1. L'ambiente R

TECNICHE DI ANALISI DI DATI I

Raccomandazioni, raccomandazioni, raccomandazioni…

Leggete la dispensa e/o seguite le slide con R aperto e rifate **CONTESTUALMENTE TUTTI** gli ESEMPI proposti.

Accompagnate allo studio e alla replica degli esempi **l'esecuzione degli ESERCIZI** proposti nella dispensa e su ELLY, e fateli correggere dal docente, SOPRATTUTTO in caso di difficoltà.

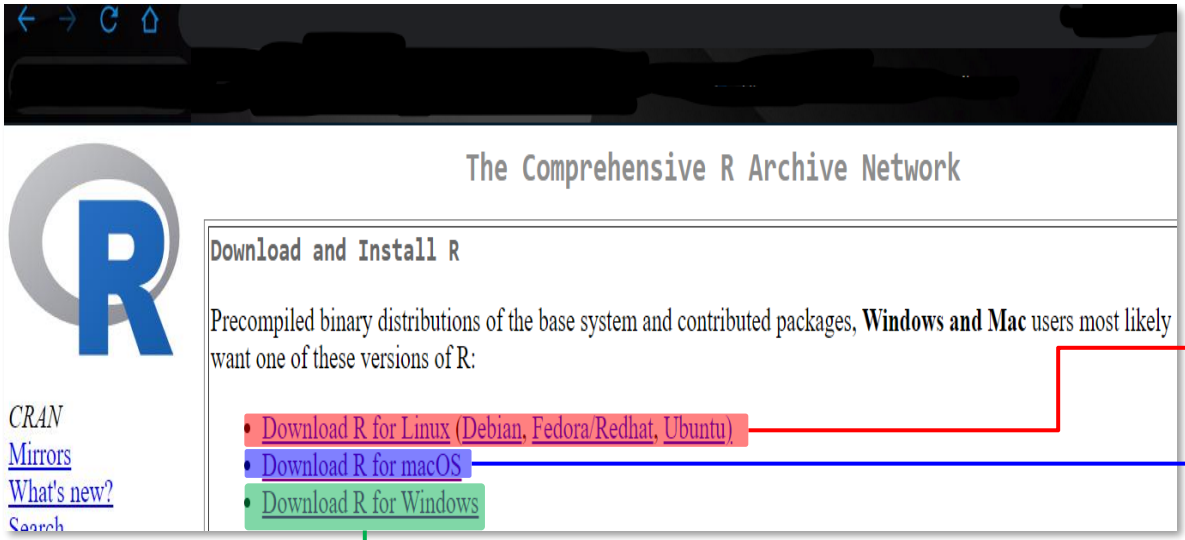
L'ambiente R

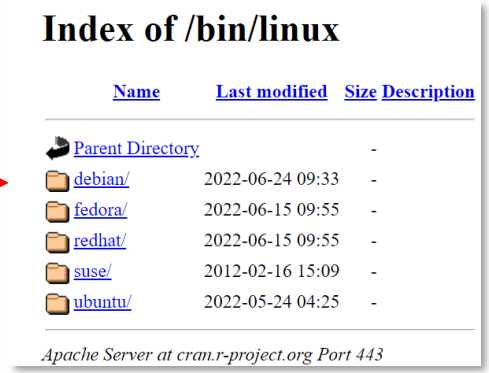
R è un ambiente per elaborazioni statistiche e rappresentazioni grafiche, gratuito, *open source*, versatile e dinamico. Dispone di un pacchetto – base con molte funzionalità, ampliabili scaricando **package** ad hoc che aggiungono ulteriori funzioni (analisi, grafici) al programma.

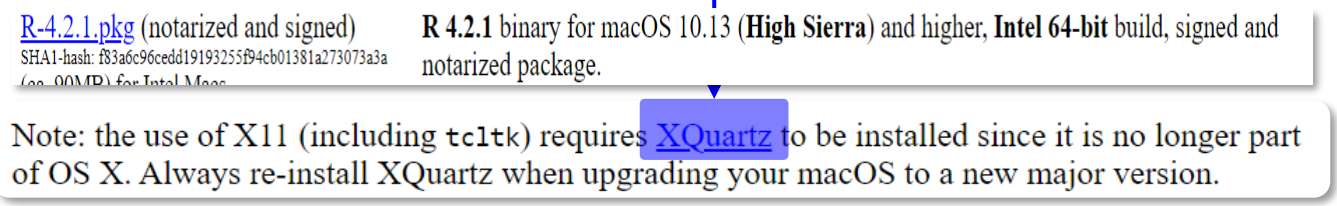
Chiunque, armato di buona volontà e buone competenze di programmazione, può creare un proprio package e metterlo a disposizione di tutti. Chi è armato di buona volontà e medio-basse-nulle competenze di programmazione (noi) potrà scaricare package che contengono le funzioni utili per le proprie analisi **da un archivio dedicato**: **CRAN** (Comprehensive R Archive Network) contiene i file di sistema da scaricare per installare R i package e molti file *help.* L'archivio CRAN è *mirrored*: sono **replicate identiche versioni dell'archivio** su server in tutto il mondo: **sceglieremo il CRAN geograficamente più vicino**.

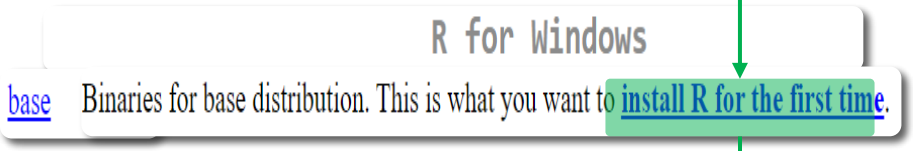
Scaricare e installare R: https://cran.r-project.org/

