stRutture di dati

Ottavia M. Epifania, Ph.D

Lezione di Dottorato @Università Cattolica del Sacro Cuore (MI)

8-9 giugno 2023

Table of contents

Table of Contents

Vengono creati **c**oncatenando diverse variabili insieme

Si usa la funzione `c()`

Tutte le variaili all'interno della funzione `c()` vanno separate da una virgola

Diversi tipi di variabili $\rightarrow$ diversi tipi di vettori:

- `int`: vettori numerici (numeri interi)
- `num`: vettori numerici (numeri continui)
- `logi`: vettori logici
- `chr`: vettori character
- `factor`: vettori factor con diversi livelli

`int` & `num`

`int`: numeri interi: -3, -2, -1, 0, 1, 2, 3

```r
mesi = c(5, 6, 8, 10, 12, 16)
```

```
[1]  5  6  8 10 12 16
```

`num`: tutti i valori numerici tra $-\infty$ e $+\infty$: 1.0840991, 0.8431089, 0.494389, -0.7730161, 2.9038161, 0.9088839

```r
peso = seq(3, 11, by = 1.5)
```

```
[1]  3.0  4.5  6.0  7.5  9.0 10.5
```

logi

Valori logici possono essere veri TRUE (T) o falsi FALSE (F):

```
v_logi = c(TRUE, TRUE, FALSE, FALSE, TRUE)
```

```
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

Si usano per testare delle condizioni:

```
mesi > 12
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

chr & factor

chr: characters: a, b, c, D, E, F

```
v_chr = c(letters[1:3], LETTERS[4:6])
```

```
[1] "a" "b" "c" "D" "E" "F"
```

factor: Usa numeri o carattarri per identificare i livelli della variabile:

```
ses = factor(c(rep(c("low", "medium", "high"), each = 2)))
```

```
[1] low    low    medium medium high   high
Levels: high low medium
```

Si può cambaire l'ordine dei livelli:

```
ses1 = factor(ses, levels = c("medium", "high", "low"))
```

```
[1] low    low    medium medium high   high
Levels: medium high low
```

Creare i vettori

Concatenare le variabili con `c()`: `vec = c(1, 2, 3, 4, 5)`

Utilizzando le sequenze:

```
-5:5 # vector of 11 numbers from -5 to 5
```

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

```
seq(-2.5, 2.5, by = 0.5) # sequence in steps of 0.5
```

```
 [1] -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0  1.5  2.0  2.5
```

Ripetendo gli elementi:

```
rep(1:3, 4)
```

```
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

Creare i vettori II

```r
rep(c("condA", "condB"), each = 3)
```

```
[1] "condA" "condA" "condA" "condB" "condB" "condB"
```

```r
rep(c("on", "off"), c(3, 2))
```

```
[1] "on"  "on"  "on"  "off" "off"
```

```r
paste0("item", 1:4)
```

```
[1] "item1" "item2" "item3" "item4"
```

Non mischiate i vettori! a meno che non lo vogliate davvero

$\texttt{int} + \texttt{num} \rightarrow \texttt{num}$

$\texttt{int/num} + \texttt{logi} \rightarrow \texttt{int/num}$

$\texttt{int/num} + \texttt{factor} \rightarrow \texttt{int/num}$

$\texttt{int/num} + \texttt{chr} \rightarrow \texttt{chr}$

$\texttt{chr} + \texttt{logi} \rightarrow \texttt{chr}$

Vettori e operazioni

I vettori possono essere sommati/divisi/moltiplicati tra di loro o anche per un numero singolo

```
a = c(1:8) # vettore di lunghezza 8
a
```

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

```
b = c(4:1) # vettore di lunghezza 4
b
```

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

```
a - b # il vettore b è "riciclato" sul vettore a
```

```
[1] -3 -1  1  3  1  3  5  7
```

Se i vettori non hanno la stessa lunghezza (o uno non è un multiplo dell'altro) ottenete un warning

Vettori e operazioni II

Applicando una funzione a un vettore → viene applicata a **tutti** gli elementi del vettore

```
sqrt(a)
```

```
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.6
```

La stessa operazione si può applicare a ogni singolo elemento del vettore

```
(a - mean(a))^2 # squared deviation
```

```
[1] 12.25  6.25  2.25  0.25  0.25  2.25  6.25 12.25
```

Indicizzare i vettori

Come si va a “raggiungere” un particolare elemento all’interno del vettore?

```
nomi = c("Pasquale", "Egidio", "Debora", "Luca", "Andrea")
```

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

Indicizzare i vettori

Come si va a “raggiungere” un particolare elemento all’interno del vettore?

```
nomi = c("Pasquale", "Egidio", "Debora", "Luca", "Andrea")
```

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

```
nome_vettore[indice]
```

Indicizzare i vettori II

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

Indicizzare i vettori II

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

`nomi[1]` $\rightarrow$

Indicizzare i vettori II

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

`nomi[1]` $\rightarrow$ Pasquale

`nomi[3]` $\rightarrow$

Indicizzare i vettori II

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

`nomi[1]` $\rightarrow$ Pasquale

`nomi[3]` $\rightarrow$ Debora

`nomi[seq(2, 5, by = 2)]` $\rightarrow$

Indicizzare i vettori II

Pasquale	Egidio	Debora	Luca	Andrea
1	2	3	4	5

`nomi[1]` → Pasquale

`nomi[3]` → Debora

`nomi[seq(2, 5, by = 2)]` → Egidio, Luca

Indicizzare i vettori: Esempi

```
peso
```

```
[1]  3.0  4.5  6.0  7.5  9.0 10.5
```

```
peso[2]        # secondo elemento del vettore peso
```

```
[1] 4.5
```

```
(peso[6] = 15.2) # sostituisce il sesto elemento del v. peso
```

```
[1] 15.2
```

```
peso[seq(1, 6, by = 2)] # elementi 1, 3, 5
```

```
[1] 3 6 9
```

```
peso[2:6]      # dal 2 al 6 elemento di peso
```

```
[1]  4.5  6.0  7.5  9.0 15.2
```

```
peso[-2]      # vettore peso senza il secondo elemento
```

Indicizzare i vettori usando la logica

```
peso[peso > 7] # valori in peso maggiori di 7
```

```
[1]  7.5  9.0 15.2
```

```
peso[peso >= 4.5 & peso < 8] # valori tra 4.5 e 8
```

```
[1] 4.5 6.0 7.5
```

Table of Contents

Un vettore che ci ha creduto abbastanza

Quel che basta per vincere una seconda dimensione

`matrix(data, nrow, ncol, byrow = TRUE)`

Crea una matrice 3×4 e la assegna all'oggetto `A`:

```
A = matrix(1:12, nrow=3, ncol = 4, byrow = FALSE)
A
```

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

WARNING: i dati all'interno della matrice devono essere tutti dello stesso tipo

Etichette

```
rownames(A) = c(paste("riga", 1:nrow(A), sep = "_"))

colnames(A) = c(paste("colonna", 1:ncol(A), sep = "_"))

A
```

```
        colonna_1 colonna_2 colonna_3 colonna_4
riga_1          1         4         7        10
riga_2          2         5         8        11
riga_3          3         6         9        12
```

Trasposta della matrice:

```
A
```

```
        colonna_1 colonna_2 colonna_3 colonna_4
riga_1          1         4         7        10
riga_2          2         5         8        11
riga_3          3         6         9        12
```

```
t(A)
```

```
          riga_1 riga_2 riga_3
colonna_1      1      2      3
colonna_2      4      5      6
colonna_3      7      8      9
colonna_4     10     11     12
```

Creare le matrici (ancora)

Le matrici si possono anche creare concatenando vettori colonna:

```r
cbind(a1 = 1:4, a2 = 5:8, a3 = 9:12)
```

```
     a1 a2 a3
[1,]  1  5  9
[2,]  2  6 10
[3,]  3  7 11
[4,]  4  8 12
```

o vettori riga:

```r
rbind(a1 = 1:4, a2 = 5:8, a3 = 9:12)
```

```
   [,1] [,2] [,3] [,4]
a1    1    2    3    4
a2    5    6    7    8
a3    9   10   11   12
```

Indicizzare le matrici

Abbiamo due dimensioni:

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

```
my_matrix[righe, colonne]
```

Indicizzare le matrici: In pratica

```
A
```

```
      colonna_1 colonna_2 colonna_3 colonna_4
riga_1         1         4         7        10
riga_2         2         5         8        11
riga_3         3         6         9        12
```

`A[1, ]` $\rightarrow$

`A[2, ]` $\rightarrow$

`A[2, 3]` $\rightarrow$

Indicizzare le matrici: In pratica

```
A
```

```
       colonna_1 colonna_2 colonna_3 colonna_4
riga_1         1         4         7        10
riga_2         2         5         8        11
riga_3         3         6         9        12
```

`A[1, ]` $\rightarrow$ 1, 4, 7, 10

`A[2, ]` $\rightarrow$

`A[2, 3]` $\rightarrow$

Indicizzare le matrici: In pratica

```
A
```

```
        colonna_1 colonna_2 colonna_3 colonna_4
riga_1          1         4         7        10
riga_2          2         5         8        11
riga_3          3         6         9        12
```

`A[1, ]` $\rightarrow$ 1, 4, 7, 10

`A[2, ]` $\rightarrow$ 2, 5, 8, 11

`A[2, 3]` $\rightarrow$

Indicizzare le matrici: In pratica

```
A
```

```
        colonna_1 colonna_2 colonna_3 colonna_4
riga_1          1         4         7        10
riga_2          2         5         8        11
riga_3          3         6         9        12
```

`A[1, ]` $\rightarrow$ 1, 4, 7, 10

`A[2, ]` $\rightarrow$ 2, 5, 8, 11

`A[2, 3]` $\rightarrow$ 8

Table of Contents

Una matrice che ci ha creduto davvero

Davvero troppo

```
array(data, c(nrow, ncol, ntab))
```

Avendo 3 argomenti oltre i dati `nrow, ncol, ntab`, la loro indicizzazione prevede l'utilizzo di due virgole per accedere ai singoli argomenti:
`nome_array[righe, colonne, tab]`

Un array

```
my_array = array(1:20, c(2, 5, 3)) # 2 x 5 x 3 array
my_array
```

```
, , 1

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

, , 2

     [,1] [,2] [,3] [,4] [,5]
[1,]   11   13   15   17   19
[2,]   12   14   16   18   20

, , 3

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

Indicizzare l’array

```
my_array[1, , ]
```

```
my_array[, 2, ]
```

```
my_array[, , 3]
```

Indicizzare l'array

```
my_array[1, , ]
```

```
     [,1] [,2] [,3]
[1,]    1   11    1
[2,]    3   13    3
[3,]    5   15    5
[4,]    7   17    7
[5,]    9   19    9
```

```
my_array[, 2, ]
```

```
my_array[, , 3]
```

Indicizzare l’array

```
my_array[1, , ]
```

```
     [,1] [,2] [,3]
[1,]    1   11    1
[2,]    3   13    3
[3,]    5   15    5
[4,]    7   17    7
[5,]    9   19    9
```

```
my_array[, 2, ]
```

```
     [,1] [,2] [,3]
[1,]    3   13    3
[2,]    4   14    4
```

```
my_array[, , 3]
```

Indicizzare l'array

```
my_array[1, , ]
```

```
     [,1] [,2] [,3]
[1,]    1   11    1
[2,]    3   13    3
[3,]    5   15    5
[4,]    7   17    7
[5,]    9   19    9
```

```
my_array[, 2, ]
```

```
     [,1] [,2] [,3]
[1,]    3   13    3
[2,]    4   14    4
```

```
my_array[, , 3]
```

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

Table of Contents

Un array con più senso

Sono dei contenitori per diversi tipi di oggetti (e.g., vettori, data frames, altre liste, matrici, array ecc.)

Ai loro elementi possono essere assegnati dei nomi:

```
my_list = list(w = peso, m = mesi, s = ses1, a = A)
names(my_list)
```

```
[1] "w" "m" "s" "a"
```

```
str(my_list)
```

```
List of 4
 $ w: num [1:6] 3 4.5 6 7.5 9 15.2
 $ m: num [1:6] 5 6 8 10 12 16
 $ s: Factor w/ 3 levels "medium","high",..: 3 3 1 1 2 2
 $ a: int [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "riga_1" "riga_2" "riga_3"
  .. ..$ : chr [1:4] "colonna_1" "colonna_2" "colonna_3" "colonna_4
```

Indicizzare le liste

Gli elementi della lista possono essere indicizzati con $ (se la lista ha dei nomi):

```
my_list$m # vettore dei mesi
```

```
[1]  5  6  8 10 12 16
```

oppure con `[[]]`:

Nome dell'elemento

```
my_list[["m"]]
```

```
[1]  5  6  8 10 12 16
```

Posizione dell'elemento:

```
my_list[[2]]
```

```
[1]  5  6  8 10 12 16
```

Table of Contents

Una lista più ordinata

I data frames sono delle liste di vettori di uguale lunghezza

I diversi vettori possono contenere informazioni di diverse natura

I data frame più comuni sono i data frame in versione wide (i.e., *soggetti* $\times$ *variabili*) $\rightarrow$ `nrow(data)` $=$ numero di soggetti:

```
id = paste0("sbj", 1:6)
babies = data.frame(id, mesi, peso)
```

```
babies
```

```
    id mesi peso
1 sbj1    5  3.0
2 sbj2    6  4.5
3 sbj3    8  6.0
4 sbj4   10  7.5
5 sbj5   12  9.0
6 sbj6   16 15.2
```

Indicizzare i data frame

Vale tutto quello visto per le matrici:

Prima riga del data frame babies
`babies[1, ]`

Prima colonna del data frame babies
`babies[, 1]`

In più:

```
babies$mesi # colonna mesi di babies

babies$mesi[2] # secondo elemento del vettore colonna

babies[, "id"] # column id

babies[2, ] # second row of babies (obs on baby 2)
```

Logic applies:

```
babies[babies$peso > 7, ] # filtra per tutte le righe con pes
```

Working with data frames II

```
dim(babies) # data frame con 6 righe e 3 colonne
```

```
[1] 6 3
```

```
names(babies) # = colnames(babies)
```

```
[1] "id"   "mesi" "peso"
```

```
View(babies) # open data viewer
```

Questi comandi possono essere usati anche su altri oggetti `R`

Working with data frames III

```
str(babies) # show details on babies
```

```
'data.frame':   6 obs. of  3 variables:
 $ id  : chr  "sbj1" "sbj2" "sbj3" "sbj4" ...
 $ mesi: num  5 6 8 10 12 16
 $ peso: num  3 4.5 6 7.5 9 15.2
```

```
summary(babies) # descriptive statistics
```

```
      id                  mesi            peso       
 Length:6           Min.   : 5.0   Min.   : 3.000  
 Class :character   1st Qu.: 6.5   1st Qu.: 4.875  
 Mode  :character   Median : 9.0   Median : 6.750  
                    Mean   : 9.5   Mean   : 7.533  
                    3rd Qu.:11.5   3rd Qu.: 8.625  
                    Max.   :16.0   Max.   :15.200  
```

Sorting

order():

```
babies[order(babies$peso), ] # sort by increasing peso
```

```
    id mesi peso
1 sbj1    5  3.0
2 sbj2    6  4.5
3 sbj3    8  6.0
4 sbj4   10  7.5
5 sbj5   12  9.0
6 sbj6   16 15.2
```

```
babies[order(babies$peso,    # sort by decreasing peso
             decreasing = T), ]
```

Multiple arguments in order:

```
babies[order(babies$peso, babies$mesi, decreasing = TRUE), ]
```

Aggregating

Aggregate a response variable according to grouping variable(s) (e.g., averaging per experimental conditions):

```r
# Single response variable, single grouping variable
aggregate(y ~ x, data = data, FUN, ...)

# Multiple response variables, multiple grouping variables
aggregate(cbind(y1, y2) ~ x1 + x2, data = data, FUN, ...)
```

Aggregating: Example

```
ToothGrowth # Vitamin C and tooth growth (Guinea Pigs)
```

```
    len supp dose
1   4.2   VC  0.5
2  11.5   VC  0.5
3   7.3   VC  0.5
....
```

```
aggregate(len ~ supp + dose, data = ToothGrowth, mean)
```

```
  supp dose   len
1   OJ  0.5 13.23
2   VC  0.5  7.98
3   OJ  1.0 22.70
4   VC  1.0 16.77
5   OJ  2.0 26.06
6   VC  2.0 26.14
```

Reshaping: Long to wide

Data can be organized in wide format (i.e., one line for each statistical unit) or in long format (i.e., one line for each observation).

```
Indometh # Long format
```

```
   Subject time conc
1        1 0.25 1.50
2        1 0.50 0.94
3        1 0.75 0.78
4        1 1.00 0.48
5        1 1.25 0.37
6        1 2.00 0.19
....
```

Long to wide

```r
# From long to wide
df.w <- reshape(Indometh, v.names = "conc", timevar = "time",
    idvar = "Subject", direction = "wide")
```

```
   Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 con
1        1      1.50     0.94      0.78   0.48      0.37   0.19   0
12       2      2.03     1.63      0.71   0.70      0.64   0.36   0
23       3      2.72     1.49      1.16   0.80      0.80   0.39   0
34       4      1.85     1.39      1.02   0.89      0.59   0.40   0
45       5      2.05     1.04      0.81   0.39      0.30   0.23   0
56       6      2.31     1.44      1.03   0.84      0.64   0.42   0
   conc.5 conc.6 conc.8
....
```

Reshaping: Wide to long

```
# From wide to long
df.l <- reshape(df.w, varying = list(2:12), v.names = "conc",
    idvar = "Subject", direction = "long", times = c(0.25, 0.5,
        0.75, 1, 1.25, 2, 3, 4, 5, 6, 8))
```

```
       Subject time conc
1.0.25       1 0.25 1.50
2.0.25       2 0.25 2.03
3.0.25       3 0.25 2.72
....
```

```
df.l[order(df.l$Subject), ] # reorder by subject
```

```
       Subject time conc
1.0.25       1 0.25 1.50
1.0.5        1 0.50 0.94
1.0.75       1 0.75 0.78
....
```

Table of Contents

Reading tabular txt files:

ASCII text files in tabular or spread sheet form (one line per observation, one column per variable) are read using `read.table()`

```
data = read.table("C:/RcouRse/file.txt", header = TRUE)
```

`data` is a data frame where the original numerical variables are converted in numeric vectors and character variables are converted in factors (not always).

Arguments:

- `header`: variable names in the first line? `TRUE/FALSE`
- `sep`: which separator between the columns (e.g., comma, `\t`)
- `dec`: `1.2` or `1,2`?

