Germano Rossi

L'uso di R in psicologia

Work in progress

Versione: 0.8.737

2012-2015

Versione: 0.8.737 (10 giugno 2015)

Germano Rossi (germano.rossi@unimib.it)
Università degli Studi di Milano-Bicocca, Dipartimento di Psicologia

Questa dispensa è sostanzialmente un "memo" cartaceo per me stesso. R è un programma complesso e spesso mi dimentico le sintassi, quindi ho deciso di mettere per iscritto le soluzioni che ho trovato nei libri su R o che ho scoperto da solo. D'altra parte, quello che ho trovato, potrebbe servire anche ad altri. E proprio per questo motivo lo uso come dispensa per i corsi su R.

Insieme al file in PDF (se scaricate la versione `zip`) ci sono alcuni file (ad es. `L.rda...`) che contengono dati e che ipotizzo vengano copiati nella stessa directory in cui poi lavorerete (cfr. l'avviso **Importante** a p. 8).

Avvertenze: è anche un lavoro *in progress*, per cui ci sono diversi buchi o parti che devo ancora completare o scrivere del tutto (sono generalmente indicati entro parentesi quadre).

Indice

Parte I

Introduzione a R

Capitolo 1

Introduzione

R (R Development Core Team, 2012) è un programma di statistica gratuito (o meglio *open-source*) che può essere pensato come una mega-calcolatrice interamente programmabile. Dopo aver installato R, l'utente si trova a disposizione moltissime procedure statistiche basilari e molte altre le può aggiungere tramite quelli che vengono chiamati "pacchetti" (*package* o librerie) messi a disposizione dagli altri utenti e che contengono procedure per statistiche non ritenute basilari. In realtà certi pacchetti contengono miglioramenti di alcune procedure basilari e se l'utente le utilizza spesso possono diventare indispensabili.

1.1 Installazione

Il primo passo è, ovviamente, quello di scaricare la versione più recente del software dal sito "engThe Comprehensive R Archive Network" (abbreviato in CRAN, http://cran.r-project.org/ o da un suo "mirror"). Arriverete ad una pagina simile a quella di Figura 1.1 e dal riquadro "Download and Install R" scegliete la versione di R per il vostro sistema operativo (SO). Una volta entrati nella pagina relativa al vostro SO, navigate fino al punto in cui potrete scaricare l'ultima versione disponibile.

Per quanto riguarda Windows, fino alla versione XP si trattava di scaricare ed eseguire senza problemi un solo file (ad es. R-3.1.1-win.exe). Per Windows Vista, Windows 7 e Windows 8 il procedimento è simile a quello di XP, ma bisogna avere i diritti di amministratore; inoltre, adesso, il file di installazione include sia la versione a 32 bit sia quella a 64 bit; tuttavia se non

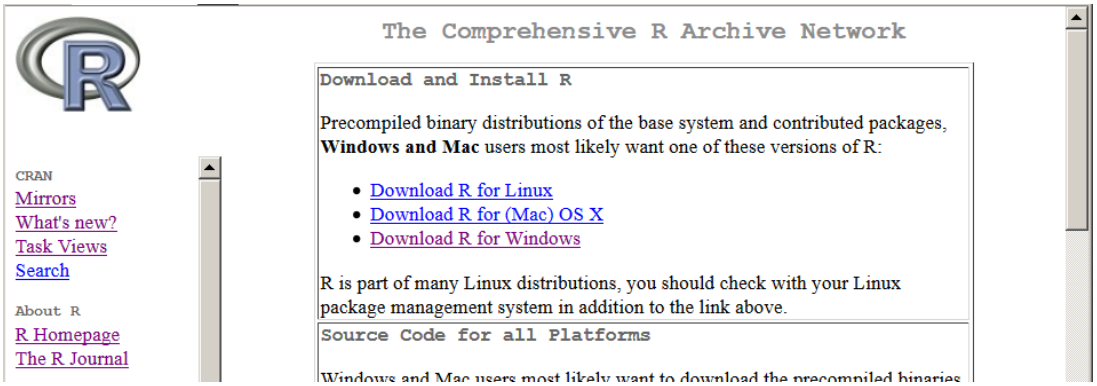

Figura 1.1: Pagina iniziale di CRAN

date indicazioni specifiche, verrà installata la versione corrispondente al vostro SO. Considerate però che certi pacchetti funzionano solo nella versione a 32 bit, mentre altri pacchetti (facendo chiamate dirette al SO, richiedono la versione 32 bit o 64 bit in base alla versione di SO installato); per fortuna è possibile avere entrambe le versioni (32 bit e 64 bit) installate contemporaneamente, anche se è un pochino difficile . Alla fine dell'installazione, il software (e alcuni pacchetti standard) sarà installato in `c:\Program Files\R\R-3.1.1` o simile.

Per i sistemi OSX di Mac e le varie implementazioni di Unix (tra cui Linux) le cose sono un pochino più complicate a causa dei diritti utente, modifiche fra le diverse versioni di OSX e simili. È quindi importante leggere la documentazione sull'installazione e i vari documenti disponibili sui vari CRAN.

Il secondo passo sarà quello di scaricare eventuali librerie di procedure e funzioni statistiche necessarie al proprio lavoro. Ma questo si può fare direttamente dall'interno di R (vedi il paragrafo 1.6).

1.2 Aggiornamenti

Abbiamo due tipi di aggiornamenti: il programma R stesso o i *package* (o librerie) di R. Per il programma R ci sono due possibilità: installare la nuova versione accanto alla prima oppure disinstallare la versione vecchia e poi installare la nuova.

1.2.1 Installazione parallela

La versione più recente di R verrà installata come se fosse nuova e dovrete installare manualmente le eventuali librerie che siete abituati ad usare. Nelle ultime versioni di Windows, quando installate il primo pacchetto, R vi verrà chiesto se volete usare una libreria personale, rispondendo di "si", verrà creata una directory `win-library\versione` nella vostra cartella `Documenti` (un'altra finestra di dialogo vi informa sul percorso esatto in cui verrà creata). Se rispondete di "no", R cercherà di installare il pacchetto nella cartella `library` dove sono stati installati i pacchetti pre-installati; l'operazione potrebbe non dare un esito positivo, in base al SO e ai diritti di cui dispone il vostro utente.

1.2.2 Sostituzione di una versione precedente

In questo caso, prima di installare la nuova versione, provvedete a disinstallare quella precedente. Dopo la rimozione della vecchia versione, rimane una cartella library contenente tutte le librerie aggiunte dall'utente (per lo meno nelle versioni Windows). È sufficiente copiarle nella cartella library della nuova installazione e poi fare l'aggiornamento dei pacchetti (vedi 1.6.3).

In alcuni casi, R attiva due cartelle library, una nella cartella in cui ha installato il software e un'altra nell'area degli utenti. Ad es. in Windows 7 e successivi, potrà esistere una directory `c:\Program Files\R\R-3.1.1\library` e una `c:\Users\nomeutenza\Documents \R\win-library\3.0`.

1.3 Primi passi

In Windows (fino alla 7), avviate R tramite la solita procedura di avvio dei software, ovvero `Start | Tutti i programmi | R | R 3.0.1` (o comunque la versione di R che avete installato). Con gli altri sistemi operativi, avviate R usando le stesse modalità degli altri software.

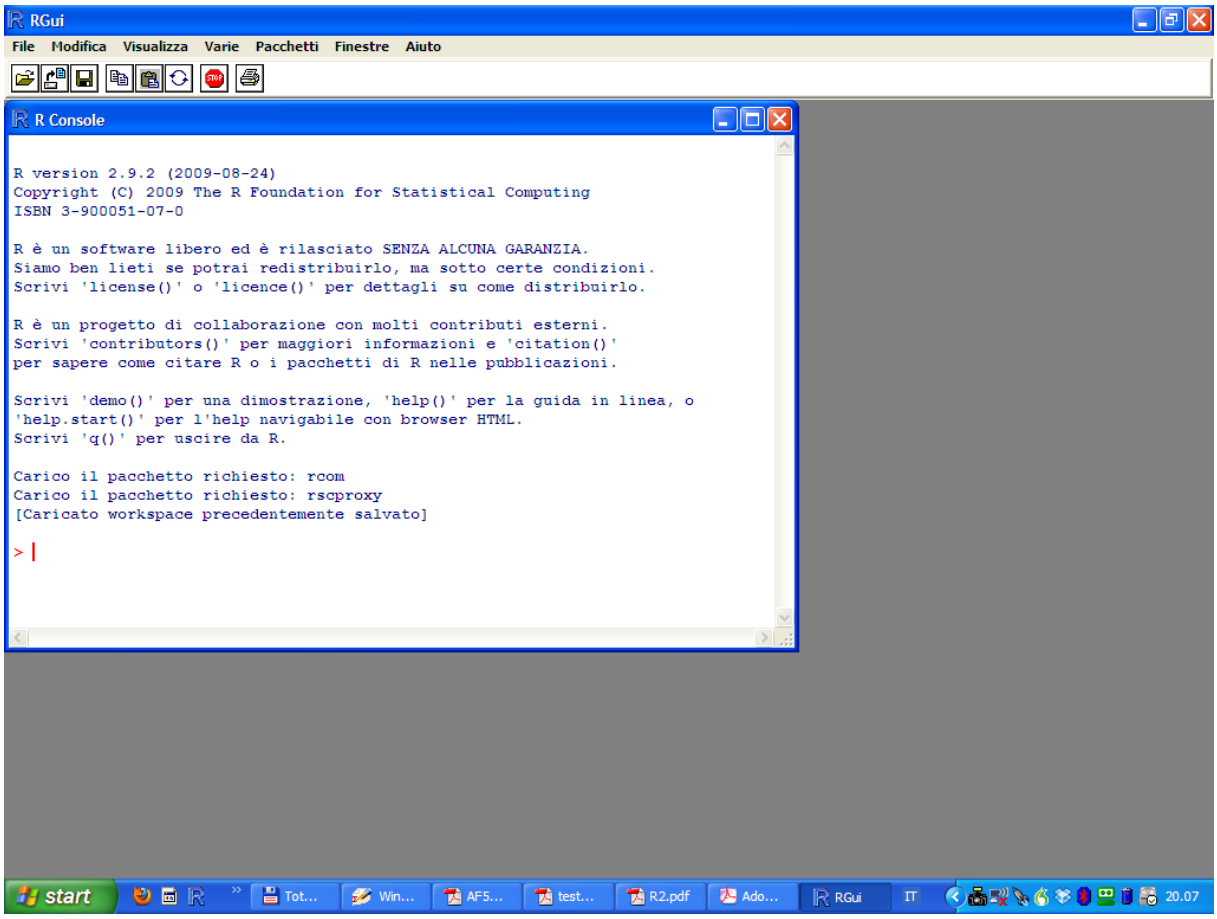

Figura 1.2: Il desktop di R e la finestra dei comandi

Viene avviato il programma R che mostra un *desktop* grigio (in Figura. 1.2 la finestra della versione Windows) e una finestra aperta (la finestra dei comandi), con un messaggio iniziale e il *prompt* dei comandi ">".[1] R usa il simbolo ">" per indicare che sta aspettando un comando e usa il simbolo "+" per indicare che il comando precedente non è stato terminato e aspetta quindi la continuazione di un comando.

Ecco un esempio di comando:

```
R> 7+(12/2)-(4*2)
[1] 5
```

Su una stessa riga ci possono essere più comandi separati da un ; (ma i risultati di ogni comando sono su righe separate):

```
R> 4 +9 ; 7/3
[1] 13
[1] 2.333333
```

1.3.1 Semplici calcoli

R si può usare come una calcolatrice; basta scrivere le operazioni matematiche che si vogliono calcolare (dopo il *prompt*) usando i simboli della Tabella 1.1. R usa la normale logica algebrica, risolvendo prima le moltiplicazione e le divisioni, quindi le addizioni e le sottrazioni. Usando le parentesi tonde si può alterare l'ordine di soluzione (si usano comunque e solo le parentesi tonde, anche se innestate):

```
R> 4+5*3
[1] 19
R> (4 + 5) * 3
[1] 27
> ((4+5) * 3) - (4+5 *3)
[1] 8
```

Tabella 1.1: Operazioni matematiche disponibili in R

Operazione	*Simbolo da usare*	*Esempio*	*Operazione*	*Simbolo da usare*	*Esempio*
Addizione	+	`4+5`	Sottrazione	`-`	`4-5`
Moltiplicazione	*	`4*5`	Divisione	`/`	`4/5`
Potenza	^	`3^2`	Radice	`sqrt()`	`sqrt(9)`

Gli spazi possono essere utilizzati per migliorare la leggibilità delle operazioni che vogliamo fare, ma non incidono sui calcoli. Nell'esempio precedente, all'inizio di ogni riga dei risultati, compare `[1]` che indica che quel risultato è composto da un solo valore. Qualora ci fossero più valori e fosse necessario andare a capo, ogni riga nuova inizierà con un numero che indica la posizione del valore mostrato. Lo si vedrà in diversi degli esempi che compariranno in questo testo.

Possiamo usare R anche per ottenere risultati logici:

[1] È possibile cambiare questo prompt dei comandi (con options(prompt='testo')) ed è quello che ho fatto per rendere più semplice la comprensione degli esempi. D'ora in avanti, tutti gli esempi useranno il simbolo "R>" per indicare un comando che dovreste inserire voi (ma non dovrete scrivere "R>").

```
R> 15*2 > 30
[1] FALSE
R> 15*2 >= 30
[1] TRUE
```

Gli operatori logici che si possono usare sono elencati nella Tabella 1.2.

Tabella 1.2: Operatori logici

simbolo	funzione
==	uguale a
!=	diverso da
>	maggiore di
>=	maggiore o uguale a
<	minore di
<=	minore o uguale a
|	OR
&	AND

1.3.2 Modifica dei comandi

L'editor di R dispone di alcuni comandi per "modificare" il testo. Quando si inserisce un comando siamo sempre in modalità "inserimento" e non serve a nulla premere il tasto "Ins" per passare alla sovrapposizione (come invece si può fare in molti altri programmi). Però, si può usare il tasto di cancellazione del carattere precedente (←—) oppure "Canc" (o "Del") per cancellare il carattere su cui è posizionato il cursore. Premendo il tasto "freccia su" si recupera l'istruzione precedente, che si può modificare, spostandosi con i tasti orizzontali del cursore.

Comandi più avanzati (ma non molto) per modificare variabili o inserire dati un poco più complessi sono edit() e fix(). Entrambi si adeguano al tipo di oggetto da modificare: se si tratta di comandi, viene aperto un piccolo editor, mentre se si tratta di dati viene usata una sorta di griglia (tipo tabellone elettronico). La differenza fra edit e fix è sostanzialmente nel modo di funzionare: edit() restituisce la versione modificata che dev'essere quindi assegnata ad una variabile. Mentre fix() modifica direttamente il contenuto della variabile. L'uso di questi due comandi verrà mostrato nel paragrafo 2.2.8.1.

Sono disponibili soluzioni alternative alle possibilità spartane di R: "editor" specializzati nella comunicazione con R, come Tinn-R (http://www.sciviews.org/Tinn-R/) o RStudio (http://rstudio.org/); macro per utilizzare alcuni editor come interfaccia con R, come Winedt (http://www.winedt.com/) o Emacs (http://www.gnu.org/software/emacs/); interfacce fra Excel ed R, come StatConn (http://www.statconn.com/products.html), di cui verrà detto qualcosa più avanti.

1.3.3 Uscire da R

Per uscire da R si usa il comando:

```
R> q()
```

abbreviazione di quit(). In alternativa, in Windows si può fare *click* sul pulsante di chiusura della finestra oppure (nelle varie versioni GUI, *Graphic User Interface*) usando il menù `File | Esci`.

Prima di uscire, il programma chiede se volete salvare l'area di lavoro ed è sufficiente rispondere S o N. Rispondendo sì, l'area di lavoro viene salvata usando il comando save.image() (cfr. 2.2.10). Tutte le variabili vengono salvate in un file chiamato "`.Rdata`" e tutti i comandi in un file chiamato "`.Rhistory`".

Se non volete essere costretti a premere S o N, potete passare il parametro "`save=`" impostato a "`yes`" o a "`no`":

```
R> q("no")
```

1.3.4 Rientrare in R

Alla successiva esecuzione di R, l'area di lavoro verrà interamente ripristinata, ovvero avrete a disposizione tutte le variabili su cui avete lavorato (compresi gli eventuali file di dati) e l'elenco dei comandi che avete usato nella sessione precedente.

I file "`.Rdata`" e "`.Rhistory`" vengono salvati e caricati in una cartella (o directory) di default. Se volete sapere (mentre siete in R) qual è questa cartella, usate il comando getwd().

Mentre siete in R, potete cambiare directory usando il comando setwd() indicando come parametro il percorso a cui spostarsi che dev'essere inserito fra virgolette (semplici o doppie) e scritto non usando la barra rovesciata ('\') tipica di Windows, ma la barra normale ('/') oppure la doppia barra rovesciata ('\\'), come nell'esempio che segue.

```
R> getwd()
[1] "c:/Documenti/Tex/Dispense/sessione"
R> setwd("c:/Documenti") # oppure
R> setwd("c:\\Documenti")
```

Le versioni più recenti di Windows permettono di avere nomi di directory contenti caratteri accentati o simboli vari. È bene però non usare questi nomi per cartelle con cui pensate di lavorare con R perché non tutte le versioni accettano i caratteri accentati o i simboli; inoltre molti simboli sono caratteri riservati del sistema e potrebbero essere mal interpretati.

Nei sistemi Windows, potete fare doppio click sul file "`.Rdata`" presente in una qualche directory, per far partire una sessione di R con le variabili e la storia dei comandi usati in quella precedente "sessione". Il percorso di cartella viene impostato automaticamente.

Importante: per gli esempi presenti in questo testo, è importante avere accesso alla directory in cui avete estratto il file con i vari dati (VeroSe.xls, Quest.rda, ecc.). Potete quindi "spostare" la directory di default di R su quella cartella, ogni volta che eseguirete gli esempi di questo libro.

Per farlo, usate setwd() con il nome della cartella oppure (se siete in Windows) fate il doppio click sul file .Rdata presente in quella cartella.

Se, per un qualunque motivo, il doppio click non funzionasse (con Windows non si mai!) potrebbe essere utile dare la seguente sequenza di comandi (ipotizziamo che i dati siano in `c:\dati\`):

```
R> rm(list=ls()) # cancella tutte le variabili
R> setwd("c:/dati/") # cambiamo cartella
R> load("c:/dati/.RData") # carichiamo l'ambiente
```

1.4 Impostazioni

R può essere configurato secondo le proprie necessità. Il comando options() visualizza tutte le opzioni impostate e permette di modificarle. Le opzioni visualizzate dipendono dai pacchetti attivi al momento. Se non avete ancora caricato pacchetti vostri, verranno visualizzate le opzioni dei pacchetti di base.

Quando chiediamo delle statistiche, R utilizza un certo numero di cifre (inclusi i decimali e il punto decimale). Per sapere quante sono queste cifre, possiamo scrivere R> options("digits") ma è consigliato usare invece

```
R> getOption("digits")
[1] 7
R>17/3
[1] 5.666667
```

options() si usa invece per modificare i parametri. Per impostare a 3 il numero delle cifre visualizzate, usiamo

```
R> options(digits=3)
R>17/3
[1] 5.67
```

È evidente da questo esempio che non conviene cambiare il valore originale, dal momento che incide su tutti i numeri e non solo sui decimali.

La stragrande maggioranza dei parametri in questione ha un valore "ottimale" e quindi non è consigliabile modificarla se non per motivi particolari. Una delle poche opzioni che potreste voler cambiare è width che corrisponde al numero di caratteri per riga in fase di stampa (ma anche di visualizzazione dei risultati). Il valore di default è 80 e va bene per il formato A4. Se volete cambiarlo usate:

```
R> options("width"=100) % esempio
```

Una volta modificati, i parametri restano fissati fino all'uscita da R. Al nuovo ri-avvio tornano quelli impostati inizialmente. Se volete modificare in modo permanente un parametro, dovete farlo tramite Rprofile.site.

1.4.1 Rprofile

Per modificare in modo permanente un parametro, dovete cercare il file Rprofile.site (generalmente nella cartella etc all'interno della cartella in cui è stato installato il software) e aggiungere, in quel file, il comando desiderato.

```
options(parametro=valore)
```

Altri file che contengono "impostazioni " varie sono presenti nella stessa directory e cambiano in base al tipo di installazione. Ad es. nella versione per Windows, il file Rprofile.site permette di impostare il formato degli aiuti o l'editor da usare in caso di modifica o scrittura di proprie funzioni, mentre il file Rconsole (senza estensione) permette di impostare alcune opzioni della finestra in cui "gira" R: dal font da utilizzare ai colori.

1.5 Aiuto

A seconda della versione di R che avete installato (e delle scelte effettuate in fase di installazione), l'aiuto sarà basato su pagine web in locale (scelta predefinita) che si apriranno usando il vostro *browser* internet di default (IE, Firefox, Chrome...) oppure su un file help di Windows (con estensione .chm). In ogni caso, si può ottenere aiuto tramite il comando help() oppure tramite una sua abbreviazione ? (punto di domanda).

R offre numerosissime forme di aiuto. Se è il vostro primo approccio con R e non avete idea di cosa fare, potete usare il comando help.start() che aprirà una pagina (in inglese) con suggerimenti su cosa fare e link a dove trovare le informazioni: introduzione a R, importare ed esportare dati da R e così via.

Se conoscete il nome della funzione o del comando su cui volete aiuto (ad es. la media, in inglese *mean*) potere scrivere:

```
R> help(mean) # oppure
R> ?mean
```

Se è impostato l'aiuto di tipo "text", comparirà una finestra che presenta la sintassi (in inglese), spiega i parametri e indica i risultati che vengono "memorizzati" se li assegnate ad una variabile; infine vengono indicati i riferimenti bibliografici e alcuni esempi di utilizzo della funzione.

Se è impostato l'aiuto di tipo "html", il comando avvia una pagina nel vostro browser di default, contenente le stesse informazioni della versione testuale.

In alcuni casi, gli esempi indicati possono essere eseguiti tramite il comando example():

```
R> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

Se invece non avete idea di come possa chiamarsi la funzione che svolge una determinata operazione, potete usare l'estensione help.search() che cerca direttamente nei testi (sempre usando l'inglese), anche tramite l'abbreviazione ?? (doppio punto interrogativo):

```
R> help.search("mean") # oppure
R> ??mean
```

In risposta a questa richiesta comparirà una finestra che elenca tutte le funzioni (e i rispettivi pacchetti) che hanno a che fare con la parola cercata (o le parole).

Un'altra possibilità del comando help() è relativa ai "pacchetti" e permette di sapere quali funzioni contiene un pacchetto (già installato):

```
R> help(package="polycor")
```

visualizzerà le funzioni e i dati disponibili all'interno del pacchetto di nome polycor.

Infine, R contiene dei comandi che danno informazioni da usare per i riferimenti bibliografici: citation(). Chiamato da solo restituisce il modo di citare R stesso. Indicando il nome di un pacchetto (ad es. citation("ca") restituisce il modo di citare quel pacchetto.

In entrambi i casi, viene visualizzata una versione che si può incollare direttamente nella bibliografia del testo dove volete fare la citazione e anche la versione BibTEX per chi scrive usando LATEX:

```
Michael Greenacre and Oleg Nenadic (2010). ca: Simple, Multiple and
Joint Correspondence Analysis. R package version 0.33.
http://CRAN.R-project.org/package=ca

A BibTeX entry for LaTeX users is

  @Manual,
    title = ca: Simple, Multiple and Joint Correspondence Analysis,
    author = Michael Greenacre and Oleg Nenadic,
    year = 2010,
    note = R package version 0.33,
    url = http://CRAN.R-project.org/package=ca,
```

1.6 Pacchetti o librerie

I pacchetti (*package*) o librerie contengono funzioni che ampliano le possibilità di R e rispondono a molte esigenze. In genere, molti statistici, dopo aver sviluppato teoricamente un nuovo metodo di analisi, lo implementano in R e solo quando la tecnica è entrata nell'uso comune, viene integrata nei pacchetti statistici più *user-friendly*.

Sono disponibili più di 1500 pacchetti. Un elenco dei pacchetti con una breve descrizione del loro contenuto si trova alla voce `Packages` del sito CRAN (il più vicino a Milano è http://rm.mirror.garr.it/mirrors/CRAN/).

1.6.1 Usare i pacchetti

Una volta trovato il pacchetto che ci interessa, per poterlo usare abbiamo due diverse operazioni da fare: installarlo e caricarlo.

- "Installare" significa scaricare da internet il pacchetto e renderlo disponibile a R; è un'operazione che si esegue una sola volta.
- "Caricare" significa invece rendere attivo il pacchetto nel momento in cui lo vogliamo usare; questa operazione può essere fatta solo se il pacchetto è già stato installato e deve essere ripetuta ogni volta che lo vogliamo usare.

La gran parte dei pacchetti sono disponibili sul sito CRAN e possono essere installati direttamente da lì. Alcuni pacchetti (in genere le versioni non definitive) sono disponibili solo in alcuni siti particolari (ad es. quello dello sviluppatore) e bisognerà quindi conoscere l'indirizzo web da cui scaricarlo.

Se il pacchetto è disponibile sul sito CRAN, avviamo R e usiamo il menù `Pacchetti | Installa pacchetti...`; si aprirà un finestra con l'elenco di tutti i *mirrors* del sito CRAN, selezioniamo il più vicino a noi, si apre l'elenco dei pacchetti disponibili, selezioniamo uno o più

pacchetti e diamo l'OK. R scaricherà il pacchetto richiesto e tutti i pacchetti ad esso collegati (i pacchetti richiesti da quello che vogliamo installare); alla fine li installerà.

Se il pacchetto è stato già scaricato da internet (in genere in formato `.zip` o `.gz`) e lo abbiamo salvato in una cartella del nostro computer, allora si può usare la voce di menù `Pacchetti | Installa pacchetti da file zip locali`. Scelto questo comando, si apre una finestra di navigazione per poter trovare e selezionare (sul proprio computer) il file da installare.

C'è anche la possibilità di installare un pacchetto direttamente da R tramite il comando install.packages(), ma bisogna conoscerne il nome esatto (ricordiamo che R distingue fra maiuscole e minuscole). Ad es. per installare il pacchetto psy si può scrivere:

```
R> install.packages("psy")
```

Il comando install.packages() è l'unione di una serie di altri comandi:

- available.packages() per avere l'elenco dei pacchetti disponibili;
- installed.packages() per sapere quali sono già stati scaricati;
- download.packages() per scaricare quelli selezionati.

Potremmo anche chiamare singolarmente ciascuno di questi pacchetti. Ovviamente il comando install.packages() (chiamato anche tramite menù) si occupa di tutto in modo automatico, a noi non resta che selezionare il *mirror* e i pacchetti che vogliamo installare.

Notate però, che per R, "installare" significa solo renderlo disponibile al caricamento; quindi una volta installato il pacchetto dev'essere caricato in R ogni volta che lo volete usare[2].

C'è anche la possibilità di avviare R con una serie predefinita di pacchetti. [da FINIRE].

Per caricare un pacchetto (che sia già stato installato sul vostro computer) ci sono due possibilità:

- ricorrere nuovamente al menù `Pacchetti`, selezionando `Carica pacchetto...`; si aprirà una finestra con tutti i pacchetti installati, da cui potrete selezionarne uno o più e dare l'OK.
- usare il comando

```
R> library(nomepacchetto)
```

Spesso esistono vari pacchetti che svolgono funzioni simili ma in modo diversi. È quindi possibile che due o più pacchetti forniscano un comando con lo stesso nome (ad es. `describe()` è disponibile nei pacchetti `prettyR`, `Hmisc` e `psych`). Se abbiamo attivato due pacchetti che hanno entrambi un comando con lo stesso nome, solo l'ultimo sarà attivo. Quando caricate il secondo pacchetto vedrete un messaggio del tipo:

```
Attaching package: 'Hmisc'

The following object is masked from 'package:psych':

    describe
```

Per usare uno o l'altro comando evitando conflitti, si fa precedere il nome del pacchetto al nome del comando (separandolo con "::"):

[2] Uscendo da R, i pacchetti caricati vengono "scaricati" e al successivo avvio non saranno più disponibili a meno di non caricarli di nuovo.

```
R> psych::describe(x) # esegue il comando di psych
R> Hmisc::describe(x) # esegue il comando di Hmisc
```

1.6.2 Informazioni e aiuto

Per sapere quali pacchetti sono già stati installati nel vostro computer, usate semplicemente library() mentre per sapere quali pacchetti sono al momento attivi nella sessione di R che state usando usate search().

Per sapere quali procedure e funzioni sono disponibili in un pacchetto, potete usare help(package="nomepacc per avere il nome e una breve descrizione, se invece volete solo sapere il nome, potete usare ls("package:nomepacchetto"). Notate che, in base alle scelte iniziali di installazione, il comando help() può visualizzare l'aiuto in una finestra del vostro *browser* internet.

Molti pacchetti contengono dati esemplificativi o addirittura degli esempi o delle dimostrazioni. Caricando un pacchetto, tutti i dati che contiene vengono caricati in memoria e sono disponibili.

Il comando generico data() visualizza tutti i dati disponibili nei pacchetti attivi al momento (quindi quelli inclusi nei pacchetti che avete caricato). Per vedere quali dati sono compresi in un determinato pacchetto usiamo:

```
R> data(package="nomepacchetto")
```

Se però volete caricare un certo blocco di dati senza caricare tutto il pacchetto, potete usare il comando:

```
R> data(nomedati,package="nomepacchetto")
```

Per attivare gli esempi disponibili in un pacchetto, usiamo example() indicando (all'interno della parentesi) il nome del pacchetto o della procedura/funzione di cui vogliamo vedere l'esempio. Se indichiamo il nome di un pacchetto vengono eseguiti tutti gli esempi disponibili, se invece indichiamo il nome di una funzione, viene eseguito solo l'esempio di quella funzione. Attenzione, però che non tutti i pacchetti o le procedure dispongono di esempi.

Per le dimostrazioni, l'istruzione `R> demo()` visualizza tutti i demo disponibili in tutti i pacchetti caricati, mentre

```
R> demo(package = .packages(all.available = TRUE))
```

visualizza i demo disponibili in tutti i pacchetti installati.

Per eseguire un demo, basta scrivere il nome entro le parentesi. Guardate un demo delle capacità grafiche di R con il comando

```
R> demo(graphics)
```

Alcuni pacchetti, oltre al manuale di riferimento, contengono anche uno o più manuali d'uso (sul modo di usare le varie procedure) che vengono chiamati "vignettes". Per sapere quali *vignette* sono disponibile nei pacchetti installati nella vostra versione di R, scrivete semplicemente:

```
R> vignette()
```

Ci sono diversi modi di usare vignette():

```
R> vignette(all=F) # pacchetti caricati
R> vignette(pac="MiscPsycho") # in un certo pacchetto
R> vignette("MP") # una vignetta specifica (in PDF)
```

Pacchetti diversi che contengono procedure statistiche simili, possono usare gli stessi nomi. In questo caso, se si carica un pacchetto che contiene un oggetto con un nome già esistente, vedrete un messaggio che vi avverte che l'oggetto con un determinato nome (del pacchetto precedente) è stato mascherato (*masked*). L'oggetto esiste ancora, ma quando lo si chiama, viene usato l'ultimo, quello attivo.

Se avete finito di usare un pacchetto (ma non avete finito di lavorare), prima di caricare altri pacchetti potete "scaricare" quello che non vi serve più (liberate memoria di lavoro ed evitate possibili conflitti di oggetti). Per scaricare un pacchetto potete usare il comando:

```
R> detach("package:nomepacchetto")
```

Se, invece volete proprio disinstallarlo (ovvero cancellarlo dal vostro computer), il comando da usare è:

```
R> remove.packages("nomepacchetto")
```

1.6.3 Aggiornamento pacchetti

Per le librerie (o pacchetti), l'aggiornamento è interno; una volta aperto il programma, basta usare la sequenza `Pacchetti | Aggiorna pacchetti...` oppure dare il comando update.packages().

In entrambi i casi, compare una finestra che elenca i *mirror* di CRAN da cui selezionare quello che volete usare, quindi tutti i pacchetti per cui esiste una versione nuova, vengono aggiornati.

Nel vostro sistema potrebbe essere installato un pacchetto che non corrisponde alla versione di R che state usando; in questo caso, quando caricate il pacchetto con `library()` vedrete un messaggio simile a questo:

```
#Errore: package 'xlsxjars' was built before R 3.0.0: please re-install it
```

che vi invita ad aggiornare il pacchetto in questione.

1.6.4 Pacchetto Rcmdr

Tra i pacchetti più utili c'è quello chiamato Rcmdr (Fox et al., 2011) il cui nome completo è "R Commander". Si tratta di un menù *user-friendly* (rientra nelle cosiddette R Gui, *Graphics User Interface*) che permette di accedere a molte altre funzioni di utilità e ad una parte di statistiche basilari.

Una volta installato, il pacchetto viene caricato tramite il solito comando

```
R> library(Rcmdr)
```

`Rcmdr` e compare una finestra simile a quella in Figura 1.3.

La finestra presenta 5 aree: l'area del menù, il pannello di gestione dati, l'area dei comandi, l'area dei risultati e, infine, quella dei messaggi.

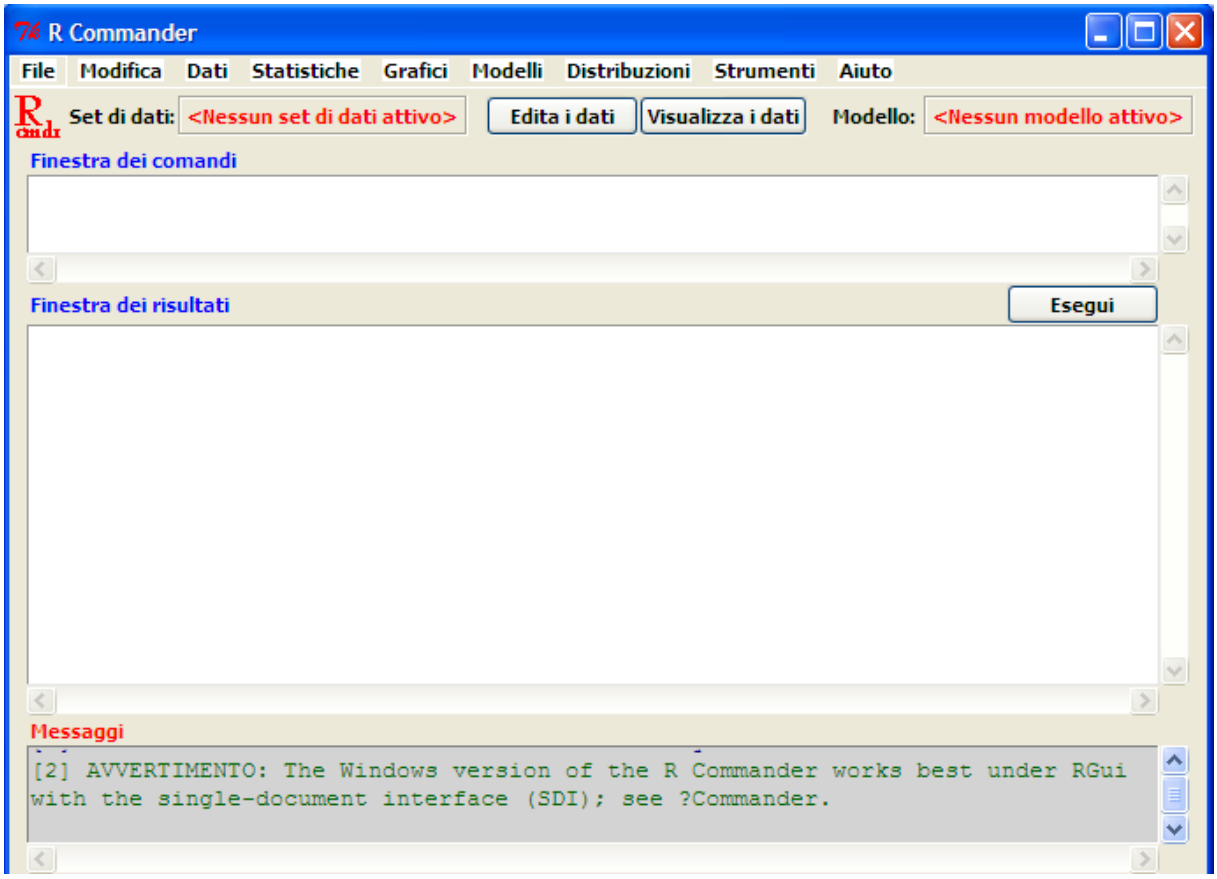

Figura 1.3: Finestra iniziale di R Commander

Lo scopo di R Commander è quello di rendere accessibile anche agli utenti meno abili, la massa di funzioni di gestione dei dati (menù `Dati`), le principali statistiche per l'esplorazione dei dati, le principali inferenze e i principali modelli multivariati (menù `Statistiche` e `Modelli`), i grafici (menù `Grafici` e `Distribuzioni`).

Per caricare dei dati già attivi, basta click-are sulla scritta rossa "<Nessun set di dati attivo>" e selezionarlo.

Nel seguito, gli eventuali riferimenti all'uso di R Commander verranno scritti in questo modo: **Rcmdr:** `Dati | Importa dati | da set di dati di SPSS`, che significa "dalla finestra di R Commander, aprite il menù `Dati`, selezionate `Importa dati` e quindi `da set di dati di SPSS`".

Tra le funzioni più interessanti di R Commander vi sono quelle di "workaround" di funzioni di caricamento dei dati la cui sintassi è sempre complicata da ricordare, ad es. il caricamento di dati da Spss o da Excel (comunque spiegate al par. 2.2.11). Questo comando lo trovate in `Dati | Importa dati`.

In alcuni casi, R Commander carica i pacchetti che gli servono per fare le analisi (e usa le stesse funzioni che potreste usare anche voi), mentre altre volte i comandi sono stati predisposti dallo stesso autore. Ci sono quindi delle situazioni in cui ciò che è possibile fare con R Commander è diverso da ciò che è possibile fare direttamente in R. Starà a voi decidere cosa fare in R Commander e cosa in R.

Capitolo 2

Tipo di dati

In R ogni dato è un "oggetto" di diverso tipo differenziato per "modo" e "classe". I comandi `mode()`, `class()` e `typeof()` danno le rispettive informazioni sugli oggetti utilizzati.

Per semplificare le cose, possiamo chiamarli genericamente "tipi" e classificarli in tipi "semplici" e "complessi". I tipi semplici sono i tipi numerici (numeric), il testo (character, anche chiamato valore "stringa") e i valori logici. I tipi "complessi" sono "vettori", "matrici", "elenchi" (list) e "dataframe" (data.frame). Li vedremo tutti fra poco.

Anche se al paragrafo 1.3 abbiamo visto che i numeri si possono usare direttamente (e la cosa vale anche per i testi e per i valori logici) l'uso più comune dei tipi semplici è con le variabili.

2.1 Variabili

I risultati delle operazioni effettuate si possono "immagazzinare" in variabili tramite un nome che gli si può assegnare.

