Introduzione a R

Test per le organizzaioni

Ottavia M. Epifania
ottavia.epifania@unipd.it

Margherita Calderan
margherita.calderan@unipd.it

Università di Padova

Introduzione

R è un linguaggio di programmazione fortemente votato alla statistica, gestione di dati e visualizzazione.

E' nato nel 1993 da **Ross Ihaka** e **Robert Gentleman**.

E' un software completamente **open-source** e **gratuito** in continua evoluzione e cambiamento.

Perchè R?

Un software si definisce open-source quando il **codice sorgente è disponibile** a tutti per essere **modificato**, **aggiornato** e **controllato**.

R è **sia open-source che gratuito** e vanta una community estremamente attiva, come spesso accade con tutti i progetti open-source e in generale i linguaggi di programmazione.

Il principale "concorrente" di R è sicuramente **Python** che offre un ambiente altrettanto potente, sviluppato e attivo.

Non è facile (e forse non è possibile) capire quale sia il migliore.

In ogni caso, una volta imparato R, imparare Python sarà molto semplice.

Nell'ambito della statistica ci sono vari software non open-source ed a pagamento come:

- Statistica
- SPSS
- STATA
- SAS

Sono degli ottimi software ma:

- Non forniscono conoscenze trasversali
- Siete legati ad uno specifico ambiente
- Le licenze possono costare molto
- La community non è altrettanto attiva (non open-source)

Ci sono degli ottimo software open-source basati su R come:

- Jamovi
  - **pros**: si può accedere al codice R sottostante
  - **cons**: le funzioni sono comunque limitate, grafici, modelli complessi
- Jasp
  - **pros**: molti modelli anche avanzati
  - **cons**: non si può vedere il codice R

Imparare in linguaggio come R vi permette di conoscere uno strumento molto potente ma anche di imparare:

- Ragionare e risolvere problemi con il codice
- Trasferire quello che avete imparato ad altri linguaggi
- Essere sempre autonomi e non legati ad uno specifico ambiente
- Avere una skill realmente di valore

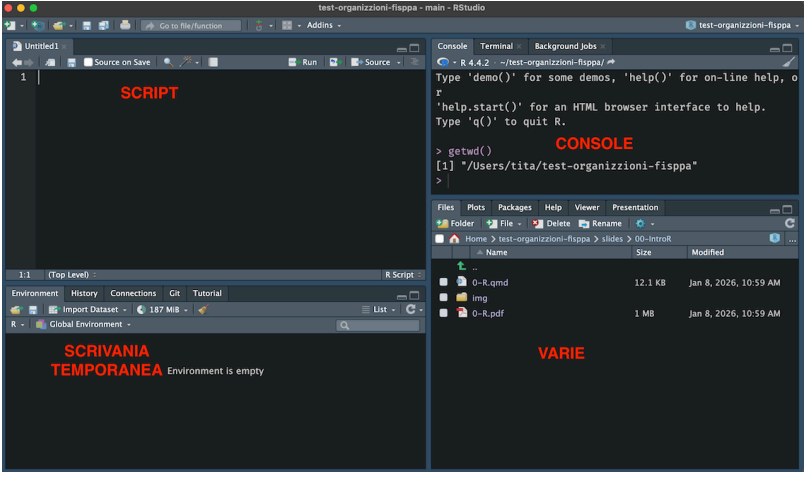
Installare R e R-Studio

Entrambi vanno installati separatamente e la procedura varia a seconda del proprio sistema operativo.

Se non l'avete già installato, seguite la procedura spiegata a questo link: https://posit.co/download/rstudio-desktop/

Se non volete/non riuscite nell'installazione, potete accedere ad R-studio attraverso il server: https://posit.cloud/ (collegatevi attraverso la mail unipd)

Come lavorare in R



Environment. La vostra scrivania quando lavorate in R. Contiene tutti gli oggetti (variabili) creati durante la sessione di lavoro.

