Tutorial di R

F. Gasperoni, N. Tarabelloni

Contents

1 R

Lo strumento con cui condurremo le nostre analisi statistiche è R. Dal sito web del Progetto CRAN si legge:

> R is GNU S, a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.

Quindi R indica sia un linguaggio che un ambiente di programmazione in cui sviluppare ed utilizzare metodi statistici per l'analisi di dati. R è uno dei principali strumenti attualmente utilizzati per questo scopo, e sicuramente uno dei più semplici e potenti al contempo. Tra le alternative, si possono sicuramente citare Python, Matlab e SAS.

R viene distribuito come software open-source per tutti i principali sistemi operativi (tra cui Windows, Mac OS X e Linux), e può essere usato sia da command-line sia tramite un'interfaccia grafica (*GUI*, Graphical User Interface) minimale. Può essere scaricato dalla pagina del CRAN (Comprehensive R Archive Network), all'indirizzo https://cran.r-project.org. Il nostro consiglio, tuttavia, è di installare insieme ad R un programma di sviluppo (*IDE*, Integrated Development Environment), molto più ricco e versatile, che permetta un utilizzo comodo ed efficace di R.

Ad oggi l'IDE migliore è senza dubbio Rstudio, che si può ottenere gratuitamente nella distribuzione Desktop all'indirizzo https://www.rstudio.com/products/RStudio/#Desktop, e presenta un'interfaccia simile a quella di Matlab.

1.1 Filosofia di utilizzo di R:

L'utilizzo più tipico di R prevede la lettura di dati, oggetto dell'analisi statistica, dall'esterno, la loro manipolazione attraverso una serie di comandi, e opzionalmente il salvataggio di risultati/immagini.

L'interazione con l'esterno (computer su cui R sta girando) è una parte importante, quindi dovete abituarvi a chiedervi e sapere quale *working directory* R sta usando per l'input/output dei dati. Nel terminale di R:

```
# Per conoscere la directory di lavoro corrente
getwd()

# Per impostare la directory di lavoro corrente
setwd('/Directory/dove/voglio/tenere/dati/e/risultati')

# Per elencare i files della directory di lavoro corrente
list.files()

# Per elencare le directory nella directory di lavoro corrente
list.dirs()
```

In alternativa potete impostare la directory di lavoro graficamente, ad esempio su RStudio fate `Session > Set Working Directory > Choose Directory`. In ogni caso, siete caldamente invitati ad usare percorsi e nomi per i file/directory **senza** spazi e caratteri speciali (accenti, segni d'interpunzione, etc).[1]

I comandi R possono essere eseguiti sia scrivendoli nel terminale, uno dopo l'altro, sia leggendoli ed eseguendoli dall'editor R/RStudio (a seconda del sistema operativo con i comandi `CTRL + R` o `CTRL/CMD + Enter`).

1.2 Uso di R sulle macchine del Politecnico

Potete usare R sulle macchine del Politecnico attraverso Virtual Desktop. In questo caso, R verrà eseguito tramite una sessione remota e il setup della working directory verrà mostrato a mano durante la lezione di laboratorio.

1.3 R e i pacchetti

R è un software open-source che raccoglie contributi dalla comunità di utilizzatori (statistici, economisti, biologi, informatici, ...) e organizza il sistema di collaborazione attorno ai *pacchetti*, raccolte di codici che implementano una particolare classe di metodi o che aggiungono un certo numero di funzionalità alla distribuzione base di R. I pacchetti sono resi disponibili attraverso particolari repositories (lo stesso CRAN, Biodonductor o anche Bitbucket, GitHub, ...).

L'organizzazione della collaborazione tramite pacchetti è una delle chiavi del successo di R, perché permette di raccogliere i contributi dai gruppi di ricerca e dagli sviluppatori di tutto il mondo e di metterli a disposizione di tutti gli utilizzatori.

```
# Lista dei pacchetti instalati
utils::installed.packages()

# Lista (molto lunga) dei pacchetti disponibili, da scaricare e installare
utils::available.packages()

# Per installare un pacchetto disponibile:
install.packages('MASS')
```

[1] In generale, questa è una *good practice* trasversale a cui aderire anche nel vostro futuro.

```
  # Per caricarlo
  library('MASS')

  # Per rimuoverlo (non vogliamo farlo!)
  # remove.packages('MASS')
```

2 Elementi di base di R

2.1 Variabili

```
  # Informazioni sulla sessione di R corrente
  sessionInfo()
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.6 (El Capitan)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] ggplot2_2.1.0      devtools_1.11.1    pryr_0.1.2
## [4] Rcpp_0.12.5        lattice_0.20-33    data.table_1.9.6
## [7] RColorBrewer_1.1-2
##
## loaded via a namespace (and not attached):
##  [1] knitr_1.13       magrittr_1.5     munsell_0.4.3    colorspace_1.2-6
##  [5] stringr_1.0.0    plyr_1.8.3       tools_3.3.0      grid_3.3.0
##  [9] gtable_0.2.0     withr_1.0.1      htmltools_0.3.5  yaml_2.1.13
## [13] digest_0.6.9     formatR_1.4      codetools_0.2-14 memoise_1.0.0
## [17] evaluate_0.9     rmarkdown_0.9.6  stringi_1.1.1    scales_0.4.0
## [21] chron_2.3-47

  # Lista delle variabili definite nell'ambiente corrente
  ls()
## character(0)

  # Dichiariamone due
  a = 1
  b = 2

  ls()
## [1] "a" "b"

  # Rimuoviamo a
  rm( list = 'a' )

  ls()
## [1] "b"
```

```
  # Rimuoviamo tutto
  rm( list = ls() )

  ls()
## character(0)
```

Per avere più informazioni su un comando, basta usare la funzione `help('comando')`, ad esempio `help(ls)`. In alternativa, `?comando` oppure, se non siete certi del nome del comando, `??nome` cercherà `nome` nella documentazione.

