Una breve introduzione a R

Paolo Vidoni
Dipartimento di Scienze Economiche e Statistiche, Università di Udine

Febbraio, 2019

1 Introduzione

1.1 Perché R

Questo documento riprende alcune tematiche sviluppate e presentate in *An Introduction to R* di Peter Haschke, disponibile all'indirizzo web www.peterhaschke.com/files/IntroToR.pdf.

Il software statistico R è un software *open source* che rappresenta uno strumento molto utile e versatile per le analisi statistiche e per la rappresentazione grafica dei risultati. Viene mantenuto e sviluppato dall'*R Core Team* e può essere scaricato gratuitamente dalla pagina web www.cran.r-project.org del CRAN (Comprehensive R Archive Network). R si basa su un linguaggio orientato agli oggetti, è nato come un dialetto del linguaggio S ed è stato sviluppato inizialmente da Ross Ihaka e Robert Gentleman (probabilmente il nome R deriva dalle iniziali dei nomi dei due ricercatori). Dalla pagina web del CRAN e da innumerevoli altri siti si possono ottenere dispense, materiale informativo, librerie aggiuntive, nonché accedere ad archivi, forum e liste di discussione.

R presenta molti vantaggi rispetto ad altri software commerciali per l'analisi di dati. Più precisamente:

- è open source e gratuito;
- può essere utilizzato con vari sistemi operativi;
- è lo strumento che molti ricercatori utilizzano abitualmente per le analisi di dati;
- ha una base di utilizzatori molto ampia, attiva e disponibile a condividere competenze e informazioni;
- viene aggiornato frequentemente e sono disponibili moltissimi pacchetti aggiuntivi;

- ha notevoli capacità per quanto riguarda le analisi e le sintesi grafiche;
- è estremamente flessibile.

R è un linguaggio compilato e quindi tutti i comandi vengono eseguiti immediatamente. È possibile digitare i comandi direttamente sulla console oppure inviare il codice a R utilizzando un *text editor*. Tra i vari text editor, si può ricordare *Notepad*, *Notepad++*, *R Editor* (che è l'editor nativo di R), Emacs e RStudio (ambiente di sviluppo integrato molto versatile, scaricabile gratuitamente da www.rstudio.com).

1.2 Utilizzare R

R si può attivare con un doppio click sull'icona corrispondente o sull'icona di RStudio; quindi risulta visibile la console di R oppure la corrispondente linea di comando. Nel caso si utilizzi RStudio, si attiva la corrispondente console. Alcuni simboli che è necessario richiamare:

- `>` prompt;
- `+` prompt nel caso di comandi non completi;
- `#` per inserire commenti;
- `;` per separare comandi, in alternativa all'interruzione di linea;
- `{}` per raggruppare espressioni o comandi;
- `-> <- =` per assegnare i risultati dell'esecuzione di comandi a oggetti.
- `q()` per uscire da R (in alternativa al menù grafico).

La sequenza dei comandi utilizzati durante una sessione di lavoro può essere salvata in file con estensione `.r`, `.R` oppure `.Rhistory`, chiamato *commands history file*, utilizzando il comando `savehistory("history.r")`. Per richiamare il file si usa il comando `loadhistory("history.r")`. Questa tipologia di file, così come i file script (più precisamente file con estensione `.r` o `.R`), possono venire usati per eseguire più comandi. Con `source("commands.r")`, l'intero file di comandi viene eseguito. Inoltre, la funzione `sink()` invia l'output di R ad un file specifico invece che al terminale di R, come accade usualmente.

Tutti gli oggetti creati durante una sessione costituiscono il così detto *workspace*, che può essere salvato in un file con estensione `.rda` oppure `.Rdata`; il comando `save.image()` salva il workspace attuale, che poi viene automaticamente caricato quando R verrà riattivato. Più precisamente, con il comando `save.image("myfile.Rdata")` il workspace viene salvato nel file `myfile.Rdata` che risiede nella directory di lavoro corrente. La directory di lavoro che R utilizza come default è quella dove R è installato. Per caricare uno specifico workspace si utilizza il comando `load("myfile.Rdata")`; se non viene specificato un percorso alternativo, il file viene cercato nella directory di lavoro corrente.

R non accetta che il percorso che individua la posizione di un file venga definito, ad esempio, con `c:\mydocuments\myfile.txt`, come viene fatto abitualmente in Windows. Infatti R interpreta `\` come un carattere di *escape*, che ha lo scopo di prevedere una interpretazione diversa dal solito per i caratteri che seguono. In alternativa, è necessario utilizzare la specificazione `c:\\mydocuments\\myfile.txt` oppure, in modo più semplice, `c:/mydocuments/myfile.txt`, dove si utilizza `/` invece di `\` come separatore di percorso.

Alcuni utili comandi:

```
getwd()  # stampa la directory di lavoro corrente
setwd("c:/docs/mydirectory")  #  cambia la directory di lavoro in mydirectory
ls()  # stampa la lista di oggetti presenti nella directory di lavoro corrente
rm() # rimuove oggetti presenti nella directory di lavoro corrente
rm(list = ls()) # rimuove tutti gli oggetti presenti nella directory di lavoro
```

1.3 R come calcolatrice

R utilizza i seguenti operatori di base:

- `+` e `-` per addizione e sottrazione;
- $*$ e `/` per moltiplicazione e divisione;
- `^` per l'elevamento a potenza;
- `%%` per l'operatore modulo;
- `%\%` per la divisione intera.

Una selezione delle funzioni matematiche di base utilizzate da R:

- `log()` logaritmo naturale;
- `exp()` funzione esponenziale;
- `sqrt()` radice quadrata;
- `abs()` valore assoluto;
- `sin()` funzione seno;
- `cos()` funzione coseno;
- `tan()` funzione tangente;

- `asin()` funzione arcoseno;
- `factorial()` fattoriale;
- `choose()` coefficiente binomiale;
- `sign()` funzione segno (negativo, nullo o positivo);
- `round()` arrotondamento alla cifra decimale specificata.

Inoltre la funzione `print()` stampa il contenuto dell'oggetto indicato come argomento. Va ricordato che R non tiene conto delle spaziature, a meno che non si considerino stringhe di caratteri, e che R distingue tra lettere minuscole e maiuscole.

Si presentano alcuni semplici esempi.

```
#  # questo è un commento
1+2+3

[1] 6

2+3*4

[1] 14

3/2+1

[1] 2.5

2+(3*4)

[1] 14

(2 + 3) * 4

[1] 20

4*3^3

[1] 108

27^(1/3)

[1] 3

2/0 # il risultato è infinito (positivo)

[1] Inf
```

```
0/0 # il risultato non è un numero, NaN (Not a Number)

[1] NaN

23%%3

[1] 2

23%/%3

[1] 7

sqrt(2)

[1] 1.414214

sin(3.14159)

[1] 2.65359e-06

sin(pi)

[1] 1.224606e-16
```

1.4 Operatori logici

Quando R valuta un'espressione che contiene operatori logici, si ottiene come risultato `TRUE` o `FALSE`. Viene ora riportata una lista con i più comuni operatori logici:

- `<` minore;
- `<=` minore o uguale;
- `>` maggiore;
- `>=` maggiore o uguale;
- `==` uguale;
- `!=` diverso;
- `&` operatore and;
- `|` operatore or;
- `xor` disgiunzione esclusiva.

In aggiunta alla forme corte `&` e `|`, gli operatori and e or presentano anche la forma lunga `&&` e `||`. Nel primo caso, se si considerano oggetti di dimensione superiore a 1, viene effettuata una valutazione elemento per elemento, mentre nel secondo caso si procede da sinistra a destra considerando solo il primo elemento di ogni oggetto. La valutazione continua fino a quando il risultato risulta determinato.

Si presentano alcuni semplici esempi.

```
1 == 1
```

```
[1] TRUE
```

```
1 == 2
```

```
[1] FALSE
```

```
1 != 2
```

```
[1] TRUE
```

```
1 <= 2 & 1 <= 3
```

```
[1] TRUE
```

```
1 == 1 | 1 == 2
```

```
[1] TRUE
```

```
1 > 1 | 1 > 2 & 3 == 3
```

```
[1] FALSE
```

```
1 > 1 & 1 > 2 & 1 > 3
```

```
[1] FALSE
```

```
xor(TRUE, TRUE)
```

```
[1] FALSE
```

```
xor(TRUE, FALSE)
```

```
[1] TRUE
```

1.5 L'help di R

La prima cosa che si può fare, quando ci sono dubbi sulle funzioni di R, è chiedere direttamente a R. Ad esempio, per aprire la pagina dell'help di R riferita alla funzione `lm()` si possono utilizzare i seguenti comandi:

```
?lm
help(lm)
# ??lm ricerca ogni funzione collegata a lm
```

Se il nome della funzione non è noto in modo preciso, si può utilizzare la funzione `apropos()`, che produce una lista con tutte le funzioni di R che contengono il termine cercato.

```
apropos("mean")  # quando il nome della funzione non è noto in modo preciso
```

Una generica pagina dell'help presenta la seguente struttura:

- `Description`: scopo della funzione;
- `Usage`: esempio con una tipica implementazione;
- `Arguments`: lista commentata con gli argomenti della funzione;
- `Details`: informazioni aggiuntive sulla funzione e i suoi argomenti;
- `Value`: informazioni sulla tipologia di risultato che la funzione produce;
- `See Also`: una lista di funzioni collegate;
- `References`: eventuali riferimenti bibliografici;
- `Examples`: esempi di codice.