Script. File di testo dove il codice viene salvato e puo essere lanciato in successione. Nello script è possibile combinare codice e commenti (#)

```
# assegno ad x il valore 30
x = 30
```

Working Directory. La posizione (cartella) sul vostro PC dove R sta lavorando e nella quale R si aspetta di trovare i vostri file, se non specificato altrimenti.

console vs. script

Console

I comandi nella console vengono eseguiti e non salvati
Per eseguire il comando $\rightarrow$ `Invio`
L'output è immediato ed appare nella console

console vs. script

Console

I comandi nella console vengono eseguiti e non salvati
Per eseguire il comando → `Invio`
L'output è immediato ed appare nella console

Script

è possibile salvare gli script con tutti i comandi salvati
Per eseguire il comando → `Ctrl + Invio` (`cmd + Enter`)
L'output è restituito nella console

Working Directory

Dove sta lavorando R ?

```
getwd()
```

```
[1] "/Users/tita/test-organizzioni-fisppa/slides/01-IntroR"
```

(Se voglio cambiare la working directory, posso utilizzare il comando `setwd`)

```
setwd('/Users/tita/Desktop')
```

Path Assoluto

```
|- Users
    |
    |- tita
        |
        |- test-organizzioni-fisppa
            |
            |-slides
                |
                |-01-IntroR
```

Io sto lavorando dentro la cartella `01-IntroR`.

Dato che sto lavorando dentro la cartella, se voglio caricare un file che si trova dentro questa cartella posso scrivere semplicemente il nome del file tra virgolette, ed utilizzare per esempio la funzione `read.csv`:

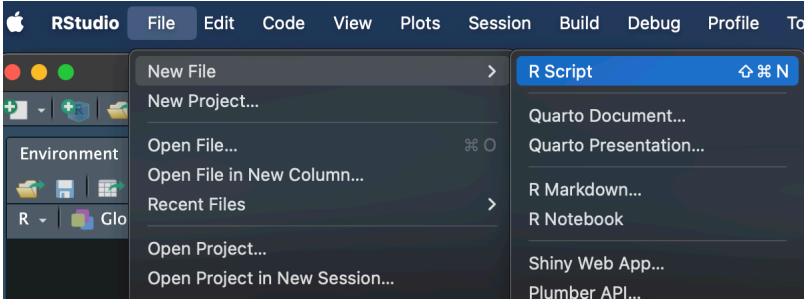
```
data = read.csv("prova.csv")
```

Se il file si trova in una sottocartella (es. data), devo aggiungere quest'informazione al path:

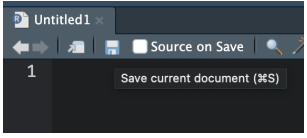
```
data = read.csv("data/prova.csv")
```

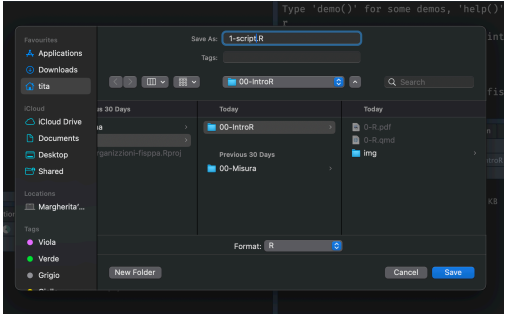
N.B. Per specificare il path scrivere / non \

Creiamo uno script:



Salviamo:


Untitled1
Source on Save
1
Save current document (⌘S)


Type 'demo()' for some demos, 'help()'
Save As: 1-script.R
Tags:
Favourites
Applications
Downloads
tita
00-IntroR
Search
iCloud
iCloud Drive
Documents
Desktop
Shared
Today
Previous 30 Days
00-Misura
0-R.pdf
0-R.qmd
img
Locations
Tags
Viola
Verde
Grigio
Format: R
New Folder
Cancel
Save

Oggetti

Tutto quello che possiamo creare in R viene definito oggetto (e.g., numeri, vettori, matrici, funzioni).

```
numero = 4; numero
```

```
[1] 4
```

```
vettore = c(1,2,3,4); vettore
```

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

```
matrice = matrix(nrow = 2, ncol = 2, data = vettore); matrice
```

