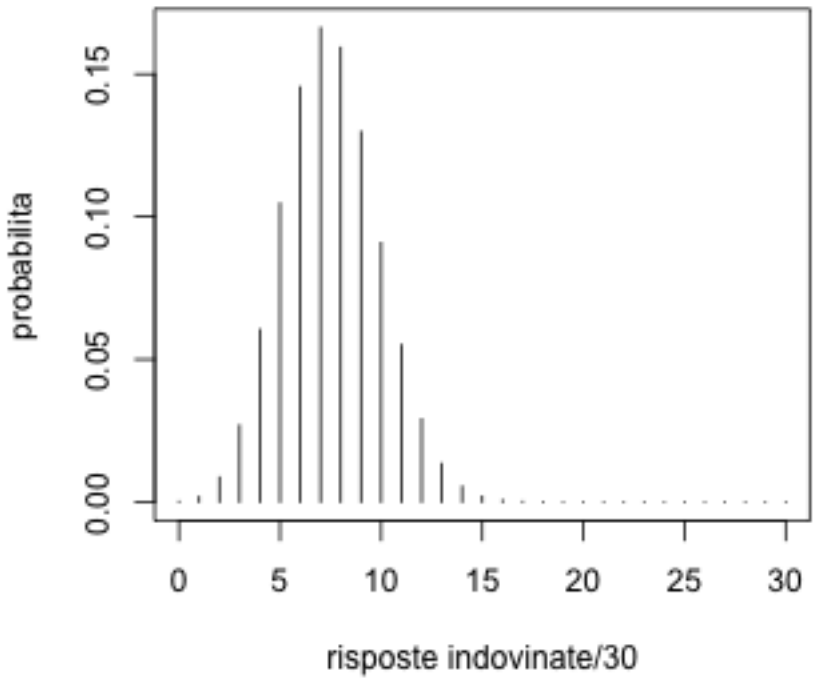
per la probabilità di più di 11 o più risposte corrette, posso scrivere

```
> 1 - pbinom(10, 30, 1/4)

[1] 0.1057281
```

Per valutare dopo quante risposte la prestazione casuale diventa improbabile, posso esaminare la distribuzione in questa maniera:

```
> x <- 0:30
> y <- dbinom(x, 30, 1/4)
> plot(x, y, type = "h")
```

Per sapere il numero di risposte che ha una probabilità minore di 0.05 di



essere ottenuto dallo studente che risponde a caso, possiamo scrivere

```
> qbinom(0.95, 30, 1/4)

[1] 12
```

Infine, per simulare 100 lanci di una moneta “onesta”, possiamo scrivere

```
> c <- rbinom(100, 1, 0.5)
```

dove 0 rappresenta un fallimento (testa o croce che sia) e 1 un successo. Guardiamo il risultato:

```
> table(c)

c
 0  1
55 45
```

Naturalmente, se ripetiamo i cento lanci il risultato può cambiare:

```
> c <- rbinom(100, 1, 0.5)
```

```
> table(c)

c
 0  1
51 49
```

6.3 Alcune altre distribuzioni

Per le altre distribuzioni si può trovare abbastanza facilmente le informazioni sull'help in linea, ricordando sempre che sono denominate in analogia a quello che abbiamo visto per la normale e la binomiale. Ad esempio, alcune delle distribuzioni che si utilizzano più di frequente sono:

dchisq(), pchisq(), qchisq(), rchisq() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione chi-quadrato

dexp(), pexp(), qexp(), rexp() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione esponenziale

dunif(), punif(), qunif(), runif() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione uniforme

dt(), pt(), qt(), rt() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione t di Student

df(), pf(), qf(), rf() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione F di Fisher

dpois(), ppois(), qpois(), rpois() - densità, probabilità cumulativa, inversa, ed estrazione casuale da una distribuzione di Poisson

Gli argomenti da utilizzare variano leggermente in funzione dei parametri di ogni distribuzione, e possono essere facilmente recuperati dall'help in linea. Useremo ancora alcune di queste distribuzioni nella sezione successiva, mentre altre vi torneranno molto utili nel prossimo corso di Tecniche di Analisi di Dati.

7. Distribuzioni campionarie

Nella ricerca psicobiologica osserviamo campioni di misurazioni, ma siamo interessati a generalizzare l'informazione fornita dal campione a una popolazione di riferimento. Questo è possibile se il campione non è affetto da errore sistematico di campionamento, e se è possibile avere un'idea di come le nostre osservazioni potrebbero variare se osservassimo non un campione solo, ma ripetuti campioni. Il primo requisito richiede cautele metodologiche nell'operazione di campionamento, e non può essere in nessuna maniera garantito da calcoli statistici successivi alle misurazioni. In questa sezione lavoreremo sempre con campioni casuali semplici (simple random samples). In un campione casuale semplice, per ogni elemento della popolazione di riferimento il fatto di essere inserito o non inserito nel campione dipende solo da un processo stocastico noto, grazie al quale ogni elemento ha la stessa probabiità di tutti gli altri elementi di essere inserito nel campione. Questo garantisce che il campione non sia in alcun modo affetto da errore sistematico, ma solo dall'errore casuale. Il secondo requisito dipende da considerazioni teoriche, da informazioni a priori sul fenomeno che stiamo osservando, e dal teorema del limite centrale, grazie al quale possiamo assumere delle medie di ripetuti campioni in molti casi può essere bene approssimata usando la distribuzione normale.

In tutto quanto segue, ricordate sempre di tenere ben distinte nella vostra mente tre diverse distribuzioni: la popolazione, che può essere anche una distribuzione teorica ma che in un contesto pratico di ricerca è quasi sempre ignota; il campione, che è composto dalle misure che avete fatto; e la distribuzione campionaria, che è la distribuzione, anch'essa teorica, che si ritiene avrebbe una statistica campionaria, tipicamente la media, se ripetessimo il campionamento un gran numero di volte. Pensate alla distribuzione campionaria come a un costrutto teorico grazie al quale possiamo generalizzare dal singolo campione alla popolazione, perché ci consente di stimare quanto potrebbe variare la nostra misura se ripetessimo le nostre osservazioni. R dispone di alcune funzioni che si rivelano utiissime per simulare il processo di campionamento e l'estrazione di ripetuti campioni, osservando, nella simulazione, la distribuzione campionaria, ossia ciò che nella pratica non osserviamo mai ma dobbiamo assumere, sperando di non essere troppo lontani dalla realtà.

7.1 Estrazione di singoli campioni da una popolazione nota

Iniziamo da un caso semplice. Estraiamo un campione casuale semplice dalla popolazione di tutti i lanci possibili di una moneta onesta ($p = 0.5$). In questo caso, la popolazione è la distribuzione teorica binomiale con $p = 0.5$.

Possiamo visualizzarla immaginando un'urna infinitamente grande, contenente infiniti biglietti. Sulla metà[14] dei biglietti c'è scritto 0, sull'altra metà c'è scritto 1. La simulazione è sorprendentemente semplice. Creiamo un vettore di due elementi, 0 e 1. Possiamo pensare che 0 = croce e 1 = testa, anche se ovviamente è indifferente.

```
> x <- 0:1
```

Usando la funzione sample(), estraiamo un campione con rimpiazzamento. Questo equivale a dire che, invece che avere un'urna con infiniti biglietti, ne abbiamo una con due. Ne estraiamo uno, annotiamo cosa c'è scritto, e lo rimettiamo dentro l'urna. Dato che reinseriamo il bliglietto, la probabiità di osservare 1 o 0 rimane sempre la stessa, indipendentemente da quante volte ripetiamo l'operazione. Dunque l'urna con due biglietti diventa a tutti gli effetti equivalente all'urna con infiniti biglietti. La funzione sample ha sintassi sample(x, size, replace = FALSE), dove x è il vettore da cui campionare, size la numerosità del campione, e replace è l'argomento che controlla se l'estrazione viene fatta con o senza rimpiazzamento (di default, size = FALSE dunque l'estrazione è senza rimpiazzamento).

```
> s <- sample(x, size = 100, replace = TRUE)
```

Esaminiamo la distribuzione del campione:

```
> table(s)/length(s)

s
    0     1
0.64 0.36
```

In questo campione ci sono molte più croci che teste. Ma se avete provato a riprodurre l'operazione sul vostro computer, presumo che la vostra distribuzione sia diversa dalla mia. C'è un problema? Non necessariamente. Abbiamo estratto due campioni diversi, non è strano che i risultati non siano gli stessi. Anzi, deve essere proprio così. Ogni volta che estraggo un campione, la distribuzione sarà diversa a causa dell'errore casuale. Per convincerci di questo, ripetiamo il campionamento:

```
> s <- sample(x, size = 100, replace = TRUE)
> table(s)/length(s)

s
```

[14] La metà di infinito è sempre infinita, ma non chiedete all'autore di questa dispensa la dimostrazione di questo fatto, che rimane profondamente controintuitivo.

```
    0     1
0.49 0.51
```

In questo secondo campione, teste e croci sono molto più bilanciate. Nel vostro, non so, ma presumo che sia diverso dal precedente. Proviamo a fare anche una cosa diversa. Estraimo un terzo campione, ma questa volta aumentiamo drasticamente la numerosità.

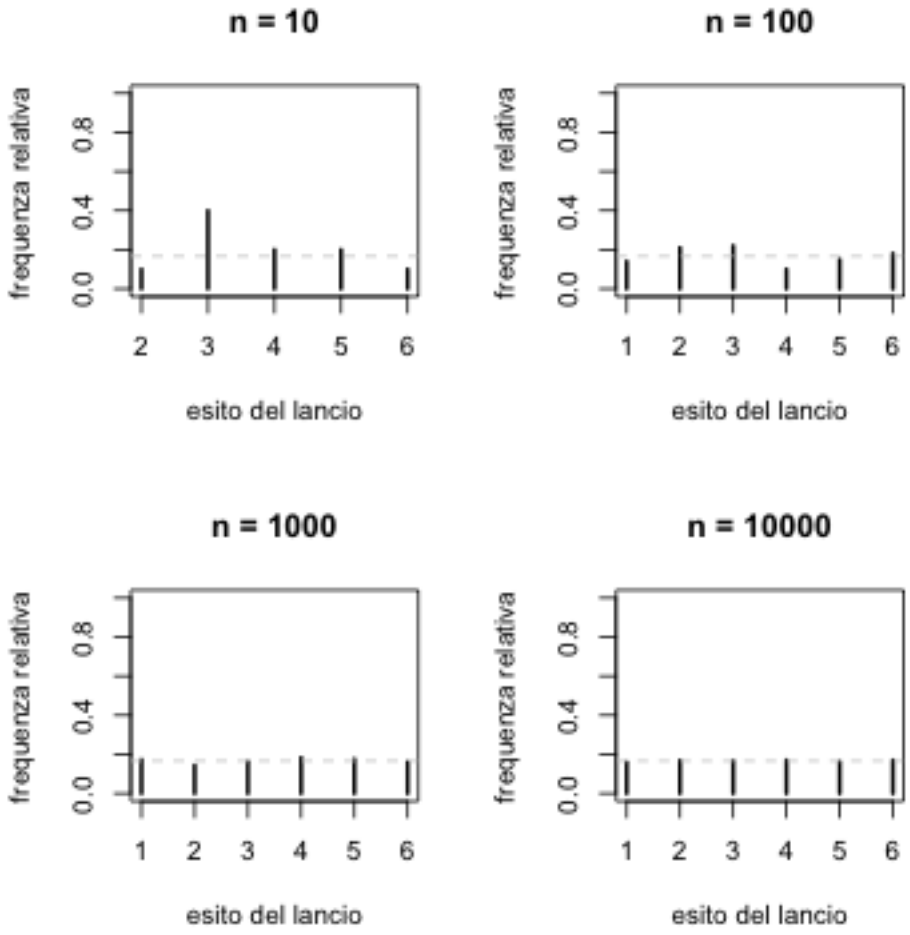
```
> ss <- sample(x, size = 10000, replace = TRUE)
> table(ss)/length(ss)

ss
      0       1
0.5033 0.4967
```

Rifacciamolo ancora una volta:

```
> ss <- sample(x, size = 10000, replace = TRUE)
> table(ss)/length(ss)

ss
      0       1
0.4981 0.5019
```

Come ci aspettavamo, ad ogni estrazione la distribuzione è diversa. Ma notate due cose: la differenza fra queste due ultime distribuzioni è molto più piccola delle differenze osservate nei primi due; le frequenze relative osservate sono molto più simili alle frequenze teoriche della popolazione. Si tratta del primo esempio di due principi generali, su cui torneremo a breve: al cresecere della numerosità di un campione, la precisione della stima aumenta e la distribuzione del campione assomiglia sempre di più alla popolazione da cui è stato estratto.