```
R> A <- 4 + 5
```

Con questo comando immagazziniamo in una variabile chiamata "`A`" il risultato dell'operazione `4+5` (cioè `9`), ma il risultato non verrà visualizzato.

Il simbolo "`<-`" significa "assegnare a" e può essere sostituito da "`assign(nome, valore)`", da "`->`" o da "`=`" usati in questo modo (gli spazi prima e dopo, non sono obbligatori):

```
R> assign("A", 4 + 5)
R> A <- 4+5
R> 4+ 5 -> A
R> A = 4 +5
```

La maggior parte dei libri su R utilizza "`<-`" (perché formalmente corretto), ma molti utenti utilizzano "`=`" perché più semplice (basta essere consapevole con non significa "uguale a" ma invece "assegna a").

Se vogliamo vedere il contenuto di `A` dobbiamo semplicemente scriverne il nome e vedremo il risultato `9`.

```
R> A
[1] 9
```

La variabile può essere usata per fare altri calcoli: `R> A * 3` mostra il risultato 27, mentre `R> B <- A * 3` immagazzina il risultato in `B`.

Per sapere quali variabili sono state create (e sono quindi disponibili) si può usare il comando `ls()` oppure `objects()`.

```
R> ls()
[1] "A"            "B"            "baseball"     "baseball.cor" "RB"
[6] "vars2"
```

In questo caso, l'area di "lavoro" conteneva 6 "oggetti".

I nomi di variabili possono essere lunghi a piacere, possono iniziare con un lettera dell'alfabeto e proseguire con lettere, cifre, punti (.) o sottolineature (_).

Un nome di variabile può anche iniziare con un punto, ma il secondo carattere non dev'essere una cifra (altrimenti verrebbe interpretato come un numero). Tuttavia, una variabile il cui nome inizia con un punto (ad es. `.a`) è una variabile nascosta che non è possibile vedere con `ls()` ma che "lavora" come qualunque altra variabile.

Si possono usare sia lettere maiuscole sia minuscole, ma R è *case-sensitive* (sensibile al maiuscolo/minuscolo), ovvero le variabili che differiscono per l'uso di maiuscolo/minuscolo sono variabili diverse: ad esempio `età`, `Età`, `ETà`, `eTà` sono tutte variabili diverse.

Il comando `ls()` visualizza non solo le variabili ma anche tutti gli altri "oggetti" che sono stati creati (come ad es. le funzioni, i risultati, ecc.) ad eccezione di quelli che iniziano con un numero o con un punto. Per visualizzare anche gli oggetti "nascosti" (il cui nome inizia con un punto) si può usare il parametro all.names=T. Se invece volete un sottoinsieme di variabili, potete usare il parametro pattern= a cui far seguire una stringa di ricerca in forma di "espressione regolare" (v. Tabella 2.1). Ad es. `ls(pat="a")` visualizzerà tutte le variabili che *contengono* una "a", `ls(pat="^a")` tutte quelle che *iniziano* per "a" e `ls(pat="a$")` tutte quelle che *terminano* con una "a".

simbolo	spiegazione	esempio
^	inizio parola	`ls(pat="^a")`
$	fine parola	`ls(pat="a$")`

Tabella 2.1: Espressioni regolari

Le variabili (e anche gli altri oggetti) restano disponibili sempre. C'è quindi la possibilità di cancellarli con il comando `remove()` o `rm()` indicando fra parentesi il nome della variabile da eliminare (oppure i nomi delle variabili separati da una virgola):

```
R> remove(A)
R> rm(b, C)
```

È possibile eliminare più variabili presenti tramite il parametro list= che richiede una lista di nomi; in genere serve per eliminare tutte le variabili che presentino un certo "pattern" (v. Tabella 2.1). Volendo eliminare tutte le variabili basta usare il comando ls() come nell'esempio seguente:

```
R> rm(list=ls()) # tutte le variabili
R> rm(list=ls(pat="^x")) # che iniziano per x
```

2.1.1 Numeri

Le variabili possono contenere tipi di informazioni di diverso formato. Le informazioni usate finora sono semplicemente numeri (o scalari), che possono essere sia interi sia decimali. R usa il punto decimale e non la virgola, quindi eventuali numeri frazionari vanno inseriti con il punto:

```
R> X <- 3.12
R> z <- 1
R> X
[1] 3.12
R> z
[1] 1
```

Il "modo" e la "classe" di questo tipo di variabili è identificato come "numeric", ma il "tipo" è a doppia precisione (anche quando non ha decimali).

```
R> mode(X)
[1] "numeric"
R> class(X)
[1] "numeric"
R> typeof(X)
[1] "double"
R> typeof(z)
[1] "double"
```

Numeri particolarmente grandi o particolarmente piccoli (ad es. alcuni numeri reali) possono essere inseriti usando la notazione scientifica. Questo tipo di notazione indica i numeri usando un solo intero, una serie di decimali, la lettera `e` (che indica esponente) e il numero di posti a destra o sinistra per cui si deve spostare la virgola (ovvero moltiplicare per il valore fisso di 10). Perciò `1e3` equivale a 1 moltiplicato per 1000 (`e3` significa 10^3), mentre `1e-3` equivale a 1 moltiplicato per 0,001 (cioè 10^{-3}).

```
R> E <- 1.345e3
R> F <- 1.345e-4
R> E; F # due comandi sulla stessa linea
[1] 1345
[1] 0.000134
```

A puro titolo informativo, R può usare anche i numeri immaginari (ad es. `3+4.5i`) che sono però già dei tipi complessi (modo "complex").

Per sapere se una variabile è di tipo numerico possiamo usare la funzione is.numeric() che restituirà vero o falso.

```
R> is.numeric(X)
[1] TRUE
R> is.numeric(z)
[1] TRUE
R> is.numeric(E)
```

```
[1] TRUE
R> is.numeric("testo")
[1] FALSE
```

2.1.2 Testo

Una variabile può contenere una stringa, cioè del testo racchiuso fra virgolette semplici (apici) o doppie:

```
R> A <- 'testo'
R> B <- "Esempio di contenuto testuale"
R> A
[1] "testo"
R> B
[1] "Esempio di contenuto testuale"
```

All'interno di un certo tipo di virgolette si può usare l'altro tipo.

```
R> C <- "L'esempio precedente non aveva apici"
R> D <- 'Un "piccolo" esempio'
```

Il risultato sarà mostrato sempre usando le virgolette doppie, eventualmente precedute da una barra rovesciata.

```
R> C
[1] "L'esempio precedente non aveva apici"
R> D
```

```
[1] "Un \"piccolo\" esempio"
```

Il tipo "carattere" corrisponde al modo, alla classe e al tipo "character":

```
R> mode(B)
[1] "character"
R> class(B)
[1] "character"
R> typeof(B)
[1] "character"
```

Per sapere se una variabile è di tipo carattere si può usare la funzione is.carachter():

```
> is.character(A) # stringa
[1] TRUE
> is.character(C) # stringa
[1] TRUE
> is.character(E) # numero
[1] FALSE
>
```

2.1.3 Valori logici

Una variabile può contenere i valori logici *vero* o *falso*, sia come parole (ma in inglese, `TRUE` o `FALSE`) sia come abbreviazione (sempre in inglese, `T` o `F`):

```
R> G <- T
R> H <- FALSE
R> G
[1] TRUE
R> H
[1] FALSE
R> mode(G)
[1] "logical"
R> class(G)
[1] "logical"
R> typeof(G)
[1] "logical"
```

Attenzione a non usare le lettere `T` e `F` come nomi di variabili, perché non sarebbero più disponibili come abbreviazioni di vero e falso:

```
R> T=15
R> T
[1] 15
R> a=T
R> a
[1] 15
R> rm(a)
R> rm(T)
R> T
[1] TRUE
```

Per sapere se una variabili è di tipo logico possiamo usare la funzione is.logical():

```
R> is.logical(15>3)
[1] TRUE
R> is.logical(15)
[1] FALSE
```

2.1.4 Vettori

Una variabile, oltre ad un singolo numero, può anche contenere un "vettore" (o serie) di numeri o di altri tipi di dati, genericamente chiamati "data vector" (vettore dati). Tutti gli elementi di un vettore devono però essere dello stesso tipo: tutti numeri o tutti testo o tutti valori logici (ma possono contenere l'indicazione di un valore mancante, `NA`, cfr. 2.2.7).

Il vettore è il tipo base per gli altri tipi "complessi", in quanto tutti dipendono da un "vettore". La funzione `vector(length=)` serve per creare un vettore e impostare la sua lunghezza. Tuttavia il vettore così creato è vuoto (contiene solo il valore logico FALSE) e bisogna riempirlo con comandi singoli. Non appena si assegna un valore ad uno qualunque degli elementi, tutti gli altri elementi si adeguano al tipo inserito.

Tabella 2.2: Esempio: numero di balene avvistate per anno

anno	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
balene	74	122	235	111	292	111	211	133	156	79

Tabella 2.3: Esempio: Simpsons e relazioni di parentela

Simpsons	Homer	Marge	Bart	Lisa	Maggie
parentela	papà	mamma	figlio	figlia 1	figlia 2

```
R> A <- vector(len=3)
R> A
[1] FALSE FALSE FALSE
> A[1] <- 7
> A
[1] 7 0 0
```

Per questo motivo, il modo più immediato e semplice per creare un vettore, e contemporaneamente immettere i dati corrispondenti, è tramite la funzione `c()`. Questa funzione concatena diversi elementi e restituisce un vettore. Ad es. per inserire i dati della Tab. 2.2.

```
R> balene <- c(74, 122, 235, 111, 292, 111, 211, 133, 156, 79)
R> anno <- c(1990:1999)
R> balene
 [1]  74 122 235 111 292 111 211 133 156  79
R> anno
 [1] 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999
```

Scrivendo il nome della variabile-vettore, si può vedere il suo contenuto.

Nel secondo esempio abbiamo usato una funzione speciale di R (il due punti, cfr. 2.2.1.1.1) che genera un insieme numerico a partire da un valore minimo (1990) fino ad un massimo (1999).

Un vettore dati può contenere anche del testo (variabili nominali/ordinali). Il testo dev'essere delimitato da virgolette semplici (') o da virgolette doppie ("). Per inserire i dati della tabella 2.3, ad es. useremo i seguenti comandi:

```
R> simpsons <- c("Homer", 'Marge', "Bart", 'Lisa', "Maggie")
R> simpsons
[1] "Homer"  "Marge"  "Bart"   "Lisa"   "Maggie"
```

Un vettore dati può associare ad ogni elemento anche un nome, tramite la funzione `names()` che può servire sia per impostarli sia per visualizzarli.

```
R> names(simpsons) <- c("papà", "mamma", "figlio", "figlia 1", "figlia 2")
R> names(balene) <- anno
R> names(simpsons)
[1] "papà"     "mamma"    "figlio"   "figlia 1" "figlia 2"
R> names(balene)
 [1] "1990" "1991" "1992" "1993" "1994" "1995" "1996" "1997" "1998" "1999"
R> simpsons
```

```
    papà    mamma   figlio figlia 1 figlia 2
 "Homer"  "Marge"   "Bart"   "Lisa" "Maggie"
R> balene
1990 1991 1992 1993 1994 1995 1996 1997 1998 1999
  74  122  235  111  292  111  211  133  156   79
```

Nel creare un vettore, si possono inserire contemporaneamente i nomi e i valori, in questo modo:

```
R> esempio <- c(maschio=1,femmina=2)
R> esempio
maschio femmina
      1       2
```

Un vettore dati può contenere anche valori logici (`T`=true=vero e `F`=false=falso) oppure l'indicazione di un dato mancante (`NA`, cfr. 2.2.7).

```
R> logico <- c(T,F,NA,T,T,F)
R> logico
[1]  TRUE FALSE    NA  TRUE  TRUE FALSE
```

Per accedere ad un qualunque elemento di un vettore, lo dobbiamo indicare fra parentesi quadre. Si può estrarre un solo elemento o più elementi tramite una sequenza di posizioni.

```
R> simpsons[2]
R> simpsons[1:3]
R> simpsons[c(1,3)]
```

Usando la funzione `:` abbiamo elencato valori consecutivi con inizio da 1 e fine a 3 (quindi 1, 2 e 3), mentre con la funzione `c()` abbiamo elencato esplicitamente valori non consecutivi. Ovviamente le due modalità si possono mischiare all'interno di `c()`: `c(1:3, 5, 9, 10:15)`

Se abbiamo associato dei nomi al vettore possiamo selezionare i valori anche attraverso i loro nomi:

```
R> simpsons["mamma"]
  mamma
"Marge"
```

Per sapere se una variabile è un vettore, si può usare la funzione is.vector():

```
R> is.vector(balene)
[1] TRUE
```

Dal momento che un vettore contiene dei valori, è possibile verificare il tipo del contenuto con le funzioni:

```
R> is.numeric(balene)
[1] TRUE
R> is.character(balene)
[1] FALSE
R> is.logical(balene)
[1] FALSE
```

2.1.5 Fattori

Con il nome "fattori" (*factors*), R chiama le variabili misurate a livello nominale o ordinale. Un fattore contiene dei valori (caratteri o numeri) suddivisi in *livelli* (le possibili categorie della variabile) ed esistono fattori con livelli ordinati e altri senza ordine.

Anche se è possibile inserire direttamente dei dati nel formato "fattore", è in genere più comodo convertire un vettore esistente in questo formato con il comando factor() (variabili nominali) oppure ordered() (variabili ordinali).

2.1.5.1 Fattori non ordinati

Con il comando factor() possiamo trasformare un vettore in un fattore. Nell'esempio seguente, abbiamo inserito i nomi dei Simpsons in una variabile di tipo carattere; con il comando levels() chiederemo di elencare i livelli della variabile e otterremo un NULL in risposta perché non ci sono livelli.

```
R> simpsons <- c("Homer", 'Marge', "Bart", 'Lisa', "Maggie")
R> class(simpsons) # vettore caratteri
[1] "character"
R> levels(simpsons) # non ci sono livelli
NULL
```

Usando il comando factor() creiamo una nuova variabile (simp) che questa volta sarà di tipo "fattore". Notate come la risposta al comando levels(), questa volta, elencherà i Simpsons in ordine alfabetico rispetto all'ordine che abbiamo usato per inserirli.

```
R> simp <- factor(simpsons) # trasformiamo in fattore
R> class(simp)
[1] "factor"
R> levels(simp)
[1] "Bart"   "Homer"  "Lisa"   "Maggie" "Marge"
```

La variabile `simp`, però, contiene i valori nell'ordine in cui li abbiamo inseriti:

```
R> simp
[1] Homer  Marge  Bart   Lisa   Maggie
Levels: Bart Homer Lisa Maggie Marge
```

Avremmo potuto creare direttamente una variabili di tipo fattore con:

```
R> simp2 <- factor(c("Homer", "Marge", "Bart", "Lisa", "Maggie"))
R> simp2
[1] Homer  Marge  Bart   Lisa   Maggie
Levels: Bart Homer Lisa Maggie Marge
>
```

Possiamo anche trasformare una variabile numerica in una di tipo "fattore" usando il comando as.factor().

```
R> A <- sample(c(1:3),20,replace=T)
R> A
 [1] 1 2 2 2 3 1 2 2 2 3 1 2 1 1 2 2 2 3 3 3
R> levels(A)
NULL
R> A=as.factor(A)
R> levels(A)
[1] "1" "2" "3"
```

In questo esempio è stata usata la funzione sample() per creare 20 valori casuali, usando i numeri da 1 a 3 con possibilità di ripetizione (vedi **??**).

Possiamo anche fare l'operazione inversa e trasformare una variabile fattore numerico in una variabile numerica con as.numeric():

```
R> a=as.numeric(A)
R> a
 [1] 1 2 2 2 3 1 2 2 2 3 1 2 1 1 2 2 2 3 3 3
```

2.1.6 Matrici

In R il termine "matrice" può assumere due significati, quella di vera e propria matrice algebrica e quella di "contenitore" bidimensionale, in quanto una matrice è semplicemente un contenitore con più righe e più colonne. In realtà, se pensiamo a righe e colonne sbagliamo perché una matrice di R può avere anche più di 2 dimensioni. Una matrice con due dimensione può essere trattata come una vera e propria "matrice" matematica.

Usando l'istruzione dim() si possono assegnare delle dimensioni ad un vettore esistente:

```
R> mat <- 1:12
R> mat
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
R> mode(mat); class(mat); typeof(mat)
[1] "numeric"
[1] "integer"
[1] "integer"
R> dim(mat) <- c(3,4) # assegno le dimensioni righe x colonne
R> mat
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
R> mode(mat); class(mat); typeof(mat)
[1] "numeric"
[1] "matrix"
[1] "integer"
```

Notate come la classe è passata da "integer" a "matrix".

Un altro modo per inserire una matrice è quello di usare il comando specifico matrix() che riceve un vettore di informazioni e le dispone in un oggetto a due dimensioni in base al numero di colonne (parametro ncol=) o di righe (nrow=) indicato. L'assegnazione dei valori è automaticamente per colonna (byrow=F, vedi l'esempio che segue).

```
R> mat <- matrix(c(1:12), ncol=4)
R> mat
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Se voglio inserirli per riga, si può usare il parametro byrow=T.

```
R> mat <- matrix(c(1:12), ncol=4, byrow=T)
R> mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
```

Per accedere ad un qualunque elemento della matrice, devo indicare la riga e la colonna in cui è (`R> mat[2,3]` restituisce 7, mentre `R> mat[3,2]` restituisce 10).

Se voglio una qualunque riga, indico solo il numero di riga (`R> mat[2,]`).

Se voglio una colonna, indico solo il numero di colonna (`R> mat[,3]`)

Se voglio più righe o più colonne, indico le righe e le colonne che mi servono. Se sono consecutive, posso usare la funzione :, se non sono consecutive devo usare la funzione c().

```
R> mat[1:2,]
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
R> mat[c(1,3),]
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    9   10   11   12
```

Dal momento che una matrice non è altro che un vettore, può contenere solo un tipo di valori (numeri, testo o valori logici). Se necessario i diversi valori vengono convertiti nel valore più generico.

```
R> matrix(c('a','b','c','d'),ncol=2)
     [,1] [,2]
[1,] "a"  "c"
[2,] "b"  "d"
R> matrix(c('a','b',5,6),ncol=2)
     [,1] [,2]
[1,] "a"  "5"
[2,] "b"  "6"
```

Così come agli elementi di un vettore posso associare un nome, altrettanto si può fare con le matrici a cui si può associare un nome sia alle righe (rownames=) sia alle colonne (colnames=).

```
R> colnames(mat) = paste('n',1:4,sep='')
R> rownames(mat) <- paste('r',1:3,sep='')
R> mat
   n1 n2 n3 n4
r1  1  4  7 10
r2  2  5  8 11
r3  3  6  9 12
```

2.1.6.1 Calcolo matriciale

Se l'oggetto "matrice" è di tipo numerico (mode o typeof), allora è possibile usare alcuni comandi specifici per il calcolo matriciale.

Addizione e sottrazione fra due matrici (oppure due vettori, un vettore o uno scalare e una matrice) si ottengono con i classici più (+) e meno (-), mentre per la moltiplicazione si usa %*% (percentuale, asterisco, percentuale). Ovviamente le matrici devono essere "conformabili" secondo le regole dell'algebra matriciale. Per cui, definiremo due vettori (uno colonna e uno riga) e due matrici con:

```
R> v.a <- matrix(c(5,12,4,9),ncol=1)
R> v.b <- matrix(c(4,3,5,2),ncol=4) #oppure
R> v.b <- matrix(c(4,3,5,2),nrow=1)
R> m.C <- matrix(c(1,3,4,6),byrow=T,ncol=2)
R> m.D <- matrix(c(3,4,5,-1,7,3,2,6,6,1,6,3),byrow=T,ncol=4)
```

$$\mathbf{a} = \begin{bmatrix} 5 \\ 12 \\ 4 \\ 9 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 4 & 3 & 5 & 2 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 1 & 3 \\ 4 & 6 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 3 & 4 & 5 & -1 \\ 7 & 3 & 2 & 6 \\ 6 & 1 & 6 & 3 \end{bmatrix}$$

e potremo fare le seguenti operazioni matriciali:

$3 + \mathbf{a}$

```
R> 3 + v.a
     [,1]
[1,]    8
[2,]   15
[3,]    7
[4,]   12
```

$3 + \mathbf{b}$

```
R> 3 + v.b
     [,1] [,2] [,3] [,4]
[1,]    7    6    8    5
```

$3 + \mathbf{C}$

```
R> 3 + m.C
     [,1] [,2]
[1,]    4    6
[2,]    7    9
```

$\mathbf{a} * \mathbf{b}$

```
R> v.a %*% v.b
     [,1] [,2] [,3] [,4]
[1,]   20   15   25   10
[2,]   48   36   60   24
[3,]   16   12   20    8
[4,]   36   27   45   18
```

Tabella 2.4: Esempio di dati

Cod	Genere	Eta	Anni
A	M	43	14
B	M	36	6
C	F	56	28
D	F	47	12
E	F	28	1
F	F	37	6
G	M	46	5
H	F	30	2
I	M	28	2
J	F	26	1

$\mathbf{b} * \mathbf{a}$

```
R> v.b %*% v.a
     [,1]
[1,]   94
```

Altre funzioni utilizzabili sono:

- t() per trasporre una matrice/vettore;
- det() per calcolare il determinante;
- solve() per calcolare l'inversa; un'altra procedura (ginv()) per l'inversa è disponibile nel pacchetto MASS e utilizza un diverso algoritmo per il calcolo che comunque produce risultati simili a quelli di solve().

2.1.7 Data frame

Un "data frame" è un insieme di dati con un nome e si può creare solo tramite l'apposito comando data.frame() o caricando dati da file esterni. È un oggetto che può contenere variabili di tipo diverso e sarà, probabilmente, il tipo che useremo più spesso. Infatti possiamo pensare alle colonne del data frame come alle variabili da analizzare e considerare le righe come fossero osservazioni (i casi statistici).[1] Ad es. una matrice di dati come quella della tabella 2.4 può essere creata in R tramite il seguente, comando:

```
R> Dati1 <- data.frame(
+ Cod=c("A","B","C","D","E","F","G","H","I","J"),
+ Genere=c("M","M","F","F","F","F","M","F","M","F"),
+ Eta=c(43,36,56,47,58,37,46,30,28,26),
+ Anni=c(14,6,28,12,1,6,5,2,2,1)
+ )
```

Ma è anche possibile riutilizzare variabili già esistenti:

```
R> Cod <- c("A","B","C","D","E","F","G","H","I","J")
R> Genere <- c("M","M","F","F","F","F","M","F","M","F")
R> Eta <- c(43,36,56,47,58,37,46,30,28,26)
```

[1] Ho deciso di usare il termine "osservazioni" come alternativa a "casi statistici", "unità statistiche", "soggetti", "partecipanti", "rispondenti", "intervistati", ecc.

```
R> Anni <- c(14,6,28,12,1,6,5,2,2,1)
R> Genere
 [1] "M" "M" "F" "F" "F" "F" "M" "F" "M" "F"
R> Eta
 [1] 43 36 56 47 58 37 46 30 28 26
R> Dati1 <- data.frame(Cod,Genere,Eta,Anni)
R> Dati1
   Cod Genere Eta Anni
1    A      M  43   14
2    B      M  36    6
3    C      F  56   28
4    D      F  47   12
5    E      F  58    1
6    F      F  37    6
7    G      M  46    5
8    H      F  30    2
9    I      M  28    2
10   J      F  26    1
```

Nel creare il dataframe in questo secondo modo, possiamo anche cambiare nome alle variabili. Se ad esempio avessi usato il comando

```
R> Dati1 <- data.frame(Soggetti=Cod, Genere, Eta, Anni)
```

il dataframe avrebbe avuto una prima variabile chiamata `Soggetti` e non `Cod`.

Dal momento che i dataframe sono il tipo più usato in ambito statistico rimando ad un apposito paragrafo le modalità di selezione dei dati (cfr. par. 2.2.2).

2.1.8 Liste o elenchi

Un "elenco" o lista (in inglese `list`) è un contenitore di vari tipi di dati, ovvero in un elenco posso inserire scalari, vettori, matrici e anche tutti i tipi di variabili (compresi i dataframe, che sono un tipo particolare di lista).

```
R> Elenco <- c(5, simpsons, Dati1)
R> Elenco
[[1]]
[1] 5

[[2]]
[1] "Homer"

[[3]]
[1] "Marge"

[[4]]
[1] "Bart"

[[5]]
[1] "Lisa"

[[6]]
[1] "Maggie"
```

```
$Cod
 [1] A B C D E F G H I J
Levels: A B C D E F G H I J

$Genere
 [1] M M F F F F M F M F
Levels: F M

$Eta
 [1] 43 36 56 47 58 37 46 30 28 26

$Anni
 [1] 14  6 28 12  1  6  5  2  2  1
```

2.2 Manipolazione di variabili e oggetti

Così come si possono usare le operazioni per manipolare gli scalari, si possono fare operazioni su vettori, su matrici e su dataframe.

Le operazioni che implicano i vettori si attuano su tutti gli elementi del vettore:

```
R> Eta
 [1] 43 36 56 47 58 37 46 30 28 26
R> Eta+2
 [1] 45 38 58 49 60 39 48 32 30 28
R> Eta/2
 [1] 21.5 18.0 28.0 23.5 29.0 18.5 23.0 15.0 14.0 13.0
```

Questo può essere molto utile per fare operazioni complesse. Ad es. possiamo trovare gli scarti dalla media di una variabile (ad es. `Eta` che è 40.7), semplicemente con:

```
R> Eta - 40.7
 [1]   2.3  -4.7  15.3   6.3  17.3  -3.7   5.3 -10.7 -12.7 -14.7
```

Possiamo lavorare anche con 2 o più vettori. Tuttavia se i vettori non sono della stessa lunghezza, il vettore più corto verrà "riciclato" quanto basta per terminare l'operazione sul vettore più lungo:

```
R> A <- c(0,0,0,0,0,0,4,4,4,4,4,4)
R> B <- c(1,2,3)
R> A+B
 [1] 1 2 3 1 2 3 5 6 7 5 6 7
```

Da questo esempio si vede come il vettore `B` è stato usato 4 volte (`0+1` e poi `0+2` e così via) essendo più corto di `A`.

La stessa cosa succede sulle matrici, con gli stessi principi. Bisogna considerare che non si tratta di "matrici matematiche" ma di semplici dati organizzati in righe e colonne. Per le vere matrici matematiche esistono funzioni apposite che utilizzano le regole dell'algebra matriciale (cfr. 2.1.6.1).

2.2.1 Funzioni

In precedenza abbiamo usato comandi particolari ("c()", "matrix()" e ":") dicendo che erano funzioni.

Le funzioni sono comandi che svolgono determinati tipi di operazioni (o manipolazioni) più complesse e che restituiscono il risultato dell'operazione. La forma generale di una funzione è `nome(parametri)` vale a dire che ogni funzione ha un nome e si aspetta dei parametri (i dati) su cui operare. Alcune funzioni possono essere chiamate senza indicare nessun parametro, ma in ogni caso dobbiamo aprire e chiudere le parentesi tonde.

Alcune funzioni hanno una versione abbreviata, ad esempio la funzione seq() ha una versione abbreviata che è : (due punti) oppure q() è la versione abbreviata di quit().

In un esempio precedente abbiamo riportato la media della variabile `Eta` (ovvero 40.7). La media di una serie di numeri è calcolata dalla funzione mean(x), dove `x` è un vettore o un dataframe e può essere utilizzata (a sua volta) come parametro di altre funzioni:

```
R> mean(Eta) # calcolo la media
[1] 40.7
R> Eta-mean(Eta) # scarti dalla media
 [1]   2.3  -4.7  15.3   6.3  17.3  -3.7   5.3 -10.7 -12.7 -14.7
R> (Eta-mean(Eta))^2 # scarti al quadrato
 [1]   5.29  22.09 234.09  39.69 299.29  13.69  28.09 114.49 161.29 216.09
R> (sum((Eta-mean(Eta))^2))/(length(Eta)-1) # varianza
[1] 126.0111
```

In questo esempio abbiamo calcolato la media, poi l'abbiamo usata per calcolare gli scarti dalla media che abbiamo elevato al quadrato, infine abbiamo sommato gli scarti al quadrato e abbiamo diviso per N-1, trovando la varianza.

Le funzioni richiedono di avere dei parametri che rappresentano ciò su cui vogliamo applicare l'operazione. La maggior parte delle funzioni svolge il proprio ruolo su un singolo valore, su un intero vettore oppure su tipi specifici ma, in genere, sono in grado di adattarsi al tipo dei parametri. I parametri possono essere passati alle funzioni per *posizione* o per *nome*. Se vengono passati per *posizione* devono rispettare l'ordine richiesto dalla sintassi del comando, se vengono passati per *nome*, invece, possono essere indicati in un ordine qualsiasi. Nel caso in cui si usino i nomi dei parametri, possono essere abbreviati. Ad es. il comando seq() può accettare 5 diversi parametri (in quest'ordine) from, to, by, length.out, along.with che possono essere abbreviati in f, t, b, l e a, ovvero il numero minimo di caratteri necessari per disambiguare il nome del parametro richiesto.

Esempi di comandi analoghi (per nome o per posizione) sono quindi:

```
R> seq(from=1, to=5, by=1)
[1] 1 2 3 4 5
R> seq(f=1, t=5, b=1)
[1] 1 2 3 4 5
R> seq(1, 5, 1)
[1] 1 2 3 4 5
```

Nella tabella 2.5 sono indicate alcune funzioni che non saranno dettagliate nei prossimi paragrafi; all'interno della tabella (e in quelle successive), la lettera `x` indica un numero, mentre `v` indica un vettore.

Tabella 2.5: Alcune delle funzioni utilizzabili in R

Operazione	*Funzione da usare*	*Esempio*
Somma	sum(v)	sum(Eta)
Numero di elementi	length(v)	lenght(Eta)
Riordino	sort(v)	sort(balene)
Ranghizzazione	rank(v)	rank(Eta)
Differenza	diff(v)	diff(Eta)
Campionamento	sample(v,size)	sample(1:100, 75)

Per sapere come agiscono le varie funzioni bisogna ricorrere all'aiuto disponibile in R (cfr. 1.5).

2.2.1.1 Funzioni di varia utilità

Alcune funzioni non hanno uno scopo matematico, ma semplicemente di utilità.

2.2.1.1.1 Funzione `seq()` Una prima funzione che abbiamo già visto è quella per generare sequenze di numeri. È una funzione che esiste in due forme, la prima è rappresentata dai due punti ed è un'abbreviazione della forma completa `seq()` la cui sintassi (ottenuta con `R> ?seq`) è:

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)
```

I parametri `from` (inizio), `to` (fine) e `by` (passo, ad es. `by=2` conta per 2) sono abbastanza intuitivi; `length.out` (lunghezza) si usa in alternativa a `by` e indica il numero di valori da generare: R utilizza la formula indicata in sintassi per calcolare il valore di `by` da usare.

```
R> seq(1, 10, length=5)
[1]  1.00  3.25  5.50  7.75 10.00
R> seq(0, 10, length=5)
[1]  0.0  2.5  5.0  7.5 10.0
```

Il parametro `along.with=` significa "lungo quanto" una variabile che viene indicata e si usa generalmente da solo; si ottiene una sequenza che parte da 1 e arriva fino alla lunghezza della variabile. Se usato con gli altri parametri produce risultati diversi:

```
R> seq(along=Eta)
 [1]  1  2  3  4  5  6  7  8  9 10
R> seq(Eta)
 [1]  1  2  3  4  5  6  7  8  9 10
R> seq(10,along=Eta)
 [1] 10 11 12 13 14 15 16 17 18 19
R> seq(10,b=2,a=Eta)
 [1] 10 12 14 16 18 20 22 24 26 28
```

Nel primo caso, la variabile `Eta` contiene 10 valori, viene quindi generato un vettore di 10 elementi a partire da 1; nei comandi successivi si cambia il punto di inizio e il passo. Un risultato analogo si ottiene anche con `R> seq(Eta)` oppure con `R> 1:10`.

Il comando `1:5` corrisponde a `seq(from=1, to=5, by=1)` e può essere indicato anche semplicemente come `seq(5)`.

2.2.1.1.2 Funzione `rep()` Un'altra funzione di utilità è `rep()` che serve per ripetere più volte la stessa cosa. È una funzione piuttosto elastica che permette di produrre molti risultati diversi. L'uso più semplice implica di indicare come primo parametro un "qualcosa" da ripetere `x` volte (secondo parametro `times=` che viene sottinteso).

```
R> rep(5,times=7) # 7 volte 5
[1] 5 5 5 5 5 5 5
R> rep(5,7) # idem
[1] 5 5 5 5 5 5 5
R> rep(1:4, 2) # 2 volte i numeri da 1 a 4
[1] 1 2 3 4 1 2 3 4
R> rep(c(1,5),4) # 4 volte i numeri 1 e 5
[1] 1 5 1 5 1 5 1 5
R> rep(c("casa", "mia"), 2) # 2 volte due stringhe
[1] "casa" "mia"  "casa" "mia"
```

Nel caso in cui vogliamo ripetere più elementi, la ripetizione può essere applicata ai singoli valori anziché all'intero, usando il parametro `each`.

```
R> rep(1:4, each=2) # ripete 2 volte ciascun numero
[1] 1 1 2 2 3 3 4 4
```

Sono possibili anche modalità più complesse, illustrate dagli esempi che seguono:

```
R> rep(1:4, c(2,1,2,1)) # il primo 2 volte, il secondo 1...
[1] 1 1 2 3 3 4
R> rep(1:4, each = 2, len = 10) # massimo 10 elementi
 [1] 1 1 2 2 3 3 4 4 1 1
R> rep(1:4, each = 2, times = 3) # ripetere 3 volte
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

2.2.1.1.3 Funzione `c()` Una seconda funzione già usata, è quella di concatenazione, che serve per creare vettori o liste. La sua sintassi è:

```
c(..., recursive=FALSE)
```

I 3 puntini stanno ad indicare che possiamo indicare più elementi da concatenare. Se gli elementi indicati non sono tutti dello stesso tipo, vengono automaticamente convertiti al tipo più generale. Ad es.

```
R> c(5, "casa", T)
[1] "5"    "casa" "TRUE"
```

convertirà tutto in stringhe (testo) che è il tipo più generico dei tre, mentre

```
R> c(5, T, F)
[1]  5 1 0
```

convertirà tutto in numeri, perché "vero" è generalmente rappresentato dal numero 1 e "falso" dal valore zero, ma non è possibile rappresentare il numero 5 con vero o falso.

Ovviamente, dentro a `c()` possiamo concatenare anche risultati prodotti da `seq()` e da `rep()`.

```
R> c(15,seq(3),10:13,rep(2,3))
 [1] 15  1  2  3  10 11 12 13  2  2  2
```

2.2.1.1.4 Funzioni `cbind()` e `rbind()` Queste due funzioni servono per combinare fra loro dei vettori (o variabili di dataframe), in senso verticale (`cbind()`, come colonne) o in senso orizzontale (`rbind()`, come righe).

```
R> cbind(1:3,6:8,10:12)
     [,1] [,2] [,3]
[1,]    1    6   10
[2,]    2    7   11
[3,]    3    8   12
R> c1=1:3; c2=6:8; c3=10:12
R> cbind(c1,c2,c3)
     c1 c2 c3
[1,]  1  6 10
[2,]  2  7 11
[3,]  3  8 12
R> rbind(c1,c2,c3)
   [,1] [,2] [,3]
c1    1    2    3
c2    6    7    8
c3   10   11   12
```

2.2.1.1.5 Funzione `with()` Vedremo fra un po' che la gestione delle variabili incluse in un dataframe non è sempre agevole. La funzione `with()` permette di applicare un determinato comando all'interno di un dataframe in modo più semplice, accedendo direttamente alle sue variabili interne. La sua utilità sarà più chiara con l'uso. Per ora ecco un esempio:

```
R> with(Dati1, mean(Eta)) # al posto di mean(Dati1$Eta)
[1] 40.7
```

2.2.1.1.6 Funzione `paste()` La funzione `paste()` serve per incollare assieme pezzi diversi e formare qualcosa di nuovo. Il risultato sarà un testo (stringa). Ad es.

```
R> n=15.34
R> paste("il risultato è",n,sep=' ')
[1] "il risultato è 15.34"
R> paste('q',c(1:5),sep='')
[1] "q1" "q2" "q3" "q4" "q5"
R> paste(1:5)
```

Tabella 2.6: Alcune funzioni aritmetiche

Operazione	*Funzione da usare*	*Esempio*
Radice quadrata	sqrt(x)	sqrt(9)
Seno	sin(x)	sin(5)
Coseno	cos(x)	cos(1)
Tangente	tan(x)	tan(3)
Arcoseno	asin(x)	asin(.1)
Arcocoseno	acos(x)	acos(.03)
Arcotangente	atan(x)	atan(3)
e^x	exp(x)	exp(1)
Logaritmo (base e)	log(x)	log(300)
Logaritmo (base 10)	log(x, 10)	log(5,10)
	log10(x)	log10(5)

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

A `paste()` possiamo passare tutti gli oggetti da incollare assieme (possono essere di tipo diverso) ed infine con il parametro `sep=` indichiamo come devono essere separati. Nel primo esempio qui sopra, abbiamo incollato assieme (separati da uno spazio) un testo e un numero per formare una stringa. Nel secondo esempio abbiamo incollato una lettera e una sequenza di numeri per formare un vettore che contiene 5 nomi di variabili. Il parametro `sep=' '` (cioè separati da uno spazio) può essere omesso perché è il default del comando. Nel terzo esempio abbiamo creato un vettore di numeri in formato testo.