Le variabili in `R` non devono necessariamente essere dichiarate e, soprattutto, possono cambiare tipo durante l'esecuzione (tipizzazione non forte):

```
  a = 1
  class( a )
## [1] "numeric"

  a = 1 : 10
  a
##  [1]  1  2  3  4  5  6  7  8  9 10

  a = list( a, 1, 'pippo', list( 'pluto', 'paperino' ) )
  a
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] "pippo"
##
## [[4]]
## [[4]][[1]]
## [1] "pluto"
##
## [[4]][[2]]
## [1] "paperino"

  # Possiamo anche verficare la classe della variabile durante l'esecuzione
  is(a)
## [1] "list"   "vector"
  is.list( a )
## [1] TRUE
```

2.2 Liste, array, vettori, matrici

Riportiamo dal sito http://adv-r.had.co.nz/Data-structures.html l'utilissima tabella chiarificatrice sulle strutture dati di `R`:

	Omogenea	Eterogenea
1D	Atomic vector	List
2D	Matrix	Data frame

	Omogenea	Eterogenea
ND	Array	

Sia le `list` che gli `atomic vectors` sono chiamati `Vector`.

```
  # Un semplice vettore
  c(1,2,3,4)
## [1] 1 2 3 4

  # Un altro modo di ottenere un vettore
  seq( 0, 1, length.out = 10 )
##  [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
##  [8] 0.7777778 0.8888889 1.0000000
  seq( 0, 1, by = 0.1 )
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

  # Questo vettore viene convertito a vettore di stringhe
  a = c(1,2,3,4,'pippo')
  a
## [1] "1"     "2"     "3"     "4"     "pippo"

  # Una semplice lista
  l = list( '1', c(1,2,3,4), 'pluto' )
  l
## [[1]]
## [1] "1"
##
## [[2]]
## [1] 1 2 3 4
##
## [[3]]
## [1] "pluto"

  # Una matrice
  matrix( data = 1 : 10, nrow = 2, ncol = 5, byrow = TRUE )
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10

  # Attenti ai valori di default degli argomenti!
  matrix( 1 : 10, nrow = 2, ncol = 5, byrow = TRUE )
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10

  # Usando le chiamate alle funzioni con il nome dei parametri
  # possiamo anche modificare l'ordine (correndo dei rischi!)
  M = matrix( byrow = TRUE, 1 : 10, ncol = 5, nrow = 2 )
  M
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
  # Tre array di dimensionalità crescente
  array( 1 : 12, dim = 12 )
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
  array( 1 : 12, dim = c( 3, 4 ) )
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
  A = array( 1 : 12, dim = c( 3, 2, 2 ) )
  A
## , , 1
##
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
##
## , , 2
##
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   11
## [3,]    9   12
```

Tra i metodi di base ci sono quelli che permettono di accedere a sottoinsiemi delle strutture dati e di valutarne le dimensioni.

```
  # Per i vettori e le liste
  length( a )
## [1] 5
  a[1]
## [1] "1"
  a[1:2]
## [1] "1" "2"
  a[c(1,3,4)]
## [1] "1" "3" "4"

  l[3]
## [[1]]
## [1] "pluto"


  # Per le matrici
  dim( M )
## [1] 2 5
  nrow( M )
## [1] 2
  ncol( M )
## [1] 5
  M[ 1 : 2, 1 : 3 ]
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    6    7    8
  M[ , 1 ]
```

```
## [1] 1 6
  M[ 2, ]
## [1]  6  7  8  9 10

  # L'accesso agli array dipende dalla loro dimensione:
  dim( A )
## [1] 3 2 2
  A[ 1, 2, 1 ]
## [1] 4
```

2.3 factor

Una classe molto importante è `factor`, che viene usata generalmente per rappresentare in `R` le variabili categoriche, sia quantitative che qualitative.

Essenzialmente, una variabile `factor` è una lista di occorrenze di valori, ognuno rappresentato attraverso un'etichetta, appartenenti ad una tabella prefissata.

Una variabile `factor` si crea specificando un insieme di livelli (`levels`), che rappresentano l'insieme dei diversi valori validi di cui si possono trovare occorrenze, e un insieme di etichette (`labels`) con cui vogliamo indicare i diversi livelli. In questo modo possiamo rappresentare variabili categoriche tramite una stratificazione dei loro livelli, che vengono poi indicati per maggiore chiarezza con un insieme di stringhe. Occorrenze che non appartengono alla lista prefissata sono resi con l'etichetta `NA`.

Per capire meglio tutto questo, vediamo un esempio:

```
  # Creiamo una variabile factor, che contiene occorrenze di tre valori, basso, medio e alto,
  altezza = factor( c(3, 1, 2, 3, 4, 1, 4 ), levels = c( 1, 2, 3 ),
                    labels = c('basso', 'medio', 'alto' ) )
  altezza
## [1] alto  basso medio alto  <NA>  basso <NA>
## Levels: basso medio alto
```

Come si può vedere, all'occorrenza `4` corrisponde `NA` perché non è presente alcun livello `4` nella definizione della variabile (e di conseguenza, non è presente nessuna etichetta).

Le variabili `factor` possono anche essere automaticamente costruite a partire da vettori numerici o di stringhe:

```
  numbers = c(1,2,3,1,3,1,2,3,1,3)
  strings = c('pippo','pluto','paperino')
  class( numbers ); class( strings )
## [1] "numeric"
## [1] "character"

  # Conversione a factor
  numbers_fac = as.factor( numbers )
  strings_fac = as.factor( strings )
  class( numbers_fac ); class( strings_fac )
## [1] "factor"
## [1] "factor"

  # I livelli delle variabili numeriche vengono automaticamente ordinati in senso crescente
  numbers_fac
##  [1] 1 2 3 1 3 1 2 3 1 3
## Levels: 1 2 3
```

```r
  # Nel caso di stringhe, l'ordinamento di default è lessico-grafico
  strings_fac
## [1] pippo    pluto    paperino
## Levels: paperino pippo pluto
```

2.4 data.frame

Strutture dati *bidimensionali* (in forma tabellare) che contengono elementi di tipo differente (come numeri o stringhe) sono gli **oggetti**[2] di *classe* il `data.frame`. Anche se possono essere creati senza problemi dall'utente, tipicamente sono il risultato della lettura di dataset salvati in un file (generalmente un file di testo) esterno ad R.

```r
  # Creiamo un data.frame fittizio
  D = data.frame( numeri = 1 : 5, parole = c('uno', 'due', 'tre', 'quattro', 'cinque' ) )

  # Scriviamolo
  write.table( D, file = 'dataset.txt' )

  rm( list = 'D' )

  # Leggiamolo
  D = read.table( 'dataset.txt', header = TRUE )
  D
##   numeri  parole
## 1      1     uno
## 2      2     due
## 3      3     tre
## 4      4 quattro
## 5      5  cinque

  # Nome di colonne e righe
  colnames( D )
## [1] "numeri" "parole"
  rownames( D )
## [1] "1" "2" "3" "4" "5"

  D[ 1, ]
##   numeri parole
## 1      1    uno
  # Le colonne di stringhe vengono rappresentate come factor (ricordatelo!!)
  D[ 1, 1 ]
## [1] 1

  # Accesso alle variabili
  D$numeri; D$parole
## [1] 1 2 3 4 5
## [1] uno     due     tre     quattro cinque
## Levels: cinque due quattro tre uno
```