Facciamo un altro esempio. Proviamo ad estrarre campioni casuali semplici dalla popolazione di tutti gli esiti possibili di un lancio di un dado onesto. In questo caso, la popolazione è la distribuzione uniforme della variabile discreta x = 1, 2, 3, 4, 5, 6. Estrarremo quattro campioni, con numerosità crescente (10, 100, 1000, 1000). Procediamo come per l'esempio precedente, ma per velocizzare la cosa scriviamo un programma e visualizziamo le distribuzioni con un barplot a cui aggiungeremo una linea tratteggiata a marcare il valore della frequenza relativa nella popolazione. Di seguito i quattro grafici che ho ottenuto e poi il programma, che dovete copiare, incollare, ed eseguire per ottenere i vostri quattro, che saranno simili ma non esattamente uguali.



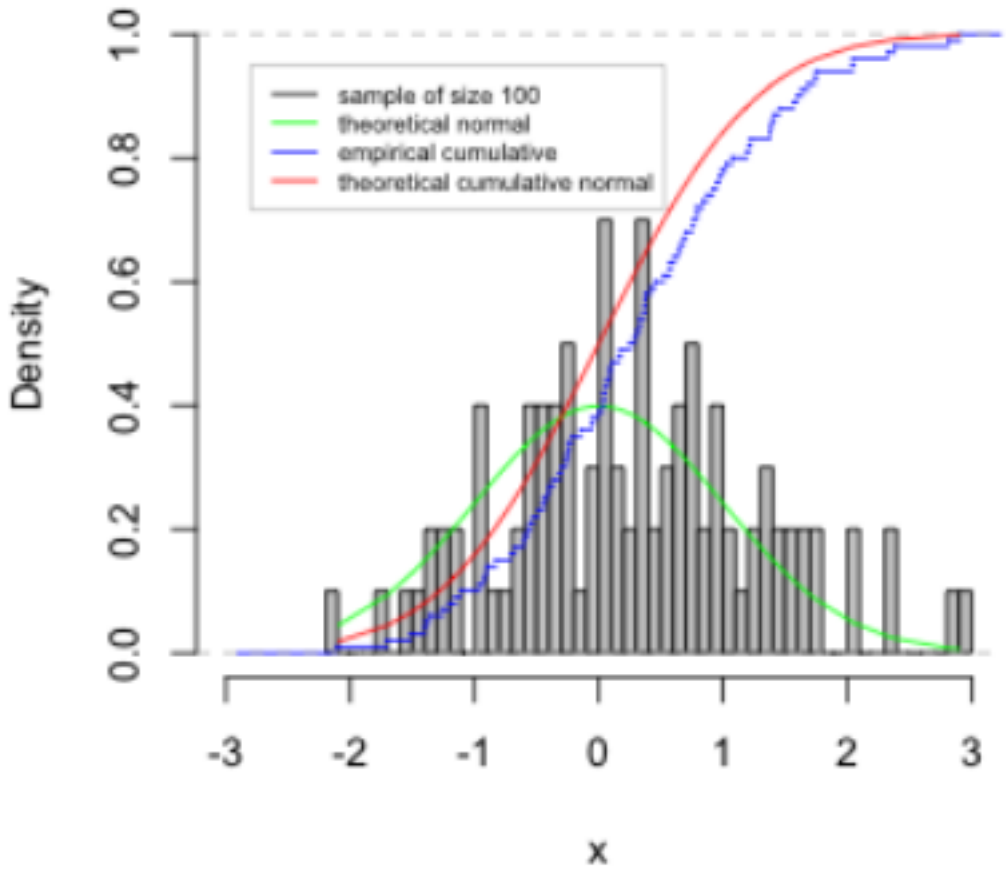

```
# il dado
x <- 1:6
# le distribuzioni dei quattro campioni
s1 <- table(sample(x, 10, replace = 1))/10
s2 <- table(sample(x, 100, replace = 1))/100
s3 <- table(sample(x, 1000, replace = 1))/1000
s4 <- table(sample(x, 10000, replace = 1))/10000
# prima di plottare salviamo il vecchio par
op <- par(mfrow=c(2,2))
# ora plottiamo
plot(s1, type = "h", xlab = "esito del lancio", ylab =
"frequenza relativa", main = "n = 10", ylim = c(0, 1))
abline(h = 1/6, col = "grey", lty = "dashed")
plot(s2, type = "h", xlab = "esito del lancio", ylab =
"frequenza relativa", main = "n = 100", ylim = c(0, 1))
abline(h = 1/6, col = "grey", lty = "dashed")
plot(s3, type = "h", xlab = "esito del lancio", ylab =
"frequenza relativa", main = "n = 1000", ylim = c(0, 1))
```

```
abline(h = 1/6, col = "grey", lty = "dashed")
plot(s4, type = "h", xlab = "esito del lancio", ylab =
"frequenza relativa", main = "n = 10000", ylim = c(0, 1))
abline(h = 1/6, col = "grey", lty = "dashed")
par(op)  # rimettiamo a posto
```

Se studiate il vostro grafico e lo confrontate con il mio, dovreste constatare che valgono i principi citati prima. Al crescere della numerosità, diminuisce la differenza fra il mio e il vostro campione e tutto diventa più simile alla popolazione.

Naturalmente nelle nostre simulazioni non dobbiamo limitarci ai giochini da tavolo. Possiamo usare tutte le distribuzioni teoriche di R per simulare popolazioni note. In realtà, abbiamo fatto tutta quella fatica a studiare le relative funzioni proprio per essere in grado di fare questo. Proviamo ad estrarre un campione n = 100 dalla distribuzione normale standard. Dopo averlo estratto lo guardiamo con un istogramma cui sovrapponiamo sia la distribuzione cumulativa empirica, quella del campione, sia la distribuzione normale teorica e la relativa cumulativa teorica. Potete divertirvi a modificare n, la grandezza del campione, per verificare come cambia la sua distribuzione rispetto a quella della popolazione.

```
n <- 100             # sample size
b <- 50             # classes for histogram
scol  <- "black"    # color for data histogram
tndcol <- "green"   # color for normal density
ecdcol <- "blue"    # color for cumulative
tcdcol <- "red"     # color for theoretical cumulative
x <- sort(rnorm(n)) # sort the sample for plotting the curves
hist(x, breaks = b, prob = TRUE, ylim = c(0,1), xlim = c(-3,3),
main = NULL, col = "grey")
lines(x, dnorm(x), col = tndcol)
lines(ecdf(x), col = ecdcol, pch=".")
lines(x, pnorm(x), col = tcdcol)
legend(-2.8, 0.95,
          legend=c(paste("sample of size", n),
                    "theoretical normal",
                    "empirical cumulative",
                    "theoretical cumulative normal"),
          lty = "solid",
          col = c(scol, tndcol, ecdcol, tcdcol),
          xjust = 0, cex=0.6, box.col="grey")
```



7.2 Distribuzione campionaria della media

A questo punto dovrebbe essere chiaro come simulare l'estrazione di un campione da una popolazione nota. Proviamo allora a vedere come simulare una distribuzione campionaria. Quello che vogliamo fare è: i) estrarre ripetuti campioni da una popolazione nota; ii) per ogni campione calcolare la media aritmetica; iii) osservare come è fatta la distribuzione delle medie di questi campioni, la distribuzione campionaria della media. Sono tutte cose che sappiamo già fare. Ad esempio, supponiamo di voler simulare l'estrazione dalla normale con media = 100 e sd = 15 di 1000 campioni di 30 osservazioni. Per il primo campione potremmo scrivere

```
> s1 <- rnorm(30, 100, 15)
> m1 <- mean(s1)
```

per il secondo

```
> s2 <- rnorm(30, 100, 15)
> m2 <- mean(s2)
```

e cos' via. Alla fine creiamo un vettore e facciamo l'istogramma della distribuzione. Con due medie è semplice:

```
> v <- c(s1, s2)
> hist(v)
```

Ma se ne vogliamo 1000, la cosa non è pratica, come avrete capito. Per ripetere la stessa operazione molte volte, possiamo usare la funzione for(). La sua sintassi è for(i in v) { funzioni }, dove v è un qualsiasi vettore, i uno dei suoi valori, e le parentesi grafe racchiudono quante funzioni volete. Usando for(), dite ad R di eseguire tutto quello che sta fra le parentesi tante volte quanti sono gli elementi di v. Ad ogni nuovo ciclo, l'indice i cambia dal primo all'ultimo degli elementi di v, e noi possiamo usare questo indice per identificare su cosa vogliamo lavorare. Ad esempio, per i nostri 1000 campioni possiamo scrivere:

```
> sm <- matrix(nrow = 1000, ncol = 30)  # una matrice vuota
> for (i in 1:1000) { sm[i, ] <- rnorm(30, 100, 15)}
> mns <- apply(sm, 1, mean)
```

il vettore msn contiene 1000 medie campionarie. Diamo un'occhiata:

```
> length(mns)

[1] 1000

> fivenum(mns)

[1]  90.99628  98.21707  99.99409 101.84643 108.30421

> mean(mns)

[1] 99.99751

> sd(mns)

[1] 2.574389
```

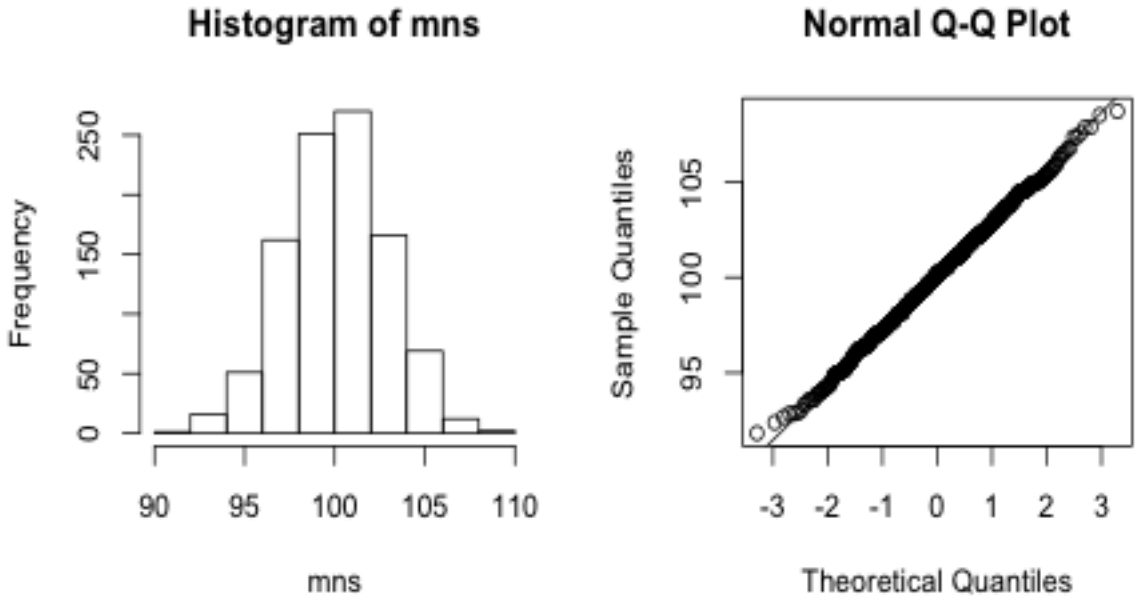
Notare che la distribuzione campionaria di 1000 medie ha praticamente la stessa media della popolazione da cui ho tratto i campioni, ma deviazione

standard più piccola. In base al teorema del limite centrale, dovrebbe essere circa pari a sigma, la sd della popolazione, diviso la radice quadrata della numerosità. Verifichiamolo:

```
> 15/sqrt(30)

[1] 2.738613
```

Vediamo ora l'istogramma, e confrontiamo la distribuzione campionaria con una distribuzione normale:

```
> op <- (par(mfrow=c(1,2)))
> hist(mns)
> qqnorm(mns)
> qqline(mns)
> par(op)
```



La distribuzione campionaria assomiglia moltissimo alla distribuzione normale.

