

# INTRODUZIONE A R E RSTUDIO

Paolo Bosetti (`paolo.bosetti@unitn.it`)

11/23/2021

# CONTENUTI DEL CORSO

## CONTENUTI

- ➊ Richiami di statistica, 4h (*slide pdf*)
- ➋ L'ambiente RStudio, 0.5h (*RIntro.Rmd*)
- ➌ Il linguaggio R, 1.5h (*RIntro.Rmd*)
- ➍ Statistica base in R, 4h (*statistics.Rmd*)
- ➎ Modelli di regressione lineare e lineare generalizzata, 2h (*reglin.Rmd*)
- ➏ Serie temporali e modelli ARIMA, 3.5h (*ARIMA.Rmd*)
- ➐ *Tidyverse*, 3.5h (*tidy.Rmd*)
- ➑ Mappe e GIS, 2h (*maps.Rmd*)
- **TOTALE: 21 h**

## NOTE

- Queste slide coprono solo la parte introduttiva ad R (2.–3.)
- Il resto è disponibile come notebook

Questa presentazione è realizzata in RStudio e sarà disponibile con il resto del materiale su github: <https://github.com/pbosetti/tsm-stat>.

# CONTENUTI DEL CORSO

## LINK UTILI

- GNU-R: <https://r-project.org>
- CRAN: <https://cran.r-project.org>
- RStudio: <https://rstudio.com>
- Tidyverse: <https://tidyverse.org>
- Cheat sheet: <https://rstudio.com/resources/cheatsheets/>
- Materiale corso: <https://github.com/pbosetti/tsm-stat>

# L'AMBIENTE RSTUDIO

## AMBIENTE

- Installazione: prima R, poi RStudio
- RStudio lavora su cartelle o (meglio) **progetti** (.Rproj)
- Un progetto contiene anche impostazioni specifiche e comuni ai file nella cartella
- Una **sessione** di RStudio può operare su un unico progetto
- Si possono aprire più sessioni contemporaneamente

## ATTIVITÀ

- Editing di script .R e notebook .Rmd
- Esecuzione del codice e gestione dell'ambiente dati
- Plotting
- Gestione file di progetto
- Generazione di report (in  $\text{\LaTeX}$ , HTML, RTF, ...)
- Tracciamento del codice (Git)
- Gestione delle librerie (estensioni di linguaggio)

# IL LINGUAGGIO R

- R è un linguaggio ad alto livello, dichiarativo, interpretato, a sintassi C-like
- R è sia un linguaggio, sia un interprete
- R è un *dynamically typed language*
- R è nato come versione GNU open source di S, un linguaggio proprietario per analisi statistiche
- RStudio è una IDE proprietaria (ma free) per R

# IL LINGUAGGIO R

## ASSEGNAZIONI

```
a <- 1
# ma anche
b = 2
# tuttavia si preferisce la notazione a freccia,
# perché funziona anche così:
3 -> c
# per visualizzare il valore di una variabile:
c
## [1] 3
# in un colpo solo:
(d <- "stringa")
## [1] "stringa"
```

## TIPI, O CLASSI NATIVE

- R ha 6+1 tipi o *classi* native
  - character: "a", "string", 'my text'
  - numeric: 1, 3.1415
  - integer: 1L
  - logical: TRUE, FALSE (oppure T e F)
  - complex: 1+4i
  - function: una *funzione*
- Ogni istanza è intrinsecamente un vettore
- Uno scalare è semplicemente un vettore di lunghezza 1

## VALORI SPECIALI

- Sono definiti i seguenti valori speciali:
  - NA: valore mancante
  - NULL: niente
  - Inf: Infinito
  - NaN: Not a Number (esempio  $0/0$ )



# IL LINGUAGGIO R

## COERCIZIONE

- Quando si mescolano tipi differenti, ad es. in un vettore, R li trasforma in un tipo comune:

```
c(1L, 7, "2")
```

```
## [1] "1" "7" "2"
```

```
c(T, 0)
```

```
## [1] 1 0
```

```
as.numeric(c("a", "1"))
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA 1
```

```
as.character(c(1, 1.7))
```

```
## [1] "1" "1.7"
```