2.2.1.2 Funzioni aritmetiche

Oltre alle quattro operazioni, sono disponibili una serie di funzioni aritmetiche (alcune sono indicate in Tab. 2.6) che possono essere usate sia su valori singoli, multipli o intere variabili e possono a loro volta essere parametri di altre funzioni. Quelle indicate in tabella operano tutte nel medesimo modo:

```
R> sqrt(125) # radice quadrata di un numero
[1] 11.18034
R> sqrt(1:5) # dei numeri da 1 a 5
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
R> sqrt(c(3,6,9)) # di 3 numeri specifici
[1] 1.732051 2.449490 3.000000
R> B # cosa c'è in B
[1] 1 2 3
R> sqrt(B) # radice quadrata di B
[1] 1.000000 1.414214 1.732051
R> sqrt(B)^2 # quadrato della radice quadrata
[1] 1 2 3
R> rep(sqrt(B),2) # ripetizione di radici quadrate
[1] 1.000000 1.414214 1.732051 1.000000 1.414214 1.732051
```

Tabella 2.7: Alcune funzioni per la statistica

Operazione	*Funzione da usare*	*Esempio*
Minimo	min(v,na.rm=F)	min(balene)
Massimo	max(v,na.rm=F)	max(balene)
Somma	sum(v,na.rm=F)	sum(balene)
Somma cumulata	cumsum(v)	cumsum(balene)
Prodotto	prod(v,na.rm=F)	prod(balene)
Prodotto cumulato	cumprod(v)	cumprod(balene)

2.2.1.3 Funzioni statistiche

Le principali funzioni statistiche verranno prese in considerazione nell'apposito paragrafo del capitolo **??**. Qui prenderò in considerazioni le funzioni base, ovvero quelle che saranno poi utilizzate per le statistiche.

Tra le funzioni di base, possiamo citare `min()` e `max()` che trovano il valore minimo e il valore massimo di un vettore o di una variabile di dataframe. Se nel vettore esiste un valore mancante, viene restituito `NA` a meno di non specificare il parametro `na.rm=T` (cfr. 2.2.7).

```
R> min(1:10,NA) # valore minimo
[1] 1
R> max(1:10) # valore massimo
[1] 10
R> min(c(1:10,NA),na.rm=T)
[1] 1
```

Fra le altre abbiamo funzioni che calcolano la somma degli elementi di un vettore (`sum`, che corrisponde a $\sum$) oppure creano un vettore con la somma cumulata (`cumsum`). Anche in questo caso si può usare il parametro `na.rm=T` (ma solo per `sum`). La somma cumulata è ottenuta sommando tutti i valori precedenti all'attuale (1, 3=2+1, 6=3+2+1..., cfr. Tab. 2.8).

```
R> sum(1:10) # somma
[1] 55
R> sum(c(1:10, NA), na.rm=T) # somma
[1] 55
R> cumsum(1:10) # valori cumulati
[1]  1  3  6 10 15 21 28 36 45 55
```

Esiste anche una funzione che somma i prodotti (`prod`, corrispondente a $\prod$) e la sua versione cumulata (`cumprod`, per il suo funzionamento vedi Tab. 2.8). Solo `prod` gestisce i mancanti.

```
R> prod(1:10)
[1] 3628800
R> cumprod(1:10)
 [1]       1       2       6      24     120     720    5040   40320  362880
[10] 3628800
R> prod(c(1:10,NA),na.rm=T)
[1] 3628800
```

2.2.2 Selezionare variabili

Per selezionare i dati di un oggetto `dataframe` dobbiamo considerarlo come se fosse una tabella bidimensionale formata da righe e da colonne, in cui le colonne corrispondono alle variabili. Per riferirci ad un singolo dato, dobbiamo ricordare la sintassi tipica delle matrici:

```
nome[riga,colonna]
```

Considerando che le colonne di un dataframe hanno (generalmente) un nome, si può far riferimento ad una singola "colonna" del dataframe tramite la notazione "`dataframe \$nomevariabile`", quindi per riferirci alla variabile `Genere` dentro al dataframe `Dati1`, useremo `Dati1$Genere`.

Ricordiamo che `names(dataframe)` mostra i nomi associati:

```
R> names(Dati1)
[1] "Cod"    "Genere" "Eta"    "Anni"
```

In alternativa possiamo usare il comando `attach(dataframe)` per esportare tutte le variabili del frame come variabili separate. Se usassi il comando

```
R> attach(Dati1)
```

avrei la possibilità di usare le parole `Genere`, `Anni` ed `Eta` senza premettere loro `Dati1`. Eventuali variabili con lo stesso nome già presenti potrebbero creare problemi.

Quando ho terminato di usarli, posso annullare l'effetto di `attach` con il comando `detach()`:

```
R> detach(Dati1)
```

Se voglio sapere cosa contiene un dataframe posso usare il commando `head()` che visualizza i nomi delle variabile e le prime righe dei dati.

```
R> head(Dati1)
  Cod Genere Eta Anni
1   A      M  43   14
2   B      M  36    6
3   C      F  56   28
4   D      F  47   12
5   E      F  58    1
```

Tabella 2.8: Procedimento delle cumulate

	cumsum		cumprod	
1	1=	1	1=	1
2	2+1=	3	2 x 1=	2
3	3+3=	6	3 x 2=	6
4	4+6=	10	4 x 6=	24
5	5+10=	15	5 x 24=	120
6	6+15=	21	6 x 120=	720
7	7+21=	28	7 x 720=	5040
8	8+28=	36	8 x 5040=	40320
9	9+36=	45	9 x 40320=	362880
10	10+45=	55	10 x 362880=	3628800

```
6   F      F  37    6
R> head(Dati1,n=3) # solo i primi 3 casi
  Cod Genere Eta Anni
1   A      M  43   14
2   B      M  36    6
3   C      F  56   28
```

Esiste anche il comando opposto (`tail()`) che mostra la parte finale.

Un altro modo per riferirsi alle variabili del dataframe è tramite la loro posizione.

Per sapere quali variabili sono incluse in un dataframe, ma con associati i numeri d'ordine, posso usare il comando:

```
R> data.frame(names(Dati1))
  names.Dati1.
1          Cod
2       Genere
3          Eta
4         Anni
```

Nel caso del dataframe `Dati1`, la variabile `Cod` è in posizione 1, `Genere` in posizione 2 e così via e possiamo usare la notazione `frame[,posizione]`, per cui `Dati1[,3]` restituirà il contenuto della variabile `Eta`. In generale, i dataframe si considerano come delle matrici speciali, per cui il primo indice si riferisce alle righe e il secondo alle colonne. Siccome però i dataframe vengono spesso usati per dati statistici e si è maggiormente interessati alle variabili, è possibile abbreviare il comando in `Dati1[3]` ovvero, se non si usa la virgola, R assume che si voglia far riferimento ad una variabile dentro al dataframe.

Se voglio riferirmi a più di una variabile del dataframe (ma non a tutte) devo accedervi come se fossero colonne di una matrice, quindi o tramite la loro posizione numerica oppure con i loro nomi.

```
R> head(Dati1[,2:3],3) # solo i primi 3 casi
  Genere Eta
1      M  43
2      M  36
3      F  56
R> tail(Dati1[,c("Eta","Anni")],3)
   Eta Anni
8   30    2
9   28    2
10  26    1
```

Siccome la maggior parte dei dataframe contiene dati di ricerca (variabili per osservazioni), la prima colonna è quasi sempre il codice del caso statistico, su cui, in genere, non vogliamo fare calcoli. In questo caso, possiamo fare riferimento a tutte le variabili del dataframe (escluso la prima) tramite:

```
R> head(Dati1[,-1],3)
  Genere Eta Anni
1      M  43   14
2      M  36    6
```

```
3      F  56   28
```

L'uso di un numero negativo indica a R di usare tutte le variabili "esclusa" quella o quelle indicate. Per cui i comandi `Dati1[-c(1,3)]` equivale a `Dati1[c(2,4)]`.

Ovviamente se un dataframe è composto da diverse sottoscale, possiamo salvare la selezione richiesta in una variabile e riusarla per la selezione:

```
R> x=-c(1,3)
R> head(Dati1[x],4)
   Genere Anni
1       M   14
2       M    6
3       F   28
4       F   12
```

È anche possibile selezionare le variabili in modo "logico". Ad es., `Dati1[c(T,T,F,F)]` selezionerà le prima due variabili:

```
R> head(Dati1[c(T,T,F,F)],3)
  Cod Genere
1   A      M
2   B      M
3   C      F
```

Infine è possibile usare il comando `subset()` per selezionare variabili per nome. Ecco un esempio:

```
R> subset(Dati1,selectl=c(Genere, Anni)
R> subset(Dati1,sel=2:3) # dataframe e posizione di variabili
R> subset(Dati1,sel=Eta:Anni) # dataframe e nomi di variabile
```

Il comando `subset()` funziona con tutti i tipi di variabili. Quando si usa con i dataframe, si indica un insieme di variabili (indicate per nome o per posizione); se le variabili da selezionare sono consecutive, si può usare il `:` (due punti) anche sui nomi di variabili.

2.2.3 Selezionare osservazioni

Con lo stesso principio usato per le variabili, possiamo selezionare ossservazioni (ovvero righe del dataframe). Ad es. il comando `R> Dati1[3:7,]` seleziona le righe da 3 a 7, mentre `R> Dati1[c(1,4,8),]` seleziona le righe 1, 4 e 8. È obbligatorio usare la virgola finale per indicare che ci stiamo riferendo alle righe del dataframe.

```
R> Dati1[3:5,]
  Cod Genere Eta Anni
3   C      F  56   28
4   D      F  47   12
5   E      F  58    1
R> Dati1[c(1,4,8),]
  Cod Genere Eta Anni
1   A      M  43   14
4   D      F  47   12
```

```
8   H      F  30    2
```

È possibile selezionare casi in base alle caratteristiche dei dati stessi, usando un vettore di valori logici, usando gli operatori della Tab. 1.2. Se vogliamo selezionare (all'interno di `Dati1`) solo le persone con un'età superiore a 40, possiamo porre la condizione:

```
R> Dati1$Eta>=40
 [1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE
R> Dati1[Dati1$Eta>=40,]
  Cod Genere Eta Anni Mesi
1   A      M  43   14  168
3   C      F  56   28  336
4   D      F  47   12  144
5   E      F  58    1   12
7   G      M  46    5   60
```

notate che: `Dati1$Eta>=40` crea un vettore composto da vero e falso che indica quali osservazioni vanno stampate; che dobbiamo obbligatoriamente usare la notazione `Dati1$Eta`, altrimenti la variabile `Eta` da sola risulta sconosciuta; che la condizione va indicata nella parte corrispondente alle righe e il tutto va posto entro parentesi quadre.

Se vogliamo selezionare solo i maschi:

```
R> Dati1[Dati1$Genere=="M",]
  Cod Genere Eta Anni Mesi
1   A      M  43   14  168
2   B      M  36    6   72
7   G      M  46    5   60
9   I      M  28    2   24
```

Ovviamente possiamo anche usare il comando `subset()`, inserendo una condizione logica:

```
R> subset(Dati1,Genere=="M") # dataframe e condizione
R> subset(Dati1,Genere=="F",sel=c(Genere, Anni)
```

2.2.4 Selezionare sottoinsiemi di dati

Possiamo ovviamente anche usare contemporaneamente le modalità per selezionare righe e colonne, selezionando quindi un sottoinsieme di dati; ecco alcuni esempi:

```
R> Dati1[3:5,c("Eta","Anni")]
  Eta Anni
3  56   28
4  47   12
5  58    1
R> Dati1[(Dati1$Genere=="M"),c(2:4)]
  Genere Eta Anni
1      M  43   14
2      M  36    6
7      M  46    5
9      M  28    2
```

Salvando il risultato (in un'altra variabile) è possibile creare un sottocampione di soggetti e/o un sottoinsieme di dati da usare per analisi particolari.

2.2.5 Ordinare osservazioni

Alcune volte (ma raramente) serve di ordinare il file dati in base ad una o più variabili. Per questo motivo abbiamo i comandi `sort()` e `order()` che permettono di riordinare un vettore.

Il comando `sort()` riordina i dati di un vettore o di una variabile di dataframe, trattandola come tale ovvero senza tener traccia della posizione iniziale del valore:

```
R> sort(Dati1$Genere)
 [1] F F F F F F M M M M
Levels: F M
R> sort(Dati1$Anni)
 [1]  1  1  2  2  5  6  6 12 14 28
```

in questo esempio abbiamo i valori riordinati, ma non abbiamo traccia della loro posizione originale

Al contrario, il comando `order()` permette di lavorare anche con i dataframe, di indicare più variabili da usare per il riordino e restituisce il numero progressivo dell'osservazione corrispondente ai valori riordinati, per cui è possibile usarlo per riordinare un dataframe in base ad una o più variabili:

```
R> order(Dati1$Genere)
 [1]  3  4  5  6  8 10  1  2  7  9
R> Dati1[order(Dati1$Genere),]
   Cod Genere Eta Anni
3    C      F  56   28
4    D      F  47   12
5    E      F  58    1
6    F      F  37    6
8    H      F  30    2
10   J      F  26    1
1    A      M  43   14
2    B      M  36    6
7    G      M  46    5
9    I      M  28    2
```

in questo esempio abbiamo non il contenuto della variabile `Genere`, ma il numero progressivo delle osservazioni riordinate.

Per ordinare su più variabili, basta specificarle tutte nell'ordine desiderato; ad es. per ordinare in base al `Genere` e, al suo intero, in base agli `Anni`, basta usare:

```
R> Dati1[order(Dati1$Genere,Dati1$Anni),]
   Cod Genere Eta Anni
5    E      F  58    1
10   J      F  26    1
8    H      F  30    2
6    F      F  37    6
4    D      F  47   12
3    C      F  56   28
9    I      M  28    2
7    G      M  46    5
```

```
2    B      M  36    6
1    A      M  43   14
```

Per riordinare in ordine inverso, bisogna usare il parametro `decreasing = T`, che però funziona contemporaneamente su tutte le variabili di riordino. Esiste anche un parametro `na.last` che è impostato a `TRUE` e che posiziona eventuali valori mancanti alla fine del file; impostandolo a `FALSE` verranno posti all'inizio.

2.2.6 Mischiare file dati

Un'altra situazione che può capitare, lavorando con dati statistici, è di avere file diversi che contengono informazioni diverse sugli stessi rispondenti e di voler creare un nuovo file che mischia le informazioni del primo e del secondo file in base ad una o più variabili. Questo lavoro si ottiene con il comando `merge()`.

Immaginiamo di avere due database (`A1` e `Af`) che contengono variabili diverse provenienti dallo stesso questionario somministrato agli stessi soggetti. `A1` contiene i dati anagrafici e `Af` alcune variabili particolari.

```
R> names(A1)
[1] "COD"     "Genere"  "Eta"     "TitStud" "STCiv"
R> names(Af)
[1] "COD"      "QUEST"    "INTRINS"  "ESTRPERS" "Authori"  "Malleab"  "world"
[8] "Liht"
```

In entrambi i file la variabile `COD` serve ad identificare l'osservazione (variabile criterio). Se vogliamo creare un nuovo dataframe contenente tutte le informazioni associate allo stesso `COD`ice osservazione, usiamo il comando `merge()` in questo modo:

```
R> Acomb <- merge(A1,Af)
R> names(Acomb)
 [1] "COD"      "Genere"   "Eta"      "TitStud"  "STCiv"    "QUEST"
 [7] "INTRINS"  "ESTRPERS" "Authori"  "Malleab"  "world"    "Liht"
```

Dal momento che in entrambi i file esiste una variabile con lo stesso nome, automaticamente questa variabile viene utilizzata per identificare le osservazioni da fondere fra loro. Non è necessario riordinare i due database in modo crescente sulle variabili criterio, perché c'è un parametro `sort=TRUE` impostato automaticamente. Se al posto di una variabile comune, i due dataframe utilizzassero i nomi di riga (impostabili con `rownames()`) i due file verrebbero fusi usando questi nomi. Se invece i file hanno una variabile che contiene i valori necessari per appaiare le osservazioni, ma queste variabili si chiamassero con nomi diversi, si possono usare i parametri `by.x=` e `by.y=` indicando la variabile da usare per l'appaiamento (ad es. `by.x='Sog', by.y='cod'`).

Il comando `merge()` è molto potente, infatti fra gli altri parametri ci sono anche `all.x=` e `all.y=` che sono generalmente impostati a `FALSE` e indicano che le osservazioni presenti in uno dei due dataframe ma non nell'altro devono essere scartate. Impostandoli a `TRUE` (anche tramite `all=T` che li imposta entrambi contemporaneamente) le osservazioni presenti in un dataframe verrebbero incluse ugualmente e i valori non disponibili verrebbero impostati a `NA`.

2.2.7 Gestione dei valori mancanti

Spesso nella raccolta dei dati ci si imbatte nei "valori mancanti", quando cioè per svariati motivi non si dispone di un valore per un determinato caso statistico in una certa variabile.

Generalmente ci sono svariati modi per affrontarli. Una prima possibilità è quella di eliminare i casi statistici contenenti dei mancanti e la seconda possibilità è quella di sostituire i valori mancanti con valori stimati (questa seconda possibilità la affronteremo nel capitolo [da FINIRE]..

Per eliminare i casi che presentano valori mancanti ci sono due possibili opzioni:

- la prima è chiamata *listwise* e consiste nell'eliminare tutti i casi che presentano anche un solo valore mancante, indipendentemente da quali variabili si voglia utilizzare; in questo modo si ottiene un file dati con solo i casi che hanno fornito una risposta a **tutte** le variabili. È possibile però che, in questo modo, i dati si riducano drasticamente se i mancanti sono sparpagliati su molti casi e su molte variabili;
- la seconda è chiamata *pairwise* e consiste nell'eliminare un caso statistico che presenta un mancante in una variabile solo quando è richiesto un calcolo che implica quella variabile. In questo caso si mantengono il maggior numero possibile di casi, ma la numerosità può cambiare da statistica a statistica.

In R, i valori mancanti sono indicati con `NA` (*not available*) e molte procedure prevedono una gestione dei valori mancanti tramite il parametro `na.rm=T` (rimuovi NA) che generalmente è impostato a `FALSE` oppure il parametro `na.action`. Ad es. il comando che calcola la media ha la possibilità di usare questo parametro:

```
R> x <- sample(c(1:10,NA),15,rep=T) # vettore con mancanti
R> x
 [1]  1  6  1  2  6  8 10  6  8 NA  1 NA  6  8 NA
R> mean(x) # per default na.rm=F
[1] NA
R> mean(x, na.rm=T)
[1] 5.25
```

Altre procedure contengono una propria gestione dei mancanti. Ad es. il comando per le correlazioni (`cor`) dispone di un parametro `use=` che può assumere diversi valori tra cui "`complete.obs`" (solo casi completi ovvero *listwise*) e "pairwise.complete.obs" (casi completi nelle variabili utilizzate ovvero *pairwise*).

In altri casi, invece la gestione dei mancanti non è prevista e bisogna eliminare tutti i soggetti che abbiano un valore mancante (metodo *listwise*). Questo si può fare con un comando apposito (`na.omit`):

```
R> load("Quest.rda")
R> QuestNA = na.omit(Quest)
```

`na.omit()` può essere usata anche all'interno di un altro comando, ad es:

```
R> cov(Quest[5:7])
       Quest4 Quest5 Quest6
Quest4  4.144     NA  0.417
Quest5     NA     NA     NA
Quest6  0.417     NA  6.769
R> cov(na.omit(Quest[5:7]))
       Quest4 Quest5 Quest6
Quest4  4.136  0.525  0.402
Quest5  0.525  6.552 -2.601
Quest6  0.402 -2.601  6.769
```

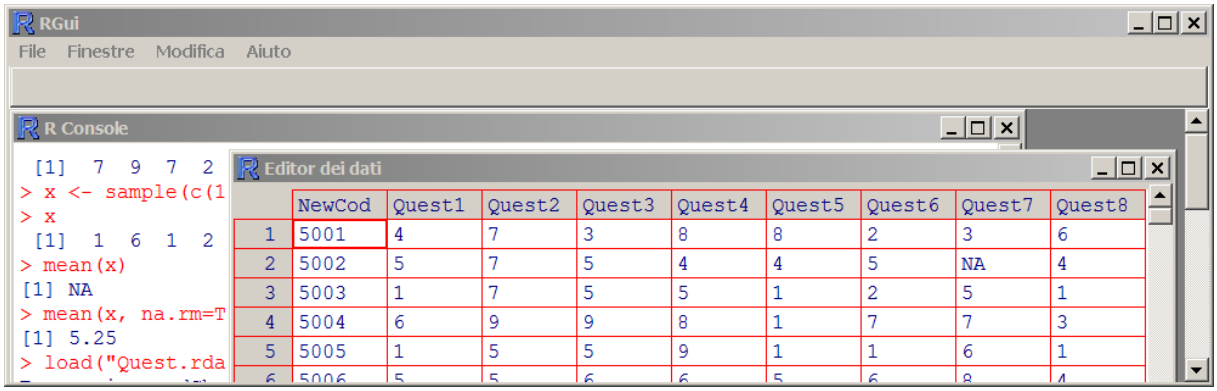

Figura 2.1: Finestra di modifica

che calcola la matrice di varianze/covarianze fra alcune variabili di `Quest`. Le variabili contengono dei valori mancanti e questo impedisce il calcolo. Si possono quindi eliminare tutti i casi mancanti.

2.2.8 Modificare i dati

2.2.8.1 I comandi edit() e fix()

R dispone al proprio interno di una procedura per la modifica dei dati a cui si può accedere tramite due comandi che si differenziano solo per il modo con cui devono essere chiamati. Ipotizzando di avere un file dati chiamato `Quest`, ecco il modo di modificarlo):

```
R> Quest <- edit(Quest)edit()
R> fix(Quest)fix()
```

in entrambi i casi si apre un finestra di editing (Figura 2.1), simile a una tabella e si possono fare le dovute modifiche. Le possibilità che offre non sono eccezionali, quindi è preferibile usare altre modalità.

2.2.8.2 Rinominare variabili

I nomi delle variabili si possono visualizzare con i comandi `names()` o `variable.names()` e con gli stessi comandi si possono modificare.

```
R> names(Dati1)
[1] "Cod"    "Genere" "Eta"    "Anni"
R> names(Dati1)[1] <- 'Sog'
R> variable.names(Dati1)
[1] "Sog"    "Genere" "Eta"    "Anni"
```

Per rinominare una variabile, dobbiamo usare il comando `names()` indicando la posizione (o le posizioni) della variabile da rinominare:

```
R> names(Dati1)[1] <- "Cod"
```

Per rinominare più variabili, usiamo la funzione `c()` per concatenare sia le posizioni sia i nuovi nomi:

```
R> names(Dati1)[c(1,2)] <- c("Sog", "Sex")
```

Se non ricordiamo le posizioni, basta usare:

```
R> data.frame(names(Dati1))
  names.Dati1.
1          Sog
2       Genere
3          Eta
4         Anni
```

In R Commander usiamo `Dati | Gestione variabili del set di dati attivo | Rinomina le variabili...`.

2.2.8.3 Calcolare nuove variabili

Per ciò che ci riguarda, "calcolare nuove variabili" significa sostanzialmente aggiungere una nuova variabile ad un dataframe. La cosa è molto semplice perché basta aggiungere un nuovo nome di variabile al dataframe (usando le varie modalità) e specificando il contenuto da associare.

Ad es. per aggiungere una costante `Gruppo` (pari a 1) al dataframe `Dati1` è sufficiente usare il comando:

```
R> Dati1$Gruppo <- 1
R> head(Dati1,3)
  Cod Genere Eta Anni Gruppo
1   A      M  43   14      1
2   B      M  36    6      1
3   C      F  56   28      1
```

Volendo trasformare in mesi l'età (attualmente espressa in anni in `Dati$Anni`):

```
R> Dati1[,"Mesi"] <- Dati1$Anni*12
R> head(Dati1,3)
  Cod Genere Eta Anni Gruppo Mesi
1   A      M  43   14      1  168
2   B      M  36    6      1   72
3   C      F  56   28      1  336
```

Un'altra possibilità è usando il comando `transform()` sempre disponibile in R. Questo comando richiede un dataframe e una regola per costruire una nuova variabile. La trasformazione in mesi dell'esercizio precedente, se svolto con `transform()` risulterebbe:

```
R> Dati1 <- transform(Dati1, Mesi=Anni*12)
R> head(Dati1,3)
  Cod Genere Eta Anni Gruppo Mesi
1   A      M  43   14      1  168
```

```
2   B      M  36    6      1   72
3   C      F  56   28      1  336
```

In genere, i comandi disponibili nelle sezioni 2.2.8 (*Modificare variabili*) e 2.2.8.6 (*Ricodificare variabili*) possono anche creare nuove variabili.

2.2.8.4 Modificare variabili

Ci sono due modi di intendere la modifica di una variabile: 1) tramite una regola di calcolo (ad es. una trasformazione in logaritmo); 2) tramite una regola logica (ad es. ricodificando i valori).

In questo paragrafo mi occupo del primo caso, mentre la seconda modalità è spiegata nel paragrafo di ricodifica (2.2.8.6).

Per modificare il contenuto di una variabile è sufficiente usare il suo nome:

```
R> Dati1$Gruppo <- Dati1$Gruppo * -1 # inverto il segno
R> head(Dati1,3)
  Cod Genere Eta Anni Gruppo Mesi
1   A      M  43   14     -1  168
2   B      M  36    6     -1   72
3   C      F  56   28     -1  336
```

Un altro modo per modificare variabili è tramite la funzione `transform()` ricordandosi che il risultato va salvato (nello stesso dataframe o in uno nuovo):

```
R> transform(Dati1, Gruppo=-Gruppo) -> Dati2
R> head(Dati2,3)
  Cod Genere Eta Anni Gruppo Mesi
1   A      M  43   14      1  168
2   B      M  36    6      1   72
3   C      F  56   28      1  336
```

Il vantaggio di usare `transform()` è che si possono modificare più variabili contemporaneamente.

```
> head(transform(L, L1=-L1, L3=-L3), 3)
   L1 L2 L3 L4 L5
1  -6  7 -6  1  1
18 -7  7 -7  1  1
59 -2  1 -1  5  3
```

Un altro tipo di modifica è quella relativa a variabili numeriche che vorremmo fossero fattori. Ricordiamo che, per R, le variabili di tipo fattore corrispondono al livello di misura nominale e ordinale. Tuttavia se carichiamo i dati da un file Excel o da un file CSV, i numeri vengono letti e trattati come numeri. Alcune volte vorremmo che queste variabili potessero essere usate come fattori (in alcuni casi sono le procedure che lo richiedono, altre volte è per il diverso tipo di "trattamento" a cui sono sottoposti i fattori rispetto alle variabili numeriche). Possiamo usare il comando `factor()` per creare una nuova variabile (o modificare l'attuale) che sia invece un fattore.

2.2.8.5 Cancellare variabili

Per cancellare una variabile è sufficiente dichiararla `NULL` usando i vari modi di selezionarla.

```
R> Dati1$Mesi <- NULL
R> Dati1[6] <- NULL # alternativa
R> Dati1["Mesi"] <- NULL # alternativa
```

2.2.8.6 Ricodificare variabili

La ricodifica di una variabile non è sempre facile con R. Alcune cose sono abbastanza facili, altre assolutamente no.

La soluzione migliore è usare il comando `recode` disponibile nel pacchetto (non precaricato) `car`, la cui sintassi è:

```
recode(var, recodes, as.factor.result, levels)
```

Il primo parametro è la variabile da ricodificare, il secondo è un testo che descrive la ricodifica che si desidera fare. Il terzo e quarto parametro si applicano se la variabile da ricodificare è un fattore: as.factor.result è impostato a `TRUE` se la variabile è un fattore, altrimenti a `FALSE`; nel caso la variabile sia un fattore, con levels si possono specificare i livelli e il loro ordine.

Ad es. se ho una variabile di un dataframe in cui il valore -1 indica delle risposte mancanti, posso facilmente ricodificarle in `NA` con:

```
R> library(car)
R> Nuova=recode(Dati$Variabile, "-1=NA")
```

La ricodifica di una variabile `x` che contiene valori da 1 a 10, che si vuole modificare come 1=1-3, 2=4-7, 3=8-10, può essere ottenuta nei seguenti modi:

```
R> x=c(1:10)
R> x1=recode(x, "1:3=1; c(4,5,6,7)=2; 8:10=3") # oppure
R> x2=recode(x, "lo:3=1; 4:7=2; 8:hi=3") # oppure
R> x3=recode(x, "lo:3=1; 4:7=2; else=3")
```

Dopo aver indicato la variabile da ricodificare, si indicano le modifiche da varie scritte entro un testo (quindi fra virgolette). **1:3=1** significa che i valori da 1 a 3 devono diventare 1 e così via. Prima dell'uguale si indicano i vecchi valori e dopo l'uguale il valore da assegnare. lo e hi indicano rispettivamente il valore minimo e il valore massimo di quella variabile. Un'altro termine che si può usare è else che corrisponde a "tutti gli altri valori" (e include anche i NA).

È possibile trasformare numeri in testo (da includere fra virgolette semplici) e viceversa:

```
R> x=c(1:10)
R> t=recode(x, "1:3='A'; 4:7='B'; 8:10='C'")
R> t
 [1] "A" "A" "A" "B" "B" "B" "B" "C" "C" "C"
R> x2=recode(t, "'A'=1; 'B'=2; 'C'=3")
R> x2
 [1] 1 1 1 2 2 2 2 3 3 3
```

Un problema di questo comando è che funziona una variabile alla volta. Per esempio se voglio ricodificare un sottoinsieme di variabili, devo ripetere il comando per ciascuna:

```
R> names(L)
R> L$L1 <- recode(L$L1, "1:3=1;4:7=2;8:10=3;NA=4")
R> L$L2 <- recode(L$L2, "1:3=1;4:7=2;8:10=3;NA=4")
R> L$L3 <- recode(L$L3, "1:3=1;4:7=2;8:10=3;NA=4")
```

Se voglio cambiare tutte le variabili, posso usare apply:

```
R> L <- apply(L, 2,
+  function(x) x <- recode(x,"1:3=1;4:7=2;8:10=3;NA=4"); x)
```

E se voglio cambiare solo alcune variabili, posso usare apply:

```
R> L[c(2,5,7)] <- apply(L[c(2,5,7)], 2,
+ function(x) x <- recode(x,"1:3=1;4:7=2;8:10=3;NA=4"); x)
```

2.2.8.7 Ribaltare item

Capita spesso in psicologia di usare variabili (che indicano un costrutto psicologico, un *tratto*) misurate tramite item tipo Likert (più precisamente scale a ranghi, che oscillano da 1 a `n`, in genere 1-5, 1-7, 1-9 oppure fra 0 e `n`), in cui alcuni item sono pro-tratto (un punteggio alto indica *alta* presenza del costrutto) ed altri contro-tratto (un punteggio alto indica *bassa* presenza del tratto). Quando queste variabili-item vanno sommate fra loro per ottenere il totale della scala, bisogna considerare che gli item contro-tratto devono essere ribaltati.

Possiamo usare un recode(), in questo modo:

```
R> library(car) # se non caricato
R> nuovavar=recode(variabile, "1=5; 2=4; 3=3; 4=2; 5=1") # ipotizzando 1-5
R> nuovavar=recode(variabile, "0=4; 1=3; 2=2; 3=1; 4=0") # ipotizzando 0-4
```

Oppure possiamo ribaltare l'item considerando che il valore massimo diventa 1, quello immediatamente inferiore diventa 2 e così via (v. Tab. 2.9) e che la somma fra il valore originale e il nuovo valore è una costante che corrisponde al valore massimo più il valore minimo. Quindi basta sottrarre il valore originale alla somma di massimo e minimo. Per fare questa operazioni basta usare (ipotizziamo un item su scala 1-5 con 20 valori):

Tabella 2.9: Ribaltamento di un item a 5 gradini

il valore	diventa	ovvero		il valore	diventa	ovvero	
1	5	1+5=6	6-1=5	0	4	0+4=4	4-0=4
2	4	2+4=6	6-2=4	1	3	1+3=4	4-1=3
3	3	3+3=6	6-3=3	2	2	2+2=4	4-2=2
4	2	4+2=6	6-4=2	3	1	3+1=4	4-3=1
5	1	5+1=6	6-5=1	4	0	4+0=4	4-4=0

```
R> X=sample(1:5,20,rep=T) # 20 valori compresi da 1 e 5
R> X #visualizzo la variabile
 [1] 1 1 3 3 4 5 4 3 2 1 1 4 4 5 5 5 2 3 3 3
```

```
R> 6-X # ribalto
 [1] 5 5 3 3 2 1 2 3 4 5 5 2 2 1 1 1 4 3 3 3
```

Se usiamo un dataframe possiamo sia modificare il singolo item, sia creare un nuovo item da aggiungere, sia creare un dataframe nuovo contenente tutti gli item correttamente orientati. Per gli esempi che seguono userò un dataframe chiamato `Quest` composto da 16 item (chiamati `Quest1`, `Quest2` e così via, tutti gli item sono a 9 gradi) di cui gli item 2, 3, 5, 8, 10, 11, 13 e 15 sono contro-tratto.

```
R> load("Quest.rda") # carichiamo i dati-aggiustare il percorso
R> # 1) ripetere per ogni item da invertire
R> Quest$Quest2=10-Quest$Quest2 # inversione del singolo item
R> # oppure, in alternativa
R> # 2) ripetere per ogni item da invertire
R> Quest$Q2r=10-Quest$Quest2 # aggiungere un item invertito
```

Nel primo caso, abbiamo sovrascritto la variabile `Quest2` del dataframe `Quest` con la versione invertita; nel secondo caso, abbiamo aggiunto in fondo al dataframe una nuova variabile (`Q2r`). Bisognerà ovviamente ripetere le due procedure per tutte le variabili da invertire.

```
R> # 3) trasformo le variabili nel file originario
R> transform(Quest, Quest2=10-Quest2, Quest3=10-Quest3, Quest5=10-Quest5,
+ Quest8=10-Quest8, Quest10=10-Quest10, Quest11=10-Quest11,
+ Quest13=10-Quest13, Quest15=10-Quest15)
R> # 4) creiamo un nuovo file
R> QuestBis = cbind(Quest[,c(1,4,6,7,9,12,14,16)+1],
+ 10-Quest[,c(2:3,5,8,10:11,13,15)+1])
```

Nel terzo caso abbiamo usato la funzione `transform()` per modificare contemporaneamente tutte le variabili implicate.

Nel quarto caso abbiamo usato la funzione `cbind()` per concatenare due pezzi di dataframe. Il primo pezzo è formato selezionando le sole variabili pro-tratto, tramite la funzione `c()` e il secondo pezzo è formato selezionando i soli item contro-tratto e ribaltandoli. Scrivere `c(1, 4, 6, 7, 9, 12, 14, 16)+1` è del tutto analogo (ma più comodo) che scrivere `c("Quest1", 'Quest4',\dots{})` e così via. Può sconcertare l'aggiunta di un 1 alla sequenza generata, ma basta ricordare che il dataframe comprende in prima posizione la variabile `Newcod`, quindi l'item `Quest1` è in seconda posizione...

2.2.9 Oggetti

Le funzioni, i comandi e i risultati di R sono in realtà degli "oggetti" (intesi in senso informatico). Per questo motivo un oggetto non è solo un insieme di operazioni matematiche, ma si può adeguare in modo (quasi-)automatico al tipo. Ad es. ogni oggetto, "sa" cosa deve visualizzare e come visualizzarlo e questo ci permette di usare un generico comando `print()` per i diversi oggetti; ogni oggetto lo interpreterà a modo suo. Se il risultato di un comando è un oggetto, può contenere molte più informazioni di quelle visualizzate immediatamente.

Ad esempio, il comando `cor.test(x,y)` non solo calcola la correlazione tra `x` e `y` ma effettua anche un test di significatività e calcola l'intervallo di confidenza. Salvando il risultato (ad es. `R> L.cor <- cor.test(L$L1,L$L2)`), si otterrà un oggetto che contiene vari tipi di informazione, ciascuna recuperabile tramite dei sottocomandi: `L.cor$statistic` restituirà il valore della

correlazione, `L.cor$p.value` restituirà la probabilità associata, `L.cor$parameter` restituirà i gradi di libertà...

```
R> L.cor <- cor.test(L$L1,L$L2)
R> L.cor$statistic
        t
0.9830107
R> L.cor$p.value
[1] 0.3277593
R> L.cor$parameter
 df
110
```

2.2.10 Esportare variabili e dati

R dispone di diverse possibilità per esportare dati (o variabili) in base al tipo di dati che volete esportare.

2.2.10.1 Esportare oggetti di R

Un primo tipo di dati è rappresentato da un oggetto di R (qualsiasi tipo). In questo caso il comando è `save()`:

```
save(..., list = character(0L),
     file = stop("'file' must be specified"),
     ascii = FALSE, version = NULL, envir = parent.frame(),
     compress = !ascii, eval.promises = TRUE, precheck = TRUE)

save.image(file = ".RData", version = NULL, ascii = FALSE,
           compress = !ascii, safe = TRUE)
```

La prima forma serve per salvare un qualunque oggetto (o una lista di oggetti), mentre il secondo salva tutto quello che avete in memoria in quel momento (cioè tutti gli oggetti caricati o creati nella sessione corrente). Va ricordato che, al momento di uscire da R, vi sarà chiesto se volete salvare la sessione e, se risponderete di sì, è esattamente questo comando che verrà utilizzato in automatico.