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

Creare/nominare oggetti

Gli oggetti si possono creare e tramite il comando <- oppure =

```
x1 = 3 # nome = oggetto
x1
```

```
[1] 3
```

```
x2 =3 # nome =oggetto
x2
```

```
[1] 3
```

```
x1 == x2 # i due oggetti sono identici?
```

```
[1] TRUE
```

Principali tipi di dato

- **character**: Stringhe di caratteri i cui valori alfanumerici vengono delimitati dalle doppie vigolette "Hello world!" o virgolette singole 'Hello world!'
- **numeri**: interi e/o decimali

```
A = 12.8 # variabile numerica
nome = "giorgio"  # variabile character
b1 = c(12, 0.3, 5, 778.3) # vettore numerico
Nomi3 = c("giorgio", "ugo", "anna") # vettore character
```

Regole sulla denominazione di oggetti

- Deve iniziare con una lettera e può contenere lettere, numeri, underscore (_), o punti (.).
- Potrebbe anche iniziare con un punto (.) ma in tal caso non può essere seguito da un numero.

```
.3 = 3
```

```
Error in `0.3 = 3`:
! invalid (do_set) left-hand side to assignment
```

```
.x = 3
```

- Non deve contenere caratteri speciali come #, &, $, ?, etc.
- Non deve essere una parola riservata ovvero quelle parole che sono utilizzate da R con un significato speciale (?reserved).

```
TRUE = 3
```

```
Error in `TRUE = 3`:
! invalid (do_set) left-hand side to assignment
```

```
if = 3
```

```
Error in parse(text = input): <text>:1:4: unexpected '='
1: if =
       ^
```

Ci sono alcuni nomi che non sono proibiti ma sono sconsigliati

```
T
```

```
[1] TRUE
```

```
F
```

```
[1] FALSE
```

```
sum(2,3)
```

```
[1] 5
```

```
sum = 4
```

Tra i diversi linguaggi, le *convenzioni di denominazione* per i nomi di variabili più lunghi e composti da più parole privilegiano `snake_case` (ad esempio, "**my_data**") o `camelCase` (ad esempio, "**myData**"), e **abbreviazioni** dove appropriato (ad esempio, "**unipdData**" meglio di "university_of_padova_dataset").

R è case-sensitive!

```
Nome = "Margherita"

nome = "margherita"

Nome
```

```
[1] "Margherita"
```

```
nome
```

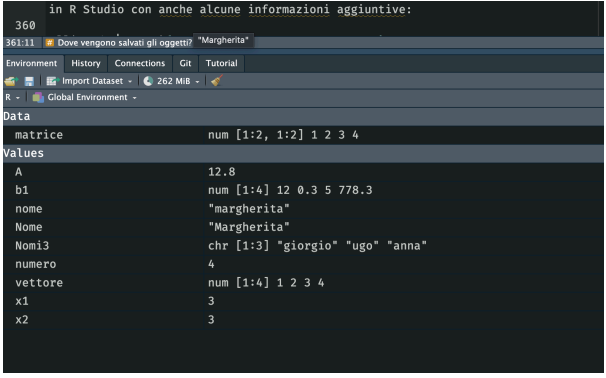
```
[1] "margherita"
```

```
Nome == nome
```

```
[1] FALSE
```

Dove vengono salvati gli oggetti?

Di default gli oggetti sono creati nel **global environment** accessibile con `ls()` o visibile in R Studio con anche alcune informazioni aggiuntive:



Possiamo eliminare un oggetto presente nel nostro enviroment attraverso il comando `rm("nomeoggetto")`.

E' possibile anche pulire completamente/svuotare il nostro enrivoment attraverso il comando `rm(list = ls())`.

Funzioni

Tutto quello che facciamo in R è chiamare **funzioni** su oggetti.