Scegliete il vostro sistema operativo









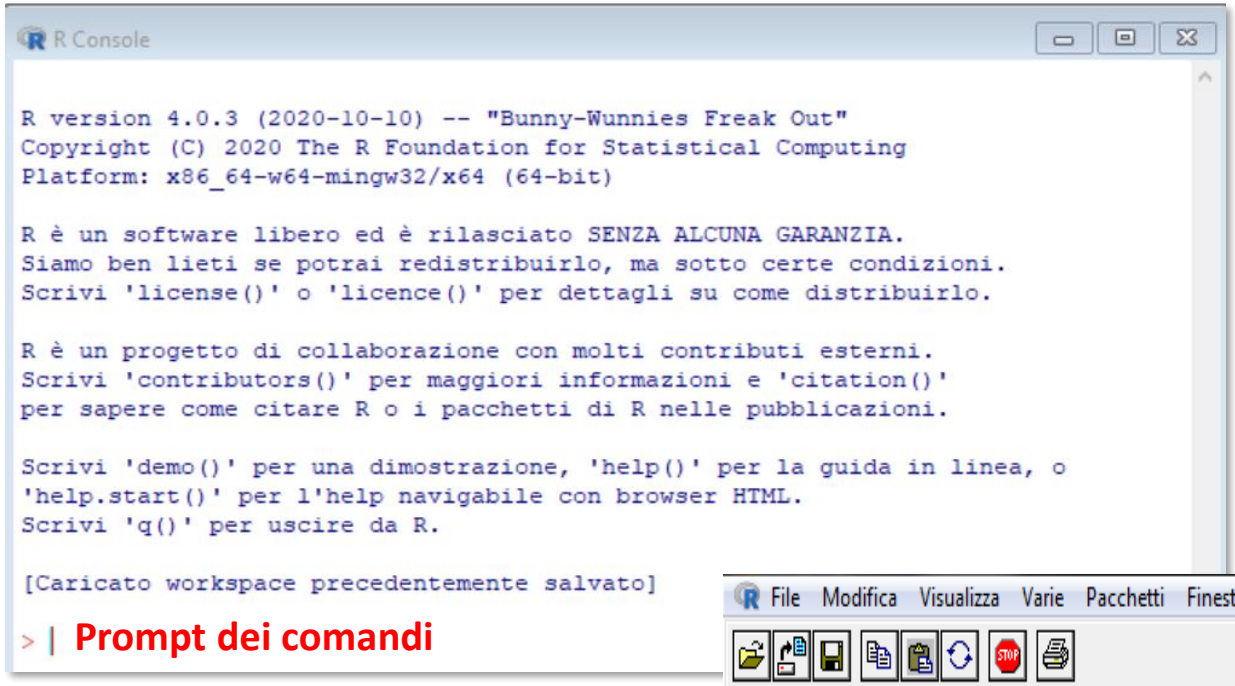
La struttura: Console, Script, Graph

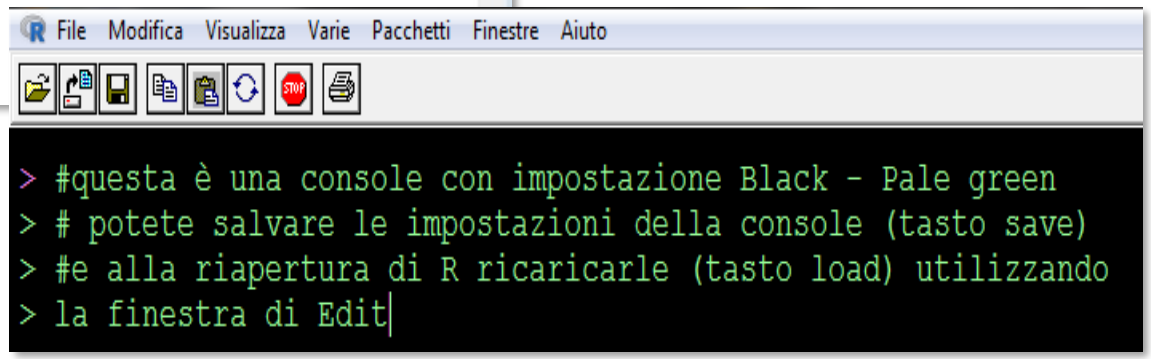
La console

Nella **console** digitiamo i comandi e vediamo i loro risultati.

> indica che R è pronto a ricevere istruzioni: si digita il comando, poi **Invio (Enter)**.

```
> 2+2
[1] 4
>
```





Per modificare la console: menu

Modifica/Edit → **Preferenze**

interfaccia

Per inserire commenti o promemoria: **#** prima del testo

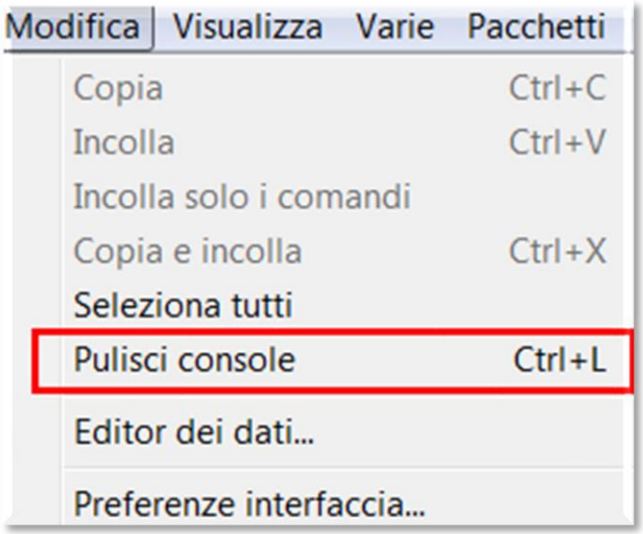
```
> #quanto fa 2+2?
> 2+2
[1] 4
> #la mia prima analisi in R!
```

Se si dimentica #, e in tante altre occasioni, l'output è un **error**: R si aspetta un comando, e non riconosce quanto scritto come una sua funzione

```
> quanto fa 2+2?
Errore: unexpected symbol in "quanto fa"
```

I comandi si seguono uno dopo l'altro in Console.

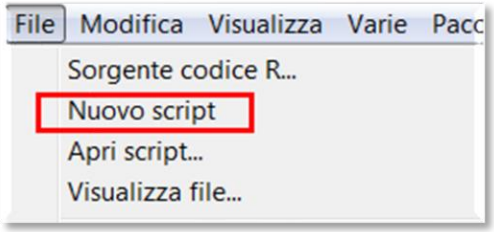
Se volete tornare ad avere una console pulita, si usa la combinazione di tastiera **Crl+L**, o, nel menu Modifica / Edit, si sceglie **Pulisci / Clear console**:

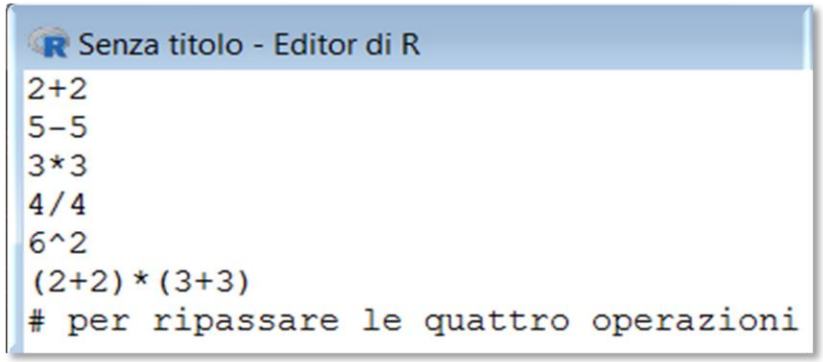


Lo script

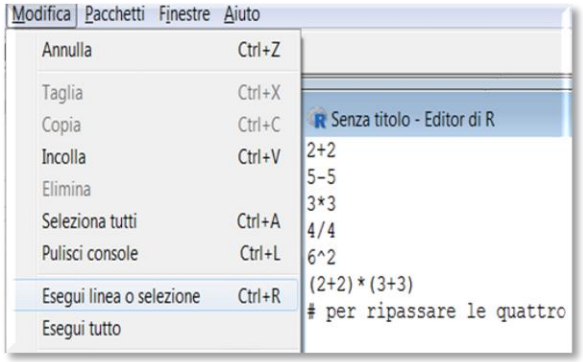
Negli script sono scritte **e salvate varie linee di comando**. Sono utili, per esempio, quando le stesse analisi devono essere fatte più volte.

Per creare un nuovo script, selezionare **File → Nuovo/New script**

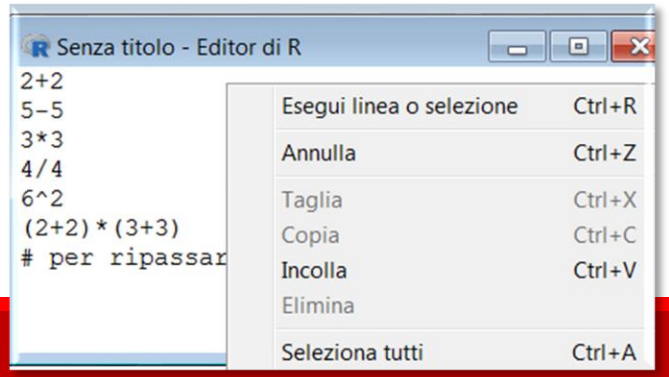




I comandi sono scritti uno dopo l'altro, **senza il prompt >**. Si possono scrivere promemoria utilizzando **#**:

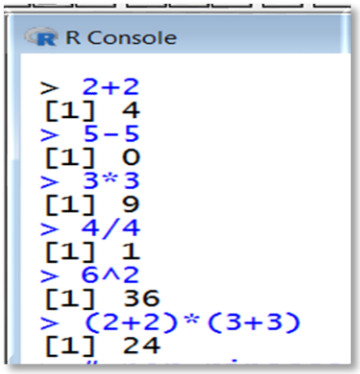


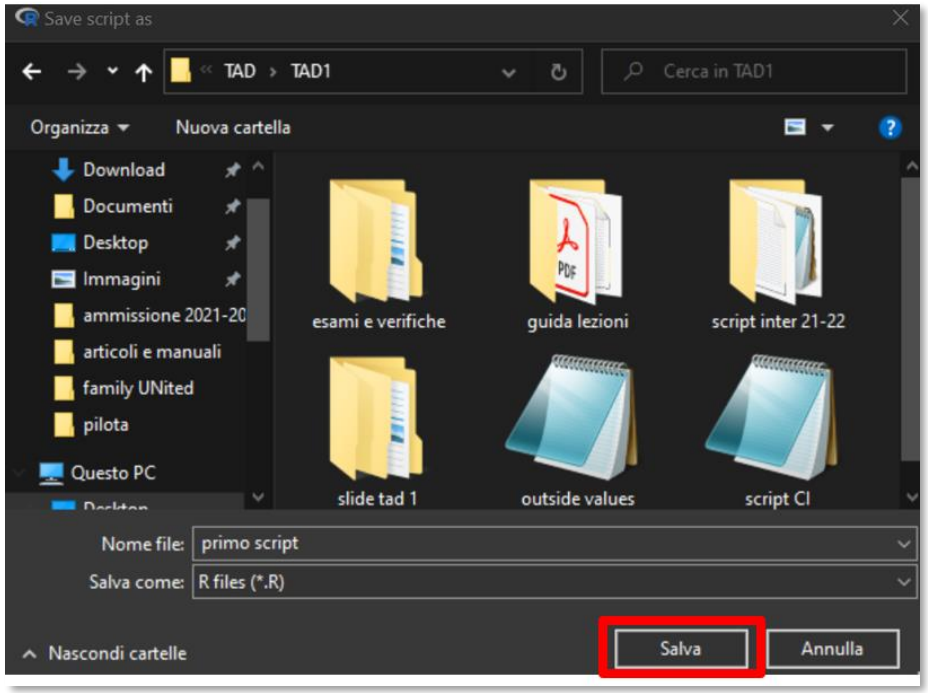
Si eseguono con **Edit → Esegui / Run all** o **Esegui/ line or selection** (in Windows, o con **Edit → Execute** in MacOs



Cliccando con il tasto destro del mouse nello script si seleziona l'azione :

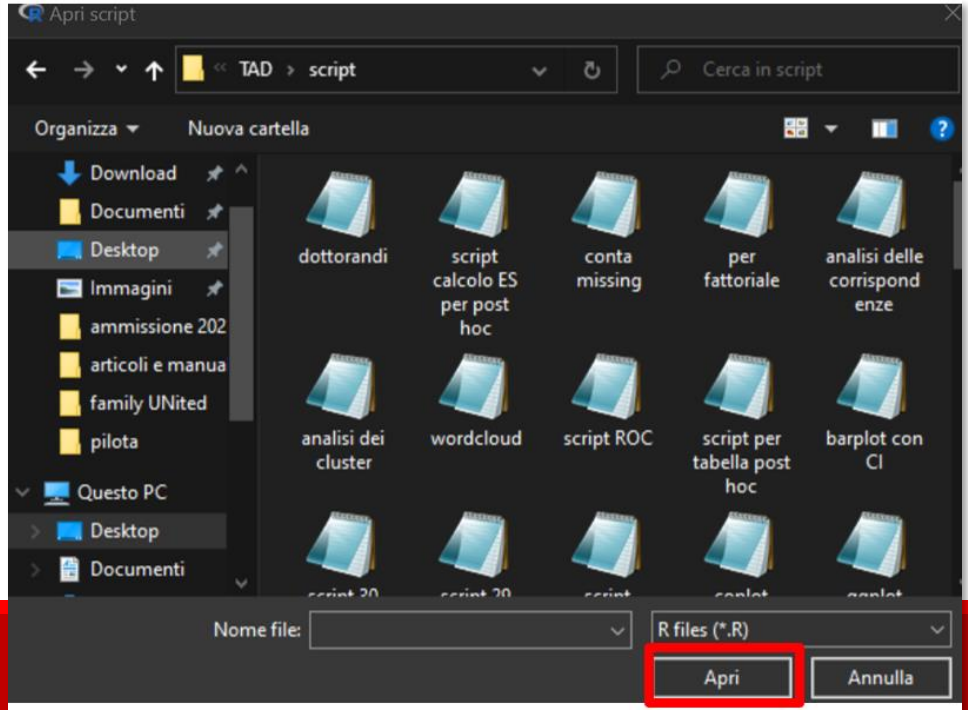
L'output dei comandi selezionati è **stampato nella Console**:





Lo script viene salvato (**File→ Salva come** o **File → Salva**), in una cartella / directory a vostro piacere, come file di R, con estensione .R

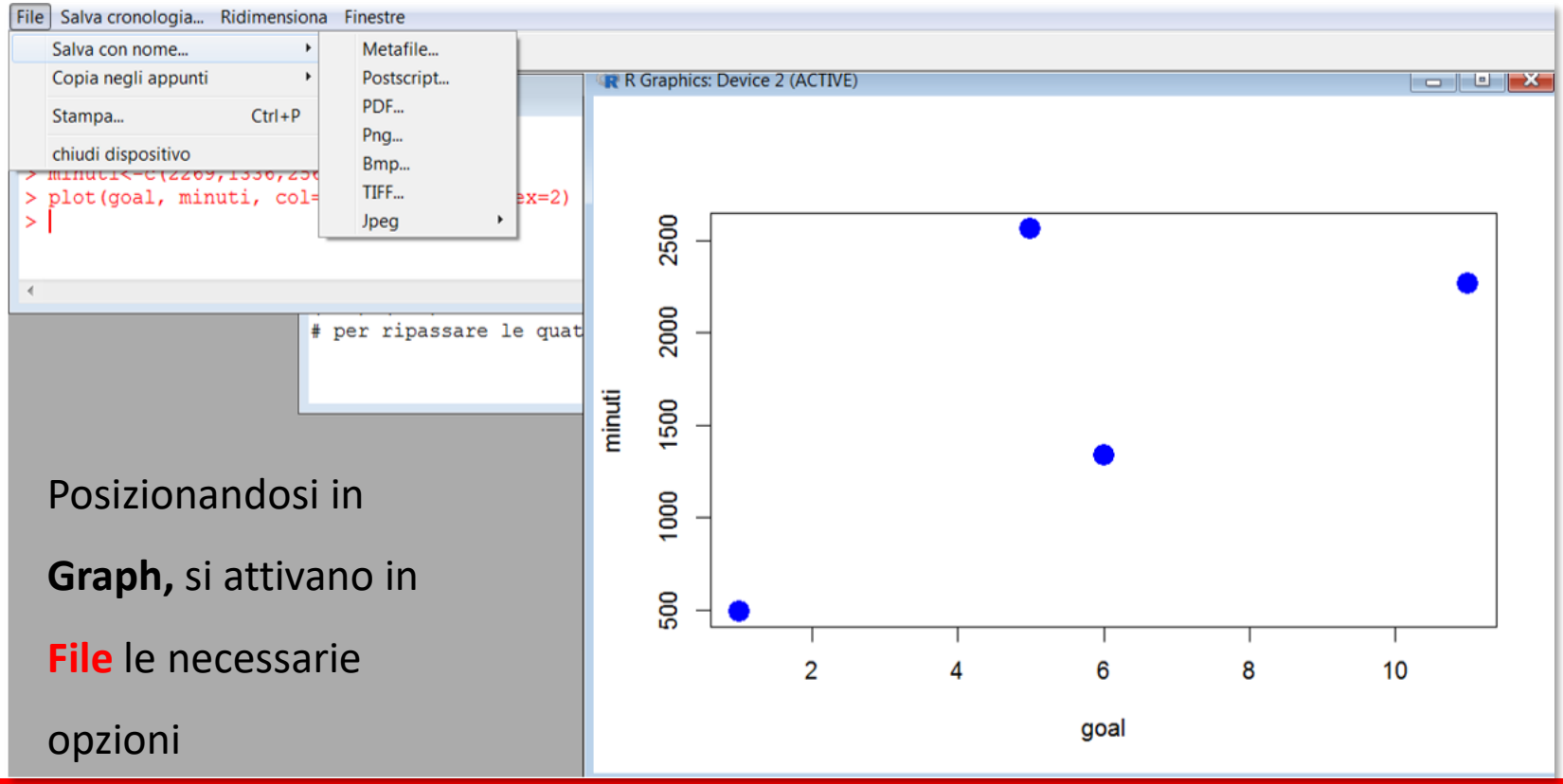
Lo script può essere aperto con il menu **File→ Open script**



Graph

Ai grafici è dedicata **una finestra, creata quando in console si produce un grafico**.