In questo manualetto abbiamo creato un dataframe di nome `Dati1`. I possibili comandi per salvarlo sono:

```
R> save(Dati1,file="Dati1.rda")
R> save(Dati1,file="c:\\Dati\\Dati1.rda")
R> save(Dati1,file="c:/Dati/Dati1.rda")
```

L'unica cosa importante è il nome dell'oggetto in prima posizione e successivamente il nome del file dove salvarlo (sempre preceduto da `file=`, perché si possono salvare più oggetti nello stesso file).

Il nome del file può essere lungo quanto permesso dal sistema operativo su cui lavorate e l'estensione suggerita da R è `".rda"` (R data), ma voi potete assegnare qualunque estensione desideriate (basta che poi ve la ricordiate).

Per quanto riguarda la directory (o cartella) in cui il file verrà salvato, se non è indicata (primo esempio) il file verrà salvato nella directory attuale di lavoro. Se invece volete salvarlo da qualche altra parte, basta indicare il percorso completo usando la notazione del vostro sistema operativo. È importante però (per gli utenti di Windows) ricordare che le barre rovesciate (`"\"`) generano un errore e allora bisogna usare o la doppia barra rovesciata (`"\\"`, secondo esempio) o la barra normale (`"/"`, terzo esempio).

Tra i parametri disponibili c'è anche `ascii=` che è impostata automaticamente a falso, per cui i comandi `save()` utilizzati nell'esempio scrivono un file binario. Se vogliamo un file leggibile da un qualunque editor di testo, dobbiamo impostare questo parametro a vero.

```
R> save(Dati1,file="Dati1.rda",asci=T)
```

2.2.10.2 Esportare dataframe in testi

Il tipo di dati più comune in psicologia è quello tabellare (righe per colonne cioè osservazioni per variabili). In questo caso, la funzione che si può usare è `write.table()`, un comando abbastanza elastico per scrivere in un file esterno dati che possono essere letti da un qualunque editor di testo (da Notepad a Microsoft Word).

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"))

write.csv(\dots{})
write.csv2(\dots{})
```

Per usare questo comando, bisogna indicare un dataframe o una matrice da esportare e il nome del file in cui scriverlo. Il parametro `append=` permette di creare un nuovo file (`FALSE`) oppure di scrivere i dati in aggiunta a quelli già presenti (`TRUE`) se il file già esiste. I parametri `quote=`, `sep=` e `dec=` permettono di scrivere file di testo sia in formato libero sia con campi delimitati da un particolare carattere. Il parametro `sep=` indica come vanno separati i contenuti delle variabili e il default è uno spazio vuoto; `quote=` è impostato a `TRUE` e scrive le variabili di tipo carattere o fattore incluse fra virgolette (`"`), se lo si pone a FALSE le scrive senza virgolette, se anziché un valore logico si indica un vettore, questo deve contenere il numero sequenziale delle variabili che volete incluse tra virgolette. Tramite `dec=` possiamo indicare quale carattere utilizzare per la virgola/punto decimale. Gli altri parametri includono `na=` per modificare il contenuto da scrivere per indicare valori mancanti; `eol=` permette di specificare il carattere di fine riga; `row.names=` e `col.names=` permettono di decidere se scrivere anche i nomi di riga e di colonna.

Siccome un particolare tipo di file di testo (utilizzabile anche con Excel) utilizza il formato `.csv` (*Comma Separated Value*), esiste un alias `write.csv()` che imposta automaticamente i parametri per scrivere i dati in questo formato. Le due versioni (con e senza il 2 finale) si differenziano per il punto decimale: la prima versione utilizza il *punto decimale* mentre quella che termina con un "2" serve per il formato europeo che utilizza la *virgola decimale*.

Siccome `na` è impostato automaticamente a NA, se volete scrivere un file di *csv* leggibile da Excel (quindi senza il testo "NA" nelle celle), dovete impostare `na=''`.

Per salvare il dataframe `Dati1` in un formato importabile da Excel (versione europea):

```
R> write.csv2(Dati1, file="c:/dati/dati1.csv")
R> write.csv2(Dati1, file="c:/dati/dati1.csv", na='') #senza NA
```

2.2.10.3 Esportare dataframe in Excel

Tramite il pacchetto `xlsReadWrite` (solo per Windows a 32 bit, da installare e poi caricare) possiamo anche scrivere direttamente file in formato Excel (fino alla versione 2003):

```
write.xls(x, file, colNames = TRUE, sheet = 1, from = 1, rowNames = NA )
```

Anche in questo caso bisogna indicare un dataframe e un nome di file; per default, i nomi delle variabili vengono scritte nella prima riga della tabella Excel e, per default, i dati vengono messi nel primo foglio (*sheet*).

```
R> library(xlsReadWrite)
R> write.xls(Dati1, file="c:/dati/dati1.xls")
```

Se invece vogliamo scrivere file Excel nel formato della versione 2007 e successivi (quelli con estensione `.xlsx`) dobbiamo usare il pacchetto `xlsx` (da installare e poi caricare), la cui sintassi è simile alla precedente:

```
write.xlsx(x, file, sheetName="Sheet 1", formatTemplate=NULL,
    col.names=TRUE, row.names=TRUE)
```

le differenze da notare sono la `x` nel nome della funzione, e che per default vengono salvati anche i nomi associati alle righe. L'uso è analogo:

```
R> library(xlsx)
R> write.xlsx(Dati1, file="c:/dati/dati1.xlsx")
```

Esistono altri due pacchetti `dataframes2xls` e `WriteXLS` che però possono solo scrivere un dataframe in formato Excel ma non sono poi in grado di leggere un file Excel e quindi non li approfondisco. [da FINIRE].

2.2.10.4 Esportare dataframe in SPSS

Il pacchetto `foreign` (da installare e caricare) permette di leggere i file SPSS (scritti con i comandi `SAVE` o `EXPORT` di Spss) ma è anche in grado di esportare dataframe di R in SPSS. Non lo fa nel formato nativo (`.sav`) ma tramite il formato tabellare (v. 2.2.10.2). In pratica, scrive un file di dati e un file di sintassi che serve per caricarli in SPSS. La sintassi (che vale anche per altri pacchetti statistici) è:

```
write.foreign(df, datafile, codefile,
              package = c("SPSS", "Stata", "SAS"), ...)
```

Nella sintassi, `df` è il nome del dataframe, `datafile` è il nome che si vuole assegnare ai dati e `codefile` è il nome per il file della sintassi. Nell'esempio che segue, il dataframe `Dati1` verrà scritto in un file di testo "Dati1.txt" che sarà possibile caricare tramite il file di sintassi "Dati1.sps".

```
R> write.foreign(Dati1, "Dati1.txt", "Dati1.sps", package="SPSS")
```

2.2.10.5 Esportare risultati

La maggior parte dei risultati sono scritti come testo a spaziatura fissa e quindi possono essere salvati in un file di testo. In teoria, si possono assegnare i risultati di una funzione ad una variabile e successivamente salvare la variabile in formato ascii con `save()`.

È più comodo usare la procedura `sink()` che usa come parametro il nome di un file a cui invia tutti i risultati finche non la si richiama senza parametri. Esempio:

```
R> sink("c:/dati/risultati.txt")
R> Dati1
R> sink()
```

se andate a vedere nella directory "dati", troverete un file `risultati.txt` che contiene il risultato del comando `R> Dati1` ovvero "visualizza il contenuto di Dati1".

Se volete salvare più risultati (oppure inserire un titolo) dovete ricordare che `sink()` non separa i vari risultati, quindi dovete prevedere un comando `cat()` che contenga degli "a capo" (rappresentato da `\n`), ad es: `cat("Contenuto del dataframe Dati1\n\n")`; è importante ricordarsi di usare le virgolette doppie (`"`).

2.2.11 Caricare dati e variabili

R dispone di diverse possibilità per importare dati (o variabili) in base al tipo di dati che volete importare.

2.2.11.1 Importare oggetti di R

Per importare un oggetto di R (cioè esportato con `save()`), si usa il comando `load()` che richiede solo il nome del file da importare (l'oggetto importato avrà lo stesso nome di quello esportato):

```
R> load("L.rda")
R> load("c:\\dati\\L.rda")
R> load("c:/dati/L.rda")
```

2.2.11.2 Importare dati da testo

Nel caso di dataframe o matrici, si può usare la procedura `read.table()` che serve per caricare dati in formato testo sia con informazioni libere sia con campi delimitati da un certo carattere:

```
read.table(file, header = FALSE, sep = "", quote = "¨'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
```

```
            allowEscapes = FALSE, flush = FALSE,
            stringsAsFactors = default.stringsAsFactors(),
            fileEncoding = "", encoding = "unknown")

read.csv(file, header = TRUE, sep = ",", quote="¨", dec=".",
         fill = TRUE, comment.char="", ...)

read.csv2(file, header = TRUE, sep = ";", quote="¨", dec=",",
          fill = TRUE, comment.char="", ...)

read.delim(file, header = TRUE, sep = "^", quote="¨", dec=".",
           fill = TRUE, comment.char="", ...)

read.delim2(file, header = TRUE, sep = "^", quote="¨", dec=",",
            fill = TRUE, comment.char="", ...)
```

`read.table()` è un comando molto elastico che si adatta a diverse situazioni; il parametro più importante è l'indicazione del file (in prima posizione). I vari parametri hanno un valore di default e, nella maggior parte dei casi non serve modificarle.

Tuttavia siccome alcuni parametri servono per file di testo particolari, esistono 4 versioni per file specifici: le due versioni di `read.csv` servono per leggere file `.csv` (*Comma Separated Value*, formato esportabile da Excel) nella versione con il punto decimale o con la virgola decimale; le due versioni di `read.delim` servono per i file di testo delimitati da tabulatori e anche in questo caso la versione che termina con un 2 è per leggere numeri con la virgola decimale.

Bisogna ricordarsi di assegnare la tabella letta ad una variabile, però:

```
R> NuovoDati1 = read.csv2("c:/dati/dati1.csv")
```

2.2.11.3 Importare dati da Excel

Il pacchetto `xlsReadWrite` (solo per Windows) può leggere direttamente file in formato Excel (fino alla versione 2003), con l'istruzione `read.xls()` la cui sintassi è:

```
read.xls(file, colNames = TRUE, sheet = 1, type = "data.frame",
         from = 1, rowNames = NA, colClasses = NA, checkNames = TRUE,
         dateTimeAs = "numeric",
         stringsAsFactors = default.stringsAsFactors() )
```

In questo caso, il parametro `type=` imposta automaticamente la variabile a `data.frame` ed è anche quello che in genere vogliamo. Ad esempio, il file letto sarà un dataframe:

```
R> NuovoDati1 = read.xls("c:/dati/dati1.xls")
```

Dopo aver installato il pacchetto, la prima volta che viene caricato, `xlsReadWrite` richiede che venga eseguito un comando che scarica da un sito un pezzo di programma che non può essere distribuito.

Tramite il pacchetto `xlsx` possiamo invece leggere file Excel delle versioni 2007 e seguenti. La sintassi del comando è (anche qui il default è un dataframe):

```
read.xlsx(file, sheetIndex, sheetName=NULL, rowIndex=NULL,
          as.data.frame=TRUE, header=TRUE, colClasses=NA,
          keepFormulas=FALSE)
```

mentre un esempio di utilizzo può essere:

```
R> # è obbligatorio indicare il foglio
R> NuovoDati1 = read.xlsx("c:/dati/dati1.xlsx", 1)
```

Se nel vostro sistema è installato il software perl (http://www.perl.org/) potete usare anche la funzione read.xls() del pacchetto gdata.

Un altro modo per accedere ad un file Excel è tramite il protocollo ODBC che permette di accedere a qualsiasi formato di database (Dbase, Access, MySql, ecc.) ed anche ad Excel.

2.2.11.4 Importare dati tramite ODBC

Tramite il protocollo ODBC si può accedere a diversi database in quanto è il sistema operativo che si preoccupa dell'accesso al database specifico. In R è possibile utilizzare tale protocollo tramite il pacchetto RODBC, ma ovviamente, la sua effettiva possibilità di utilizzao dipenderà dal fatto che il sistema operativo che si sta usando fornisca i driver ODBC necessari. Inoltre, in base al sistema operativo, il protocollo ODBC potrà permettere di aprire non solo i database relazionali (Oracle, PostgreSQL, Microsoft SQL, mySQL, ecc.) ma anche Excel, dBase, FoxBase e simili. In alcun casi sono i singoli fornitori a fornire i driver per determinati sistemi operativi.

Alcuni database hanno specifici pacchetti, come dbConnect, RMySQL per MySQL; RpgSQL, RPostgreSQL per PostgreSQL; RSQLite per SQLite. [da FINIRE].

2.2.11.5 Importare dati da Spss

Il pacchetto `foreign` (da installare e caricare) permette di leggere i file SPSS (scritti con i comandi `save` o `export` di Spss) e di importarli in R. La sintassi è:

```
read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE,
          max.value.labels = Inf, trim.factor.names = FALSE,
          trim_values = TRUE, reencode = NA, use.missings = to.data.frame)
```

Siccome il parametro `to.data.frame` è (per default) impostato a FALSE, bisogna ricordarsi di attivarlo. Un'altra cosa a cui fare attenzione è il parametro `use.value.labels` impostato a vero, che implica che tutte le variabili a cui siano state associate delle etichette dei valori vengono automaticamente trasformati in "fattori". Infine, il parametro `use.missings = to.data.frame` importa i valori effettivamente indicati in Spss anche se sono stati indicati come "mancanti definiti dall'utente" (*user missing*); se vogliamo che diventino dei veri mancanti dobbiamo impostarlo a `NA`.

Per caricare il file `Liht.sav` accluso a questo manualetto, usiamo il comando:

```
R> library(foreign) # se non già caricato
R> Liht=read.spss("c:/Dati/Liht.sav", to.data.frame=T)
```

Lo stesso pacchetto fornisce la possibilità di leggere file in altri formati statistici, come SAS, Stata, Systat, Minitab oppure in formati con loro compatibili come il DBF (Dbase).

Un altro comando per caricare dati da SPSS si trova nel pacchetto `Hmisc`. La funzione si chiama `spss.get` è la sua sintassi è:

```
spss.get(file, lowernames=FALSE, datevars = NULL,
         use.value.labels = TRUE, to.data.frame = TRUE,
         max.value.labels = Inf, force.single=TRUE,
         allow=NULL, charfactor=FALSE)
```

I parametri `use.value.labels`, `to.data.frame` e `max.value.labels` sono identici a quelli di `read.spss`, ma i primi due sono automaticamente impostati a `TRUE` (scelta che di solito si utilizza con i dataframe), per cui è sufficiente un solo comando:

```
R> library(Hmisc) # se non già caricato
R> new <- spss.get("c:/Dati/Liht.sav")
```

Rcmdr: `Dati | Importa dati | da set di dati di SPSS`

2.2.12 Importare dati da altri software statistici

Nel pacchetto `foreign` oltre alla funzione per importare dati dai file in formato SPSS, ci sono funzioni per importare file dati da Systat, SAS, Stata, Minitab e Epi Info, nonché dal formato DBF (usato da dBase e FoxBase) e Octave. Altri software statistici (Spad, Statistica, etc.) hanno la possibilità di salvare i dati nei formato di SPSS oppure in Excel o CSV e si può quindi bisogna passare tramite qualcuno dei formati indicati nei precedenti paragrafi.

Parte II

Statistica descrittiva e analisi esplorativa dei dati

Capitolo 3

Statistica descrittiva

La statistica descrittiva ha lo scopo di capire "cosa abbiamo" dentro ad un insieme di dati, prima di iniziare ad analizzarli.

Tuttavia, prima di iniziare ad applicare le statistiche in R, vale la pena assicurarsi di cosa contiene il dataframe su cui intendiamo lavorare. In particolare, capire come sono stati "intesi" i dati che abbiamo inserito nel dataframe. Un comando utile a questo scopo è `str()` che ci riassume le principali caratteristiche delle variabili. Applichiamolo al dataframe `Dati1`:

```
R> str(Dati1)
'data.frame':   10 obs. of  4 variables:
 $ Cod   : Factor w/ 10 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10
 $ Genere: Factor w/ 2 levels "F","M": 2 2 1 1 1 1 2 1 2 1
 $ Eta   : num  43 36 56 47 58 37 46 30 28 26
 $ Anni  : num  14 6 28 12 1 6 5 2 2 1
```

Il comando `str()` ci dice che il nostro dataframe contiene 10 casi statistici e 4 variabili. Le prime due sono di tipo "fattore", la prima con 10 livelli e la seconda con 2; le etichette della prima variabile sono `"A","B","C",\ldots{}` , quelle della seconda variabile sono `"F","M"` e sono rappresentate internamente da numeri. Le ultime due variabili sono di tipo numerico e vengono elencati tutti i valori presenti.

Se vogliamo vedere le informazioni separatamente per ogni variabile abbiamo la possibilità di usare la funzione `unique()` che riporta una sola i valori presenti in una variabile:

```
R> unique(Dati1$Genere) # oppure unique(Dati1[,2])
[1] M F
Levels: F M
R> unique(Dati1[2])
  Genere
1      M
3      F
```

Nel primo caso, ha trattato la variabile come un fattore, indicando i valori trovati e i livelli; nel secondo caso ha trattato la variabile come una "lista" e ha indicato il numero dell'osservazione in cui compare per la prima volta un determinato valore.

In R sono disponibili le principali statistiche descrittive che si possono utilizzare su un vettore o su una variabile di un dataframe. La maggior parte presume di lavorare con un vettore di dati, ma negli esempi che seguono userò variabili di un dataframe (che sono "appunto" dei vettori).

Tabella 3.1: Alcune funzioni statistiche

Operazione	*Funzione da usare*	*Esempio*
Media	mean(v)	mean(balene)
Varianza (N-1)	var(v)	var(balene)
Covarianza (N-1)	cov(v)	var(balene)
Deviazione standard (N-1)	sd(v)	sd(balene)
Mediana	median(v)	median(balene)
Quartili	quantile(v,p)	quantile(balene,.25)
5 numeri	fivenum(v)	fivenum(balene)

```
R> v=c(1:10) # numeri da 1 a 10
R> v
 [1]  1  2  3  4  5  6  7  8  9 10
R> mean(v) # media
[1] 5.5
R> median(v) # mediana
[1] 5.5
R> var(v) # varianza (N-1)
[1] 9.17
R> sd(v) #deviazione standard (N-1)
[1] 3.03
```

3.1 Distribuzione di frequenza

La distribuzione di frequenza di una variabile si ottiene con il comando `table` che si può applicare ad una singola variabile sia di tipo numerico che di tipo fattore:

```
R> table(Dati1$Genere) # fattore
F M
6 4
R> table(Dati1$Anni) # numerico
 1  2  5  6 12 14 28
 2  2  1  2  1  1  1
```

Nella prima riga dei risultati vengono elencati i "valori" trovati, mentre nella seconda compaiono le frequenze relative.

È possibile "trasporre" i risultati chiedendo che siano visualizzati come dataframe:

```
R> as.data.frame(table(Dati1$Anni))
  Var1 Freq
1    1    2
2    2    2
3    5    1
4    6    2
5   12    1
6   14    1
7   28    1
```

Per ottenere le percentuali o le proporzioni si deve fare il relativo calcolo (eventualmente arrotondando, con `round()`):

```
R> table(Dati1$Anni)/sum(Dati1$Anni) # oppure
R> with(Dati1, table(Anni)/sum(Anni)) # proporzione

     1      2      5      6     12     14     28
0.0260 0.0260 0.0130 0.0260 0.0130 0.0130 0.0130
R> round((table(Dati1$Anni)/sum(Dati1$Anni))*100,2) # percentuale

  1   2   5   6  12  14  28
2.6 2.6 1.3 2.6 1.3 1.3 1.3
```

È anche possibile salvare il risultato della tabella in una variabile temporanea per facilitare l'accesso ai risultati:

```
R> .tabella <- table(Dati1$Genere)
R> .tabella

F M
6 4
R> .tabella/sum(.tabella) # proporzioni

  F   M
0.6 0.4
R> round(100*.tabella/sum(.tabella), 2)  # percentuali

 F  M
60 40
R> rm(.tabella)
```

Ricordiamo che le variabili che iniziano con un punto (ad es. `.tabella`) non sono visibili fra gli oggetti e quindi è opportuno eliminarle dopo averle usate.

Un risultato analogo si può ottenere con la funzione prop.table() che calcola le proporzioni su una tabella:

```
R> prop.table(table(Dati1$Anni))

  1   2   5   6  12  14  28
0.2 0.2 0.1 0.2 0.1 0.1 0.1
```

3.2 Indici di posizione

Gli indici di posizione indicano il valore che cade in corrispondenza di una determinata posizione. Generalmente si ottengono dividendo l'intera distribuzione in parti uguali: i terzili la dividono in 3 parti uguali, i quartili in 4, i decili in 10 e i centili o percentili in 100. La maggior parte delle suddivisioni si possono esprimere come percentile.

Il comando per ottenere gli indici di posizione in R è quantile() a cui bisogna passare come parametri il nome di un vettore e il quantile richiesto espresso come percentile (si possono ignorare i mancanti con na.rm=T):

```
R> quantile(Dati1$Eta, .25,na.rm=F)
 25%
31.5
R> quantile(Dati1$Eta, c(.25,.5,.75),na.rm=F) # quartili
 25%  50%  75%
31.5 40.0 46.8
```

I quartili e il valore minimo e massimo si possono ottenere anche con le funzioni min(), max(), `range()` e `fivenum()` (con tutti si possono gestire i mancanti con na.rm).

```
R> min(L$L1,na.rm=F) # min
[1] 1
R> max(L$L1) # massimo
[1] 7
R> range(L$L1) # minimo e massimo
[1] 1 7
R> quantile(L$L1,c(.25,.5,.75)) # quartili
25% 50% 75%
  4   6   7
R> fivenum(L$L1) # 5 numeri riassuntivi
[1] 1 4 6 7 7
```

Con i valori del primo e terzo quartile (per variabili quantitative) è possibile calcolare la differenza interquartilica (o IQR). In R è disponibile la funzione `IQR()` (attenzione al fatto che è scritta in tutto maiuscolo):

```
R> IQR(Dati1$Eta))
[1] 15.25
```

3.3 Statistiche della tendenza centrale

In R non esiste una funzione per calcolare la moda, sia perché è il peggior indice della tendenza centrale, sia perché il comando mode() serve a indicare il tipo di un oggetto.

La mediana si può calcolare con il comando median() ricordandosi che considera qualunque vettore di numeri come se fosse una scala ad intervallo, usando quindi una interpolazione lineare per la stima esatta del valore corrispondente alla metà della distribuzione. La funzione median() lavora con un vettore per volta, quindi non si può applicare ad una parte di dataframe.

```
R> median(Quest[,8])
[1] NA
R> median(Quest[,8],na.rm=T)
[1] 5
R> median(Quest[,3:8],na.rm=T)
Errore in median.default(Quest, na.rm = T) : necessario dati numerici
```

Per l'uso con i dataframe si può usare la funzione sapply() che provvede ad applicare una funzione ad ogni elemento del dataframe:

```
R> sapply(Quest[,3:8],median)
Quest2 Quest3 Quest4 Quest5 Quest6 Quest7
     5      5      7     NA      5     NA
R> sapply(Quest[,3:8],median,na.rm=T)
Quest2 Quest3 Quest4 Quest5 Quest6 Quest7
     5      5      7      5      5      5
```

Abbiamo già visto la funzione `mean()` che calcola la media di un vettore. Se però al posto di un vettore passiamo un dataframe, la funzione restituirà la media di tutte le variabili presenti (oppure di un blocco di variabili).

```
R> mean(L[,1:5])
      L1       L2       L3       L4       L5
5.294643 4.973214 5.098214 2.098214 1.892857
```

Se ci fossero dei valori mancanti in una delle variabili, il risultato sarebbe `NA`. Per escludere i valori mancanti si deve usare il parametro `na.rm=T`:

```
R> mean(Quest[,2:8])
Quest1 Quest2 Quest3 Quest4 Quest5 Quest6 Quest7
  4.55   5.06   5.18   6.39     NA   4.33     NA
R> mean(Quest[,2:8],na.rm=T)
Quest1 Quest2 Quest3 Quest4 Quest5 Quest6 Quest7
  4.55   5.06   5.18   6.39   4.50   4.33   5.14
```

Nelle versioni più recenti, usando `mean()` si ottiene un avviso sul fatto che questa funzione è "deprecata" ovvero sarebbe bene non usarla perché potrebbe cambiare nel futuro o non essere più utilizzabile.

Se vogliamo adeguarci, bisogna usare la funzione sapply() nella forma dataframe, funzione ed eventuali parametri:

```
R> sapply(Quest[,2:8], mean, na.rm=T)
  Quest1   Quest2   Quest3   Quest4   Quest5   Quest6   Quest7
4.553648 5.058655 5.177396 6.391989 4.497848 4.331903 5.143472
```

3.4 Statistiche di variabilità

Una delle statistiche della variabilità più utilizzata è la deviazione standard che in R si può ottenere tramite la funzione `sd()` (usando $N-1$ al denominatore) e che funziona esattamente come `mean()`.

```
R> sd(Quest$Quest2)
[1] 2.602785
R> sd(Quest[,3:8])
  Quest2   Quest3   Quest4   Quest5   Quest6   Quest7
2.602785 2.650621 2.035711       NA 2.601795       NA
R> sd(Quest[,3:8],na.rm=T)
  Quest2   Quest3   Quest4   Quest5   Quest6   Quest7
2.602785 2.650621 2.035711 2.559703 2.601795 2.547157
```

Anche questa funzione non dovrebbe essere usata con i data frame e quindi può essere sostituita da:

```
R> sapply(Quest[,2:8], sd, na.rm=T)
  Quest1   Quest2   Quest3   Quest4   Quest5   Quest6   Quest7
2.409014 2.602785 2.650621 2.035711 2.559703 2.601795 2.547157
```

Per calcolare la varianza (sempre con $N-1$ al denominatore), usiamo la funzione `var()` che può operare sia su un singolo vettore sia su un dataframe. La funzione `var()` restituisce la varianza se usiamo un vettore ma restituisce una matrice di varianze/covarianze se indichiamo un dataframe.

```
R> var(L$L1)
[1] 3.597088
R> round(var(L),3) # arrotondo a 3 decimali
       L1     L2     L3     L4     L5
L1  3.597  0.386  1.664 -1.263 -0.680
L2  0.386  4.765  1.777 -0.295 -0.805
L3  1.664  1.777  3.639 -1.658 -1.242
L4 -1.263 -0.295 -1.658  2.071  1.281
L5 -0.680 -0.805 -1.242  1.281  2.277
```

In una matrice di varianze/covarianze, lungo la diagonale abbiamo la varianza e fuori dalla diagonale, le covarianze. È una matrice simmetrica perché la covarianza di `L1` con `L2` è uguale a quella di `L2` con `L1`.

Esiste anche un comando `cov()` per il calcolo della covarianza, ma si differenzia da `var()` solo quando si calcola una covarianza singola; è infatti necessario indicare due vettori.

```
R> cov(L$L1,L$L2)
[1] 0.3863417
R> round(cov(L[,1:5]),3) # arrotondo
       L1     L2     L3     L4     L5
L1  3.597  0.386  1.664 -1.263 -0.680
L2  0.386  4.765  1.777 -0.295 -0.805
L3  1.664  1.777  3.639 -1.658 -1.242
L4 -1.263 -0.295 -1.658  2.071  1.281
L5 -0.680 -0.805 -1.242  1.281  2.277
```

3.5 Statistiche di normalità

Le statistiche di normalità sono l'indice di asimmetria e quella di curtosi. Entrambi gli indici sono uguali a 0 quando la variabile è perfettamente normale; valori molti vicini allo 0 indicano piccole deviazioni dalla normalità (per sapere quanto ci si possa allontanare dal valore "normale" si veda il par. 5).

La prima si ottiene con `skewness()` del pacchetto `e1071` che si può usare solo con un vettore/variabile oppure con `skew()` del pacchetto `psych` che invece funziona anche con variabili di dataframe.

```
R> library(e1071)
R> skewness(L$L1,na.rm=T)
[1] -0.8386405
R> library(psych) # se non già caricato
R> skew(L$L1,na.rm=T)
```

```
[1] -0.8386405
R> skew(L[,1:5],na.rm=T)
[1] -0.8386405 -0.7489366 -0.6638548  1.5009624  1.9424770
```

Analogamente per la curtosi, con `kurtosis()` presente nel pacchetto `e1071` (solo un vettore alla volta) oppure con `kurtosi()` nel pacchetto `psych` (vettore e dataframe).

```
R> kurtosis(L[,1],na.rm=T) # con e1071
[1] -0.6928887
R> kurtosi(L[,1:5]) # con psych
        L1         L2         L3         L4         L5
-0.6928887 -0.9421235 -0.9067945  1.8629273  3.2715461
```

Per default sia `e1071` sia `psych` usano la formula generalmente indicata come $\mathbf{G}_3$ usata da software come MINITAB e BMDP. Per avere una statistica simile a quella di SPSS o di SAS, bisogna usare il parametro `type=2`. Per grandi campioni le varie formule di asimmetria e curtosi tendono allo stesso risultato.

```
R> skewness(L$L1,na.rm=T, type=2)
[1] -0.8615812
R> kurtosis(L[,1],na.rm=T, type=2)
[1] -0.6256169
```

3.6 Statistiche riassuntive

Non è comune chiedere a R singole statistiche descrittive per singole variabili, per cui è spesso più utile utilizzare funzioni che producano le principali statistiche descrittive (tendenza centrale, posizione, variabilità, asimmetria) in un colpo solo su più variabili.

Disponibile direttamente in R c'è la funzione `summary()` che produce gli stessi risultati di `fivenum()` per le variabili numeriche e la distribuzione di frequenza per quelle nominali o fattori.

```
R> summary(Dati1[2:4])
 Genere      Eta             Anni
 F:6    Min.   :26.0   Min.   : 1.0
 M:4    1st Qu.:31.5   1st Qu.: 2.0
        Median :40.0   Median : 5.5
        Mean   :40.7   Mean   : 7.7
        3rd Qu.:46.8   3rd Qu.:10.5
        Max.   :58.0   Max.   :28.0
```

In realtà l'uso delle funzioni native di R non è molto comoda sia per le informazioni che forniscono sia per la loro scarsa praticità con i dataframe; per questo motivo esistono alcuni pacchetti che forniscono delle scorciatoie.

Il primo pacchetto che prendo in considerazione è `prettyR` (Lemon & Grosjean, 2012) che contiene una funzione `describe()` che riporta le principali statistiche descrittive, separatamente per le variabili numeriche e quelle "factor":

```
R> library(prettyR)
R> describe(Dati1[2:4])
Description of Dati1[2:4]

Numeric
              mean    median       var        sd   valid.n
Eta           40.7        40       126     11.23        10
Anni           7.7       5.5      70.9      8.42        10

Factor

Genere
Value   Count   Percent
F       6       60
M       4       40
mode=F  Valid n=10

R> detach("package:prettyR")
```

Un'altro pacchetto che contiene una funzione `describe()` è `Hmisc` (Harrell & with contributions from many other users, 2012) il cui comportamento è piuttosto complesso. In base al tipo e al numero di valori, considera automaticamente la variabile come quantitativa o qualitativa. Nell'esempio seguente, la prima variabile ha più di 20 valori e viene riconosciuta come quantitativa, mentre le successive sono identificate come qualitative anche se sono numeriche (ovviamente le variabili "fattori" sono considerate qualitative).

```
R> library(Hmisc) # se non già caricato
R> describe(Quest[c(1,6:7)])
Quest[c(1, 6:7)]

 3  Variables      699  Observations
--------------------------------------------------------------------------------
NewCod
      n missing  unique    Mean     .05     .10     .25     .50     .75     .90
    699       0     699    5158    4037    4072    4176    5063    6026    7022
    .95
   7057

lowest : 4002 4003 4004 4005 4006, highest: 8029 8030 8031 8032 8033
--------------------------------------------------------------------------------
Quest5
      n missing  unique    Mean
    697       2       9   4.498

            1  2  3  4   5  6  7  8  9
Frequency 133 60 72 78 116 55 70 63 50
%          19  9 10 11  17  8 10  9  7
--------------------------------------------------------------------------------
Quest6
      n missing  unique    Mean
    699       0       9   4.332
```

```
            1  2  3  4   5  6  7  8  9
Frequency 161 72 56 56 101 83 72 58 40
%          23 10  8  8  14 12 10  8  6
--------------------------------------------------------------------------------
R> detach("package:Hmisc")
```

Infine anche nel pacchetto `psych` (Revelle, 2014) esiste una funzione `describe()` che riporta, fra le altre, le stesse statistiche di `prettyR` ma aggiunge minimo, massimo, intervallo, asimmetria e curtosi ed errore standard della media.

```
R> library(psych)
R> describe(Quest[6:7])
       var   n mean   sd median trimmed  mad min max range skew kurtosis  se
Quest5   1 697 4.50 2.56      5    4.41 2.97   1   9     8 0.14    -1.14 0.1
Quest6   2 699 4.33 2.60      5    4.22 2.97   1   9     8 0.14    -1.25 0.1
```

Se un dataframe contiene delle variabili "stringa" oppure a livello "factor", vengono automaticamente trasformate in valori progressivi. Nell'esempio che segue, Genere è codificato con "M" e "F".

```
R> describe(Dati1[2:4])
         var  n mean    sd median trimmed   mad min max range skew kurtosis   se
Genere*    1 10  1.4  0.52    1.0    1.38  0.00   1   2     1 0.35    -2.05 0.16
Eta        2 10 40.7 11.23   40.0   40.38 12.60  26  58    32 0.18    -1.53 3.55
Anni       3 10  7.7  8.42    5.5    6.00  5.93   1  28    27 1.29     0.54 2.66
```

Nello stesso pacchetto è disponibile anche il comando `describeBy()` che serve per suddividere le statistiche in base ad una variabile indipendente.

```
R> describeBy(Dati1[3:4],group=Dati1$Genere)
group: F
     var n  mean    sd median trimmed   mad min max range skew kurtosis   se
Eta    1 6 42.33 13.43     42   42.33 19.27  26  58    32 0.01    -2.02 5.48
Anni   2 6  8.33 10.52      4    8.33  4.45   1  28    27 0.94    -0.86 4.29
------------------------------------------------------------
group: M
     var n  mean   sd median trimmed  mad min max range  skew kurtosis   se
Eta    1 4 38.25 8.02   39.5   38.25 7.41  28  46    18 -0.25    -2.08 4.01
Anni   2 4  6.75 5.12    5.5    6.75 2.97   2  14    12  0.50    -1.81 2.56
R> detach("package:psych")
```

In entrambi i casi, un parametro di default è `na.rm=T`

Capitolo 4

Grafici esplorativi

R ha ottime capacità grafiche. Vediamo alcuni comandi che permettono di esplorare i dati (in genere ispirati dalla *Exploratory Data Analysis* di Tukey).

La maggior parte dei grafici permette di specificare titolo e sottotitoli tramite i parametri main=, sub=, xlab=, ylab=.

Incominciamo con i grafici "testuali" ramo e foglie (*steam-and-leafs*) che si ottengono con l'istruzione `stem()` che non richiede particolari parametri.

```
R> stem(Dati1$Anni)

  The decimal point is 1 digit(s) to the right of the |

  0 | 1122566
  1 | 24
  2 | 8
```

Gli istogrammi e i grafici a barre si ottengono con hist() a cui si possono passare i parametri generali (in fig. 4.1 un istogramma senza parametri e con):

```
R> hist(Dati1$Eta)
R> hist(Dati1$Eta, main="Istogramma dell'Età", xlab="Età", ylab="Frequenza")
```

Tuttavia la funzione `hist()` lavora sempre nell'ipotesi di fare un istogramma e non un grafico a barre, per cui le barre sono unite fra loro anche se la variabile fosse nominale o ordinale.

Con variabili nominali o ordinali, si può usare la funzione plot() che si adegua al tipo di dati.

