concordance=TRUE

Dispense per l'editing e le manipolazione dei dati con R

Domenico De Stefano

19 novembre 2021

Lo scopo di questo laboratorio è quello di fornire alcuni strumenti per la manipolazione dei dati. In particolare ci concentreremo su alcune operazioni base che vanno sotto l'etichetta generale di "data preparation".
Esistono vari modi per raccogliere dei dati (studi osservazionali, dati secondari, survey o indagini da questionario, ecc.), quasi tutti però sostanzialmente conducono ad un ouput comune che è la **matrice dati**. Questa può essere vista come un array (in linguaggio informatico) di dimensioni $N \times M$, dove N è il numero di osservazioni (casi) e M è il numero di variabili o items (se i dati provengono da questionario).
La prima e più dispendiosa serie di operazioni da effettuare per poter utilizzare metodi e modelli statistici è quella di manipolare i dati in maniera opportuna.
In questo laboratorio vedremo come impiegare R per una serie di operazioni tipiche di questa fase (ricodifica, gestione filtri, ecc.). A questo scopo utilizzeremo sia funzioni built-in che procedure e metodi *ad hoc*.

1 I dati e il dataframe

1.1 Il dataframe

Le operazioni che vedremo di seguito (di input e di import), riguardano tutte un particolare tipo di oggetto: il **dataframe**. Esso in sostanza, è una sorta di contenitore in cui sono conservati elementi di diversa natura con alcune caratteristiche e nel rispetto di alcuni vincoli.
Il dataframe è la struttura dati di R caratteristica per trattare le tipiche matrici casi per variabili ($N \times M$). Intuitivamente, è come un matrice con N righe e M colonne. Tuttavia, tecnicamente, in R esso è un oggetto di tipo **lista** in quanto ogni colonna (di uguali dimensioni) può contenere diversi tipi di dati (numerici, stringhe, ecc.). Invece in R una matrice è semplicemente un array contenente un solo tipo di dati (es. solo dati numerici).
Esistono tre modi per disporre di un insieme di dati (dataset in forma di dataframe) in R: 1) utilizzo di dataset disponibili all'interno di determinati pacchetti; 2) data input manuale; 3) importazione (acquisizione) dati da fonti o file esterni.

Nel caso 1, quello più semplice ed immediato, i dati sono disponibili in R o in un pacchetto. Pertanto, per il loro utilizzo, occorre semplicemente caricarli. A tale scopo si utilizza la funzione:

```
data(<nome dataset>)
```

Digitando invece solo `data()` oppure `data(package=<nome del pacchetto>)`, si visualizzerà rispettivamente la lista dei dataset disponibili in R (nella sessione corrente) o in determinati pacchetti[1]. Ad es.:

```
data(package="datasets")
```

restituirà la lista dei dataset disponibili nel pacchetto "`datasets`".
Invece `data(<nome del dataset>)`, caricherà il dataset richiesto rendendolo disponibile nel workspace. Ad es.:

```
data("iris")
```

caricherà il dataset iris[2]. che potrà essere richiamato semplicemente digitandone il nome.

```
head(iris)
```

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

Il caso 2 riguarda l'input manuale di dati. In particolare ci interessa la funzione `data.frame()`. Soffermiamoci brevemente sulla costruzione di un semplice dataframe che chiameremo `conversion` e che contiene un vettore di temperature in gradi celsius (0, 10, ..., 40) e la loro trasformazione in gradi fahrenheit.

```
celsius <- (0:4)*10
fahrenheit <- 9/5*celsius+32
conversion <- data.frame(Celsius = celsius, Fahrenheit = fahrenheit)
print(conversion)
```

```
##   Celsius Fahrenheit
## 1       0         32
## 2      10         50
## 3      20         68
## 4      30         86
## 5      40        104
```

[1] L'elenco di tutti i pacchetti installati lo si può ottenere con la funzione `library()`, non specificando alcun argomento.

[2] Per dettagli sui dati contenuti in Iris si veda: `http://en.wikipedia.org/wiki/Iris_flower_data_set`

Il caso 3 è quello più tipico. Spesso, infatti, disporremo di un insieme di dati (di solito una matrice dati) in un certo tipo di formato a seconda delle specifiche del software usato per la raccolta dati o della fonte da cui sono state estratte le informazioni.
Se si ha la possibilità di controllare il tipo di formato in cui salvare i dati è pereferibile usare formati semplici e il più possibile portabili da un sistema all'altro. Si consigliano formati file di "tipo" testo, come: `.txt`, `.dat`, `.csv`.
Una volta disponibile la matrice dati in uno dei formati di cui sopra è possibile importarla in R attraverso la funzione:

```
read.table(<percorso + nome file>, <argomenti>)
```

o alternativamente

```
read.csv(<percorso + nome file>, <argomenti>)
```

Si noti che occorre fornire il "percorso" del file. Se lo si omette, R andrà a cercare il file specificato nella funzione all'interno della cartella di lavoro corrente. Per verificare il percorso della cartella di lavoro corrente si usa la funzione:

```
getwd()
```

Per cambiare la cartella di lavoro corrente con un'altra occorre usare la funzione `setwd(<Percorso Cartella>)` oppure dal menù "File", usando il comando "Cambia Directory" (in ambiente Windows).
Utilizzando il comando `read.table` proviamo ad importare il file `studenti.csv`[3], denominandolo `s`:

```
s <- read.table("studenti.csv", header=TRUE, sep=";", na.string="NA", dec=".")
```

Se, come questo caso, il file dati è disponibile nella cartella di lavoro non occorre specificarne il percorso.
Attraverso tale comando il file `studenti.csv` viene importato e automaticamente convertito nel dataframe `s`. Alcune osservazioni:

- il percorso del file (path), se non salvato nella cartella di lavoro, viene scritto come negli ambienti Unix (o Linux) con lo "slash" /;
- l'argomento `header=TRUE` specifica che la prima riga del file contiene i nomi delle variabili;
- l'argomento `sep=";"` indica che i diversi campi sono separati da un carattere ";" (come di solito avviene per i file `.csv` in ambiente DOS). Nel caso di tab o altri separatori, come la virgola, si può specificare `sep="t"` `sep=","`;

[3]Il file studenti contiene i dati rilevati sugli studenti della facoltà di Economia dell'Università di Trieste relativi agli esiti dei test d'accesso.

- l'argomento `na.strings="NA"` può essere particolarmente utile se nel file sono presenti valori mancanti, in questo caso individuati con `NA`;
- l'argomento `dec="."` specifica il tipo di carattere utilizzato per separare i decimali, in questo caso un punto.