Ogni grafico si sovrappone ai precedenti. Quindi, se si desidera salvare il grafico, bisogna **salvarlo** (come immagine o, meglio, come .pdf) o **copiarlo** in un altro file.



Posizionandosi in **Graph,** si attivano in **File** le necessarie opzioni

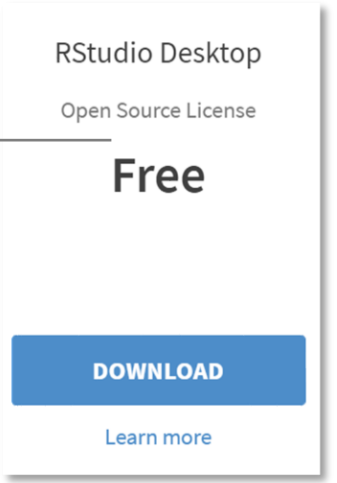
Altre interfacce: RStudio e RCommander

Non è assolutamente obbligatorio usare l'una o l'altra, o una e l'altra, o né l'una né l'altra: il **software è sempre R**, e quello che ci interessa è il prodotto, non la facciata. **Scegliete l'interfaccia che vi pare più coerente** con il vostro stile cognitivo e più produttiva per

---

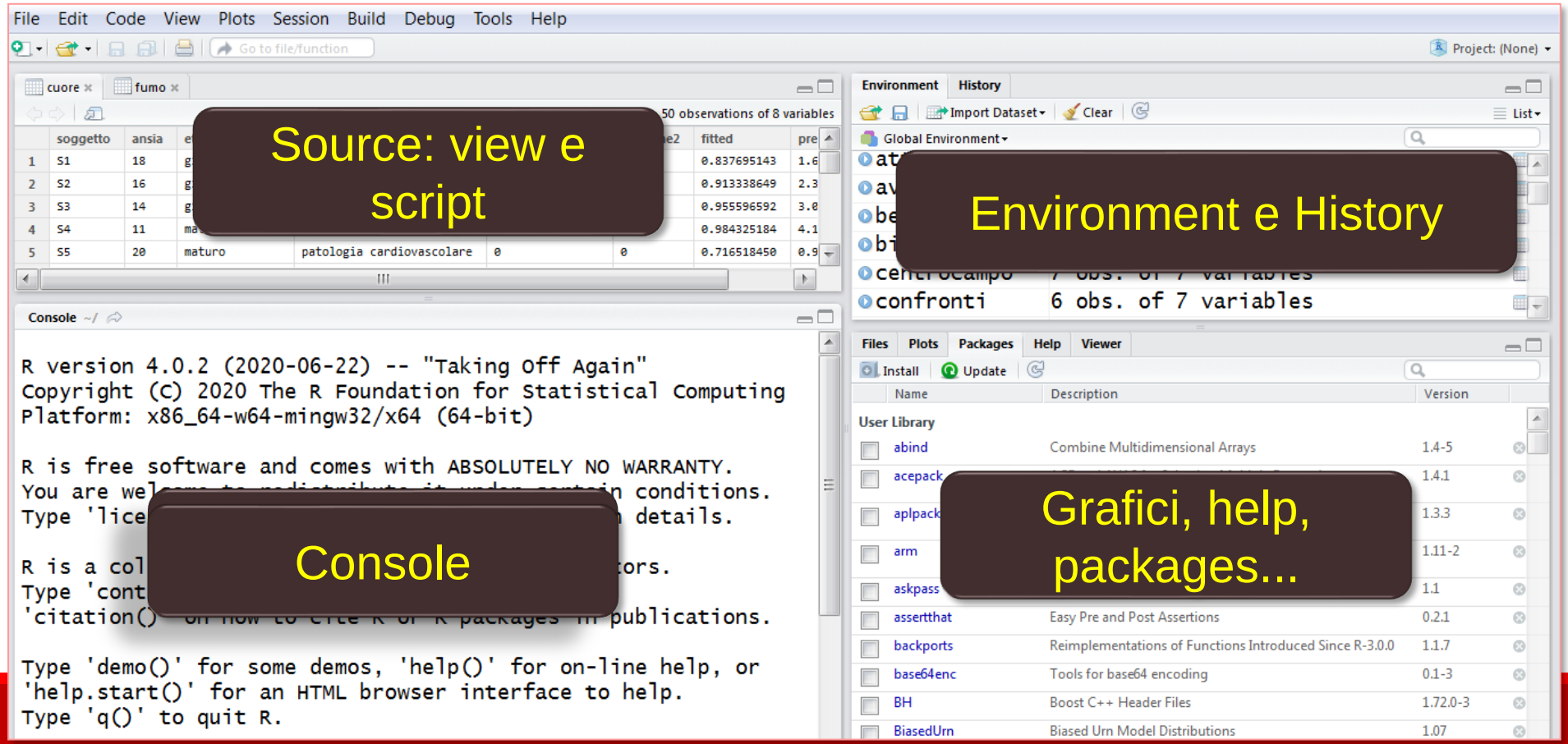
arrivare ai nostri fini, ovvero capire qualcosa da una massa di dati.

RStudio



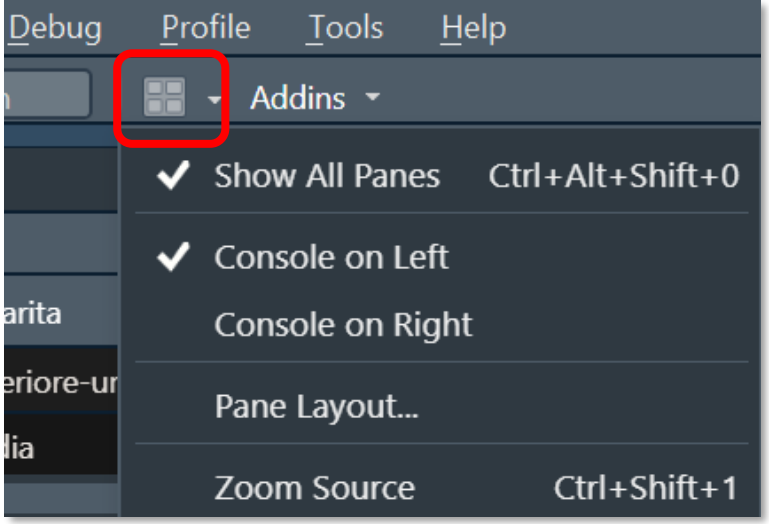
L'interfaccia grafica RStudio non fa parte di CRAN: **scaricate la versione free Desktop: https://www.rstudio.com/products/rstudio/download/**.