Reading other files

```
data = read.csv("C:/RcouRse/file.csv",
                header = TRUE, sep = ";",
                dec = ",")
```

From SPSS (file `.sav`):

```
install.packages("foreign")
library(foreign)
data = read.spss("data.sav", to.data.frame = TRUE)
```

Combine data frames

If they have the same number of columns/rows

```
all_data = rbind(data, data1, data2) # same columns
all_data = cbind(data, data1, data2) # same rows
```

If they have different rows/columns but they share at least one characteristic (e.g., `ID`):

```
all_data = merge(data1, data2,
                 by = "ID")
```

If there are different IDs in the two datasets $\rightarrow$ added in new rows

`all_data` contains all columns in `data1` and `data2`. The columns of the IDs in `data1` but not in `data2` (and vice versa) will be filled with `NA`s accordingly

Export data

Writing text (or csv) file:

```
write.table(data, # what you want to write
            file = "mydata.txt", # its name + extension
            header = TRUE, # first row with col names?
            sep = "\t",  # column separator
            ....) # other arguments
```

R environment (again):

```
save(dat, file = "exp1_data.rda") # save something specific
save(file = "the_earth.rda")      # save the environment
load("the_earth.rda")             # load it back
```

Table of Contents

Be ready to make mistakes (a lot of mistakes)

Coding is ~~hard~~ art

Eyes on the prize, but take your time (and the necessary steps) to get there

Remember: You're not alone $\rightarrow$ `stackoverflow` (or Google in general) is your best friend

ifelse()

Conditional execution:

Easy: `ifelse(test, if true, if false)`

```
ifelse(peso > 7, "big boy", "small boy")
```

```
[1] "small boy" "small boy" "small boy" "big boy"   "big boy"
```

Pros
- Super easy to use
- Can embed multiple `ifelse()` cycles

Cons
- It works fine until you have simple tests

if () {} else {}

If you have only one condition:

```
if (test_1) {
  command_1
} else {
  command_2
}
```

if () {} else {}

Multiple conditions:

```
if (test_1) {
  command_1
} else if (test_2) {
  command_2
} else {
  command_3
}
```

`test_1` (and `test_2`, if you have it) **must** evaluate in either `TRUE` or `FALSE`

```
if(!is.na(x)) y <- x^2 else stop("x is missing")
```

Loops

`for()` and `while()`

Repeat a command over and over again:

```r
# Don't do this at home
x <- rnorm(10)
y <- numeric(10)   # create an empty container
for(i in seq_along(x)) {
y[i] <- x[i] - mean(x)
}
```

The best solution would have been:

```r
y = x - mean(x)
```

Avoiding loops

Don't loop, `apply()`!

`apply()`

```
X <- matrix(rnorm(20),
            nrow = 5, ncol = 4)
apply(X, 2, max) # maximum for each column
```

```
[1] 1.4203744 1.0716663 2.3329026 0.9084482
```

Avoiding loops

Don't loop, `apply()`!

`apply()`

```
X <- matrix(rnorm(20),
            nrow = 5, ncol = 4)
apply(X, 2, max) # maximum for each column
```

```
[1] 1.4203744 1.0716663 2.3329026 0.9084482
```

`for()`

```
y = NULL
for (i in 1:ncol(X)) {
  y[i] = max(X[, i])
}
```

Avoiding loops

Group-wise calculations: `tapply()`

`tapply()` (`t` for table) may be used to do group-wise calculations on vectors. Frequently it is employed to calculate group-wise means.

```
with(ToothGrowth,
     tapply(len, list(supp, dose), mean))
```

```
     0.5     1     2
OJ 13.23 22.70 26.06
VC  7.98 16.77 26.14
```

(You could have done it with `aggregate()`)

Writing functions

Compute Cohen's *d*:

```
dcohen = function(group1, group2) { # Arguments
  mean_1 = mean(group1) ; mean_2 = mean(group2)
  var_1 = var(group1) ; var_2 = var(group2) # body
  d = (mean_1 - mean_2)/sqrt(((var_1 + var_2)/2) )
  return(d) # results
}
```

Use it:

```
dcohen(data$placebo, data$drug)
```

Named arguments

Take this function:

```
fun1 <- function(data, data.frame, graph, limit) { ... }
```

It can be called as:

```
fun1(d, df, TRUE, 20)
fun1(d, df, graph=TRUE, limit=20)
fun1(data=d, limit=20, graph=TRUE, data.frame=df)
```

Positional matching and keyword matching (as in built-in functions)

Defaults

Arguments can be given default values → the arguments can be omitted!

```
fun1 <- function(data, data.frame, graph=TRUE,
                 limit=20) { ... }
```

It can be called as

```
ans <- fun1(d, df)
```

which is now equivalent to the three cases above, but:

```
ans <- fun1(d, df, limit=10)
```

which changes one of the defaults.

Methods and classes

The return value of a function may have a specified `class` $\rightarrow$ determines how it will be treated by other functions.

For example, many classes have tailored `print` methods.

```
methods(print)
```

```
  [1] print.acf*
  [2] print.AES*
  [3] print.all_vars*
  [4] print.anova*
  [5] print.any_vars*
  [6] print.aov*
  [7] print.aovlist*
....
```

Define a print method!

. . . as another function:

```
print.cohen <- function(obj){
  cat("\nMy Cohen's d\n\n")
  cat("Effect size: ", obj$d, "\n")
  invisible(obj) # return the object
}
```

We have to change our dcohen function a bit:

```
dcohen = function(group1, group2) { # Arguments
  ...
  dvalue = list(d = d)
  class(dvalue) = "cohend"
  return(dvalue) # results
}
```

Example

Compute the Cohen's *d* between a test group and a control group:

```r
set.seed(082022) # results equal for everyone
data <- data.frame(drug = rnorm(6, 10),
                   placebo = rnorm(6, 2))
my_d = dcohen(data$drug, data$placebo)
print.cohen(my_d)
```

```
My Cohen's d

Effect size:  6.900794
```

Debugging

Use the `traceback()` function:

```
foo <- function(x) { print(1); bar(2) }
bar <- function(x) { x + a.variable.which.does.not.exist }
```

Call `foo()` and...

```
foo() #
[1] 1
Error: object 'a.variable.which.does.not.exist' not found
```

Use `traceback()`:

```
traceback() # find out where the error occurred
2: bar(2)
1: foo()
```

Note: `traceback()` appears as default

Table of Contents

- Traditional graphics
- Grid graphics & `ggplot2`

For both:

- High level functions $\rightarrow$ actually produce the plot
- Low level functions $\rightarrow$ make it looks better =)

Export graphics file

```
postscript()  # vector graphics
pdf()

png()            # bitmap graphics
tiff()
jpeg()
bmp()
```

Remember to run off the graphic device once you've saved the graph:

```
dev.off()
```

(You can do it also manually)

Traditional graphics I

High level functions

```
plot()          # scatter plot, specialized plot methods
boxplot()
hist()          # histogram
qqnorm()        # quantile-quantile plot
barplot()
pie()           # pie chart
pairs()         # scatter plot matrix
persp()         # 3d plot
contour()       # contour plot
coplot()        # conditional plot
interaction.plot()
```

`demo(graphics)` for a guided tour of base graphics!