# IL LINGUAGGIO R

## VETTORI

```
# Si costruiscono con l'operatore/funzione c():  
v1 <- c(10, 2, 7.5, 3)  
# oppure con una sequenza:  
v2 <- 1:10  
# anche con passo specificato:  
v3 <- seq(1, 10, 0.5)  
# Le funzioni si chiamano con le parentesi tonde,  
# separando argomenti con ,
```

## INTROSPEZIONE

- Funzioni utili per ispezionare gli oggetti:
  - `class()`: classe (alto livello)
  - `typeof()`: tipo (basso livello)
  - `length()`: lunghezza vettore
  - `attributes()`: metadati

## MATRICI

- Si costruiscono con la funzione `matrix()`

```
(m1 <- matrix(1:10, 2, 5))
```

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

- la funzione `array()` costruisce matrici  $n$ -dimensionali

# IL LINGUAGGIO R

UNA MATRICE È UN VETTORE CON ATTRIBUTO `DIM`

```
attr(m1, "dim")
```

```
## [1] 2 5
```

```
v <- 1:4
```

```
attr(v, "dim") <- c(2,2) # equivale a dim(m) <- c(2,2)
```

```
v
```

```
##      [,1] [,2]
```

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

```
## [2,]    2    4
```

# IL LINGUAGGIO R

## FATTORI

- Una classe aggiuntiva (non base) ma molto comune è factor
- Rappresenta variabili categoriche (ordinate o non)

```
(vf <- factor(LETTERS[1:5], ordered=T))
```

```
## [1] A B C D E
```

```
## Levels: A < B < C < D < E
```

```
class(vf)
```

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

```
typeof(vf)
```

```
## [1] "integer"
```

```
vf[1] < vf[3]
```

```
## [1] TRUE
```

# IL LINGUAGGIO R

## VETTORI

```
# Le variabili sono nativamente dei vettori.
```

```
# Gli scalari sono solo vettori di dimensione 1
```

```
length(a)
```

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

```
length(v1)
```

```
## [1] 4
```

```
# le funzioni agiscono quindi sempre su vettori:
```

```
a * 2
```

```
## [1] 2
```

```
v3 + 2
```

```
## [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

```
## [16] 10.5 11.0 11.5 12.0
```

# IL LINGUAGGIO R

## INDICIZZAZIONE

- La sintassi di indicizzazione di R è molto flessibile e potente
- si usano sempre le parentesi quadre `[r,c]`, la **base è 1**
- se un indice manca, significa “tutte le righe|colonne”

```
v1[3]
```

```
## [1] 7.5
```

```
m1[1,1]
```

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

```
m1[2,]
```

```
## [1] 2 4 6 8 10
```

```
m1[,]
```

```
##      [,1] [,2] [,3] [,4] [,5]
```

```
## [1,]    1    3    5    7    9
```

```
## [2,]    2    4    6    8   10
```



# IL LINGUAGGIO R

## INDICIZZAZIONE

- Un indice può essere anche un vettore di posizioni o un vettore di valori booleani

```
v1[c(2,4,1)]
```

```
## [1] 2 3 10
```

```
v2[v2 %% 2 == 0]
```

```
## [1] 2 4 6 8 10
```

```
v2 %% 2 == 0 # operatore modulo (resto)
```

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

## NOTA

- TRUE e FALSE possono essere abbreviati in T e F

# IL LINGUAGGIO R

## FUNZIONI

- Le funzioni sono *first class objects*, cioè sono variabili come altre
- possono essere assegnate a variabili e passate a funzioni

```
my_fun <- function(x) x^2
```

```
my_fun(1:5)
```

```
## [1] 1 4 9 16 25
```

```
your_fun <- my_fun
```

```
your_fun(6)
```

```
## [1] 36
```

```
my_apply <- function(x, f) f(x)
```

```
my_apply(10, my_fun)
```

```
## [1] 100
```

## FUNZIONI FRECCIA (*REPLACEMENT FUNCTIONS*)

- Abbiamo visto cose come `dim(v) <- c(2,3)`: come si dichiarano?