Infine, con il comando `help.start()` è possibile accedere alla pagina html riferita all'help generale di R, che contiene tra l'altro manuali e documentazione aggiuntiva. Inoltre, come ricordato in precedenza, le risorse a supporto dell'utilizzazione di R che si possono ricavare dal web sono innumerevoli e molto utili.

1.6 Pacchetti e librerie

La versione base di R contiene alcuni utili pacchetti ma ci sono centinaia di pacchetti aggiuntivi che si possono installare. In particolare, si possono considerare i pacchetti disponibili sulla pagina dedicata del CRAN, che risultano certificati dall'R Core Team. Per installare pacchetti dal CRAN si utilizza il comando `install.packages("package")`. Successivamente, viene chiesto di identificare il CRAN *mirror* da cui scaricare il pacchetto (in genere il più vicino o il più veloce). È possibile installare pacchetti anche da file locali.

Dopo aver installato un pacchetto, per poterlo utilizzare, è necessario caricarlo utilizzando il comando `library("package")`. Questa operazione andrà ripetuta ogni qualvolta si inizia una nuova sessione di lavoro. Usualmente i pacchetti vengono modificati con una certa periodicità e quindi vanno aggiornati periodicamente con il comando `update.packages()`.

2 I componenti fondamentali di R

2.1 Oggetti, definizione e individuazione di elementi

R è un linguaggio orientato agli oggetti, quindi qualsiasi cosa in R è un oggetto e ogni operazione corrisponde sostanzialmente alla creazione o alla manipolazione di oggetti. Gli oggetti principali di R sono:

- vettori: sequenze unidimensionali di elementi dello stesso tipo, in particolare, real (double), integer, logical o character;
- matrici e array: oggetti rettangolari a due dimensioni (matrici) o a più dimensioni (array) con elementi dello stesso tipo;
- liste: sequenze di elementi non necessariamente dello stesso tipo; ad esempio, il primo elemento potrebbe essere un vettore con 26 lettere dell'alfabeto, il secondo un vettore con i numeri primi minori di 1000 e il terzo una matrice con 2 righe e 7 colonne;
- data frame: particolari tipologie di liste, che possono essere interpretate come matrici di dati; nelle applicazioni che coinvolgono data set, questi vengono usualmente definiti come data frame dove le righe corrispondono alle osservazioni e le colonne alle variabili;
- fattori: vettori che contengono solo elementi predefiniti e che vengono utilizzati per memorizzare dati categoriali; vengono interpretati in modo diverso da come vengono interpretati i vettori;
- funzioni: oggetti che utilizzano determinati oggetti in input e producono in output un nuovo oggetto.

Gli oggetti possono essere di un determinato *tipo* e appartenere ad una specifica *classe*, inoltre possono contenere informazioni e avere attributi. Ad ogni oggetto possono essere applicati metodi specifici a seconda della classe di appartenenza. Per quanto riguarda i vettori, si possono ricordare i seguenti tipi:

- `integer`: numeri interi (classe `numeric`);
- `double` (`real`): numeri reali (classe `numeric`);
- `complex`: numeri complessi (classe `complex`);
- `character`: stringhe di testo (classe `character`), ad esempio, `"text"`, `"Hello!"`, o `"123"`;
- `logical`: elementi logici, ad esempio `TRUE` e `FALSE` (classe `logical`).

Per visualizzare il contenuto di un oggetto basta stampare il suo contenuto digitando il nome e premendo invio. Per conoscere il tipo e la classe di un oggetto si utilizzano le funzioni `typeof()` e `class()`, rispettivamente. Inoltre, `str()` descrive la struttura di un oggetto (tipo, dimensioni, informazioni sui contenuti) e `attributes()` individua altre caratteristiche, non intrinseche, dell'oggetto. Infine la funzione `is()` evidenzia la classe e la tipologia dell'oggetto. Si ricorda che R manipola e modifica oggetti ma, se il risultato non viene assegnato ad uno specifico oggetto, esso viene sono stampato sullo schermo e non salvato. Vengono ora riportati alcuni esempi.

```
1 + 2 # il risultato viene solo stampato sullo schermo

[1] 3

a <- 1+2 # il risultato viene salvato nell'oggetto a
typeof(a)

[1] "double"

class(a)

[1] "numeric"

is(a)

[1] "numeric" "vector"

str(a)

 num 3

x <- sqrt(2)   #
x   # per stampare il contenuto di x

[1] 1.414214

typeof(x)

[1] "double"

class(x)

[1] "numeric"

is(x)

[1] "numeric" "vector"

str(x)
```

```
 num 1.41
```

```
b <-"hello"
typeof(b)
```

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

```
class(b)
```

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

```
is(b)
```

```
[1] "character"           "vector"              "data.frameRowLabels"
[4] "SuperClassMethod"
```

```
str(b)
```

```
 chr "hello"
```

```
x^3
```

```
[1] 2.828427
```

```
y <- x^3
y
```

```
[1] 2.828427
```

```
x <- pi # un nuovo assegnamento cancella quello precedente
x
```

```
[1] 3.141593
```

```
is(x)
```

```
[1] "numeric" "vector"
```

```
b <-"hello"
typeof(b)
```

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

```
class(b)
```

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

```
is(b)
```

```
[1] "character"           "vector"              "data.frameRowLabels"
[4] "SuperClassMethod"
```

Vengono presentate con maggiore dettaglio le principali strutture che R utilizza per salvare e organizzare dati.

2.2 Vettori

Per creare un vettore si può utilizzare la funzione `c()`, dove `c` è un abbreviazione per *concatenare*, con argomenti, separati da virgola, che definiscono gli elementi del vettore.

```
Vector1 <- c(1,2,3,4,5,6,7,8,9,10) # vettore numerico
Vector1

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

x <- c(2,3,5,7)
x

[1] 2 3 5 7

x <- c(x,11)
x

[1]  2  3  5  7 11
```

Si possono anche creare vettori di caratteri definendo come elementi delle stringhe di caratteri tra virgolette.

```
Vector2 <- c("a","b","c","d") # vettore di caratteri
Vector2

[1] "a" "b" "c" "d"

Vector3 <- c("1","2","3","4")  # vettore di caratteri (i numeri vengono
# interpretati come caratteri)
Vector3

[1] "1" "2" "3" "4"
```

I vettori logici sono invece costituiti da elementi logici.

```
x = c(TRUE, FALSE, TRUE, FALSE)
y = !x
x

[1]  TRUE FALSE  TRUE FALSE

y
```

```
[1] FALSE  TRUE FALSE  TRUE

x & y

[1] FALSE FALSE FALSE FALSE

x | y

[1] TRUE TRUE TRUE TRUE
```

Gli argomenti della funzione `c()` possono essere essi stessi vettori.

```
Vector4 <- c(Vector2 , Vector3 , Vector2 , Vector2 , Vector2)
Vector4

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

In alternativa alla funzione `c()`, si possono utilizzare altri comandi per definire vettori: i due punti `:` per creare vettori formati da sequenze regolari, `seq()` per creare sequenze di diverso tipo e `rep()` per definire vettori con opportune ripetizioni di elementi.