Esistono numerosi argomenti da specificare nella funzione `read.table`. In generale per accedere all'help delle funzioni in R basta digitare `?<Nome Funzione>`.
Una volta importati i dati, per verificare la dimensione $N \times M$ del nostro dataframe usiamo il comando `dim`:

```
dim(s)

## [1] 356  15
```

Dato che ogni elemento del dataframe rappresenta una variabile statistica, il comando `length()` restituisce il numero delle variabili:

```
length(s)

## [1] 15
```

mentre `names()` i rispettivi nomi:

```
names(s)

##  [1] "corso"        "sesso"        "anno"         "provincia"    "scuola"
##  [6] "voto"         "test"         "votoTest"     "numEsamiTot"  "creditiTot"
## [11] "numEsami"     "crediti"      "mediaEsami"   "reiscrizioni" "matematica"
```

è anche possibile aggiungere/modificare i nomi di riga attraverso il comando `row.names()`, ma probabilmente per un dataframe questo potrebbe non essere molto utile.

1.2 Indicizzare elementi di un dataframe

Adesso che abbiamo importato un dataframe vediamo alcuni comandi che serviranno ad indicizzare casi e/o variabili.
Ad esempio, per accedere al contenuto di un'intera "variabile" del nostro dataframe basterà usare il comando `<nome dataframe>[[i]]`, dove al posto della variabile i metteremo la posizione corrispondente al vettore (variabile) desiderato. Questa modalità è quella tecnicamente "più corretta" ma non è quella consigliabile. Inoltre non è l'unica, vi sono altri due possibili comandi che restituiscono il medesimo risultato.
In sintesi, se vogliamo visualizzare la prima variabile (i=1) abbiamo tre modi per farlo che implicano altrettanti modi di considerare un dataframe:

1. dataframe come 'LIST'

```
s[[1]]
```

2. dataframe come 'MATRIX'

```
s[,1]
```

3. dataframe come 'DATAFRAME'

```
s$corso
```

Nel caso 1 stiamo puntando al primo elemento (un vettore, nel nostro caso) nella lista; nel caso 2 stiamo chiedendo di estrarre la prima colonna[4] nella 'matrice' `s`; nel terzo caso stiamo invece chiedendo la restituzione della variabile 'corso'. Quest'ultimo caso rappresenta la modalità consigliabile.

1.3 L'estrazione di sottoinsiemi di dati e l'uso dei filtri

Una delle operazioni più comuni che sarà necessario effettuare sia per controllo dati che per altre finalità è selezionare soltanto una parte del dataframe iniziale. Ad esempio solo alcune variabili o soltanto alcuni casi soddisfacenti certi criteri specifici.
A tale scopo è possibile applicare sia comandi simili a quelli disponibili anche per altre strutture dati (come le matrici) sia altri specifici per la struttura dataframe.
Prima di vedere tali operazioni dataframe, vediamo come le operazioni di estrazione di sottoinsiemi funzionano per strutture dati più semplici: i vettori. tali operazioni saranno utili anche per manipolare interi dataframe.
Sia `x` il vettore che contiene i valori `3, 11, 8, 15, 12`:

```
x <- c(3,11,8,15,12)
```

Le strade più comuni per estrarre sottoinsiemi di vettori si ottengono:

- Specificando l'**indice** degli elementi che si vogliono estrarre[5]. Ad esempio, estraiamo solo gli elementi di `x` in posizione `2` e `4`:

```
x[c(2,4)]


## [1] 11 15
```

[4]Nella struttura del comando `<dataframe>[<righe>,<colonne>]`, lasciare uno degli argomenti `<righe>` o `<colonne>` vuoto indica che desideriamo, rispettivamente, la restituzione in output di tutte le righe o tutte le colonne.

[5]Attenzione a non confondere l'indice dell'elemento di un vettore con il suo valore. I valori (elementi) del nostro vettore `x` sono i numeri `3, 11, 8, 15, 12`; gli indici sono le posizioni di tali valori che ovviamente vanno da `1` a `length(x)`. Per es. il valore `11` è nella posizione `2`, il valore 12 è nella posizione `length(x)`.

- Specificando il valore negativo degli indici corrispondenti agli elementi che si vogliono rimuovere. Ad esempio, rimuoviamo gli elementi di `x` in posizione `2`, `3` e `4`

```
x[-c(2,3,4)]

## [1]  3 12
```

o alternativamente:

```
x[-(2:4)]

## [1]  3 12
```

- Specificando vettori booleani (i cui elementi sono i valori logici `TRUE` e/o `FALSE`). In questo caso gli elementi che saranno estratti saranno soltanto quelli per cui il valore logico è uguale a `TRUE`, ossia quelli per cui una determinata condizione logica è soddisfatta. Ad esempio, supponiamo di voler estrarre esclusivamente i valori di `x` maggiori di `10`, ossia che soddisfano la condizione logica `x > 10`

```
x>10

## [1] FALSE  TRUE FALSE  TRUE  TRUE

x[x>10]

## [1] 11 15 12
```

E' possibile che agli elementi di un vettore (analogamente alle variabili di un dataframe) sia associato un nome. In questi casi l'estrazione può avvenire utilizzando questi nomi:

```
altezza<-c(Andrea=178, Carlo=185, Marco=170)

altezza[c("Andrea", "Marco")]

## Andrea  Marco
##    178    170
```

Le precedenti operazioni possono essere utilizzate ed adattate anche al caso di estrazione di sottoinsiemi da dataframe. Come accennato, visto che è possibile guardare ad un dataframe in termini di matrice (ossia collezione di vettori), possiamo estrarre sottoinsiemi di dati indicizzando per righe e/o colonne.
Ad esempio, con il seguente comando:

```
s[2:5,]
```

```
##   corso sesso anno provincia scuola voto test votoTest numEsamiTot creditiTot
## 2  EC01     F   90        TS      4  100    0        1           4         15
## 3  EC01     M   90        VE     11   60    2       NA           0          0
## 4  EC01     M   76        TS     NA   62    2       NA           0          0
## 5  EC01     M   88        TS      5   65    2       NA           0          0
##   numEsami crediti mediaEsami reiscrizioni matematica
## 2        2       9       25.5            1          0
## 3        0       0        0.0            1          0
## 4        0       0        0.0            0          0
## 5        0       0        0.0            0          0
```

abbiamo estratto i casi dal 2 al 5 (quindi dalla riga 2 alla riga 5, inclusa).
oppure con il seguente comando:

```
s[2:5,2]
```