Le funzioni ci permettono di **creare** e **modificare** oggetti.

```
vettore
```

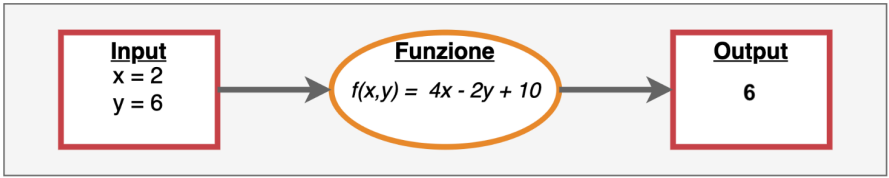
```
[1] 1 2 3 4
```

```
mean(x = vettore)
```

```
[1] 2.5
```

Funzioni

Possiamo pensare alle funzioni in R in modo analogo alle classiche funzioni matematiche. Dati dei valori in **input**, le funzioni eseguono dei specifici calcoli e restituiscono in **output** il risultato ottenuto.



Argomenti

Gli argomenti delle funzioni sono quelli che da *utenti* dobbiamo conoscere ed impostare nel modo corretto per fare in modo che la funzioni faccia quello per cui è stata pensata. Nell'esempio precedente l'unico argomento era `x`. Vediamo invece l'`help` della funzione `mean()`.

Per impostare questi argomenti ci sono 2 regole:

- l'ordine non conta SE DEFINISCO NOME DELL'ARGOMENTO con `x = vettore`, `na.rm = TRUE`, etc.
- l'ordine conta SE NON DEFINISCO IL NOME DELL'ARGOMENTO. Posso quindi omettere `argomento = valore` ma devo rispettare l'ordine con cui è stata scritta la funzione.

In questo caso proviamo ad usare la funzione `mean()`:

```
myvec = rnorm(n = 1000, mean = 1, sd = 1)
# x definito,trim e na.rm non definito, quindi uguali a?
mean(x = myvec)
```

```
[1] 1.035053
```

```
?mean
```

In questo caso proviamo ad usare la funzione `mean()`:

```
mean(x = myvec, na.rm = TRUE) # x definito, na.rm definito
```

```
[1] 1.035053
```

```
mean(myvec, TRUE) # cosa succede?
```

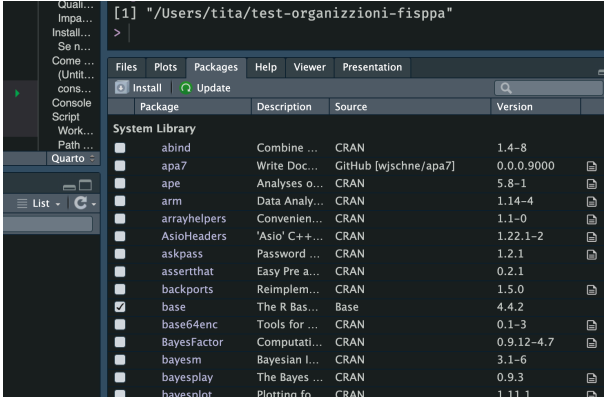
```
Error in `mean.default()`:
! 'trim' must be numeric of length one
```

Packages

In R è possibile installare e caricare pacchetti aggiuntivi che non fanno altro che rendere disponibili librerie di funzioni create da altri utenti. Per utilizzare un pacchetto:

- Installare il pacchetto con `install.packages("nomepacchetto")`
- Caricare il pacchetto con `library(nomepacchetto)`
- Accedere ad una funzione senza caricare il pachetto `nomepacchetto::nomefunzione()`. Utile se serve solo una funzione o ci sono conflitti)

Packages


[1] "/Users/tita/test-organizzioni-fisppa"
>
Files
Plots
Packages
Help
Viewer
Presentation
Install
Update
Package
Description
Source
Version
System Library
abind Combine ... CRAN 1.4-8
apa7 Write Doc... GitHub [wjschne/apa7] 0.0.0.9000
ape Analyses o... CRAN 5.8-1
arm Data Analy... CRAN 1.14-4
arrayhelpers Convenien... CRAN 1.1-0
AsioHeaders 'Asio' C++... CRAN 1.22.1-2
askpass Password ... CRAN 1.2.1
assertthat Easy Pre a... CRAN 0.2.1
backports Reimplem... CRAN 1.5.0
base The R Bas... Base 4.4.2
base64enc Tools for ... CRAN 0.1-3
BayesFactor Computati... CRAN 0.9.12-4.7
bayesm Bayesian I... CRAN 3.1-6
bayesplay The Bayes ... CRAN 0.9.3
Console
Script
Work...
Path ...
Quarto
List