```
xx <- 1:10
xx

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

5:-5

 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5

seq(from=0,to=10)

 [1]  0  1  2  3  4  5  6  7  8  9 10

# si possono omettere i nomi degli argomenti e il passo  della sequenza,
# se non è indicato, corrisponde a 1 (valore di dafault)
seq(0,10)

 [1]  0  1  2  3  4  5  6  7  8  9 10

seq(0,10,by=2) # per definire il passo della sequenza

[1]  0  2  4  6  8 10

seq(0,10,length.out=25) # per definire la lunghezza della sequenza
```

```
 [1]  0.0000000  0.4166667  0.8333333  1.2500000  1.6666667  2.0833333
 [7]  2.5000000  2.9166667  3.3333333  3.7500000  4.1666667  4.5833333
[13]  5.0000000  5.4166667  5.8333333  6.2500000  6.6666667  7.0833333
[19]  7.5000000  7.9166667  8.3333333  8.7500000  9.1666667  9.5833333
[25] 10.0000000

rep(0,time=10) # ripete l'elemento 10 volte

 [1] 0 0 0 0 0 0 0 0 0 0

rep("Hello",3) # ripete l'elemento 3 volte

[1] "Hello" "Hello" "Hello"

rep(Vector1,2) # il vettore viene ripetuto 2 volte

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

rep(Vector2,each=2) # ogni elemento del vettore viene ripetuto 2 volte

[1] "a" "a" "b" "b" "c" "c" "d" "d"
```

2.3 Operazioni con vettori

Molte funzioni matematiche standard possono venire utilizzate con i vettori. Gli operatori di base, gli operatori logici e le funzioni vengono applicate elemento per elemento. Vengono ora richiamate alcune funzioni:

- `sum()` somma gli elementi del vettore;
- `prod()` prodotto degli elementi del vettore;
- `min()` minimo degli elementi del vettore;
- `max()` massimo degli elementi del vettore;
- `mean()` media aritmetica;
- `median()` mediana;
- `range()` campo di variazione (minimo e massimo);
- `var()` varianza (divisione per `n-1`);
- `sd()` deviazione standard;
- `cov()` covarianza (due argomenti, ad esempio `cov(x,y)`);

- `cor()` coefficiente di correlazione (due argomenti, ad esempio `cov(x,y)`);
- `sort()` ordinamento degli elementi (come default `decreasing = FALSE`)
- `order()` indice degli elementi ordinati con ordinamento crescente;
- `length()` lunghezza del vettore;
- `summary()` fornisce opportune sintesi statistiche;
- `which()` fornisce l'indice dell'elemento compatibile con una affermazione logica;
- `which.min()` fornisce l'indice del minimo;
- `which.max()` fornisce l'indice del massimo;
- `unique()` fornisce un vettore senza elementi ripetuti;
- `round()` arrotonda i valori fino alla cifra decimale indicata (il valore di default è 0).

```
x <- 0:10
x+1

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

y <- -5:5
abs(y)

 [1] 5 4 3 2 1 0 1 2 3 4 5

x+y

 [1] -5 -3 -1  1  3  5  7  9 11 13 15

x*y

 [1]  0 -4 -6 -6 -4  0  6 14 24 36 50

y <- 0:8
x+y

Warning in x + y:  longer object length is not a multiple of shorter object length

 [1]  0  2  4  6  8 10 12 14 16  9 11
```

Quando gli oggetti hanno lunghezza diversa, R usa di nuovo gli elementi del vettore più piccolo.

```
x
 [1]  0  1  2  3  4  5  6  7  8  9 10
x > 5
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
x <- 3:26
max(x)
[1] 26
min(x)
[1] 3
sum(x)
[1] 348
prod(x)
[1] 2.016457e+26
x <- c(32,18,25:21,40,17)
x
[1] 32 18 25 24 23 22 21 40 17
sort(x)  # ordine crescente
[1] 17 18 21 22 23 24 25 32 40
order(x) # posizione degli elementi in ordine crescente
[1] 9 2 7 6 5 4 3 1 8
Vector1+Vector1
 [1]  2  4  6  8 10 12 14 16 18 20
Vector1/Vector1
 [1] 1 1 1 1 1 1 1 1 1 1
log(Vector1)
 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595
 [7] 1.9459101 2.0794415 2.1972246 2.3025851
round(log(Vector1))
 [1] 0 1 1 1 2 2 2 2 2 2
round(log(Vector1),digit = 3)
 [1] 0.000 0.693 1.099 1.386 1.609 1.792 1.946 2.079 2.197 2.303
which(Vector1>=5) # indice degli elementi >=5
[1]  5  6  7  8  9 10
```

2.4 Selezione di elementi di un vettore

La selezione di alcuni elementi di un vettore si può definire utilizzando la notazione con le parentesi quadre `[ ]`. Si possono considerare i seguenti esempi.

```
Vector6 <- c("The","Starlab","Fellow","is","a Fool")
Vector6 [3] # si seleziona il terzo elemento

[1] "Fellow"

Vector6[2:4]

[1] "Starlab" "Fellow"  "is"

Vector6[c(1 ,3 ,4)]

[1] "The"    "Fellow" "is"

Vector6[-2] # si selezionano tutti gli elementi ad esclusione del secondo

[1] "The"    "Fellow" "is"     "a Fool"

Vector6[5] <- "great"
Vector6

[1] "The"     "Starlab" "Fellow"  "is"      "great"

xx <- 100:1
xx[7]

[1] 94

xx[c(2,3,5,7,11)]

[1] 99 98 96 94 90

xx[85:91]

[1] 16 15 14 13 12 11 10

xx[91:85]

[1] 10 11 12 13 14 15 16

xx[c(1:5,8:10)]

[1] 100  99  98  97  96  93  92  91

xx[c(1,1,1,1,2,2,2,2)]
```

```
[1] 100 100 100 100  99  99  99  99

yy <- xx[c(1,2,4,8,16,32,64)]
yy

[1] 100  99  97  93  85  69  37

x <- c(32,18,25:21,40,17)
x

[1] 32 18 25 24 23 22 21 40 17

sort(x)

[1] 17 18 21 22 23 24 25 32 40

x[order(x)] # si ottiene lo stesso risultato di sort(x)

[1] 17 18 21 22 23 24 25 32 40

x <- c(1,2,4,8,16,32)
x

[1]  1  2  4  8 16 32

x[-4]

[1]  1  2  4 16 32

x[-c(3,4)]

[1]  1  2 16 32
```

Gli operatori logici possono essere utili per l'individuazione di elementi di un vettore.

```
x <- -8:7
x<0

 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE

x[x<0]

[1] -8 -7 -6 -5 -4 -3 -2 -1

x[x<0&x<(-2)]
```

```
[1] -8 -7 -6 -5 -4 -3

x[x<0|x>5]

 [1] -8 -7 -6 -5 -4 -3 -2 -1  6  7

x[x!=6]

 [1] -8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  7

x[x==6]

[1] 6
```

2.5 Ulteriori funzioni

In un vettore si possono avere anche dei valori mancanti, indicati con la notazione `NA` (Not Available). La funzione `na.omit()` restituisce il vettore senza gli eventuali valori mancanti e aggiunge l'attributo `na.action` che indica la posizione degli `NA` nel vettore originario. Questa funzione è molto utile perché non tutte le funzioni si possono applicare quando ci sono valori mancanti. In alternativa, in alcuni casi, si può specificare l'argomento `na.rm=TRUE`.

```
z<-c(2,3,4,3,NA,NA,6,6,10,11,2,NA,4,3)
max(z) # questa funzione non si può utilizzare in presenza di NA

[1] NA

na.omit(z) # fornisce il vettore senza NA

 [1]  2  3  4  3  6  6 10 11  2  4  3
attr(,"na.action")
[1]  5  6 12
attr(,"class")
[1] "omit"

max(na.omit(z))

[1] 11

max(z,na.rm=TRUE) # l'argomento na.rm=TRUE permette la rimozione degli NA

[1] 11
```

La funzione `is.na()` indica quali sono gli elementi mancanti e, in combinazione con la funzione `subset()`, permette la rimozione manuale degli `NA`.

```
is.na(z)

 [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
[12]  TRUE FALSE FALSE

z.noNA <- subset(z,is.na(z)==FALSE )
z.noNA

 [1]  2  3  4  3  6  6 10 11  2  4  3
```

La funzione `subset()` può venire utilizzata, più in generale, per specificare opportuni sottoinsiemi dell'oggetto di partenza. Ad esempio, con i seguenti comandi, si individuano i numeri tra 1 e 70 che sono divisibili per 7.

```
X <- 1:70
Multiple7 <- subset(X,X%%7==0) # si utilizza l'operatore modulo
Multiple7

 [1]  7 14 21 28 35 42 49 56 63 70
```

2.6 Fattori ordinati e non ordinati

I fattori sono vettori che contengono solo valori predefiniti e vengono utilizzati per memorizzare dati categoriali. Questa specificazione risulta molto utile per le applicazioni statistiche, dal momento che le variabili categoriali vengono analizzate in modo particolare e concorrono alla specificazione dei modelli in modo diverso dalle variabili numeriche. I fattori in R vengono definiti con la funzione `factor()`, che richiede come argomento i valori che poi verranno interpretati come le categorie osservate del fattore (anche ripetute più volte). Si possono considerare sia valori numerici che caratteri, ma i livelli del fattore (cioè i possibili valori che si possono osservare) saranno sempre interpretati come caratteri. Si possono estrarre i livelli di un fattore con la funzione `levels()`. Si possono definire sia fattori ordinati che non ordinati.

Un fattore è quindi un vettore di caratteri che può venire utilizzato, ad esempio, per individuare una classificazione in gruppi degli elementi di un altro vettore della stessa lunghezza. Si consideri, come semplice applicazione, il caso dell'osservazione di 6 soggetti: due ricevono il trattamento `a`, tre il trattamento `b` e uno il trattamento `c`. Queste informazioni si possono memorizzare in un opportuno fattore. Inoltre, per ognuno dei 6 soggetti si osserva una specifica variabile risposta, i cui valori vengono memorizzati in un vettore numerico.

