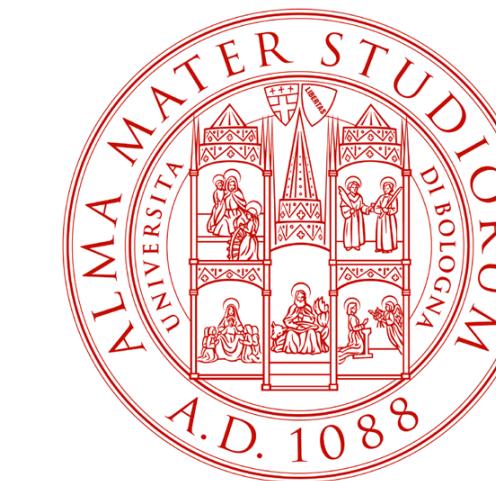


Attività di allineamento Laboratorio Informatico-Statistico

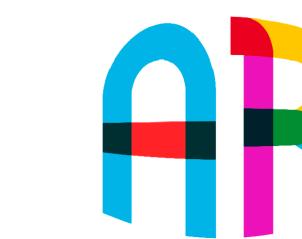


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Statistica Economia e Impresa

Dipartimento di Scienze Statistiche Paolo Fortunati

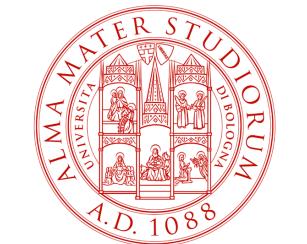
Anno Accademico 2021/2022



Who am I?

raffaele.anselmo2@unibo.it

raffaleanselmo@gmail.com



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Laurea in **Scienze Statistiche**,
curriculum Economia e Impresa



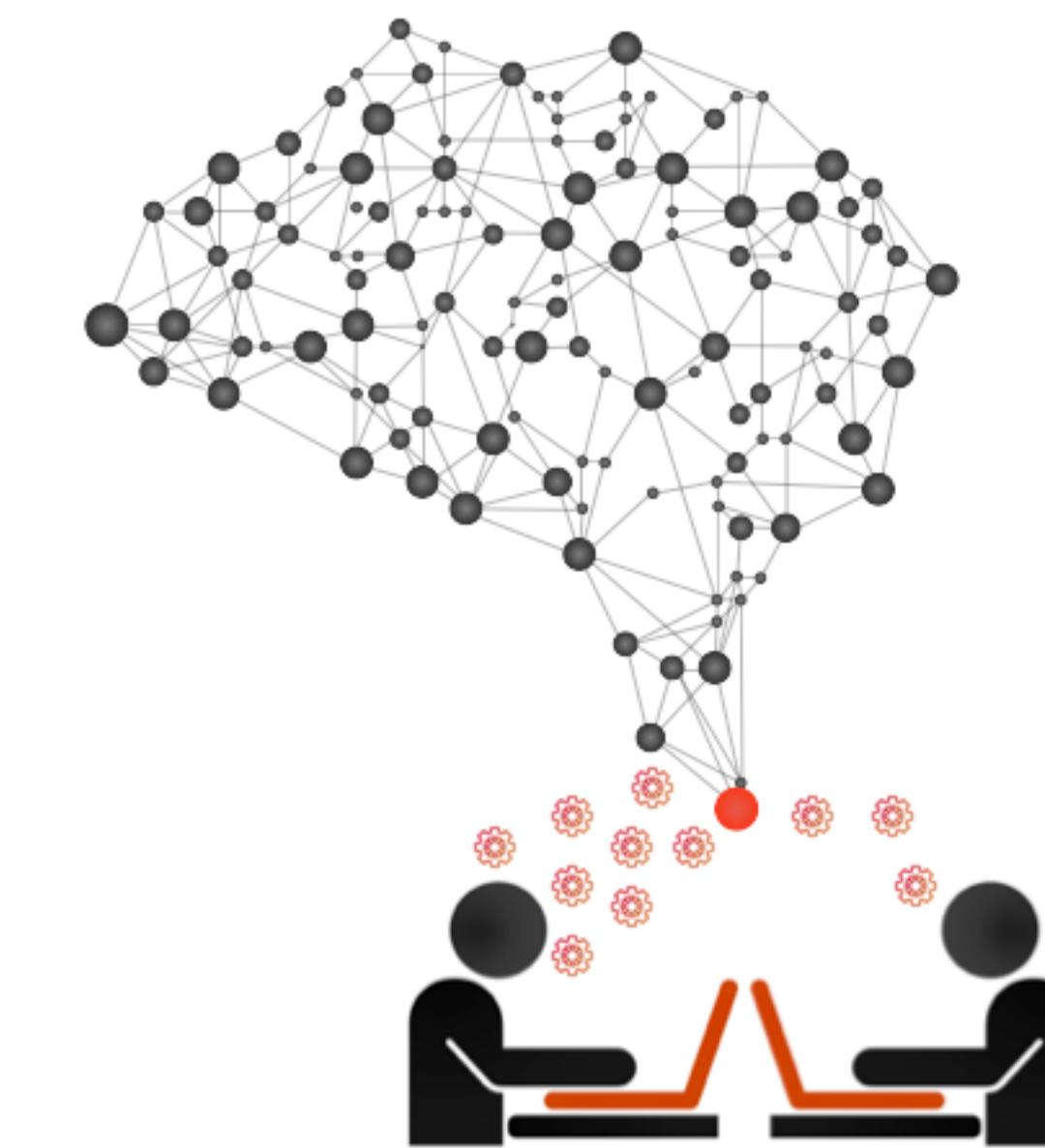
Master Degree in
Data Science



Tensorflow
Certified developer



Senior Data Scientist
@CRIF



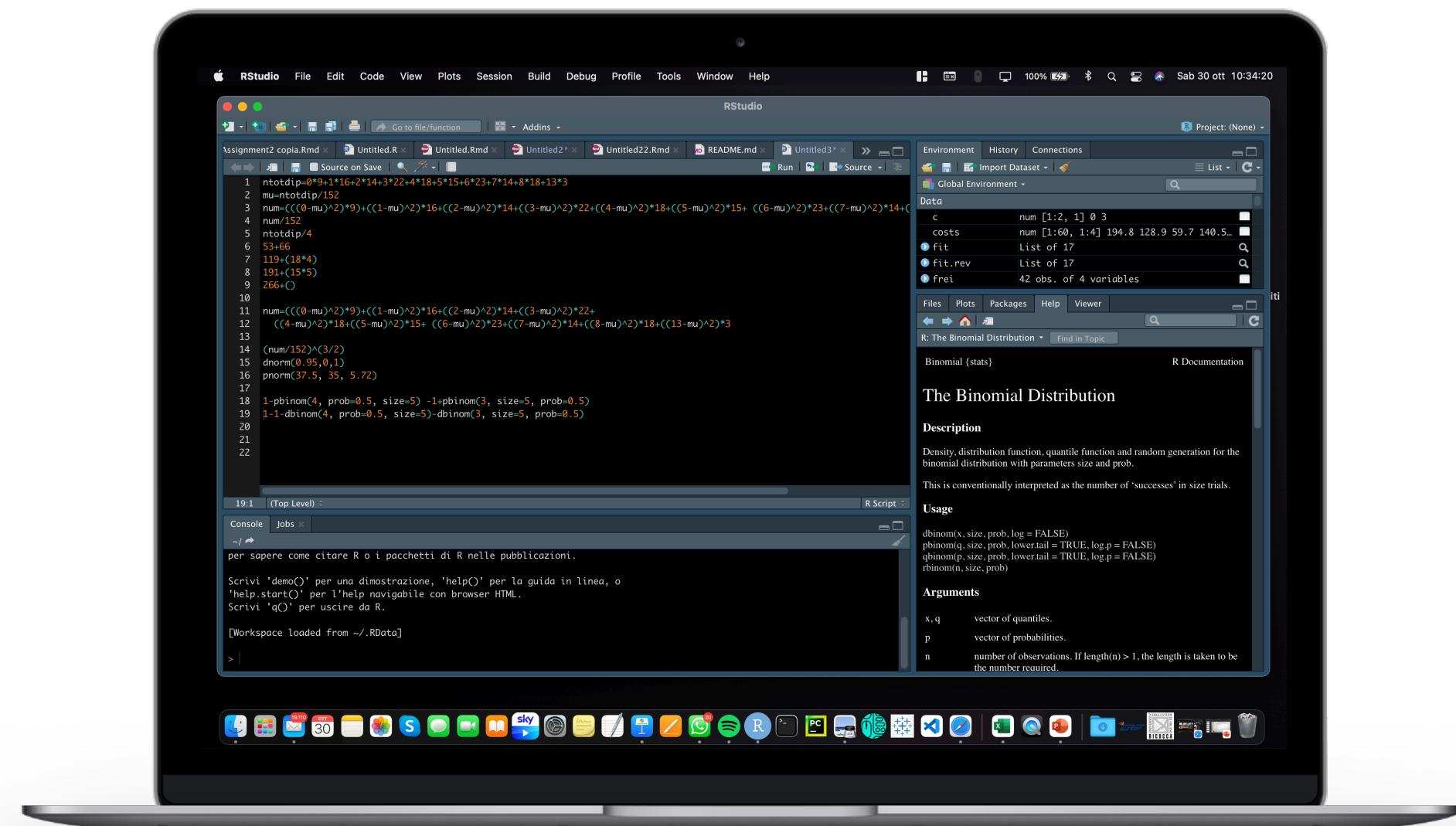
What is this course about?

21
3² 4⁷ 5
6⁹

Il mondo *Data* e gli strumenti per gestirlo

R

Linguaggio di programmazione per analisi statistiche



Why this course?

Apprendere
competenze utili

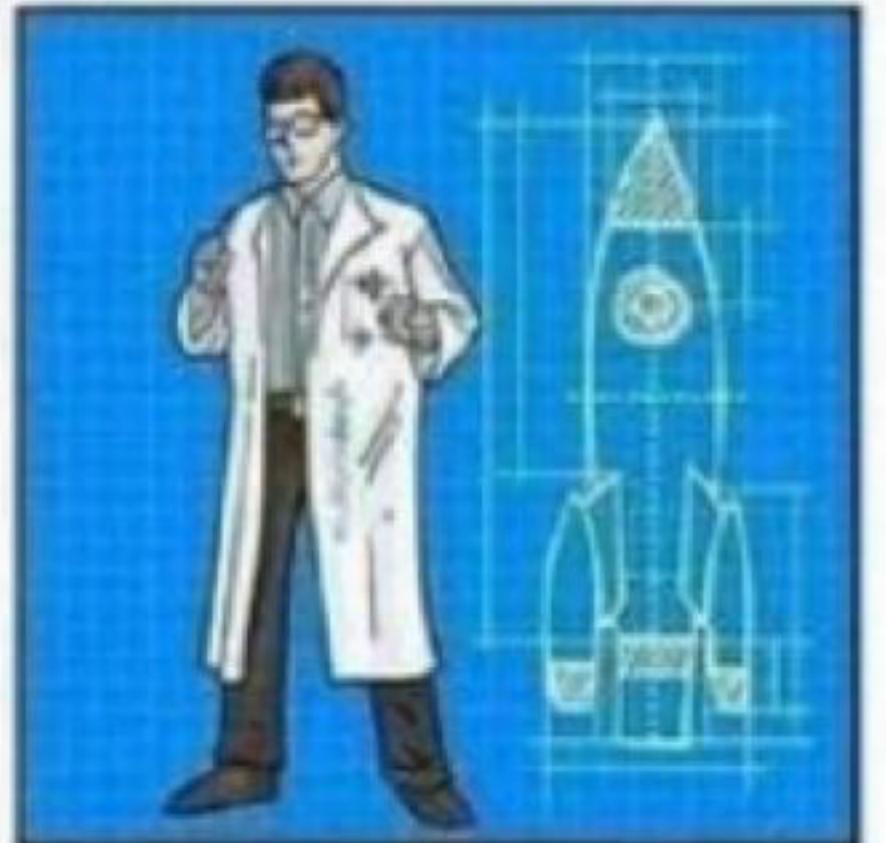
IMPARARE A COPIARE!