7.3 Teorema del limite centrale

Nella sezione precedente, abbiamo provato a simulare l'estrazione di ripetuti campioni da una popolazione normale. Abbiamo osservato che la distribuzione delle medie campionarie ha caratteristiche ben precise: i) la media della distribuzione campionaria assomiglia molto alla media della popolazione da cui sono stati estratti i campioni (nel nostro esempio, 100);

la deviazione standard della distribuzione campionaria è più piccola della deviazione standard della popolazione, di un fattore pari alla radice quadrata di questa deviazione standard (nel nostro esempio, la deviazione standard della popolazione era 15, ma la deviazione standard del campione circa 15/sqrt(30); la forma della distribuzione campionaria era abbastanza simile alla distribuzione normale.

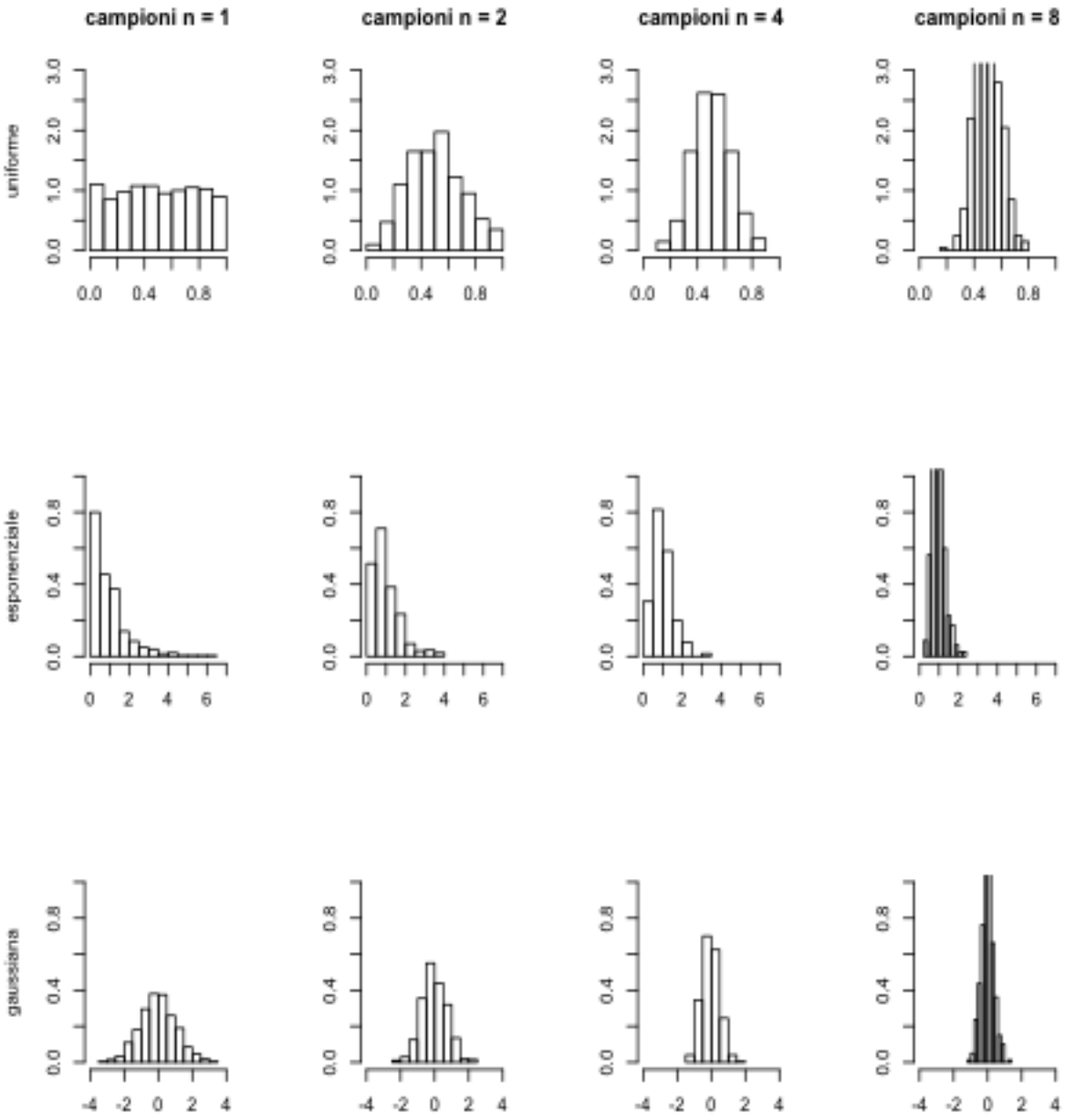
Nel caso dell'esempio precedente, questo potrebbe sembrare non particolarmente soprendente. I campioni sono stati estratti da una normale, la distribuzione campionaria è anch'essa normale. Il principio tuttavia vale più in generale. Qualsiasi sia la forma della distribuzione di partenza, al crescere di n, la numerosità dei campioni, la distribuzione campionaria della media assomiglia sempre più alla distribuzione normale con media = media della popolazione e deviazione standard = deviazione standard della popolazione diviso la radice quadrata della numerosità dei campioni. Questo principio si chiama teorema del limite centrale[15], ed è il principio fondamentale grazie al quale è possibile usare la media campionaria per stimare la media di una popolazione ignota, e per misurare l'incertezza di questa stima (vedi sezione successiva).

Proviamo a verificarlo confrontando le distribuzioni campionarie estratte da una popolazione uniforme o rettangolare, da una distribuzione esponenziale, e dalla normale. Nel programma esploriamo anche una nuova strategia per programmare la simulazione in R. In questo caso non usiamo cicli controllati da for() ma lavoriamo direttamente con le matrici. Creiamo matrici di 8 righe e 400 * 8 colonne. Ogni elemento della prima matrice contiene una estrazione dalla distribuzione uniforme; ogni elemento della seconda, dalla esponenziale; della terza, dalla normale. Quindi creiamo delle matrici che contengono le medie campionarie: la prima colonna contiene semplicemente i 400 elementi della prima riga (considerati come la media di un campione n = 1); la seconda, le medie delle prime due righe; la terza, delle prime 4 righe; la quarta, di tutte le 8 righe. Quindi plottiamo.

```
rmt1 <- matrix(runif(400*8), nrow=8)
rmt2 <- matrix(rexp(400*8),  nrow=8)
rmt3 <- matrix(rnorm(400*8), nrow=8)

mns1 <- cbind(rmt1[1,],
              apply(rmt1[1:2, ], 2, mean),
              apply(rmt1[1:4, ], 2, mean),
              apply(rmt1[1:8, ], 2, mean)
```

[15] Una formulazione più compatta è la seguente: data una popolazione arbitraria con media μ e deviazione standard σ, la distribuzione campionaria delle medie di campioni di n elementi, d(M, n) -> Norm(μ , σ /$\sqrt{n}$).

```
                )

mns2<- cbind(rmt2[1,],
                apply(rmt2[1:2, ], 2, mean),
                apply(rmt2[1:4, ], 2, mean),
                apply(rmt2[1:8, ], 2, mean)
                )

mns3  <- cbind(rmt3[1,],
                apply(rmt3[1:2, ], 2, mean),
                apply(rmt3[1:4, ], 2, mean),
                apply(rmt3[1:8, ], 2, mean)
                )

par(mfrow=c(3,4))

hist(mns1[,1],xlim=c(0,1), ylim=c(0,3), prob=TRUE,
main="campioni n = 1", xlab="", ylab="uniforme")
hist(mns1[,2],xlim=c(0,1), ylim=c(0,3), prob=TRUE,main="campioni
n = 2", xlab="", ylab="")
hist(mns1[,3],xlim=c(0,1), ylim=c(0,3), prob=TRUE,
main="campioni n = 4", xlab="", ylab="")
hist(mns1[,4],xlim=c(0,1), ylim=c(0,3), prob=TRUE,
main="campioni n = 8", xlab="", ylab="")

hist(mns2[,1],xlim=c(0,7), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="esponenziale")
hist(mns2[,2],xlim=c(0,7), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")
hist(mns2[,3],xlim=c(0,7), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")
hist(mns2[,4],xlim=c(0,7), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")

hist(mns3[,1],xlim=c(-4,4), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="gaussiana")
hist(mns3[,2],xlim=c(-4,4), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")
hist(mns3[,3],xlim=c(-4,4), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")
hist(mns3[,4],xlim=c(-4,4), ylim=c(0,1), prob=TRUE, main="",
xlab="", ylab="")
```



Come previsto, la forma della distribuzione tende ad assomigliare sempre più alla normale quando n cresce. Con n =1, la distribuzione campionaria è la stessa cosa della distribuzione di un solo campione con n = 400. Come abbiamo già visto, al crescere della numerosità di un campione, la distribuzione di questo campione tende ad assomigliare sempre più alla distribuzione della popolazione[16]. Infatti, per n = 1 nella prima riga la distribuzione campionaria assomiglia alla distribuzione uniforme, nella seconda, alla esponenziale. Con n > 1, rapidamente la forma della distribuzione campionaria muta e tende a diventare simile ad una campana.

[16] Quest'altro principio, che riguarda la distribuzione del singolo campione, non la distribuzione campionaria, viene talvolta chiamato la legge dei grandi numeri.

Questo avviene più rapidamente se la distribuzione della popolazione è simmetrica, meno rapidamente se è asimmetrica.

7.4 Intervalli di fiducia

Grazie al teorema del limite centrale, sappiamo che la media della distribuzione delle medie di ripetuti campioni è uguale alla media della popolazione. Questo non garantisce naturalmente che la media di un singolo campione sia uguale alla media della popolazione. Tipicamente non lo sarà. Ma garantisce invece che la differenza sia dovuta solo all'errore variabile di campionamento, e non ad un errore sistematico di sovrastima o sottostima. Ma quanto può essere grande l'errore variabile di campionamento? Dipende dalla distribuzione campionaria della media. Più bassa è la sua deviazione standard, minore sarà l'errore. Grazie al teorema del limite centrale, sappiamo che la deviazione standard della distribuzione campionaria è uguale alla deviazione standard della popolazione, divisa per la radice quadrata della numerosità del campione. Possiamo usare questo fatto per stimare l'errore variabile di campionamento, e calcolare una "finestra" di incertezza all'interno della quale deve stare la media della popolazione, con un certo grado di probabilità. Questa "finestra" viene chiamata intervallo di fiducia, e il grado di probabilità di solito viene fissato al 95%.