```
`pwr<-` <- function(obj, value) obj ** value  
a <- 2  
pwr(a) <- 10  
a  
## [1] 1024
```

- L'ultimo argomento **deve** chiamarsi `value` e rappresenta la rhs dell'assegnazione!

# IL LINGUAGGIO R

## FUNZIONI

- Se la definizione richiede più righe, si usa un **blocco** tra `{}`
- Ogni funzione ritorna **sempre** l'ultima espressione valutata
- Oppure esplicitamente mediante `return()`

## CONTROLLO DI FLUSSO

- `if(cond) expr`
- `if(cond) cons.expr else alt.expr`
- `for(var in seq) expr`
- `while(cond) expr`
- `repeat expr`
- `break`
- `next`

# IL LINGUAGGIO R

## ARGOMENTI DELLE FUNZIONI

- Gli argomenti possono essere indicati per posizione o per nome
- Gli argomenti nominati possono comparire in qualsiasi ordine
- Gli argomenti possono avere un default, in tal caso sono opzionali

```
f <- function(x, y, n=10, test=F) {  
  ifelse(test, 0, x^y + n)  
}
```

```
f(2, 10)
```

```
## [1] 1034
```

```
f(test=F, y=10, x=2)
```

```
## [1] 1034
```

```
f(test=T)
```

```
## [1] 0
```

# IL LINGUAGGIO R

## DIFFERENZA TRA `<-` E `=`

- L'operatore `=` è valido solo al *top-level*
- L'operatore `<-` è valido ovunque, anche come argomento di funzione:

```
system.time(m <- mean(1:1E6))
```

```
##      user  system elapsed  
##    0.005    0.000    0.006  
  
m
```

```
## [1] 500000.5
```

# IL LINGUAGGIO R

## DATAFRAME

- In R più che matrici si usano dataframe
- Si tratta di tabelle organizzate per colonne, internamente omogenee ma potenzialmente di tipi differenti

```
df <- data.frame(A=1:10, B=letters[1:10])
```

```
head(df)
```

```
##    A B
## 1 1 a
## 2 2 b
## 3 3 c
## 4 4 d
## 5 5 e
## 6 6 f
```

# IL LINGUAGGIO R

## DATAFRAME

- Un dataframe può essere indicizzato come una matrice (due indici)
- Oppure selezionando una colonna con la notazione \$

```
df[2,2]
```

```
## [1] "b"
```

```
df$B[2]
```

```
## [1] "b"
```

```
# anche in assegnazione
```

```
df$C <- LETTERS[1:10]
```

```
head(df, 3)
```

```
##      A B C
```

```
## 1  1 a A
```

```
## 2  2 b B
```

```
## 3  3 c C
```



## ALGORITMI DI USO COMUNE

- Ordinamento: `sort`, `rev`, `order`
- Campionamento: `sample`, `expand.grid`
- Aggregazione: `by`, `aggregate`
- Mappatura: `apply`, `lapply`, `sapply`
- Tabelle di contingenza: `table`

# IL LINGUAGGIO R

## ORDINAMENTO DI VETTORI

```
v <- runif(5, 1, 10)
sort(v)
## [1] 2.402919 2.531490 4.523276 6.242139 7.334885
rev(sort(v))
## [1] 7.334885 6.242139 4.523276 2.531490 2.402919
sort(v, decreasing = T)
## [1] 7.334885 6.242139 4.523276 2.531490 2.402919
```

# IL LINGUAGGIO R

## ORDINAMENTO DI DATAFRAME

```
df <- data.frame(A=1:5, B=runif(5))  
df[order(df$B),]
```

```
##      A          B  
## 3 3 0.1055260  
## 4 4 0.2659783  
## 1 1 0.4967314  
## 2 2 0.7435799  
## 5 5 0.8288405
```

# IL LINGUAGGIO R

## CAMPIONAMENTO

```
sample(1:10) # con reinserimento
```

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

```
sample(1:10, replace = T) # senza reinserimento
```

```
## [1] 2 4 7 1 6 2 2 8 2 2
```

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

```
## [1] 2 4 10 9 1
```

```
sample(10) # generazione interi casuali senza ripetizione
```

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

## GRIGLIE

```
(df <- expand.grid(A=1:2, B=c("-", "+")))
```

```
##      A B
```

```
## 1 1 -
```

```
## 2 2 -
```

```
## 3 1 +
```

```
## 4 2 +
```

# IL LINGUAGGIO R

## AGGREGAZIONE

```
by(df$A, INDICES = df$B, FUN=sum)