Data Scientist

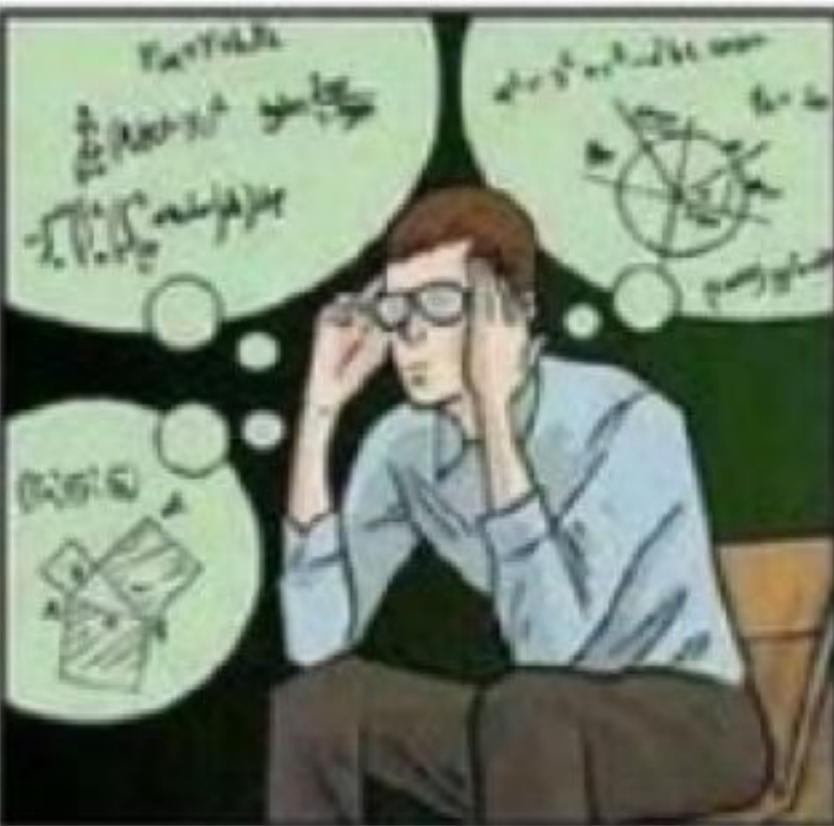
Cosa le persone
pensano che faccia



Cosa i miei genitori
pensano che faccia



Cosa io penso di fare



Cosa realmente faccio



Why this course?

Apprendere
competenze utili

IMPARARE A COPIARE!



How is it structured?

1 Intro

Nozioni di base sulla
programmazione ed R

2 Data Exploration

Matrici,
Dataframes,
statistiche e grafici

3 Next Steps

Funzioni,
Approfondimenti e
library utili

Introduzione

First Steps

Download

Google

Download R 

<https://cran.r-project.org/bin/windows/base/>

Google

Download RStudio 

<https://www.rstudio.com/products/rstudio/download/>

RStudio Desktop <small>Open Source License</small> Free	RStudio Desktop Pro <small>Commercial License</small> \$995 /year	RStudio Server <small>Open Source License</small> Free	RStudio Workbench <small>Commercial License</small> \$4,975 /year (5 Named Users)
DOWNLOAD <small>Learn more</small>	BUY <small>Learn more</small>	DOWNLOAD <small>Learn more</small>	BUY <small>Evaluation Learn more</small>

First Steps

La finestra RStudio

Code Editor

The screenshot displays the RStudio interface with four main sections highlighted by orange boxes:

- Code Editor:** The leftmost section shows an R script with code for model development, including logistic regression and decision trees, and ROC curve plotting.
- Console:** The bottom-left section shows the R console output, which includes the command to generate the ROC curve and the resulting AUROC value.
- Workspace:** The top-right section shows the Global Environment pane, listing various objects such as datasets and models.
- Plot/Help:** The bottom-right section shows an ROC Curve plot with the AUROC value displayed as 0.8155.

```
766 #MODELLO LOGIT
767 logit_model<- glm(TARGET ~ NUM_SEND_PREV + OPEN_RATE_PREV + CLICK_RATE_PREV + SEND_WEEKDAY + ID_NEG + TYP_CLI_FID + STATUS_FID + NUM_FIDs +
768   family = binomial(link = "logit"),
769   data = training_set)
770
771 summary(logit_model)
772
773 logit_prob <- plogis(predict(logit_model, test_set))
774
775 logit_pred <- ifelse(logit_prob > 0.5, 1, 0)
776
777 #ROC Curve
778 ROC_logit <- plotROC(test_set$TARGET, logit_prob)
779
780 #Confusion Matrix and Statistics
781 logit_pred <- as.factor(logit_pred)
782 CONFUSION_MATRIX_LOG <- confusionMatrix(test_set$TARGET, logit_pred)
783 CONFUSION_MATRIX_LOG
784
785
786 #TREE MODEL
787 library(tree)
788 tree_model<- tree(TARGET ~ NUM_SEND_PREV + OPEN_RATE_PREV + CLICK_RATE_PREV + SEND_WEEKDAY + TYP_CLI_FID + STATUS_FID + NUM_FIDs + AGE_FID +
789   data = training_set)
790
791 summary(tree_model)
792 tree_prob <- plogis(predict(tree_model, test_set))
793 tree_prob <- tree_prob[,2]
794
795
796 # EMAIL ENGAGEMENT
797
798 confusionMatrix, precision, sensitivity, specificity
799
800 > #ROC Curve
801 > ROC_logit <- plotROC(test_set$TARGET, logit_prob)
802 > #campionamento stratificato secondo la variabile TARGET (sbilanciata)
803 > training_indices <- master_df$TARGET %>% createDataPartition(p=0.75, list = FALSE)
804 > ### TRAINING - TEST SET ####
805 > library(caret)
806 > set.seed(12345)
807 > #campionamento stratificato secondo la variabile TARGET (sbilanciata)
808 > training_indices <- master_df$TARGET %>% createDataPartition(p=0.75, list = FALSE)
809 > training_set <- master_df[training_indices, ]
810 > test_set <- master_df[-training_indices, ]
811 > #MODELLO LOGIT
812 > logit_model<- glm(TARGET ~ NUM_SEND_PREV + OPEN_RATE_PREV + CLICK_RATE_PREV + SEND_WEEKDAY + ID_NEG + TYP_CLI_FID + STATUS_FID + NUM_FIDs + AGE_FID +
813   D + W_PHONE + TYP_CLI_ACCOUNT + EMAIL_PROVIDER_CLEAN + REGION,
814   family = binomial(link = "logit"),
815   data = training_set)
816 > logit_prob <- plogis(predict(logit_model, test_set))
817 > logit_pred <- ifelse(logit_prob > 0.5, 1, 0)
818 > #ROC Curve
819 > ROC_logit <- plotROC(test_set$TARGET, logit_prob)
820 >
```

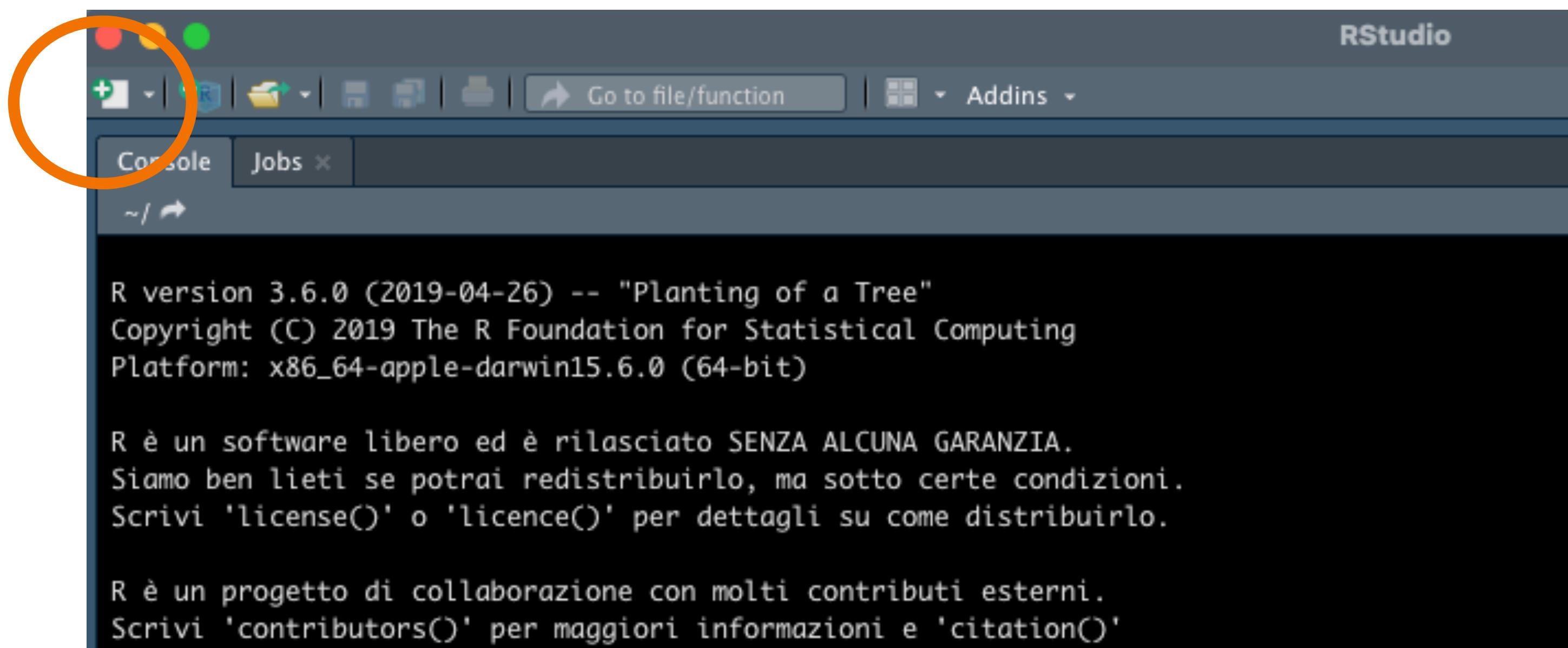
Console

Workspace

Plot/Help

First Steps

“Hello world”