```
R> plot(Dati1[,2],main='Grafico a barre di Genere') # grafico~4.2
```

Un'altro grafico che serve per esplorare le variabili è quello a scatola che si ottiene con boxplot().

Il comando

```
R> boxplot(Dati1$Anni)
```

produce il grafico di Figura 4.3a. In questo caso non è stato usato nessun parametro aggiuntivo (titolo, gestione dei mancanti...).

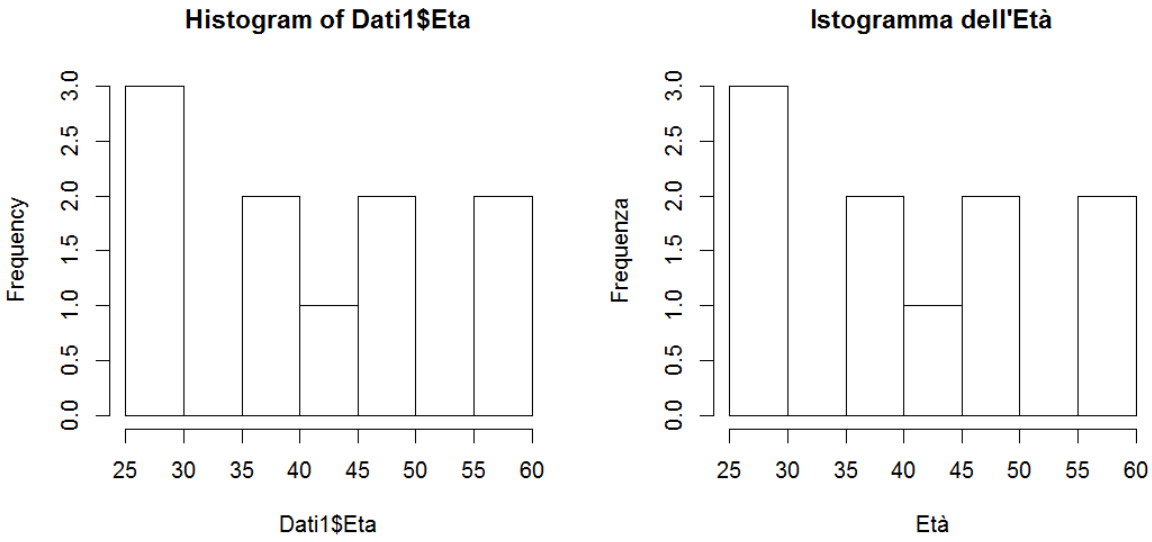

Figura 4.1: Istogrammi

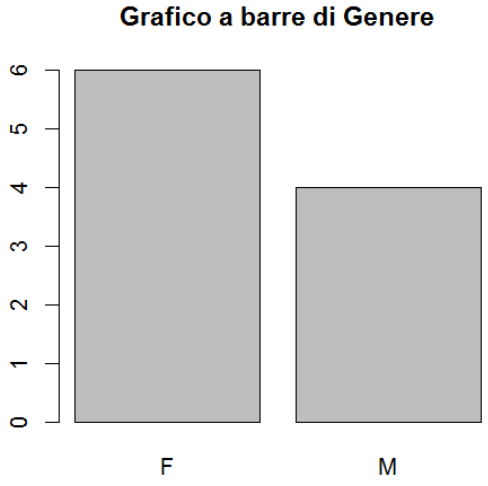

Figura 4.2: Grafico a barre

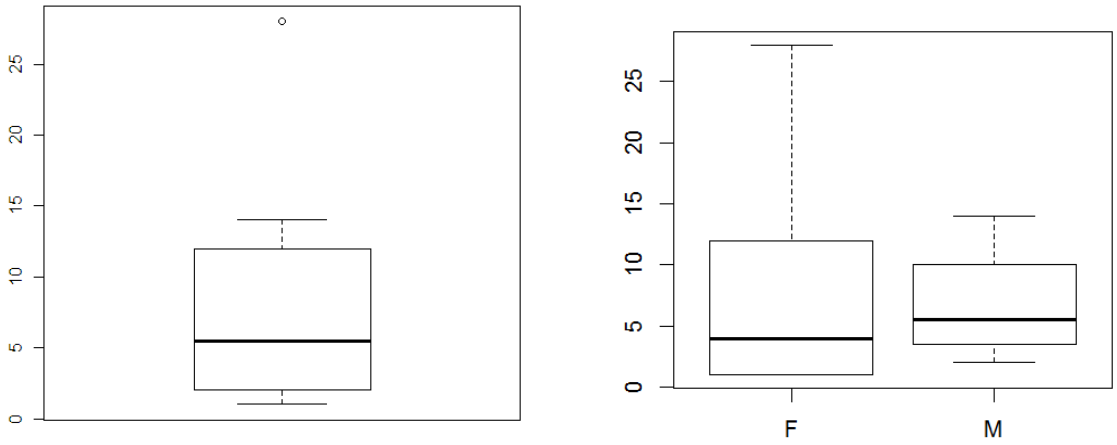

Figura 4.3: Grafico a scatola

Impostando `outline=` a falso, non vengono visualizzati gli outliers, mentre al posto del nome di un variabile, si può usare la modalità "formula" per suddividere le scatole in base ad una variabile di raggruppamento (Figura 4.3b):

```
R> boxplot(Dati1$Anni ~ Dati1$Genere) # oppure
R> with(Dati1, boxplot(Anni ~ Genere))
```

Capitolo 5

Statistica inferenziale di base

Con "statistica inferenziale di base" intendo i test per la differenza delle medie e alcuni test inferenziali non parametrici come il chi-quadrato.

5.1 Test di normalità

Molte tecniche di analisi statistica si basano sull'assunto di normalità. Al paragrafo 3.5 abbiamo visto le due statistiche di normalità che però ci indicano solo se [da FINIRE]. Shapiro-Wilk normality test, Kolmogorov – Smirnov test, D'agostino's K-squared test (D'Agostino-Pearson normality test), Jarque-Bera normality test, Anderson-Darling test for normality, Cramer-von Mises normality test, Lilliefors (Kolmogorov-Smirnov) test for normality, Shapiro-Francia normality test, Pearson chi-square test for normality

```
shapiro.test(x)
ks.test(x, "pnorm")

library(fBasics)
dagoTest(x)
library(tseries)
jarque.bera.test(x)
library(nortest)
ad.test(x)
library(nortest)
cvm.test(x)
library(nortest)
lillie.test(x)
library(nortest)
sf.test(x)
library(nortest)
pearson.test(x)
```

5.2 Differenze di medie

Il test sulla differenza delle medie si può applicare in 3 modalità. La sintassi generica del comando `t.test` è:

```
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

Il parametro `y` si usa solo nel confronto appaiato, in genere in associazione con `paired=T`, `mu` per il campione unico, mentre i parametri `alternative` e `var.equal` valgono in base ai singoli metodi. La procedura assume che la varianza fra i due gruppi (o variabili) non sia uguale (`var.equal=F`) e che l'ipotesi alternativa sia bidirezionale (`alternative = "two.sided"`).

Iniziamo gli esempi di confronto della media con quella relativa ad un campione confrontato con il parametro di una popolazione (usando il parametro `mu`):

```
R> t.test(Acomb$QUEST, mu=80)

        One Sample t-test

data:  Acomb$QUEST
t = 3.5441, df = 608, p-value = 0.0004242
alternative hypothesis: true mean is not equal to 80
95 percent confidence interval:
 81.88747 86.57886
sample estimates:
mean of x
 84.23317
```

Per il confronto di medie fra due campioni indipendenti è indispensabile usare la modalità formula "continua ~ dicotomica", indicando prima la variabile dipendente e poi l'indipendente, quindi indicando l'eventuale dataframe.

```
R> t.test(QUEST ~ Genere, data= Acomb, var=F)

        Two Sample t-test

data:  QUEST by Genere
t = 2.1688, df = 607, p-value = 0.03049
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  0.4960595 10.0039516
sample estimates:
mean in group 1 mean in group 2
       87.32800        82.07799
```

Nel caso di un test appaiato, bisogna indicare due variabili; se fanno parte di un dataframe, si possomno indicare in due modi:

```
R> with(Acomb, t.test(Authori, Malleab, pair=T))

        Paired t-test

data:  Authori and Malleab
t = 3.3549, df = 623, p-value = 0.0008422
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
 0.3953815 1.5116698
sample estimates:
mean of the differences
              0.9535256
R> t.test(Acomb$Authori, Acomb$Malleab, pair=T)

        Paired t-test

data:  Acomb$Authori and Acomb$Malleab
t = 3.3549, df = 623, p-value = 0.0008422
[...]
```

In queste procedure, il parametro `var.equal` permette di scegliere se applicare il procedimento per la varianza uguale o per la varianza diversa. Ma come possiamo sapere se i due gruppi hanno la stessa varianza?

```
R> var.test(QUEST ~ Genere, data= Acomb)

        F test to compare two variances

data:  QUEST by Genere
F = 1.0233, num df = 249, denom df = 358, p-value = 0.8381
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.8156588 1.2899501
sample estimates:
ratio of variances
          1.023301

> var.test(Acomb$Authori, Acomb$Malleab)

        F test to compare two variances

data:  Acomb$Authori and Acomb$Malleab
F = 1.0313, num df = 623, denom df = 623, p-value = 0.7011
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.8812452 1.2067918
sample estimates:
ratio of variances
          1.031251
```

5.3 [Chi quadrato]

[chi quadrato: chisq.test, vcd::assoc]
[test esatto di Fischer: fisher.test]
[wilcox.test, mcnemar.test, binom.test, HSAUR:xtabs]

Capitolo 6

Correlazioni

La correlazione è un indice di associazione e viene espressa con un numero che oscilla fra -1 e 1. Il segno del valore della correlazione indica la direzione dell'associazione, mentre il valore numerico indica l'intensità (zero indica la mancanza di legame, all'aumentare del valore aumenta l'associazione fino a valore 1 che indica una perfetta sovrapposizione).

L'indice più conosciuto è la correlazione prodotto-momento di Bravais-Pearson. Altre correlazioni generalmente indicate nei libri di testo sono la punto-biseriale e la correlazione a ranghi di Spearman. In tab. 6.1 sono indicati altri tipi di correlazione utilizzabili. Nella tabella, la dicitura "dicotomica vera" (oppure "ordinale vera") si riferisce ad eventi che sono effettivamente dicotomici (ad es. il genere, maschi vs. femmine) oppure ordinali, mentre la dicitura "dicotomica artificiale" si riferisce a variabili continue o discrete che vengono poi trasformate in (o misurate come) dicotomiche (ad es. le risposte vero/falso ad un'affermazione) oppure ordinali (come le scale tipo Likert).

Tabella 6.1: Diversi tipi di correlazione

	Coefficiente di correlazione	*Livelli di misurazione*
cor()	Prodotto-momento di Pearson	Entrambe intervallo
cor(met="spearman")	A ranghi di Spearman	Entrambe ordinali (oppure una intervallo "ranghizzata" e una ordinale)
cor(met="kendall")	Tau di Kendall	Entrambe ordinali
cor()	Punto-biseriale	Una intervallo e una vera dicotomica (0,1)
biserial()	Biseriale	Una intervallo e una dicotomica artificiale
	Phi	Entrambe dicotomiche
tetrachoric()	Tetracorica	Entrambe dicotomiche artificiali
polyserial()	Poliseriale	Una intervallo e una ordinale
polychor()	Policorica	Due ordinali artificiali

6.1 Pearson, Spearman e Kendall

La funzione cor() per il calcolo della correlazione (per default è quella di Pearson) funziona in modo analogo a var() o a cov(). Se passiamo due vettori (x e y), viene calcolata una singola correlazione, se passiamo più vettori (o un dataframe) viene calcolata una matrice di correlazioni (con 1 lungo la diagonale principale). La sintassi del comando è:

```
cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
```

Tramite il comando `cor()` sono immediatamente disponibili in R tre diverse formule per il calcolo della correlazione. La correlazione di Pearson richiede variabili quantitative (a livello intervallo/rapporto), la correlazione di Spearman è generalmente suggerita per variabili ordinali (ma è preferibile usarla per variabili intervallo che non si distribuiscono normalmente e che vengono trasformate in ranghi), mentre la correlazione di Kendall è pensata appositamente per variabili ordinali. La differenza principale fra Spearman e Kendall è nell'uso dei ranghi: Spearman usa i ranghi come punteggi di calcolo, mentre Kendall utilizza il numero di ranghi invertiti nella seconda variabile rispetto alla prima variabile.

Se non viene indicato il parametro `method=` si presume automaticamente la correlazione di Pearson.

```
R> cor(L$L1,L$L2)
[1] 0.09331741
R> round(cor(L[,1:5]),3)
       L1     L2     L3     L4     L5
L1  1.000  0.093  0.460 -0.463 -0.238
L2  0.093  1.000  0.427 -0.094 -0.244
L3  0.460  0.427  1.000 -0.604 -0.431
L4 -0.463 -0.094 -0.604  1.000  0.590
L5 -0.238 -0.244 -0.431  0.590  1.000
```

Il parametro `use=` serve per gestire i valori mancanti (`NA`): `use="all.obs"` usa tutte le osservazioni e genera un errore quando ci sono dei mancanti, `use="everything`" usa tutti i dati e pone a `NA` le correlazioni non calcolabili per via dei mancanti, `use="complete.obs"` usa solo le osservazioni con tutte le informazioni (eliminando i casi con valori mancanti), infine con `use="pairwise.complete.obs"` vengono ignorate le osservazioni con valori mancanti nel calcolo di una determinata correlazione ma vengono usati nelle altre correlazioni.

```
R> cor(Dati1)
        Cod Genere       Eta      Anni
Cod       1     NA        NA        NA
Genere   NA      1        NA        NA
Eta      NA     NA 1.0000000 0.5502608
Anni     NA     NA 0.5502608 1.0000000
Warning message:
In cor(Dati1) : si è prodotto un NA per coercizione
```

In questo esempio, non è possibile calcolare la correlazione di Pearson fra dati nominali o ordinali e quindi viene indicato `NA`.

Se utilizziamo la correlazione di Kendall, avremo invece:

```
R> cor(Dati1, method="kendal")
              Cod     Genere        Eta       Anni
Cod     1.0000000 -0.1825742 -0.4666667 -0.6440612
Genere -0.1825742  1.0000000 -0.1825742  0.1259882
Eta    -0.4666667 -0.1825742  1.0000000  0.4140393
Anni   -0.6440612  0.1259882  0.4140393  1.0000000
```

Per escludere la variabile `Cod` dobbiamo usare `Dati1[,2:4]` oppure (considerando che `Cod` è la prima variabile) `Dati1[-1]`.

6.2 Punto-biseriale

La correlazione punto-biseriale implica una variabile intervallo e una variabile dicotomica. In R non è disponibile una procedura specifica, ma è possibile usare `cor()` in quando è dimostrabile matematicamente che, se la variabile dicotomica è codificata con 0 e 1, la correlazione di Pearson coincide con quella punto-biseriale.

Usiamo come esempio le variabili `I1` e `D1` presenti nel file `dicot.xls`.

La formula della correlazione punto-biseriale è la seguente:

$$r_{pb} = \frac{\overline{X}_1 - \overline{X}_0}{s_X}\sqrt{\frac{N_1 N_0}{N(N-1)}}$$

dove $\overline{X}_1$ e $\overline{X}_0$ sono le medie della variabile intervallo X per i gruppi formati dalla variabile dicotomica Y, s_x è la deviazione standard di X, mentre N, N_1 e N_0 sono le numerosità del campione totale, del sottocampione che presenta il valore 1 o il valore 0 nella dicotomica Y.

```
R> library(psych) # se non già caricato
R> describe(dicot[,1])
  vars  n mean   sd median trimmed  mad min max range  skew kurtosis   se
1    1 20 13.5 7.41   14.5      14 7.41   0  23    23 -0.39    -1.04 1.66
R> describeBy(dicot[,1],dicot[,3])
group: 0
  vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
1    1 10 11.6 7.38     13   11.62 8.15   0  23    23 -0.1    -1.47 2.33
------------------------------------------------------------------------
group: 1
  vars  n mean   sd median trimmed  mad min max range skew kurtosis   se
1    1 10 15.4 7.31     16   16.38 7.41   0  23    23 -0.7    -0.57 2.31
```

Se applichiamo la formula manualmente, avremmo:

$$r_{pb} = \frac{15.4 - 11.6}{7.409098}\sqrt{\frac{10 * 10}{20(19)}} = 0.2631034$$

e usando la procedura `cor()` otteniamo lo stesso valore (a parità di decimali visualizzati):

```
R> cor(dicot[,1],as.integer(dicot[,3]))
          D1
I1 0.2631034
```

In realtà `as.integer(dicot[,3])` viene trasformata con i valori 1 e 2 (non 0 e 1), ma la correlazione non cambia e resterebbe uguale se togliessimo 1 o ne aggiungessimo un'altro (provare con `as.integer(dicot[,3])-1` oppure con `as.integer(dicot[,3])+2`).

6.3 Tetracoriche, poliseriali e policoriche

Le correlazioni tetracoriche si usano fra variabili dicotomiche (codificate con 0 o 1) nell'ipotesi che ciascuna delle variabili sia in realtà una latente continua che viene misurata dicotomizzata.

Analogo discorso se ipotizziamo che una variabile ordinale sia in realtà una variabile latente continua normale che viene poi approssimata a valori interi oppure misurata con valori interi (ad es. con le scale a ranghi tipo Likert), le correlazioni da usare sono la policorica (due ordinali) o la poliseriale (una intervallo e una ordinale).

Questo tipo di correlazioni sono disponibili nei pacchetti polycor di John Fox (2010) e psych di William Revelle (2014).

6.3.1 Correlazione tetracorica

La correlazione tetracorica si può calcolare con il comando tetrachoric() di psych o con il comando polychor() di polycor. Per l'applicazione, usiamo le due variabili dicotomiche del dataframe `dicot.xls`:

```
R> library(polycor)
R> table(dicot[3:4])
   D2
D1  0 1
  0 3 7
  1 7 3
R> polychor(table(dicot[3:4]))
[1] -0.5877944
R> library(psych)
R> tetrachoric(dicot[3:4])

tetrachoric correlation
   D1    D2
D1  1.00
D2 -0.59  1.00

 with tau of
D1 D2
 0  0
```

La funzione polychor() è pensata (6.3.2) per calcolare la correlazione tetracorica quando i dati sono entrambi dicotomici e la correlazione policorica quando sono entrambi ordinali e quindi può ricevere come primo parametro una tabella di contingenza 2x2, due variabili dicotomiche da un dataframe oppure due vettori di uguali dimensioni. Esempi equivalenti:

```
polychor(table(dicot[3:4]))
```

```
polychor(dicot[3:4])
polychor(dicot[,3],dicot[,4])
```

6.3.2 Correlazione policorica

La sintassi della correlazione policorica del pacchetto polycor è:

```
polychor(x, y, ML = FALSE, control = list(), std.err = FALSE, maxcor=.9999)
```

`x` e `y` devono essere 2 variabili ordinali oppure si può usare solo `x` e indicare una tabella di contingenza fra due variabili ordinali oppure due variabili di un dataframe. Con `ML` posto a falso viene usata una procedura veloce di stima, mentre se è posto a vero viene usata una stima di massimaverosimiglianza.

Immaginando un file di dati (in un dataframe chiamato `L`) contenente gli item di una scala Likert a 7 punti (chiamati `L1`, `L2` e così via), di cui prendiamo in considerazione solo i primi 2 item:

```
R> polychor(L$L1,L$L2)
[1] 0.08778793
R> table(L$L1,L$L2)

    1  2  3  4  5  6  7
  1 0  0  0  0  0  0  3
  2 5  0  0  0  4  2  4
  3 0  0  3  1  0  0  2
  4 2  0  3  0  0  0  2
  5 0  1  0  0  6  2  6
  6 5  3  1  1  0  8  5
  7 4  1  4  2  3 12 17
R> polychor(table(dati$L1,dati$L2))
[1] 0.08778793
```

6.3.3 Correlazione poliseriale

Sempre nel pacchetto polycor è disponibile il comando polyserial() che permette di calcolare una correlazione poliseriale, ovvero la correlazione fra una variabile quantitativa e una ordinale che sottintende una latente continua.

La sintassi della correlazione poliseriale è:

```
polyserial(x, y, ML = FALSE, ....)
```

In questo caso bisogna usare sia `x` sia `y`: `x` indica la variabile intervallo (continua) e `y` la variabile ordinale. Anche in questo caso è disponibile una stima veloce oppure tramite `ML`. Se le variabili non vengono inserite nell'ordine corretto (prima la continua e poi l'ordinale) bisogna usare la dicitura con i nomi per evitare che la correlazione non sia corretta.

```
R> polyserial(dicot[,1],dicot[,5])
[1] -0.3189208
R> polyserial(y=dicot[,5],x=dicot[,1])
[1] -0.3189208
R> polyserial(dicot[,5],dicot[,1]) # errata
```

```
[1] -0.3091761
```

6.3.4 Correlazione miste

Nello stesso pacchetto polycor è disponibile anche il comando hetcor() che permette di calcolare una matrice di correlazione a partire da dati misti: intervallo, ordinali, dicotomici. In base alla misura implicata dalla coppia di variabili verrà applicata la correlazione di Pearson oppure una delle tre correlazioni presenti nel pacchetto.

Purtroppo il meccanismo utilizzato per riconoscere il tipo di variabili, generalmente, riconosce le variabili numeriche (anche se ordinali o nominali) come intervallo e utilizza sempre Pearson. Quindi tutte le variabili nominale andrebbero preventivamente trasformate in "fattori" e quelle ordinali in "fattori" ordinati.

Usiamo il dataframe `dicot` che contenente 2 variabili intervallo (`I1` e `I2`), 2 ordinali (`O1` e `O2`) e due dicotomiche (`D1` e `D2`). Il comando hetcor(dicot) produrrà una matrice con correlazioni di Pearson.

```
R> hetcor(dicot)

Two-Step Estimates

Correlations/Type of Correlation:
        I1      I2       D1       D2      O1      O2
I1       1 Pearson  Pearson  Pearson Pearson Pearson
I2 -0.3438       1  Pearson  Pearson Pearson Pearson
D1  0.2631 -0.4145        1  Pearson Pearson Pearson
D2 -0.1385  0.1057     -0.4        1 Pearson Pearson
O1 -0.3367 -0.3524   0.3204 -0.08737       1 Pearson
O2  0.1221  0.1531 -0.07667    -0.23  -0.556       1
```

Se ricodifichiamo le ultime 4 variabili in fattori e chiamiamo nuovamente la procedura `hetcor()` otterremo invece:

```
R> dicot$D1 <- as.factor(dicot$D1)
R> dicot$D2 <- as.factor(dicot$D2)
R> dicot$O1 <- as.ordered(dicot$O1)
R> dicot$O2 <- as.ordered(dicot$O2)
R> hetcor(dicot)

Two-Step Estimates

Correlations/Type of Correlation:
        I1      I2         D1         D2         O1         O2
I1       1 Pearson Polyserial Polyserial Polyserial Polyserial
I2 -0.3438       1 Polyserial Polyserial Polyserial Polyserial
D1  0.3303 -0.5267          1 Polychoric Polychoric Polychoric
D2 -0.1766  0.1349    -0.5877          1 Polychoric Polychoric
O1 -0.3502 -0.4087     0.3928    -0.1323          1 Polychoric
O2 0.06084   0.171    -0.0731    -0.1932    -0.5807          1
```

Le ultime variabili sono adesso identificate come "fattori". Quando il calcolo della correlazione avviene fra due ordinali, viene usata la policorica (ad es. D1 con O1), fra due intervallo la correlazione di Pearson (I1 con I2) e fra un'ordinale e una intervallo la poliseriale (I1 con O2).

Il risultato di `hetcor()` è, in realtà, un oggetto che contiene diversi risultati: la matrice delle correlazioni (`correlations`), il tipo di correlazioni applicato (`type`), gli errori standard (`std.errors`), la numerosità (`n`), la probabilità associata ad ogni correlazione (`tests`). E lo vediamo se salviamo i risultati in una variabile e poi la stampiamo, usando ad esempio `hetcor(dicot)$correlations`

6.4 Rappresentazione grafica

È possibile rappresentare graficamente una matrice di correlazioni usando tre procedure differenti, che utilizzano rispettivamente grafica a caratteri, grafica ad ellissi e grafica a gradiente di colore. La rappresentazione grafica dovrebbe aiutare ad individuare le variabili molto o poco correlate fra loro.

Il comando `symnum()` (parte del pacchetto precaricato `stats`) permette di rappresentare una matrice di correlazioni usando simboli grafici (`.,+*B`) che rappresentano le correlazioni inferiori a 0.3, 0.6, 0.8, 0.9, 0.95. La sintassi completa del comando è un po' complessa, ma i parametri più importanti sono:

```
symnum(x, cutpoints = c(0.3, 0.6, 0.8, 0.9, 0.95),
       symbols = c(" ", ".", ",", "+", "*", "B"),
       lower.triangular = TRUE,
       diag.lower.tri   = FALSE
```

Il primo parametro dev'essere un oggetto di tipo matrice di correlazione.

```
R> symnum(cor(L))
   L1 L2 L3 L4 L5
L1 1
L2    1
L3 .  .  1
L4 .     ,  1
L5       .  .  1
attr(,"legend")
[1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
```

Possiamo notare che le correlazioni sono rappresentate in base al loro valore, ignorando il segno. Con il comando `symbols=c()` possiamo cambiare i caratteri da usare per la rappresentazione, che devono essere tanti quanti sono i punti di taglio indicati dal comando `cutpoints=c()`, che a sua volta può essere utilizzato per modificare il numero e il valore dei punto di taglio.

[da FINIRE].

Il pacchetto `ellipse` (Murdoch & Chow, 2007) permette di fare una rappresentazione grafica delle correlazioni tramite il comando `plotcorr()` che utilizza come primo parametro l'output di `cor()`. L'ellisse viene usata per rappresentare graficamente l'entità della correlazione, in senso metaforico rappresent la nuvola di punti di diagramma a dispersione, per cui un cerchio perfetto rappresenta la correlazione nulla ($r = 0$) mentre una piccola riga la correlazione perfetta ($r = 1$). L'orientamento dell'ellisse indica le correlazioni positive (L3 con L1) o negative (L4 con L3).

```
R> library(ellipse)
```

```
R> plotcorr(cor(L))
R> plotcorr(cor(L),col='lightgreen',type='lower')
```

Il primo comando `plotcorr` dell'esempio produce il grafico di figura 6.1a. Con il parametro `col=` possiamo scegliere il colore da utilizzare al posto del grigio, con `type=` possiamo decidere se la rappresentazione dev'essere solo del triangolo inferiore (`lower`), superiore (`upper`) o completo (`full`, di default). Quando la richiesta non è full, non viene visualizzata la diagonale (v. figura 6.1b).

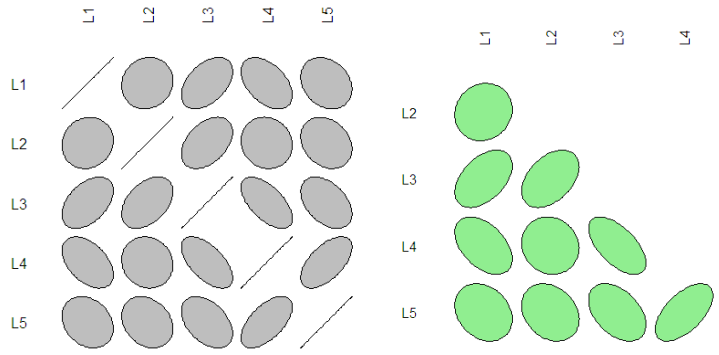

Figura 6.1: Rappresentazione grafica delle correlazioni di L

Nel pacchetto `psych` è disponibile il comando `cor.plot()` che utilizza invece dei quadrati colorati con saturazione differente di colori fra il rosso (correlazione negativa) e il blu (correlazione positiva). Più il colore è intenso più la correlazione è elevata (in valore assoluto). Vediamo un esempio:

```
R> library(psych) % se non caricato
R> cor.plot(cor(L))
```

Il comando dell'esempio precedente visualizza il grafico di Figura 6.2a. La correlazione fra `L1` e `L2` è praticamente nulla (r=.09) e viene visualizzata come bianca, mentre quella di `L3` con `L1` (leggermente più alta di quella con `L2`) usa un azzurro leggermente più intenso di quell con `L2`. E le correlazioni negative con le ulyime due variabili vengono visualizzate in rosso. È anche possibile visualizzare l'entità della correlazione visualizzata (moltiplicata per 100) tramite il parametro `numbers=T` (cfr. Figura 6.2b).

6.5 Inferenza sulla correlazione

La procedura `cor()` calcola la correlazione ma non produce nessuna informazione di tipo inferenziale. Per sapere se una correlazione è significativa, bisogna usare il comando `cor.test()`.

```
R> cor.test(L$L1,L$L3)

        Pearson's product-moment correlation

data:  L$L1 and L$L3
t = 5.4345, df = 110, p-value = 3.336e-07
alternative hypothesis: true correlation is not equal to 0
```

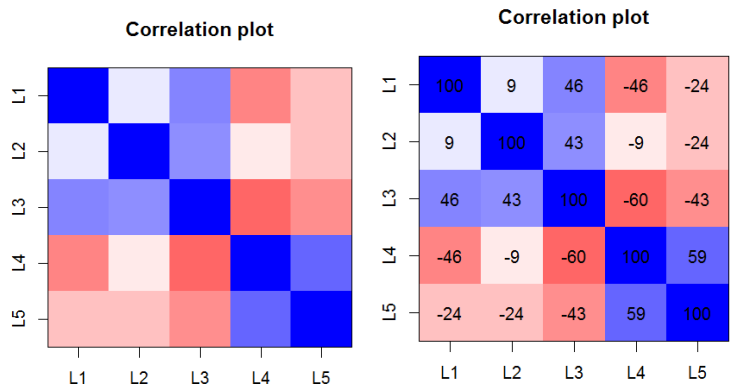

Figura 6.2: Rappresentazione delle correlazioni tramite cor.plot()

```
95 percent confidence interval:
 0.3001328 0.5948421
sample estimates:
      cor
0.4600668
```

Si possono usare dei parametri per cambiare il tipo di ipotesi (`alternative=`, il default è l'ipotesi bidirezionale), il metodo di correlazione (per default è `method='pearson'`, ma può essere Spearman o Kendall) o modificare l'intervallo di fiducia (`conf.level=`).

Questo comando pur visualizzando molte informazioni, non è comodo da usare in quanto richiede rigidamente di indicare due variabili nei due parametri `x` e `y`, vale a dire che non è possibile scrivere qualcosa tipo `cor.test(L[c('L1','L3')])` né tantomeno `cor.test(L[1:3])`.

Il pacchetto `psych` contiene un comando `corr.test()` (con 2 'r') che può lavorare con più variabili contemporaneamente e visualizza una matrice delle correlazioni, la matrice delle numerosità e quella delle probabilità associate (in quest'ultima, 0 significa $p < .001$):

```
R> corr.test(L[3:5])
Call:corr.test(x = L[3:5])
Correlation matrix
      L3    L4    L5
L3  1.00 -0.60 -0.43
L4 -0.60  1.00  0.59
L5 -0.43  0.59  1.00
Sample Size
    L3  L4  L5
L3 112 112 112
L4 112 112 112
L5 112 112 112
Probability values (Entries above the diagonal are adjusted for multiple tests.)
   L3 L4 L5
L3  0  0  0
L4  0  0  0
L5  0  0  0
```

La matrice delle probabilità, potrebbe presentare valori differenti nelle due metà, in quanto il triangolo inferiore alla diagonale visualizza la probabilità corrispondente al test di significatività, mentre quello superiore presenta i valori corretti con tecniche per il controllo dei test multipli.

Una seconda forma del comando, permette di confrontare determinate variabile verso altre. In questo esempio vengo richieste le correlazioni fra le prime 3 variabili verso le ultime 2:

```
R> corr.test(L[1:3],L[4:5])
Call:corr.test(x = L[1:3], y = L[4:5])
Correlation matrix
      L4    L5
L1 -0.46 -0.24
L2 -0.09 -0.24
L3 -0.60 -0.43
Sample Size
    L4  L5
L1 112 112
L2 112 112
L3 112 112
Probability values  adjusted for multiple tests.
     L4   L5
L1 0.00 0.03
L2 0.33 0.03
L3 0.00 0.00
```

Capitolo 7

Concordanza fra giudici

7.1 K di Cohen

Quando si vuole verificare l'accordo nei giudizi di due giudici, il pacchetto `psy` (Falissard, 2009) mette a disposizione i comandi `ckappa()` e `wkappa()` a cui bisogna passare 2 variabili (in genere due colonne di un dataframe). Entrambi restituiscono la tabella di contingenza dei valori e l'indice *kappa*: il primo considera le variabili come nominali mentre il secondo le considera ordinali. In entrambi i casi, i valori mancanti vengono eliminati prima di produrre la tabella.

```
R> ckappa(L[,c(1,2)])
$table
  1 2 3 4 5  6  7
1 0 0 0 0 0  0  3
2 5 0 0 0 4  2  4
3 0 0 3 1 0  0  2
4 2 0 3 0 0  0  2
5 0 1 0 0 6  2  6
6 5 3 1 1 0  8  5
7 4 1 4 2 3 12 17

$kappa
[1] 0.1178431

R> wkappa(L[,c(1:2)])
$table
  1 2 3 4 5  6  7
1 0 0 0 0 0  0  3
2 5 0 0 0 4  2  4
3 0 0 3 1 0  0  2
4 2 0 3 0 0  0  2
5 0 1 0 0 6  2  6
6 5 3 1 1 0  8  5
7 4 1 4 2 3 12 17

$weights
[1] "squared"

$kappa
```

```
[1] 0.09126494
```

Se abbiamo più di due giudici, il comando `lkappa()` del pacchetto `psy` calcola un indice *kappa* che è la media di tutte le coppie di *kappa* possibili. Il comando riceve in input 2 o più variabili e può avere un parametro: `type="Cohen"` (impostato per default) considera i valori come nominali, mentre impostandolo a vuoto (`type=""`) oppure a `weighted`, le variabili sono considerate ordinali. Se si passano solo due variabili, il risultato di `lkappa()` è identico a `ckappa()` o a `wkappa()`.

```
R> lkappa(L[,c(1:4)])
[1] 0.06219241
R> lkappa(L[,c(1:4)],type="")
[1] 0.09164206
R> lkappa(L[,c(1:4)],type="weighted")
[1] 0.09164206
```

Parte III

Analisi multivariata

Capitolo 8

Affidabilità

8.1 Coerenza interna

Una volta applicata l'analisi fattoriale esplorativa (v. 9.2) e indentificati i fattori, uno dei passi successivi è quello della stima della coerenza interna. Quindi, questo paragrafo dovrebbe essere affrontato dopo l'analisi fattoriale. In realtà, in molti casi, noi vogliamo calcolare la coerenza interna di una o più scale anche se non abbiamo effettuato un'analisi fattoriale, magari semplicemente per confrontare i dati italiani con quelli di un'altra nazione o di un altro ricercatore.

L'indice più conosciuto per il calcolo della coerenza interna (*reliability*) è l'alfa di Cronbach, che in R è disponibile nei pacchetti `psy` e `psych`. Per gli esempi che seguono userò il dataframe `Quest` composto da 16 item (a 9 gradi) di cui gli item 2, 3, 5, 8, 10, 11, 13 e 15 sono da ribaltare.

Nel pacchetto `psy` (Falissard, 2009) è disponibile una funzione chiamata `cronbach()` che richiede come solo parametro una matrice di dati oppure un dataframe:

```
R> # aggiustare il percorso
R> load("Quest.rda") # se non è ancora stato caricato
R> library(psy) # carico la libreria
R> cronbach(Quest[-1]) # tutte le variabili, meno la prima (NewCod)
$sample.size
[1] 690

$number.of.items
[1] 15

$alpha
[1] 0.5570438
```

La procedura `cronbach()` ha automaticamente cancellato 9 osservazioni perché contenevano dei mancanti e quindi corrisponde al comando cronbach(na.omit(Quest[-1])). L'alfa ottenuto è basso perché gli item non sono stati ribaltati. Per ovviare al problema, siamo costretti a creare un nuovo file dati oppure invertirli al volo mentre passiamo il dataframe `Quest`. Ricordiamo che la variabile in posizione 1 di `Quest` è `NewCod` e non ci interessa:

```
R> Quest2 = cbind(Quest[,c(1,4,6:7,9,12,14,16)+1],
+ 10-Quest[,c(2:3,5,8,10:11,13,15)+1])
R> cronbach(Quest2)$alpha
```

```
[1] 0.888969
```

In questo caso, abbiamo creato un nuovo file dati (`Quest2`) e, usando le funzioni `cbind()` (incolla per colonne) e `c()` abbiamo selezionato prima gli item pro-tratto e quindi quelli contro-tratto che abbiamo ribaltato, usando l'algoritmo `10-x` (ricordiamo che gli item di `Quest` oscillano da 1 a 9).

L'alfa aumenta notevolmente.

Decisamente più utile la funzione `alpha()` disponibile nel pacchetto `psych` (Revelle, 2014) che lavora su una matrice o un dataframe ma permette di ribaltare gli item senza necessità di creare un nuovo file dati. La sintassi, infatti è:

```
alpha(x, keys=NULL, cumulative=FALSE, title=NULL, max=10, na.rm = TRUE,
   check.keys=TRUE)
```

dove `x` può essere una matrice di dati o un dataframe (quindi osservazioni sulle righe e variabili nelle colonne) ma anche una matrice di varianze/covrianze o di correlazioni (quindi solo variabili).

```
R> library(psych) # carico la libreria
R> alpha(Quest[,-1],check=F)$total # stampo solo gli alfa
 raw_alpha std.alpha   G6(smc)  average_r     mean        sd
 0.5598928 0.5543032 0.7312169 0.07656382 4.901669 0.9558224
```

Il parametro `keys` ci permette di ribaltare gli item direttamente, in quanto è un vettore lungo quanto gli item da analizare; se un elemento vale 1, l'item è usato così com'è, se vale -1 viene ribaltato (moltiplicando la variabile per -1):

```
R> keys=rep(1,15) # vettore con 15 volte 1
R> keys[c(2, 3, 5, 8, 10, 11, 13, 15)]=-1 # ribalto alcuni item
R> alpha(Quest[-1],keys)$total
 raw_alpha std.alpha   G6(smc) average_r     mean       sd
  0.888969 0.8835341 0.9069158 0.3216373 4.879828 0.894488
R> alpha(Quest2)$total # qui sono già ribaltati
 raw_alpha std.alpha   G6(smc) average_r     mean       sd
  0.888969 0.8835341 0.9069158 0.3216373 5.142883 1.566461
```

Tuttavia questa procedura non è più necessaria, perché dalla versione 1.1.10, questo package ribalta automaticamente le variabili che correlano negativamente con il totale della scala (`check=T` per default). Ovviamente, questo procedimento è ottimo per scoprire la struttura interna di una scala monodimensionale, ma non per verificare l'alfa di Cronbach su una scala di cui si conosce (e si dovrebbe riprodurre) la struttura pro- o contro-tratto, per cui l'uso di `keys` è ancora molto utile.

Anche l'output di questa procedura è più informativo. Le prime informazioni sono relative all'alfa calcolato tramite i dati (`raw_alpha`), poi quello calcolato tramite matrice di covarianza o correlazione (`std.alpha`, spesso identico) poi la stima di Guttman chiamata *lambda 6* (`G6`), la media delle correlazioni (`average_r`), la media e la deviazione standard della scala ottenute sommando gli item. Confrontando i risultati ottenuti con i dati invertiti (`Quest2`) con quelli ottenuti con `alpha()`, possiamo osservare che le uniche diversità sono proprio su `mean` e `sd`, in quanto `alpha()` si limita a invertire il segno, la qual cosa altera la media e la deviazione standard della variabile, ma lascia inalterata la matrice delle correlazioni.

I risultati della procedura `alpha()` sono suddivisi in 4 blocchi: `total` (che abbiamo già visto e descritto), `alpha.drop`, `item.stats` e `response.freq`. Se chiamati singolarmente (come negli esempi precedenti, visualizzano tutti i decimali possibili, mentre se non vengono indicati la procedura stampa automaticamente le informazioni arrotondate a 2 decimali.

