Matematica e Statistica con R

Federico Comoglio e Maurizio Rinaldi

13 marzo 2016

Indice

Capitolo 1

Introduzione

Per accedere ai dati richiesti in questa parte occorre caricare il pacchetto allegato `libroR`. Per farlo conviene scaricare il file `libroR_0.0.tgz` sul proprio computer e selezionare il menu `Install`

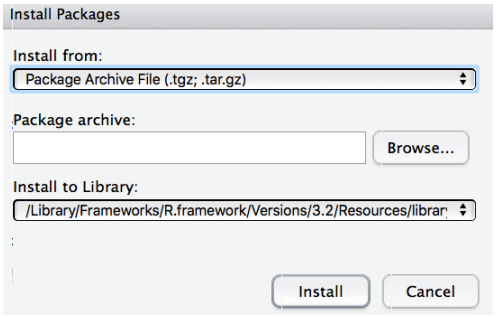

Figura 1.1: Procedura di installazione del pacchetto

Il file viene poi localizzato usando `Browse...`
Alternativamente si può utilizzare direttamente il comando

```
install.packages("libroR_0.0.tgz", repos = NULL, type = .Platform$pkgType)
```

a patto di impostare la *working directory* precisamente dove si trova il file. Il pacchetto va poi successivamente caricato con il comando

```
library("libroR")

##
## Attaching package:  'libroR'
## The following objects are masked from 'package:EsamiR':
##
##    meteo, studenti
```

```
par(mar = rep(3, 4))
for (i in seq(pi/2, -4/3 * pi, length = 12)) {
    plot(0, 0, pch = 20, ann = FALSE, axes = FALSE)
    arrows(0, 0, cos(i), sin(i))
    axis(1, 0, "VI"); axis(2, 0, "IX")
    axis(3, 0, "XII"); axis(4, 0, "III"); box()
}
```

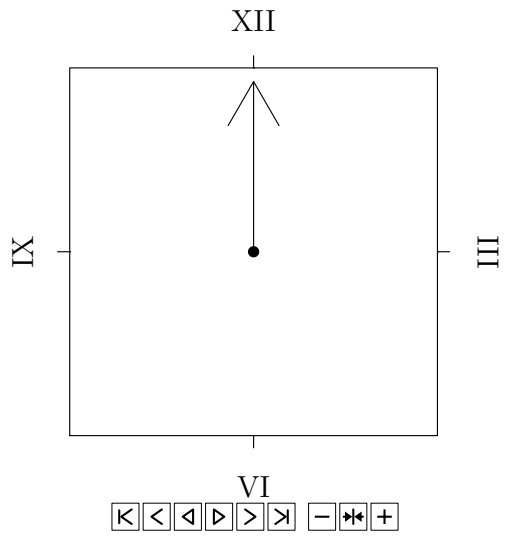

Figura 1.2: A clock animation. You have to view it in Adobe Reader: click to play/pause; there are also buttons to speed up or slow down the animation.

Precisiamo inoltre che questa è una versione assolutamente preliminare.

Capitolo 2

Strutture di dati

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

2.1 I diversi tipi di vettori

2.1.1 Vettori di caratteri/stringhe

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

Operare con le stringhe

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

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

## "sin"

nchar("\"sin\"")

## [1] 5

cat("\\")

## \

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

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

cat("ora spazio\triprendo")

## ora spazio riprendo
```

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

```
"Tab\t"

## [1] "Tab\t"

cat("Tab\t")

## Tab

nchar("Tab\t")

## [1] 4
```

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

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

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

```
class(x)

## [1] "character"

nchar(x)

## [1] 24
```

e anche considerare sottostringhe

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

```
substr(x,3,8)

## [1] "vorare"
```

o abbreviazioni ottenibili con il comando `abbreviate`

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

## Mario Rossi
##      "MrRs"
```

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

```
i=2;toString(i)

## [1] "2"

as.numeric(toString(i))

## [1] 2
```

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

```
letters[1:10]

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

LETTERS[1:10]

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

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

```
month.abb

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

Le stringhe possono poi essere "incollate" con il comando

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

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

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

## a1 a2 a3 a4 a5
```

Il comando può anche essere utilizzato su vettori. Per esempio

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

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

La *recycling rule* continua a valere

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.1.2 Vettori logici

I valori logici in R sono i valori `TRUE` e `FALSE` e corrispondono alla veridicità di un'affermazione. Per esempio

```
1:10 > 4
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## [10]  TRUE
```

Ci fornisce simultaneamente i valori dei 10 confronti.

2.2 Vettori numerici e operazioni di aritmetica modulare

```
1:10

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

```
resto=(1:10)%%4
resto

##  [1] 1 2 3 0 1 2 3 0 1 2

quoziente=(1:10)%/%4
quoziente

##  [1] 0 0 0 1 1 1 1 2 2 2

resto+4*quoziente

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

2.3 Matrici

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

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

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

class(a)

## [1] "matrix"
```

Se i parametri hanno natura diversa vengono resi uniformi

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

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

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

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

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

Se i numeri sono insufficienti vengono *riciclati*

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

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

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

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

Si possono anche definire gli ingressi attraverso opportune funzioni

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

Per assegnare nomi alle righe e alle colonne:

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

```
colnames(a)=c("c1","c2","c3","c4")
rownames(a)=c("r1","r2","r3")
a
```

```
##    c1 c2 c3 c4
## r1  1  4  3  2
## r2  2  1  4  3
## r3  3  2  1  4
```