```
treat <- factor(c("a","b","b","c","a","b"))
treat

[1] a b b c a b
Levels: a b c
```

```
levels(treat) # i livelli del fattore treat

[1] "a" "b" "c"

resp <- c(10,3,7,6,4,5)
resp[treat=="a"] # le osservazioni riferite ai soggetti con trattamento "a"

[1] 10  4

sum(resp[treat=="b"])

[1] 15

# la somma delle osservazioni riferite ai soggetti con trattamento "b"
```

I livelli di un fattore vengono memorizzati in ordine alfabetico o nell'ordine che viene eventualmente definito nella fase di specificazione del fattore. In alcuni casi i livelli del fattore seguono un ordine naturale che si vuole mantenere. La funzione `ordered()` permette la creazione di fattori ordinati ed è molto simile alla funzione `factor()`. L'unica differenza è che nel caso ordinato, oltre agli elementi del fattore, vengono stampati anche i livelli con l'ordinamento che è stato definito.

```
treat1 <- ordered(c("a","b","b","c","a","b"), levels=c("c","b","a")) # si ha
# un ordinamento diverso rispetto a quello naturale (alfabetico)
treat1

[1] a b b c a b
Levels: c < b < a

levels(treat1)

[1] "c" "b" "a"
```

È possibile creare un fattore raggruppando valori numerici in opportune classi con la funzione `cut()`.

```
x<-c(1:12,25:38,-3:0,13:24)
x1 <- cut(x,c(-5,5,25,40),labels=c("B","M","A")) # classi (-5,5],(5,25],(25,40]
# che corrispondono ai livelli B, M, A
x1

 [1] B B B B B M M M M M M M M A A A A A A A A A A A A A B B B B M M M
[34] M M M M M M M M M
Levels: B M A

levels(x1)
```

```
[1] "B" "M" "A"

f_x1 <- table(x1) # frequenze assolute per ogni livello (classe)
f_x1

x1
 B  M  A
 9 20 13
```

Utilizzando il comando `tapply()` è possibile applicare una stessa funzione ai valori di un vettore numerico raggrupparti in classi secondo i livelli di un determinato fattore. Si considerino, ad esempio, i seguenti comandi.

```
x<-1:20
y<-factor(rep(0:1,10))
y

 [1] 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Levels: 0 1

tapply(x,y,sum)

  0   1
100 110

# la funzione sum viene applicata ai dati x classificati sulla base degli
# associati livelli osservati del fattore y
```

I fattori si basano su una modalità di memorizzazione dei valori carattere che risulta molto efficiente. Per questo motivo, molte funzioni di R, definite per il caricamento dati, convertono automaticamente un vettore di caratteri in fattore. Per evitare tutto ciò, se non pertinente, è necessario specificare opportunamente gli argomenti della funzione utilizzata.

2.7 Matrici e array

Un array è una collezione a più dimensioni di elementi dello stesso tipo. Si considerano in particolare le matrici, che possono essere interpretate come array a due dimensioni. Per creare una matrice si può usare la funzione `matrix()`, che presenta i seguenti argomenti:

- `data`: un oggetto di R, ad esempio un vettore;
- `nrow`: il numero di righe;
- `ncol`: il numero di colonne;
- `byrow`: un argomento logico per indicare se la matrice viene popolata per colonne (opzione di default) o per righe.

```
Matrix1 <- matrix(data=1,nrow=3,ncol=3) # tutti gli elementi pari a 1
Matrix1

     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

dim(Matrix1) # la dimensione della matrice

[1] 3 3

Vector8 <- 1:12
Vector8

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

Matrix3 <- matrix(data=Vector8,nrow=4)  # come default byrow=FALSE
Matrix3

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

Matrix4 <- matrix(data=Vector8,nrow=4,byrow=TRUE)  # matrice popolata per righe
Matrix4

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

Si possono anche definire matrici accostando vettori per riga, utilizzando la funzione `rbind()`, o accostando vettori per colonna, usando la funzione `cbind()`. Le funzioni `rbind()` e `cbind()` definiscono in modo automatico i nomi delle righe e delle colonne considerando i nomi dei vettori componenti. Utilizzando le funzioni `rownames()` e `colnames()` è possibile modificare i nomi di righe e colonne.

```
Vector9 <- 1:10
Vector9

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

```r
Vector10 <- Vector9^2
Vector10
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

```r
Matrix5 <- rbind(Vector9,Vector10)
Matrix5
```

```
         [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
Vector9     1    2    3    4    5    6    7    8    9    10
Vector10    1    4    9   16   25   36   49   64   81   100
```

```r
Matrix6 <- cbind(Vector9,Vector10,Vector9)
Matrix6
```

```
      Vector9 Vector10 Vector9
 [1,]       1        1       1
 [2,]       2        4       2
 [3,]       3        9       3
 [4,]       4       16       4
 [5,]       5       25       5
 [6,]       6       36       6
 [7,]       7       49       7
 [8,]       8       64       8
 [9,]       9       81       9
[10,]      10      100      10
```

```r
colnames(Matrix6)
```

```
[1] "Vector9"  "Vector10" "Vector9"
```

```r
rownames(Matrix6) # le righe non hanno nome
```

```
NULL
```

```r
colnames(Matrix6) <- c("A","B","C")
rownames(Matrix6) <- c("a","b","c","d","e","f","g","h","i","j")
Matrix6
```

```
   A   B  C
a  1   1  1
b  2   4  2
c  3   9  3
d  4  16  4
e  5  25  5
f  6  36  6
g  7  49  7
h  8  64  8
i  9  81  9
j 10 100 10
```

La funzione `diag()`, in corrispondenza degli argomenti specificati, definisce una matrice diagonale o estrae la diagonale principale di una matrice.

```
Matrix7 <- diag(5) # crea una matrice identica
Matrix7

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

Vector11 <- c(1,2,3,4,5)
Matrix8 <- diag(Vector11) # matrice con Vector11 come diagonale principale
Matrix8

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

diag(Matrix7) # estrae la diagonale della matrice

[1] 1 1 1 1 1
```

Viene ora presentata una lista di operatori per il calcolo matriciale:

- `+ - * /` operazioni standard scalari o elemento per elemento;
- `%*%` moltiplicazione tra matrici;
- `t()` calcolo della matrice trasposta;
- `solve()` calcolo della matrice inversa;
- `det()` calcolo del determinante;
- `chol()` decomposizione di Cholesky;
- `eigen()` calcolo di autovalori e autovettori;
- `crossprod()` prodotto incrociato (dati `x` e `y`, corrisponde a `t(x)%*%y`);
- `%x%` prodotto di Kronecker.

Per quanto riguarda gli array, la funzione che usualmente si utilizza per la loro costruzione è `array()`.