[2] In soldoni, gli *oggetti* sono particolari elementi di un programma che costituiscono una rappresentazione informatica di un concetto astratto o di un ente composito non esprimibile dai tipi nativi del linguaggio (ad esempio: un dataset). Potete pensarli come generalizzazioni delle `struct` di C.

```
  # Col solo nome, non potete accedere alle variabili
  numeri[1]
## Error in eval(expr, envir, enclos): object 'numeri' not found

  # Rendiamo visibile il nome delle variabili (colonne)
  attach( D )

  # Questo permette di accedere direttamente alle variabili del data.frame
  numeri[1]
## [1] 1
  parole[1:2]
## [1] uno due
## Levels: cinque due quattro tre uno

  # Facciamo il detach del data.frame (fare sempre!) per mascherarle di nuovo
  detach( D )
```

2.5 Indici statistici di posizione/dispersione

I principali indici statistici di posizione/dispersione (media, mediana, max, min, range, IQR, quantili, etc.) si possono calcolare con i seguenti comandi R:

```
  # Leggiamo un data.frame built-in di R
  help( USArrests )

  # Dataset su arresti negli stati USA
  data( USArrests )

  attach( USArrests )

  # Media campionaria
  mean( Murder )
## [1] 7.788
  colMeans( USArrests )
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232
  apply( USArrests, 2, mean )
##   Murder  Assault UrbanPop     Rape
##    7.788  170.760   65.540   21.232

  # Varianza
  var( Assault ) # ATTENZIONE: è quella campionaria!
## [1] 6945.166
  apply( USArrests, 2, var )
##     Murder    Assault   UrbanPop       Rape
##   18.97047 6945.16571  209.51878   87.72916

  # Deviazione standard
  sd( Assault )
## [1] 83.33766

  # Mediana
  median( Rape )
```

```
## [1] 20.1

  # Quantili
  quantile( Rape )
##     0%    25%    50%    75%   100%
##  7.300 15.075 20.100 26.175 46.000
  quantile( Rape, probs = c( 0.49, 0.67 ) )
##    49%    67%
## 20.002 23.813

  # min, max, range, IQR
  min( Assault )
## [1] 45
  max( Assault )
## [1] 337

  # N.B.: range non calcola il range per come lo intendiamo noi!
  diff( range( Assault ) )
## [1] 292

  IQR( Assault )
## [1] 140

  # Una visione d'insieme
  summary( USArrests )
##      Murder          Assault         UrbanPop          Rape
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00

  # Ricordarsi!
  detach( USArrests )
```

2.6 Manipolazione di dati

Spessissimo vi si presenterà l'esigenza di manipolare i dati in un `data.frame` per poter estrarre informazioni di vostra utilità.

```
  # Riempiamo un data.frame
  set.seed( 0618033 )
  D = data.frame( altezza = rnorm( 10, 165, 1 ), peso = rnorm( 10, 70, 2 ),
                  eta = sample( seq(25,50), size = 10 ),
                  sesso = sample( c('M','F'), 10, replace = TRUE ))

  # Diamo un'occhiata ai dati
  summary( D )
##     altezza           peso            eta        sesso
##  Min.   :162.7   Min.   :66.24   Min.   :25.00   F:3
##  1st Qu.:164.4   1st Qu.:67.65   1st Qu.:30.25   M:7
##  Median :164.8   Median :68.77   Median :35.50
```

```
##  Mean   :164.8   Mean   :68.87   Mean   :36.10
##  3rd Qu.:165.7   3rd Qu.:69.30   3rd Qu.:40.75
##  Max.   :166.3   Max.   :72.20   Max.   :48.00
```

```
  # Estriamo gli uomini
  D[ which( D$sesso == 'M' ), ]
```

```
##     altezza     peso eta sesso
## 2  164.8544 68.72403  40     M
## 3  164.5518 69.14335  46     M
## 4  165.7770 67.37736  35     M
## 7  164.6823 66.24261  41     M
## 8  165.5271 68.81866  29     M
## 9  166.3278 72.20060  25     M
## 10 166.2951 69.34872  36     M
```

```
  subset( D, sesso == 'M' )
```

```
##     altezza     peso eta sesso
## 2  164.8544 68.72403  40     M
## 3  164.5518 69.14335  46     M
## 4  165.7770 67.37736  35     M
## 7  164.6823 66.24261  41     M
## 8  165.5271 68.81866  29     M
## 9  166.3278 72.20060  25     M
## 10 166.2951 69.34872  36     M
```

```
  # Estraiamo il peso degli uomini
  D[ which( D$sesso == 'M' ), 'peso' ]
```

```
## [1] 68.72403 69.14335 67.37736 66.24261 68.81866 72.20060 69.34872
```

```
  D[ which( D$sesso == 'M' ), 2 ]
```

```
## [1] 68.72403 69.14335 67.37736 66.24261 68.81866 72.20060 69.34872
```

```
  subset( D, sesso == 'M', 'peso' )
```

```
##        peso
## 2  68.72403
## 3  69.14335
## 4  67.37736
## 7  66.24261
## 8  68.81866
## 9  72.20060
## 10 69.34872
```

```
  # Estraiamo peso ed altezza degli uomini
  D[ which( D$sesso == 'M' ), c( 'peso', 'altezza' ) ]
```

```
##        peso  altezza
## 2  68.72403 164.8544
## 3  69.14335 164.5518
## 4  67.37736 165.7770
## 7  66.24261 164.6823
## 8  68.81866 165.5271
## 9  72.20060 166.3278
## 10 69.34872 166.2951
```

```
  subset( D, sesso == 'M', c( 'peso', 'altezza' ) )
```

```
##        peso  altezza
## 2  68.72403 164.8544
## 3  69.14335 164.5518
```

```
## 4  67.37736 165.7770
## 7  66.24261 164.6823
## 8  68.81866 165.5271
## 9  72.20060 166.3278
## 10 69.34872 166.2951

  # Aggiungiamo una colonna al dataset con una flag `giovane`
  D$giovane = 'no'
  D[ which( D$eta < 30 ), 'giovane' ] = 'si'

  # Le variabili categoriche vengono rappresentate come factor
  D$sesso
##  [1] F M M M F F M M M M
## Levels: F M
  # Se le aggiungiamo noi, dobbiamo forzare R a rappresentarle così
  D$giovane
##  [1] "si" "no" "no" "no" "no" "no" "no" "si" "si" "no"
  D$giovane = as.factor( D$giovane )
  D$giovane
##  [1] si no no no no no no si si no
## Levels: no si

  # Eliminiamola
  D$giovane = NULL

  # Eliminiamo le osservazioni di chi ha età superiore a 40 anni
  D = D[ - which( D$eta > 40 ), ]
  # ed ora di chi ha più di 50 anni
  D = D[ - which( D$eta > 50 ), ]
```

In fase di manipolazione possono tornare molto utili le funzioni della famiglia `*apply`, e in particolare `sapply`, `lapply`, `apply` e `tapply`.