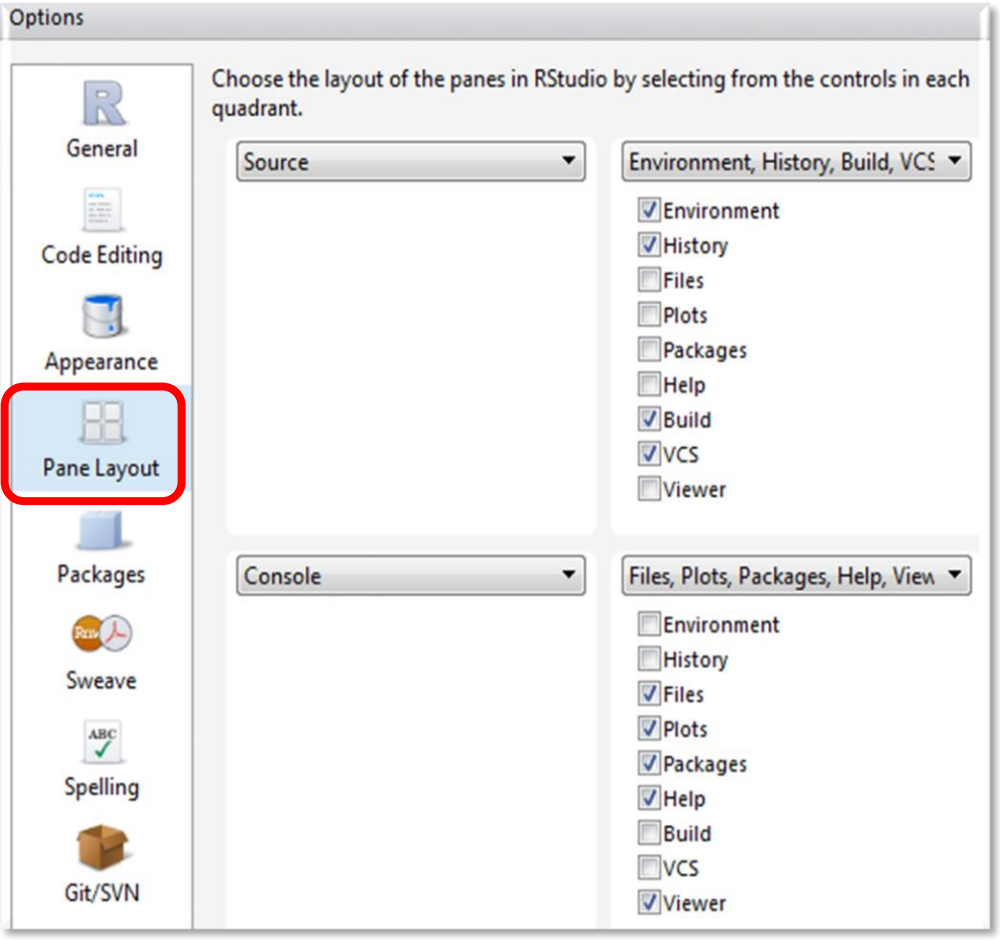
RStudio riconosce la versione di R che avrete installato sul vostro computer, e si terrà aggiornata. La forma – tipo è quadripartita:



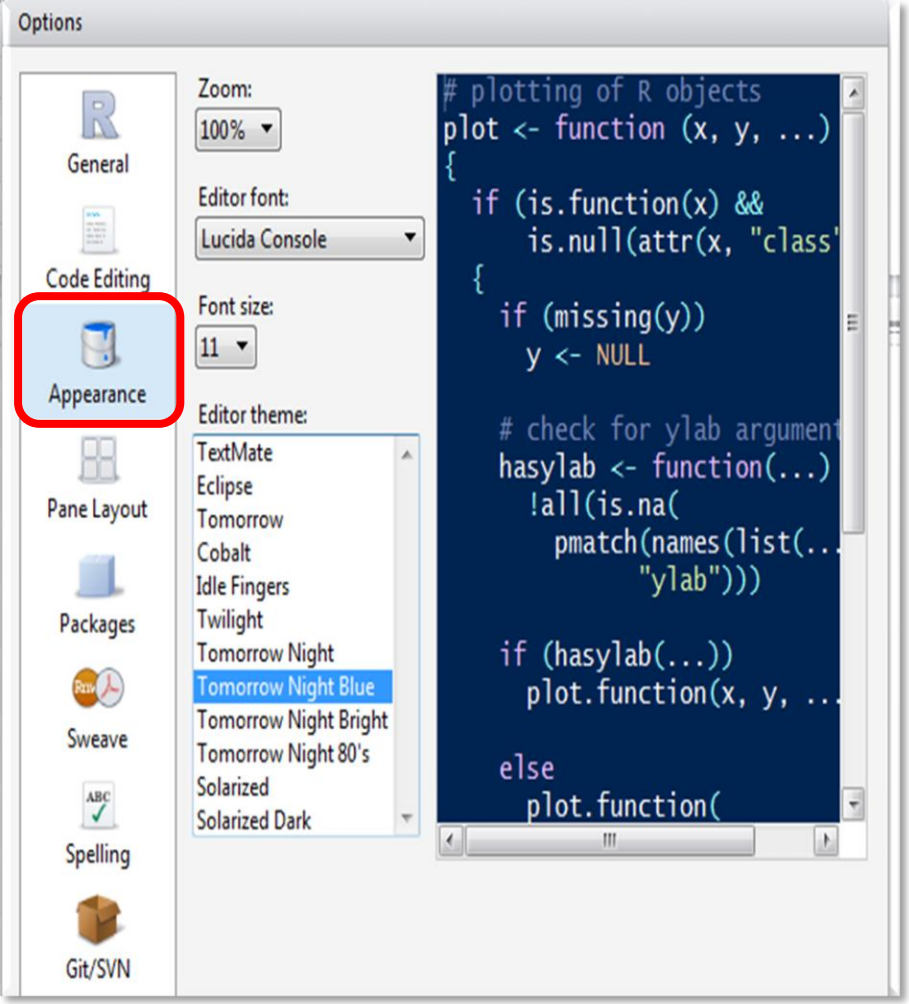
Con **l'icona** sulla barra dei menu…

… o nel Menu *Tools* → *Global Options* → *Panel Layout* è possibile specificare un layout diverso o ripristinarlo.