Nel gruppo `alpha.drop` viene mostrato come cambiano le statistiche quando si cancella un item. Ad es. cancellando l'item `Quest1`, l'alfa aumenta a .89, mentre cancellando l'item `Quest3`, l'alfa diminuisce a .86:

```
 Reliability if an item is dropped:
        raw_alpha std.alpha G6(smc) average_r
Quest1       0.89      0.89    0.90      0.36
Quest2-      0.87      0.86    0.89      0.31
Quest3-      0.86      0.86    0.88      0.30
Quest4       0.89      0.89    0.90      0.36
Quest5-      0.87      0.86    0.88      0.30
Quest6       0.87      0.86    0.89      0.31
Quest7       0.87      0.87    0.89      0.32
Quest8-      0.86      0.85    0.88      0.29
Quest9       0.87      0.86    0.89      0.30
Quest10-     0.87      0.86    0.88      0.30
Quest11-     0.86      0.85    0.88      0.29
Quest12      0.87      0.86    0.89      0.31
Quest13-     0.87      0.86    0.89      0.31
Quest14      0.87      0.87    0.89      0.32
Quest15-     0.86      0.85    0.88      0.30
```

Nel gruppo `item.stats` vengono fornite le statistiche sui singoli item: la numerosità, la correlazione con il totale della scala, la correlazione corretta (parzializzando le correlazioni degli altri item), la correlazione dell'item con il totale della scala tolto il contributo dell'item, la media e la deviazione standard dell'item.

```
 Item statistics
           n    r r.cor  r.drop mean  sd
Quest1   699 0.15 0.060 0.02779  4.6 2.4
Quest2-  699 0.58 0.539 0.50351  5.1 2.6
Quest3-  699 0.71 0.692 0.65272  5.2 2.7
Quest4   699 0.12 0.025 0.00084  6.4 2.0
Quest5-  697 0.70 0.693 0.64456  4.5 2.6
Quest6   699 0.67 0.634 0.59339  4.3 2.6
Quest7   697 0.54 0.489 0.44628  5.1 2.5
Quest8-  698 0.79 0.789 0.74406  4.6 2.8
Quest9   699 0.68 0.647 0.60488  5.7 2.6
Quest10- 698 0.69 0.676 0.63436  5.3 2.6
Quest11- 699 0.77 0.770 0.72743  4.4 2.7
Quest12  699 0.63 0.591 0.55639  3.9 2.6
Quest13- 698 0.64 0.609 0.57451  4.4 2.6
Quest14  698 0.57 0.519 0.47913  5.7 2.4
Quest15- 697 0.76 0.762 0.71384  4.4 2.4
```

Nel gruppo `response.freq` vengono mostrate le proporzioni dei singoli valori di ciascun item (se sono meno di 20) e il numero dei mancanti.

```
Non missing response frequency for each item
           1    2    3    4    5    6    7    8    9 miss
Quest1  0.15 0.12 0.11 0.08 0.13 0.15 0.16 0.05 0.05    0
Quest2  0.10 0.11 0.13 0.08 0.12 0.10 0.12 0.10 0.13    0
Quest3  0.13 0.09 0.07 0.10 0.15 0.10 0.12 0.10 0.15    0
Quest4  0.04 0.02 0.05 0.04 0.13 0.17 0.24 0.17 0.15    0
Quest5  0.19 0.09 0.10 0.11 0.17 0.08 0.10 0.09 0.07    0
Quest6  0.23 0.10 0.08 0.08 0.14 0.12 0.10 0.08 0.06    0
Quest7  0.11 0.10 0.09 0.07 0.15 0.12 0.14 0.10 0.11    0
Quest8  0.19 0.13 0.09 0.09 0.11 0.07 0.10 0.09 0.13    0
Quest9  0.09 0.08 0.08 0.08 0.11 0.11 0.14 0.14 0.17    0
Quest10 0.14 0.07 0.06 0.08 0.14 0.14 0.12 0.10 0.14    0
Quest11 0.22 0.11 0.08 0.10 0.14 0.08 0.07 0.07 0.11    0
Quest12 0.27 0.14 0.10 0.09 0.16 0.05 0.08 0.06 0.06    0
Quest13 0.17 0.13 0.12 0.12 0.11 0.09 0.11 0.06 0.10    0
Quest14 0.07 0.07 0.06 0.09 0.13 0.13 0.18 0.14 0.13    0
Quest15 0.15 0.13 0.10 0.11 0.20 0.09 0.08 0.06 0.08    0
```

Nel package `psych` c'è anche la possibilità di testare uno strumento composto da più scale, anche se la cosa è leggermente complicata, perché bisogna usare 2 diversi comandi:

```
score.items(keys, items, totals = FALSE, ilabels = NULL,
  missing = TRUE,impute="median",  min = NULL, max = NULL,
  digits = 2,short=FALSE)
make.keys(nvars, keys.list, key.labels = NULL, item.labels = NULL)
```

con `make.keys` associamo le variabili alle scale e con `score.items` calcoliamo gli alfa.

Ipotizziamo di lavorare con il dataframe `Liht` (usare `load` per caricare `Liht.rda`) che contiene 15 variabili (tipo Likert a 7 punti) che forma una scala totale e 3 sottoscale. Il dataframe è già stato riordinato in modo che le prime 5 variabili formino la prima sottoscala e così via (5 per scala). Gli item 4, 5 e 9 sono contro-tratto.

Per creare l'associazione fra variabili e scale, dobbiamo creare una matrice con tante righe quanti sono le variabili (in questo caso, 15) e tante colonne quante sono le scale (in questo caso, 3 più il totale).

```
R> Liht.keys= make.keys(15, # 15 item
+   keys.list= list(c(1:3,-4,-5,6:8,-9,10:15), #totale scala
+   c(1:3,-4,-5), # prima sottoscala
+   c(6:8,-9,10), # seconda sottoscala
+   c(11:15)), # terza sottoscala
+   key.labels=c("Liht","Authori","Malleab","World"), #nome scale
+   item.labels=names(Liht)) # nome variabili
R> Liht.keys
    Liht Authori Malleab World
L1     1       1       0     0
L2     1       1       0     0
L3     1       1       0     0
L4    -1      -1       0     0
L5    -1      -1       0     0
L6     1       0       1     0
L7     1       0       1     0
L8     1       0       1     0
```

```
L9    -1       0      -1     0
L10    1       0       1     0
L11    1       0       0     1
L12    1       0       0     1
L13    1       0       0     1
L14    1       0       0     1
L15    1       0       0     1
```

A questo punto possiamo chiamare la funzione `score.items`:

```
R> score.items(Liht.keys, Liht)
Call: score.items(keys = Liht.keys, items = Liht)

(Unstandardized) Alpha:
      Liht Authori Malleab World
alpha 0.68     0.7     0.8  0.36

Average item correlation:
          Liht Authori Malleab World
average.r 0.12    0.32    0.45   0.1

 Guttman 6* reliability:
         Liht Authori Malleab World
Lambda.6 0.83     0.8    0.87  0.56

Scale intercorrelations corrected for attenuation
 raw correlations below the diagonal, alpha on the diagonal
 corrected correlations above the diagonal:
        Liht Authori Malleab World
Liht    0.68    1.00    1.17  0.51
Authori 0.69    0.70    0.58 -0.54
Malleab 0.86    0.44    0.80  0.03
World   0.25   -0.27    0.02  0.36

Item by scale correlations:
 corrected for item overlap and scale reliability
     Liht Authori Malleab World
L1   0.20    0.45    0.01 -0.12
L2   0.59    0.53    0.53 -0.08
L3   0.50    0.78    0.36 -0.41
L4  -0.33   -0.69   -0.16  0.39
L5  -0.53   -0.65   -0.47  0.32
L6   0.52   -0.01    0.57  0.49
L7   0.83    0.60    0.85 -0.07
L8   0.67    0.31    0.80  0.00
L9  -0.49   -0.42   -0.53  0.21
L10  0.69    0.49    0.74 -0.11
L11  0.19   -0.17    0.18  0.51
L12  0.35    0.20    0.18  0.43
L13  0.11   -0.08    0.02  0.42
L14 -0.32   -0.49   -0.27  0.33
L15 -0.29   -0.46   -0.24  0.31
```

Capitolo 9

Analisi fattoriale

Per analisi fattoriale si intende generalmente l'analisi fattoriale esplorativa (generalmente abbreviata per l'inglese con EFA) che è una tecnica per ridurre una matrice di dati a "fattori" sottostanti. La maggior parte delle volte, anche l'analisi delle componenti principali (PCA) viene trattata fra le analisi fattoriali esplorative perché serve allo stesso scopo (pur non essendo un'EFA).

Con il tempo, è stata sviluppata anche l'analisi fattoriale confermativa (CFA) che è invece una versione particolare di un modello di equazioni strutturali e che non serve per trovare una serie di variabili sottostanti ma per verificare che determinate varabili osservate siano spiegabili tramite una variable latente (quello che in EFA è chiamato "fattore").

Sappiamo che sotto il nome di analisi fattoriale, gli psicologi inseriscono tutte le tecniche di riduzione dei dati includendo anche le componenti principali e che l'uso prevalente negli articoli su riviste di psicologia è quello di presentare un'analisi in componenti principali con rotazione Varimax (dipende forse dal fatto che è il default di software statistici come l'SPSS?) con il nome "analisi fattoriale".

Vediamo quindi di quali metodi dispone R, suddividendoli in componenti principali, analisi fattoriale esplorativa e confermativa. Inoltre nella sezione dell'analisi esplorativa affronteremo anche la cosiddetta analisi parallela.

In modo nativo (ovvero tramite il pacchetto `stats` che è precaricato) sono disponibili in R una procedura per il calcolo delle componenti principali e per l'analisi fattoriale con il metodo della massima verosimiglianza. Parecchi altri metodi sono disponibili in svariati pacchetti.

9.1 Componenti principali

Una prima procedura per l'analisi delle componenti principali si chiama `princomp()` e si può usare tramite una "formula" oppure in modalità dataframe. La sintassi del modo "formula" è:

```
princomp(formula, data = NULL, subset, na.action, ...)
```

Con `data=` si indica dove sono disponibili i dati (in genere un dataframe), mentre `formula` usa le variabili presenti nei dati tramite la modalità `~VAR1+Var2...`; `subset=` permette di selezionare un sottoinsieme di casi statistici; `na.action` indica come gestire gli eventuali dati mancanti.

Ipotizzando che il dataframe `dati` contenga 95 variabili miste (come succede generalmente nei dati di un questionario) e 15 di queste siano item di un questionario chiamati progressivamente

da L1 a L15, i possibili comandi per il calcolo delle componenti (salvando i risultati in `dati.pca`) sono:

```
R> load('dati.rda') # se non sono già stati caricati
R> dati.pca=princomp(~L1+L2+L3+L4+L5+L6+L7+L8+L9+L10+L11+L12+L13+L14+L15,
  cor=TRUE, data=dati)
R> dati.pca
Call:
princomp(formula = ~L1 + L2 + L3 + L4 + L5 + L6 + L7 + L8 + L9 +
    L10 + L11 + L12 + L13 + L14 + L15, data = dati, cor = TRUE)

Standard deviations:
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
2.0732982 1.5302482 1.2005211 1.1391531 1.0386521 0.9535538 0.8670484 0.7988688
   Comp.9   Comp.10   Comp.11   Comp.12   Comp.13   Comp.14   Comp.15
0.7639072 0.6379847 0.5891101 0.5785687 0.4924560 0.4355032 0.3718604

 15  variables and  59 observations.
```

La modalità dataframe ha invece questa sintassi:

```
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL,
         subset = rep(TRUE, nrow(as.matrix(x))), ...)
```

In questo caso, `x` può essere sia una matrice sia un dataframe; `cor=` permette di richiedere l'analisi a partire da una matrice di correlazioni (`cor=T`) o di covarianza (`cor=F`); con `covmat=` si può indicare una matrice di covarianze da usare al posto di quella generata dai dati. Un esempio di utilizzo è:

```
R> dati.pc = princomp(dati[,11:25], cor=T)
```

che produce esattamente lo stesso risultato dell'esempio precedente.

Il comando `princomp()` non stampa le saturazioni ma solo gli autovalori sottoforma di deviazioni standard. Per avere le varianze (stampate dalla maggior parte dei programmi) basta elevarle al quadrato.

```
R> dati.pc$sdev^2
   Comp.1    Comp.2    Comp.3    Comp.4    Comp.5    Comp.6    Comp.7    Comp.8
4.2985654 2.3416594 1.4412508 1.2976698 1.0787982 0.9092648 0.7517730 0.6381914
   Comp.9   Comp.10   Comp.11   Comp.12   Comp.13   Comp.14   Comp.15
0.5835542 0.4070245 0.3470507 0.3347417 0.2425129 0.1896630 0.1382801
R> sum(princomp(dati[,11:25], cor=T)$sdev^2)
[1] 15
```

Per avere le saturazioni bisogna chiederle esplicitamente tramite un comando `print()` specificando (eventualmente) il numero dei decimali (`digits=`) oppure il livello minimo di saturazioni da tenere (`cut=`). Con le saturazioni si ottengono anche le percentuali di varianza spiegata e i relativi valori cumulati:

```
R> print(dati.pc$load,dig=3,cut=.25)

Loadings:
    Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9 Comp.10
L1           0.323 -0.510                                    -0.489 -0.274
L2  -0.323                 0.258         0.418                 0.396 -0.374
L3  -0.325  0.268                        0.265 -0.257                -0.252
L4          -0.347                               0.603
L5   0.326                 0.364  0.258                              -0.384
L6          -0.507                              -0.330  0.366
L7  -0.395
L8  -0.302 -0.314                 0.401                       -0.291
L9   0.289         -0.392                              0.653
L10 -0.361                                  -0.336  0.275
L11         -0.402                -0.638                       -0.369 -0.317
L12                -0.489 -0.431         0.386  0.307                0.274
L13                -0.516  0.462                -0.292 -0.413  0.254
L14                       -0.552                       -0.259  0.416 -0.471
L15                                0.432  0.508 -0.338
    Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
L1           0.280           0.299
L2  -0.306   0.380
L3   0.296  -0.530           0.265  -0.315
L4                           0.321  -0.423
L5          -0.347  -0.460  -0.294
L6  -0.281          -0.328   0.475
L7                           -0.496  -0.569
L8                   0.558           0.354
L9                   0.302
L10  0.604                           0.260
L11  0.302
L12         -0.312
L13                  0.274
L14
L15  0.298   0.354

               Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
SS loadings     1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000  1.000
Proportion Var  0.067  0.067  0.067  0.067  0.067  0.067  0.067  0.067  0.067
Cumulative Var  0.067  0.133  0.200  0.267  0.333  0.400  0.467  0.533  0.600
               Comp.10 Comp.11 Comp.12 Comp.13 Comp.14 Comp.15
SS loadings      1.000   1.000   1.000   1.000   1.000   1.000
Proportion Var   0.067   0.067   0.067   0.067   0.067   0.067
Cumulative Var   0.667   0.733   0.800   0.867   0.933   1.000
```

Con i risultati salvati si può visualizzare lo scree-test (sia a barre sia a linee). La versione a barre (Figura 9.1 a sx) si può ottenere sia con `plot()` sia con `screeplot()`, mentre quello a linee (Figura 9.1 a dx), solo con `screeplot()`.

```
R> plot(dati.pca)
R> screeplot(dati.pr) # identico a plot
R> screeplot(dati.pr, type="lines")
```

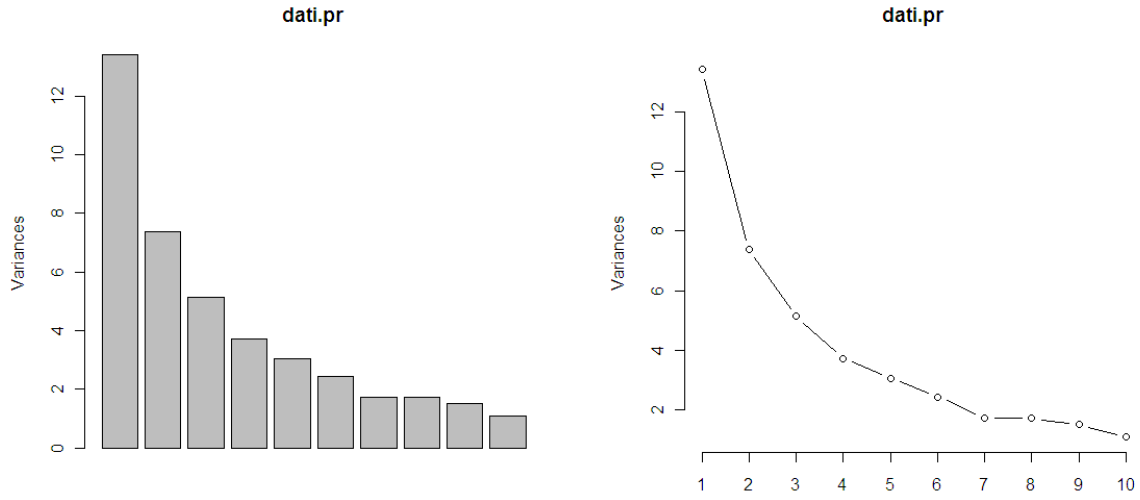

Figura 9.1: Scree plot

Esiste un analoga procedura (chiamata `prcomp`) che ha la stessa sintassi e restituisce gli stessi dati e che differisce solo per l'algoritmo con cui vengono estratte le componenti: tramite il metodo classico degli autovalori in `princomp` e tramite il metodo svd (*singular value decomposition*) per `prcomp`. Se lo scopo è di usare le componenti principali come un metodo di riduzione dei dati, è da preferire il secondo metodo, perché favorisce l'estrazione di componenti più complesse.

Per l'uso in psicologia, queste due procedure non hanno molta utilità, in quanto restituiscono informazioni (autovalori, saturazioni...) per tutte le componenti ovvero 15 variabili sottoposte ad analisi generano 15 componenti; inoltre non è prevista la rotazione degli assi.

Per avere una soluzione ruotata si devono estrarre le saturazioni con la funzione `loadings()` e passarli alle procedure `varimax()` o `promax()`.

Nel pacchetto `psych` è disponibile una procedura `principal()` che è decisamente più agevole da gestire. Tuttavia, dal momento che la sua sintassi e i suoi risultati sono simili a quelli delle procedure di analisi fattoriale, verrà affrontata al par.9.2.2.

9.1.1 Analisi parallela

L'analisi parallela è un ausilio statistico per la scelta del numero dei fattori. Va premesso che ci sono molti criteri diversi per scegliere il numero dei fattori, alcuni puramente arbitrari ed altri che fanno riferimento a criteri statistici.

Revelle (2013, pp. 51-52) indica i seguenti come i più utilizzati (l'ordine di presentazione è suo) e ciascuno ha dei vantaggi e degli svantaggi (traduzione "maccheronica" mia):

1. Estrarre fattori finché il chi-quadro della matrice dei residui diventa non significativo.
2. Estrarre fattori finché la differenza dei chi-quadri per n fattori e per n+1 fattori diventa non significativo.
3. Estrarre fattori finché gli autovalori dei dati reali sono inferiori ai corrispondenti fattori dei dati randomizzati (*analisi parallela*).
4. Rappresentare graficamente i valori degli autovalori e applicare lo *scree test*.
5. Estrarre fattori finché siano interpretabili.

6. Usare il criterio della Struttura molto semplice (*Very Simple Structure*, VSS).
7. Usare il criterio di Wayne Velicer delle medie minime parziali (*Minimum Average Partial*, MAP).
8. Estrarre componenti principali finché ci sono autovalori maggiori di 1.

Nei prossimi paragrafi ci occuperemo dell'analisi parallela e dei criteri VSS e MAP.

9.1.1.1 Analisi parallela in R

Nel pacchetto `paran` (Dinno, 2012) vi è una procedura con il medesimo nome che effettua un'analisi parallela usando i dati grezzi (la procedura non gestisce i mancanti, quindi dobbiamo usiamo la versione dei dati che non li contiene).

Se non abbiamo una versione dei dati senza valori mancanti e non vogliamo crearne una, basta usare `na.omit()` direttamente nel comando:

```
R> library(paran)
R> paran(na.omit(Quest[-1]))

Using eigendecomposition of correlation matrix.
Computing: 10%  20%  30%  40%  50%  60%  70%  80%  90%  100%

Results of Horn's Parallel Analysis for component retention
450 iterations, using the mean estimate

-------------------------------------------------
Component   Adjusted    Unadjusted    Estimated
            Eigenvalue  Eigenvalue    Bias
-------------------------------------------------
1           5.925310    6.183470      0.258159
2           1.597248    1.797821      0.200572
-------------------------------------------------

Adjusted eigenvalues > 1 indicate dimensions to retain.
(2 components retained)
```

L'analisi parallela viene svolta usando il metodo delle componenti principali e mostrando solo gli autovalori corrispondenti alle variabili da estrarre. Il parametro `graph=T` visualizza anche una rappresentazione grafica (Figura 9.2 a sx).

Se si vuole un'analisi parallela che utilizzi gli assi principali al posto delle componenti, basta specificare il parametro `cfa=T`:

```
R> paran(na.omit(Quest[-1]), cfa=T)
```

Sempre nel pacchetto `psych` vi sono due possibilità di scegliere il numero dei fattori: `fa.parallel()` e `VSS()`.

Nel primo caso, viene effettuata una vera e propria analisi parallela usando gli autovalori oppure con un approccio polinomiale. La sintassi del comando è:

```
fa.parallel(x, n.obs = NULL,fm="minres", fa="both",
    main = "Parallel Analysis Scree Plots",n.iter=20,
    error.bars=FALSE,SMC=FALSE,ylabel=NULL,show.legend=TRUE,sim=TRUE)
```

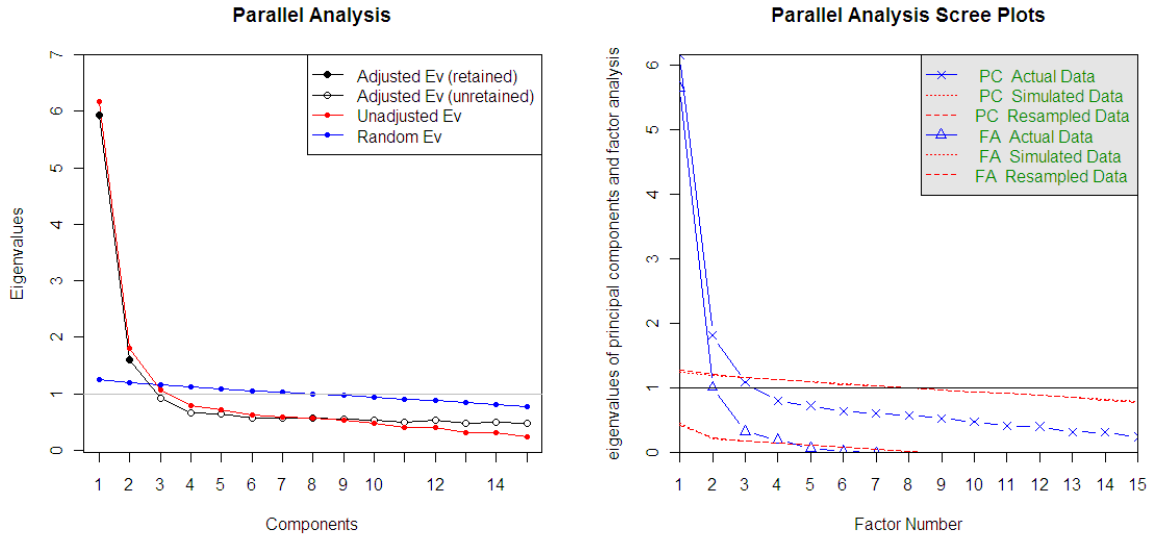

Figura 9.2: Analisi parallela: comandi `paran()` e `fa.parallel()`

```
fa.parallel.poly(x ,n.iter=10,SMC=TRUE,fm="minres",correct=TRUE,
    sim=FALSE, fa="both",global=TRUE)
```

Le due funzioni sono simili, ma la seconda lavora su dati dicotomici o polinomiali.

Questa funzione può essere usata con un dataframe oppure con una matrice di correlazione. Usando le correlazioni, è necessario indicare il numero dei casi statistici tramite il parametro `n.obs`.

Anche se è possibile indicare il metodo di estrazione dei fattori (con `fm`) i risultati non cambiano di molto.

```
R> library(psych) # se non caricato
R> fa.parallel(Quest[,-1])
Parallel analysis suggests that the number of factors =  4
and the number of components =  2
R> Quest.cor=polychoric(Quest[,-1])
R> fa.parallel(Quest.cor,n.obs=699,ntrials=100)
Parallel analysis suggests that the number of factors =  4
and the number of components =  2
```

La procedura `fa.parallel` visualizza anche un grafico (Figura 9.2 a dx) che però non è facilmente interpretabile, perché visualizza sia l'analisi parallela basata sulle componenti sia quella basata sui fattori (parametro `fa="both"`). Per renderlo più comprensibile, bisogna selezionare solo le componenti (con `fa="pc"`) o solo i fattori (con `fa="fa"`). Per vedere gli autovalori reali e simulati, bisogna salvare i risultati in una variabile e poi stamparli, oppure stamparli direttamente.

```
R> print(fa.parallel(Quest.cor,n.obs=699,ntrials=100))
Parallel analysis suggests that the number of factors =  4
and the number of components =  2
```

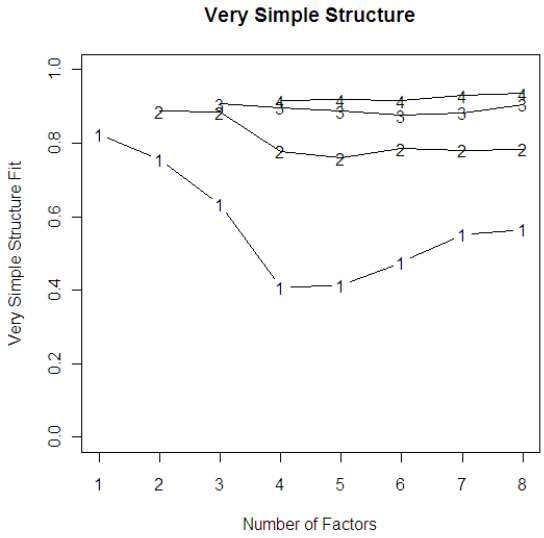

Figura 9.3: Analisi parallela: comando `VSS()`

```
Call: fa.parallel(x = Quest.cor, n.obs = 699, ntrials = 100)
Parallel analysis suggests that the number of factors =  4
and the number of components =  2

 Eigen Values of
  Original factors Simulated data Original components simulated data
1             6.71           0.39                 7.18           1.27
2             1.25           0.23                 1.98           1.21
3             0.37           0.18                 1.08           1.17
4             0.19           0.15                 0.83           1.13
```

La seconda procedura disponibile in `psych` è `VSS` che segue un procedimento diverso. Questa procedura calcola una soluzione fattoriale, ricostruisce la matrice di correlazione e la confronta con quella originale, calcolando un indice. Anche in questo caso si possono usare sia i dati grezzi sia una matrice di correlazioni (in tal caso bisogna indicare il numero dei casi statistici).

```
VSS(x, n = 8, rotate = "varimax", diagonal = FALSE, fm = "minres",
  n.obs=NULL, plot=TRUE, title="Very Simple Structure",\ldots{})
```

Il secondo parametro indica il numero massimo di fattori da verificare ed è automaticamente impostato ad 8, la rotazione predefinita è la varimax e il metodo di estrazione quello dei minimi quadrati. Anche questa procedura visualizza un grafico (Figura 9.3) poco comprensibile, mentre i risultati testuali sono più utili (*opinione assolutamente personale!*).

```
R> VSS(Quest[,-1],fm="pa")

Very Simple Structure
Call: VSS(x = Quest[, -1], n = 4, fm = "pa")
VSS complexity 1 achieves a maximimum of 0.83  with  1  factors
```

```
VSS complexity 2 achieves a maximimum of 0.9  with  2  factors

The Velicer MAP criterion achieves a minimum of 0.02  with  2  factors

Velicer MAP
[1] 0.03 0.02 0.02 0.03

Very Simple Structure Complexity 1
[1] 0.83 0.63 0.62 0.45

Very Simple Structure Complexity 2
[1] 0.00 0.90 0.89 0.81
```

Usando una matrice di correlazione, il comando sarà:

```
R> VSS(Quest.cor, n.obs=699, n=4)
```

9.2 Analisi fattoriale esplorativa

9.2.1 Massima verosimiglianza

Un'altra procedura di R calcola un'analisi fattoriale esplorativa usando la massimaverosimiglianza; la sintassi è:

```
factanal(x, factors, data = NULL, covmat = NULL, n.obs = NA,
         subset, na.action, start = NULL,
         scores = c("none", "regression", "Bartlett"),
         rotation = "varimax", control = NULL, ...)
```

Anche in questo caso, `x` può essere una "formula" o un dataframe (o meglio una matrice di numeri e in tal caso non serve indicare il dataframe con `data=`). Con `factors` indichiamo il numero di fattori che vogliamo estrarre. Il parametro `rotation=` permette di specificare il tipo di rotazione; quelle possibili sono `varimax` e `promax`.

```
R> factanal(dati[,11:25], factors=3) # varimax è il default
R> factanal(dati[,11:25], factors=3, rotation="promax")
```

I risultati che questa procedura visualizza, sono più utili delle precedenti e includono: l'unicità, le saturazioni (quelle inferiori a .100 non vengono visualizzate), la tabella riassuntiva della varianza e il chi-quadro sul numero dei fattori.

```
R> factanal(dati[,11:25], factors=3)

Call:
factanal(x = dati[, 11:25], factors = 3)

Uniquenesses:
   L1    L2    L3    L4    L5    L6    L7    L8    L9   L10   L11   L12
0.421 0.663 0.482 0.547 0.454 0.419 0.200 0.367 0.498 0.419 0.815 0.881
   L13   L14  L15
 0.806 0.8140.767
```

```
Loadings:
    Factor1 Factor2 Factor3
L1          -0.695   0.305
L2   0.573
L3   0.520  -0.492
L4  -0.394   0.544
L5  -0.692   0.214   0.147
L6   0.310   0.554   0.422
L7   0.718           0.527
L8   0.584   0.269   0.468
L9  -0.687  -0.111   0.131
L10  0.649           0.399
L11          0.421
L12                  0.339
L13 -0.193           0.395
L14 -0.375   0.215
L15 -0.372   0.308

                Factor1 Factor2 Factor3
SS loadings       3.401   1.786   1.260
Proportion Var    0.227   0.119   0.084
Cumulative Var    0.227   0.346   0.430

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 98.39 on 63 degrees of freedom.
The p-value is 0.00289
```

Scegliendo la rotazione `promax` vengono visualizzati gli stessi tipi di risultati e, in aggiunta, le correlazioni fra i fattori.

```
....

Factor Correlations:
        Factor1 Factor2 Factor3
Factor1   1.000   0.315   0.265
Factor2   0.315   1.000   0.247
Factor3   0.265   0.247   1.000

...
```

9.2.2 Pacchetto psych

Il pacchetto `psych` contiene una serie di soluzioni alternative che risolve in modo uniforme il problema dell'analisi fattoriale, perché mette a disposizione una procedura `principal` per le componenti principali e una generica procedura `fa` piuttosto versatile. Il vantaggio di queste procedure è l'uniformità delle sintassi e della presentazione dei risultati. Inoltre tutte le procedure calcolano un test di adeguatezza sul numero dei fattori, ricostruendo la matrice delle correlazioni a partire dalle saturazioni trovate e quindi applicando il test del chi-quadro sulle due matrici.

```
principal(r, nfactors = 1, residuals = FALSE, rotate="varimax", n.obs=NA,
   scores=FALSE, missing=FALSE, impute="median")
fa(r, nfactors=1, residuals = FALSE, rotate = "varimax", n.obs = NA,
```

```
    scores = FALSE, SMC=TRUE, missing=FALSE, impute="median",
    min.err = 0.001, max.iter = 50, symmetric=TRUE, warnings=TRUE,
    fm="minres", ...)
```

Con queste procedure si può fare un'analisi in componenti principali o un'analisi fattoriale esplorativa usando diversi metodi e usando sia una matrice di dati grezzi, sia una matrice di correlazione (in questo caso è necessario usare il parametro `n.obs` per indicare l'ampiezza del campione).

Il numero dei fattori si indica in seconda posizione (per decidere il numero dei fattori cfr. 9.1.1).

Il metodo di estrazione dei fattori si indica con il parametro `fm`: se `fm="pa"` viene usato il metodo degli assi principali, con `fm="ml"` la massimaverosimiglianza, con `fm="minres"` i minimi residui, con `fm="wls"` i minimi quadrati pesati, con `fm="gls"` i minimi quadrati pesati generalizzati.

Tabella 9.1: Metodi di estrazione disponibili in `psych`

parametro fm	metodi	comando abbreviato
pa	assi principali	factor.pa()
ml	massima verosimiglianza	
minres	minimi residui	factor.minres()
wls	minimi quadrati pesati	factor.wls()
gls	minimi quadrati pesati generallizzati	
minchi	minimizza il chiquadro pesato sull'ampiezza del campione	

Per semplificare le cose, esistono anche le procedure `factor.pa`, `factor.minres` e `factor.wls` che sono solo scorciatoie per i rispettivi metodi di estrazione.

I metodi di rotazione che è possibile utilizzare sono riassunti in tabella 9.2.

Esempi di uso del comando `fa` (il dataframe `Quest` è descritto al paragrafo 8.1):

```
> fa(Quest[,-1],nfactors=4, fm="pa")
Factor Analysis using method =  pa
Call: fa(r = Quest[, -1], nfactors = 4, fm = "pa")
        item   PA1   PA3   PA4   PA2   h2   u2
Quest1     1                    0.48 0.26 0.74
Quest2     2  0.65                   0.48 0.52
Quest3     3  0.66                   0.60 0.40
Quest4     4                    0.53 0.29 0.71
Quest5     5  0.50        0.61       0.70 0.30
Quest6     6        0.56             0.49 0.51
Quest7     7        0.59             0.41 0.59
Quest8     8  0.54 -0.40  0.46       0.68 0.32
Quest9     9 -0.37  0.55             0.48 0.52
Quest10   10  0.52 -0.37  0.30  0.32 0.60 0.40
Quest11   11  0.69 -0.39             0.66 0.34
Quest12   12        0.64             0.52 0.48
Quest13   13  0.61                   0.47 0.53
```

Tabella 9.2: Metodi di rotazione disponibili in `psych`

	Componenti principali	Altri metodi
ortogonali	none	none
	varimax	varimax
	quartimax	quartimax
		bentlerT
		geominT
obliqui	promax	promax
	oblimin	oblimin
	simplimax	simplimax
		bentlerQ
		geominQ
	cluster	cluster

```
Quest14   14        0.55             0.37 0.63
Quest15   15  0.57        0.57       0.75 0.25
Quest16   16        0.76             0.65 0.35

                PA1  PA3  PA4  PA2
SS loadings    3.21 3.02 1.24 0.93
Proportion Var 0.20 0.19 0.08 0.06
Cumulative Var 0.20 0.39 0.47 0.53

Test of the hypothesis that 4 factors are sufficient.

The degrees of freedom for the null model are 120
and the objective function was 7.42 with Chi Square of 5132.6
The degrees of freedom for the model are 62 and the objective function was 0.26
The number of observations was 699 with Chi Square = 178.75 with prob < 3e-13

Tucker Lewis Index of factoring reliability =  0.95
Fit based upon off diagonal values = 1
Measures of factor score adequacy
                                                PA1  PA3  PA4  PA2
Correlation of scores with factors             0.85 0.88 0.76 0.76
Multiple R square of scores with factors       0.72 0.77 0.58 0.57
Minimum correlation of factor score estimates 0.45 0.54 0.16 0.15
```

Volendo più decimali, si può salvare l'analisi in una variabile e usare la procedura `print()` con il parametro `digit=3`; analogamente se si vogliono visualizzare tutte le saturazioni o se si vuole cambiare il valore massimo delle saturazioni da non visualizzare, si può usare il parametro `cut`. Ecco un esempio senza risultati:

```
> fa(Quest[,-1],nfactors=4, fm="ml")->Quest.ml
> print(Quest.ml,cut=.020,digits=3)
> fa(Quest.cor,n.obs=699,nfactors=2, fm="ml")->Quest.ml2
```

```
> print(Quest.ml2, cut=.25)
```

Volendo vedere le saturazioni riordinate per fattore, si può usare la funzione `fa.sort()` sui risultati di un'analisi fatta con `fa` o con `principal`.

```
> fa.sort(Quest.ml)
```

Dal momento che `fa.sort()` restituisce un oggetto dello stesso tipo di `fa`, si può usare contemporaneamente `fa.sort()` e `print()`:

```
> print(fa.sort(Quest.ml), cut=.25)
```

È possibile anche avere una rappresentazione grafica dei fattori e dei relativi item associati (a imitazione dei diagrammi di Lisrel) con il comando `fa.diagram()`:

```
> fa.diagram(Quest.ml)
```

questa funzione, automaticamente utilizza gli item monofattoriali più saturi di ogni fattore (`simple=T`) e riordina gli item in modo da migliorare la grafica (`sort=T`), visualizzando quindi soluzioni congeneriche e azzera tutte le saturazioni inferiori a .30. Gli item con saturazione negativa vengono visualizzati in rosso; se la soluzione è obliqua, vengono rappresentate anche le correlazioni fra i fattori. Un esempio di applicazione è presentata nel prossimo paragrafo (9.2.3).

Ovviamente è possibile anche utilizzare alcuni parametri: `simple=F` non elimina le variabili multifattoriali; `cut=` permette di cambiare la soglia di azzeramento; `sort=F` non riordina le dimensioni in base alle saturazioni.

9.2.3 Un esempio completo

Iniziamo con il file `Quest` che è una scala che misura la religiosità di ricerca. Secondo gli autori (Altemeyer e Hunsberger, 1992) è una sola scala (monofattoriale) composta da 16 item (a 9 gradi tipo Likert) di cui gli item 2, 3, 5, 8, 10, 11, 13 e 15 sono da ribaltare.