## df$B: -
## [1] 3
## -----
## df$B: +
## [1] 3

aggregate(A~B, data = df, FUN = sum)

##      B A
## 1 - 3
## 2 + 3
```

# IL LINGUAGGIO R

## MAPPATURA

- `apply` applica una funzione ai margini di una matrice o array

```
(m <- matrix(sample(6), 2, 3))
```

```
##      [,1] [,2] [,3]  
## [1,]    6    3    5  
## [2,]    2    4    1
```

```
apply(m, 1, sum)
```

```
## [1] 14  7
```

```
apply(m, 2, sum)
```

```
## [1] 8 7 6
```

## MAPPATURA

- `lapply` applica una funzione agli elementi di un vettore e ritorna una lista

```
v <- 1:10  
head(lapply(v, sqrt), n = 2)  
## [[1]]  
## [1] 1  
##  
## [[2]]  
## [1] 1.414214
```



## MAPPATURA

- `sapply` applica una funzione agli elementi di un vettore e ritorna un vettore

```
v <- 1:5
names(v) <- paste0("sqrt(",v,")")
sapply(v, sqrt)

##  sqrt(1)  sqrt(2)  sqrt(3)  sqrt(4)  sqrt(5)
## 1.000000 1.414214 1.732051 2.000000 2.236068
```

# IL LINGUAGGIO R

## TABELLE DI CONTINGENZA

```
head(airquality, n = 3)
```

```
##      Ozone Solar.R Wind Temp Month Day
## 1      41      190  7.4   67     5   1
## 2      36      118  8.0   72     5   2
## 3      12      149 12.6   74     5   3
```

```
with(airquality, table(OzHi = Ozone > 80, Month,
                        useNA = "ifany"))
```

```
##           Month
## OzHi      5  6  7  8  9
## FALSE  25  9 20 19 27
## TRUE   1  0  6  7  2
## <NA>   5 21  5  5  1
```

# IL LINGUAGGIO R

## TABELLE DI CONTINGENZA

- È anche utile `tapply()`:
- **NOTA:** `with()` serve per risparmiarsi di scrivere `airquality$Ozone`.

```
round(with(airquality,  
          tapply(Ozone, Month, mean, na.rm=T)), 1)
```