`sapply` e `lapply` applicano una funzione, tipicamente definita dall'utente, ad una sequenza. `lapply` restituisce l'output in forma di lista, mentre `sapply` cerca di restituirlo come vettore o matrice:

```
  # Creiamo un vettore fittizio, per studiare sapply e lapply
  v = c( 1, 2, 3, 4 )

  sapply( v, function( x ) ( x + 1 ) )
## [1] 2 3 4 5

  # Ora creiamo una lista
  l = list( diag( 1 ), diag( 2), diag( 3 ) )

  sapply( l, dim )
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    2    3
  lapply( l, dim )
## [[1]]
## [1] 1 1
##
## [[2]]
## [1] 2 2
```

```
##
## [[3]]
## [1] 3 3
```

La funzione `apply`, più generica delle due precedenti, applica una funzione (tipicamente definita dall'utente, ma anche una di quelle standard di `R`) ad un array (2D, 3D, ..., ND) o ad una matrice (2D):

```
  # Creiamo una matrice fittizia per capire meglio il comportamento di apply
  D = matrix( 1 : 10, nrow = 4, ncol = 10, byrow = TRUE )
  D
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    1    2    3    4    5    6    7    8    9    10
## [2,]    1    2    3    4    5    6    7    8    9    10
## [3,]    1    2    3    4    5    6    7    8    9    10
## [4,]    1    2    3    4    5    6    7    8    9    10

  apply( D, 1, mean )
## [1] 5.5 5.5 5.5 5.5
  apply( D, 2, mean )
##  [1]  1  2  3  4  5  6  7  8  9 10

  # In questo caso avremmo potuto usare la funzione rowMeans e colMeans
  rowMeans( D )
## [1] 5.5 5.5 5.5 5.5
  colMeans( D )
##  [1]  1  2  3  4  5  6  7  8  9 10

  # Però a volte può capitare di voler fare operazioni più complicate
  apply( D, 1, var )
## [1] 9.166667 9.166667 9.166667 9.166667
  apply( D, 1,  function( x ) ( mean( x ) + 2 * min( x ) ) )
## [1] 7.5 7.5 7.5 7.5

  # Domanda: sappiamo già cosa fa apply( D, 2, var) ?
```

L'uso di `tapply` è lievemente più elaborato: `tapply` applica una funzione (eventualmente definita dall'utente) ai sottogruppi di una struttura dati (e.g. vettore) secondo i lvelli di una variabile `factor` ausiliaria.

```
  # Usiamo ad esempio un dataset built-in di R
  help(chickwts)

  data(chickwts)

  head( chickwts )
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean

  attach( chickwts )

  # Calcoliamo la media del peso corrispondenti alle varie classi di alimentazione
```

```r
  tapply( weight, feed, mean )
##    casein horsebean   linseed  meatmeal   soybean sunflower
##  323.5833  160.2000  218.7500  276.9091  246.4286  328.9167

  # Calcoliamo la deviazione standard campionaria
  tapply( weight, feed, function( x ) ( sd = sd( x ) ) )
##    casein horsebean   linseed  meatmeal   soybean sunflower
##  64.43384  38.62584  52.23570  64.90062  54.12907  48.83638

  # tapply può anche essere usato in maniera piuttosto complessa
  tapply( weight, feed,
          function( x )( list( left =  mean( x ) +
                                 1.96 * sd( x ) / sqrt( length( x ) ),
                               center = mean( x ),
                               right = mean( x ) -
                                 1.96 * sd( x ) / sqrt( length( x ) ) ) ) )
## $casein
## $casein$left
## [1] 360.0402
##
## $casein$center
## [1] 323.5833
##
## $casein$right
## [1] 287.1265
##
##
## $horsebean
## $horsebean$left
## [1] 184.1405
##
## $horsebean$center
## [1] 160.2
##
## $horsebean$right
## [1] 136.2595
##
##
## $linseed
## $linseed$left
## [1] 248.3051
##
## $linseed$center
## [1] 218.75
##
## $linseed$right
## [1] 189.1949
##
##
## $meatmeal
## $meatmeal$left
## [1] 315.2629
##
```

```
## $meatmeal$center
## [1] 276.9091
##
## $meatmeal$right
## [1] 238.5553
##
##
## $soybean
## $soybean$left
## [1] 274.7831
##
## $soybean$center
## [1] 246.4286
##
## $soybean$right
## [1] 218.074
##
##
## $sunflower
## $sunflower$left
## [1] 356.5485
##
## $sunflower$center
## [1] 328.9167
##
## $sunflower$right
## [1] 301.2849
```

```
  # Ricordarsi!
  detach( chickwts )
```

Molto utili, infine, sono i comandi per realizzare tabelle di contingenza di singole o gruppi di variabili (tipicamente categoriche).

```
  attach( chickwts )

  # Tabella delle occorrenze della variabile factor chickwts$feed
  table( feed )
```

```
## feed
##    casein horsebean   linseed  meatmeal   soybean sunflower
##        12        10        12        11        14        12
```

```
  # Ricordarsi!
  detach( chickwts )
```

2.7 Operazioni matematiche

Le operazioni matematiche tra variabili numeriche non riservano sorprese.

Per quanto riguarda strutture come array e matrici, bisogna tenere a mente che `R` cerca di vettorizzarle e di procedere con in maniera *element-wise*, cioè elemento per elemento, ove possibile.

```
  x = seq( 0, 1, length.out = 11 )
```

```
  x * 2
##  [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
  x + x
##  [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
  x^2
##  [1] 0.00 0.01 0.04 0.09 0.16 0.25 0.36 0.49 0.64 0.81 1.00
  log( x )
##  [1]       -Inf -2.3025851 -1.6094379 -1.2039728 -0.9162907 -0.6931472
##  [7] -0.5108256 -0.3566749 -0.2231436 -0.1053605  0.0000000

  # Attenti al re-cycling!
  # Quando le dimensioni degli oggetti ammettono multipli comuni, R cicla le operazioni
  # sulla struttura dati più piccola (vettore, matrice, array, etc)
  # Non sempre questo è desiderabile. A volte questi sono errori di programmazione, ma R
  # esegue le istruzioni fino in fondo senza dare warning.
  y = 1
  x + y
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0

  X = matrix( 1 : 12, nrow = 3, ncol = 4, byrow = TRUE )
  X
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
  y = 1 : 4

  X + y
##      [,1] [,2] [,3] [,4]
## [1,]    2    6    6    6
## [2,]    7    7   11   11
## [3,]   12   12   12   16
```