Abbiamo visto che l'analisi parallela suggerisce due componenti o quattro fattori. Dal momento che, se esistono più fattori, essendo parte di una scala unica, dovrebbero essere correlati, facciamo quindi un'analisi fattoriale esplorativa sia con 2 che con 4 fattori, usando gli assi principali e la massimaverosimiglianza entrambe con rotazione promax e poi chiediamo una rappresentazione grafica. Tutte le analisi le facciamo con una matrice di correlazione policorica (indico anche il caricamento dei pacchetti, per semplicità; non è però necessario caricarli se lo avete già fatto).

```
> load("Quest.rda")
> Quest.cor=poly.mat(Quest[,-1])
> Quest.pa2=fa(Quest.cor,n.obs=699,nfactors=2, fm="pa", rot="Promax")
> Quest.pa4=fa(Quest.cor,n.obs=699,nfactors=4, fm="pa", rot="Promax")
> Quest.ml2=fa(Quest.cor,n.obs=699,nfactors=2, fm="ml", rot="Promax")
> Quest.ml4=fa(Quest.cor,n.obs=699,nfactors=4, fm="ml", rot="Promax")
> fa.diagram(Quest.pa2,digit=2,main="Quest - assi principali 2 fattori Promax")
> fa.diagram(Quest.ml2,digit=2,main="Quest - ML 2 fattori Promax")
> fa.diagram(Quest.pa4,digit=2,main="Quest - assi principali 4 fattori Promax")
> fa.diagram(Quest.ml4,digit=2,main="Quest - ML 4 fattori Promax")
```

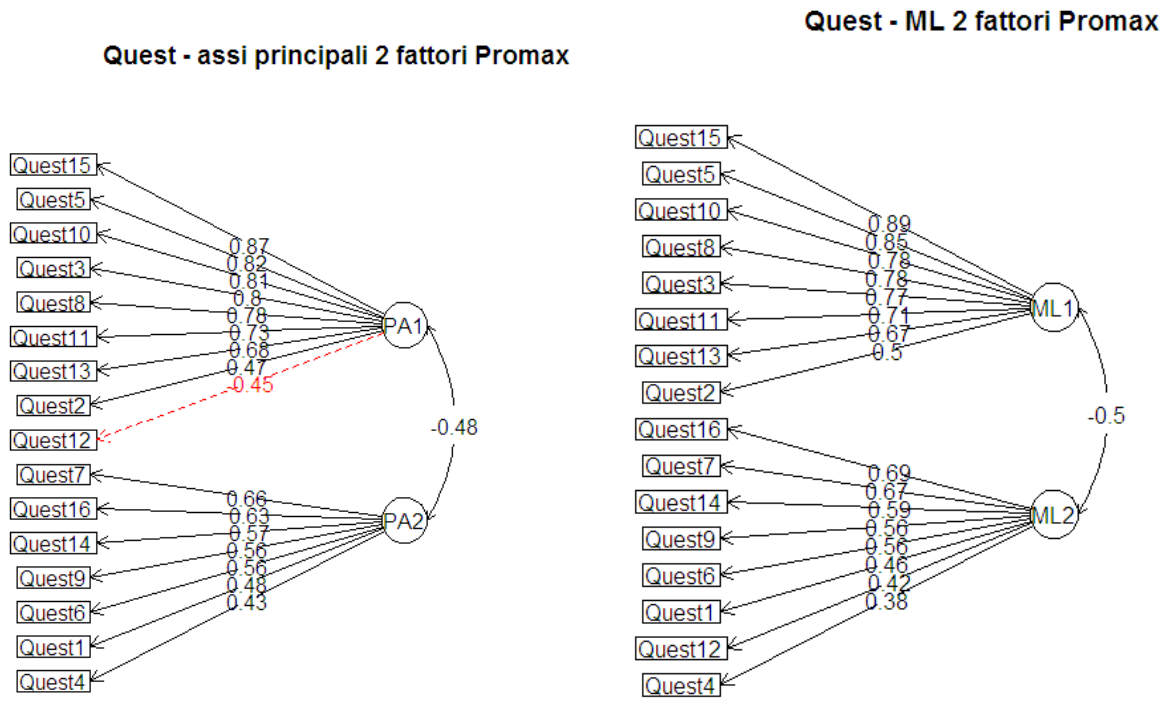

Figura 9.4: Quest - 2 fattori estratti

Confrontando le soluzioni a 2 fattori (Figura 9.4), vediamo che un fattore raccoglie gli item contro-tratto (con in più l'item 12) e l'altro quelli pro-tratto. Entrambi i fattori sono molto correlati fra loro.

Confrontando invece le soluzioni a 4 fattori (Figura 9.5), vediamo di nuovo una buona somiglianza della struttura fattoriale: alcuni fattori contro-tratto nel primo fattore, alcuni pro-tratto nel secondo, mentre gli ultimi due raccolgono pochissimi item.

Siamo di fronte alla tipica situazione di una soluzione monofattoriale in cui i due fattori spiegano la direzione della risposta.

9.3 Analisi fattoriale confermativa

L'analisi fattoriale confermativa è parte dei più generici "modelli di equazioni strutturali" o SEM (da Structural Equation Model), ma la tratto qui per omogeneità con l'argomento. I modelli SEM, in R, si possono calcolare tramite i pacchetti `sem` (Fox, with contributions from Kramer, & Friendly, 2010), `lavaan` (Rosseel, with contributions of Oberski, & Byrnes, 2011) o `OpenMx` (Boker et al., 2011, 2012).

I modelli di CFA possono essere simili all'esplorativa (tutti dati standardizzati) oppure più simili alle procedure SEM.

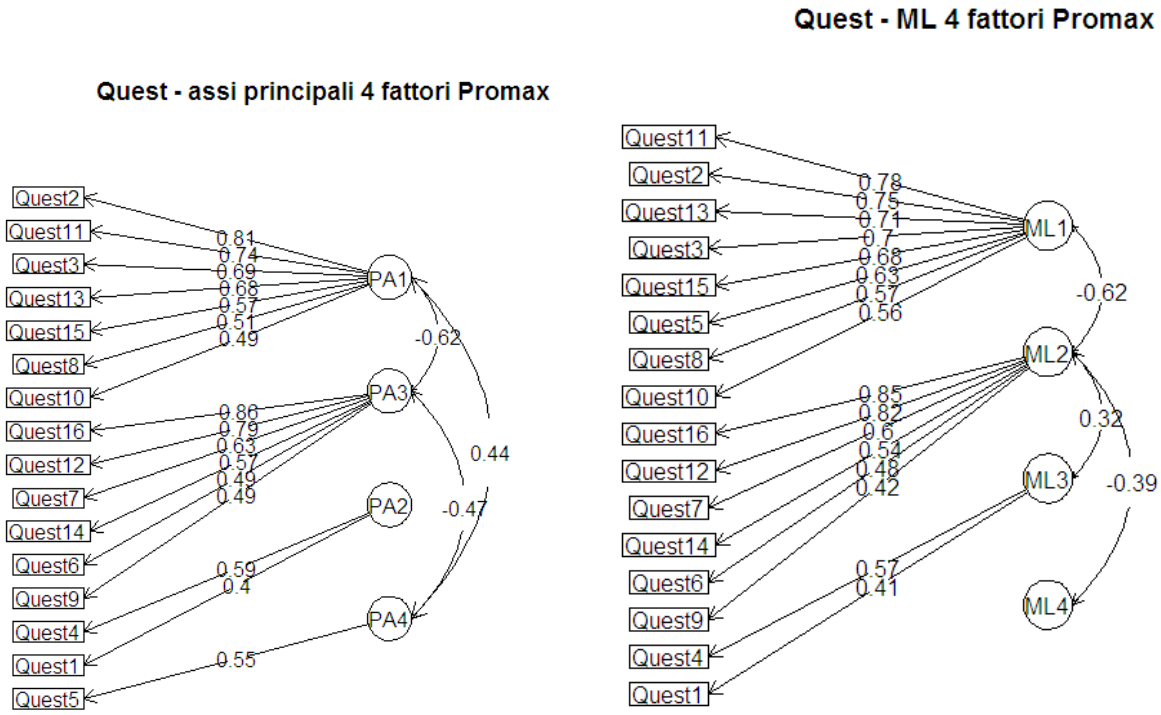

Figura 9.5: Quest - 4 fattori estratti

9.3.1 Pacchetto sem

Questo pacchetto è pensato per persone abituate a lavorare con le equazioni strutturali e che abbia familiarità con la notazione relativa: gamma (Γ), beta (B), lambda x (Λ_x), ecc.

Accanto al più generico comando `sem`, il pacchetto contiene un comando `cfa` che esegue un'analisi fattoriale confermativa. Tuttavia, non può essere usato direttamente e bisogna prima preparare i dati e il modello su cui lavorare. Vediamo le regole di rappresentazione usate dai comandi `sem` e `cfa` di questo pacchetto. Nei modelli di `sem`, l'influenza di una variabile su un'altra è indicata con '`->`', mentre la relazione reciproca (varianza, covarianza) con '`<->`'.

In questo senso, se l'osservata `A` viene inflenzata/spiegata dalla latente `X`, la relazione si scriverà come '`X -> A`', mentre la covarianza tra le latenti `X` e `Y` si scriverà '`X <-> Y`' o '`Y <-> X`'.

Usando il dataframe Liht, costruiamo una analisi fattoriale confermativa per le tre sottoscale Autorità, Religione, Mondo a cui vengono associati (nel nostro database) gli item da 1 a 5, da 6 a 10 e da 11 a 15. La struttura, in `sem` può essere descritta in 3 modi. Il primo è tramite un modello come il seguente, usando il comando `specifyModel()` (un riga vuota chiude l'immissione):

```
R> Liht.model <- specifyModel()
A -> L1,  NA, 1
A -> L2,  lam21, NA
A -> L3,  lam31, NA
A -> L4,  lam41, NA
A -> L5,  lam51, NA
R -> L6,  NA, 1
R -> L7,  lam72, NA
R -> L8,  lam82, NA
R -> L9,  lam92, NA
R -> L10,  lam102, NA
M -> L11,  NA, 1
M -> L12,  lam123, NA
M -> L13,  lam133, NA
M -> L14,  lam143, NA
M -> L15,  lam153, NA
A <-> R, psi21
A <-> M, psi31
R <-> M, psi32
```

Il fattore A spiega gli item da L1 a L5 (e così via) e questi legami sono legati a dei parametri da stimare (tramite la notazione `->`) qui indicati come `lamXX` (il nome da usare è arbitrario; in questo caso ho seguito la convenzione usata dal software Lisrel) e per i primi item di ogni fattore, si imposta il valore a 1, per tutti gli altri si lascia non definito. Infine impostiamo le covarianze fra i fattori con la notazione (`<->`).

È problematico riuscire ad inserire un modello di questo tipo senza commettere errori di immissione, per cui si può scrivere il modello in un file di testo (ad es. `Lihtmodel.txt`) e poi caricarlo.

```
R> Liht.model <- specifyModel(file='Lihtmodel.txt')
```

In alternativa il modello può essere indicato tramite delle espressioni:

```
R> L1 = 1*A
R> L2 = lam21*A
R> L3 = lam31*A
R> L4 = lam41*A
R> L5 = lam51*A
R> L6 = 1*R
R> L7 = lam72*R
R> L8 = lam82*R
R> L9 = lam92*R
R> L10 = lam102*R
R> L11 = 1*M
R> L12 = lam123*M
R> L13 = lam133*M
R> L14 = lam143*M
R> L15 = lam153*M
```

```
R> library(sem)
```

[da FINIRE].

9.3.2 Pacchetto lavaan

In lavaan il modello SEM per una CFA si scrive in questo modo:

```
A =~ L1 + L2 + L3 + L4 + L5
R =~ L6 + L7 + L8 + L9 + L10
M =~ L11 + L12 + L13 + L14 + L15
```

dove A, R e M indicano le variabili latenti (non esistono nel dataframe) e L1... L15 sono le variabili osservate (e devono esistere nel dataframe). Il comando `=~` indica che la variabile latente (cioè il fattore) è "misurato" tramite le variabili indicate. Automaticamente la prima osservata di ogni latente è impostata con un parametro 1 (come se avessimo scritto `=~ 1 * L1`) imponendo quindi la metrica alla latente, mentre le successive sono libere e i loro parametri vengono stimati. Aggiungendo un valore di moltiplicazione ad una variabile, si imposta il parametro di quell'osservata, che non verrà stimato.

Se invece vogliamo che le osservate e le latenti siano tutte standardizzate (modello classico con varianza/correlazone uguale a 1), è sufficiente usare i parametri `std.lv=T` e `std.ov=T` che standardizzano le latenti (`lv`) e le osservate (`ov`).

Se invece vogliamo che solo una delle latenti sia standardizzata e le altre prendano la metrica dalla prima osservata, il modello è un poco più complesso perché bisogna impostare esplicitamente la varianza (tramite il simbolo `~~`) della latente a 1:

```
A =~ L1 + L2 + L3 + L4 + L5
R =~ L6 + L7 + L8 + L9 + L10
M =~ L11 + L12 + L13 + L14 + L15
M ~~ 1*M
```

Implicitamente, Lavaan assume che tutte le latenti siano correlate. Se vogliamo avere una soluzione ortogonale, basta usare il parametro `orthogonal=T`, mentre se vogliamo che solo alcuni fattori siano correlati fra loro, dobbiamo porre a 0 la covarianza fra tutti gli altri. Ad es. nel modello seguente otterremo che la latente M non correli con le altre due:

```
A =~ L1 + L2 + L3 + L4 + L5
R =~ L6 + L7 + L8 + L9 + L10
M =~ L11 + L12 + L13 + L14 + L15
A ~~ 0*M
R ~~ 0*M
```

Questi modelli possono essere utilizzati in due modi: 1) scrivendoli direttamente da tastiera:

```
R> Liht.model <- 'A =~ L1 + L2 + L3 + L4 + L5
                  R =~ L6 + L7 + L8 + L9 + L10
                  M =~ L11 + L12 + L13 + L14 + L15'
```

oppure salvando il modello in un file (ad es. `LihtModel.txt`) con un editor qualsiasi e poi caricandolo in R:

```
R> # modello per Liht
R> Liht.model=readLines("LihtModel.txt")
```

Il comando cfa() permette di effettuare un'analisi fattoriale confermativa e la sua sintassi (completa) è:

```
cfa(model = NULL, data = NULL,
    meanstructure = "default", fixed.x = "default",
    orthogonal = FALSE, std.lv = FALSE, std.ov = FALSE,
    missing = "default", ordered = NULL,
    sample.cov = NULL, sample.cov.rescale = "default",
    sample.mean = NULL, sample.nobs = NULL,
    ridge = 1e-05, group = NULL,
    group.label = NULL, group.equal = "", group.partial = "",
    cluster = NULL, constraints = '',
    estimator = "default", likelihood = "default",
    information = "default", se = "default", test = "default",
    bootstrap = 1000L, mimic = "default", representation = "default",
    do.fit = TRUE, control = list(), WLS.V = NULL, NACOV = NULL,
    start = "default", verbose = FALSE, warn = TRUE, debug = FALSE)
```

Il comando minimale per una CFA è basato sull'indicazione del modello e del dataframe da usare:

```
R> cfa(Liht.model, data = Liht)
```

in questo caso, l'analisi viene eseguita e viene visualizzato un riassunto sintetico:

```
lavaan (0.5-12) converged normally after  67 iterations

                                                  Used       Total
  Number of observations                           332         343

  Estimator                                         ML
  Minimum Function Test Statistic              310.465
  Degrees of freedom                                87
```

```
  P-value (Chi-square)                          0.000
```

da questi risultati possiamo verificare che è attivo il parametro `missing='listwise'` (che è il default), mentre possiamo chiedere che i valori mancanti vengano sostituiti usando le opzioni `missing='ml'` oppure `missing='fiml'`.

```
R> cfa(Liht.model, data = Liht, miss='ml')
lavaan (0.5-12) converged normally after  76 iterations

  Number of observations                           343

  Number of missing patterns                        10

  Estimator                                         ML
  Minimum Function Test Statistic              308.719
  Degrees of freedom                                87
  P-value (Chi-square)                           0.000
```

D'ora in avanti, per gli esempi successivi, uso un modello basato sui dati di Holzinger e Swineford (1939), abbondantemente usati in letteratura.

Per vedere gli altri risultati dobbiamo salvarli in un oggetto e quindi chiamare le funzioni di visualizzazione. Con `summary()`, vengono visualizzate le informazioni iniziali, senza indici di adattamento e i parametri delle latenti e delle osservate:

```
R> data(HolzingerSwineford1939)
R> # modello per Holzinger e Swineford (1939)
R> HS.model <- ' visual  =~ x1 + x2 + x3
              textual =~ x4 + x5 + x6
              speed   =~ x7 + x8 + x9 '
R> HS.fit <- cfa(HS.model, data = HolzingerSwineford1939, std.lv=T, std.ov=T)
R> summary(HS.fit)
lavaan (0.5-12) converged normally after  22 iterations

  Number of observations                           301

  Estimator                                         ML
  Minimum Function Test Statistic               85.306
  Degrees of freedom                                24
  P-value (Chi-square)                           0.000

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  visual =~
    x1                0.771    0.069   11.127    0.000
    x2                0.423    0.066    6.429    0.000
    x3                0.580    0.066    8.817    0.000
  textual =~
```

```
    x4                0.850    0.049   17.474    0.000
    x5                0.854    0.049   17.576    0.000
    x6                0.837    0.049   17.082    0.000
  speed =~
    x7                0.569    0.064    8.903    0.000
    x8                0.722    0.065   11.090    0.000
    x9                0.664    0.064   10.305    0.000

Covariances:
  visual ~~
    textual           0.459    0.064    7.189    0.000
    speed             0.471    0.073    6.461    0.000
  textual ~~
    speed             0.283    0.069    4.117    0.000

Variances:
    x1                0.403    0.083
[...]
    x9                0.556    0.069
    visual            1.000
    textual           1.000
    speed             1.000
```

Per le variabili osservate, viene visualizzata la stima dei parametri, l'errore standard, il relativo punto z e la probabilità associata. Se il punto z è inferiore a 1.96 (e quindi la probabilità è non significativa ad $\alpha = .05$), il parametro non da abbastanza informazioni al modello per essere importante. Lo stesso vale per le covarianze delle latenti (in questo esempio, avendo richiesto la standardizzazione, le covariate coincidono con le correlazioni). Infine vengono visualizzate le stime delle varianze delle osservate e relativo errore standard.

Con `parameterEstimates()` si possono ottenere maggiori dettagli sui parametri stimati. Oltre alle informazioni fornite da `summary()`, usando il parametro `standard=T` viene visualizzata anche la stima intervallare e i parametri standardizzati (utili quindi se la soluzione chiesta non era completamente standardizzata).

```
R> parameterEstimates(HS.fit, standard=T)
       lhs op     rhs   est    se      z pvalue ci.lower ci.upper std.lv std.all std.nox
1   visual =~      x1 0.771 0.069 11.127      0    0.635    0.906  0.771   0.772   0.772
2   visual =~      x2 0.423 0.066  6.429      0    0.294    0.552  0.423   0.424   0.424
[...]
9    speed =~      x9 0.664 0.064 10.305      0    0.538    0.790  0.664   0.665   0.665
10      x1 ~~      x1 0.403 0.083  4.833      0    0.239    0.566  0.403   0.404   0.404
[...]
18      x9 ~~      x9 0.556 0.069  8.003      0    0.420    0.692  0.556   0.558   0.558
19  visual ~~  visual 1.000 0.000     NA     NA    1.000    1.000  1.000   1.000   1.000
20 textual ~~ textual 1.000 0.000     NA     NA    1.000    1.000  1.000   1.000   1.000
21   speed ~~   speed 1.000 0.000     NA     NA    1.000    1.000  1.000   1.000   1.000
22  visual ~~ textual 0.459 0.064  7.189      0    0.334    0.584  0.459   0.459   0.459
23  visual ~~   speed 0.471 0.073  6.461      0    0.328    0.613  0.471   0.471   0.471
24 textual ~~   speed 0.283 0.069  4.117      0    0.148    0.418  0.283   0.283   0.283
```

Anche con il comando `standardizedSolution()` si possono ottenere i parametri standardizzati (se la soluzione non era standardizzata)

Passando a `summary()` il parametro `fit.measures=T` vengono riportati anche alcuni indici di *fit* (CFI, TLI, AIC, RMSEA) subito prima di `Parameter estimates:`

```
R> summary(HS.fit, fit.measures=T)
[...]
Model test baseline model:

  Minimum Function Test Statistic              918.852
  Degrees of freedom                            36
  P-value                                        0.000

Full model versus baseline model:

  Comparative Fit Index (CFI)                    0.931
  Tucker-Lewis Index (TLI)                       0.896

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)              -3422.624
  Loglikelihood unrestricted model (H1)      -3379.971

  Number of free parameters                     21
  Akaike (AIC)                               6887.248
  Bayesian (BIC)                             6965.097
  Sample-size adjusted Bayesian (BIC)        6898.497

Root Mean Square Error of Approximation:

  RMSEA                                          0.092
  90 Percent Confidence Interval          0.071  0.114
  P-value RMSEA <= 0.05                          0.001

Standardized Root Mean Square Residual:

  SRMR                                           0.065
[...]
```

Una buona quantità di indici di adattamento si possono ottenere con il comando `fitMeasures()`:

```
R> fitMeasures(HS.fit)
             fmin             chisq                df            pvalue    baseline.chisq
            0.142            85.306            24.000             0.000           918.852
      baseline.df   baseline.pvalue               cfi               tli              nnfi
           36.000             0.000             0.931             0.896             0.896
              rfi               nfi              pnfi               ifi               rni
            0.861             0.907             0.605             0.931             0.931
             logl unrestricted.logl              npar               aic               bic
        -3422.624         -3379.971            21.000          6887.248          6965.097
           ntotal              bic2             rmsea    rmsea.ci.lower    rmsea.ci.upper
          301.000           6898.497             0.092             0.071             0.114
     rmsea.pvalue               rmr        rmr_nomean              srmr       srmr_nomean
            0.001             0.065             0.065             0.065             0.065
             cn_05             cn_01               gfi              agfi              pgfi
          129.490           152.654             0.943             0.894             0.503
              mfi               ecvi
            0.903             0.423
```

Tuttavia la maggior parte delle volte, il ricercatore è interessato solo ad alcuni indici di adattamento ed è quindi possibile specificare quali si desiderano tramite un secondo parametro contenente un vettore con i nomi degli indici di *fit*, ad esempio:

```
R> fitMeasures(HS.fit, c("chisq", "df", "pvalue", "cfi", "rmsea"))
 chisq     df pvalue    cfi  rmsea
85.306 24.000  0.000  0.931  0.092
```

Con `modificationindices()` (oppure `modIndices()`) si ottengono gli indici di modifica relative ai legami di ogni osservata con ogni latente (indicate con `=~`) e degli errori fra osservate (indicate con `~~`). Ovviamente i parametri già inseriti nel modello presentano un *modification index* pari a 0, mentre quelli privi di senso sono indicati con `NA`:

```
R> modificationIndices(HS.fit)
       lhs op     rhs     mi    epc sepc.lv sepc.all sepc.nox
1   visual =~      x1  0.000  0.000   0.000    0.000    0.000
[..]
22   speed =~      x4  0.003 -0.003  -0.003   -0.003   -0.003
23   speed =~      x5  0.201 -0.021  -0.021   -0.021   -0.021
24   speed =~      x6  0.273  0.025   0.025    0.025    0.025
25   speed =~      x7  0.000  0.000   0.000    0.000    0.000
[..]
70      x8 ~~      x8  0.000  0.000   0.000    0.000    0.000
71      x8 ~~      x9 14.946 -0.414  -0.414   -0.415   -0.415
72      x9 ~~      x9  0.000  0.000   0.000    0.000    0.000
73  visual ~~  visual     NA     NA      NA       NA       NA
74  visual ~~ textual  0.000  0.000   0.000    0.000    0.000
75  visual ~~   speed  0.000  0.000   0.000    0.000    0.000
76 textual ~~ textual     NA     NA      NA       NA       NA
77 textual ~~   speed  0.000  0.000   0.000    0.000    0.000
78   speed ~~   speed     NA     NA      NA       NA       NA
```

I primi indici di modifica da considerare sono quelli relativi ai fattori e ai loro legami con le osservate. Si può sfruttare le capacità di R per selezionare solo questi:

```
R> mi.HS <- modificationIndices(HS.fit)
R> mi.HS[mi.HS$op == "=~",]
       lhs op rhs     mi    epc sepc.lv sepc.all sepc.nox
[..]
5   visual =~  x5  7.441 -0.146  -0.146   -0.147   -0.147
6   visual =~  x6  2.843  0.091   0.091    0.092    0.092
7   visual =~  x7 18.631 -0.348  -0.348   -0.349   -0.349
8   visual =~  x8  4.295 -0.187  -0.187   -0.187   -0.187
9   visual =~  x9 36.411  0.514   0.514    0.515    0.515
10 textual =~  x1  8.903  0.297   0.297    0.297    0.297
[..]
```

Per riordinare in modo decrescente gli indici di modifica e avere quindi il valore più elevato per primo:

```
R> mi.HS[order(mi.HS$mi,decreasing=T),]
       lhs op     rhs     mi    epc sepc.lv sepc.all sepc.nox
1   visual =~      x9 36.411  0.514   0.514    0.515    0.515
2       x7 ~~      x8 34.145  0.486   0.486    0.488    0.488
3   visual =~      x7 18.631 -0.348  -0.348   -0.349   -0.349
4       x8 ~~      x9 14.946 -0.414  -0.414   -0.415   -0.415
5  textual =~      x3  9.151 -0.238  -0.238   -0.238   -0.238
6       x2 ~~      x7  8.918 -0.142  -0.142   -0.143   -0.143
[..]
```

Chi è abituato a lavorare con Lisrel, può usare il comando `inspect()` per vedere le varie infomazioni tramite le classiche matrici Lambda-x (Λ_x), Theta-delta (Θ_δ) e Phi (Φ). Al comando, oltre al solito oggetto che contiene i risultati di una CFA, si possono chiedere i parametri da stimare (`"free"`), i parametri (`"parameters"`), i parametri standardizzati (`"standardized"`), la percentuale di varianza spiegata (`"rsquare"`), gli indici di modifica (`"mi"`). L'unico problema è che ogni parametro deve essere usato da solo:

```
> inspect(HS.fit, "free")
$lambda
   visual textul speed
x1      1      0     0
x2      2      0     0
x3      3      0     0
x4      0      4     0
x5      0      5     0
x6      0      6     0
x7      0      0     7
x8      0      0     8
x9      0      0     9

$theta
   x1 x2 x3 x4 x5 x6 x7 x8 x9
x1 10
x2  0 11
x3  0  0 12
x4  0  0  0 13
x5  0  0  0  0 14
x6  0  0  0  0  0 15
x7  0  0  0  0  0  0 16
x8  0  0  0  0  0  0  0 17
x9  0  0  0  0  0  0  0  0 18

$psi
        visual textul speed
visual   0
textual 19      0
speed   20     21      0
```

```
R> inspect(HS.fit, "rsquare")
```

```
      x1        x2        x3        x4        x5        x6        x7        x8        x9
0.5957988 0.1794379 0.3377152 0.7251919 0.7311368 0.7022613 0.3243469 0.5227924 0.4422377
```

Infine è possibile specificare che alcune variabili devono essere considerate come "ordinali", usando il parametro `ordered=c()` con cui vengono indicati i nomi delle variabili, ad esempio:

```
Liht.ord <- cfa(Liht.model, data = Liht, ordered=c("L1","L2","L3","L4",
"L5","L6","L7","L8","L9","L10","L11","L12","L13","L14","L15"))
```

Un'altra possibilità offerta da lavaan è quella di CFA multi-gruppo (o *multisample*). Serve che nel dataframe esista una variabile di raggruppamento che andrà specificata con il parametro `group="variabile"`

[da FINIRE].

Capitolo 10

Analisi delle corrispondenze

10.1 Introduzione

L'analisi delle corrispondenze si può dividere in due categorie, quella semplice e quella multipla. La prima è chiamata (nelle diverse lingue) *correspondence analysis*, *analyse de correspondance* e viene abbreviata in AC o CA. La seconda si trova come *multiple correspondence analysis* o *analise de correspondance multiple*, abbreviati in MCA o ACM.

L'AC (analisi delle corrispondenze semplice) utilizza una semplice tabella a due entrate (righe e colonne) che sono generalmente due variabili categoriali. All'incrocio tra le righe e le colonne, la cella contiene la frequenza della co-occorrenza fra la categoria rappresentata sulla riga e quella sulla colonna. Quindi l'AC si attua su una tabella di contingenza (ossia lavora con variabili categoriali).

Nell'analisi delle corrispondenze multiple, invece, la tabella è generalmente formata da osservazioni (sulle righe) e da variabili categoriali (sulle colonne). Sarà il software stesso a creare la tabella delle contingenze da sottoporre ad analisi.

Per l'analisi delle corrispondenze, i diversi software trasformano le celle della tabella prodotta in una misura metrica che viene poi sottoposta ad una sorta di analisi fattoriale, cioè si cerca di identificare le dimensioni sottostanti. In genere il metodo usato è quello delle componenti principali, ma ci sono anche metodi alternativi.

Una volta estratte le dimensioni, per ciascuna variabili di riga e di colonna (nell'AC) o per le varie categorie delle variabili e per i casi statistici (per l'ACM) vengono calcolati dei punteggi sulle dimensioni, che possono essere poi rappresentate su un piano cartesiano per facilitare l'interpretazione.

In genere, non vengono estratte molte dimensioni, ma solo le prime due o tre.

Su queste dimensioni è poi possibile proiettare altre variabili chiamate "supplementari" o "illustrative" (categoriali o quantitative) per vedere il legame fra queste e quelle usate nell'analisi (chiamate "attive").

10.2 Analisi delle Corrispondenze (semplici) in R

Per gli esempi di AC, usiamo la tabella seguente (perché utilizzati come esempio in molti testi, articoli e manuali di software) che riproduce i dati proposti da Greenagre (1984) e conosciuti come `smoke`. La tabella indica le frequenze di contingenza fra categorie di impiegati (SM=senior

managers, JM=Junior managers, SE=senior employees, JE=junior employes, SC=secretaries) e l'abitudine a fumare.

	no	bassa	media	pesante	tot.
SM	4	2	3	2	11
JM	4	3	7	4	18
SE	25	10	12	4	51
JE	18	24	33	13	88
SC	10	6	7	2	25
tot	61	45	62	25	193

Tabella 10.1: Dati smoke. Fonte: Greenacre (1984)

I dati `smoke` sono disponibili nel pacchetto `ca`; per caricarlo senza caricare anche il pacchetto `ca` (sempre che sia stato installato) possiamo usare il comando:

```
R> data(smoke,package="ca")
```

L'analisi delle corrispondenze semplici (AC) in R si può fare usando diversi package: `anacor`, `ca`, `MASS`.

10.2.1 Simple correspondence analysis (ca::ca)

Il pacchetto `ca` è stato sviluppato da Greenacre e Nenadic (2010) ed utilizza l'approccio SVD (*singular value decomposition*) che è un metodo efficiente di estrazione delle componenti principali. È il pacchetto che più si avvicina ai risultati del software più conosciuto per l'analisi della corrispondenze (lo SPAD).

La procedura principale è `ca`, la cui sintassi è:

```
ca(obj, nd = NA, suprow = NA, supcol = NA,
   subsetrow = NA, subsetcol = NA)
```

dove il primo parametro (`obj`) è una tabella a due entrate composta da frequenze; `nd=` è il numero di dimensioni (automaticamente estrae tutte le dimensioni possibili) ed è quindi preferibile indicarlo sempre. Tramite i parametri `suprow` e `supcol` si possono indicare delle variabili supplementari da includere. Infine con `subsetrow` e `subsetcol` possiamo selezionare un sottoinsieme di variabili. Eco un esempio d'uso di questi parametri:

```
R> ca(smoke,nd=2,subsetrow=c(1:4),suprow=c(5))
```

Chiamando la procedura `ca` si ottengono alcuni risultati importanti: l'inerzia (il nome che Benzecrì ha dato agli autovalori) e la sua percentuale, quindi, sia per le variabili di riga sia per quelle di colonna, la massa, l'inerzia e le coordinate sulle dimensioni.

```
R> library(ca)
R> ca(smoke) -> smoke.ca
R> smoke.ca

 Principal inertias (eigenvalues):
           1        2        3
Value      0.074759 0.010017 0.000414
Percentage 87.76%   11.76%   0.49%
```

```
 Rows:
               SM        JM        SE       JE        SC
Mass     0.056995  0.093264  0.264249 0.455959  0.129534
ChiDist  0.216559  0.356921  0.380779 0.240025  0.216169
Inertia  0.002673  0.011881  0.038314 0.026269  0.006053
Dim. 1  -0.240539  0.947105 -1.391973 0.851989 -0.735456
Dim. 2  -1.935708 -2.430958 -0.106508 0.576944  0.788435

 Columns:
            none    light   medium     heavy
Mass     0.316062 0.233161 0.321244  0.129534
ChiDist  0.394490 0.173996 0.198127  0.355109
Inertia  0.049186 0.007059 0.012610  0.016335
Dim. 1  -1.438471 0.363746 0.718017  1.074445
Dim. 2  -0.304659 1.409433 0.073528 -1.975960
```

In questo esempio, non abbiamo specificato il numero delle dimensioni, gli autovalori sono 3, ma l'ultimo spiega pochissima varianza (o inerzia totale) e quindi non vengono calcolate le relative coordinate.

I risultati si possono stampare o con il comando `print` che mostra esattamente gli stessi risultati ottenuti in output oppure chiamando la variabile in cui abbiamo salvato i risultati, oppure si può usare il comando `summary` che fornisce i risultati nella forma a cui SPAD ci ha abituato.

```
R> summary(smoke.ca)

Principal inertias (eigenvalues):

 dim    value      %   cum%   scree plot
 1      0.074759  87.8  87.8  ************************
 2      0.010017  11.8  99.5  ***
 3      0.000414   0.5 100.0
        -------- -----
 Total: 0.085190 100.0

Rows:
    name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1 |   SM |   57  893   31 |  -66  92   3 | -194 800 214 |
2 |   JM |   93  991  139 |  259 526  84 | -243 465 551 |
3 |   SE |  264 1000  450 | -381 999 512 |  -11   1   3 |
4 |   JE |  456 1000  308 |  233 942 331 |   58  58 152 |
5 |   SC |  130  999   71 | -201 865  70 |   79 133  81 |

Columns:
    name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1 | none |  316 1000  577 | -393 994 654 |  -30   6  29 |
2 | lght |  233  984   83 |   99 327  31 |  141 657 463 |
3 | medm |  321  983  148 |  196 982 166 |    7   1   2 |
4 | hevy |  130  995  192 |  294 684 150 | -198 310 506 |
```

Rispetto al comando `print`, `summary` presenta per ogni variabile le informazioni importanti moltiplicate per 1000 (316 equivale a .316); le informazioni sono la massa, la qualità (`qlt`), l'iner-

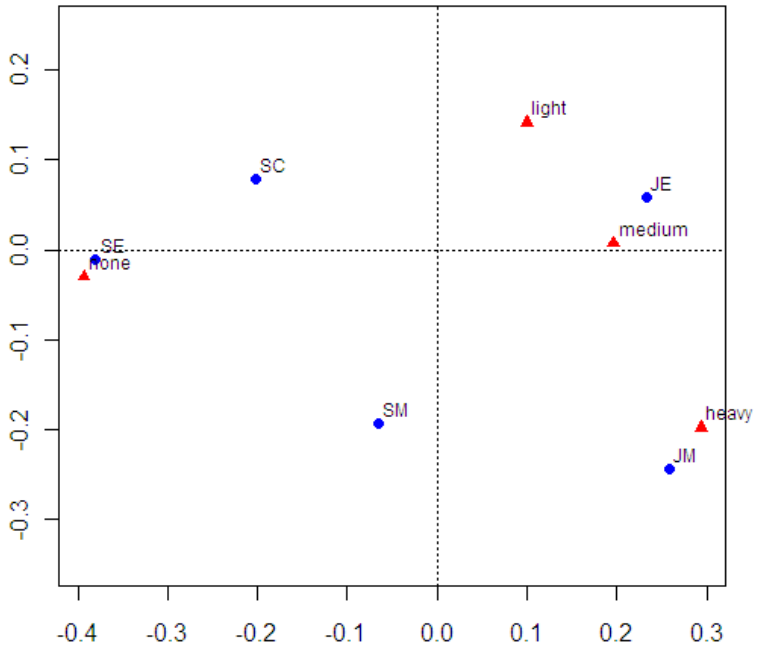

Figura 10.1: Rappresentazione grafica dei risultati di `smoke` usando `ca` con 2 dimensione

zia (`inr`) e per ciascuna delle dimensioni, le coordinate (`k`), il contributo relativo (`cor`) e quello assoluto (`ctr`). Se ci fossero variabili supplementari, verrebbero stampate solo le informazioni applicabili (qualità e contributo relativo).

Il risultato può essere anche rappresentato graficamente (su un piano cartesiano) con il solito comando `plot` (v. Figura 10.1). In questo caso, se ci sono più due dimensioni è possibile indicare quelle che devono essere rappresentate tramite il parametro `dim=c()`, cosa rappresentare con il parametro `what=c()` che per default è "all" (le alternative sono "active", "passive", "none"). Con il parametro `labels=` scegliamo se rappresentare solo i punti (`labels=0`) oppure solo le etichette (`labels=1`); il default è `labels=2` cioè entrambi. Se si analizza una tabella con molte righe e/o colonne, per migliorare l'interpretazione, si possono usare altri due parametri. Il parametro `mass=c(riga,colonna)` serve per rappresentare la massa tramite l'area dei simboli; maggiore la massa, più grandi saranno i simboli utilizzati nel grafico. il parametro richiede valori logici, ad es. `mass=c(T,F)` richiede che la funzione venga applicata sulle variabili di riga (`true`) ma non su quelle di colonna (`false`). Sempre per migliorare la rappresentazione grafica con i dati di grandi tabelle, si può usare il parametro contrib=c("par","par") che è generalmente impostato sul `''none''`. I valori utilizzabili sono "absolute" oppure "relative" da applicare alle variabili di riga e/o di colonna; in entrambi i casi, il contributo viene rappresentato variando l'intensità del colore dal bianco (per i contributi vicini allo 0) al colore prescelto (per i contributi più elevati. In questo modo, le categorie che non contribuiscono molto alle dimensioni, vengono "mascherate"

con colori tenui o addirittura il bianco.