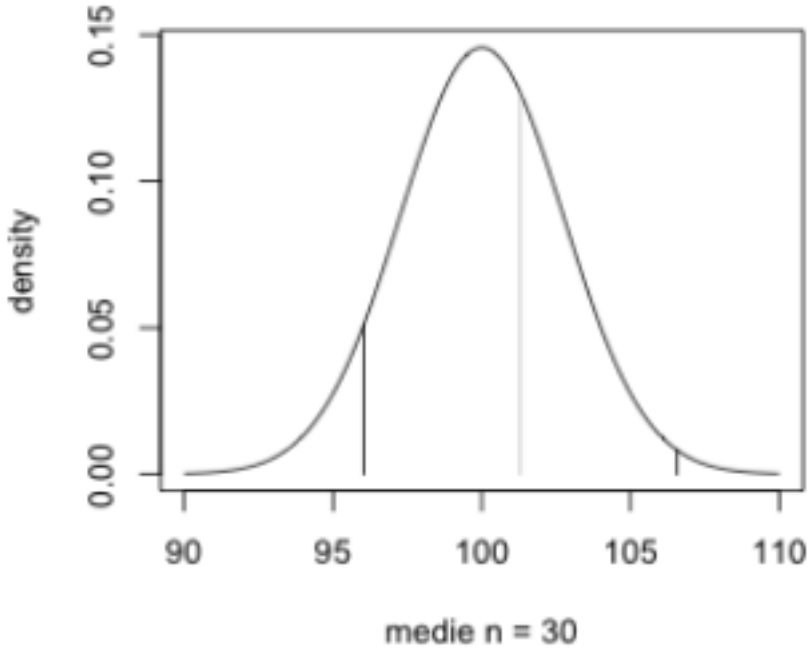
Facciamo un esempio. In un'altra stanza, un vostro amico estrae con R un campione di numerosità n = 30 da una distribuzione normale con media = 100 e deviazione standard = 15.

```
> s <- rnorm(30, 100, 15)
```

Ora vi chiama nella stanza, e davanti al suo portatile e vi lancia una sfida. A partire dal vettore s, il campione, dovete indovinare qual era la media della popolazione da cui era stato estratto. Una maniera velocissima di fare questo è la seguente:

```
> t.test(s)

data:  s
t = 39.4206, df = 29, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  96.03576 106.54616
sample estimates:
mean of x
  101.291
```

la funzione t.test() non è, come avrete capito, una funzione specializzata per il calcolo di un intervallo di fiducia. Ma fra le tante altre cose fa anche questo. Sotto alla riga che dice “95 percent confidence interval” ci sono due numeri. Quei numeri sono i limite inferiore e superiore dell’intervallo. La risposta giusta a una simile sfida è che indovinare esattamente non si può, ma è possibile calcolare un intervallo all’interno del quale la media della popolazione dovrebbe trovarsi, con un certo grado di probabilità. Per la precisione, dire che, con una probabilità = 0.95, la media della popolazione sta fra (circa) 96 e 107 equivale ad affermare che, se estraessi 100 campioni e calcolassi le rispettive medie campionarie, in 95 di questi la media della popolazione sarebbe compresa nell’intervallo. Nei cinque rimanenti, commetterei un errore. Proviamo a disegnare una normale con media 100 e deviazione standard 15/sqrt(30), e a localizzare con dei segmenti la media del campione del vostro amico e i limiti dell’intervallo di fiducia:



Se non vi fosse chiaro (li abbiamo già visti), per produrre il grafico ho usato i comandi qui sotto:

```
> x <- seq(90, 110, 0.1)
> y <- dnorm(x, 100, 15/sqrt(30))
> plot(x, y, type = "l", xlab = "medie n = 30", ylab = "density")
> liminf <- 96.03576
> limsup <- 106.54616
```

```
> segments(liminf, 0, liminf, dnorm(liminf, 100, 2.738613))
> segments(limsup, 0, limsup, dnorm(limsup, 100, 2.738613))
> segments(mean(s), 0, mean(s), dnorm(mean(s), 100, 2.738613), col =
"grey")
```

Vediamo in dettaglio come è stato calcolato l'intervallo di fiducia. Come già detto, per stimare quanto potrebbe cambiare la media in altri campioni che non ho estratto, devo sapere come è fatta la distribuzione campionaria della media. Per il teorema del limite centrale, so che ha media uguale alla media del mio campione, a meno di un errore casuale che dipende dalla dispersione (la deviazione standard) della distribuzione campionaria. So anche che questa deviazione standard è uguale alla deviazione standard della popolazione, diviso la radice quadrata della numerosità del campione. Il calcolo dell'intervallo di fiducia è quindi

```
> liminf <- mean(s) - qnorm(0.975) * 15/sqrt(30)
> limsup <- mean(s) + qnorm(0.975) * 15/sqrt(30)
> liminf

[1] 95.92338

> limsup

[1] 106.6585
```

ma osservate con attenzione i limiti ottenuti. Assomigliano molto, ma non sono identici a quelli che avevamo trovato usando t.test(). Abbiamo sbagliato qualcosa? Non proprio. Il problema è che per calcolare l'intervallo abbiamo usato la deviazione standard della popolazione (15). Ma se ricordate come è stato posto il problema, il vostro amico non vi ha detto come era fatta la popolazione da cui ha estratto il campione. Se non conosco la deviazione standard della popolazione, devo usare il campione per stimarla, esattamente come ho fatto per la media. Inoltre, la distribuzione campionaria non è esattamente la distribuzione normale, ma la distribuzione t di Student con 30-1 gradi di libertà. Il calcolo che ha usato t.test è quindi:

```
> liminf <- mean(s) - qt(0.975, 29) * sd(s)/sqrt(30)
> limsup <- mean(s) + qt(0.975, 29) * sd(s)/sqrt(30)
> liminf

[1] 96.03576

> limsup
```

```
[1] 106.5462
```

che da esattamente i valori di prima (a meno dell'approssimazione al terzo decimale, per il limite superiore).

Usando R, è facile simulare l'estrazione di ripetuti campioni e calcolare gli intervalli di fiducia. Qui sotto trovate un programma che lo fa estraendo i campioni dalla distribuzione uniforme con media = 0.5. Se provate a farlo girare un po' di volte, noterete che la percentuale di campioni che copre la media della popolazione non è mai molto lontana dal 95%. Se provate a modificarlo per simulare estrazioni da una popolazione normale, noterete che la percentuale si avvicina ancora di più al 95%, mentre se usate una distribuzione fortemente asimmetrica, ad esempio, la esponenziale (il default di R ha media = 1), il metodo funziona un po' meno bene, ma comunque ancora abbastanza bene.

```
nc <- 1000  # quanti campioni?
ns <- 30 # quante osservazioni per campione?
a <- qt(0.975, ns-1) # il valore di t che mi serve
sqns <- sqrt(ns) # la radice di n
sm <- matrix(nrow = nc, ncol = ns)  # una matrice vuota
for (i in 1:nc) { sm[i, ] <- runif(ns)}
mns <- apply(sm, 1, mean)
sds <- apply(sm, 1, sd)
liminf <- mns -  a * sds/sqns
limsup <- mns + a * sds/sqns
hits <- length(mns[liminf < 0.5 & limsup > 0.5])
print(hits/nc*100) # % di IF che coprono la media della popolazione
```

7.5 Riportare la media in un report scientifico

Quando si riportano le statistiche di un campione di misurazioni, bisogna sempre integrarle con informazioni sulla precisione della stima. Di solito, questo viene fatto riportando la media assieme alla la stima della deviazione standard della distribuzione campionaria. La deviazione standard della distribuzione campionaria di una statistica viene chiamata anche il suo errore standard. Ad esempio, la media del campione s, descritto nella sezione precedente, è

```
> m <- mean(s)
> m

[1] 101.291
```

l'errore standard della media è

```
> esm <- sd(s)/sqrt(30)
> esm
```

```
[1] 2.569491
```

tipicamente, in un report scientifico questa misura verrebbe riportata con una frase come questa: "la media del campione è 101.3 ± 2.6".

8. Lavorare con i grafici in R

In questa sezione vengono presentati alcuni concetti generali riguardanti l'uso della grafica di R. Vengono anche presentati numerosi esempi relativi ai tipi di funzione nell'ambiente standard di R. Abbiamo già provato ad usare alcune di queste funzioni. Qui descrivo più in dettaglio come usarne gli argomenti per controllare le diverse componenti di un grafico. Le funzioni grafiche di R sono estremamente sofisticate, e qui approndiremo solo quelle di base necessarie per il corso. Darò comunque alcuni accenni anche ad alcune funzioni più complesse, il cui utilizzo potrà essere affrontato da chi si trovasse in futuro ad avere dei problemi che le richiedono.

8.1 I data set preinstallati

Scrivere

>data()

dovrebbe comparire qualcosa di simile alla figura qui sotto :

il pacchetto 'datasets', installato assieme ad R, contiene file con esempi di

dati. Scrivere

```
>data(cars)
```

abbiamo caricato il dataset cars. Il dataset è un oggetto di R e quindi per scoprire cos'è posso dire

```
> help(cars)
```

notate che l'help spiega il significato dei dati e specifica il formato. Cars è un data frame, ha 50 righe e 2 colonne. Infatti

```
>dim(cars)

[1] 50  2
```

la prima colonna è la velocità (miglia/ora) e la seconda è la distanza di arresto in piedi. Infatti

```
>names(cars)

[1] "speed" "dist"
```

diamo un'occhiata alle distribuzioni

```
>summary(cars)

     speed           dist
 Min.   : 4.0   Min.   :  2.00
 1st Qu.:12.0   1st Qu.: 26.00
 Median :15.0   Median : 36.00
 Mean   :15.4   Mean   : 42.98
 3rd Qu.:19.0   3rd Qu.: 56.00
 Max.   :25.0   Max.   :120.00
```

il data set è vecchiotto, la velocità massima è 25 m/h (circa 40 km/h). Consultando l'help potete vedere che si riferisce a dati del 1920. Infatti è un esempio classico dei manuali di statistica.

Provate ad esplorare questo e altri data set di R. Nelle sezioni successive ne useremo diversi per illustrare le funzioni grafiche in dettaglio.

Oltre al pacchetto data sets ve ne sono molti altri disponibili. Provate ad esplorare il vostro programma per capire come accedervi.

8.2. Le librerie di R

Scrivere

```
>library(graphics)
>help(graphics)
```

leggere l'help in linea. Il pacchetto, o libreria, chiamato graphics contiene tutte le funzioni standard (o "tradizionali") di R e molte altre possibilità per grafici non standard (o "moderni"). Per vedere la lista completa di queste funzioni, scrivere

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

Scorrete la lista, abbiamo già visti alcune di queste funzioni in azione. Ricordate che per ognuna delle funzioni elencate potete sempre leggere l'help in linea. Non è quindi necessario memorizzare nulla.

Il pacchetto graphics viene caricato automaticamente quando lanciate R, quindi in realtà non occorre caricarlo di nuovo come abbiamo fatto adesso. Esplorate il vostro programma per capire quali sono i pacchetti che vengono caricati automaticamente e come potete caricarne altri. Ad esempio, alcune funzioni molto utili sono contenute in un package chiamato Hmisc, che non viene di solito caricato automaticamente.

In questa dispensa abbiamo usato le funzioni standard nel pacchetto graphics. R offre anche altre possibilità. Scrivete

```
>library(lattice)
>help(lattice)
```

Il package chiamato lattice offre funzioni per creare grafici di tipo Trellis, ossia grafici multipli organizzati su una singola pagina per la visualizzazione di strutture multidimensionali in una sorta di grigilia o tabella (in inglese, trellis vuol dire graticcio o pergolato). Leggere l'help per saperne di più. L'utilizzo delle funzioni nel pacchetto lattice è più complicato rispetto alle funzioni standard. Un altro pacchetto che contiene funzioni grafiche non standard è vcd, che abbiamo già citato nella sezione sulle distribuzioni bivariate di variabili categoriali.

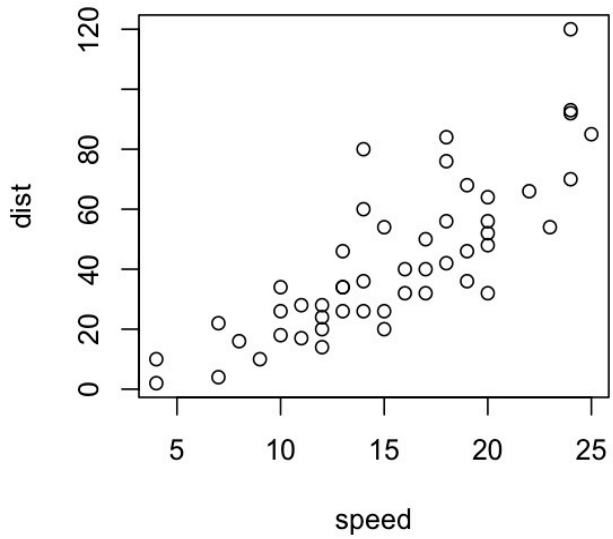
8.3. I tipi di funzione grafica

Per lavorare con la grafica di R è utile distinguere fra tre tipi di funzioni.