Le operazioni di algebera lineare si svolgono in questo modo:

```
A = matrix( 1 : 4, nrow = 2, byrow = TRUE )

b = c( 1, 0 )

# Prodotto matrice - vettore
A %*% b
##      [,1]
## [1,]    1
## [2,]    3

# Soluzione del sistema lineare A x = b
solve( A, b )
## [1] -2.0  1.5

# Trasposta di una matrice
t( A )
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
# Autovalori ed autovettori di A
eigen( A )
## $values
## [1]  5.3722813 -0.3722813
##
## $vectors
##            [,1]       [,2]
## [1,] -0.4159736 -0.8245648
## [2,] -0.9093767  0.5657675

# Somma di matrici
B = diag( 2 )
A + B
##      [,1] [,2]
## [1,]    2    2
## [2,]    3    5
```

2.8 Strutture di controllo

Essendo un linguaggio di programmazione, `R` possiede delle strutture di controllo quali `if-then-else` oppure cicli (`while` o `for`).

```
  # If statements ed operazioni logiche
  a = 3

  if( a == 1 )
  {
    print( 'a = 1' )
  } else if( a == 2 ){
    print( 'a = 2' )
  } else {
    print( ' a != 1 & a != 2 ' )
  }
## [1] " a != 1 & a != 2 "

  a == 1
## [1] FALSE
  a != 1 & a == 2
## [1] FALSE
  a == 1 | a == 3
## [1] TRUE

  # Ciclo for
  for( i in 1 : 10 )
  {
    a = a + 1
  }
  a
## [1] 13

  for( j in seq_along( c(10,9,8,7,6) ) )
  {
```

```
    print( j )
  }
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5

  for( lettera in c('c','i','a','o','!') )
  {
    print( lettera )
  }
## [1] "c"
## [1] "i"
## [1] "a"
## [1] "o"
## [1] "!"

  # Ciclo while
  it = 0
  sum = 0
  while( sum < 49 )
  {
    sum = sum + 2 * it + 1
    it = it + 1
  }
  it; sum
## [1] 7
## [1] 49
```

3 Visualizzazione

In R potete trovare moltissime funzioni per visualizzare i vostri dati, specialmente nei molti pacchetti accessori disponibili per gli utenti. R lavora con *device* grafici, che vengono aperti e chiusi per ospitare i grafici. A seconda del sistema operativo possono esserci comandi diversi per aprire un device grafico:

```
  # Per Windows o Linux
  x11()

  # Per Mac OS X
  quartz()

  # Per ogni piattaforma
  dev.new()

  # Per chiudere tutti i devices aperti (potreste averne molti!)
  graphics.off()
```

3.1 plot

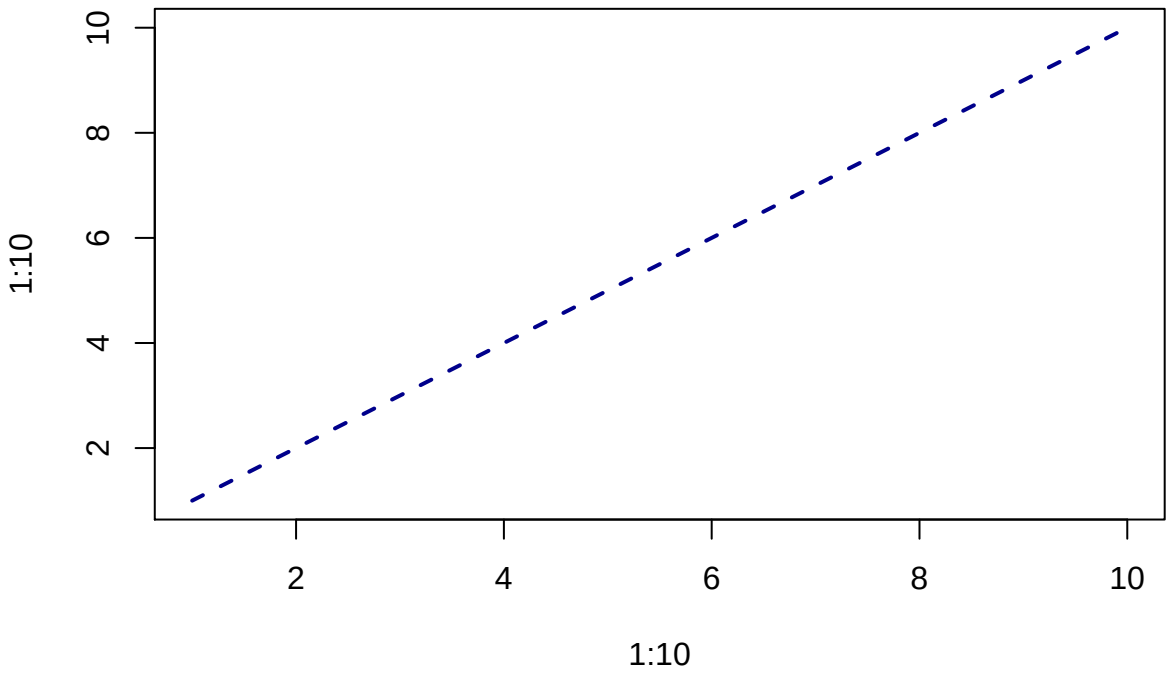
La funzione principale per creare grafici è `plot`, che può avere una chiamata anche molto complicata, ma nel caso più semplice accetta un vettore di coordinate x e un vettore di coordinate y dei punti da rappresentare:

```
# Utilizzo base
dev.new()
plot( 1 : 10, 1 : 10 )

# Possiamo modificare i parametri grafici del plot
help(par)
plot( 1 : 10, 1 : 10, col = c('blue','red'), pch = 16, cex = c(2,1.5,1),
      xlab = 'x', ylab = 'y', main = 'Un plot' )
```