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

abbiamo estratto, sempre per gli stessi casi, solo la variabile 2 ('sesso'). Una cosa importante da notare nell'output precedente è che estraendo in questo modo una parte del dataframe essa è considerata nella sua struttura originale e cioè come *fattore*[6]. Infatti con il comando `class()` possiamo esplorare la natura della struttura dati considerata:

```
class(s[2:5,2])
```

```
## [1] "factor"
```

Se vogliamo che l'estrazione di una parte di dataframe sia ancora un dataframe occorre aggiungere all'istruzione il parametro `drop=FALSE`:

```
s[2:5,2,drop=FALSE]
```

```
##   sesso
## 2     F
## 3     M
## 4     M
## 5     M
```

verifichiamo con `class()`:

[6] I fattori saranno definiti nel paragrafo successivo.

```r
class(s[2:5,2,drop=FALSE])

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

Un altro modo per selezionare parti di dataframe è attraverso l'uso di **filtri**.
Ad esempio, se fossimo interessati a 'filtrare' la variabile 'anno' (l'anno di nascita degli intervistati) solo per i maschi, basterà utilizzare il seguente comando:

```r
#seleziona i valori della variabile anno di nascita per i maschi:
s[s$sesso=="M","anno"]

##   [1] 85 90 76 88 90 90 90 90 87 91 88 74 90 90 85 84 85 90 90 87 90 89 88 90 88
##  [26] 90 90 90 90 90 90 88 85 90 89 90 90 87 90 89 90 90 89 90 90 90 90 65 90 88
##  [51] 90 90 90 89 90 90 89 89 90 90 90 90 90 85 90 88 89 90 90 89 90 90 83 86 88
##  [76] 86 89 90 78 90 90 90 89 89 90 90 89 90 90 89 90 88 90 90 90 90 89 89 90 84
## [101] 89 90 90 90 88 88 90 88 86 88 89 82 90 89 89 82 90 88 89 89 90 89 88 89 89
## [126] 90 87 90 89 90 90 90 88 90 89 90 89 89 89 90 90 90 88 90 89 90 90 90 90 88
## [151] 83 90 90 90 90 90 90 90 89 90 90 90 90 88 90 83 90 90 84 90 89 88 90 86 88
## [176] 72 90 90 79 90 90 87 90 90 90
```

o alternativamente:

```r
s$anno[s$sesso=="M"]
```

Volendo, invece, selezionare solo la variabile sesso per i nati prima del '90 (incluso):

```r
#seleziona i valori della variabile sesso per cui anno<=90:
s$sesso[s$anno<=90]

##   [1] M F M M M M M F F F M M F M F F M F M M F F M M M M F M M F F M M F M F M
##  [38] M M F F M F F F F M M M M F M F F F M F M F F M F M F F M F M F F F M M M
##  [75] F F F F M M F M M F F F M F M F M M M M F F M F M F M F M F F F M M M F F F F
## [112] M M F M M F F M F M F F M M M F F M M F F M M F F M F F F M M M M F F F M
## [149] M F M M F M F M M M M M M M M F M M F M M M M M M M F M M M F M F M F M F
## [186] F F M F M M M F M M M M M M M M F M F M F F M M F F F M F M F F F M F F M
## [223] F F F F F M M F F F M M F M M M F F M F M M F M F M F M M F F F F M M M F
## [260] M M M M M M M M F F F M M F M F F M M F M M M F M M F M M F M F F M F F F
## [297] M M F F M M M M M M F F M F F M M F F F M M F F F F F F M F F F M M F M M
## [334] F F M F M M F F F F F F F M F F M M
## Levels: F M
```

Quindi vi sono due modi per estrarre alcuni valori all'interno di una variabile contenuta in un dataframe: in un caso la selezione viene effettuata sugli elementi del vettore estratto (ad es. `s$sesso`); nell'altro la condizione per selezionare i casi viene specificata all'interno delle parentesi quadre prima della virgola, mentre dopo la virgola appaiano i nomi delle variabili a cui la condizione va

applicata. Il discorso può essere facilmente generalizzato per estendere il numero delle variabili da selezionare e per complicare la condizione di inclusione (o esclusione) contemplando contemporaneamente anche altri simboli quali `&` (operatore logico *and*) e `|` (operatore logico *or*). Questi due operatori logici vengono solitamente utilizzati per specificare che le condizioni si verifichino "simultaneamente" o "alternativamente".
Ad esempio se fossimo interessati al vettore 'sesso' solo per i nati a Trieste prima del '90 (incluso)

```
#seleziona i valori della variabile sesso per cui anno<=90 e provincia=TS:
s$sesso[s$anno<=90 & s$provincia=="TS"]
```

```
##   [1] F    M    M    M    F    M    F    F    M    M    F    M    <NA> F    <NA>
##  [16] F    F    M    F    M    F    F    F    F    M    F    F    M    M    F
##  [31] F    F    M    F    F    F    F    M    F    M    M    F    M    F    F
##  [46] F    M    M    F    M    F    M    F    F    M    F    F    M    M    F
##  [61] M    F    M    F    M    F    M    M    M    M    F    M    M    F    F
##  [76] M    M    M    M    F    M    F    M    M    F    M    F    F    M    M
##  [91] M    F    F    M    F    M    M    M    M    M    F    M    M    F    M
## [106] M    F    M    M    M    F    M    F    M    M    M    M    M    M    M
## [121] M    M    F    M    F    F    M    M    M    F    M    M    F    F    F
## [136] M
## Levels: F M
```

Invece una selezione un po più complicata è la seguente:

```
#seleziona i valori delle variabili "numEsamiTot" e "creditiTot" per cui anno<=80 e provincia
s[s$anno<=80 & s$provincia!="TS", c("numEsamiTot","creditiTot")]
```

```
##     numEsamiTot creditiTot
## 20            0          0
## 42            4         24
## 63            3         18
## 155           0          0
## 272           0          0
## 290           0          0
```

Qui siamo interessati alle variabili 'numEsamiTot' (numero di esami totali) e 'creditiTot' (numero di crediti totali) per i nati prima del 1980 non a Trieste (l'operatore logico `!=` indica la *negazione logica*). Il comando `c(<oggetto1>, <oggetto2>, ...)` corrisponde all'operazione concatenazione di oggetti.
La condizione di selezione dei record è sostanzialmente un vettore logico (ottenuto combinando altri vettori logici) che quando utilizzato come filtro fa in modo che solo gli elementi in corrispondenza di `TRUE` vengano selezionati.
Anche se l'uso dei codici appena visti possa servire a selezionare parti di un dataframe, un modo più diretto è attraverso la funzione `subset()`, in cui gli argomenti `subset` e `select` specificano rispettivamente i casi e le variabili da includere (o escludere) nella selezione.
Ad esempio,