Le funzioni di alto livello producono direttamente un intero grafico. Vediamo alcuni esempi.

```
>data(cars)
>plot(cars)
```

Produce un diagramma di dispersione (scattergram) che mostra la relazione fra spazio di frenata e velocità:




```
> hist(cars$speed, prob = TRUE)
```

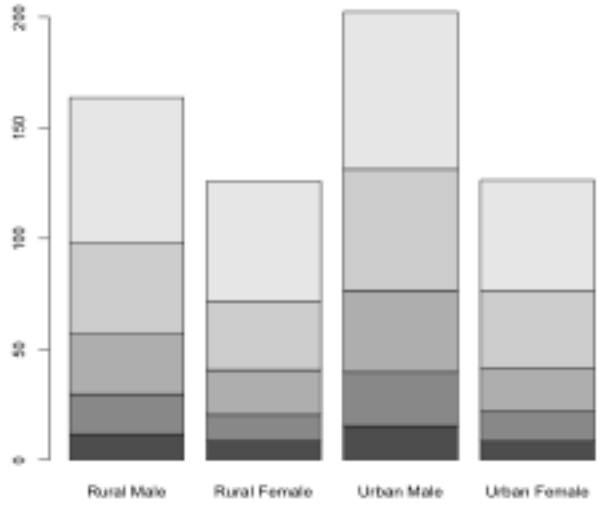
produce l'istogramma ad esempio di speed. Abbiamo già visto questa funzione, studiare l'help per tutte le opzioni. In particolare, le funzioni di alto livello consentono di modificare i nomi degli assi (usando gli argomenti xlab e ylab), il minimo e il massimo degli assi (xlim e ylim), i caratteri e i colori per i simboli plottati (vedremo come fra poco).

```
> boxplot(cars$speed, notch = TRUE)
```

produce il boxplot. Anche questa funzione la abbiamo già vista, l'argomento notch è molto utile perché aggiunge al boxplot l'intervallo di fiducia attorno alla mediana.

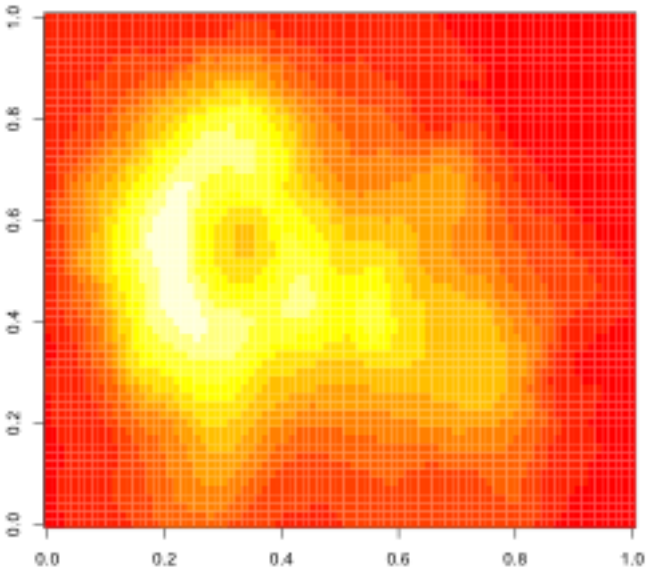
```
>data(VADeaths)
>barplot(VADeaths)
```

Produce un diagramma a barre sovrapposte (stacked bars) come quello in figura 3.3. Come di consueto, utilizzare l'help in linea per capire la stuttura dei dati e cosa fa la funzione.



>data(volcano)
>image(volcano)

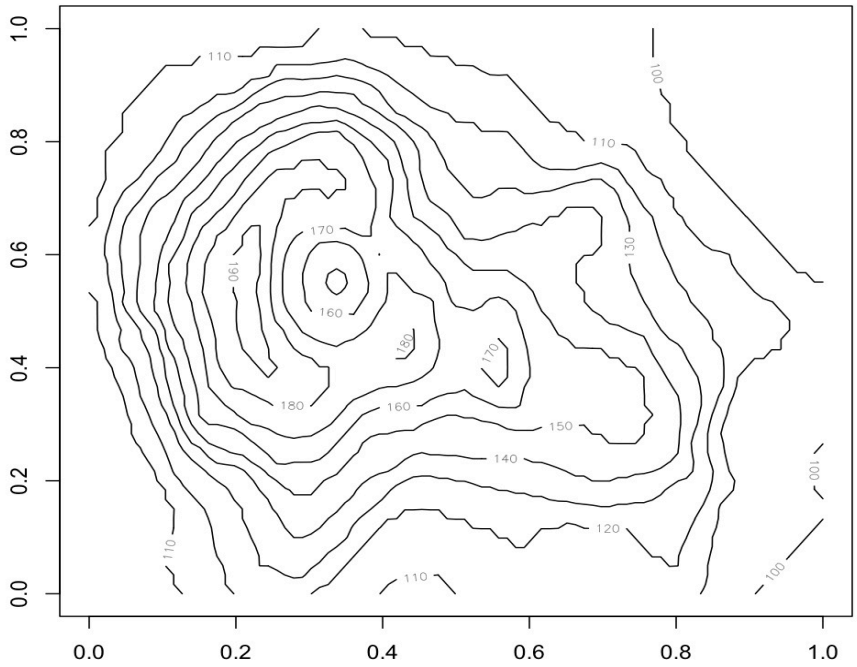
Produce un grafico a falsi colori (Figura 3.4).



>contour(volcano)

produce la figura sotto, in cui gli stessi dati sono presentati come contour

plot.

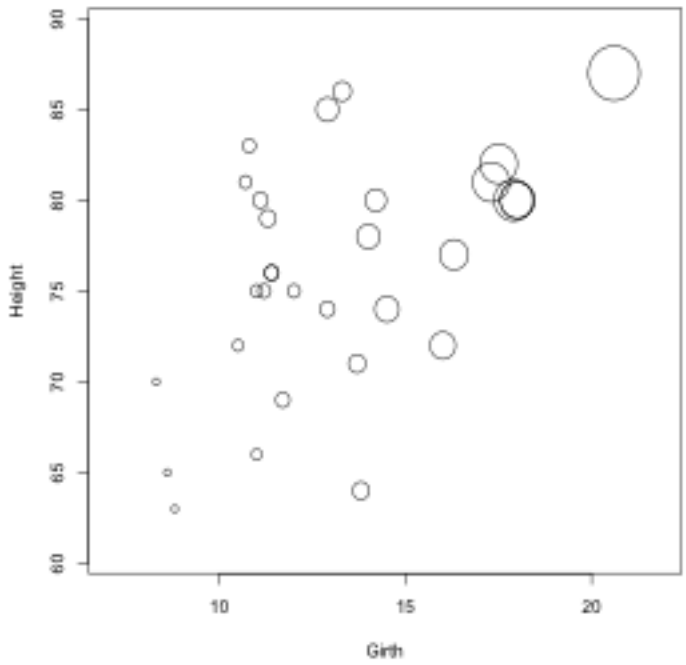


Molte funzioni alto livello producono grafici ragionevoli semplicemente specificando il nome del data set. Questo è molto utile nella fase iniziale di esplorazione, in cui lo scopo non è produrre una presentazione grafica ma capire la struttura dei dati. In altri casi, una funzione di alto livello richiede di specificare alcuni parametri.

Ad esempio, symbols presenta una struttura di dati tridimensionale ponendo sul piano x, y simboli la cui grandezza è proporzionale a z. Ad esempio

```
>data(trees)
>attach(trees)
>symbols(trees, circles = Volume, inches = 0.25)
```

l'argomento circles dice a symbols() che vogliamo plotare dei cerchi, il cui diametro è proporzionale al valore della variabile Volume nel dataset. Possiamo usare anche quadrati o rettangoli, o addirittura boxplots se in ogni punto vogliamo riassumere un sottoinsieme dei dati.



Provate ad esplorare il comportamento della funzione symbols se non passate alcun parametro o se modificate quelli utilizzati.

Grazie al comando example() è possibile vedere esempi preconfezionati di tutte le funzioni di alto livello. Per vedere gli esempi è opportuno usare par(ask=”TRUE”) in modo da mandare avanti la presentazione con la tastiera. Ad esempio

```
>par(ask=”TRUE”)
>example(image)
```

Per rendere il grafico più adatto alla presentazione, tutte le funzioni di alto livello consentono di controllare aspetti di dettaglio specificando opzioni. Ad esempio provate a scrivere

```
>plot (cars, type=”l”)
>plot (cars, type=”b”)
>plot (cars, type=”o”)
>plot (cars, type=”h”)
```

oppure

```
plot(cars,type = "o", xlim = c(0,100), ylim = c(0,200), main =
"datasetcars",  xlab = "velocita (miglia/ora)",  ylab = "spazio di frenata
```

(piedi)", col =" red")

Abbiamo già visto altri esempi di questa maniera di produrre grafici di alto livello nelle sezioni precedenti della dispensa. In generale, la maniera di procedere è per prove ed errori, utilizzando help ed esempi in linea. Inoltre è sempre possibile cercare informazioni in rete. R esiste già da diversi anni e le risposte alle proprie domande sono quasi sempre reperibili in tutorial, faq, o mailing list. Un buon punto di partenza per questo è googlare R-Help.

Le funzioni di basso livello consentono di controllare individualmente tutti gli elementi necessari a costruire un grafico. Ad esempio

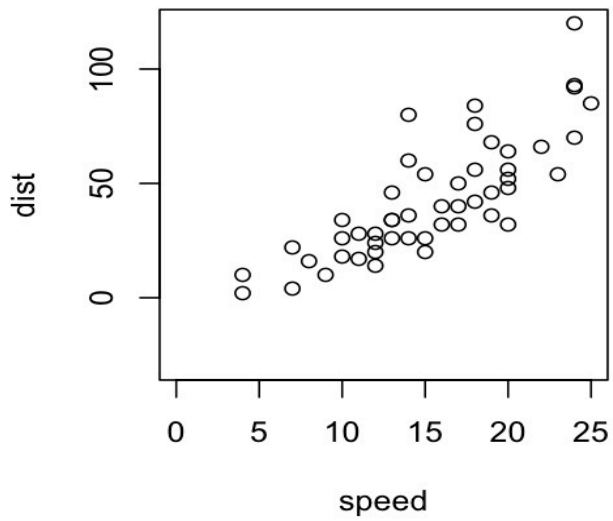
```
> plot.new()
> plot.window(xlim = c(0,1), ylim = c(5,10))
> axis(1)
> axis(2)
> title(xlab = "asse x")
 > title(ylab = "asse y")
> box()
> y<-rnorm(100)
> x<-rnorm(100)
> points(x, y)
```

come di consueto potete usare l'help per i dettagli delle funzioni. Tuttavia a questo punto il funzionamento dovrebbe essere chiaro, soprattutto se controllate cosa succede dopo ogni comando.