Operatori Matematici

Funzione	Cosa fa?	Esempio	Risultato
+	addizione	`5.4 + 6.1`	`11.5`
-	sottrazione	`9 - 4.3`	`4.7`
*	moltiplicazione	`7 * 1.4`	`9.8`
/	divisione	`9/3`	`3`
%%	resto	`9%%2`	`1`
^	potenza	`15 ^ 2`	`225`

Funzione	Cosa fa?	Esempio	Risultato
`abs`	valore assoluto	`abs(-8)`	8
`sqrt`	radice quadrata	`sqrt(225)`	15
`exp`	funzione esponenziale	`exp(0)`	1
`log`	logaritmo naturale	`log(1)`	0
`round`	arrotondamento, intero	`round(1.738)`	2
`round`	arrotondamento	`round(1.738, 2)`	1.74

Operazioni Matematiche

L'ordine delle operazioni in R segue le regole della matematica.

Esempi

```
# Senza parentesi
1 + 2 * 3
```

```
[1] 7
```

```
# Con le parentesi
(1 + 2) * 3
```

```
[1] 9
```

Operatori Relazionali

In R è possibile valutare se una data relazione è vera o falsa. R valuterà le proposizioni e ci restituirà il valore **TRUE** se la proposizione è vera oppure **FALSE** se la proposizione è falsa.

Funzione	Nome	Esempio	Risulato
==	uguale	30 == 30	TRUE
!=	diverso	30 != 30	FALSE
>/>=	maggiore/o	30 > 10	TRUE
	uguale	30 >= 10	TRUE
</<=	minore/o uguale	30 < 10	FALSE
		10 <= 10	TRUE
%in%	inclusione	10%in%c(1,2,10)	TRUE

Non vale solo per i numeri!

```
Nome = "Margherita"

nome = "margherita"

Nome == nome
```

```
[1] FALSE
```

PS. Ricordatevi che = è diverso da ==

Operatori Logici

In R è possibile congiungere più relazioni per valutare una desiderata proposizione.

```
x = 30 #Assegnamo a x il valore 30.
```

Funzione	Nome	Esempio	Risulato
&	Congiunzione	x>25 & x<60	TRUE
|	Disgiunzione Inclusiva	x>25 | x>60	TRUE
!	Negazione	!(x<18)	TRUE

```r
x = 30 #Assegnamo a x il valore 30.
x
```

```
[1] 30
```

```r
x>25 & x>60
```

```
[1] FALSE
```

```r
x>25 | x>60
```

```
[1] TRUE
```

```r
!(x>18)
```

```
[1] FALSE
```

R ed errori

In R gli errori sono:

- inevitabili
- parte del codice stesso
- educativi

Ci sono diversi livelli di **allerta** quando scriviamo codice:

- `messaggi`: la funzione ci restituisce qualcosa che è utile sapere, ma tutto liscio
- `warnings`: la funzione ci informa di qualcosa di *potenzialmente* problematico, ma (circa) tutto liscio
- `error`: la funzione non solo ci informa di un **errore** ma le operazioni richieste non sono state eseguite

Come risolvere?

- Capire il messaggio
- Leggere la documentazione della funzione
- Cercare il messaggio su internet
- Chiedere aiuto nei forum dedicati

- Ogni funzione ha una pagina di documentazione accessibile con `?nomefunzione`, `??nomefunzione` oppure `help(nomefunzione)`
- Possiamo cercare anche la documentazione del pacchetto
- Possiamo cercare su internet il nome della funzione o l'eventuale messaggio che riceviamo

Facciamo un po' di pratica!

Aprite e tenete aperto questo link:
https://etherpad.wikimedia.org/p/test-organizzazioni