2.8 Selezione di elementi di una matrice

Come per i vettori, la selezione di elementi di una matrice avviene usualmente utilizzando le parentesi quadre `[ ]`. In particolare, con `[i,j]` si ottiene il j-esimo elemento della riga i-esima. Inoltre, per estrarre un vettore riga o un vettore colonna è sufficiente lasciare vuota la specificazione dell'indice di colonna o di riga, rispettivamente.

```
Matrix9 <- matrix(1:9,3)
Matrix9

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

Matrix9[1,1] # primo elemento della prima riga

[1] 1

Matrix9[2,3] # terzo elemento della seconda riga

[1] 8

Matrix9[,1] # prima colonna

[1] 1 2 3

Matrix9[2,] # seconda riga

[1] 2 5 8

Matrix9[1:2, ] # prima e seconda riga

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8

Matrix9[Matrix9[,2]>4,]

     [,1] [,2] [,3]
[1,]    2    5    8
[2,]    3    6    9

# tutte le righe che come secondo elemento hanno un numero maggiore di 4
```

La funzione `apply(x,margine,fun)` fornisce una vettore, un array o una lista di valori ottenuti applicando la funzione `fun` alla dimensione `margine` di una matrice o di un array `x`. Nel caso delle

matrici, `margine=1` corrisponde alle righe (prima dimensione) e `margine=2` corrisponde alle colonne (seconda dimensione).

```
x <- matrix(1:16,ncol=4)
x

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

apply(x,1,sum) # si applica sum alle righe (margine=1)

[1] 28 32 36 40

y<-apply(x,2,prod) # si applica prod alle colonne (margine=2)
y

[1]    24  1680 11880 43680
```

La funzione `outer(x,y,fun)`, se applicata ai vettori numerici `x` e `y`, calcola una matrice i cui elementi sono stati ottenuti applicando la funzione `fun` (la funzione di default è il prodotto $*$) a tutte le combinazioni di elementi di `x` e `y`.

```
x<-1:5
y<-1:5
outer(x,y)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    2    4    6    8   10
[3,]    3    6    9   12   15
[4,]    4    8   12   16   20
[5,]    5   10   15   20   25

# matrice con elementi dati dal prodotto (funzione di default)  di tutte le
# combinazioni di elementi di x e y
outer(x,y,"+")

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

```
# matrice con elementi dati dalla somma di tutte le combinazioni di elementi
# di x e y
```

2.9 Liste

In R una lista è un oggetto costituito da una collezione di oggetti che definiscono le varie componenti della lista stessa. Tali oggetti possono anche essere di tipo diverso. Ad esempio, una lista può essere formata da un vettore numerico, un vettore logico, una matrice, una array di caratteri e una funzione. Per creare una lista si utilizza la funzione `list()`.

Gli oggetti che compongono una lista sono sempre numerati e possono essere richiamati tenendo conto di ciò, utilizzando comandi del tipo `Lst[[1]]`, dove `Lst` è il nome della lista. È possibile, se pertinente, individuare gli elementi dell'oggetto considerato con comandi del tipo `Lst[[4]][1]`. Con il comando `length(Lst)` si ottiene il numero di componenti (di alto livello) della lista. Dal momento che le componenti di una lista possono avere un nome, invece di utilizzare la notazione con le doppie parentesi quadre, è possibile richiamate un oggetto della lista considerando il suo nome `nomecomponente` con il comando `Lst$nomecomponente`.

```
Lst <- list(name="Fred", wife="Mary", no.children=3,child.ages=c(4,7,9))
Lst

$name
[1] "Fred"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9

Lst[[1]] # il primo oggetto della lista

[1] "Fred"

Lst[[4]] # il quarto oggetto della lista

[1] 4 7 9

Lst$child.ages # l'oggetto chiamato child.ages

[1] 4 7 9

Lst$child.ages[2] # il secondo elemento dell'oggetto child.ages

[1] 7
```

2.10 Data frame

Un data frame è una lista di classe `data.frame`, che deve soddisfare ad alcune specifiche restrizioni:

- le componenti possono essere vettori (numerici, carattere o logici), fattori, matrici numeriche, liste o altri data frame;
- matrici, liste e data frame forniscono al data frame tante variabili quante sono le loro colonne, elementi e variabili, rispettivamente;
- vettori numerici e fattori sono inclusi come elementi (variabili) e, come default, i vettori di caratteri sono trasformati in fattori i cui livelli corrispondono ai valori presenti nel vettore;
- i vettori presenti nel data frame devono avere la stessa lunghezza e le matrici lo stesso numero di righe.

Quindi un data frame si può interpretare come una lista di vettori con nome (chiamate colonne) tutti della stessa lunghezza (come per le matrici), interpretabile come una matrice di dati o database. Ogni colonna ha la stessa lunghezza e contiene dati dello stesso tipo, mentre le varie colonne possono anche essere di tipo diverso (come per le liste).

Usualmente i data set vengono strutturati in R nella forma di data frame. Possono quindi venire interpretati come matrici di dati dove le righe corrispondono alle unità statistiche e le colonne alle variabili di interesse, che possono essere anche di tipo diverso. Le righe, le colonne e gli elementi di un data frame possono essere estratti considerando sia le procedure introdotte per le matrici che le procedure introdotte per le liste. Il modo più semplice per caricare un data set nella sessione di lavoro corrente è utilizzare la funzione `data()`. Se non viene specificato l'argomento, vengono caricati tutti i data set presenti in R. Per salvare una copia del data set si utilizza la funzione `save()`.

Si consideri il data set `airquality`, disponibile in R, che contiene misure giornaliere sulla qualità dell'aria effettuate a New York da maggio a settembre 1973. Il data set `airquality` è un data frame con 154 osservazioni su 6 variabili.

```
data(airquality) # viene caricato il data set
dim(airquality) # dimensione del data frame: 154 osservazioni su 6 variabili

[1] 153   6

help(airquality)  #  descrizione del data frame

starting httpd help server ...  done

names(airquality)  # nomi delle variabili

[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
```

```
head(airquality)

  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6

# fornisce la prime 6 righe del data frame, in alternativa si può
# usare airquality[1:6,]
```

La funzione `str()` fornisce un riassunto della struttura dell'oggetto indicato come argomento, in questo caso del data frame in esame.

```
str(airquality) # struttura del data frame airquality

'data.frame': 153 obs. of  6 variables:
 $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Come ricordato in precedenza, per estrarre elementi da un data frame si possono utilizzare sia le procedure definite per le matrici, riferite all'uso delle parentesi quadre `[ ]`, sia le procedure utilizzate per le liste e basate sulla notazione `$`. È possibile anche combinare entrambi gli approcci.

```
airquality$Ozone

  [1]  41  36  12  18  NA  28  23  19   8  NA   7  16  11  14  18  14
 [17]  34   6  30  11   1  11   4  32  NA  NA  NA  23  45 115  37  NA
 [33]  NA  NA  NA  NA  NA  29  NA  71  39  NA  NA  23  NA  NA  21  37
 [49]  20  12  13  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 135  49  32
 [65]  NA  64  40  77  97  97  85  NA  10  27  NA   7  48  35  61  79
 [81]  63  16  NA  NA  80 108  20  52  82  50  64  59  39   9  16  78
 [97]  35  66 122  89 110  NA  NA  44  28  65  NA  22  59  23  31  44
[113]  21   9  NA  45 168  73  NA  76 118  84  85  96  78  73  91  47
[129]  32  20  23  21  24  44  21  28   9  13  46  18  13  24  16  13
[145]  23  36   7  14  30  NA  14  18  20

airquality$Ozone[1:5]

[1] 41 36 12 18 NA
```