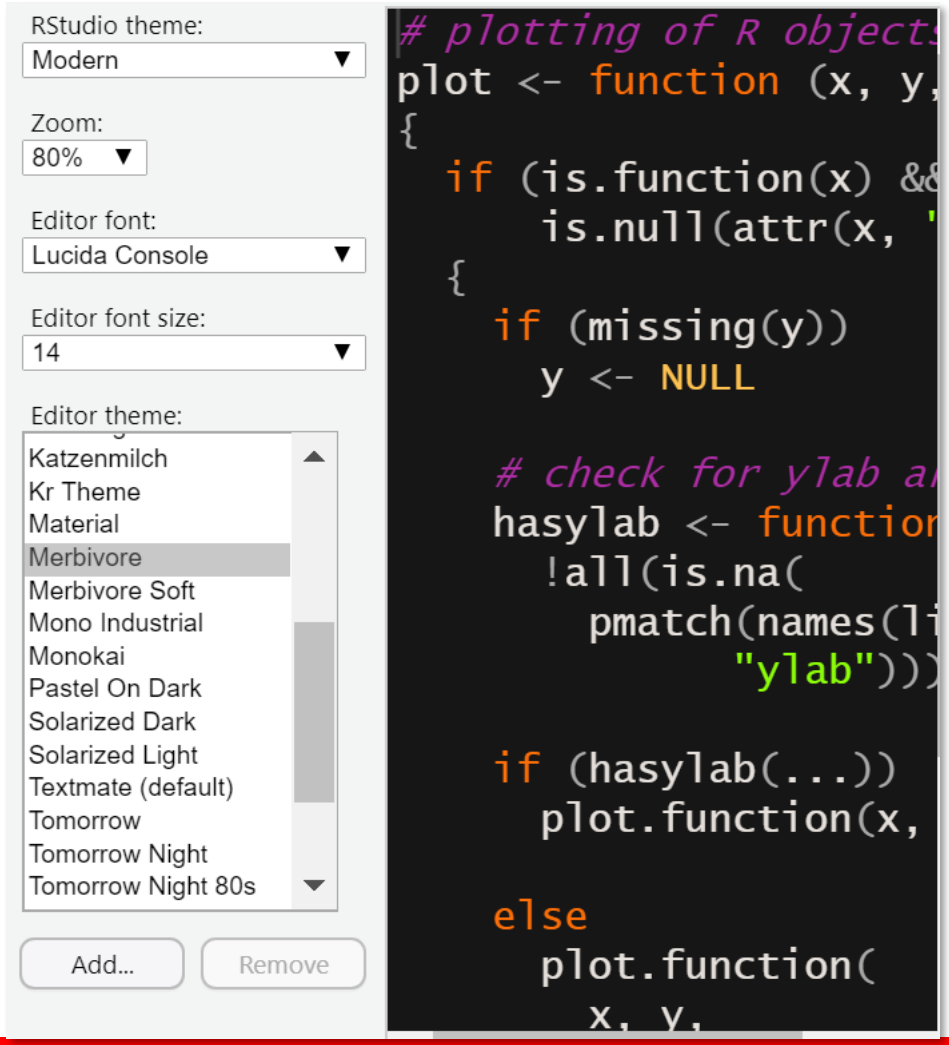




Le **ampiezze** orizzontali e verticali delle finestre possono essere aumentate o diminuite **posizionando il cursore sul telaio interno che separa le finestre e spostandolo**.

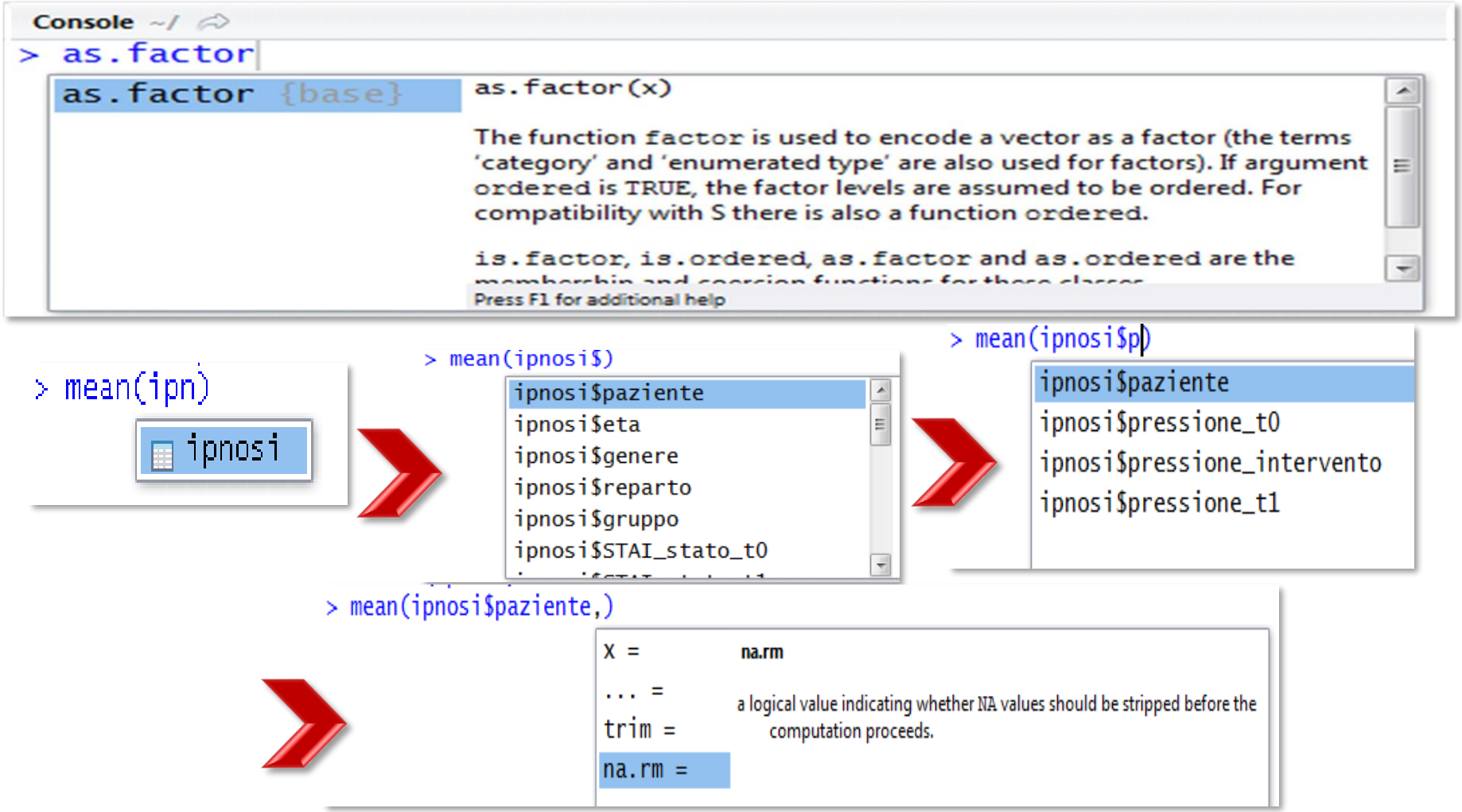


In *Tools→ Global Options→ Appearance* potete specificare zoom, font e varianti di colore-sfondo molto nerd.



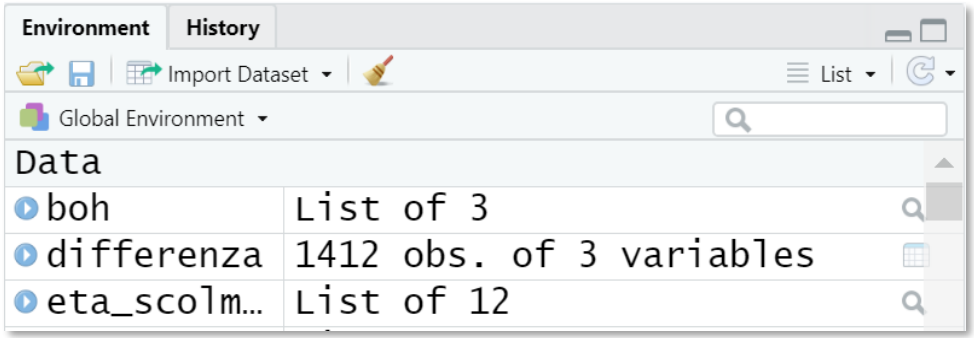
La console

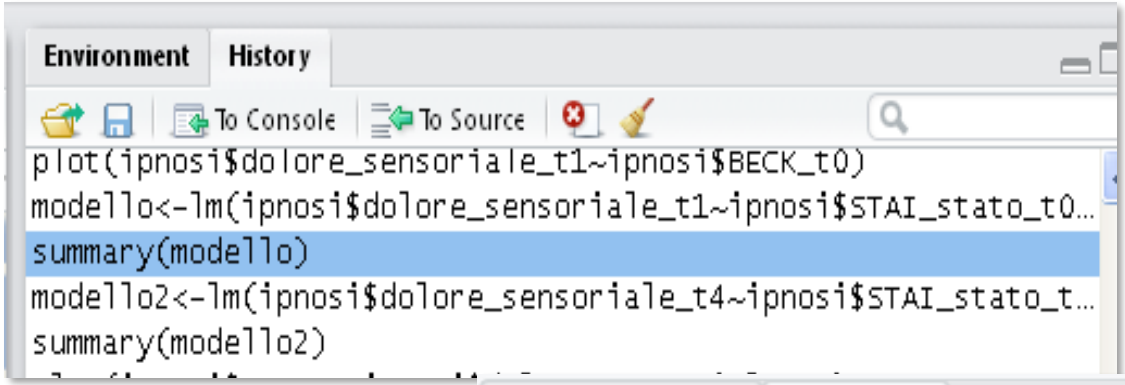
I suggerimenti, gli help e i completamenti sono più ricchi e sensibili al contesto; gli elementi in coppia come ( ) “ “ [ ] sono inseriti automaticamente digitando il primo.