Traditional graphics II

Low level functions

```
points()        # add points
lines()         # add lines
rect()
polygon()
abline()        # add line with intercept a, slope b
arrows()
text()          # add text in plotting region
mtext()         # add text in margins region
axis()          # customize axes
box()           # box around plot
legend()
```

Plot layout

Each plot is composed of two regions:

- The plotting regions (contains the actual plot)
- The margins region (contain axes and labels)

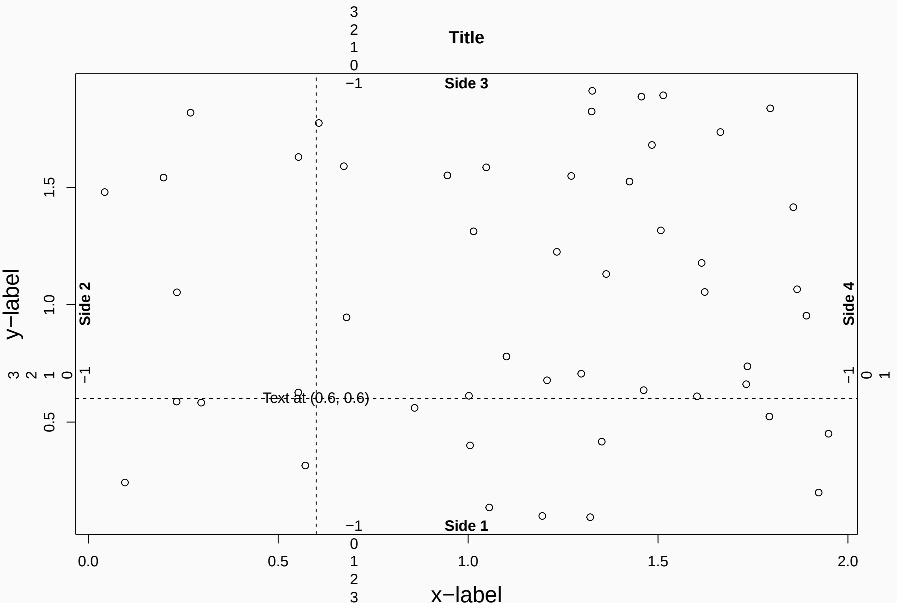
A scatter plot:

```
x <- runif(50, 0, 2) # 50 uniform random numbers
y <- runif(50, 0, 2)
plot(x, y, main="Title",
     sub="Subtitle", xlab="x-label",
     ylab="y-label") # produce plotting window
```

Now add some text:

```
text(0.6, 0.6, "Text at (0.6, 0.6)")
abline(h=.6, v=.6, lty=2) # horizont. and vertic.
                          # lines
```

Margins region



Rome wasn't built in a day and neither your graph

Display the interaction between the two factors of a two-factorial experiment:

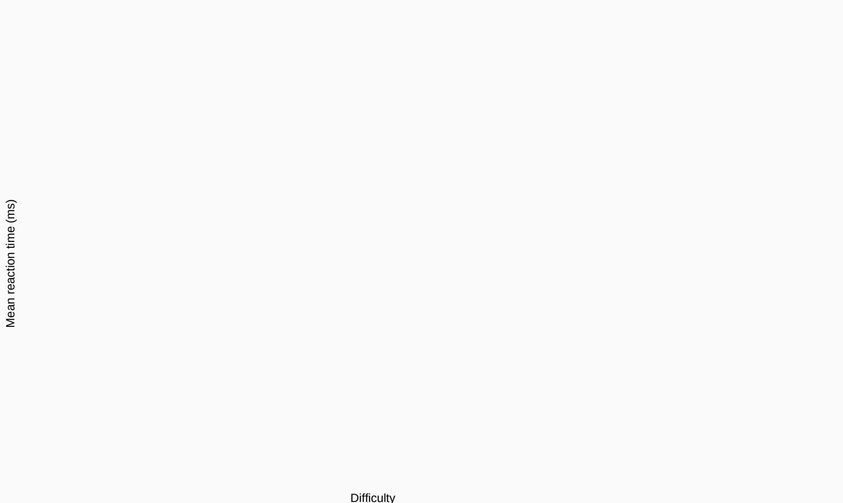
```
dat <- read.table(header=TRUE, text="
A B rt
a1 b1 825
a1 b2 792
a1 b3 840
a2 b1 997
a2 b2 902
a2 b3 786
")
```

Force `A` and `B` to be `factor`:

```
dat[,1:2] = lapply(dat[,1:2], as.factor)
```

First: The plot

```r
plot(rt ~ as.numeric(B), dat, type="n", axes=FALSE,
     xlim=c(.8, 3.2), ylim=c(750, 1000),
     xlab="Difficulty", ylab="Mean reaction time (ms)")
```



Populate the content

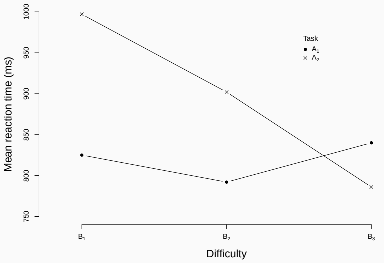
Plot the data points separately for each level of factor `A`.

```
points(rt ~ as.numeric(B), dat[dat$A=="a1",],
       type="b", pch=16)
points(rt ~ as.numeric(B), dat[dat$A=="a2",],
       type="b", pch=4)
```

Add axes and a legend.

```
axis(side=1, at=1:3, expression(B[1], B[2], B[3]))
axis(side=2)
legend(2.5, 975, expression(A[1], A[2]), pch=c(16, 4),
       bty="n", title="Task")
```

Final result



- Error bars may be added using the `arrows()` function.
- Via `par()` many graphical parameters may be set (see `?par`), for example `par(mgp=c(2, .7, 0))` reduces the distance between labels and axes

Graphical parameters I

```
adj # justification of text
bty # box type for legend
cex # size of text or data symbols (multiplier)
col # color, see colors()
las # rotation of text in margins
lty # line type (solid, dashed, dotted, ...)
lwd # line width
mpg # placement of axis ticks and tick labels
pch # data symbol type
tck # length of axis ticks
type # type of plot (points, lines, both, none)
```

Graphical parameters II

par()

```
mai # size of figure margins (inches)
mar # size of figure margins (lines of text)
mfrow # number of sub-figures on a page:
      # par(mfrow=c(1, 2)) creates two sub-figures
oma # size of outer margins (lines of text)
omi # size of outer margins (inches)
pty # aspect ratio of plot region (square, maximal)
```