Dal momento che utilizzare la notazione `$` può, in alcuni casi, non essere conveniente, è possible svolgere operazioni su un data frame in modo più semplice utilizzando la funzione `with()`. Tale funzione permette l'esecuzione di opportune espressioni con riferimento ad un *ambiente* riferito al data frame in esame.

```
airquality$Ozone + airquality$Wind/airquality$Temp
```

```
  [1]  41.110448  36.111111  12.170270  18.185484         NA
  [6]  28.225758  23.132308  19.233898   8.329508         NA
 [11]   7.093243  16.140580  11.139394  14.160294  18.227586
 [16]  14.179688  34.181818   6.322807  30.169118  11.156452
 [21]   1.164407  11.227397   4.159016  32.196721         NA
 [26]         NA         NA  23.179104  45.183951 115.072152
 [31]  37.097368         NA         NA         NA         NA
 [36]         NA         NA  29.118293         NA  71.153333
 [41]  39.132184         NA         NA  23.097561         NA
 [46]         NA  21.193506  37.287500  20.141538  12.157534
 [51]  13.135526         NA         NA         NA         NA
 [56]         NA         NA         NA         NA         NA
 [61]         NA 135.048810  49.108235  32.113580         NA
 [66]  64.055422  40.131325  77.057955  97.068478  97.061957
 [71]  85.083146         NA  10.195890  27.183951         NA
 [76]   7.178750  48.085185  35.125610  61.075000  79.058621
 [81]  63.135294  16.093243         NA         NA  80.100000
 [86] 108.094118  20.104878  52.139535  82.084091  50.086047
 [91]  64.089157  59.113580  39.085185   9.170370  16.090244
 [96]  78.080233  35.087059  66.052874 122.044944  89.114444
[101] 110.088889         NA         NA  44.133721  28.140244
[106]  65.121250         NA  22.133766  59.079747  23.097368
[111]  31.139744  44.132051  21.201299   9.198611         NA
[116]  45.122785 168.041975  73.093023         NA  76.100000
[121] 118.024468  84.065625  85.067021  96.075824  78.055435
[126]  73.030108  91.049462  47.085057  32.184524  20.136250
[131]  23.132051  21.145333  24.132877  44.183951  21.203947
[136]  28.081818   9.153521  13.161972  46.088462  18.205970
[141]  13.135526  24.151471  16.097561  13.196875  23.129577
[146]  36.127160   7.149275  14.263492  30.098571         NA
[151]  14.190667  18.105263  20.169118
```

```
with(airquality, Ozone + Wind/Temp)  # stesso risultato
```

```
  [1]  41.110448  36.111111  12.170270  18.185484         NA
  [6]  28.225758  23.132308  19.233898   8.329508         NA
 [11]   7.093243  16.140580  11.139394  14.160294  18.227586
 [16]  14.179688  34.181818   6.322807  30.169118  11.156452
```

```
 [21]   1.164407  11.227397   4.159016  32.196721         NA
 [26]         NA         NA  23.179104  45.183951 115.072152
 [31]  37.097368         NA         NA         NA         NA
 [36]         NA         NA  29.118293         NA  71.153333
 [41]  39.132184         NA         NA  23.097561         NA
 [46]         NA  21.193506  37.287500  20.141538  12.157534
 [51]  13.135526         NA         NA         NA         NA
 [56]         NA         NA         NA         NA         NA
 [61]         NA 135.048810  49.108235  32.113580         NA
 [66]  64.055422  40.131325  77.057955  97.068478  97.061957
 [71]  85.083146         NA  10.195890  27.183951         NA
 [76]   7.178750  48.085185  35.125610  61.075000  79.058621
 [81]  63.135294  16.093243         NA         NA  80.100000
 [86] 108.094118  20.104878  52.139535  82.084091  50.086047
 [91]  64.089157  59.113580  39.085185   9.170370  16.090244
 [96]  78.080233  35.087059  66.052874 122.044944  89.114444
[101] 110.088889         NA         NA  44.133721  28.140244
[106]  65.121250         NA  22.133766  59.079747  23.097368
[111]  31.139744  44.132051  21.201299   9.198611         NA
[116]  45.122785 168.041975  73.093023         NA  76.100000
[121] 118.024468  84.065625  85.067021  96.075824  78.055435
[126]  73.030108  91.049462  47.085057  32.184524  20.136250
[131]  23.132051  21.145333  24.132877  44.183951  21.203947
[136]  28.081818   9.153521  13.161972  46.088462  18.205970
[141]  13.135526  24.151471  16.097561  13.196875  23.129577
[146]  36.127160   7.149275  14.263492  30.098571         NA
[151]  14.190667  18.105263  20.169118
```

Un'altra funzione utile per estrarre elementi da un data frame è la funzione `subset()`.

```
subset(airquality, airquality$Month == 5 & airquality$Solar.R >= 150)
```

```
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
4     18     313 11.5   62     5   4
7     23     299  8.6   65     5   7
10    NA     194  8.6   69     5  10
12    16     256  9.7   69     5  12
13    11     290  9.2   66     5  13
14    14     274 10.9   68     5  14
16    14     334 11.5   64     5  16
17    34     307 12.0   66     5  17
19    30     322 11.5   68     5  19
22    11     320 16.6   73     5  22
```

```
26    NA     266 14.9   58     5  26
29    45     252 14.9   81     5  29
30   115     223  5.7   79     5  30
31    37     279  7.4   76     5  31

# si estrae un data set che contiene le misurazioni di maggio con Solar.R >= 150
```

Si può creare un data frame con la funzione `data.frame()`.

```
x<-1:5
y<-factor(c("a","b","a","a","b"))
z<-matrix(rep(7,15),nrow=5,byrow=F)
es.df<-data.frame(z,uno=x,due=y)
es.df

  X1 X2 X3 uno due
1  7  7  7   1   a
2  7  7  7   2   b
3  7  7  7   3   a
4  7  7  7   4   a
5  7  7  7   5   b
```

Un matrice può venire trasformata in una data frame con la funzione `as.data.frame()`.

```
z<-matrix(rep(7,15),nrow=5,byrow=F) # matrice
z

     [,1] [,2] [,3]
[1,]    7    7    7
[2,]    7    7    7
[3,]    7    7    7
[4,]    7    7    7
[5,]    7    7    7

class(z)

[1] "matrix"

z_df<- as.data.frame(z) # data frame
z_df

  V1 V2 V3
1  7  7  7
2  7  7  7
3  7  7  7
4  7  7  7
5  7  7  7
```

```
class(z_df)

[1] "data.frame"
```

2.11 Importare e salvare data set

Succede molto spesso di dover caricare in R data set creati con altri programmi (come ad esempio Excel, Stata, SPSS) oppure semplicemente data set salvati come file di testo. Per caricare un data frame si utilizza abitualmente la funzione `read.table()`; in alternativa si può usare la funzione `scan()`, che però risulta più primitiva, avendo una funzionalità limitata.

Per poter leggere e importare direttamente un data frame, questo deve presentarsi in una forma specifica: la prima riga deve contenere eventualmente i nomi delle variabili e le righe successive devono avere come primo elemento l'etichetta che definisce la riga e, successivamente, i valori riferiti alle diverse variabili. Come default, nella procedura di caricamento, gli oggetti numerici vengono interpretati come variabili numeriche mentre oggetti non numerici vengono interpretati come fattori. In particolare, R converte automaticamente i vettori di caratteri in fattori. Per evitare questa modificazione è possibile utilizzare l'opzione `stringsAsFactors=FALSE` nella specificazione di `read.table()`. In alternativa, è possibile riconvertire in vettore di caratteri il fattore caricato da R.

La funzione `read.table()` è una funzione generale che si utilizza per leggere e importare data frame. Permette, tra l'altro, la specificazione del simbolo utilizzato come delimitatore (argomento `sep=`) e fornisce l'opzione logica per includere o meno l'intestazione del data frame (argomento `header=`). Più precisamente, con l'opzione `header=TRUE` si definisce che la prima riga è la riga con l'intestazione, e quindi non viene specificata la corrispondente etichetta di riga. Se `sep = " "` (che è il valore di default in R), il separatore è lo *spazio bianco*, invece `sep = ","` definisce la virgola come separatore, come accade con dati in formato `.csv`, ad esempio ottenuti utilizzando Excel. La funzione `read.csv()` è una variante di `read.table()` che, come default, adotta il formato `.csv`.

```
dat <- read.table("file.txt", sep=" ", header=T)
dat <- read.table("file.cvs", sep=",", header=T)
```

È possibile caricare data set creati con altri programmi, come Excel (utilizzando, ad esempio, la funzione `read_xlsx()` del pacchetto `readxl`), SPSS (utilizzando la funzione `read.spss()` del pacchetto `foreign`), STATA (utilizzando la funzione `read.dta()` del pacchetto `foreign`) oppure caricare dati direttamente da pagine web o mediante procedure di *web scraping*. Inoltre, quando il data set presenta colonne di uguale ampiezza (come numero di caratteri) separate da spazi, si può utilizzare la funzione `read.table()` con l'opzione `strip.white=TRUE`, per rimuovere gli spazi in eccesso. Infine, se il data set presenta colonne con spazi all'interno o se non ci sono spazi tra le varie colonne, si considera la funzione `read.fwf()`, che permette la specificazione della larghezza delle colonne misurata in numero di caratteri.

Si prende in esame il data frame `airquality`, che viene ora modificato escludendo le righe che presentano un `NA` in corrispondenza della variabile `Ozone`. Il nuovo data frame viene poi salvato in un file di testo, nella directory di lavoro corrente, utilizzando la funzione `write.table()` (oppure la funzione `write.csv()`, se si desidera adottare come default il formato `.csv`).

```
airq<-airquality[!is.na(airquality$Ozone),]
write.table(airq,file="airq.txt",sep=" ",row.names=F)
write.table(airq,file="airq.csv",sep=",",row.names=F) # formato .csv
```

In `write.table()` si è utilizzata l'opzione `row.names=FALSE` di modo che non vengano memorizzate le etichette delle righe nei file `airq.txt` e `airq.csv`. In questo modo, quando questi nuovi file verranno caricati, non rimarrà traccia del fatto che alcune righe del data set originario `airquality` sono state cancellate.

```
airq1<-read.table("airq.txt",header=T,sep=" ")
airq2<-read.table("airq.csv",header=T,sep=",")
```

Come evidenziato in precedenza, l'uso della notazione `$` per selezionare le colonne di un data frame può non essere conveniente. In questa prospettiva può essere utile considerare la funzione `attach()`, che presenta come argomento la lista o il data frame di interesse. In questo modo, le componenti del data frame o della lista sono visibili con il loro nome nell'ambiente di lavoro, senza la necessità di richiamare in modo esplicito il nome del data frame o della lista. È possibile ritornare alla configurazione originaria con la funzione `detach()`.

```
attach(airquality)
Ozone[1:3]

[1] 41 36 12

detach(airquality)
Ozone[1:3]

Error in eval(expr, envir, enclos):  oggetto "Ozone" non trovato
```

2.12 Ulteriori funzioni utili

Può essere utile capire se un oggetto presenta determinate caratteristiche, riferite ad esempio al tipo, alla classe o alla struttura. A tale scopo di possono utilizzare le seguenti funzioni, dal significato intuitivo, che ritornano come risultato un valore logico:

- `is.numeric()`
- `is.vector()`

- `is.factor()`
- `is.matrix()`
- `is.data.frame()`
- `is.character()`

Le seguenti funzioni invece sono utili per trasformare oggetti di R:

- `as.numeric()` trasforma vettori e matrici di altre classi nella classe `numeric`;
- `as.character()` trasforma vettori e matrici di altre classi/di altro tipo in `character`;
- `as.integer()` trasforma vettori e matrici di altro tipo in `integer`;
- `as.factor()` trasforma vettori e matrici in fattori;
- `as.matrix()` trasforma un vettore o un data frame in una matrice;
- `as.vector()` trasforma una matrice in un vettore;
- `as.data.frame()` trasforma vettori e matrici in data frame;
- `as.list()` trasforma vettori e matrici in liste.