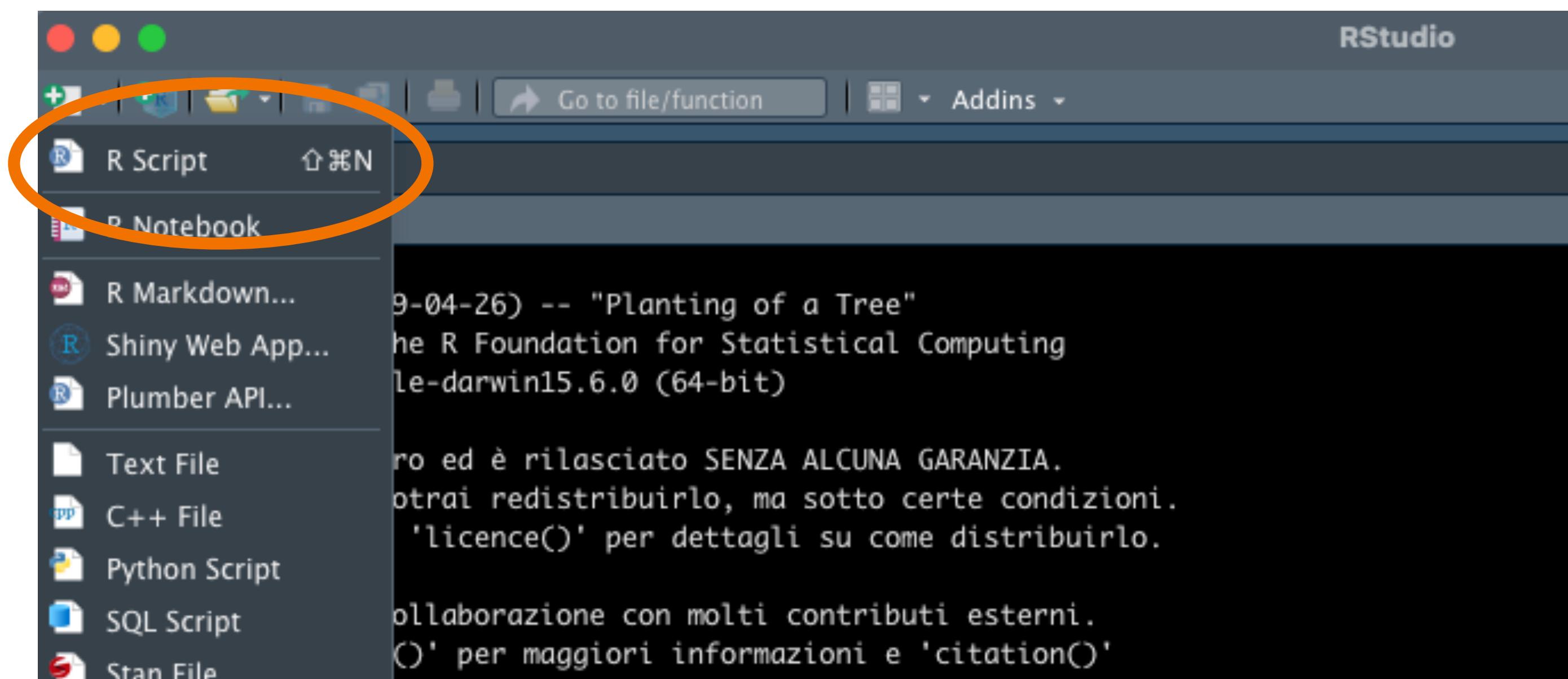
```
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R è un software libero ed è rilasciato SENZA ALCUNA GARANZIA.
Siamo ben lieti se potrai redistribuirlo, ma sotto certe condizioni.
Scrivi 'license()' o 'licence()' per dettagli su come distribuirlo.

R è un progetto di collaborazione con molti contributi esterni.
Scrivi 'contributors()' per maggiori informazioni e 'citation()'
```

First Steps

“Hello world”



First Steps

“Hello world”

The screenshot shows the RStudio interface with the following details:

- Script Editor (Top Left):** An untitled R script file named "Untitled1" is open. It contains the line of code: `print('Hello world')`. This line is highlighted with an orange oval.
- Console (Bottom Left):** The console output shows the results of running the script. It displays the message: "Scrivi 'demo()' per una dimostrazione, 'help()' per la guida in linea, o 'help.start()' per l'help navigabile con browser HTML. Scrivi 'q()' per uscire da R." followed by "[Workspace loaded from ~/.RData]".
- Console Output (Bottom Left):** The console also shows the command: `> print('Hello world')` and its result: [1] "Hello world". This line is highlighted with an orange oval.
- Environment Tab (Top Right):** The "Environment" tab is selected, showing the "Global Environment" pane which is currently empty.
- Files Tab (Bottom Right):** The "Files" tab is selected, showing the "List" pane which is currently empty.

First Steps

Gestione dei pacchetti

R contiene poche semplici funzioni. Gran parte delle funzioni più *potenti* sono rese disponibili tramite *librerie* (*packages*) direttamente installabili da R.

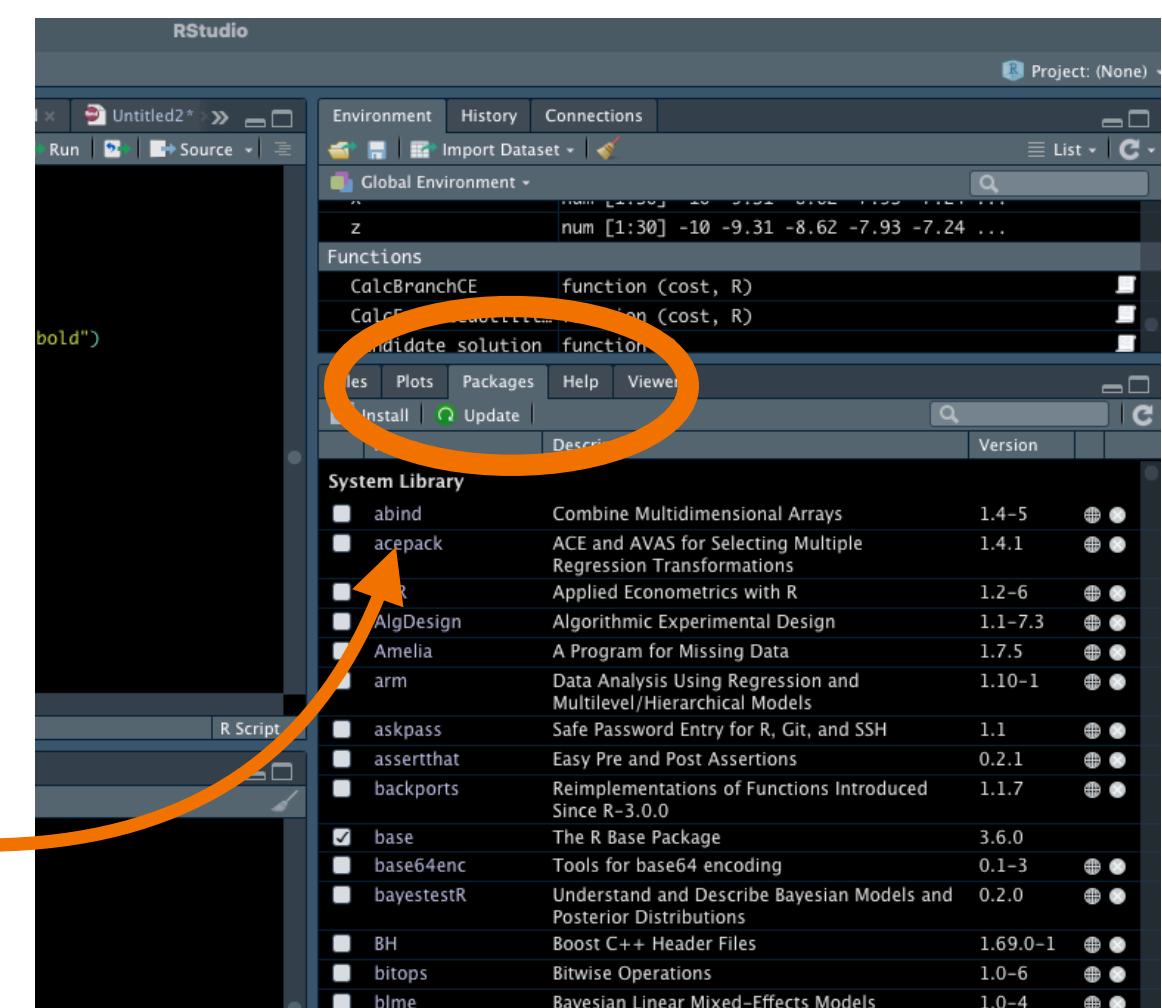
Per installare una libreria si utilizza la funzione:

```
>install.packages("nomelibreria")
```

Per utilizzare le funzioni di una libreria già installata è necessario importare la libreria nell'ambiente di lavoro (*workspace*) tramite la funzione:

```
>library(nomelibreria)
```

Alternativamente, i pacchetti possono essere direttamente gestiti tramite il tab *packages* in basso a destra



First Steps

I primi comandi

In R tutto ciò che esiste è un oggetto. Per assegnare qualcosa (il numero 5) ad un oggetto (a) si possono utilizzare gli operatori “=“ e “<-“

```
> a = 5
```

```
> a
```

```
[1] 5
```

```
> a <- 5
```

```
> a
```

```
[1] 5
```

Gli oggetti possono essere di diverso tipo. Per controllare la tipologia dell'oggetto si usa la funzione:

```
> class(a)
```

```
[1] "numeric"
```

First Steps

I primi comandi

Se il valore è compreso tra apici, questo sarà considerato di tipo *character*:

```
> a = '5'  
> class(a)  
[1] "character"
```

Nota bene che abbiamo assegnato il carattere ‘5’ all’oggetto a che precedentemente conteneva il valore numerico 5. In questo caso l’oggetto numerico 5 è stato sostituito con l’oggetto carattere ‘5’. Per vedere la lista degli oggetti in memoria (per oggetti si intendono sia le variabili che le funzioni) si utilizza la funzione:

```
> ls()  
[1] "a"
```

Per rimuovere un oggetto si usa la funzione:

```
> rm('a')  
> ls()  
character(0)
```

First Steps

I primi comandi

Le funzioni vengono utilizzate nella seguente forma:

```
> nomefunzione(arg_1, arg_2, ..., arg_n)
```

Dove arg_i sono gli argomenti della funzione.

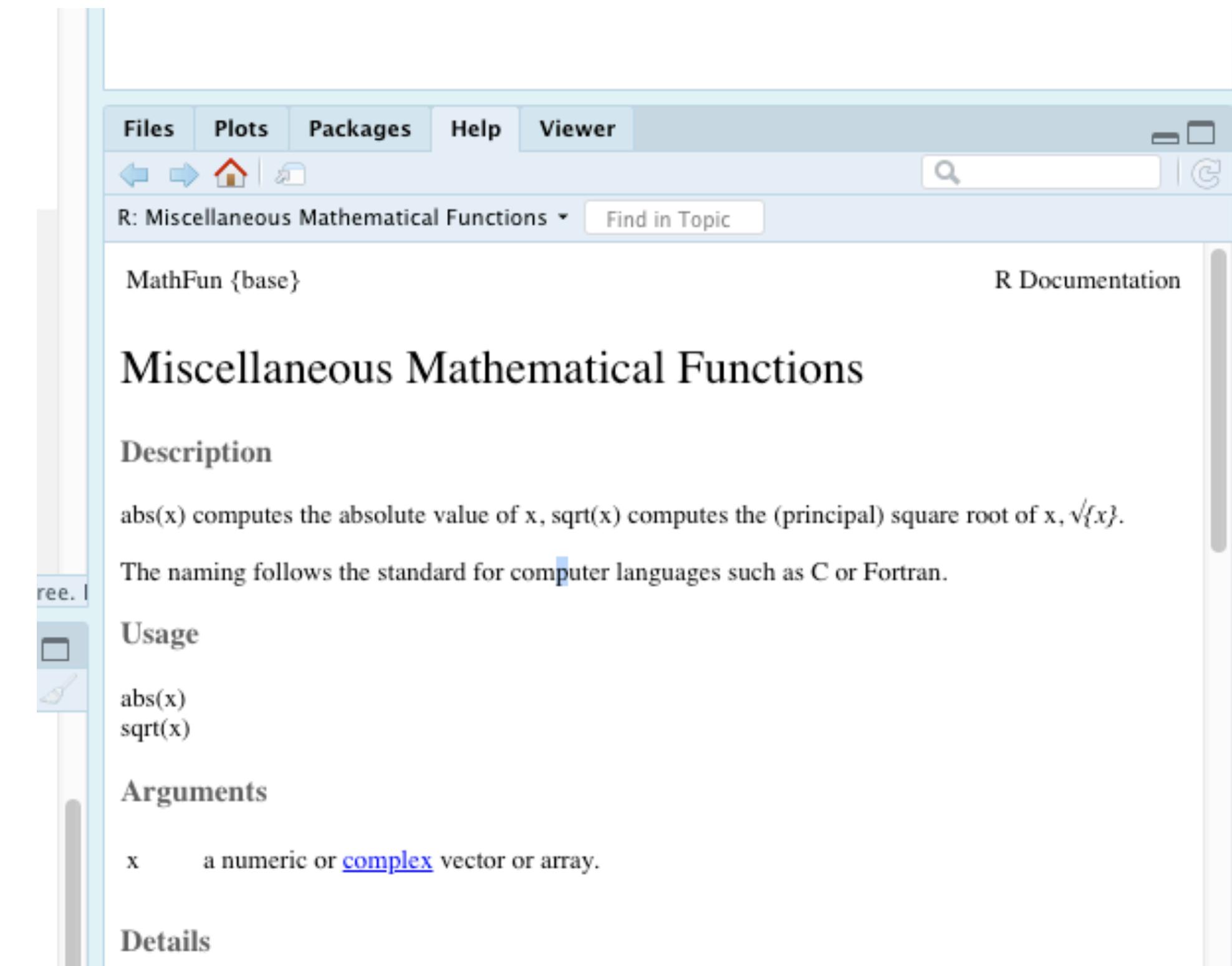
Ogni funzione è accompagnata da una documentazione racchiusa nel comando *help*. Ad esempio per conoscere la funzione *sqrt* utilizziamo il comando:

```
> help(sqrt)
```