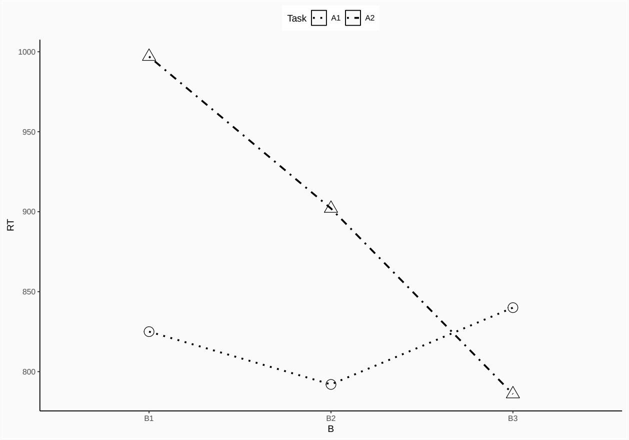
ggplot2

ggplot2 (Grammar of Graphics plot, Wickman, 2016) is one of the best packages for plotting raw data and results:

```
install.packages("ggplot2") ; library(ggplot2)
```

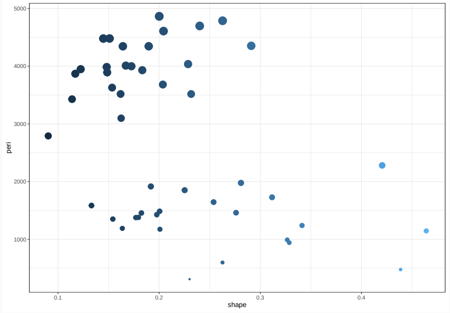
The code for the previous plot:

```
ggplot(dat, aes(x = B, y = rt, group = A)) +
  geom_point(pch=dat$A, size = 5) +
  geom_line(aes(linetype=A), size=1)  + theme_classic() +
  ylab("RT") +  scale_linetype_manual("Task", values =c(3,4),
                                   labels = c("A1", "A2")) +
  scale_x_discrete(labels = c("B1", "B2", "B3")) +
  theme(legend.position="top",
        panel.background =  element_rect(fill = "#FAFAFA",
                                         colour = "#FAFAFA"),
        plot.background = element_rect(fill = "#FAFAFA"),
        legend.key = element_rect(fill = "#FAFAFA"))
```


Task
A1
A2
1000
950
900
850
800
RT
B1
B2
B3
B

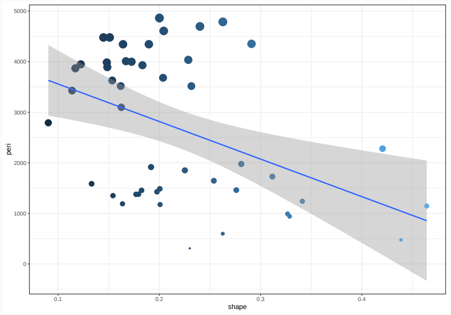
Raw data

```
ggplot(rock,
       aes(y=peri,x=shape, color =shape,
           size = peri)) + geom_point() +
  theme_bw() + theme(legend.position = "none")
```

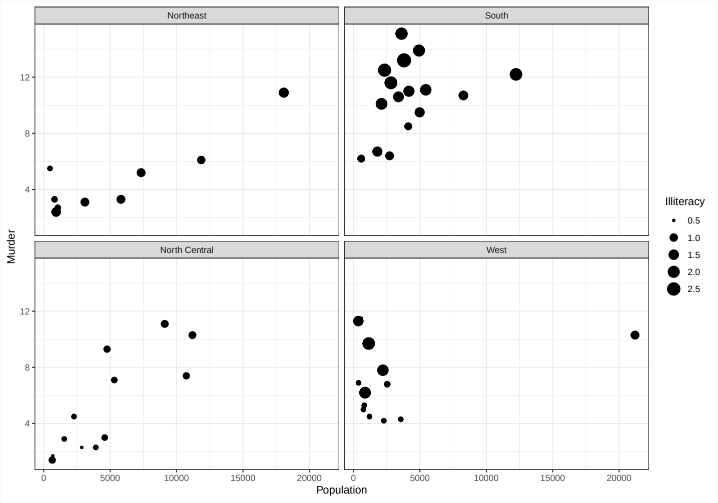


Linear model

```
ggplot(rock,
       aes(y=peri,x=shape, color =shape,
           size = peri)) + geom_point() +
  theme_bw() + theme(legend.position = "none") +
  geom_smooth(method="lm")
```



Multi Panel


Northeast
South
North Central
West
Murder
Population
4
8
12
0
5000
10000
15000
20000
Illiteracy
0.5
1.0
1.5
2.0
2.5

Multi panel code

```
states = data.frame(state.x77, state.name = state.name,
                    state.region = state.region)

ggplot(states,
       aes(x = Population, y = Murder,
           size = Illiteracy)) + geom_point() +
  facet_wrap(~state.region) + theme_bw()
```

Different plots in the same panel

use grid.arrange() function from the gridExtra package:

```
install.packages("grideExtra") ; library(gridExtra)
```

```
murder_raw = ggplot(states,  # raw data
                    aes(x = Illiteracy, y = Murder)) +
           .....

murder_lm = ggplot(states,  # lm
                    aes(x = Illiteracy, y = Murder)) +
           .....
```

Combine the plots together:

```
grid.arrange(murder_raw, murder_lm,
             nrow=1) # plots forced to be the same row
```

Combine the plots together

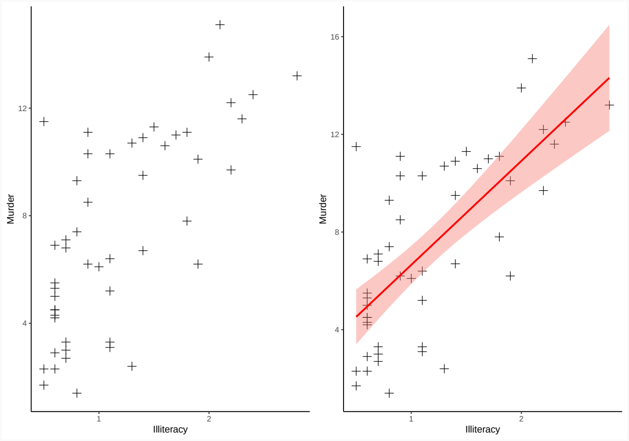


Table of Contents

The `stats` package (built-in package in `R`) contains function for statistical calculations and random number generator

see `library(help=stats)`

Table of Contents

Nominal data:

- `binom.test()`: exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment
- `chisq.test()`: contingency table χ^2 tests