```
##      5      6      7      8      9  
## 23.6 29.4 59.1 60.0 31.4
```

```
# 0 anche:
```

```
aggregate(Ozone~Month, data=airquality, FUN=mean, na.rm=T)
```

```
##      Month      Ozone  
## 1         5 23.61538  
## 2         6 29.44444  
## 3         7 59.11538  
## 4         8 59.96154  
## 5         9 31.44828
```

## TIDYVERSE

Assieme a RStudio è emersa una *new wave* di librerie R che modificano radicalmente l'approccio. Vanno sotto il nome collettivo di `tidyverse`

- `ggplot2`: grafici
- `purrr`: programmazione funzionale
- `dplyr`: manipolazione dati
- `stringr`: manipolazione stringhe
- `tibble`: data frame migliorati
- `readr`: importazione dati
- `tidyr`: preparazione dati
- `lubridate`: manipolazione date

## TIDYVERSE

L'approccio tidyverse ha alcune caratteristiche comuni:

- dati in formato **tidy** (un'osservazione per riga, un osservando per colonna)
- composizione di funzioni con `+` (`ggplot(...)` `+` `geom_line()`)
- notazione prefissa con `%>%` (a `%>% str()` invece di `str(a)`)

È utile consultare i cheat sheet:

<https://www.rstudio.com/resources/cheatsheets/>

## PLOT

- I principali comandi di plot sono:
  - `plot`: interfaccia generica
  - `lines`: plotta una serie come linea, accetta l'opzione `add=T`
  - `points`: plotta una serie come punti, accetta l'opzione `add=T`
  - `curve`: plotta una funzione di `x`, accetta l'opzione `add=T`
  - `abline`: plotta una linea retta
  - `hist`: istogramma
  - `boxplot`: ditto

## STIMATORI

- Media campionaria: `mean(x, na.rm=F)`
- Varianza campionaria: `var(x, na.rm=F)`
- Deviazione standard: `sd(x, na.rm=F)`
- Mediana: `median(x, na.rm=F)`
- Moda: `mode(x, na.rm=F)`
- Covarianza: `cov(x, y, na.rm=F)`
- Correlazione: `cor(x, y, na.rm=F)`  $\quad =: \text{cov}(x, y) / (\sigma_x \sigma_y)$

## DISTRIBUZIONI

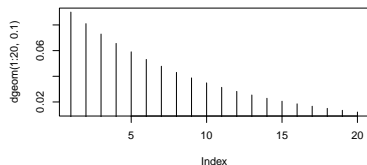
- In R, le funzioni relative alle distribuzioni sono 4 per ciascuna distribuzione:
  - generazione di numeri casuali, prefisso `r`
  - densità di probabilità (PDF), prefisso `d`
  - probabilità cumulata (CDF), prefisso `p`
  - quantile ( $CDF^{-1}$ ), prefisso `q`
- Ai prefissi vanno aggiunti i nomi delle distribuzioni: `unif`, `norm`, `t`, `f`, `chisq`, `pois`, `binom`, `geom`, `gamma`, `weibull`, ...
- Ad esempio:
  - `rnorm(n, 0, 1)` genera  $n$  campioni normali standard  $N(0, 1)$
  - `pt(t, n)` calcola la CDF del valore  $t$  da una distribuzione  $t_n$
  - `qf(p, n1, n2)` calcola il quantile per la probabilità  $p$  su una  $\chi_{n_1, n_2}^2$



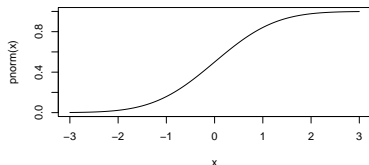
# STATISTICA DESCRITTIVA

## ESEMPIO

```
plot(dgeom(1:20, 0.1), typ="h")
```



```
curve(pnorm(x), xlim=c(-3, 3))
```

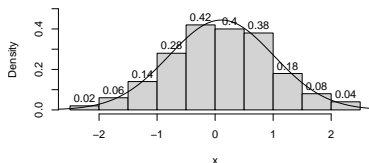


# STATISTICA DESCRITTIVA

## ESEMPIO

```
set.seed(1)
x <- rnorm(100)
m <- mean(x); s <- sd(x)
hist(x, prob=T, ylim=c(0, 0.5), labels=T)
curve(dnorm(x, mean=m, sd=s), xlim=c(-3, 3), add=T)
```

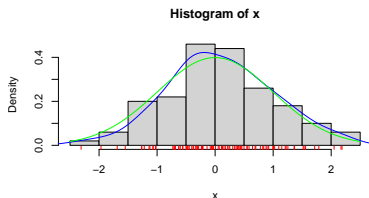
Histogram of x



# STATISTICA DESCRITTIVA

## KERNEL DENSITY ESTIMATE (KDE)

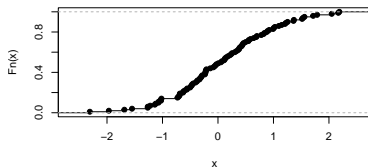
```
set.seed(123)
x <- rnorm(100)
hist(x, prob=T)
rug(x, col="red")
lines(density(x), col="blue")
curve(dnorm(x), add=T, col="green")
```



## KERNEL DENSITY ESTIMATE (KDE)