Aggiungere righe o colonne

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

```
dim(a)

## [1] 3 4

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

##    c1  c2  c3  c4
## r1 "1" "4" "3" "2"
## r2 "2" "1" "4" "3"
## r3 "3" "2" "1" "4"
##    "a" "b" "c" "d"

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

##    c1  c2  c3  c4
## r1 "1" "4" "3" "2" "a"
## r2 "2" "1" "4" "3" "b"
## r3 "3" "2" "1" "4" "c"
```

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

```
colSums(a)

## c1 c2 c3 c4
##  6  7  8  9

rowSums(a)

## r1 r2 r3
## 10 10 10
```

2.3.1 Operazioni con le matrici

Trasposizione

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

```
t(a)

##    r1 r2 r3
## c1  1  2  3
## c2  4  1  2
## c3  3  4  1
## c4  2  3  4
```

Prodotto

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

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

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

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

```
b%*%a

##      c1 c2  c3  c4
## [1,] 26 26  30  38
## [2,] 50 54  62  74
## [3,] 86 96 110 128
```

Determinante

Il determinante di una matrice quadrata si ottiene con il comando

$$\texttt{det}(\textit{matrice}) \tag{2.1}$$

```
det(b)

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

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

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

2.4 I *dataframe*

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

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

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

Per avere una stampa abbreviata

```
head(iris)

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

```
tail(iris)

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

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

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

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

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

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

```
## 9  1  9   B
## 10 1 10   A
```

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

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

```
data(farmacia)
```

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

```
colnames(farmacia)

## [1] "Sex"  "W"    "H"    "Eyes" "Hair" "Sh"   "hM"
## [8] "hF"
```

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

```
attach(farmacia)
```

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

```
Sex

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

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

```
Sex=as.vector(Sex)
Sex

##  [1] "F" "M" "M" "M" "F" "M" "F" "F" "M" "F" "F" "F"
## [13] "F" "F" "M" "F" "F" "F" "M" "M" "M" "F" "F" "F"
## [25] "F" "F" "M" "F" "M" "M" "F" "F" "M" "M" "F" "F"
## [37] "F" "F" "M" "F" "M" "M" "F" "F" "F" "F" "F" "F"
## [49] "F" "M" "M" "M" "F" "F" "M"

class(Sex)

## [1] "character"
```

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

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

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

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

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

```
Sex=factor(Sex)
Sex

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

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

Scrivendo

```
class(biotec)

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

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

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

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

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

```
rownames(studenti)
```

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

Per correggere la strana numerazione possiamo scrivere

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

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

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

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

```
levels(studenti$Hair)

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

Ci incuriosisce il dato con gli occhi neri. Verifichiamo:

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

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

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

```
levels(studenti$Eyes)=c("azzurri","castani","castani", "castani", "castani",
                        "verdi","azzurri","castani","castani","verdi")
```

A questo punto

```
levels(studenti$Eyes)

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

Facciamo lo stesso con i capelli

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

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

Selezione in base a criteri

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

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

```
##    Sex  W    H  Eyes    Hair Sh   hM   hF
## 7    F 63 1.70 verdi castani 38 1.72 1.82
## 17   F 51 1.55 verdi castani 37 1.60 1.70
## 25   F 91 1.81 verdi  biondi 42 1.60 1.87
## 33   M 75 1.82 verdi castani 43 1.60 1.75
## 35   F 46 1.64 verdi castani 37 1.56 1.89
## 37   F 56 1.70 verdi castani 39 1.68 1.90
## 38   F 55 1.65 verdi castani 38 1.68 1.70
## 41   M 56 1.70 verdi castani 39 1.65 1.80
## 42   M 67 1.73 verdi castani 42 1.55 1.85
## 47   F 52 1.75 verdi  biondi 38 1.62 1.80
## 51   M 75 1.76 verdi castani 42 1.60 1.68
## 58   M 64 1.74 verdi castani 41 1.63 1.80
## 60   M 64 1.80 verdi castani 42 1.56 1.75
## 72   F 55 1.67 verdi castani 40 1.60 1.80
## 76   M 85 1.84 verdi castani 43 1.69 1.69
## 77   F 60 1.67 verdi castani 38 1.64 1.70
## 81   F 62 1.61 verdi castani 39 1.60 1.66
## 90   F 49 1.60 verdi castani 40 1.58 1.75
## 91   F 62 1.76 verdi  biondi 40 1.70 1.73
## 93   F 53 1.65 verdi castani 38 1.55 1.85
## 95   M 80 1.80 verdi castani 44 1.68 1.70
```

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

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

```
##       Hair
## 7  castani
## 17 castani
## 25  biondi
## 33 castani
## 35 castani
## 37 castani
## 38 castani
## 41 castani
## 42 castani
## 47  biondi
## 51 castani
## 58 castani
## 60 castani
```

```
## 72 castani
## 76 castani
## 77 castani
## 81 castani
## 90 castani
## 91  biondi
## 93 castani
## 95 castani
```

2.5 Gli *array*

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

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

## , , 1
##
##      [,1] [,2] [,3]
## [1,] "A"  "C"  "E"
## [2,] "B"  "D"  "F"
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,] "G"  "I"  "K"
## [2,] "H"  "J"  "L"
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,] "M"  "O"  "Q"
## [2,] "N"  "P"  "R"
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,] "S"  "U"  "W"
## [2,] "T"  "V"  "X"
```

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

##    K  L  M  N
## A  7  8 80 89
## B 69  3 37 26
## C 71 79 64 10
```

2.6 Le liste

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

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

## $a
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul"
##  [8] "Aug" "Sep" "Oct" "Nov" "Dec"
##
## $b
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
##
## $c
## [1] "your name"
```

Possiamo annidare anche liste entro liste

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

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

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
##
## $c
## [1] "testo"
##
## $d
## $d$g
## [1] "h"
##
## $d$r
##  [1]  1  2  3  4  5  6  7  8  9 10
```