Metric response variable:

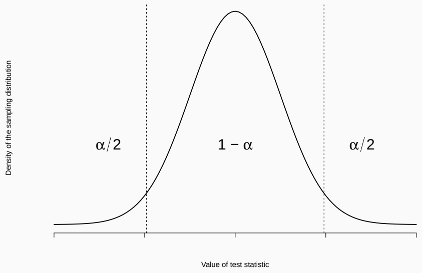
- `cor.test()`: association between paired samples
- `t.test()`: one- and two-sample t tests (also for paired data)
- `var.test()`: F for testing the homogeneity of variances

What is the *p*-value?

p-value:

> *conditional probability of obtaining a test stastic that is at least as extreme as the one observed, given that the null hyphothesis is true*

If $p < \alpha$ (i.e., the probability of rejecting the null hypothesis when it is true) $\rightarrow$ the null hypothesis is rejected



Binomial test

Observations X_i **must** be independent

Hypotheses:

1. $H_0\colon\ p = p_0 \quad H_1\colon\ p \neq p_0$
2. $H_0\colon\ p = p_0 \quad H_1\colon\ p < p_0$
3. $H_0\colon\ p = p_0 \quad H_1\colon\ p > p_0$

Test statistic:

$$T = \sum_{i=1}^{n} X_i, \quad T \sim \mathcal{B}(n, p_0)$$

In R:

```
binom.test(5, 10, p = 0.25)
```

χ^2 test

Independence of observations

Hypothesis:

- H_0: $P(X_{ij} = k) = p_k$ for all $i = 1, \ldots, r$ and $j = 1, \ldots, c$
- H_0: $P(X_{ij} = k) \neq P(X_{i'j} = k)$ for *at least* one $i \in \{1, \ldots, r\}$ and $j \in \{1, \ldots, c\}$

Test statistic:

$$\chi^2 = \sum_{i=1}^{n} \frac{(x_{ij} - \hat{x}_{ij})^2}{\hat{x}_{ij}}, \; \chi^2 \sim \chi^2(r-1)(c-1)$$

In R:

```r
tab <- xtabs(~ education + induced, infert)
chisq.test(tab)
```

Correlation test

Hypothesis:

- H_0: $\rho_{XY} = 0$, H_1: $\rho_{xy} \neq 0$
- H_0: $\rho_{XY} = 0$, H_1: $\rho_{xy} < 0$
- H_0: $\rho_{XY} = 0$, H_1: $\rho_{xy} > 0$

Test statistic:

$$T = \frac{r_{xy}}{\sqrt{1 - r_{xy}^2}} \sqrt{n-2}, \;\; T \sim t(n-2)$$

In R:

```r
cor.test(~ speed + dist, cars,
         alternative = "two.sided")
```

Two (indepdent) sample t test

Independent samples from normally distributions where σ^2 are unknown but homogeneous

- H_0: $\mu_{x_1-x_2} = 0$, H_1: $\mu_{x_1-x_2} \neq 0$
- H_0: $\mu_{x_1-x_2} = 0$, H_1: $\mu_{x_1-x_2} < 0$
- H_0: $\mu_{x_1-x_2} = 0$, H_1: $\mu_{x_1-x_2} > 0$

Test statistic:

$$T = \frac{\bar{x_1} - \bar{x_2}}{\sigma_{\bar{x_1}-\bar{y_2}}},\ \ T \sim t(n_1 + n_2 - 2)$$

R function:

```
t.test(len ~ supp, data = ToothGrowth,
       var.equal = TRUE)
```

Two (depedent) sample t test

Observations on the same sample

Hypothesis:

- H_0: $\mu_D = 0$, H_1: $\mu_D \neq 0$
- H_0: $\mu_D = 0$, H_1: $\mu_D < 0$
- H_0: $\mu_D = 0$, H_1: $\mu_D > 0$

Test statistic:

$$T = \frac{d}{\sigma_d}, \quad T \sim t(m-1)$$

R function:

```
with(sleep,
     t.test(extra[group == 1],
            extra[group == 2], paired = TRUE))
```

Table of Contents

Formulae

Statistical models are represented by formulae which are extremely close to the actual statistical notation:

in `R`	Model
`y ~ 1 + x`	$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$
`y ~ x`	(same but short)
`y ~ 0 + x`	$y_i = \beta_1 x_i + \varepsilon_i$
`y ~ x_A * x_B`	$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_j + (\beta_1 \beta_2) x_{ij} + \varepsilon_{ij}$

Linear models

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon$$

In R:

```
lm(y ~ x1 + x2 + ... + xk, data)
```

Extractor functions I

```
coef()    # Extract the regression coefficients
summary() # Print a comprehensive summary of the results of
          # the regression analysis
anova()   # Compare nested models and produce an analysis
resid()   # Extract the (matrix of) residuals
plot()    # Produce four plots, showing residuals, fitted
          # values and some diagnostics
model.matrix()
          # Return the design matrix
```

Extractor functions II

```
vcov()    # Return the variance-covariance matrix of the
          # main parameters of a fitted model object
predict() # A new data frame must be supplied having the
          # same variables specified with the same labels
          # as the original. The value is a vector or
          # matrix of predicted values corresponding to
          # the determining variable values in data frame
step()    # Select a suitable model by adding or dropping
          # terms and preserving hierarchies. The model
          # with the smallest value of AIC (Akaike's
          # Information Criterion) discovered in the
          # stepwise search is returned
```

Generalized linear models

$$g(\mu) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \varepsilon$$

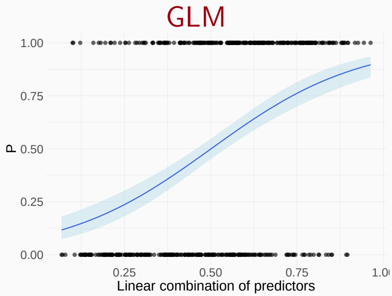
$g()$ is the link functions that connects the mean to the linear combination of predictors.