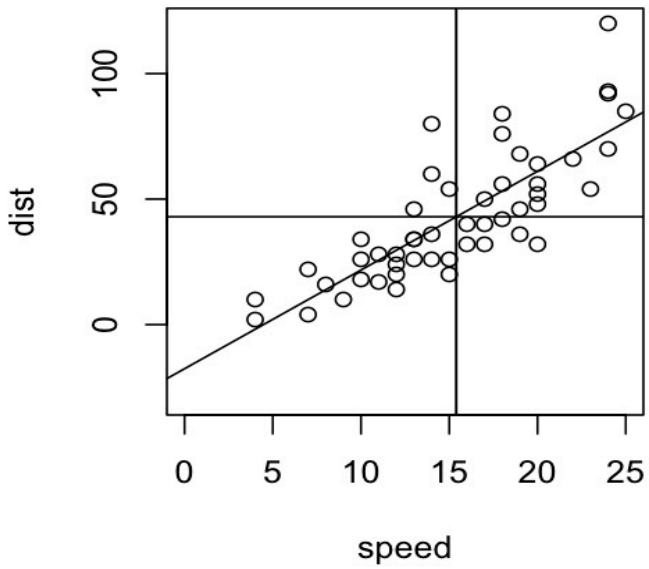
Tutte le funzioni di alto livello sono scritte usando R e sfruttando le funzioni di basso livello. L'utiizzo tipico delle funzioni di basso livello tuttavia non è questo. Tipicamente, invece, le funzioni di basso livello vengono utilizzate per modificare i grafici creati con una funzione di alto livello. Nei manuali di R questa operazione viene chiamata anche "annotazione". La sequenza tipica è come nell'esempio sotto:

```
>data(cars)
>attach(cars)
> plot(cars, xlim = c(0, max(speed)), ylim = c(-30, max(dist)))
```

che produce il diagramma di dispersione nella figura alla pagina successiva (notare la modifica alla gamma dell'asse x).




```
> abline(h=mean(dist))
> abline(v=mean(speed))
> abline(lm(dist~speed))
```



aggiunge al diagramma tre linee (figura 3.8). Una linea orizzontale che marca la media di y, una linea verticale che marca la media media di x, e la retta di regressione. L'intersezione fra le rette delle medie è il centroide della distribuzione bivariata (il "punto delle medie" secondo la definizione di

Freedman. La retta di regressione è la retta che passa per il punto delle medie e per l'intercetta (visualizzabile grazie al fatto che l'asse y ora si estende nella parte negativa di y. Leggere l'help in linea relativo alla funzione abline() per capire come funziona.

Infine, notare che la funzione abline() può avere come argomento anche l'espressione lm(dist~speed). Quest'ultima è una caratteristica specifica di R. L'espressione dist ~ speed in R è una formula che va interpretata come "dist in funzione di speed", mentre la funzione lm indica il fit di un modello lineare. La chiamata di abline significa quindi "traccia una linea sul grafico corrente utilizzando i parametri (intercetta e pendenza) del modello lineare che predice dist in funzione di speed". Infatti

```
> lm(dist ~ speed)
```

produce l'output

```
Call:
lm(formula = dist ~ speed)

Coefficients:
(Intercept)        speed
    -17.579        3.932
```

che contiene i valori numerici dell'intercetta e della pendenza. Notare che la retta di regressione nel grafico tocca l'asse y sotto lo zero e che il tasso di crescita è proprio circa 4 piedi per ogni aumento unitario di speed.

La maggior parte delle funzioni di R accettano come input una formula. Ad esempio

```
> gruppi <- c("a", "a" ,"b", "b")
>dati <- c(1, 2, 6, 7)
>boxplot(dati ~ gruppi)
```

produce i due boxplot relativi ai gruppi. Abbiamo già visto anche questa maniera di usare i boxplot.

Una funzione di basso livello molto interessante è par(). Prima di disegnare un grafico, R consulta lo "stato" dell'ambiente grafico. Lo stato grafico del device di output corrente (di solito, la finestra di output, ma può essere anche un file) è definito da una lunga serie di parametri (o setting dello stato grafico del device). Ad esempio, il colore da usare, il font da usare, la grandezza e la posizione delle regioni che conterranno grafici, e così via.

Provate a scrivere

```
> par()
```

per vedere la lista dei setting attuali del vostro ambiente. Fra le altre cose dovrebbe esserci scritto

```
$col.main
[1] "black"
```

il colore di default è il nero. Per evitare di scorrere la lista avremmo potuto dire anche

```
> par("col")
```

che richiede il setting corrente per il colore, oppure

```
par(c("lty", "pch", "col"))
```

per i setting del tipo di linea, di punto, e di colore. Come abbiamo già visto, molti di questi setting possono venire modificati anche dalle funzioni di alto livello. La differenza è che in quel caso la modifica si applica solo a quel comando. Modificando il setting in par invece la modifica si applica a tutto l'output successivo. Ad esempio

```
>par(lty = "dashed")
```

forza come default l'utilizzo di linee tratteggiate in tutti i comandi grafici successivi, a meno che nelle funzioni non sia specificato altrimenti.

Spesso par() viene utilizzato per suddividere la regione di output del grafico in subregioni per organizzare più grafici sulla stessa pagina. Come abbiamo già visto in alcuni esempi , questo viene fatto utilizzando i parametri mfrow o mfcol di par().

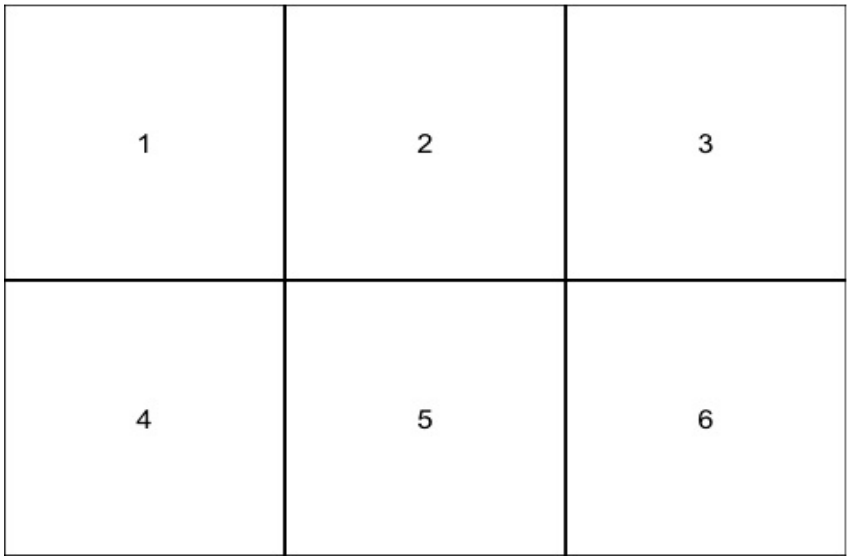
La funzione layout() funziona allo stesso modo di par() ma consente una maggiore flessibilità nella suddivisione.

```
>layout(rbind(c(1,2,3),
              c(4,5,6)))
```

è equivalente a

```
>par(mfcol = c(3,2))
```

>layout.show (6)



è la funzione che consente di visualizzare il layout corrente. Per controllare il layout si utilizza l'opzione heights che definisce l'altezza relativa delle righe (o widths per la larghezza delle colonne).

```
>m <- matrix(c(1, 2))
>layout(m, heights = c(2,1))
```

In questo caso la matrice m ha una colonna e due righe. la prima riga occupa due terzi del totale 2/(1+2), la seconda, un terzo 1/(1+2).

```
> layout.show(2)
```

vi mostra il layout. Vediamo altri esempi.

```
>m <- rbind(c(1, 2),
           c(3, 4))
>layout(m,widths=c(1,2))
>layout.show(4)
```

la prima colonna occupa un terzo dello spazio totale, la seconda due terzi.

```
>m <- rbind(c(1, 2),
           c(0, 0),
           c(3, 4))
>layout(m, widths = c(1, 2))
>layout.show(4)
```

come il precedente ma con una riga vuota in mezzo.

```
>m<-rbind(c(1,3), c(2, 2))
>layout(m, heights = 2,1)
>layout.show(3)
```

in cui la seconda figura occuperà le due zone inferiori. Grazie alla funzione layout è possibile organizzare grafici multipli sulla pagina con grande flessibilità. Tuttavia in generale è necessario preoccuparsi di questi dettagli solo quando si vuole produrre dei grafici complessi per la stampa.

Le funzioni Trellis, infine, consentono di costruire presentazioni con grafici multipli per visualizzare strutture di dati multidimensionali. Vediamo un esempio

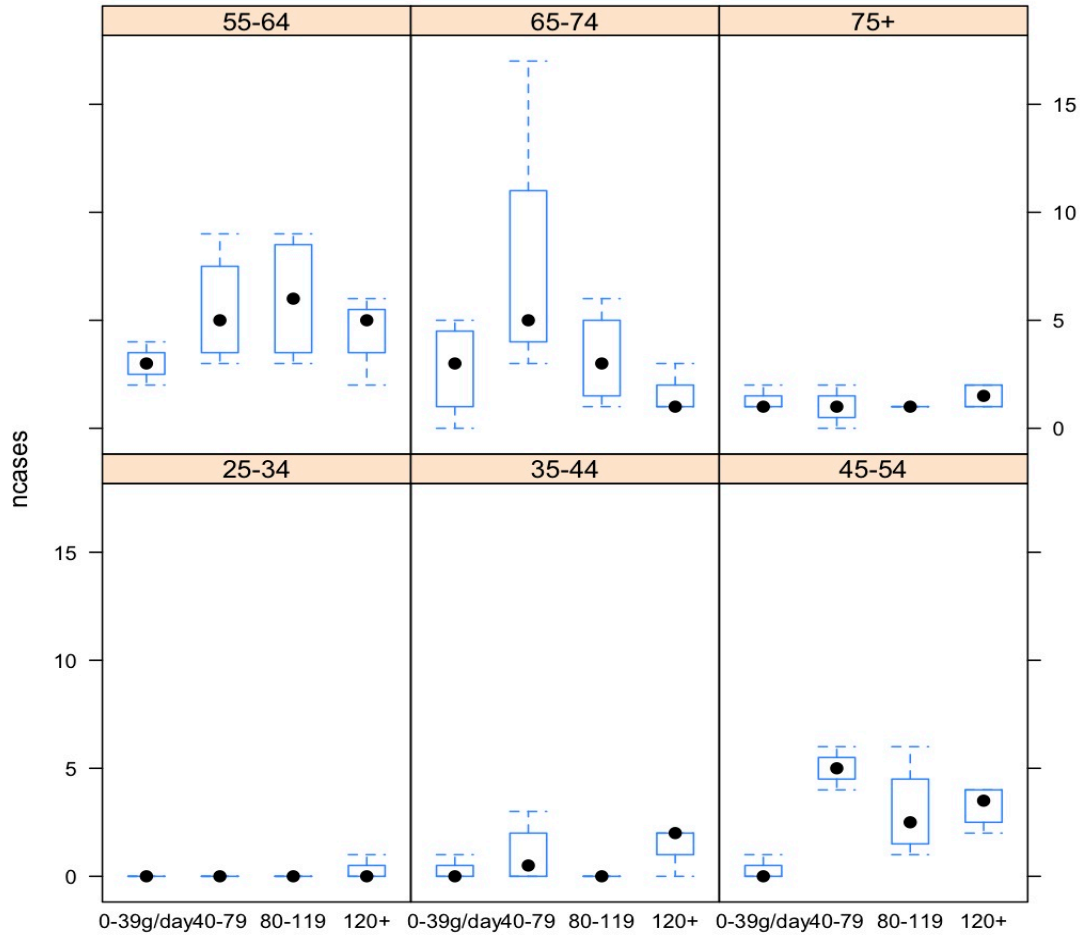
```
>data(esoph)
>attach(esoph)
>names(esoph)

[1] "agegp"     "alcgp"     "tobgp"     "ncases"    "ncontrols"
```

esaminare l'help in linea per i dettagli sulle colonne. Potrebbe essere interessante visualizzare la distrubuzione dei casi di cancro all'esofago in funzione dell'età e del consumo di alcool. Scrivere

```
> library(lattice)
> bwplot(ncases ~ alcgp | agegp, data = esoph)
```

La funzione bwplot (Box-and-Whiskers plot) è l'equivalente Trellis di boxplot. Il simbolo | indica che i boxpot vanno suddivisi in base al gruppo di età. R produce il Trellis alla pagina successiva.

L'esempio illustra tre caratteristiche tipiche delle funzioni Trellis: i) i dati vanno sempre specificati nel formato formula; ii) è possibile utilizzare il simbolo | per"condizionare" i grafici al livello di un'altra variabile; iii) la visualizzazione avviene sempre sotto forma di tabelle, con etichette che identificano il livello della variabile condizionante.