```
subset(s,subset=(anno<80 | anno>90),select=c(numEsamiTot,creditiTot))
```

```
##     numEsamiTot creditiTot
## 4             0          0
## 15            6         39
## 20            0          0
## 42            4         24
## 78            0          0
## 93            4         30
## 155           0          0
## 161           4         24
## 186           4         24
## 272           0          0
## 287           6         57
## 290           0          0
## 320           4         27
## 334           6         42
## 338           5         48
## 349           2         15
```

restituisce i valori delle varibili 'numEsamiTot' e 'creditiTot' per i nati prima del 1980 o dopo il 1990.
Un altro esempio:

```
subset(s,subset=(anno<80 | anno>90),select=-anno)
```

```
##     corso sesso provincia scuola voto test votoTest numEsamiTot creditiTot
## 4    EC01     M        TS     NA   62    2       NA           0          0
## 15   EC01     M        TS     11   70    1     7.00           6         39
## 20   EC01     M        TV      4   73    2       NA           0          0
## 42   EC01     F        VE      4   83    2       NA           4         24
## 78   EC01     F        TS      4   66    2       NA           0          0
## 93   EC01     M        TS     11   63    2       NA           4         30
## 155  EC11     M   TUNISIA     12   62    0     0.00           0          0
## 161  EC11     F        TS     12   98    0     1.00           4         24
## 186  EC11     F   CROAZIA     12   87    0     0.00           4         24
## 272  EC11     F        GO     12   78    2       NA           0          0
## 287  EC21     F        UD     11  100    2       NA           6         57
## 290  EC21     F        GO     11   91    2       NA           0          0
## 320 EC01E     F        TS     12   87    0     4.75           4         27
## 334 EC01E     M        TS      5   75    1     8.25           6         42
## 338 EC11E     M        TS     11   66    2       NA           5         48
## 349 EC11E     F        TS     12   90    2       NA           2         15
##     numEsami crediti mediaEsami reiscrizioni matematica
## 4          0       0       0.00            0          0
```

```
## 15        4      33      23.25            1          1
## 20        0       0       0.00            0          0
## 42        3      21      27.00            1          1
## 78        0       0       0.00            1          0
## 93        4      30      25.50            1          0
## 155       0       0       0.00            0          0
## 161       2      18      19.50            1          0
## 186       2      18      22.00            1          0
## 272       0       0       0.00            0          0
## 287       5      54      26.80            1          1
## 290       0       0       0.00            0          0
## 320       3      24      20.00            1          1
## 334       4      36      21.50            1          1
## 338       5      48      26.20            1          1
## 349       1      12      20.00            0          1
```

restituisce i valori di tutte le variabili (escluso la variabile 'anno')[7] per i nati prima del 1980 o dopo il 1990.

1.4 Trattamento dei dati mancanti

Molto spesso i dati raccolti (soprattutto da questionario) sono caratterizzati da valori mancanti che in R sono etichettati con la stringa `NA`. Se si stanno importando dati da un file esterno, è molto probabile che i dati mancanti siano etichettati con qualcosa di diverso da `NA`; ad esempio un asterisco, una cella vuota oppure valori numerici volutamente implausibili, come 9999. In quest'ultimo caso, ad esempio, nel dataframe tutti i 9999 dovranno essere sostituiti da un `NA`. Questo può essere ottenuto facilmente utilizzando l'argomento `na.strings=999` nella funzione `read.table()`, oppure una sostituzione puµo avvenire facilmente nel seguente modo:
`<dataframe>[<dataframe> == 999] <- NA`
Una funzione molto utile per individuare i dati mancanti è `is.na()` che restituisce il valore logico `TRUE` se il dato è mancante:

```
is.na(s)
```

Alcune funzioni in R hanno speciali argomenti per utilizzarle anche in presenza di dati mancanti. Tipicamente impostando `na.rm=TRUE`, la funzione verrà eseguita escludendo i dati mancanti. Ad esempio, mediante tale parametro, possiamo escludere i valori mancanti dal calcolo della media aritmetica. Calcoliamo la media della varibile 'votoTest' che contiene numerosi `NA`:

```
mean(s$votoTest,na.rm=TRUE)

## [1] 6.444601
```

[7] Il segno '-' davanti al nome di una (o più) variabile (variabili) all'interno del comando `select` indica l'eliminazione di quella variabile. Comunque se si vuole eliminare soltanto una variabile allora è possibile procedere riassegnando a quella variabile `NULL`, ovvero un oggetto vuoto; ad esempio `s$anno<-NULL` elimina `s$anno`.

se invece non impostiamo `na.rm=TRUE`, otterremo come risultato `NA`

```
mean(s$votoTest)

## [1] NA
```

Tuttavia, alcune funzioni non dispongono di alcun parametro per i dati mancanti. Pertanto, riconoscere i dati mancanti risulta fondamentale ed è auspicabile avere "pieno controllo" su di essi. Ad esempio, è possibile selezionare a priori solo i dati non-mancanti utilizzando la stessa funzione `is.na()`.
Per esempio, volendo selezionare dal nostro dataframe i record con dati non-mancanti nella variabile 'votoTest':

```
s1<-s[!is.na(s$votoTest),]
head(s1)

##    corso sesso anno provincia scuola voto test votoTest numEsamiTot creditiTot
## 1   EC01     M   85      TOGO     12   62    0    -1.00           5         36
## 2   EC01     F   90        TS      4  100    0     1.00           4         15
## 7   EC01     M   90        TS      7   77    1    11.75           6         45
## 8   EC01     F   90        TV     11   81    1     6.25           6         39
## 9   EC01     F   90        TS     10   90    1     7.00           6         45
## 11  EC01     M   90        PN     11   69    1    11.75           6         39
##    numEsami crediti mediaEsami reiscrizioni matematica
## 1         4      33      20.75            1          1
## 2         2       9      25.50            1          0
## 7         5      42      26.40            1          0
## 8         4      33      27.25            1          1
## 9         5      42      26.60            1          1
## 11        4      33      26.75            1          1
```

Come in precedenza, attravero l'operatore logico `&`, è possibile combinare tante variabili per estrarre il sotto-dataframe che contiene valori non-mancanti. In generale quando ci sono dati mancanti relativi a diverse variabili, il sotto-dataframe avente soltanto record con valori completi potrà essere ottenuto mediante la funzione `na.omit()`, cioè semplicemente con:

```
na.omit(s)
```

1.5 Le funzioni `rbind()` e `cbind()`

Le funzioni `rbind()` e `cbind()` valide per le matrici funzionano anche con i dataframe. In particolare esse servono, rispettivamente, ad aggiungere righe o colonne ad un dataframe, posto che queste righe-colonne aggiuntive abbaino dimensioni compatibili.
Ad esempio se vogliamo aggiungere una colonna avente un numero identificativo per ciascun caso creiamo il vettore 'id':