```
plot(ecdf(x))
```

ecdf(x)



# STATISTICA INFERENZIALE

## TEST DI IPOTESI

- Un *test statistico* prevede sempre una coppia di ipotesi
- L'ipotesi nulla  $H_0$  è sempre l'ipotesi di non-significatività
- L'ipotesi alternativa  $H_1$  suppone che un effetto sia statisticamente significativo, cioè maggiore della varianza tipica del sistema

$$H_0 : \quad \mu_1 = \mu_2$$

$$H_1 : \quad \mu_1 \neq \mu_2$$

## MATRICE DI CONFUSIONE

$H_0$	Accettata	Rifiutata
Vera	OK	Errore Tipo I
Falsa	Errore Tipo II	OK

## PROBABILITÀ DI ERRORE

- $\alpha$  è la probabilità di rifiutare  $H_0$  quando  $H_0$  è vera (tipo-I)
- $\beta$  è la probabilità di accettare  $H_0$  quando  $H_0$  è falsa (tipo-II)
- $P = 1 - \beta$  è la **potenza** di un test

## SE FOSSE UN ALLARME ANTI-INTRUSIONE:

- $\alpha$  è la probabilità di falso allarme
- $\beta$  è la probabilità di un mancato allarme
- $P$  è l'affidabilità del sistema (probabilità che suoni quando deve)

## ESEMPIO ZERO

- Abbiamo due campioni  $y_1$  e  $y_2$  di  $n_1$  e  $n_2$  osservazioni
- Le due medie campionarie  $\bar{y}_1$  e  $\bar{y}_2$  sono sufficientemente simili da chiederci se vengano dalla stessa popolazione o no
- Nel primo caso, il valore atteso della popolazione del primo campione è uguale al valore atteso della popolazione del secondo:  $H_0 : \mu_1 = \mu_2$
- Nel secondo caso invece  $H_1 : \mu_1 \neq \mu_2$
- Questo problema corrisponde all'originale formulazione del test di Student di William Gosset, detto *Student*

## TEST DI STUDENT, O T-TEST

- Risulta che:

$$t_0 = \frac{\bar{y}_1 - \bar{y}_2}{S_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \sim t_{n_1+n_2-1}, \text{ dove } S_p = \frac{(n_1 - 1)S_1 + (n_2 - 1)S_2}{n_1 + n_2 - 2}$$

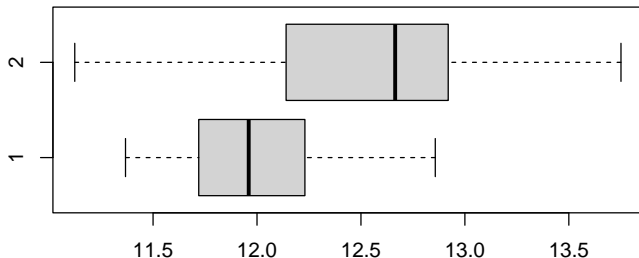
- dato che conosciamo la distribuzione di  $t_0$ , possiamo calcolare la probabilità di riscontrare un valore uguale o superiore a  $t_0$
- tanto più bassa è tale probabilità, detta  $p$ -value, tanto più è forte  $H_1$
- il  $p$ -value corrisponde alla probabilità di un errore di tipo-I,  $\alpha$



# STATISTICA INFERENZIALE

## T-TEST IN R

```
set.seed(123)  
y1 <- rnorm(10, 12, 0.5); y2 <- rnorm(12, 12.5, 0.7)  
boxplot(y1, y2, horizontal=T)
```



# STATISTICA INFERENZIALE

## T-TEST IN R

- per prima cosa si verifica l'**omoschedasticità**:

```
(vt <- var.test(y1, y2))
```

```
##
```

```
## F test to compare two variances
```

```
##
```

```
## data: y1 and y2
```

```
## F = 0.4531, num df = 9, denom df = 11, p-value = 0.2446
```

```
## alternative hypothesis: true ratio of variances is not equal to 1
```

```
## 95 percent confidence interval:
```

```
## 0.1262847 1.7725487
```

```
## sample estimates:
```

```
## ratio of variances
```

```
## 0.4530969
```

# STATISTICA INFERENZIALE

## T-TEST IN R

- Poi si effettua il T-test appropriato:

```
t.test(y1, y2, var.equal=(vt$p.value>0.05))  
  
##  
## Two Sample t-test  
##  
## data: y1 and y2  
## t = -1.9339, df = 20, p-value = 0.0674  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -1.05880157 0.04004476  
## sample estimates:  
## mean of x mean of y  
## 12.03731 12.54669
```

## OSSERVAZIONE

- In R, alcune funzioni possono ritornare degli *oggetti*
- Un oggetto raggruppa uno o più *attributi*, visualizzabili col comando `names()` e accessibili con la notazione `$`:

```
names(vt)
```

```
## [1] "statistic"      "parameter"      "p.value"        "conf.int"  
## [6] "null.value"     "alternative"     "method"         "data.name"
```

```
vt$p.value
```

```
## [1] 0.2446474
```