8.4. Le funzioni di basso livello

In questa sezione approfondiamo l'utiizzo delle funzioni grafiche di basso livello in R. Capendo meglio le funzioni di basso livello avrete anche una migliore consapevolezza di come funzionano le funzioni di alto livello. Inoltre potrete avere un controllo completo su ogni dettaglio dei vostri grafici e potrete risolvere molti problemi la cui soluzione non è ovvia. Scrivete

```
>plot.new()
```

R apre la finestra grafica.

```
>plot.window(xlim = c(0,1), ylim = c(0,1))
```

R non disegna nulla ma predispone le dimensioni degli assi che userà.

```
>axis(1)
>axis(2)
>box()
```

traccia gli assi e la cornice che racchiude il grafico. Questo è utile perché qando si creano grafici per la pubblicazione in molti casi il risultato è migliore omettendo la cornice (o perlomeno a me pare così). La cornice viene automaticamente aggiunta quando si usano le funzioni di alto livello come plot(). Tuttavia è possibile cambiare questo setting usando par(). Ad esempio

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

disegna il diagramma di dispersione del vettore y in funzione di x e lo racchiude in un cornice. Settando

```
>par(bty = ”n”)
>plot(x, y)
```

ridisegna il diagramma omettendo la cornice. Settando par() in realtà avete modificato il comportamento della funzione di basso livello box(), utilizzata da plot(). Consultando help(par) è possibile visualizzare le diverse opzioni per bty. La cornice può essere omessa o può avere qualsiasi forma si desideri. Per quanto riguarda infine i limiti degli assi, R di default li ingrandisce del 6% per evitare che simboli disegnati verso le estremità ricadano al di fuori del grafico. Nei casi (rari) in cui sia necessario anche questo può essere modificato usando gli argomenti xaxs e yaxs di plot.window (vedi help in linea).

Per etichettare gli assi si usa

```
> title(main= "Titolo del Grafico", xlab = "Asse x", ylab = "Asse y")
```

l’etichettatura degli assi può essere fatta anche nelle funzioni di alto livello, come abbiamo già visto. In realtà queste controllano il funzionamento di title(). Per controllare font e dimensioni del testo si utilizza par(). Ad esempio scrivendo

```
>par(cex.main = 2)
```

tutti setting successivi di main vengono scritti di dimensioni doppie rispetto al setting corrente di cex in par(). Questo può sembrare un po’ oscuro ma studiando la documentazione e facendo qualche prova dovrebbe diventare chiaro.A proposito del testo associato agli assi, un dettaglio interessante è

che con axis() di default i valori in y vengono scritti paralleli all'asse. Per cambiare questo si può settare las in par() o direttamente in axis(). Ad esempio

```
> axis(2, las = 2)
```

scrive in orizzontale. Settando altri parametri di axis() è possibile controllare anche la posizione dei segmentini divisori (at) e le loro etichette (lab). Ad esempio

```
> axis(1, at = c(-2, 0, 2), lab = c("basso",  "medio", "alto") )
```

mette solo tre divisori sull'asse x e li etichetta come indicato.

La funzione di basso livello per mettere dei punti nel grafico è points(). Ad esempio

```
>x <- rnorm(10)   # 10 valori a caso dalla normale
>y <- rexp(10) # altri 10 dalla esponenziale
>plot.new()
>plot.window(xlim = c(min(x), max(x)),  ylim = c(min(y), max(y)))
>axis(1)
>axis(2)
>points(x, y)
```

Di default i punti sono cerchietti vuoti. Usando pch è possibile controllare il tipo di simbolo utilizzato. I valori da 1 a 25 si riferiscono a questi simboli:

I valori da 33 a 126 si riferiscono a caratteri ASCII, che possono essere specificati anche indicando fra virgolette il carattere desiderato.

Per disegnare un segmento, scrivere

```
>segments(x1, y1, x2, y2)
```

dove x1, y1 sono le coordinate del punto iniziale del segmento, e x2, y2 sono le coordinate del punto finale. Gli argomenti sono vettori, quindi con una sola chiamata a segments() è possibile disegnare n segmenti in base alle coordinate specificate nei vettori che devono avere tutti lunghezza n.

Per disegnare una retta in tutta la finestra grafica, è possibile specificarne i

parametri (intercetta e coefficiente angolare) usando abline(). Ad esempio

```
>a <- 0.5
>b <- 2
>abline(a, b)
```

disegna una retta in cui y = 0.5 quando x = 0 e in cui y cresce di b unità quando x cresce di 1 unità. Per tracciare una retta orizzontale basta specificare il valore di y con

```
>abline(h = posizione.su.y)
```

allo stesso modo per tracciare una retta verticale

```
>abline(v= posizione.su.x)
```

Infine, per disegnare una serie di linee connesse (una "spezzata") è possibile usare lines(). Ad esempio

```
>lines(x, y)
```

traccia una serie di linee a partire dalla prima coppia di valori nei vettori x, y e fino all'ultima coppia.

La funzione arrows() disegna una freccia, usando lo stesso metodo di segments(). Inoltre è possibile specificare le dimensioni della punta della freccia col parametro length; l'angolo formato dai segmenti che formano la punta rispetto all'asta con il parametro angle; e con il parametro code se la punta va disegnata all'inizio (code = 1), alla fine (code = 2) o sia all'inizio sia alla fine (code = 3). Oltre a disegnare una freccia, la funzione arrows() può tornare comoda per disegnare barre di errore. Settando angle = 90 i due segmenti sono ortogonali all'asta e quindi formano una T come nelle barre di errore. Più sotto trovate un esempio.

Per disegnare un poligono di quattro lati paralleli agli assi è possibile usare rect(). Ad esempio

```
>rect(x1, y1, x2,  y2)
```

dove x1, y1 sono le coordinate di uno dei vertici, e x2, y2 sono le coordinate del vertice opposto sulla diagonale.

Per disegnare un poligono generico si usa polygon(), che funziona come lines() ma connette l'ultima coppia di coordinate con la prima. Pertanto

```
>polygon(c(x1, x2, x2, x1), c(y1, y1, y2, y2))
```

è equivalente alla precedente chiamata a rect(). Nel caso dei disegni dipoligoni chiusi, settando col si controlla il colore con cui il poligono viene riempito, mentre per controllare il colore del margine si setta il parametro border. Quindi

```
>polygon(c(x1, x2, x2, x1), c(y1, y1, y2, y2), col = ”green”, border= ”blue”)
```

disegna un rettangolo verde col bordo blu.

Per disegnare una curva si utilizza una sequenza di linee, l’approssimazione è tanto migliore quanto minore è l’intervallo. Ad esempio,

```
>x <- seq(0, 30, length = 1000)
>y <- dchisq(x, 10)
> plot.new()
> plot.window(xlim = range(x), ylim = range(y))
> lines(x,y)
> axis(2)
> axis(1)
```

disegna la funzione di densità chi-quadrato con 10 gradi di libertà. Oppure

```
> x = seq(-3, 3, length = 1000)
> y = dnorm(x)
> plot.new()
> plot.window(xlim = range(x), ylim = range(y))
> lines(x, y)
> axis(2)
> axis(1)
```

disegna la funzione normale standard. La lunghezza del vettore (nell’esempio creato con seq) determina l’ampiezza dell’intervallo: maggiore la lunghezza minore l’intervallo. Nell’esempio il valore 1000 è arbitrario ma dovrebbe andare bene quasi sempre.

Per scrivere stringhe di testo R usa la funzione text(). La funzione ha la sintassi

```
>text(x, y, testo)
```

dove x, y sono le coordinate che specificano dove scrivere, e testo è una

stringa di caratteri. Come sempre in R, x, y e testo sono vettori e quindi la funzione può essere utilizzata per scrivere più stringhe. La funzione è molto flessibile e consente di controllare anche la giustificazione del testo (adj), la sua rotazione (srt) le dimensioni rispetto alle dimensioni standard (cex), il colore con cui scrivere (col) e il font (font). Ad esempio

```
>text(1, 1, "ciao mondo", adj = c(0,0), srt=45, cex = 2, col = "red", font=4)
```

scrive “ciao mondo” rosso in bold-italico, ruotato di 45 gradi e con dimensioni doppie rispetto allo standard, giustificato in basso a sinistra. Se l’esempio non è chiaro consultare l’help in linea o (meglio) fare qualche prova verificando il comportamento di text() al variare dei diversi parametri.

8.5 Colori

Il colore da utilizzare nelle funzioni grafiche è specificato settando il parametro col. Ad esempio,

```
> barplot( rep(1, 3) , col = c("red", "blue", "green3"))
```

disegna tre barre di altezza unitaria, con i tre colori specificati nel vettore corrispondente. I colori possono essere specificati con il nome (fra virgolette) o con un numero. Il numero si riferisce alla posizione nella paletta attualmente definita. Di default, la paletta contiene otto colori:

```
>palette()

[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

quindi, per disegnare tre barre uguali alle precedenti avrei potuto scrivere anche

```
> barplot( rep(1, 3) , col = c(2,  4, 3))
```

se i colori necessari sono più di otto, gli otto colori nella paletta corrente vengono “riciclati” fino a che è necessario. Provate con

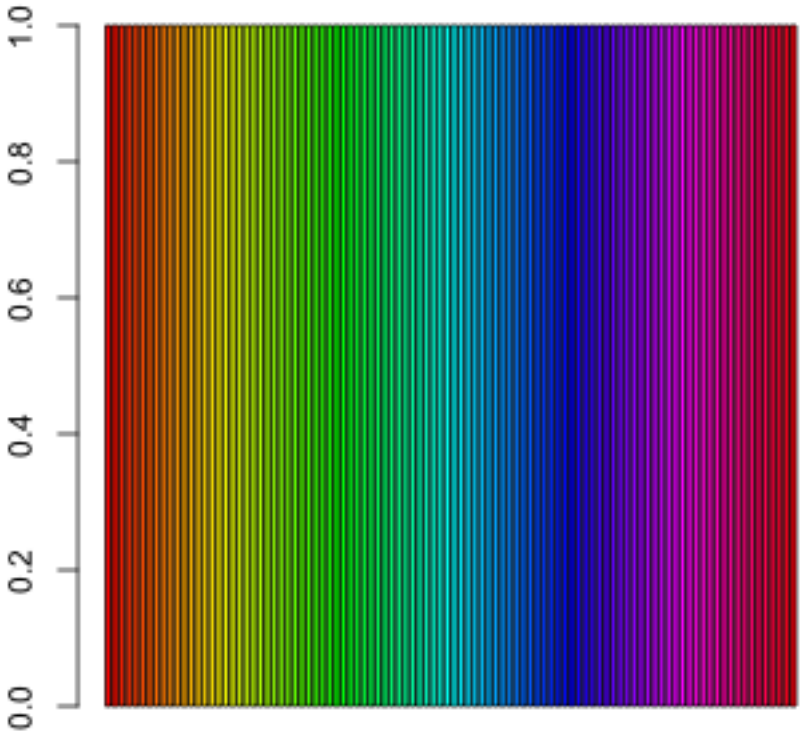
```
> barplot( rep(1,100) , col = palette())
```

I colori contenuti nella paletta possono essere ridefiniti. Ad esempio, utilizzando rainbow():