Per integrare in R i principali sistemi per la gestione di database, si possono utilizzare le seguenti interfacce, che permetto l'accesso a varie tipologie di database:

- `RODBC`
- `RMySQL`
- `ROracle`
- `RJDBC`

3 Funzioni per costruire grafici

R dispone di molte funzioni che permettono la definizione di efficaci rappresentazioni grafiche, sia nella versione base che con riferimento alle librerie aggiuntive. Le funzioni utili per costruire grafici si possono dividere nei seguenti tre gruppi fondamentali:

- funzioni grafiche di alto livello, che permettono la creazione di nuovi grafici con assi, etichette, titoli, ecc.; tra le varie funzioni, si possono ricordare in particolare `plot()`, `curve()`, `pie()`, `hist()`, `pairs()`, `boxplot()`, `barplot()`;
- funzioni grafiche di basso livello, che permettono di aggiungere ad un grafico esistente ulteriori elementi, come ad esempio, punti aggiuntivi, linee, curve, etichette, ecc.; tra le varie funzioni, si possono ricordare in particolare `points()`, `lines()`, `abline()`, `title()`;

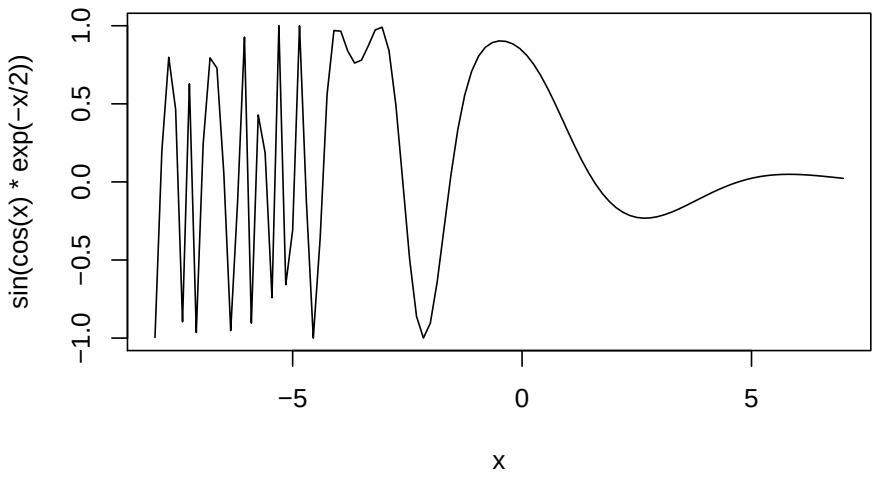
- funzioni grafiche interattive, che permettono di aggiungere o estrarre informazioni da un grafico utilizzando, ad esempio, il mouse.

Una delle funzioni grafiche più utilizzate è `plot()`, che è una funzione generale dal momento che il grafico prodotto dipende dal numero, dal tipo e dalla classe degli oggetti utilizzati come argomenti. Ad esempio,

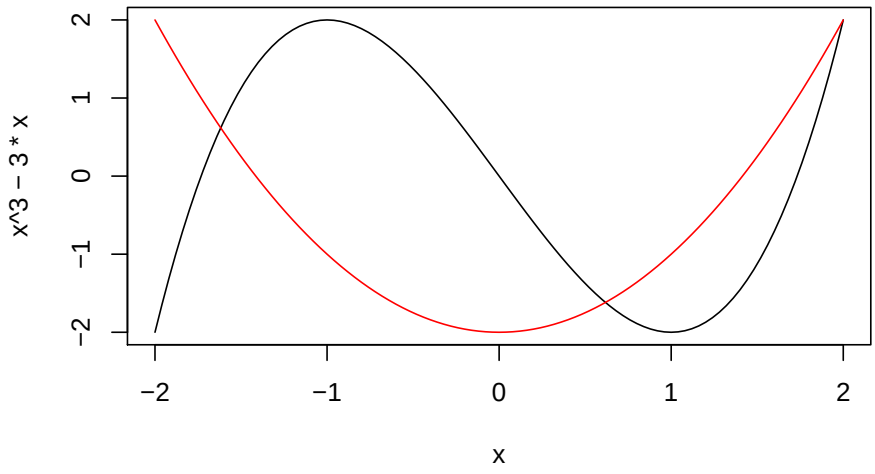
- `plot(x,y)`, con `x` e `y` vettori numerici, produce il diagramma di dispersione (*scatterplot*) di `y` rispetto a `x`;
- `plot(x)`, con `x` una serie temporale (dati indicizzati dal tempo), produce il grafico con l'evoluzione temporale delle osservazioni; se invece `x` è un vettore numerico, produce il grafico dei valori del vettore rispetto al loro indice di posizione.

Inoltre, la funzione `curve()` disegna una curva con riferimento all'intervallo di valori `[from,to]`.

```
curve(sin(cos(x)*exp(-x/2)),from=-8,to=7)
```




```
# funzione grafica di alto livello
curve(x^3-3*x,-2,2) # opzioni from=-2 e to=2
# funzione grafica di basso livello, poiché con l'opzione add=T si aggiunge una
# curva al grafico esistente
curve(x^2-2,add=T,col="red") # il colore selezionato è il rosso
```




```
# lo stesso risultato si ottiene con
# x <- seq(-2,2,0.01)
# plot(x,x^3-3*x,type='l') # opzione type='l' per rappresentare una linea
## continua invece dei punti
# lines(x,x^2-2,col="red") # il colore selezionato è il rosso
```

Infine, la funzione `persp()` permette la rappresentazione grafica di superfici, mentre la funzione `contour()` crea grafici con linee di livello oppure aggiunge linee di livello ad un grafico esistente.

4 Definire funzioni

R prevede la possibilità di creare nuove funzioni, che sono oggetti di classe `function`. Le funzioni già presenti in R, come ad esempio `mean()`, `var()` ecc., sono anch'esse definite in linguaggio R e non differiscono, nella sostanza, da quelle create dall'utilizzatore.

4.1 Funzioni

Una funzione si definisce con un assegnamento che ha la seguente forma:

```
nome<-function(arg1,arg2,...) {espressione1;espressione2;...;return(espressione)}
```

Con `espressione` si indica un'espressione di R o un gruppo di espressioni che utilizza gli argomenti `arg` (in genere più di uno) per ottenere un risultato finale. Tale valore finale corrisponde a quello prodotto dall'espressione che viene specificata come argomento di `return()`; in alternativa, se manca il comando `return(espressione)`, il risultato corrisponde a quello prodotto dall'ultima espressione. Quando è presente una sola espressione, le parentesi graffe non sono necessarie. Un funzione viene chiamata, quando ciò risulta legittimo, utilizzando un comando della forma

`nome(espressioneA,espressioneB,...)`. Gli assegnamenti ordinari definiti nel corpo della funzione sono locali e temporanei e quindi vengono cancellati dopo l'esecuzione della funzione. L'oggetto `NULL` corrisponde all'oggetto nullo e costituisce un nome riservato in R; viene spesso ottenuto quando espressioni o funzioni non producono alcun valore o producono un valore non definito.