Oppure:

```
> ?sqrt
```

L'output dell'help appare nel tab in basso a destra



First Steps

Le prime operazioni

La console può essere utilizzata come una vera e propria calcolatrice:

```
> 3+2*10/5-6
```

```
[1] 1
```

R segue l'ordine logico delle operazioni. Le operazioni di base sono:

- + : addizione
- - : sottrazione
- * : moltiplicazione
- / : divisione
- ^ : elevamento a potenza

First Steps

Le prime operazioni

Oltre agli operatori matematici, vi sono gli operatori logici:

- > : maggiore
- < : minore
- >= : maggiore o uguale
- <= : minore o uguale
- == : identico
- != : diverso (! Indica la negazione)
- & : intersezione (e)
- | : unione (o)

L'output di un operatore logico è a sua volta un valore logico, ovvero *TRUE*, *FALSE* e *NA*, che indica "risposta non disponibile".

```
> 3>2  
[1] TRUE  
  
> 3<2  
[1] FALSE  
  
> A=7  
[1] TRUE  
  
> A==7  
[1] TRUE  
  
> A!=3  
[1] TRUE
```

First Steps

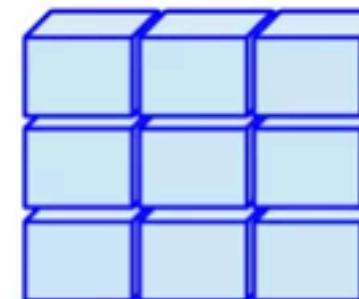
Gli oggetti

Variabili



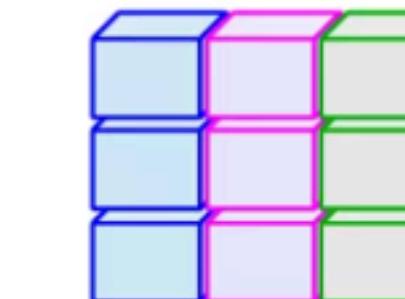
Dato in forma atomica

Matrici



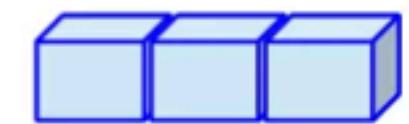
Vettori bidimensionali

Dataframe



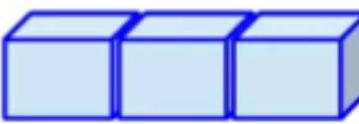
Matrici con vettori
eterogenei per tipo

Vettori



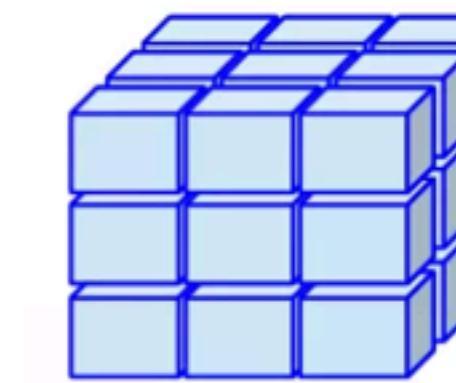
Insieme lineare di
elementi omogenei per
tipologia

Fattori



Classifica o suddivide in
livelli gli elementi di un
altro vettore

Array



Matrici n-dimensionali

First Steps

Gli oggetti - Variabili e Vettori

Le variabili sono oggetti atomici come quelli già visti in precedenza:

```
> a = 5
```

```
> a
```

Un vettore è invece un insieme di dati omogenei. Un vettore può essere creato utilizzando la funzione `c`, che combina gli argomenti al suo interno:

```
> vec.1 = c(18,2,3,6,6,4,3)
```

Se i vettori contengono valori numerici si possono applicare le funzioni statistiche come:

```
> min(vec.1) #valore minimo
```

```
> max(vec.1) #valore minimo
```

```
> length(vec.1) #numero di valori
```

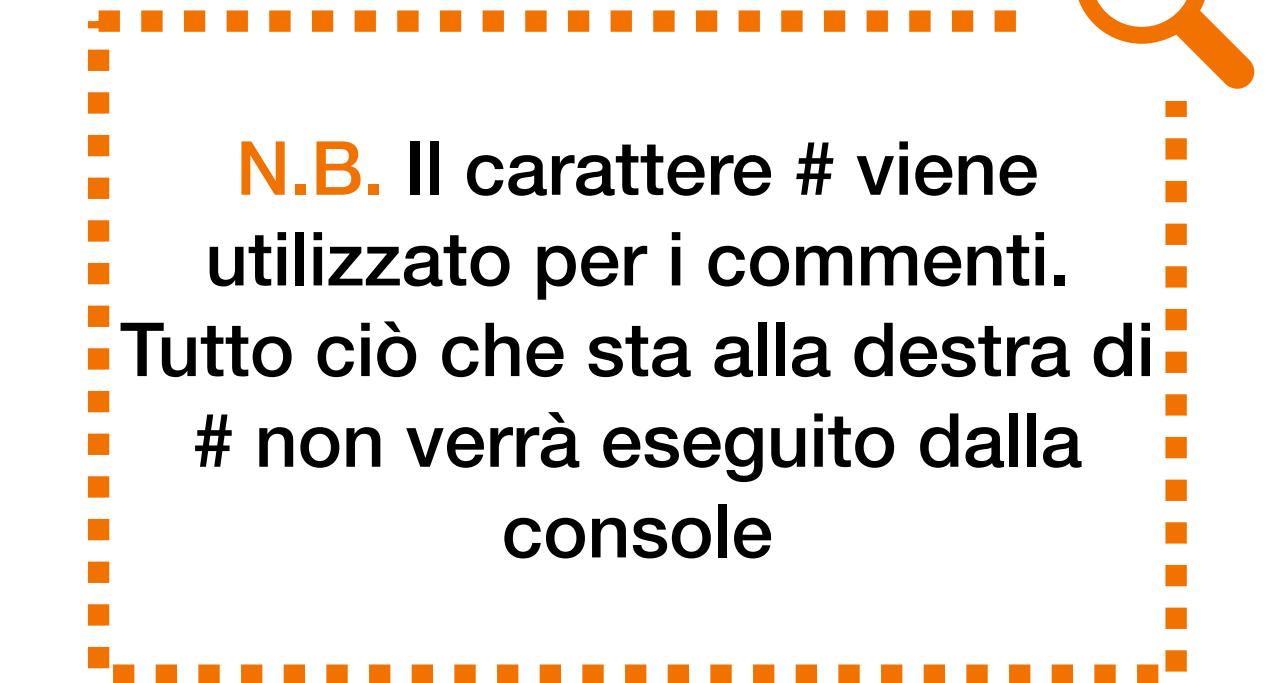
```
> mean(vec.1) #media
```

```
> median(vec.1) #valore mediano
```

```
> range(vec.1) #restituisce un vettore con min e max
```

```
> sd(vec.1) #deviazione standard
```

```
> var(vec.1) #varianza
```



First Steps

Gli oggetti - Variabili e Vettori

```
> vec.1 = c(18,2,3,6,6,4,3)
```

Si può accedere agli elementi del vettore tramite **posizione**:

```
> vec.1[2] #secondo elemento
```

```
[1] 2
```

```
> vec.1[-3] #tutto tranne il terzo elemento
```

```
[1] 18 2 6 6 4 3
```

```
> vec.1[2:4] #elementi dal secondo al quarto
```

```
[1] 2 3 6
```

```
> vec.1[-(2:4)] #tutto tranne gli elementi dal secondo al quarto
```

```
[1] 18 6 4 3
```

```
> vec.1[c(1,5)] #primo e quinto elemento
```

```
[1] 18 6
```

First Steps

Gli oggetti - Variabili e Vettori

```
> vec.1 = c(18,2,3,6,6,4,3)
```

o tramite **valore**:

```
> vec.1[vec.1==6] #elementi uguali a 6 [1] 6 6
```

```
> vec.1[vec.1<4] #elementi minori di 4 [1] 2 3 3
```

```
> vec.1[vec.1 %in% c(2,18)] #elementi contenuti nel set (2,18) [1] 18 2
```

First Steps

Gli oggetti - Fattori

I Fattori vengono utilizzati per i vettori che contengono variabili categoriche in quanto suddividono in *livelli* gli elementi del vettore:

```
> vec.2 = c('blu', 'giallo', 'blu', 'verde', 'verde', 'blu', 'blu')
> fattore.2 = factor(vec.2)
> fattore.2
[1] blu  giallo blu  verde verde blu  blu
Levels: blu giallo verde
```

First Steps

Gli oggetti - Array e Matrici

Le matrici sono vettori bidimensionali che possono essere costruiti partendo da uno o più vettori di dati.

```
> Matrix.1 = matrix(data=
  c(vec.1, vec.2), nrow=7, ncols=2)
> Matrix.1
```

```
[,1] [,2]
[1,] "18" "blu"
[2,] "2"  "giallo"
[3,] "3"  "blu"
[4,] "6"  "verde"
[5,] "6"  "verde"
[6,] "4"  "blu"
[7,] "3"  "blu"
```

Gli array possono invece avere anche più di due dimensioni.

```
> vec.3 = c(1,20,8,5,6,1,5)
> vec.4 = c('giallo', 'verde', 'verde', 'blu', 'verde', 'giallo',
  'verde')
> Array.1 = array(data= c(vec.1, vec.2, vec.3, vec.4),
  dim=c(7,2,2))
> Array.1
```

```
, , 1                  , , 2
                               [,1] [,2]      [,1] [,2]
[1,] "18" "blu"      [1,] "1"  "giallo"
[2,] "2"  "giallo"   [2,] "20" "verde"
[3,] "3"  "blu"       [3,] "8"  "verde"
[4,] "6"  "verde"    [4,] "5"  "blu"
[5,] "6"  "verde"    [5,] "6"  "verde"
[6,] "4"  "blu"       [6,] "1"  "giallo"
[7,] "3"  "blu"       [7,] "5"  "verde"
```

First Steps

Gli oggetti - Dataframes

I Dataframes sono delle matrici in cui i tipi delle colonne possono essere diversi. È il formato dati più comodo da utilizzare in quanto le colonne (e anche le righe) possono essere identificate da un nome.

```
> dataframe = data.frame(age = vec.1, color=vec.2)  
>dataframe
```

	age	color
1	18	blu
2	2	giallo
3	3	blu
4	6	verde
5	6	verde
6	4	blu
7	3	blu

First Steps

Gli oggetti - Dataframes

Per accedere ai singoli vettori (colonne) del dataframe si utilizza il metodo \$:

```
> dataframe$color
```

	age	color
1	18	blu
2	2	giallo
3	3	blu
4	6	verde
5	6	verde
6	4	blu
7	3	blu

```
[1] blu  giallo blu  verde verde blu  blu
```

Per accedere ai singoli elementi del dataframe si utilizzano gli indici di riga e colonna:

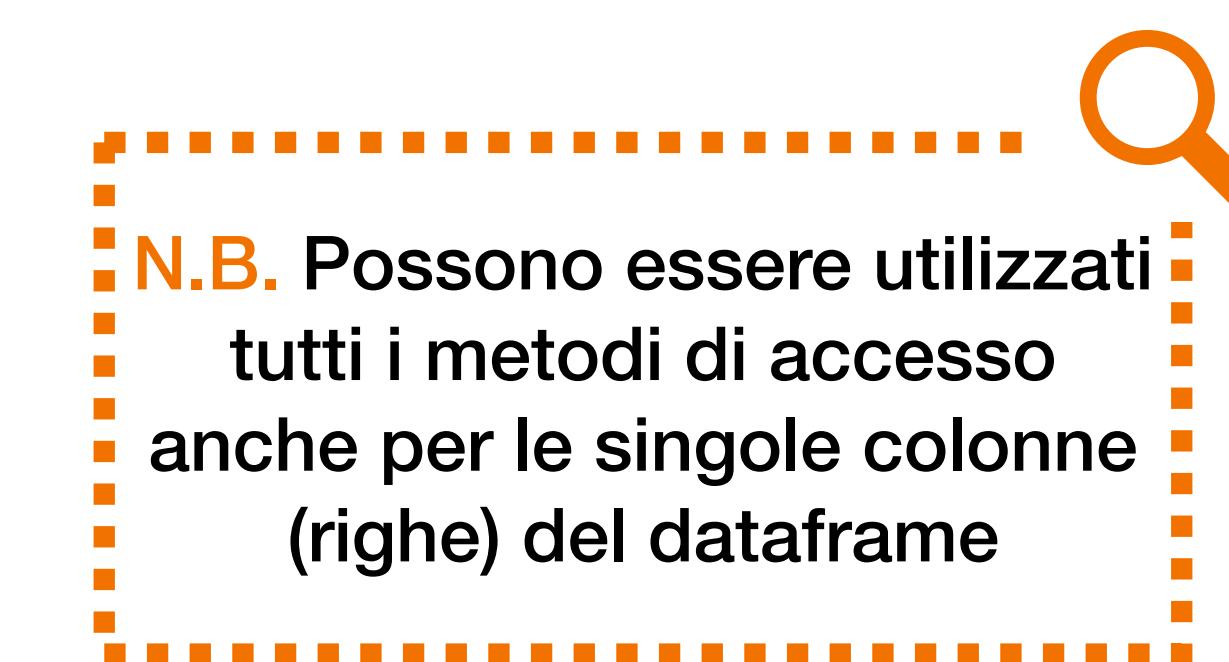
```
> dataframe[2,1]
```

```
[1] 2
```

Oltre all'indice posizionale è possibile utilizzare il nome della colonna (riga):

```
> dataframe[3,"color"]
```

```
[1] blu
```



First Steps

Importare dati esterni

L'elaborazione di dati esterni è la principale funzione di R. Proprio per questo motivo è possibile importare file di diversa natura, dai file di testo separati da tabulazioni a quelli prodotti da altri software (es: SAS).

La funzione più utilizzata per caricare dati esterni è *read.table()* della libreria *utils*:

```
> df = read.table(file.choose())
```

L'unico argomento obbligatorio della funzione *read.table* è il path del file da caricare. Per comodità si usa al suo posto la funzione *file.choose()* che fa apparire il classico pop-up di navigazione dei file.

Una volta aperto il file *esempio.csv* lo si analizza chiamandolo nella console:

```
> df
```

	V1
1	id;sesso;anni;peso;altezza
2	MT ; M ; 69 ;76 ;1.78
3	GF ; F ; 56 ;63 ;
4	MC ; F ; 53 ;71 ;1.60
5	SB ; M ; 28 ;73 ;1.78
6	FE ; F ; 61 ;54 ;1.54
7	AB ; M ; 46 ;92 ;1.84
8	RF ; F ; 31 ;81 ;1.56

Il file risulta essere evidentemente corrotto.

L'errato caricamento del file è dovuta alla non specificazione degli altri argomenti che risultano essere necessari per questo tipo di file.

First Steps

Importare dati esterni

Per importare correttamente il file è necessario specificare la presenza dei nomi delle colonne nella prima riga del file (*header*) ed il separatore (;):

```
> df = read.table(file.choose(), sep=";", header=TRUE )  
> df
```

	id	sesso	anni	peso	altezza
1	MT	M	69	76	1.78
2	GF	F	56	63	NA
3	MC	F	53	71	1.60
4	SB	M	28	73	1.78
5	FE	F	61	54	1.54
6	AB	M	46	92	1.84
7	RF	F	31	81	1.56

Si noti che oltre al *sep* e *header* sono presenti molti altri argomenti nella funzione *read.table*, proprio per la molitudine di file che possono essere importati.

```
> ?read.table
```



Data Input

Description

Reads a file in table format and creates a data frame from it, with cases correspond

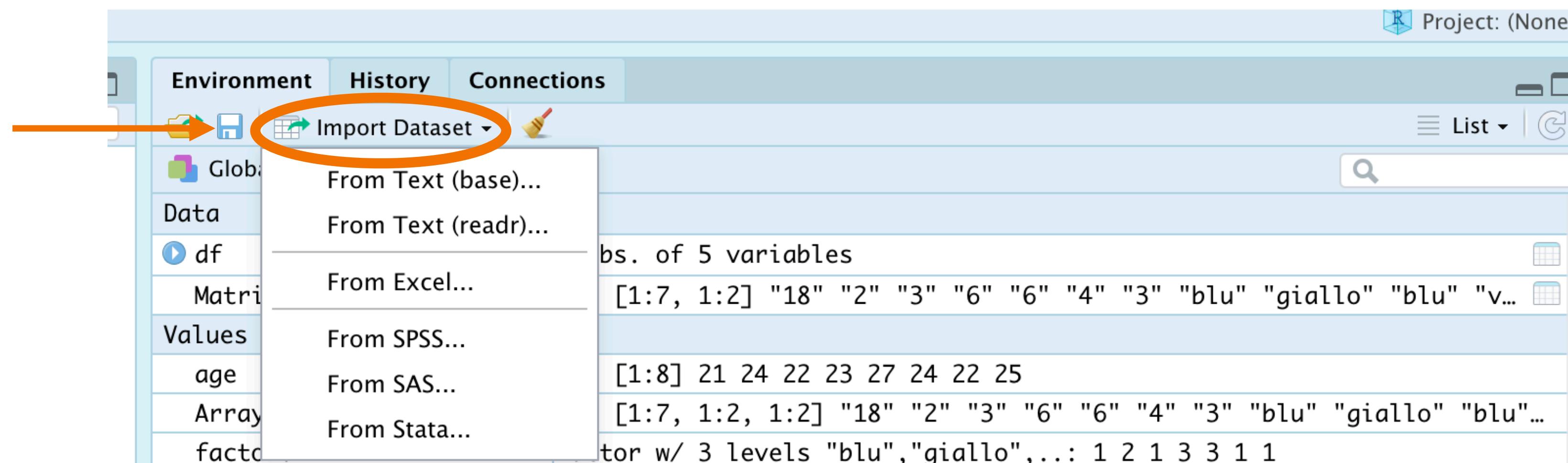
Usage

```
read.table(file, header = FALSE, sep = "", quote = "",  
dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),  
row.names, col.names, as.is = !stringsAsFactors,  
na.strings = "NA", colClasses = NA, nrows = -1,  
skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
strip.white = FALSE, blank.lines.skip = TRUE,  
comment.char = "#",  
allowEscapes = FALSE, flush = FALSE,  
stringsAsFactors = default.stringsAsFactors(),  
fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

First Steps

Importare dati esterni

Come gran parte delle funzioni di base, l'import dei dati può essere eseguito sia da console che in modo grafico sfruttando l'apposito tab in Rstudio.



First Steps

Importare dati esterni

Row names

Automatic
Automatic
Use first column
Use numbers

Comments

None
None

!
%
@
/
~

Import Dataset

Name: esempio

Input File:

id;sesso;anni;peso;altezza
MT ; M ; 69 ;76 ;1.78
GF ; F ; 56 ;63 ;
MC ; F ; 53 ;71 ;1.60
SB ; M ; 28 ;73 ;1.78
FE ; F ; 61 ;54 ;1.54
AB ; M ; 46 ;92 ;1.84
RF ; F ; 31 ;81 ;1.56

Encoding: Automatic

Heading: Yes

Row names: Automatic

Separator: Semicolon

Decimal: Period

Quote: Double quote ("")

Comment: None

na.strings: NA

Strings as factors

Data Frame

id	sesso	anni	peso	altezza
MT	M	69	76	1.78
GF	F	56	63	NA
MC	F	53	71	1.60
SB	M	28	73	1.78
FE	F	61	54	1.54
AB	M	46	92	1.84
RF	F	31	81	1.56

Import Cancel

Separator

Semicolon
Whitespace
Comma
Semicolon
Tab

Decimal

Period
Period
Comma

Data manipulation

Data Exploration

Dataframe Manipulation

Una volta importato il df si passa alla fase di esplorazione dati.

```
> View(df)
```

	id	sesso	anni	peso	altezza
1	MT	M	69	76	1.78
2	GF	F	56	63	NA
3	MC	F	53	71	1.60
4	SB	M	28	73	1.78
5	FE	F	61	54	1.54
6	AB	M	46	92	1.84
7	RF	F	31	81	1.56

```
> str(df)
```

```
data.frame': 7 obs. of 5 variables:  
 $ id     : Factor w/ 7 levels "AB ","FE ","GF ",...: 5 3 4 7 2 1 6  
 $ sesso   : Factor w/ 2 levels " F "," M ": 2 1 1 2 1 2 1  
 $ anni    : num  69 56 53 28 61 46 31  
 $ peso    : num  76 63 71 73 54 92 81  
 $ altezza: num  1.78 NA 1.6 1.78 1.54 1.84 1.56
```

La funzione *summary()* fornisce invece una descrizione delle variabili

```
> str(df)
```

	id	sesso	anni	peso	altezza
AB :1	F :4	Min. :28.00	Min. :54.00	Min. :1.540	
FE :1	M :3	1st Qu.:38.50	1st Qu.:67.00	1st Qu.:1.570	
GF :1		Median :53.00	Median :73.00	Median :1.690	
MC :1		Mean :49.14	Mean :72.86	Mean :1.683	
MT :1		3rd Qu.:58.50	3rd Qu.:78.50	3rd Qu.:1.780	
RF :1		Max. :69.00	Max. :92.00	Max. :1.840	
SB :1				NA's :1	

Data Exploration

Dataframe Manipulation

Per visualizzare il numero di NA su ogni colonna:

```
> colSums(is.na(df))
```

	id	sesso	anni	peso	altezza
.	0	0	0	0	1

Gli NA sono problematici, ad esempio se volessimo calcolare la media dell'altezza:

```
> mean(df$altezza)  
[1] NA
```

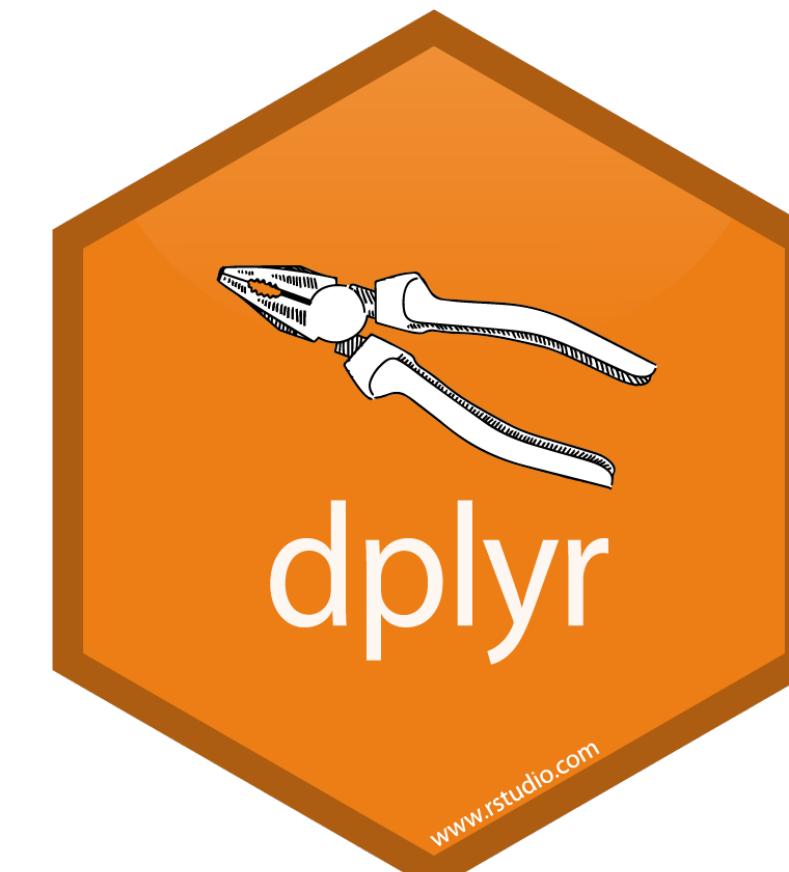
Per ovviare a questo problema usiamo l'argomento *na.rm*:

```
> mean(df$altezza, na.rm=T)  
[1] 1,68333
```

Per eliminare gli NA ed eseguire la maggior parte delle operazioni di data manipulation ci avvaliamo del pacchetto *dplyr*.

Per installare ed importare *dplyr* utilizziamo il pacchetto *tidyverse* che contiene al suo interno anche altre librerie e funzioni utili:

```
> install.packages("tidyverse")  
> library("tidyverse")
```



Data Exploration

Dataframe Manipulation

Per eliminare gli NA utilizziamo la funzione `drop_na()`:

```
> df = drop_na(df)
```

La libreria `dplyr` permette di filtrare il dataframe in maniera agile:

```
> filter(df, sesso == "F")
```

Ovviamente la condizione di filtro può essere più complessa:

```
> filter(df, sesso == "F" | altezza > 1.78)
```

Per selezionare le righe per posizione si usa invece `slice()`:

```
> slice(df, 1:3)
```

Data Exploration

Dataframe Manipulation

Per ordinare il df utilizziamo *arrange()*:

```
> arrange(df, altezza, peso)
```

Il primo argomento è il dataframe, mentre successivamente possiamo inserire le colonne sulle quali vogliamo ordinare il df. Se ne indichiamo più di una, ogni colonna addizionale verrà usata per ordinare i record che hanno pari valore nella colonna precedente. Per ordinare in maniera discendente si utilizza la funzione *desc()* prima del nome della colonna:

```
> arrange(df, desc(altezza), desc(peso))
```

Per selezionare una o più colonne si utilizza *select()*:

```
> select(df, sesso, anni)  
> select(df, anni:altezza)
```

Per rinominare una colonna si utilizza *rename()*:

```
> rename(df, ID = id)
```

Data Exploration

Dataframe Manipulation

Per creare una (o più) nuova colonna si utilizza *mutate()*:

```
> mutate(df, scarto_peso = peso - mean(peso))
```

La funzione *summarise()* collassa il dataframe su una sola riga:

```
> summarise(df, avg_altezza = mean(altezza, na.rm=T))
```

Per campionare n righe utilizziamo *sample_n()* se vogliamo specificare il numero di record da restituire o *sample_frac()* se siamo interessati ad una porzione percentuale del df:

```
> sample_n(df, 10)
```

```
> sample_frac(df, 0.2)
```

Per campionare con reinserimento si usa il parametro *replace=T*:

Data Exploration

Dataframe Manipulation

È spesso utile aggregare i dati secondo i livelli dei fattori o gruppi di record di particolare tipologia. Per fare ciò si usa la funzione *groupby()*:

```
> group_by(df, sesso)
```

Per essere utile però la funzione di *groupby* deve essere accompagnata da altre funzioni, come ad esempio applicare la funzione *media* ai gruppi ottenuti precedentemente.

È per questo motivo che si usa l'operatore *pipe* : %>%

Per capire il funzionamento dell'operatore *pipe* si pensi agli argomenti delle funzioni di *dplyr* viste finora. Tutte presentano come primo argomento il dataframe su cui applicare la funzione. L'operatore *pipe* sfrutta questa particolarità permettendo di applicare più funzioni allo stesso dataframe.

Data Exploration

Dataframe Manipulation

Per utilizzare l'operatore pipe si dichiara il dataset su cui si lavora e poi si concatenano le funzioni da applicare l'una dopo l'altra con il *pipe*:

```
> df %>% group_by(sesto) %>% summarise(media = mean(anni))
```

Si noti come l'argomento *data* delle funzioni *dplyr* non viene più utilizzato se è presente l'operatore pipe

Data visualization

Data Viz

From data to viz

L'elaborazione dei dati ha come fine ultimo quello di comunicare dei risultati.
La forma più semplice, diretta ed esaustiva di condivisione dei risultati è quella grafica

“A picture is worth a thousand words”

Con il termine data visualization si intende la rappresentazione di informazioni tramite grafici, diagrammi, immagini eccetera.
La dataviz non si limita ai soli grafici, ma a tutto ciò che è “narrato” in maniera grafica.

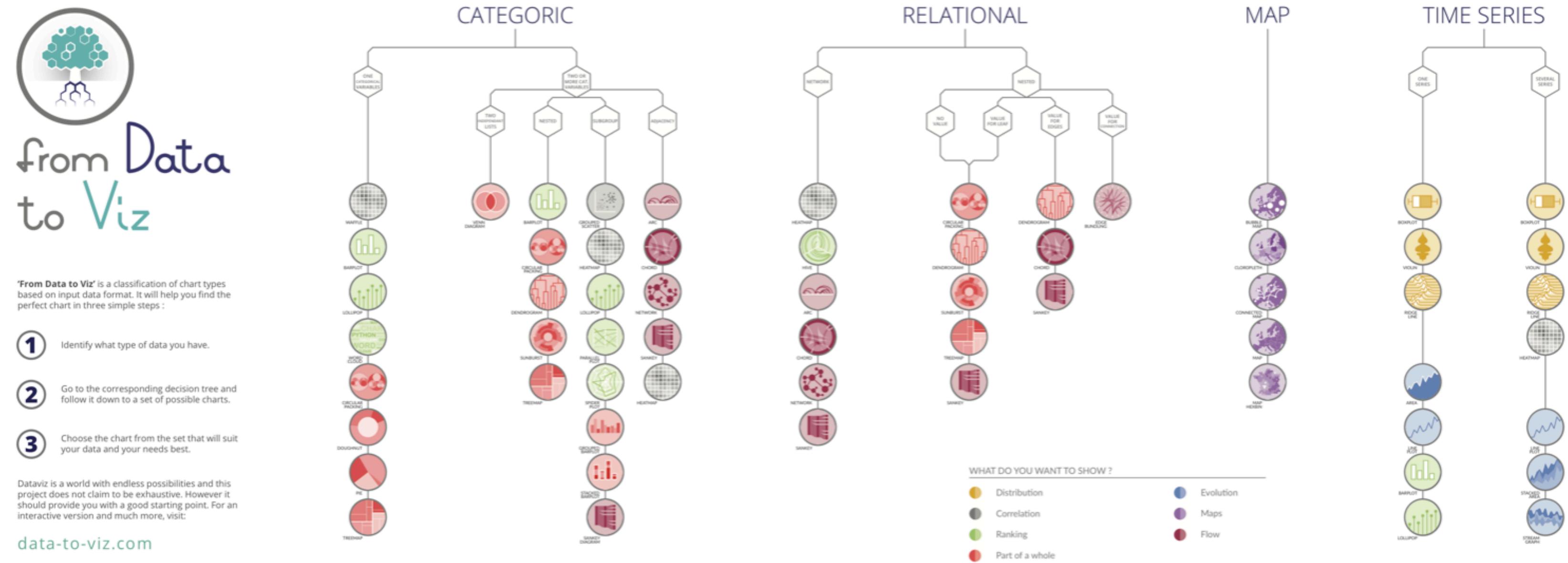
In questa parte del corso ci concentreremo su uno degli strumenti della dataviz, i grafici.

Data Viz

From data to viz

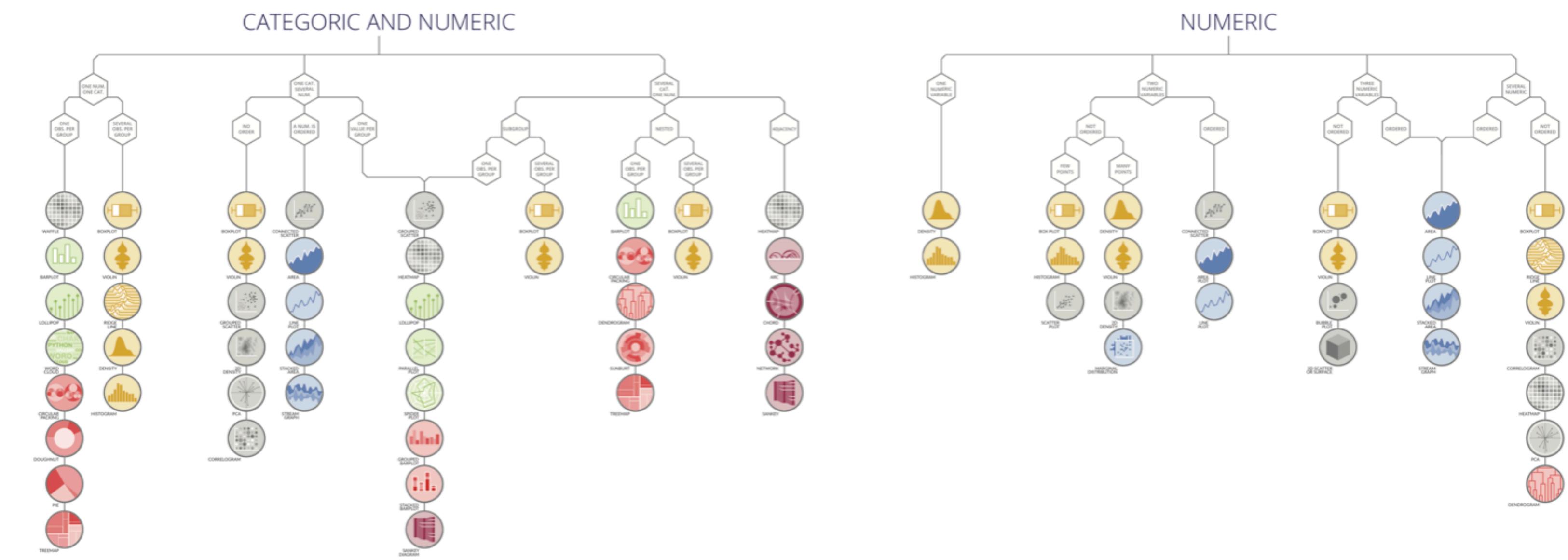
Quanti grafici esistono?

TROPPI.



Una tassonomia dei grafici è quella di *data-to-viz*:

https://www.data-to-viz.com/img/poster/poster_big.png



Data Viz

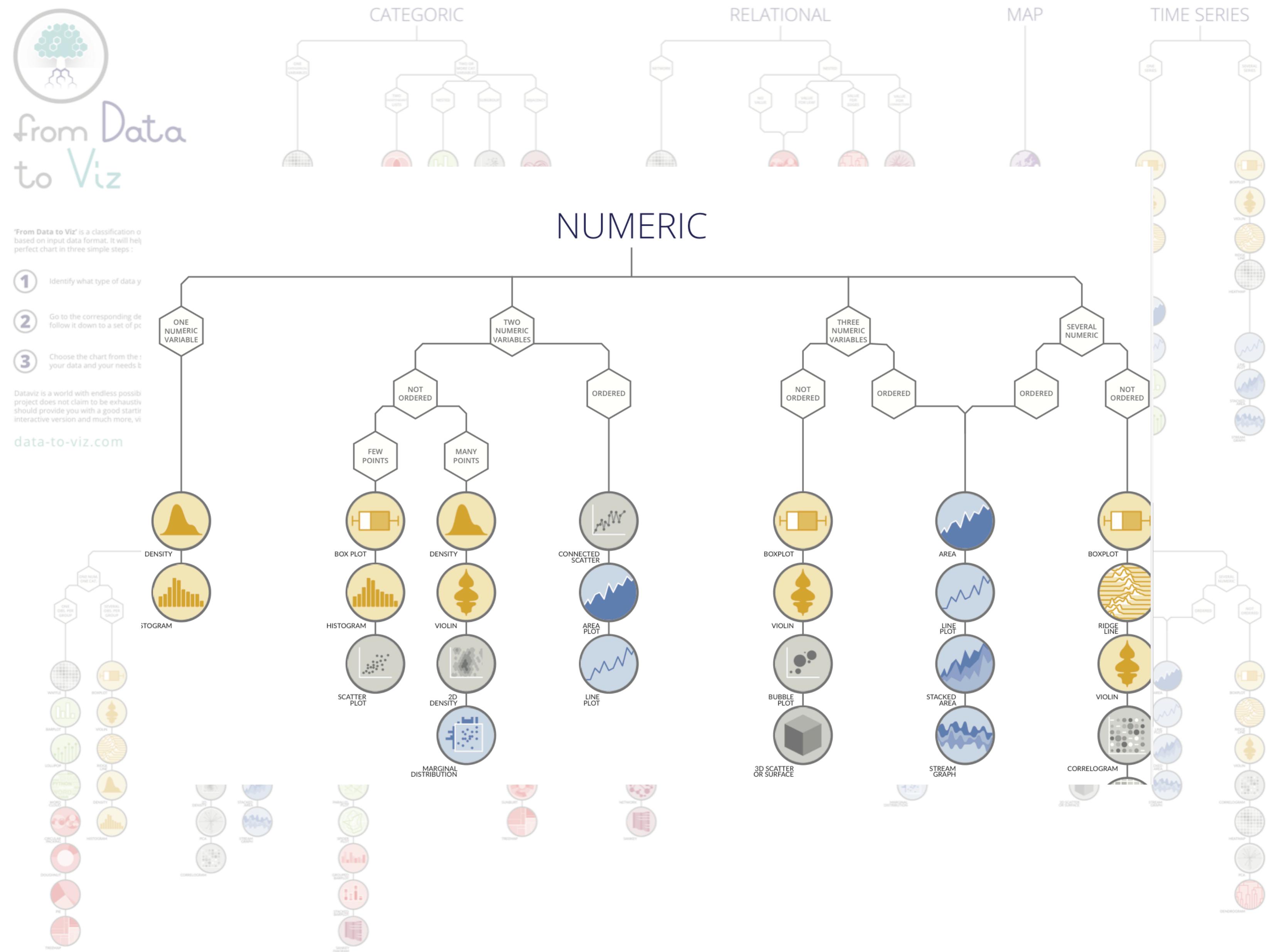
From data to viz

Quanti grafici esistono?

TROPPI.

Una tassonomia dei grafici
è quella di *data-to-viz*:

https://www.data-to-viz.com/img/poster/poster_big.png



Data Viz

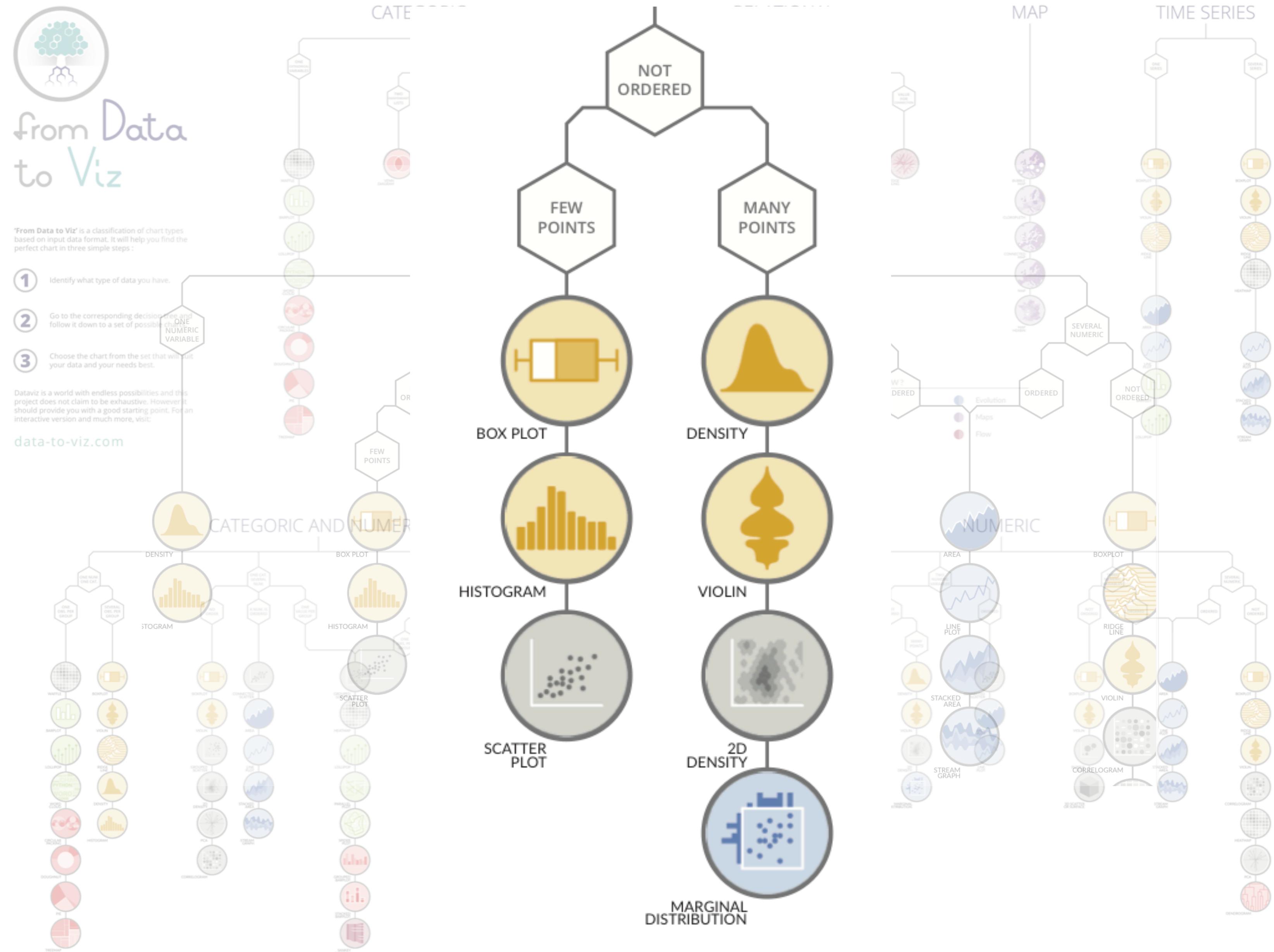
From data to viz

Quanti grafici esistono?

TROPPI.

Una tassonomia dei grafici è quella di *data-to-viz*:

https://www.data-to-viz.com/img/poster/poster_big.png



Data Viz

From data to viz

Basta scegliere il grafico corretto per una buona infografica?

NO.

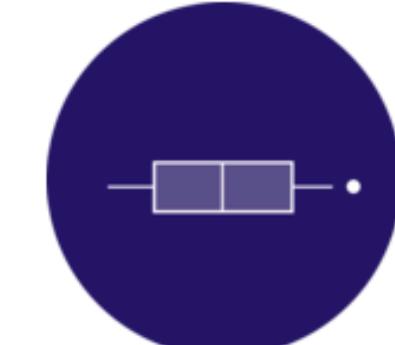
Gli errori più comuni da evitare sono riassunti in:

<https://www.data-to-viz.com/caveats.html>

CAVEATS

A collection of dataviz caveats by data-to-viz.com

Show all Top 10 Improvement Misleading Map Bar

 <p>Order your data</p> <p>When displaying the value of several entities, ordering them makes the graph much more insightful.</p>	 <p>To cut or not to cut?</p> <p>Cutting the Y-axis is one of the most controversial practice in data viz. See why.</p>	 <p>The spaghetti chart</p> <p>A line graph with too many lines becomes unreadable: it is called a spaghetti graph.</p>	 <p>Pie chart</p> <p>The human eye is bad at reading angles. See how to replace the most criticized chart ever.</p>
 <p>Play with histogram bin size</p> <p>Always try different bin sizes when you build a histogram, it can lead to different insights.</p>	 <p>Do boxplots hide information?</p> <p>Boxplots are a great way to summarize a distribution but hide the sample size and their distribution.</p>	 <p>The problem with error bars</p> <p>Barplots with error bars must be used with great care. See why and how to replace them.</p>	 <p>Too many distributions.</p> <p>If you need to compare the distributions of many variables, don't clutter your graphic.</p>

Data Viz

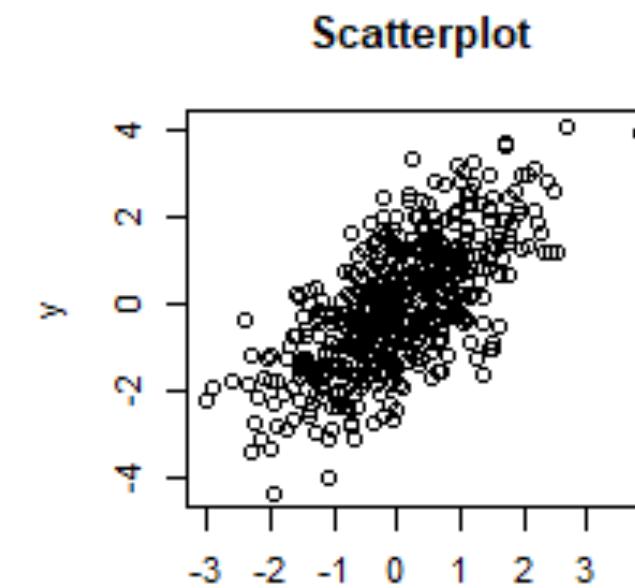
I grafici su R - *graphics*

Le 3 librerie più comuni per creare grafici su R sono *graphics* (che fa parte delle librerie “base” di R), *ggplot2* e *lattice*. Partiamo dal pacchetto più semplice, *graphics*.

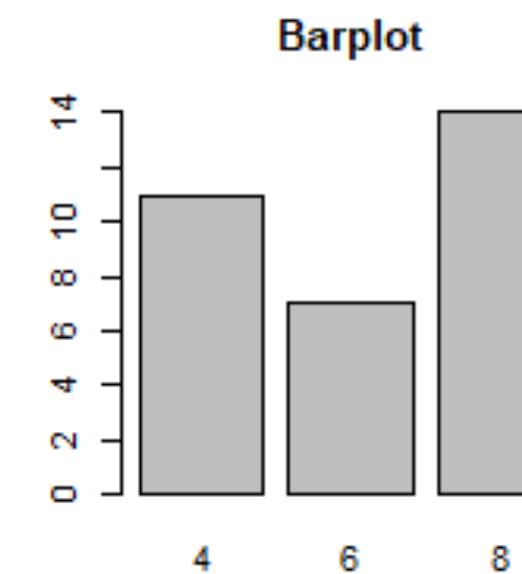
La funzione *plot()* permette di creare un generico grafico. La tipologia, l'apparenza e le specifiche del grafico vengono definite tramite gli argomenti.

Il primo argomento della funzione *plot()* è il dato da rappresentare. In base alla tipologia di dato passato alla funzione (vettore numerico, fattore, time series, ...) verrà restituito un grafico diverso

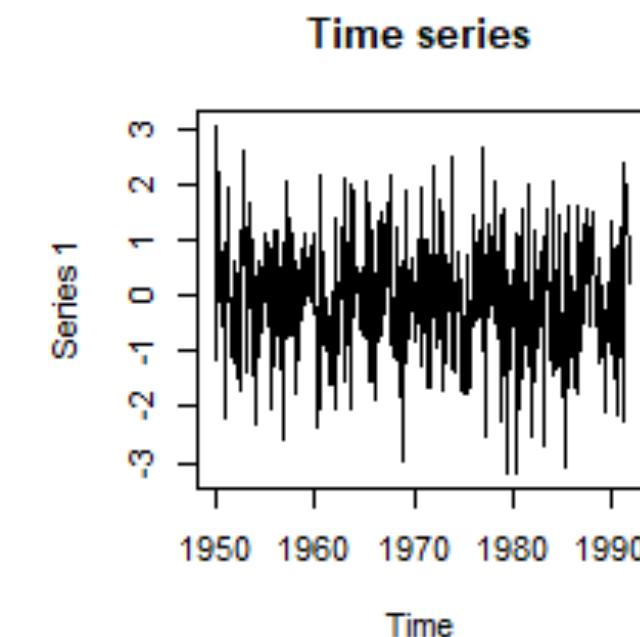
```
> x = rnorm(500)  
> y = x + rnorm(500)  
> plot(x, y, main="scatterplot")
```