```
> palette(rainbow(100))
```

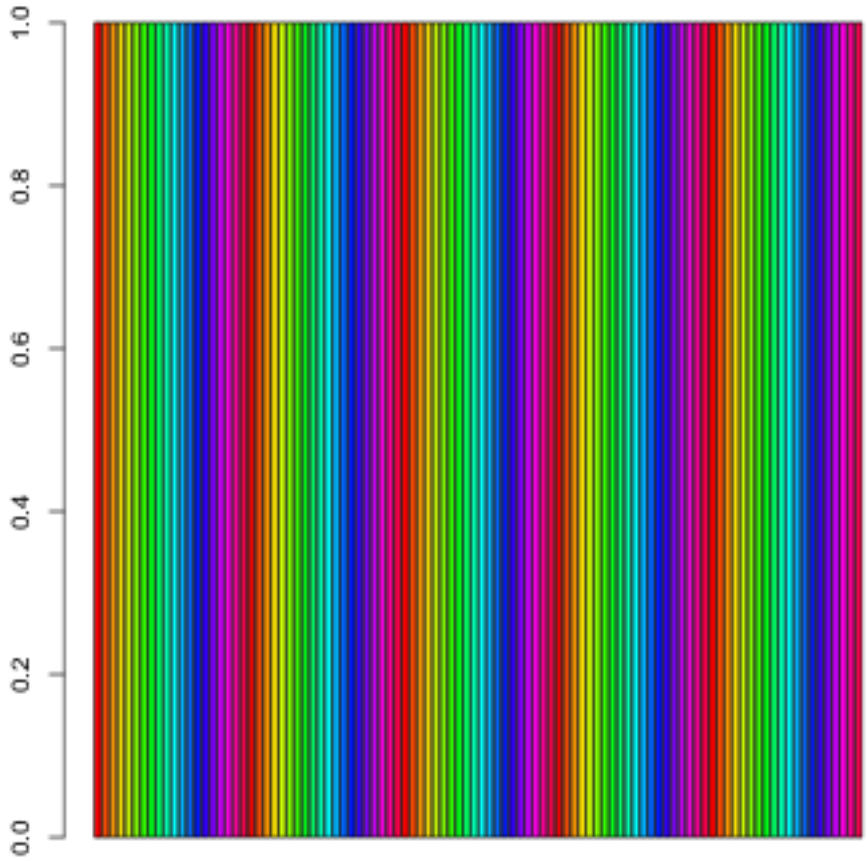
```
> barplot(rep(1,100) , col = palette())
```

disegna 100 barre usando una paletta di 100 colori.



Anche in questo caso se i colori disponibili sono meno delle cose da disegnare, questi vengono “riciclati”:

```
> palette(rainbow(20))
> barplot( rep(1,100) , col = palette())
```



Esattamente lo stesso risultato si ottiene con

> barplot(rep(1,100) , col = 1:100)

Per resettare la paletta ai valori di default scrivere

>palette("default")

per vedere tutti i nomi di colore che R conosce scrivere

>colors()

ridefinendo la paletta è possibile utilizzarli tutti. Ad esempio, provate a confrontare i comandi seguenti con i precedenti:

>palette(colors())

```
> barplot(rep(1, 100) , col = 1:100)
```

Esistono molte altre maniere di usare il colore in R ma in questa dispensa ci fermeremo qui.

8.6 Due problemi interessanti

Concludiamo questa sezione con un esempio di come risolvere due problemi interessanti: i) mostrare nello stesso grafico la tendenza in media dei propri dati e la variabilità delle singole osservazioni; ii) mostrare la tendenza in media assieme all'errore della stima, aggiungendo al grafico delle barre di errore. Creiamo un file di dati:

ss	lun	lp
1	2	1.8
1	2	2.1
1	2	1.9
1	2	2.2
2	2	1.4
2	2	1.7
2	2	1.9
2	2	2.5
3	2	2.2
3	2	1.5
3	2	2.1
3	2	2.3
1	4	3.3
1	4	4.7
1	4	4.1
1	4	3.5
2	4	4.4
2	4	4.4
2	4	3.9
2	4	3.8
3	4	3.3
3	4	2.9
3	4	2.9
3	4	1.2
1	6	6.8
1	6	7.7
1	6	3.9
1	6	6.2
2	6	6.9
2	6	6.7

```
2       6       3.9
2       6       5.8
3       6       6.3
3       6       8.5
3       6       5.9
3       6       4.2
```

come di consueto ricordate che il file deve avere il formato .txt, che ogni caso va separato da un tab, e che alla fine di ogni riga dovete dare invio. (Copia e incolla in un text editor dovrebbe fare tutto il lavoro). Salviamo il file col nome “aggiustamento”.

Leggiamo il file in un dataframe

```
>df<- read.table(“aggiustamento.txt”)
```

ricordate sempre che va specificato il path appropriato per il vostro computer. Controlliamo cosa c’è nel dataframe

```
>summary(df)

 V1       V2          V3
 1 :12   2  :12   3.9    : 3
 2 :12   4  :12   1.4    : 2
 3 :12   6  :12   1.9    : 2
 ss: 1   lun: 1    2.2    : 2
                  2.3    : 2
                  2.9    : 2
                  (Other):24
```

R non ha letto i dati correttamente. Non ha riconosciuto i nomi delle variabili e il sommario non ha nessun senso. Avete capito il motivo?

Esatto, occorre specificare che la prima riga contiene i nomi. Scrivere

```
>df<- read.table(“aggiustamento.txt”, header=TRUE)
```

e quindi

```
>summary(df)

          ss         lun          lp
 Min.    :1    Min.    :2    Min.    :1.400
 1st Qu.:1    1st Qu.:2    1st Qu.:2.200
```

```
 Median :2   Median :4   Median :3.650
 Mean   :2   Mean   :4   Mean   :3.867
 3rd Qu.:3   3rd Qu.:6   3rd Qu.:4.975
 Max.   :3   Max.   :6   Max.   :8.500
```

Adesso R ha letto i dati correttamente. Notate che i nomi delle variabili sono quelli in testa al file, e che per ogni variabile R ha stampato le statistiche riassuntive. Con dei veri dati il file sarebbe accompagnato da informazioni sulla natura delle variabili. Eccole:

Col 1 (ss): id soggetto
Col 2 (lun): lunghezza fisica (cm) del target
Col 3 (lp): lunghezza percepita, ossia aggiustamento di una figura di riferimento (cm) con task= uguagliare la lunghezza del target

I dati catturano, in forma semplificata, la struttura di dati di un esperimento di psicofisica (metodo dell'uguagliamento). Ci interessa rappresentare come la grandezza percepita (gli aggiustamenti) vari in funzione della grandezza fisica. Ci potrebbe anche interessare verificare che la precisione decresce al crescere della grandezza fisica (legge di Weber).

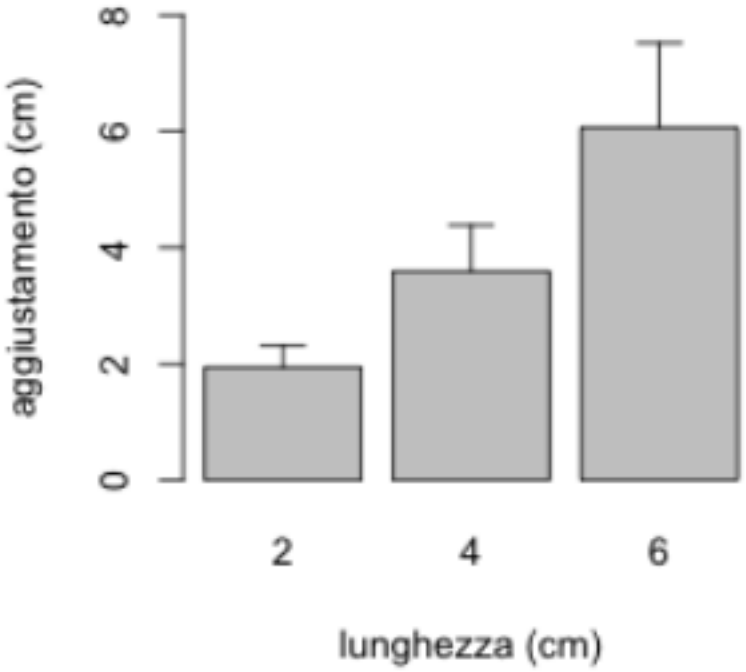
Calcoliamo innnanzi tutto le medie e le deviazioni standard in funzione dei livelli di lunghezza fisica.

```
>attach(df)
> mns <- tapply(lp, lun, mean)
> sds <- tapply(lp,  lun, sd)
```

Per presentare la medie di gruppi di dati spesso vengono usati i diagrammi a barre con dei "baffi" (whiskers), o barre di errore, per rappresentare la variabilità. Vediamo come farlo in R:

```
> barplot(mns, ylim = c(0, 8),
                    xlab = "lunghezza (cm)",
                    ylab = "aggiustamento (cm)")
> arrows(0.7, mns[1],  0.7, mns[1] + sds[1], length=0.1, angle=90)
> arrows(1.9, mns[2], 1.9,  mns[2] + sds[2], length=0.1, angle=90)
> arrows(3.1, mns[3], 3.1,  mns[3] + sds[3], length=0.1, angle=90)
```

R produce la figura alla pagina successiva



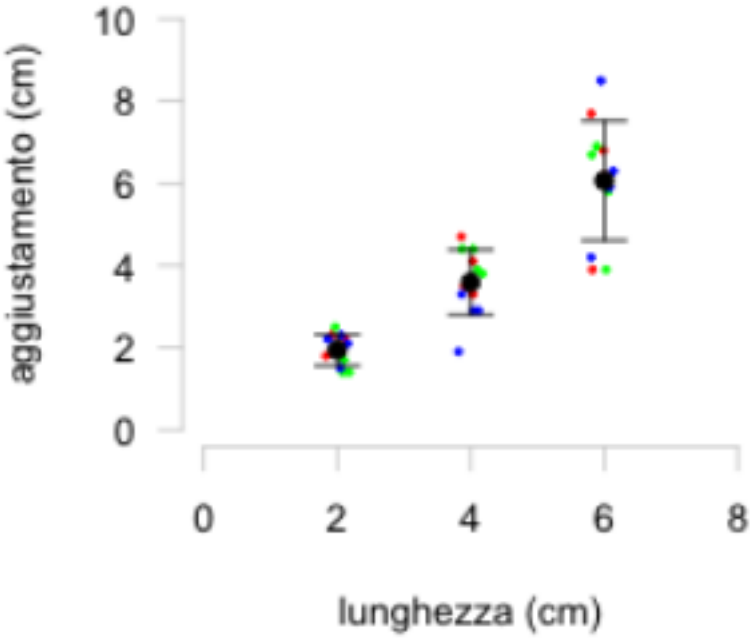
Notare l'uso della funzione arrows per disegnare le barre di errore.

La figura non è male ma a ben guardare solo parzialmente soddisfacente. Abbiamo ridotto tutte le informazioni a due soli numeri (le medie) e le deviazioni standard associate. Si vede, come è giusto attendersi, che gli aggiustamenti rispecchiano in media la grandezza fisica, e la variabilità cresce come previsto dalla legge di Weber. Ma abbiamo perso ogni informazione sulle distribuzioni dei giudizi e sulle differenze fra i due soggetti. Inoltre l'uso delle funzioni di R è poco efficiente. Se avessi un maggior numero di livelli di x dovrei, per ognuno, inserire un comando ad hoc per disegnare la barra di errore. Notate anche che le posizioni x delle barre di errore non sembra avere una giustificazione chiara (infatti le ho trovate per prove ed errori).

Ecco un esempio di programma R che presenta gli stessi dati in maniera più efficace, usando solo funzioni di basso livello:

```
# lettura del file
dtf <- read.table("~/Desktop/R/dati/es1.txt", header=TRUE)
# finestra grafica e assi
plot.new()
```

```
plot.window(xlim = c(0, max(dtf$lun) + 2), ylim = c(0,
max(dtf$lp)+2) )
axis(1, col = "grey")
axis(2, las = 2, col = "grey")
title(xlab = "lunghezza (cm)", ylab = "aggiustamento (cm)")
# definisci i colori da usare a seconda del numero di soggetti
nc <- length(levels(as.factor(dtf$ss)))
palette(rainbow(nc))
# plotta i dati, leggermente sfasati per ridurre sovrapposizioni
points(jitter(dtf$lun, amount=0.1), dtf$lp, col = dtf$ss, pch =
20, cex = 0.6)
# calcola medie e ds
mbyl <- tapply(dtf$lp, dtf$lun, mean)
sdbyl <- tapply(dtf$lp, dtf$lun, sd)
# aggiungi medie e barre di errore
xx <- as.numeric(levels(as.factor(dtf$lun)))
points(xx, mbyl,pch=19,cex=1.5)
arrows(xx, mbyl + sdbyl, xx, mbyl - sdbyl, length = 0.1, angle =
90, code = 3)
```



Il grafico è ora più informativo. Le medie e le deviazioni standard per ogni livello di x sono sovrapposte ai dati grezzi, con i soggetti identificati dal colore. Oltre a rappresentare le tre dimensioni della struttura di dati, il grafico consente di visualizzare le differenze individuali.