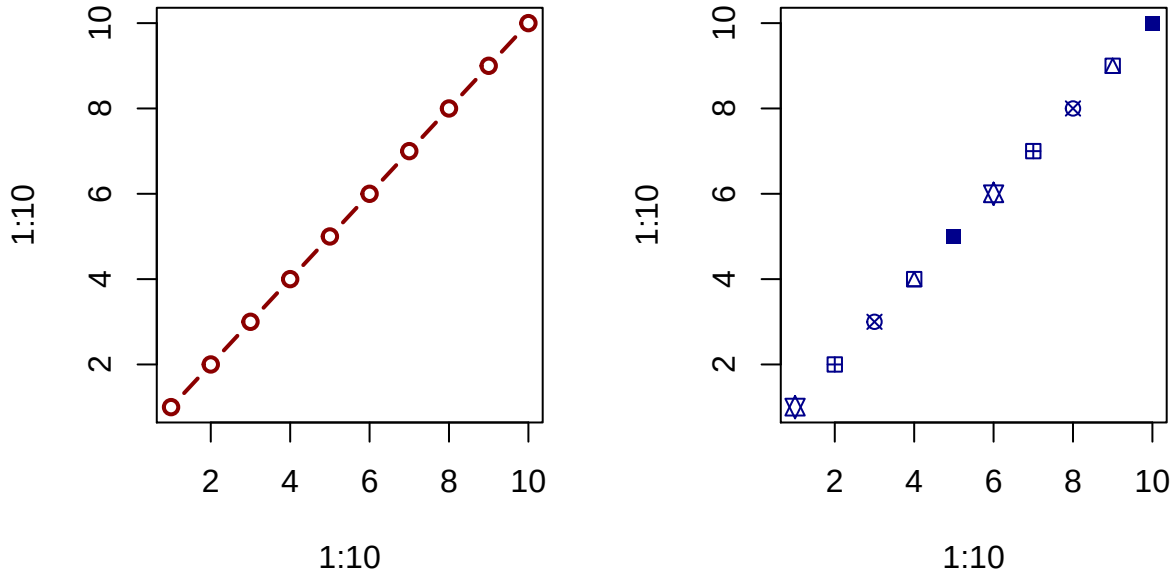
Di base `plot` rappresenta i dati con uno scatterplot. Se volete rappresentare i dati tramite linee dovete usare l'opzione `type = 'l'`; potete controllare il tipo di linea con il parametro `lty`, con valori `1` (linea continua), `2` (linea spezzata), `3` (linea a puntini), etc.

```
# Grafico in forma di linea, e uso di altri parametri
plot( 1 : 10, 1 : 10, type = 'l', lty = 2, lwd = 2, col = 'darkblue')
```



I device grafici possono anche essere suddivisi in finestre con il comando `par`; ad esempio, il comando `par( mfrow = c( 1, 2 ) )` produrrà un device grafico con una riga di due finestre, che verranno riempite con due successive chiamate a funzioni grafiche (e.g. `plot`):

```
par( mfrow = c( 1, 2 ) )
plot( 1 : 10, 1 : 10, type = 'b', lwd = 2, col = 'darkred' )
plot( 1 : 10, 1 : 10, pch = 11 : 15, col = 'darkblue' )
```

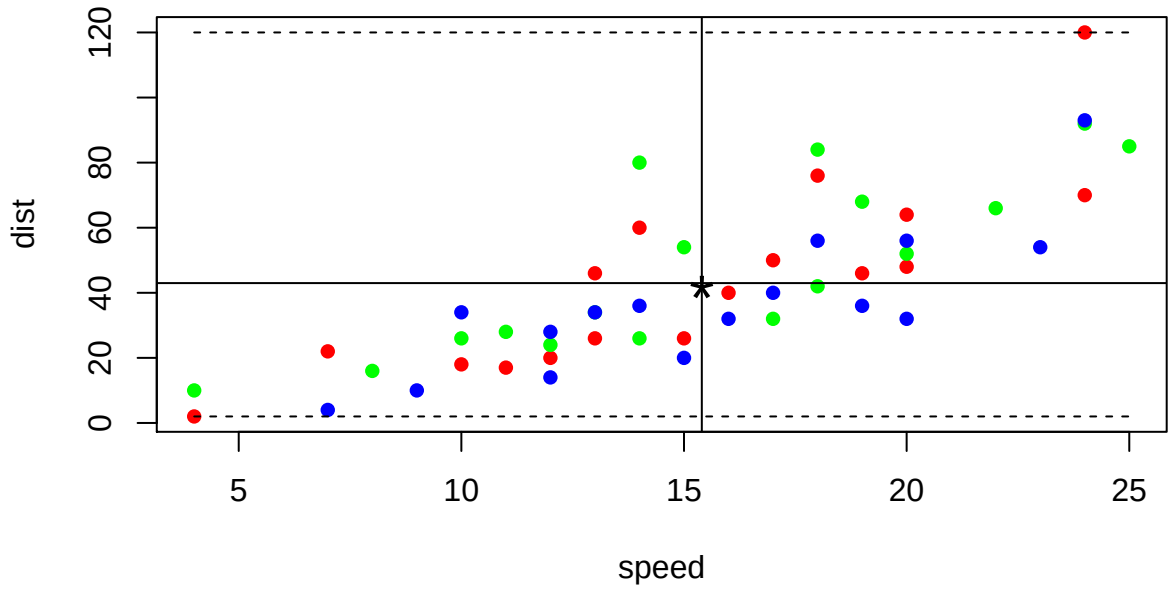


Per **aggiungere** elementi grafici ad una finestra su cui è già stato rappresentato qualcosa, si possono usare i comandi `lines`, `abline` e `points` per rappresentare, linee e punti:

```
# Prendiamo i dati dal dataset cars di R
help(cars)

data(cars)

plot( cars, pch = 16, col = c('red','green','blue') )
points( mean( cars$speed ), mean( cars$dist), col = 'black', cex = 2,
        pch = '*' )
lines( range( cars$speed ), rep( min( cars$dist ), 2 ), lty = 2, col = 'black' )
lines( range( cars$speed ), rep( max( cars$dist ), 2 ), lty = 2, col = 'black' )
abline( v = mean( cars$speed ) )
abline( h = mean( cars$dist ) )
```



3.2 Istogramma

Per capire come rappresentare dati secondo un istogramma, importiamo un dataset distribuito con `R`:

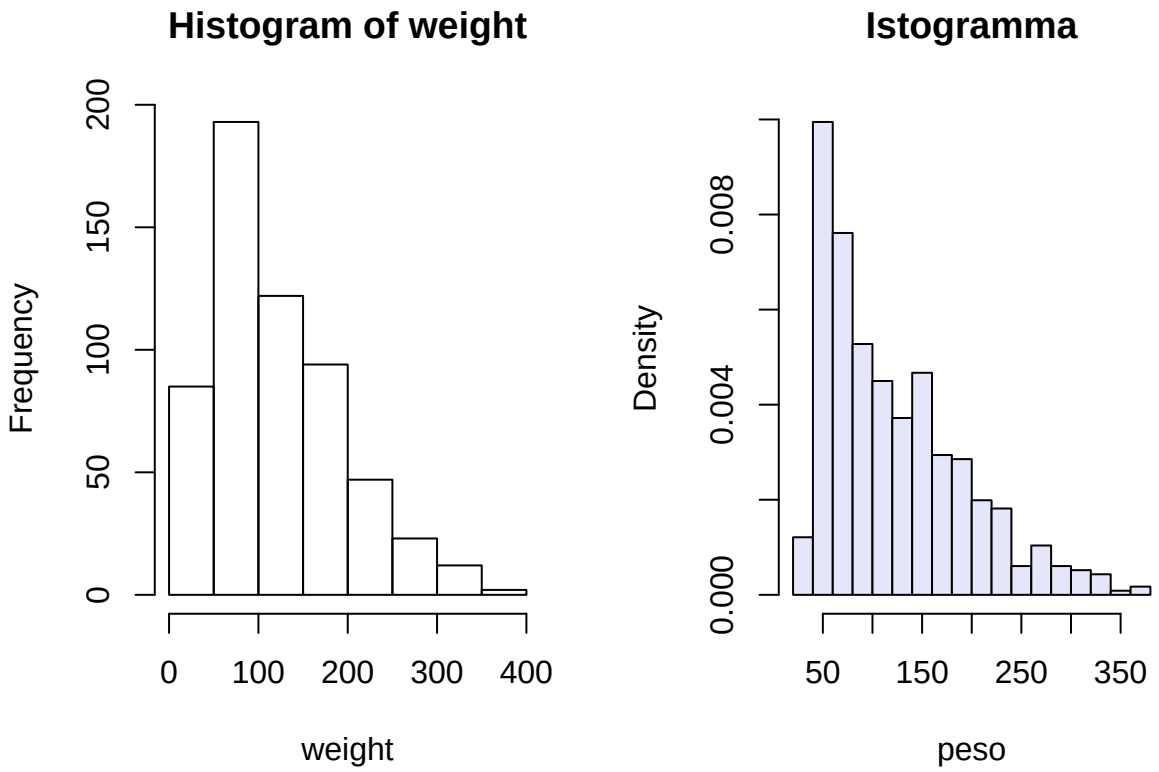
```
  help(ChickWeight)

  data(ChickWeight)

  attach(ChickWeight)

  summary( weight )
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    35.0    63.0   103.0   121.8   163.8   373.0

  par( mfrow = c( 1, 2 ) )
  # Istogramma con parametri di default
  hist( weight )
  # Istogramma con scelta di parametri (DOVETE usare la densità, probability = TRUE)
  hist( weight, breaks = sqrt( length( weight ) ), probability = TRUE,
        col = 'lavender', main = 'Istogramma', xlab = 'peso' )
```



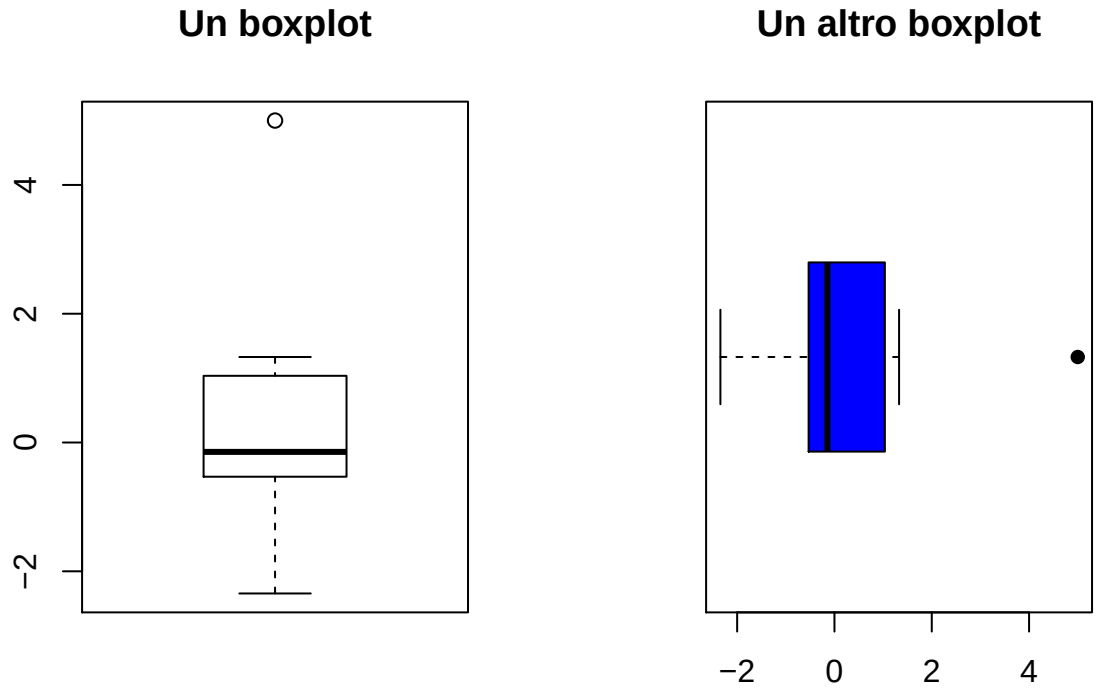

```
  # Ricordatevi!
  detach(ChickWeight)
```

3.3 Boxplot

Il boxplot si ottiene con il comando `boxplot`:

```r
set.seed(0618033)
# Creiamo dei dati fittizi
D = rnorm( 10, 0, 1 )
# Aggiungiamo un outlier
D = c( D, 5 )

par( mfrow = c(1, 2 ) )
# Un boxplot
boxplot( D, main = 'Un boxplot' )
# Ecco un altro possibile boxplot
boxplot( D, col = 'blue', pch = 16, horizontal = TRUE,
         main = 'Un altro boxplot' )
```



3.4 Pie chart e diagramma a barre

La pie chart si ottiene con il comando `pie`, mentre il diagramma a barre si ottiene con il comando `barplot`:

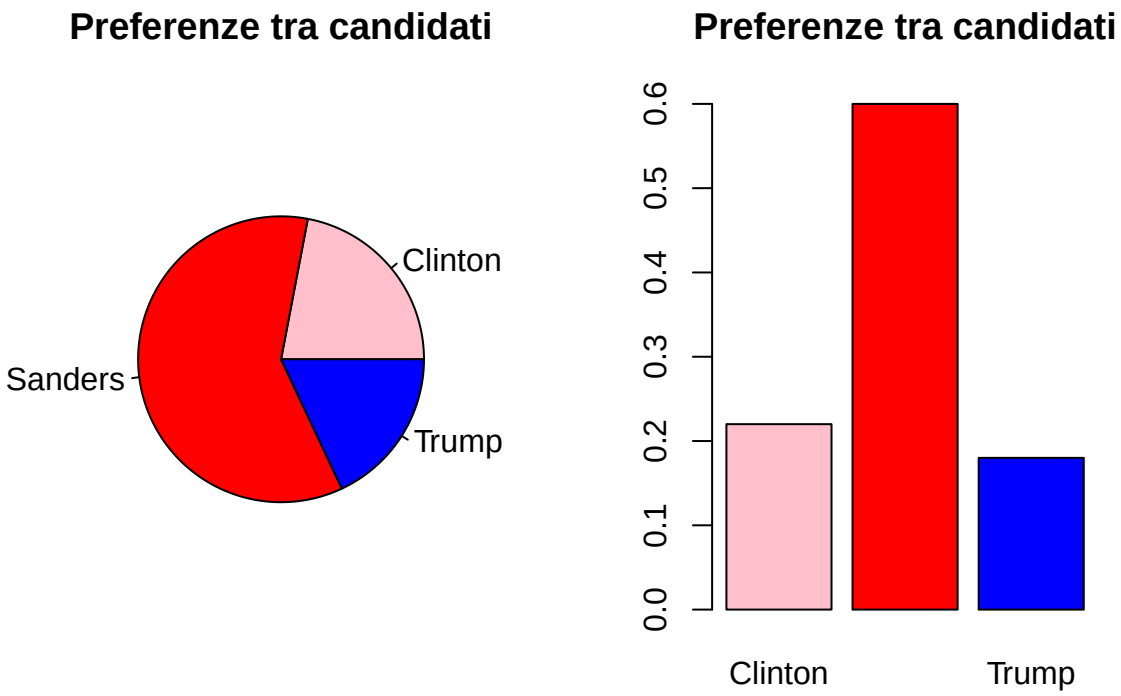
```r
# Creiamo un insieme di dati fittizio
set.seed(0618033)
v = sample( c( 'Trump', 'Clinton', 'Sanders' ),
            size = 50,
            replace = TRUE,
            prob = c( 0.2, 0.25, 0.55 ) )

# Trasformiamo la variabile in factor, poiché variabile categorica
# (anche se non è necessario perché i comandi vadano a buon fine)
v_fac = as.factor( v )

# Calcoliamo la tabella di contingenza
tab = table( v_fac )
tab
```

```
## v_fac
## Clinton Sanders   Trump
##      11      30       9
```

```
  par( mfrow = c( 1, 2 ) )
  pie( tab, col = c('pink', 'red', 'blue' ), main = 'Preferenze tra candidati' )
  barplot( tab / length( v_fac ), col = c('pink', 'red', 'blue' ),
           main = 'Preferenze tra candidati' )
```



3.5 Salvare i grafici

Il contenuto delle finestre grafiche può anche essere salvato in immagini di vari formati, che possono essere incluse in report, articoli, etc.

Il modo più tipico per fare si ciò consiste nel:

1) Inizializzare il processo di scrittura del contenuto del device grafico su file. Questo si fa invocando un opportuno comando **prima** di aver eseguito il comandi di plot che scrivono il contenuto grafico (`plot`, `lines`, `points`, etc.). Ci sono varie possibilità: `pdf` produce un PDF, `jpeg` un JPEG, `png` un PNG, etc. Per una lista completa, `help(Devices)`.[3]

2) se richiesto, personalizzare la finestra grafica (con `par`), ed eseguire i comandi grafici desiderati;

3) finalizzare il processo di scrittura del grafico su file, col comando `dev.off()`.

Vediamo tutto questo in un esempio:

```
  # Importiamo un altro dataset
  help(iris)
```

[3] Un consiglio: se i grafici non sono troppo ricchi, prediligete i formati vettoriali (e.g. PDF), perché sono lossless e stabili rispetto a possibili zoom. Altrimenti, per grafici troppo ricchi, per cui ad esempio i PDF diventano troppo pesanti, usate un formato lossy non vettoriale (e.g. `jpeg` o `png`), ma cercate di usare una risoluzione abbastanza alta (un valore piuttosto alto è `res = 300`, cioè `300 ppi`) e un *aspect ratio* (rapporto di ampiezza e altezza) simile a quello con cui volete poi mostrare l'immagine nel documento.

```r
data(iris)

attach( iris )

ans = require('scales', quietly = TRUE )

# Impostiamo la paletta per i grafici seguenti
if( ! ans )
{
  palette( c( 'blue', 'red', 'forestgreen' ) )
} else {
  palette( hue_pal()(3) )
}

# Inizializziamo un file PDF
#
#  NON USATE SPAZI E CARATTERI SPECIALI PER I NOMI
#
pdf( './il_mio_primo_grafico.pdf', width = 12, height = 6 )

par( mfrow = c( 1, 2 ) )
plot( Sepal.Length, Sepal.Width, col = Species, pch = 16,
      xlab = 'Sepal Length', ylab = 'Sepal Width', main = 'Plot #1' )
plot( Petal.Length, Petal.Width, col = Species, pch = 16,
      xlab = 'Petal Length', ylab = 'Petal Width', main = 'Plot #2' )
# finalizziamo il grafico
dev.off()

# Inizializziamo un file jpeg
jpeg( './il_mio_secondo_grafico.jpg', units = 'in',
      width = 12, height = 6, res = 300 )

par( mfrow = c( 1, 2 ) )
plot( Sepal.Length, Sepal.Width, col = Species, pch = 16,
      xlab = 'Sepal Length', ylab = 'Sepal Width', main = 'Plot #1' )
# Aggiungiamo una legenda (complicato)
legend( 'topright', legend = unique( as.character( iris$Species ) ),
        pch = 16, col = palette(), cex = 1, text.font = 1 )

plot( Petal.Length, Petal.Width, col = Species, pch = 16,
      xlab = 'Petal Length', ylab = 'Petal Width', main = 'Plot #2' )
# Aggiungiamo una legenda (complicato)
legend( 'bottomright', legend = unique( as.character( iris$Species ) ),
        pch = 16, col = palette(), cex = 1, text.font = 1 )
# finalizziamo il grafico
dev.off()

detach(iris)
```

4 Approfondimenti e referenze

1) Il sito web del CRAN https://cran.r-project.org
2) Il Blog di Hadley Wickham, uno dei guru dello sviluppo R: http://hadley.nz
3) Il sito web di riferimento di ggplot2, una libreria grafica per R completa e versatile http://ggplot2.org
4) Le sezioni su R dei siti web di Question&Answers StackOverflow (http://stackoverflow.com) e CrossValidated (http://stats.stackexchange.com)