```
> data(mtcars)  
> attach(mtcars)  
> myfactor = factor(mtcars$cyl)  
> plot(myfactor, main="barplot")
```



```
>myts = ts(matrix(rnorm(500), nrow =  
500, ncol = 1), start = c(1950, 1),  
frequency = 12)  
>plot(myts, main="time series")
```



Data Viz

I grafici su R - *graphics*

Per stampare diversi grafici contemporaneamente si utilizza la funzione *par()* e si indicano il numero di grafici (e la loro disposizione) tramite l'argomento *mfrow*:

```
> par(mfrow = c(1,3))  
> plot(...)  
> plot(...)  
> plot(...)
```

Produrrà 3 grafici in linea.

Per personalizzare il grafico esistono diversi argomenti. Ad esempio *type* permette di specificare la tipologia di rappresentazione dei dati (ma dipende dal tipo di dato passato), oppure *pch* permette di modificare il simbolo dei punti nel grafico

Data Viz

I grafici su R - *graphics*

I diversi elementi del grafico possono anche essere aggiunti in maniera sequenziale utilizzandoli come funzioni:

```
> plot(x,y)  
> title("Titolo")
```

Equivale a:

```
> plot(x, y, main="Titolo")
```

Gli assi possono essere rinominati con i parametri *xlab* e *ylab* oppure si può rimuovere il loro nome tramite *ann*. Anche gli assi possono essere aggiunti (e gestiti) successivamente tramite la funzione *axis()*.

Una panoramica completa sulla personalizzazione dei grafici di base può essere trovata su:

<https://r-coder.com/plot-r/>

Data Viz

I grafici su R - *ggplot2*

ggplot2 è una libreria grafica molto potente basata sulla *Grammar of Graphics* di Wilkinson, pubblicazione che descrive le caratteristiche fondamentali di un grafico statistico. Le componenti principali di un grafico sono definite come i dati, gli attributi estetici e gli oggetti geometrici.

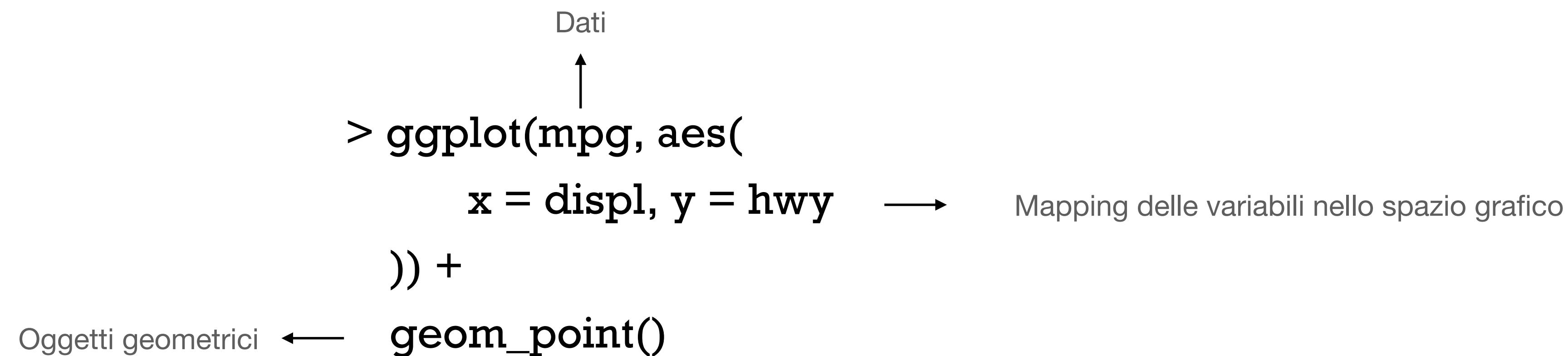
In generale, i grafici sono composti dai dati e da un *mapping* che descrive come le variabili vengono mappate agli attributi estetici. In particolare esistono 5 tipologie di mapping:

1. **layer**: collezione di elementi geometrici (*geom*) e trasformazioni statistiche (*stat*)
2. **scale**: mappa i valori dallo spazio dati allo *spazio estetico* (include l'uso di legenda, colori, forme e dimensioni)
3. **coord**: descrive come le coordinate dei dati vengono mappate sul piano del grafico
4. **facet**: specifica come dividere e mostrare subset di dati
5. **theme**: controlla le particolarità della visualizzazione, come colore di sfondo e dimensione dei caratteri

Data Viz

I grafici su R - *ggplot2*

La funzione per produrre un grafico con *ggplot2* è *ggplot()* che accetta come argomenti principali i dati da cui estrarre il grafico e i parametri *aes*. A questi vanno aggiunti gli oggetti geometrici con l'operatore +:



Data Viz

I grafici su R - *ggplot2*

Si possono inserire anche più oggetti geometrici

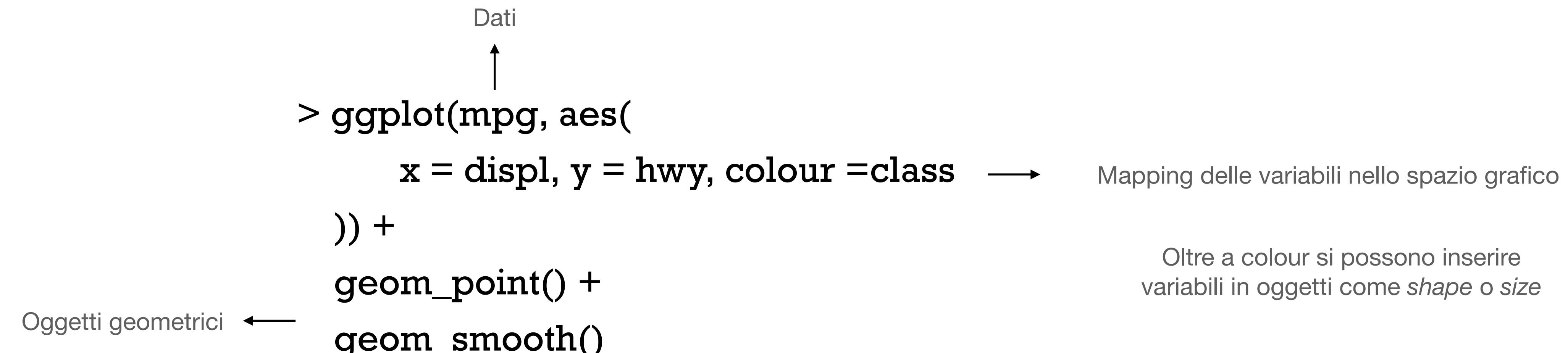
```
Dati  
↑  
> ggplot(mpg, aes(  
      x = displ, y = hwy) → Mapping delle variabili nello spazio grafico  
      )) +  
      geom_point() +  
      geom_smooth()  
Oggetti geometrici ←
```

Altri oggetti geometrici sono
geom_histogram, *geom_boxplot*,
geom_bar, *geom_path*, *geom_line*,
geom_violin, ecc..

Data Viz

I grafici su R - *ggplot2*

O aggiungere altre variabili in elementi grafici diversi dagli assi



Altri oggetti geometrici sono
`geom_histogram`, `geom_boxplot`,
`geom_bar`, `geom_path`, `geom_line`,
`geom_violin`, ecc..

Data Viz

I grafici su R - *ggplot2*

Gli assi vengono gestiti tramite le funzioni *xlab* e *ylab* per il nome, mentre si usano *xlim* e *ylim* per modificarne i limiti

```
> ggplot(mpg, aes(  
      x = displ, y = hwy, colour = class) → Mapping delle variabili nello spazio grafico  
      )) +  
      geom_point() +  
      geom_smooth() +  
      xlab("nome asse x") +  
      ylab("nome asse y") +  
      xlim(0,5) +  
      ylim(0,30)
```

Dati
↑
Oggetti geometrici ←
Parametri sugli assi ←

Data Viz

I grafici su R - *ggplot2*

Per salvare un grafico si può usare il tasto *export* nel tab *plots* di Rstudio oppure la funzione `ggsave()`

```
> ggsave("nomedelfile.estensione", graficodasalvare)
```

Per approfondire la creazione di grafici con ggplot si consiglia la lettura della documentazione:

<https://ggplot2-book.org/introduction.html>

Next steps

Next steps

For loop

Utile quando si deve eseguire un'azione su più elementi

```
> for(variabile contatore in range di variazione) {  
    azione  
}
```

Next steps

For loop

Utilizzando il df *mpg*, per stampare il nome delle prime 5 variabili:

```
Azione      ←—————> Variabile contatore      Range di variazione  
> for( i in 1:5 ) {  
    print(names(mpg[,i]))  
}
```

n.b. questo esempio serve ad apprendere il funzionamento del ciclo, quale sarebbe stato il modo più efficiente per vedere il nome delle prime 5 colonne?

```
> names(mpg)[1:5]
```

Next steps

If - else

Dato l'oggetto $a = 5$, la condizione

```
> if ( is.numeric(a) ) {  
    print("l'oggetto a è un numero")  
} else {  
    print("l'oggetto a non è un numero")  
}
```

Restituirà “ l'oggetto a è un numero “ in quanto la condizione risulta essere soddisfatta

Next steps

For loop & If - else

Combiniamo il ciclo for e la condizione if - else in modo da estrarre degli indici sintetici (mediana, range interquartile) dalle variabili del df mpg nel caso in cui siano numeriche

```
>variables = names(mpg)
>out = c()
>for (i in 1:length(variables)) {
  if (is.numeric(mpg[, variables[i]])){
    out = c(out, paste("mediana :", as.numeric(summary(mpg$displ)[3])))
  } else {
    out = c(out, paste("(non numerica)"))
  }
}
```

Next steps

Functions

Utile quando per snellire il codice e renderlo più ordinato

```
> nome della funzione = function(argomenti) {  
    azioni  
    return(oggetto da restituire)  
}
```

Gli **argomenti** possono essere obbligatori o facoltativi (o anche assenti). Un argomento facoltativo viene accompagnato da un valore di default, uno obbligatorio no.

Next steps

Exercises

1. Importare come `data.frame` il dataset `airquality` del pachetto `datasets`
2. Contare i null values per colonna
3. Eliminare i null values
4. Trasformare la colonna `Temp` (espressa in °F) in °C
5. Trasformare la colonna `Wind` (espressa in miglia/h) in km/h
6. Qual'è la correlazione fra il livello di ozono e la temperatura?
7. Produrre un grafico con la temperatura media per mese

Next steps

Portali utili

1. github.com
2. kaggle.com
3. towardsdatascience.com
4. stackoverflow.com

Next steps

Reach me

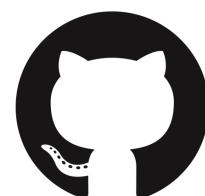


raffaele.anselmo2@unibo.it: *domande sul corso*

r.anselmo@crif.com : *curiosità sul mondo professionale in ambito data science*



<https://www.linkedin.com/in/raffaele-anselmo/>



<https://github.com/RaffaeleAns>

Thanks

Rufus Selway