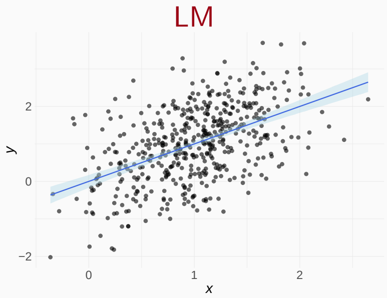
A GLM is composed of three elements: The response distribution, the link function, and the linear combination of predictors

In `R`:

```
glm(y ~ x1 + x2 + ... + xk, family(link), data)
```

LM vs GLM


GLM
1.00
0.75
0.50
0.25
0.00
P
0.25
0.50
0.75
1.00
Linear combination of predictors


LM
2
0
−2
y
0
1
2
x

Families

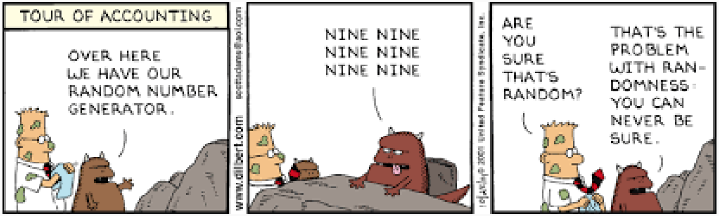
A special link function to each response variable. In R some different link functions are available by default:

```
## Family name       Link functions
binomial            logit, probit, log, cloglog
gaussian            identity, log, inverse
Gamma               identity, inverse, log
inverse.gaussian    1/mu^2, identity, inverse, log
poisson             log, identity, sqrt
quasi               logit, probit, cloglog, identity, inverse,
                    log, 1/mu^2, sqrt
```

Table of Contents

Random numbers generation

Use a random monster:


TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
NINE NINE NINE NINE NINE NINE
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.
www.dilbert.com
scottadams@aol.com
© 2001 United Feature Syndicate, Inc.

but its better with R

Random numbers drawn from a statistical distribution → the distribution name (see `??Distributions` for an exhaustive list of distribution) prefixed by `r` (random)

```
rnorm(10, mean = 0, sd = 1) # draw 10 numbers from a
                            # normal distr.
rt(10, df = 20)             # draw 10 numbers from a
                            # t distr. with 20df
```

Sampling (with or without replacement) from a vector:

```
sample(1:5, size = 10, replace = T)
```

```
 [1] 5 1 1 2 5 4 4 1 2 5
```

Make the simulations replicable by *seeding* them:

```
set.seed(999)
rpois(4, 5)
```

Bootstrap by resampling

- Compute the sample statistics on multiple bootstrap samples Bs drawn with replacement from the original data
- Assess the variability of the statistics via the distribution of the bootstrap replicates (i.e., the statics computed on the bootstrap samples)

Bootstrap confidence intervals

Percentile intervals are the $1 - \alpha$ confidence intervals for the sample statistics with limits given by the quantiles of the bootstrap distribution

In R

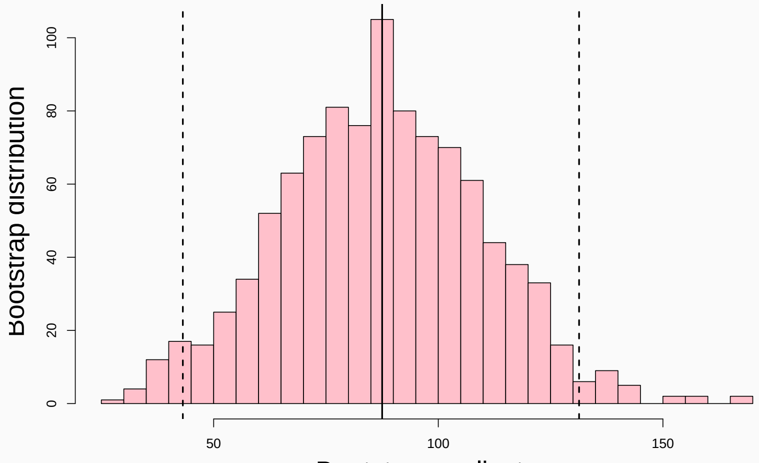
```r
# example taken from Prof. Wickelmaier
mouse <- data.frame(
   grp = rep(c("trt", "ctl"), c(7, 9)),
   surv = c(94, 197, 16, 38, 99, 141, 23, # trt
            52, 104, 146, 10, 50, 31, 40, 27, 46) # ctl
   )
mean(mouse$surv[mouse$grp == "trt"]) #
```

```
[1] 86.85714
```

```r
## Resampling
sam1 <- numeric(1000) # 1000 bootstrap replicates
for(i in seq_along(sam1)){
 trt <- sample(mouse$surv[mouse$grp == "trt"], 7, replace=T)
 sam1[i] <- mean(trt)
}
```

```
quantile(sam1, c(.025, .975))
```

```
    2.5%     97.5%
 43.14286 131.33929
```



Parametric bootstrap

For the likelihood ratio test:

- Fit a general (M_1) and a restricted model (M_0) to the original data x. Compute the original likelihood ratio s(x) between M_1 and M_0
- Simulate B bootstrap samples based on the stochastic part of the restricted model: These are observations for which H_0 is true
- For each sample, fit M_1 and M_0 and compute the bootstrap replicate of the likelihood ratio between them
- Assess the significance of the original likelihood ratio via the sampling distribution of bootstrap replicates

```
## Model fit to original data
lm0 <- lm(surv ~ 1, mouse)  # H0: no difference between gr
lm1 <- lm(surv ~ grp, mouse) # H1: group effect
anova(lm0, lm1)             # original likelihood ratio
```

```
[1] 1.257516
```

```
 ## Parametric bootstrap
sim1 <- numeric(1000)
for(i in seq_along(sim1)){
  surv0 <- simulate(lm0)$sim_1  # simulate from null model
  m0 <- lm(surv0 ~ 1, mouse)    # fit null model
  m1 <- lm(surv0 ~ grp, mouse)  # fit alternative model
  sim1[i] <- anova(m0, m1)$F[2] # bootstrap likeli. ratio
  }
```

The bootstrap $p - value$ is the proportion of bootstrap replicates that exceed the original likelihood ratio:

```
mean(sim1 >
anova(lm0, lm1)$F[2])
```

```
[1] 0.304
```

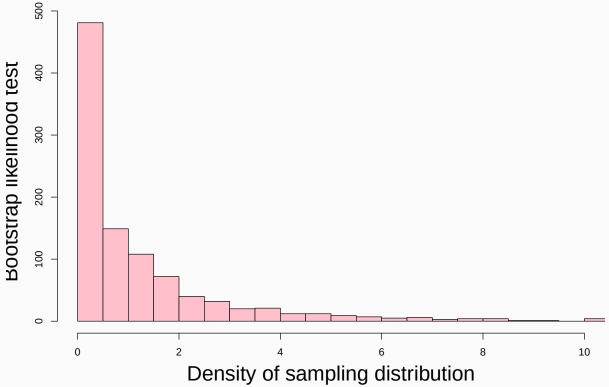


Table of Contents

Table of Contents

Table of Contents

Table of Contents