Viene ora definita la funzione `BMI` (Body Mass Index) che permette di calcolare l'indice di massa corporea considerando specifici valori per il peso (in chilogrammi) e per l'altezza (in metri).

```
# Indice di massa corporea sulla base di peso (chilogrammi) e altezza (metri)
bmi<-function(weight,height)
  {
        x<- weight/height^2
        return(x)
  }

bmi(75,1.7)

[1] 25.95156

x<-c(60,65,75,80,90)
y<-c(1.5,1.6,1.7,1.8,1.9)
bmi(x,y)

[1] 26.66667 25.39062 25.95156 24.69136 24.93075

bmi(1.7,y)

[1] 0.7555556 0.6640625 0.5882353 0.5246914 0.4709141
```

4.2 Espressioni di controllo condizionali

Nel linguaggio R è possibile specificare espressioni condizionali, chiamate `if statement`, in una delle seguenti forme:

```
if (espressione1) espressione2
if (espressione1) espressione2 else espressione3
```

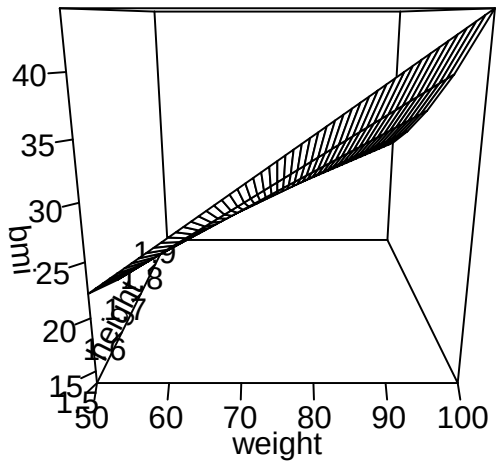
In questo caso l'espressione `espressione1` viene valutata e deve produrre un valore logico, che determina l'esecuzione dell'equazione o di una delle equazioni specificate di seguito. Gli operatori `&` e `|`, considerati eventualmente nella forma lunga `&&` e `||`, vengono spesso considerati nella parte condizionale dell'espressione.

Viene ora definita un'ulteriore funzione che calcola l'indice di massa corporea, tenendo conto della possibilità che l'altezza venga misurata in centimetri.

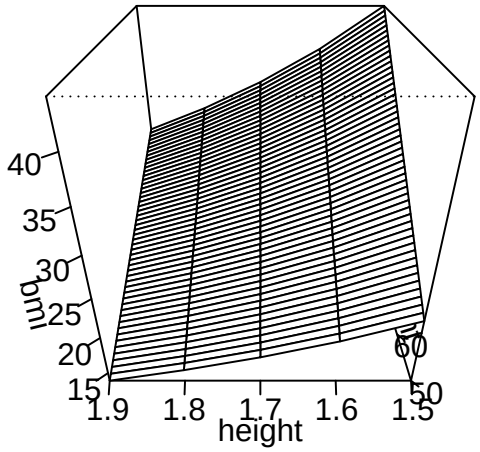
```
# Indice di massa corporea con altezza in metri o centimetri
bmi1<-function(weight,height,cm=FALSE)
  {
        if(cm==TRUE) height<-height/100
        weight/height^2
  }

bmi1(75,1.7)

[1] 25.95156

bmi1(75,170,cm=T)

[1] 25.95156
```

Usando le funzioni `perps()` e `contour()` si ottengono le seguenti rappresentazioni grafiche per i valori dell'indice di massa corporea al variare dei livelli di peso e altezza.

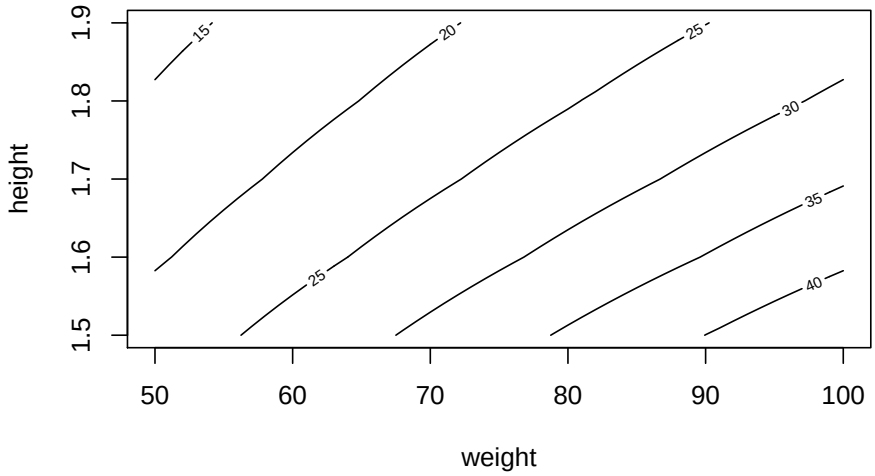
```
x<-seq(50,100,by=1)
y<-seq(1.5,1.95, by=0.1)
index<-outer(x,y,bmi)
# rappresentazione grafica tridimensionale
persp(x,y,index,xlab="weight",ylab="height",zlab="bmi",ticktype ="detailed")
```



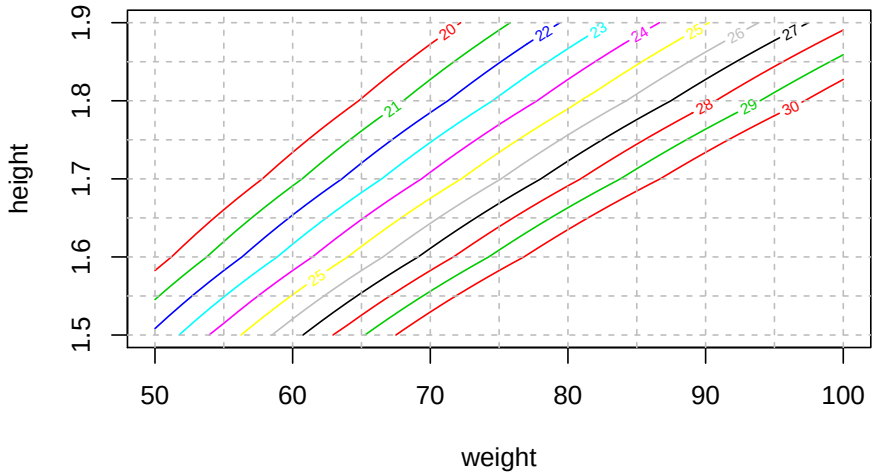
```r
persp(x,y,index,xlab="weight",ylab="height",zlab="bmi",theta=-90,phi=30,
      ticktype ="detailed") # rotazione della figura
```




```r
contour(x,y,index,xlab="weight",ylab="height") # linee di livello
```



```r
# linee di livello da 20 a 30 con colori diversi
contour(x,y,index,xlab="weight",ylab="height",col=2:11,levels=20:30)
# si aggiunge una griglia
abline(h=seq(1.5,1.9,by=0.05),lty=2,col="grey")  # funzione grafica di secondo livello
abline(v=seq(50,100,by=5),lty=2,col="grey")
```



4.3 Espressioni di controllo ripetitive

L'espressione di controllo ripetitiva (`loop statement`) chiamata `for statement` ha la seguente forma:

```r
for (nome in espressione1) espressione2
```

In questo contesto `nome` è la variabile che determina le iterazioni, `espressione1` è un vettore di espressioni, in genere è semplicemente una sequenza di numeri come ad esempio `1:20`, e `espressione2` è un gruppo di espressioni, di cui alcune coinvolgono la variabile di iterazione `nome`. Quindi `espressione2` viene valutata ripetutamente fino a che `nome` si trova nell'insieme dei valori specificati dal vettore prodotto da `espressione1`. L'utilizzazione di `for loop statement` in R non è molto frequente, dal momento che l'implementazione non risulta molto efficiente dal punto di vista dei tempi di esecuzione.

La seguente funzione calcola i primi termini di una serie geometrica, avendo specificato il valore iniziale e la ragione.

```
# I primi n termini di una serie geometrica con valore iniziale a0 e ragione r
geom<-function(n,r,a0)
  {
        ser<-numeric(n)
        ser[1]<-a0
        for (i in 2:n)
        ser[i]<-ser[i-1]*r
        return(ser)
  }

geom(10,0.5,1)

 [1] 1.000000000 0.500000000 0.250000000 0.125000000 0.062500000
 [6] 0.031250000 0.015625000 0.007812500 0.003906250 0.001953125
```

Altre espressioni di controllo ripetitive sono il `repeat statement`, che ha la seguente forma

```
repeat espressione
```

e il `while statement`, definito come

```
while (condizione) espressione
```

Infine si può ricordare il `break statement`, che può essere utilizzato per terminare l'esecuzione di un qualsiasi `loop statement`. In particolare, il `break statement` è l'unico modo per terminare un `repeat statement`. Infine, utilizzando un `next statement`, è possible interrompere una particolare esecuzione di comandi e passare all'esecuzione delle espressioni successive.