10.2.2 Simple Correspondence Analysis (MASS::corresp)

La procedura `corresp()` presente in MASS (Venables & Ripley, 2002) calcola l'analisi delle corrispondenze semplice usando le correlazioni canoniche principali (*principal canonical correlation*). La sintassi è semplicemente:

```
corresp(x, ...)
```

dove `x` può essere una tabella, una matrice che contiene i dati di una tabella, una tabella prodotta dal comando `xtabs()`, un dataframe oppure una formula.

Tra i parametri che si possono indicare c'è `nf=` che indica il numero di dimensioni. Se non viene indicato, si assume `nf=1`.

Usando i dati `smoke`, avremo:

```
R> library(MAAS)
R> corresp(smoke)
First canonical correlation(s): 0.2734211

 Row scores:
        SM         JM         SE         JE         SC
-0.2405388  0.9471047 -1.3919733  0.8519895 -0.7354557

 Column scores:
      none      light     medium      heavy
-1.4384714  0.3637463  0.7180168  1.0744451

R> corresp(smoke,nf=2)
First canonical correlation(s): 0.2734211 0.1000859

 Row scores:
         [,1]       [,2]
SM -0.2405388 -1.9357079
JM  0.9471047 -2.4309584
SE -1.3919733 -0.1065076
JE  0.8519895  0.5769437
SC -0.7354557  0.7884353

 Column scores:
             [,1]        [,2]
none   -1.4384714 -0.30465911
light   0.3637463  1.40943267
medium  0.7180168  0.07352795
heavy   1.0744451 -1.97595989
```

Se i risultati vengono salvati in una variabile, possono essere stampati sia chiamando la variabile sia con il comando `print` (che produce gli stessi risultati).

I risultati si possono anche rappresentare graficamente. Se abbiamo chiesto una dimensione, il grafico (cfr. Figura 10.2) rappresenterà un incrocio fra le categorie indicate in riga e quelle indicate in colonna

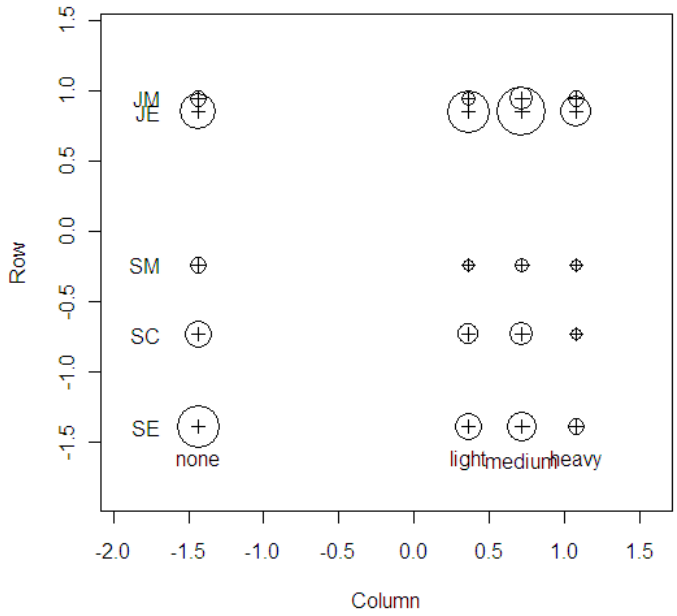

Figura 10.2: Rappresentazione grafica dei risultati di `smoke` usando `corresp` con 1 dimensione

```
R> plot(corresp(smoke))
```

Se abbiamo chiesto due o più dimensioni, il grafico (cfr. Figura 10.3) sarà una proiezione sul piano cartesiano dei primi due assi dimensionali e le variabili di riga e di colonna saranno differenziate per colore

```
R> plot(corresp(smoke,nf=3))
```

10.3 Analisi delle corrispondenze multiple in R

Se per l'analisi delle corrispondenze semplici serviva una tabella di contingenza, per quella multipla, la tabella viene generata direttamente dai dati. Ogni categoria delle variabili qualitative viene trasformata in una variabile dicotomica.

10.3.1 Multiple Correspondence Analysis (MASS::mca)

Per fare una analisi delle corrispondenze multiple con il pacchetto MASS (Venables & Ripley, 2002) si usano generalmente 3 comandi:

```
library(MASS) # per caricare il pacchetto
# per l'analisi multipla
mca(df, nf = 2, abbrev = FALSE)
# per la rappresentazione grafica dei risultati
plot(x, rows = TRUE, col, cex = par("cex"), ...)
```

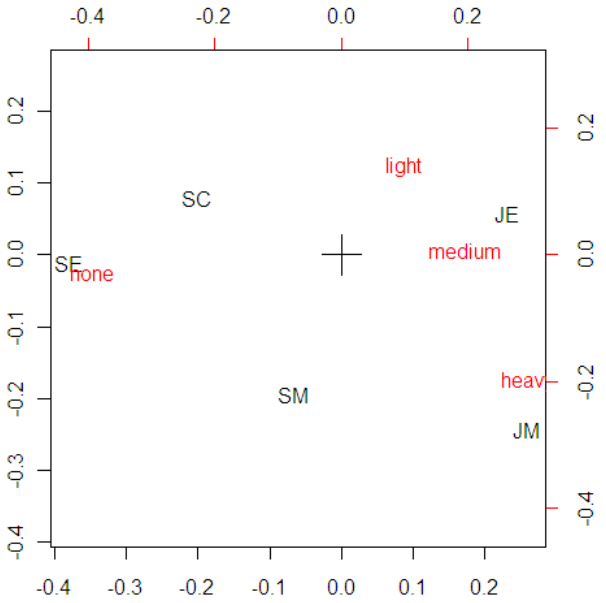

Figura 10.3: Rappresentazione grafica dei risultati di `smoke` usando `corresp` con 2 dimensioni

```
# per aggiungere variabili descrittive o supplementari
predict(object, newdata, type = c("row", "factor"), ...)
```

Il comando `mca` esegue i calcoli per la sola analisi delle corrispondenze multiple. L'unico parametro obbligatorio è `df` che dev'essere un dataframe composto da sole variabili di tipo "factor"; `nf=` indica il numero di dimensioni da estrarre ed è impostato automaticamente a 2. La MCA produce delle coordinate per ogni caso statistico e per ogni livello di ogni variabile. I soggetti vengono etichettati con un numero sequenziale, mentre per i livelli, se `abbrev=` è impostato a falso, viene generato un identificativo formato dal nome della variabile e dal nome del livello separati da un punto, se invece `abbrev=` è impostato a `true`, allora l'identificativo usa solo i livelli. Se la variabile `ss10` ha 4 livelli numerati da 1 a 4 senza etichetta, gli identificatori saranno `ss10.1`, `ss10.2`, `ss10.3`, `ss10.4` con `abbrev=F` e saranno `1`, `2`, `3`, `4` con `abbrev=T`. Se invece i livelli hanno delle etichette (ad es. "niente", "poco", "abbastanza", "molto"), gli identificatori saranno `ss10.niente`, `ss10.poco`, `ss10.abbastanza`, `ss10.molto` oppure `niente`, `poco`, `abbastanza`, `molto`. Attenti quindi alle conseguenze per l'interpretazione dei risultati, in particolare se chiedete delle rappresentazioni grafiche.

I risultati più importanti esportati da `mca` nell'oggetto dei risultati sono: le coordinate dei soggetti (`mca$rs`), quelle dei livelli di tutte le variabili usate (`mca$cs`) e il numero dei fattori estratti (`mca$Sp`).

Esempio temporaneo:

```
#caricare dati da SPSS
library(foreign)
```

```
mydata <- read.spss("d:
Documenti
My Dropbox
Albanese
sostegno.sav",to.data.frame=T,
  use.value.labels = TRUE) -> sostegno
#oppure
library(Hmisc)
mydata <- spss.get("d:
Documenti
My Dropbox
Albanese
sostegno.sav",
  use.value.labels=TRUE)

# caricare MAAS
library(MASS)
mca(sostegno)->sostegno.mca
sostegno.mca
plot(sostegno.mca)
plot(sostegno.mca, row=F)
sostegno.mca$rs
sostegno.mca$cs
```

10.3.2 [Multiple and joint correspondence analysis (ca::mjca)]

```
library(ca)
mjca(newsost[,c(1:28,32:34,37)],lambda="indicator",supcol=c(29:31,35:36))
mjca(newsost[,c(1:28,32:34,37)],lambda="indicator",supcol=newsost[,c(29:31,35:36)])
mjca(newsost[,c(1:28,32:34,37)])
plot.mjca(mjca(newsost[,c(1:28,32:34,37)]))
plot(mjca(newsost[,c(1:28,32:34,37)]))
plot(mjca(newsost[,c(1:28,32:34,37)]),what="active")
plot(mjca(newsost[,c(1:28,32:34,37)]),what=c("active","none")
)
plot(mjca(newsost[,c(1:28,32:34,37)]),what=c("none","active"))
plot(mjca(newsost[,c(1:28,32:34,37)]),what=c("none","active"),xlim=c(-1,2))
plot(mjca(newsost[,c(1:28,32:34,37)]),lambda="Burt",what=c("none","active"),xlim=c(-1,2))
plot(mjca(newsost[,c(1:28,32:34,37)]),lambda="adjusted",what=c("none","active"),xlim=c(-1,2))
plot(mjca(newsost[,c(1:28,32:34,37)]),lambda="JCA",what=c("none","active"),xlim=c(-1,2))
plot(mjca(newsost[,c(1:28,32:34,37)]),supcol=c(29:31),what=c("none","active"),xlim=c(-1,2))
a<-mjca(newsost[,c(1:28,32:34,37)]),supcol=c(29:31))
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(29:31))
a
plot(a,what=c("none","active"))
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(29:31,35:36))

a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(29:31,35:36))
print(a)
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(29:31))
plot(a)
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(35:36))
```

```
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(29:31))
a<-mjca(newsost[,c(1:28,32:34,37)],supcol=c(35:36))
```

Capitolo 11

Analisi dei cluster

L'analisi dei cluster cerca di creare dei gruppi (in genere di casi statistici, ma è possibile anche farla con variabili) di elementi che siano il più possibile omogenei fra loro. Ci sono diversi metodi di *analisi dei cluster* (dei raggruppamenti), suddivisibili in metodi gerarchici e non gerarchici. I metodi gerarchici iniziano unendo fra loro i due casi statistici più simili (o meno dissimili) e quindi procedono associando pian piano coppie di casi statistici, fra loro o con precedenti gruppetti, fino al momento in cui tutti i casi sono uniti in un unico raggruppamento. I metodi non gerarchici cercano invece di creare raggruppamenti separati a partire da casi simili fra loro e procedono inserendo, togliendo o spostando casi statistici nei gruppi formati.

11.1 Metodi gerarchici

11.1.1 Hierarchical Clustering (stats::hclust)

Nel pacchetto `stats` pre-caricato in R, è disponibile una serie di procedure per l'analisi dei cluster. Il primo passo è quello di creare una matrice delle somiglianze/dissimilarità:

```
# preparare la matrice di somiglianza/dissimilarità
dist(x, method = "euclidean", diag = FALSE, upper = FALSE, p = 2)
```

Il comando `dist()` permette di calcolare la matrice delle somiglianze/dissomiglianze fra le righe di un dataframe (primo parametro), scegliendo il metodo per calcolare la somiglianza. Per default, il metodo usato è la distanza euclidea, gli altri metodi possibili sono "maximum", "manhattan", "canberra", "binary" or "minkowski". `diag=` e `upper=` servono per stampare il risultato della procedura quando viene assegnato ad una variabile. Il primo dei due parametri stampa anche la somiglianza/distanza del caso con se stesso, mentre il secondo stampa la matrice delle distanze completamente. I due parametri possono essere usati anche nel comando `print()` di un oggetto prodotto da `dist()`.

Come esempio usiamo il file dati `USArrests` disponibile in `stats`, usando soltanto i primi 3 casi statistici. Quelli che seguono sono alcuni modi di usare il comando `dist()`.

```
R> dist(USArrests[1:3,]) # distanza euclidea
         Alabama   Alaska
Alaska  37.17701
Arizona 63.00833 46.59249
R> dist(USArrests[1:3,], diag=T)
```

```
          Alabama   Alaska  Arizona
Alabama  0.00000
Alaska  37.17701  0.00000
Arizona 63.00833 46.59249  0.00000
R> dist(USArrests[1:3,], met='manhattan',upper=T) # metodo manhattan
        Alabama Alaska Arizona
Alabama           63.5    94.9
Alaska     63.5           78.4
Arizona    94.9   78.4
```

Il comando `hclust()` produce la vera e propria analisi dei cluster gerarchici.

```
# fare l'analisi sulla matrice
hclust(d, method = "complete", members=NULL)
```

Il parametro `method=` può assumere i valori "ward", "single", "complete", "average", "mcquitty", "median" o "centroid". Sempre usando `USArrests` (ma questa volta completo), ecco come usare `hclust()`:

```
R> hc <- hclust(dist(USArrests), "ward")
```

Avendo salvato il risultato del calcolo in una variabile (`hc`) non vediamo nulla, ma se anche decidiamo di vederne il contenuto, otteniamo solo un riepilogo

```
R> hc

Call:
hclust(d = dist(USArrests), method = "ward")

Cluster method   : ward
Distance         : euclidean
Number of objects: 50
```

La variabile in cui abbiamo memorizzato il risultato di `hclust()` contiene diversi risultati che possono essere recuperati separatamente: `merge` contiene la lista dei risultati dell'operazione di raggruppamento. Se il numero è negativo indica il numero progressivo dei casi, se positivo il cluster generato:

```
R> hc$merge
      [,1] [,2]
 [7,]   -7  -38
 [8,]  -19  -41
 [9,]  -21  -30
[10,]  -48    8
[11,]  -50    6
```

Il questo frammento di risultato, al passaggio 8, i casi 19 e 41 sono stati uniti in un cluster, al decimo passaggio sono stati uniti fra loro il caso 48 e il cluster 8.

Altri "oggetti" dentro al risultato sono: `order` che indica l'ordine con cui i casi devono essere visualizzati nel dendrogramma affinché non ci siano linee che si incrociano (e quindi rendano

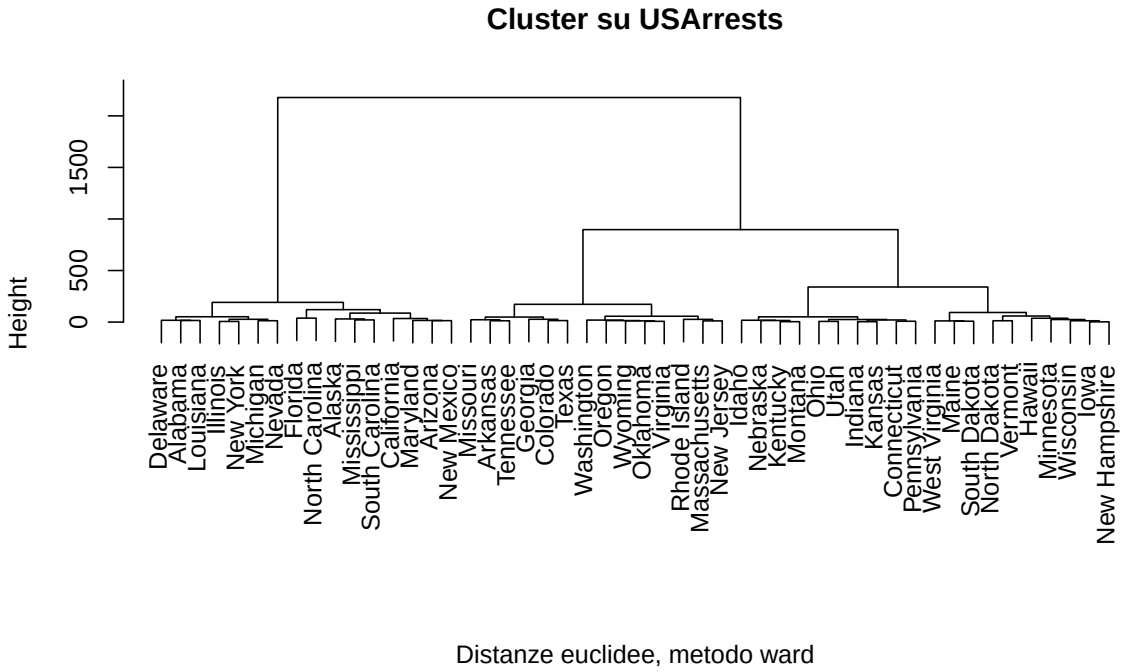

Figura 11.1: Dendrogramma dei risultati di `hclust()`

ulteriormente difficoltoso interpretare un grafico già complesso); `labels` contiene le etichette associate ai casi.

In realtà le informazioni che `hclust()` memorizza nel risultato sono pensate per essere rappresentate graficamente. Il comando `plot.hclust()` (abbreviabile in `plot()`) serve per visualizzare il dendrogramma ottenuto con `hclust()`, in cui le osservazioni sono indicate tramite il nome di riga (rownames()), se è stato assegnato, oppure tramite un numero sequenziale.

Il seguente comando, visualizza il grafico di Figura 11.1:

```
R> plot(hc) # visualizzazione minimale
R> plot(hc,main='Cluster su USArrests',
+ xlab='Distanze euclidee, metodo ward', sub='')
```

Ovviamente, con molti casi statistici, il grafico diventa comunque poco leggibile. Una prima cosa da fare è quella di ampliare orizzontalmente il grafico il più possibile (come è stato fatto nella figura precendente), quindi si può usare `identify.hclust()` e/o `rect.hclust()` per identificare o evidenziare il numero di cluster desiderati.

Il primo comando (`identify.hclust`) è interattivo e permette di identificare graficamente i cluster; può essere abbreviato in `identify()` e può essere usato in modi diversi:

- `R> identify(hc)` (Figura 11.2) permette di vedere diversi possibili raggruppamenti: una volta dato il comando, il grafico viene posto in primo piano e il cursore del mouse diventa una croce. Ci si sposta sul punto che, a nostro parere, identifica un cluster e si preme il tasto del mouse. Immediatamente compare un riquadro colorato che delimita il cluster. Ci si può spostare su

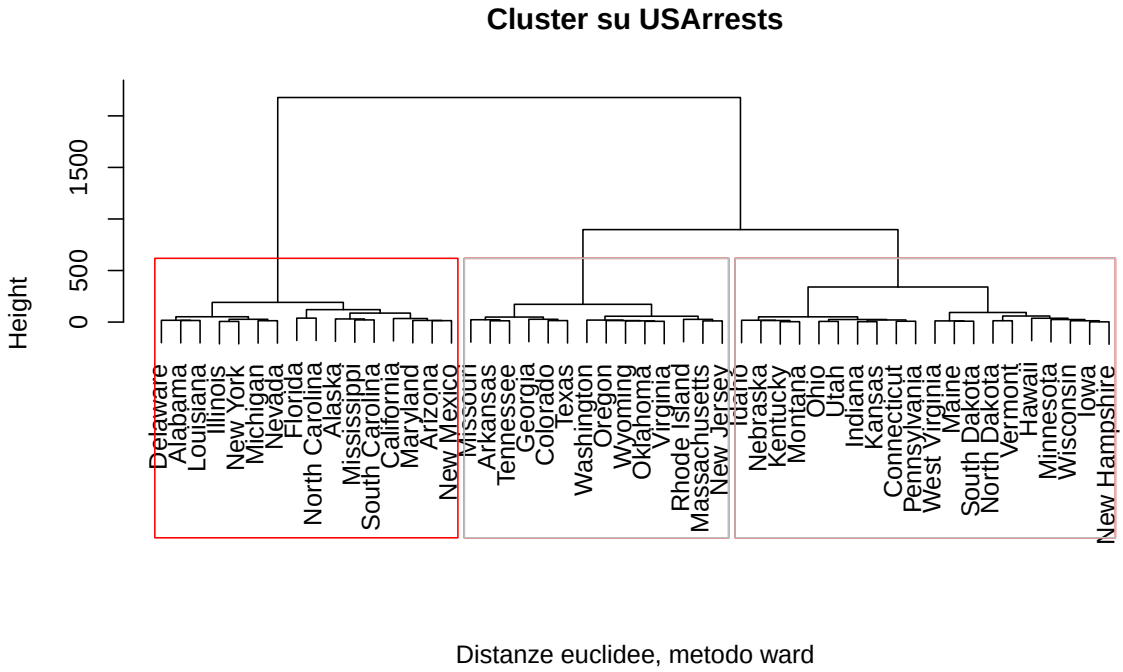

Figura 11.2: Dendrogramma con cluster evidenziati da `identify()`

altri raggruppamenti ed evidenziarlo. Il precedente resterà evidenziato in grigio chiaro. Per terminare, basta premere il tasto destro del mouse e scegliere "Stop".

- `R> (x <- identify(hc))` permette di effettuare le stesse cose del precedente, ma nel momento in cui si esce con lo "Stop", stampa l'appartenenza dei casi ai vari cluster evidenziati. Riporto una parte dei risultati:

```
R> (x <- identify(hc))
[[1]]
  Connecticut        Hawaii         Idaho       Indiana          Iowa
            7            11            12            14            15

[[2]]
     Arkansas      Colorado       Georgia Massachusetts      Missouri
            4             6            10            21            25

[[3]]
       Alabama        Alaska       Arizona    California      Delaware
             1             2             3             5             8
```

Il secondo comando (`rect.hclust`) permette di evidenziare con dei quadrati i cluster in base al numero di raggruppamenti che si desiderano (parametro `k=`) oppure in base alla somiglianza desiderata (parametro `h=` che corrisponde al valore `height` stampato sull'ordinata del grafico).

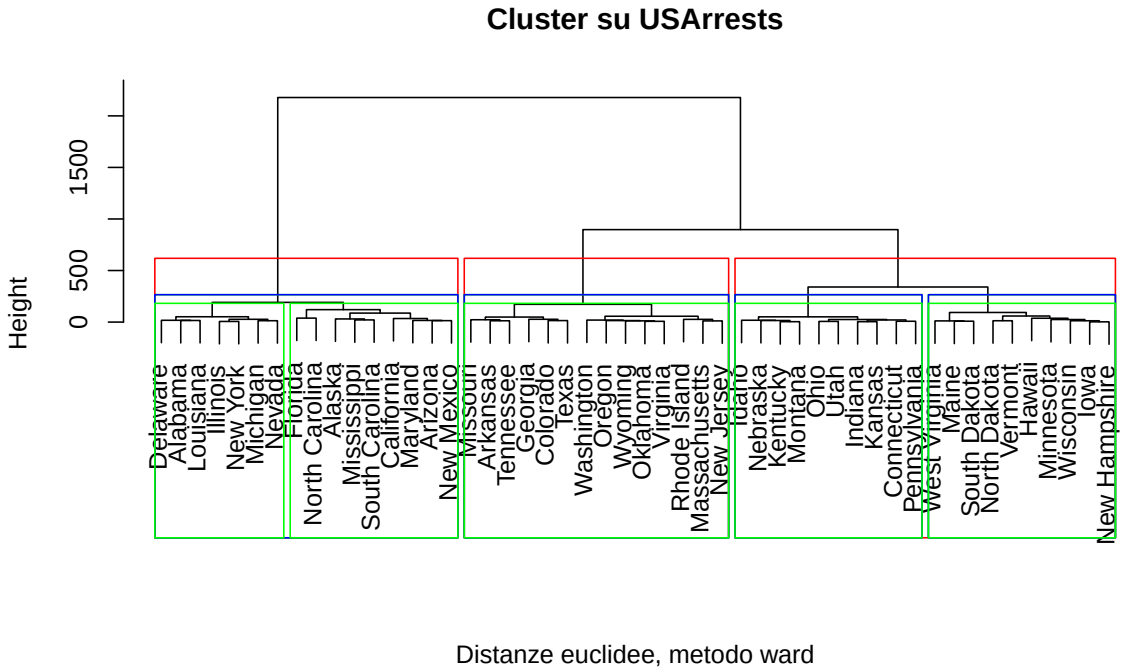

Figura 11.3: Dendrogramma con cluster evidenziati con `rect.hclust()`

Il parametro `border=` può contenere un'indicazione di colore oppure un vettore con un colore per ciascuno dei raggruppamenti desiderati.

```
R> # vedi Figura~11.3
R> rect.hclust(hc, k=3) # 3 cluster in rettangoli rossi
R> rect.hclust(hc, k=4, border="blue") # 4 in blu
R> rect.hclust(hc, k=5, border="green") # 5 in verde
```

Per conoscere l'appartenenza dei casi ai cluster (oltre all'uso di identify) possiamo usare anche il comando `cutree()` a cui passare come parametri sia l'oggetto "cluster" sia il parametro `k=` o `h=` con lo stesso significato di `rect.hclust()`

```
R> cutree(hc, k=5) # risultato troncato per motivi di spazio
       Alabama         Alaska        Arizona       Arkansas
             1              2              2              3
      Delaware        Florida        Georgia         Hawaii
             1              2              3              5
```

11.1.2 Agglomerative Nesting (cluster::agnes)

Nel pacchetto `cluster` è disponibile la procedura `agnes` che calcola un'analisi gerarchica dei cluster.

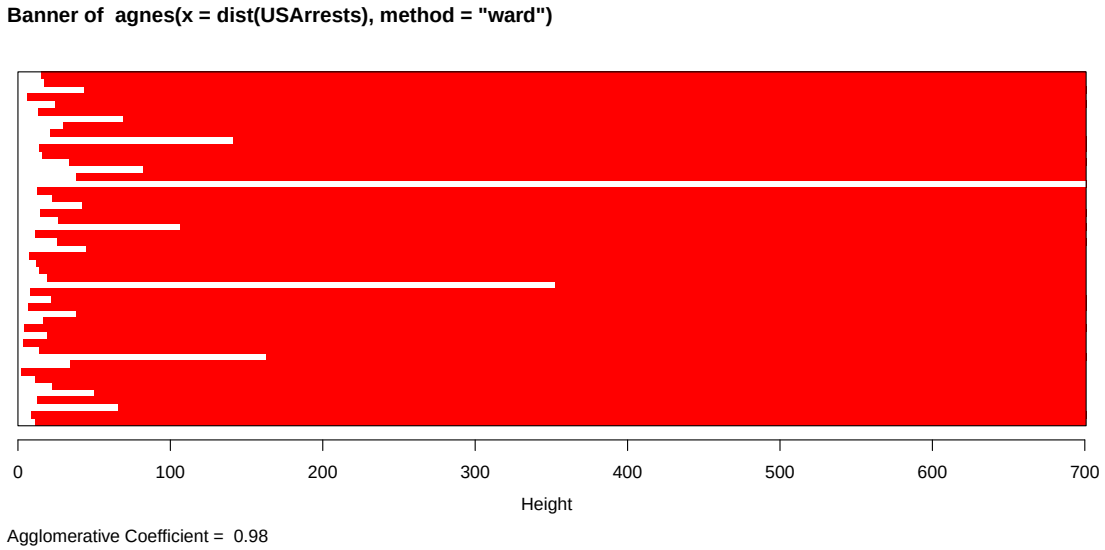

Figura 11.4: Banner prodotto con `agnes()`

```
agnes(x, diss = inherits(x, "dist"), metric = "euclidean",
      stand = FALSE, method = "average", par.method,
      keep.diss = n < 100, keep.data = !diss)
```

agnes() lavora sia con un dataframe sia con una matrice di dati sia con una matrice di somiglianza/dissimilarità. Nei primi due casi, non sono ammessi valori mancanti (usare na.omit()). Il parametro `metric=` è automaticamente impostato su "`euclidean`", ma si può usare in alternativa solo "`manhattan`" (se si vogliono altri metodi bisogna usare il comando dist() visto prima); method= è automaticamente impostato su "`average`", ma si possono selezionare anche "`single`", "`complete`", "`ward`", "`weighted`", "`flexible`" e "`gaverage`".

Il semplice comando `agnes()` è il corrispettivo del precedente `hclust()`

```
R> ag <- agnes(dist(USArrests), method="ward")
```

Per vedere i risultati possiamo usare il comando `plot()` che però è impostato automaticamente in modalità interattiva e mostra due diversi grafici (prima uno poi l'altro), uno chiamato *banner* e il dendrogramma. Per vedere uno solo dei due bisogna usare il parametro `which.plots=` con il valore 1 per *banner* (v.Figura 11.4) e il valore 2 per il dendrogramma (v.Figura 11.5).

```
R> plot(ag, which.plots=1 # banner
R> plot(ag, which.plots=2 # dendrogramma
```

Il banner può essere visualizzato anche con il comando `bannerplot()` che ha una serie di opzioni maggiori del corrispondente plot mentre il dendrogramma si può visualizzare anche con `pltree()`;

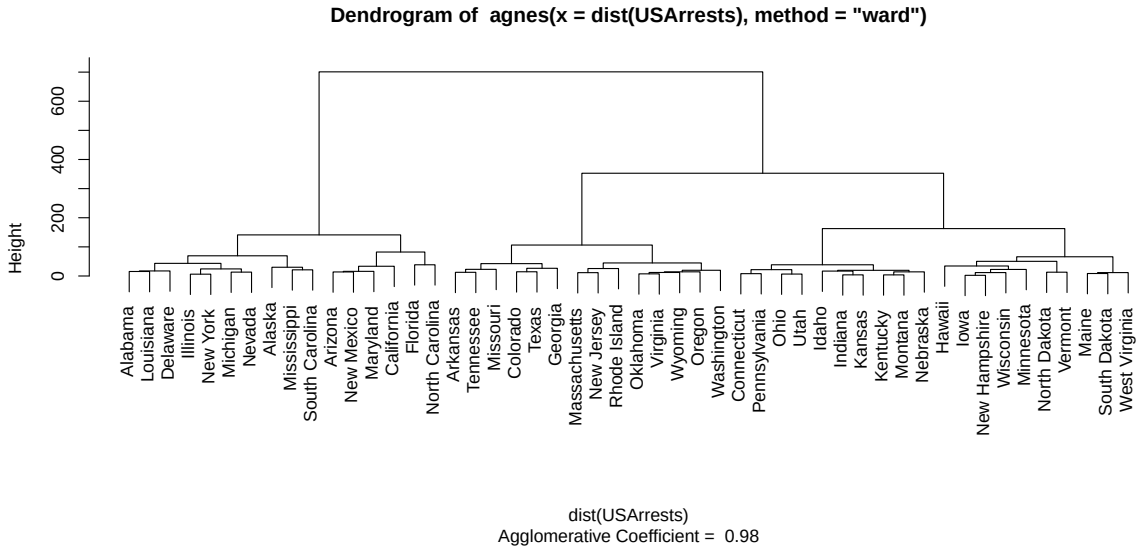

Figura 11.5: Dendrogramma prodotto con `agnes()`

11.2 Metodi non gerarchici

Nel pacchetto `stats` pre-caricato in R, è disponibile anche un comando per l'analisi non gerarchica dei cluster.

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))
```

Bibliografia

Boker, S. M., Neale, M. C., Maes, H. H., Wilde, M. J., Spiegel, M., Brick, T. R., ... Fox, J. (2011). Openmx: An open source extended structural equation modeling framework. *Psychometrika*.

Boker, S. M., Neale, M. C., Maes, H. H., Wilde, M. J., Spiegel, M., Brick, T. R., ... Brandmaier, A. M. (2012). Openmx 1.3 user guide [Computer software manual].

Dinno, A. (2012). paran: Horn's test of principal components/factors [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=paran (R package version 1.5.1)

Falissard, B. (2009). psy: Various procedures used in psychometry [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=psy (R package version 1.0)

Fox, J. (2010). polycor: Polychoric and polyserial correlations [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=polycor (R package version 0.7-8)

Fox, J., & Weisberg, S. (2011). *An R companion to applied regression* (Second ed.). Thousand Oaks CA: Sage. Retrieved from http://socserv.socsci.mcmaster.ca/jfox/Books/Companion

Fox, J., with contributions from Kramer, A., & Friendly, M. (2010). sem: Structural equation models [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=sem (R package version 0.9-21)

Fox, J., with contributions from Liviu Andronic, Ash, M., Boye, T., Calza, S., Chang, A., ... Wolf, P. (2011). Rcmdr: R commander [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=Rcmdr (R package version 1.7-0)

Greenacre, M., & Nenadic, O. (2010). ca: Simple, multiple and joint correspondence analysis [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=ca (R package version 0.33)

Harrell, F. E., & with contributions from many other users. (2012). Hmisc: Harrell miscellaneous [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=Hmisc (R package version 3.9-3)

Lemon, J., & Grosjean, P. (2012). prettyr: Pretty descriptive stats. [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=prettyR (R package version 2.0-4)

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2014). e1071: Misc functions of the department of statistics (e1071), tu wien [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=e1071 (R package version 1.6-3)

Murdoch, D., & Chow, E. D. (2007). ellipse: Functions for drawing ellipses and ellipse-like confidence regions [Computer software manual]. (R package version 0.3-5, porting to R by Jesus M. Frias Celayeta)

R Development Core Team. (2012). R: A Language and Environment for Statistical Computing [Computer software manual]. Vienna, Austria. Retrieved from http://www.R-project.org/

(ISBN 3-900051-07-0)

Revelle, W. (2013). An overview of the psych package (??th ed.) [Computer software manual]. Evanston, Illinois. Retrieved from http://www.personality-project.org/r/psych/vignettes/overview.pdf

Revelle, W. (2014). psych: Procedures for psychological, psychometric, and personality research [Computer software manual]. Evanston, Illinois. Retrieved from http://CRAN.R-project.org/package=psych (R package version 1.4.8)

Rosseel, Y., with contributions of Oberski, D., & Byrnes, J. (2011). lavaan: Latent variable analysis [Computer software manual]. Retrieved from http://CRAN.R-project.org/package=lavaan (R package version 0.4-9)

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth ed.). New York: Springer. Retrieved from http://www.stats.ox.ac.uk/pub/MASS4 (ISBN 0-387-95457-0)

Parte IV

Appendici

Tabella dei comandi

Tabella 1: Elenco dei comandi

Comando	Descrizione	Esempio
-	Sottrazione	`4-5`
*	Moltiplicazione	`4*5`
/	Divisione	`4/5`
?	Aiuto per un comando	?mean
??	Aiuto per frase	??(“mean”)
#	inizio commento	# commento
%*%	Moltiplicazione matriciale	A %*% B
^	Potenza	`2^3`
+	Addizione	`4+5`
acos()	Arcocoseno	acos(.03)
asin()	Arcoseno	asin(.1)
atan()	Arcotangente	atan(3)
boxplot()	Disegna un boxplot	boplot(x)
c()	Concatena	c(1,3,6)
cbind()	Concatena vettori per colonna	cbind(x,y,z)
citation()	Citare un pacchetto	citation(”ca”)
cor()	Correlazione	cor(balene)
cos()	Coseno	cos(1)
cov()	Covarianza (N-1)	cov(balene)
cumsum()	Cumulata	cumsum(balene)
diff()	Differenza	diff(Eta)
edit()	Modifica	x <- edit(Dati1)
exp()	e^x	exp(3)
expm1()	$e^x - 1$	expm1(3)
fivenum()	Riassunto a 5 numeri	fivenum(balene)
fix()	Modifica	fix(Dati1)
getwd()	Trovare la cartella di lavoro	getwd()
help()	Aiuto per un comando	help(mean)
help.search()	Aiuto per frase	help.search(“mean”)
length()	Numero di elementi	lenght(Eta)
library()	Carica un pacchetto	library(psych)
log()	Logaritmo in base e (naturale)	log(5)
log(x, n)	Logaritmo in base n	log(5,2)

Tabella 1: (continua)

Comando	**Descrizione**	**Esempio**
log2()	Logaritmo in base 2	log2(5)
log10()	Logaritmo in base 10	log10(5)
log1p()	Logaritmo di 1+x	log1p(5)
max()	Massimo	max(balene)
mean()	Media	mean(balene)
median()	Mediana	median(balene)
min()	Minimo	min(balene)
q()	Uscire da R	q()
quantile()	Quartili	quantile(balene,.25)
q()	Uscire da R	quit()
rank()	Trasformare in ranghi	rank(Eta)
rbind()	Concatena vettori per riga	rbind(x,y,z)
rep()	Ripete	rep(“m”,15)
sample()	Campionamento	sample(1:100, 75)
sd()	Deviazione standard (N-1)	sd(balene)
setwd()	Imposta la cartella di lavoro	setwd(”c:/cartella”)
sin()	Seno	sin(5)
solve()	Inversa di matrice	solve(mat)
sort()	Riordino	sort(balene)
sqrt()	Radice quadrata	sqrt(9)
sum()	Somma	sum(Eta)
t()	Trasposta di matrice	t(mat)
tan()	Tangente	tan(3)
update.packages()	Aggiorna pacchetti	update.packages()
var()	Varianza (N-1)	var(balene)

Tabella dei pacchetti usati

Questa tabella riepiloga i pacchetti usati in questo testo e le parti del testo in cui sono trattati (ricordarsi che il nome va usato rispettando le maiuscole e le minuscole):

Pacchetto	vers.	rif.	cfr.
car		Fox e Weisberg (2011)	2.2.8.6, 2.2.8.7
e1071	1.6-3	Meyer et al. (2014)	3.5
ellipse			
foreign			
Hmisc			
MAAS			10.2.2 (CA)
lavaan	0.5-17		9.3.2 (CFA)
paran			
polycor			
prettyR			
psy			
psych			
Rcmdr	1.7-0	Fox et al. (2011)	1.6.4
sem			9.3.1 (CFA)
xlsReadWrite			
xlsx			

Elenco delle figure

Elenco delle tabelle

Indice analitico