```
id <-1:nrow(s)
```

dove`nrow` identifica il numero di righe (casi) del nostro dataframe `s`.
Per aggiungere questo vettore usiamo `cbind()`:

```
s <-cbind(id,s)
head(s)

##   id corso sesso anno provincia scuola voto test votoTest numEsamiTot
## 1  1  EC01     M   85      TOGO     12   62    0       -1           5
## 2  2  EC01     F   90        TS      4  100    0        1           4
## 3  3  EC01     M   90        VE     11   60    2       NA           0
## 4  4  EC01     M   76        TS     NA   62    2       NA           0
## 5  5  EC01     M   88        TS      5   65    2       NA           0
## 6  6  EC01     M   90        TV     11   80    2       NA           2
##   creditiTot numEsami crediti mediaEsami reiscrizioni matematica
## 1         36        4      33      20.75            1          1
## 2         15        2       9      25.50            1          0
## 3          0        0       0       0.00            1          0
## 4          0        0       0       0.00            0          0
## 5          0        0       0       0.00            0          0
## 6         15        2      15      22.50            1          0
```

Un'applicazione possibile di `cbind` è di utilizzare tale funzione per creare nuove variabili da quelle originali. Ad esempio, immaginiamo di voler aggiungere una variabile che sia la differenza tra numero di esami totali (comprese le idoneità, ovvero esami solo con esito positivo o negativo) e numero di esami (senza le idoneità). Le variabili corrispondenti sono 'numEsamiTot' e 'numEsami':

```
s <-cbind(s,s$numEsamiTot-s$numEsami)
head(s)

##   id corso sesso anno provincia scuola voto test votoTest numEsamiTot
## 1  1  EC01     M   85      TOGO     12   62    0       -1           5
## 2  2  EC01     F   90        TS      4  100    0        1           4
## 3  3  EC01     M   90        VE     11   60    2       NA           0
## 4  4  EC01     M   76        TS     NA   62    2       NA           0
## 5  5  EC01     M   88        TS      5   65    2       NA           0
## 6  6  EC01     M   90        TV     11   80    2       NA           2
##   creditiTot numEsami crediti mediaEsami reiscrizioni matematica
## 1         36        4      33      20.75            1          1
## 2         15        2       9      25.50            1          0
## 3          0        0       0       0.00            1          0
## 4          0        0       0       0.00            0          0
## 5          0        0       0       0.00            0          0
## 6         15        2      15      22.50            1          0
```

```
##   s$numEsamiTot - s$numEsami
## 1                          1
## 2                          2
## 3                          0
## 4                          0
## 5                          0
## 6                          0
```

Ovviamente la variabile avrà un nome che richiamerà l'operazione effettuata. Se vogliamo cambiare il nome allora possiamo inserire quello desiderato (ad esempio idoneità) nella posizione corrispondente (la 17ma in questo caso) in `names()`:

```
names(s)[17]<-"idoneità"
head(s)
```

```
##   id corso sesso anno provincia scuola voto test votoTest numEsamiTot
## 1  1  EC01     M   85      TOGO     12   62    0       -1           5
## 2  2  EC01     F   90        TS      4  100    0        1           4
## 3  3  EC01     M   90        VE     11   60    2       NA           0
## 4  4  EC01     M   76        TS     NA   62    2       NA           0
## 5  5  EC01     M   88        TS      5   65    2       NA           0
## 6  6  EC01     M   90        TV     11   80    2       NA           2
##   creditiTot numEsami crediti mediaEsami reiscrizioni matematica idoneità
## 1         36        4      33      20.75            1          1        1
## 2         15        2       9      25.50            1          0        2
## 3          0        0       0       0.00            1          0        0
## 4          0        0       0       0.00            0          0        0
## 5          0        0       0       0.00            0          0        0
## 6         15        2      15      22.50            1          0        0
```

1.6 Merge di Dataframes

Un problema frequente, soprattutto nei casi reali, è quello di dover effettuare un **merge** (unione) di dataframe diversi. La condizione che devono rispettare i due dataframe è che abbiano almeno una variabile in comune.
Ad esempio immaginiamo di avere i seguenti sottoinsiemi di dati (a partire dal nostro dataframe originale).
Un dataframe `s1`:

```
s1<-subset(s,subset=(anno<75),select=-c(corso, sesso, anno))
```

e un dataframe `s2`:

```
s2<-subset(s,subset=(anno<75),select=-anno)
```

Per come li abbiamo costruiti `s1` e `s2` avranno in comune tutte le variabile, eccetto corso e sesso che sono contenute solo in `s2`.
Applicando la funzione `merge()` avremo le stesse righe (ovviamente) am nel datset finale compariranno tutte le variabili conetenute nei due insiemi di dati:

```
merge(s1,s2)

##    id provincia scuola voto test votoTest numEsamiTot creditiTot numEsami
## 1  20        TV      4   73    2       NA           0          0        0
## 2 334        TS      5   75    1     8.25           6         42        4
## 3  42        VE      4   83    2       NA           4         24        3
## 4  78        TS      4   66    2       NA           0          0        0
## 5  93        TS     11   63    2       NA           4         30        4
##   crediti mediaEsami reiscrizioni matematica idoneità corso sesso
## 1       0        0.0            0          0        0  EC01     M
## 2      36       21.5            1          1        2 EC01E     M
## 3      21       27.0            1          1        1  EC01     F
## 4       0        0.0            1          0        0  EC01     F
## 5      30       25.5            1          0        0  EC01     M
```

Qualora i due dataset da 'unire' non avessero le stesse etichette per le variabili è possibile usare degli argomenti aggiuntivi, in particolare `by.x` e `by.y`. Questi parametri indicheranno in nomi delle variabili chiave con cui si uniranno i due oggetti nel primo (`by.x`) e nel secondo dataframe `by.y`.
Ad esempio, chiamiamo la variabile 'id' con il nome 'identificativo' nel primo dataframe `s1`:

```
names(s1)[1]<-"identificativo"
s1

##     identificativo provincia scuola voto test votoTest numEsamiTot creditiTot
## 20              20        TV      4   73    2       NA           0          0
## 42              42        VE      4   83    2       NA           4         24
## 78              78        TS      4   66    2       NA           0          0
## 93              93        TS     11   63    2       NA           4         30
## 334            334        TS      5   75    1     8.25           6         42
##     numEsami crediti mediaEsami reiscrizioni matematica idoneità
## 20         0       0        0.0            0          0        0
## 42         3      21       27.0            1          1        1
## 78         0       0        0.0            1          0        0
## 93         4      30       25.5            1          0        0
## 334        4      36       21.5            1          1        2
```

e usiamo tali variabili per unirli:

```
merge(s1,s2,by.x=1,by.y=1)
```

Tuttavia, in questo modo `s1` e `s2` saranno considerati dataframe diversi e le loro variabili contrassegnate con `.x` e `.y` rispettivamente.

1.7 Attach() e detach()

Ulteriori funzioni disponibili per i dataframe sono le funzioni `attach()` e `detach()`. Esse rappresentano scorciatoie al richiamo delle variabili all'interno di un dataframe.
Ad esempio, se stiamo lavorando esclusivamente sul dataframe `s` il comando:

```
attach(s)

## The following object is masked _by_ .GlobalEnv:
##
##     id
```

Tecnicamente in questo modo aggiungeremo al *search-path* il dataframe `s` e questa operazione ci consentirà di richiamare direttamente le variabili, senza più l'uso dell'operatore `$`. Ad esempio per la variabile 'anno' basterà digitare il suo nome:

```
head(anno)
```

Una volta terminate le operazioni su `s` è necessario utilizzare il comando

```
detach(s)
```

che è semplicemente un *undo* di `attach()`.

1.8 Applicare funzioni sui dataframe

Analogamente alle liste,anche per i dataframe è possibile applicare le funzioni `lapply` e `apply`.
Teniamo sempre presente che un dataframe è un caso particolare di lista le cui componenti sono le colonne del dataframe. pertanto, se si usa `lapply()` su tale struttura dati al fine di applicare una specifica funzione `f()`, quest'ultima sarà applicata a ciascuna colonna restituendo in output dei valori anch'essi inseriti in una lista.
In casi particolari, è possibile usare la funzione `apply()` anche ai dataframe. La condizione è che essi siano formati da variabili dello stesso tipo (es. tutte numeriche).

Applicazione lapply(): ordinamento delle variabili di un dataframe .
Immaginiamo di voler ordinare tutte le variabili nel nostro dataframe `s` in ordine crescente. A tale scopo basterà utilizzare la funzione `sort()` all'interno del comando `lapply()`:

```
ssorted <- lapply(s,sort)
```

L'output **ssorted** sarà a sua volta una lista le cui componenti non sono altro che tutti vettori (variabili) di **s** ordinati in maniera crescente (in ordine alfabetico o numerico).
In realtà così non otteniamo l'ordinamento del dataframe ma delle singole variabili indipendentemente l'una dall'al
Se vogliamo ordinare il nostro dataframe rispetto ad una sua variabile preservando la struttura del dataframe. Ad esempio, ordiniamo **s** rispetto ad 'anno' mediante la seguente funzione:

```
sordered<-s[order(s$anno),]
head(sordered)

##      id corso sesso anno provincia scuola voto test votoTest numEsamiTot
## 93   93  EC01     M   65        TS     11   63    2       NA           4
## 78   78  EC01     F   70        TS      4   66    2       NA           0
## 42   42  EC01     F   72        VE      4   83    2       NA           4
## 334 334 EC01E     M   72        TS      5   75    1     8.25           6
## 20   20  EC01     M   74        TV      4   73    2       NA           0
## 4     4  EC01     M   76        TS     NA   62    2       NA           0
##     creditiTot numEsami crediti mediaEsami reiscrizioni matematica idoneità
## 93          30        4      30       25.5            1          0        0
## 78           0        0       0        0.0            1          0        0
## 42          24        3      21       27.0            1          1        1
## 334         42        4      36       21.5            1          1        2
## 20           0        0       0        0.0            0          0        0
## 4            0        0       0        0.0            0          0        0
```

2 I fattori (factors)

Abbiamo visto che all'interno di un dataframe coesistono differenti tipi di vettori (numerici, di caratteri, ecc.) che rappresentano diversi tipi di variabili. In statistica, come è noto, si distinguono sommariamente due principali gruppi di variabili: quelle numeriche e quelle categoriali. Come abbiamo visto, il nostro dataframe **s** contiene alcune variabili categoriali, ad esempio la variabile 'sesso'. In particolare, questa è composta da due categorie: "M" e "F".
In R, una variabile categoriale è specificata come **factor**. Le modalità di tale variabile invece come **levels**.
Nel caso di dati importati le variabili categoriali sono riconosciute direttamente come factors, tuttavia in altri casi è necessario costruire questa particolare struttura dati. Questo è possibile attraverso la funzione `factor()` che sostanzialmente trasforma i numeri in etichette:

```
#creiamo una variabile categoriale

f<-factor(c(1,1,2,3,1))
f
```

```
## [1] 1 1 2 3 1
## Levels: 1 2 3

 #creiamo una variabile categoriale e rinominiamo le etichette:
f<-factor(c(1,1,2,3,1), labels=c("gruppo1","gruppo2","gruppo3"))
f

## [1] gruppo1 gruppo1 gruppo2 gruppo3 gruppo1
## Levels: gruppo1 gruppo2 gruppo3
```

Con l'argomento `labels` si è semplicemente stabilita una corrispondenza tra l'etichetta (il numero) e le categoria in modo tale che nei risultati R visualizzi i nomi delle categorie invece che i semplici numeri. Questa operazione di **ricodifica** potrebbe tornare comoda in qualche circostanza, ad esempio al momento di produrre grafici, ma è bene ribadire che l'uso di `factor()` con il suo solo primo argomento specificato è già di per se sufficiente per definire una variabile categoriale. Le modalità di tale variabile, sia sotto forma di numero che di etichetta, sono chiamate `levels` in R. Infatti con la funzione `levels()` possiamo richiamare quelle che sono le modalità della variabile qualitativa:

```
l<-levels(f)
l

## [1] "gruppo1" "gruppo2" "gruppo3"
```

E' fondamentale notare che l'ordine dei livelli del fattore determinerà l'apparizione di questi in grafici, tabelle e output di funzioni che utilizzano tale informazione.
Se volessimo cuustomizzare l'ordine di apparizione dei livelli basterà rispecificarli, utilizzando il nuovo ordine desiderato, all'interno della funzione `factor`:

```
f1<-factor(f, levels=c("gruppo2", "gruppo3", "gruppo1"))
f1

## [1] gruppo1 gruppo1 gruppo2 gruppo3 gruppo1
## Levels: gruppo2 gruppo3 gruppo1
```

La funzione `factor()` è particolarmente utile quando il file di dati da importare contiene variabili categoriali con modalità espresse in forma numerica piuttosto che esplicitamente dalle categorie. In tal caso sarà sufficiente riassegnare alla variabile in questione la sua versione "factor". Ad esempio, assumendo la variabile "`v1`" nell'ipotetico dataframe "`dati`", allora il seguente comando:

```
dati[,"v1"]<-factor(dati[,"v1"])
```

"trasformerà" gli elementi della variabile (vettore) "`v1`" in etichette.
La corretta specificazione degli argomenti della funzione `factor()` consentono ulteriori operazioni utili per codificare variabili categoriali; ad esempio l'argomento `ordered` (o la funzione `ordered()`)

serve a definire variabili ordinali e l'argomento `levels` (già visto sopra) può anche tornare utile per eliminare dall'analisi qualche categoria. Ad esempio considerando la variabile categoriale `f` creata in precedenza attraverso il seguente comando escluderemo la categoria "3":

```
factor(c(1,1,2,3,1),levels=1:2) #escludi la categoria 3

## [1] 1    1    2    <NA> 1
## Levels: 1 2
```

o con quest'altro la categoria (ri-etichettata con la label) "gruppo 2":

```
factor(f,levels=c("gruppo1","gruppo3")) #escludi "gruppo2"

## [1] gruppo1 gruppo1 <NA>    gruppo3 gruppo1
## Levels: gruppo1 gruppo3
```

I fattori sono alla base di numerose operazioni e funzioni in R, incluse quelle ottenute mediante la funzione `table()`.

2.1 Applicazione: ricodifica variabile quantitativa in classi

A volte può essere necessario "categorizzare" una variabile quantitativa, ad esempio formando classi di età da una variabile che riporta l'età esatta di ogni osservazione.
Nel nostro caso, nel dataframe `s` abbiamo alcune variabili che si prestano a tale trasformazione. Ad esempio proviamo a trasformare la variabile "mediaEsami" in una variabile categoriale. Prima di fare ciò è importante ottenere alcune statistiche della variabile di interesse. Mediante la funzione `summary()` otteniamo alcune informazioni:

```
summary(s$mediaEsami)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   18.00   22.00   17.57   24.52   30.00
```

La cosa importante è notare che il minimo valore assunto dalla variabile è 0 e che il massimo è (ovviamente) 30.
Ora possiamo definire gli intervalli delle nostre classi. Ad esempio immaginiamo di voler formare quattro classi di voto: nessun esame (0-17.99), basso (18-22.00), medio (22.01-27.00) e alto (27.01-30).
Questa operazione in R la si effettua mediante la funzione `cut()` e avendo l'accortezza di inserire l'estremo della prima classe con un valore inferiore al minimo osservato (nel nostro caso possiamo usare -1):

```
mediaEsami.cat<-cut(s$mediaEsami, breaks=c(-1, 17.99,22.01,27.01,30),labels=c("noEsami", "Bas
mediaEsami.cat
```

```
##   [1] Basso   Medio   noEsami noEsami noEsami Medio   Medio   Alto    Medio
##  [10] Alto    Medio   Medio   Medio   Medio   Medio   noEsami Basso   noEsami
##  [19] Medio   noEsami Medio   Medio   Medio   Medio   noEsami noEsami noEsami
##  [28] noEsami Medio   Medio   Basso   Medio   Basso   Medio   noEsami noEsami
##  [37] noEsami Basso   Basso   noEsami Basso   Medio   Medio   Basso   Medio
##  [46] Alto    Medio   Basso   Medio   Medio   Medio   Medio   noEsami Medio
##  [55] noEsami Basso   Medio   noEsami Basso   Medio   Basso   noEsami Basso
##  [64] Basso   Basso   Basso   Basso   Alto    Medio   Medio   Medio   Medio
##  [73] Basso   noEsami Medio   Medio   Medio   noEsami Medio   noEsami Medio
##  [82] noEsami Medio   Basso   Medio   Basso   noEsami Medio   noEsami Medio
##  [91] Medio   noEsami Medio   Basso   Basso   Medio   noEsami Basso   Basso
## [100] Medio   Medio   noEsami noEsami Medio   Alto    noEsami Medio   Medio
## [109] Basso   Basso   Medio   Basso   Basso   noEsami noEsami Medio   Medio
## [118] Basso   noEsami Alto    Basso   Basso   noEsami Medio   Medio   noEsami
## [127] Basso   Basso   noEsami Medio   Basso   Alto    noEsami Medio   Medio
## [136] Medio   Medio   Medio   Basso   Medio   Medio   Alto    Medio   Basso
## [145] Medio   Basso   Medio   Basso   noEsami Medio   Medio   Medio   Medio
## [154] Medio   noEsami Medio   Medio   noEsami noEsami noEsami Basso   Medio
## [163] Medio   Basso   Medio   Basso   noEsami Medio   Basso   Basso   noEsami
## [172] noEsami Basso   Basso   Medio   Basso   Medio   noEsami Medio   Alto
## [181] noEsami noEsami Alto    Medio   Basso   Basso   Basso   Basso   Medio
## [190] Medio   Basso   Basso   Medio   Basso   Basso   Medio   Medio   Basso
## [199] Medio   noEsami Basso   noEsami Medio   Basso   Basso   Basso   Basso
## [208] noEsami Medio   Medio   noEsami Medio   Basso   Medio   Basso   Basso
## [217] Medio   noEsami noEsami Medio   Basso   noEsami noEsami Medio   noEsami
## [226] Medio   noEsami Basso   Basso   Basso   Basso   Basso   Basso   Medio
## [235] Medio   Medio   Basso   Basso   noEsami Basso   Basso   Medio   Medio
## [244] Basso   Medio   Basso   Medio   Medio   noEsami Alto    Medio   noEsami
## [253] Medio   Medio   Medio   Basso   noEsami Basso   Medio   Basso   noEsami
## [262] Medio   Basso   Basso   noEsami Basso   Medio   Medio   Basso   noEsami
## [271] noEsami noEsami Basso   noEsami Basso   noEsami noEsami Medio   Medio
## [280] noEsami Medio   Basso   Alto    noEsami noEsami noEsami Medio   Medio
## [289] Medio   noEsami Medio   noEsami noEsami Medio   noEsami Medio   Basso
## [298] Alto    Medio   noEsami Medio   Medio   noEsami Alto    Medio   noEsami
## [307] noEsami Alto    Basso   Medio   Medio   noEsami Basso   Medio   Basso
## [316] noEsami Medio   Medio   Alto    Basso   Medio   Medio   Basso   Basso
## [325] Medio   Basso   Medio   Basso   Medio   Medio   Medio   Basso   Medio
## [334] Basso   Basso   Medio   Basso   Medio   noEsami Medio   Basso   noEsami
## [343] Basso   noEsami Medio   Medio   Basso   Medio   Basso   Basso   Basso
## [352] Medio   Medio   Medio   Medio   Basso
## Levels: noEsami Basso Medio Alto
```

La funzione `cut()` crea una variabile categoriale (fattore) dividendo la variabile numerica che figura come suo argomento in intervalli tipo $]xi; xi + 1]$ dove gli estremi possono o meno essere inclusi in funzione di come vengono specificati gli argomenti `include.lowest` e `right`. In `breaks` possono

essere espressi sia gli estremi degli intervalli (come nell'esempio sopra) o il numero degli intervalli desiderato, mentre `labels` serve soltanto a nominare le etichette della variabile categoriale appena creata; ponendo `labels=FALSE` i semplici interi $1, 2, \ldots$ vengono utilizzati come etichette.

2.2 Applicare funzioni sui fattori

Lavorando su strutture dati come i fattori si rendono disponibili una serie di funzioni dedicate: `tapply()`, `split()` e `by()`.

La funzione `tapply()`. Consideriamo le variabili 'voto' e un fattore come ad esempio la variabile 'sesso'. La funzione `tapply()` consente nell'applicare una funzione `g()` ad una variabile ripartendo i risultati sulla base dei livelli di un fattore. Immaginiamo di voler calcolare il voto medio di diploma per ogni 'gruppo' definito dalla variabile 'sesso', ovvero il voto medio di maschi e femmine.
La struttura base della funzione tapply è la seguente: `tapply(x,f,g)`, dove `x` è un vettore, `f` un fattore o una lista di fattori e `g` è una funzione. Nel nostro semplice esempio la funzione `g()` è la funzione built-in `mean()`:

```
tapply(s$numEsamiTot, s$sesso, mean)

##        F        M
## 3.257310 2.735135
```

Se volessimo ottenere risultati basati su un ulteriore fattore basta inserire i fattori `f` come lista del tipo `list(<factor1>, <factor2>, ...)`. L'unica restrizione è che i fattori in `f` devono avere la stessa dimensione.
Ad esempio aggiugniamo la variabile 'corso':

```
tapply(s$numEsamiTot, list(s$sesso, s$corso), mean)

##       EC01    EC01E     EC11    EC11E     EC21
## F 3.056338 4.294118 3.400000 3.692308 2.133333
## M 2.888889 4.000000 2.546667 3.250000 2.050000
```

L'operazione effettuata mediante `tapply()` consiste nel suddividere (temporaneamente) il vettore `x` in tanti gruppi quanti sono i livelli (o le combinazioni dei livelli) del fattore considerato e poi applicare la funzione `g` ai sotto-vettori risultanti da tale suddivisione.

La funzione `split()`. La funzione `split()` si ferma al primo stadio della funzione `tapply()`. Infatti tale funzione può essere utilizzata per dividere i valori di una variabile sulla base di un certo fattore (anche qui i gruppi sono identificati dai livelli del fattore).
Ad esempio suddividiamo la variabile 'voto' rispetto al fattore 'sesso' nel nostro dataframe `s`:

```
split(s$voto, s$sesso)
```

```
## $F
##   [1] 100  81  90  67 100  70  71  61  70  75  60  67  73  63  76  62  83  60
##  [19]  98  85  71  62  95  60  60 100  90  74  84  73  62 100  68 100  67  65
##  [37]  83  66 100  70  87  72  65  89  68  70  65 100  62  62  81  87  76  62
##  [55]  75  60  73  75  79  76 100  95  70  92  75  84  70  80  65 100  61  75
##  [73]  63  60  66  66  96  98  60  90  75  74  61  61  87  64  77  87  73  82
##  [91]  76  87  85 100  79  74  73 100  92  85  66  86  75  85  60  72  70  89
## [109]  73  85  75  60  90  65  64 100  96  60  91  66  82  87  87  65  78  96
## [127]  82  82  84  82  75 100  91  62  71  82 100  66  70  83  94  76  75  87
## [145]  65 100 100  87  80  98  71  NA  98  73  85  65  60  76  92  61  82  73
## [163]  95  78  68  97  90  NA  71 100  96
##
## $M
##   [1]  62  60  62  65  80  77  69  66  75  70  60  73  67  70  77  66  75  66
##  [19]  71  70  64  76  65  75  NA  77  65  73  78  80  65  78  62  87  86  60
##  [37]  87  67  68  70  74  82  75  70  61  98  65  63  63  65  74 100  69  80
##  [55]  77  80  80  73  83  74 100  72  62  62  83  62  60  81 100  80  86  68
##  [73]  63  67  66  66  64  87  62  78  74  62  87  75  72  62  66  71  64  74
##  [91]  83  72  81  68  82  66  76  62  79  78  76  72  62  73  74  61  75  62
## [109]  62  60  65  85  83  70  60  60  70  60  60  67  78  73  88  64  83  84
## [127]  69  67  60  62  60  73  72  62  67  83  74  61  90 100  98  87  70  63
## [145] 100  80  71  73  68  67  79  82  82  80  62  76  80  68  63  81  68  80
## [163]  72  67  68  74  85  73  62  85  64  68 100  60  80  75  82  66  66  60
## [181]  64  73  66  74  96
```

La funzione `by()`. In alcuni casi la funzione `tapply()` non può essere applicata. In particolare, `tapply()` richiede che il primo argomento sia un vettore e spesso ciò non è quello che ci serve. Ad esempio immaginiamo di voler stimare un modello di regressione semplice per l'effetto della variabile 'voto' sulla variabile 'mediaEsami' per i maschi e le femmine. In questo caso dobbiamo usare la funzione `lm()` in cui in input è richiesta una matrice (o un dataframe). Non essendo tale oggetto un vettore non è possibile utilizzare `tapply()` ma una funzione più flessibile, ossia `by()`. La sua logica è simile ad `tapply` ma si applica anche ad oggetti più complessi dei vettori. Nell'esempio del modello di regressione semplice cui abbiamo accennato il suo uso è il seguente:

```
by(s,s$sesso, function(s) lm(s$mediaEsami~s$voto))
```

```
## s$sesso: F
##
## Call:
## lm(formula = s$mediaEsami ~ s$voto)
##
## Coefficients:
## (Intercept)       s$voto
```

```
##      4.6607       0.1776
##
## ------------------------------------------------------------
## s$sesso: M
##
## Call:
## lm(formula = s$mediaEsami ~ s$voto)
##
## Coefficients:
## (Intercept)       s$voto
##      6.8498       0.1352
```