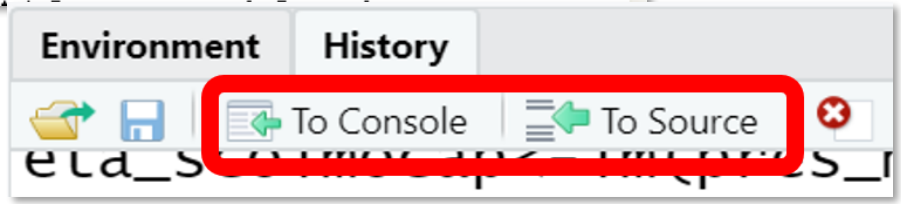
Environment e History

In `Environment` salviamo e importiamo dataframe e oggetti; cliccando sul triangolino, si ottiene una sintetica descrizione degli oggetti.





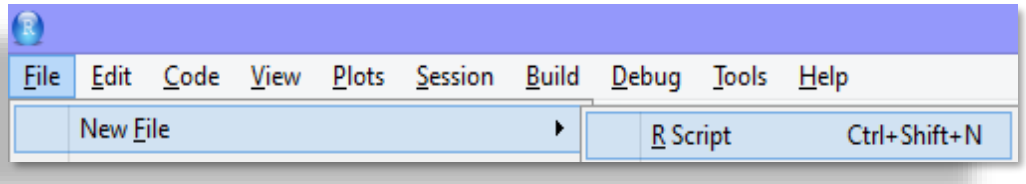
Nel foglio *History* vengono registrati, in ordine temporale, **tutti i comandi della sessione** digitati man mano nella Console,

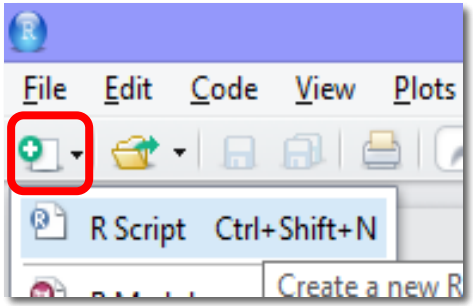


Ogni riga di comando può essere selezionata e richiamata in Console (To Console) o inviata in in **script** (To Source)

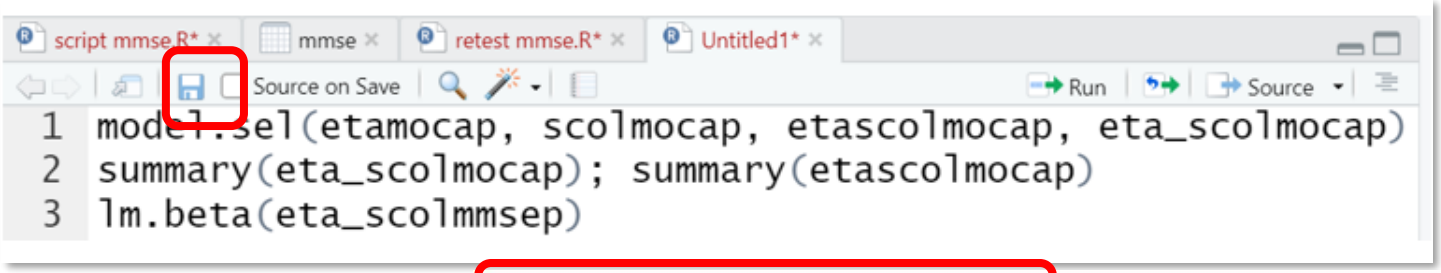
Script

Per creare nuovi script:

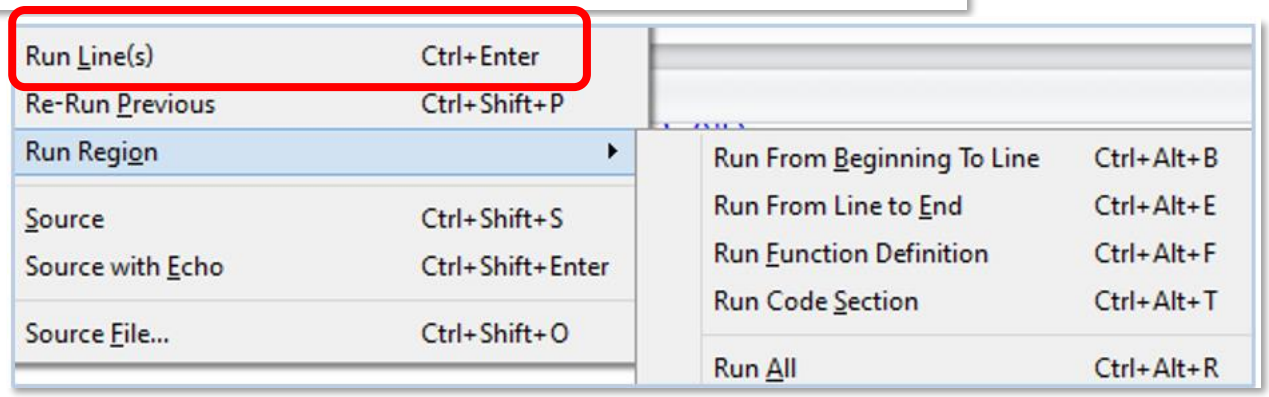




Ricordate di salvarlo!

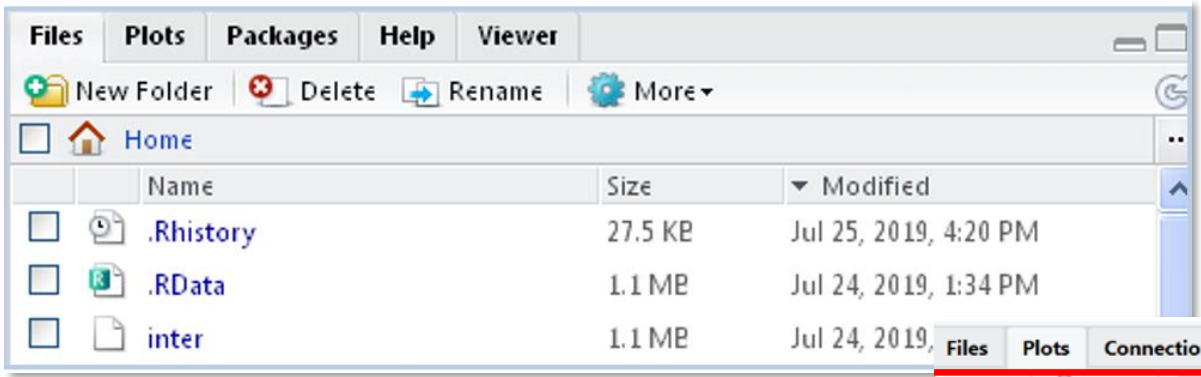


Per eseguirlo:



Potete anche selezionare il comando nello script e digitare **Crl+Invio**

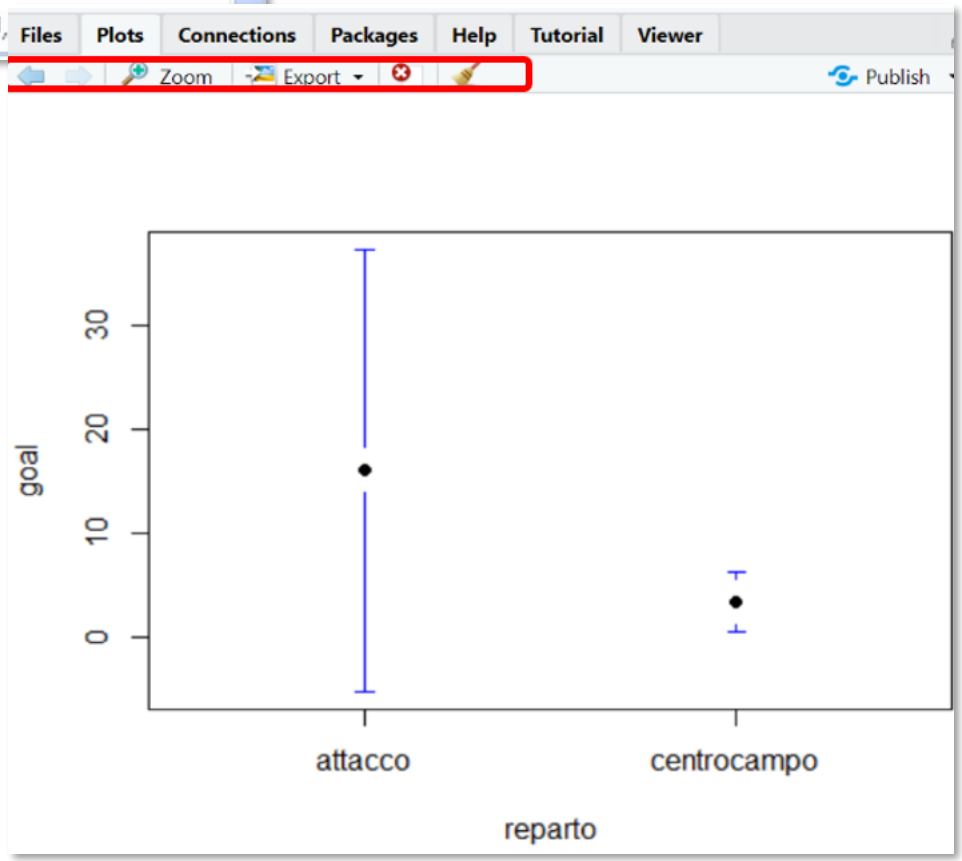
Miscellanea



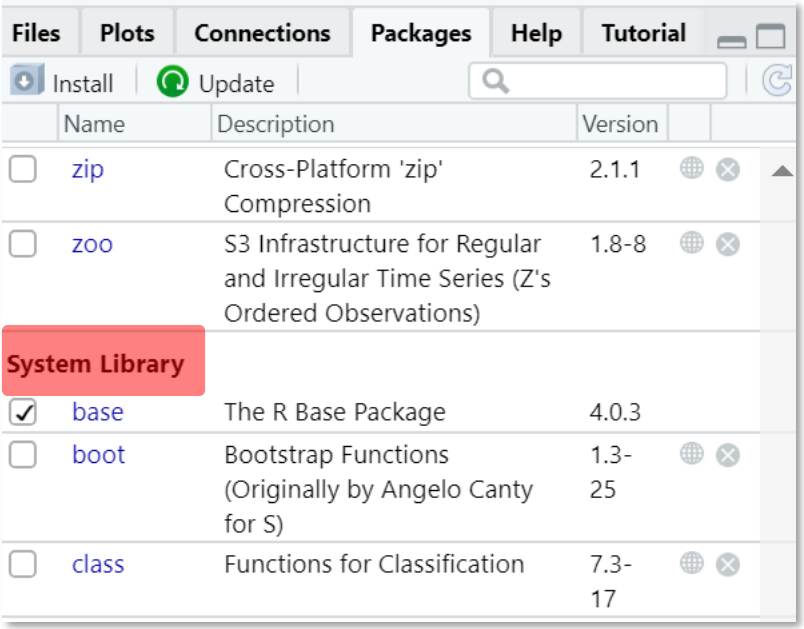
Files: cliccate per richiamarli

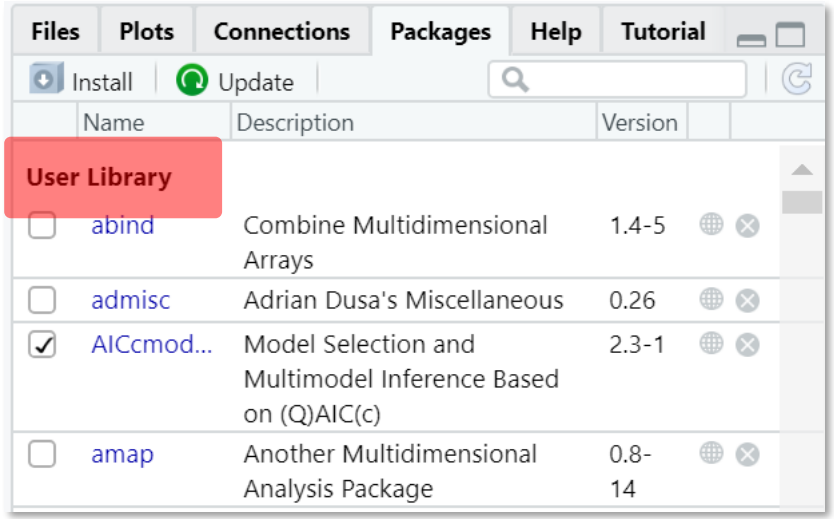
Plots: possono essere esportati (*Export*) e salvati come immagini (*Save as image*) o .pdf(*Save as PDF*), o copiati e incollati in altre applicazioni.

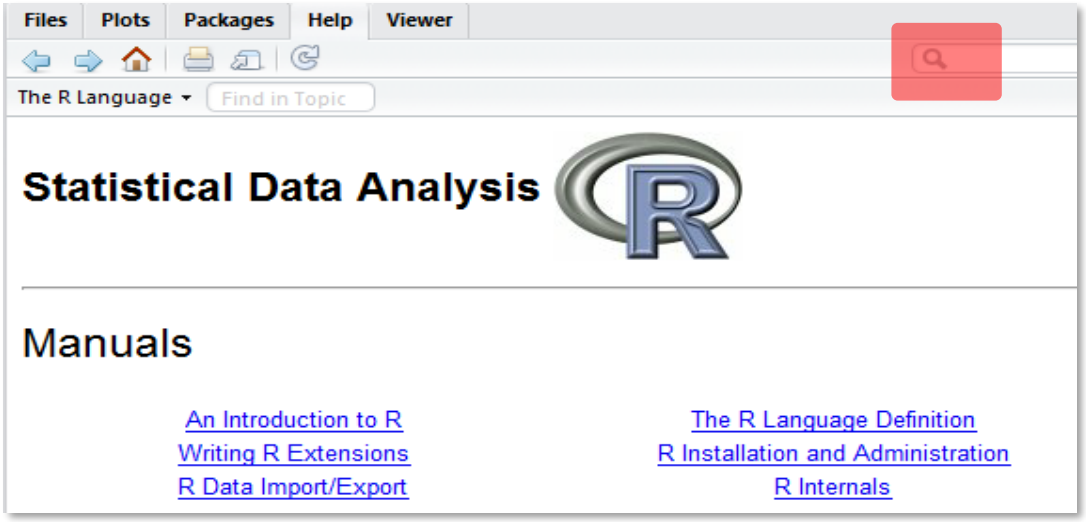
Se si creano più grafici nella stessa sessione, ci si sposta dall'uno altro usando le frecce



Packages: contiene l'elenco dei **packages** a disposizione; il pacchetto caricato per l'analisi ha un **segno di spunta**:





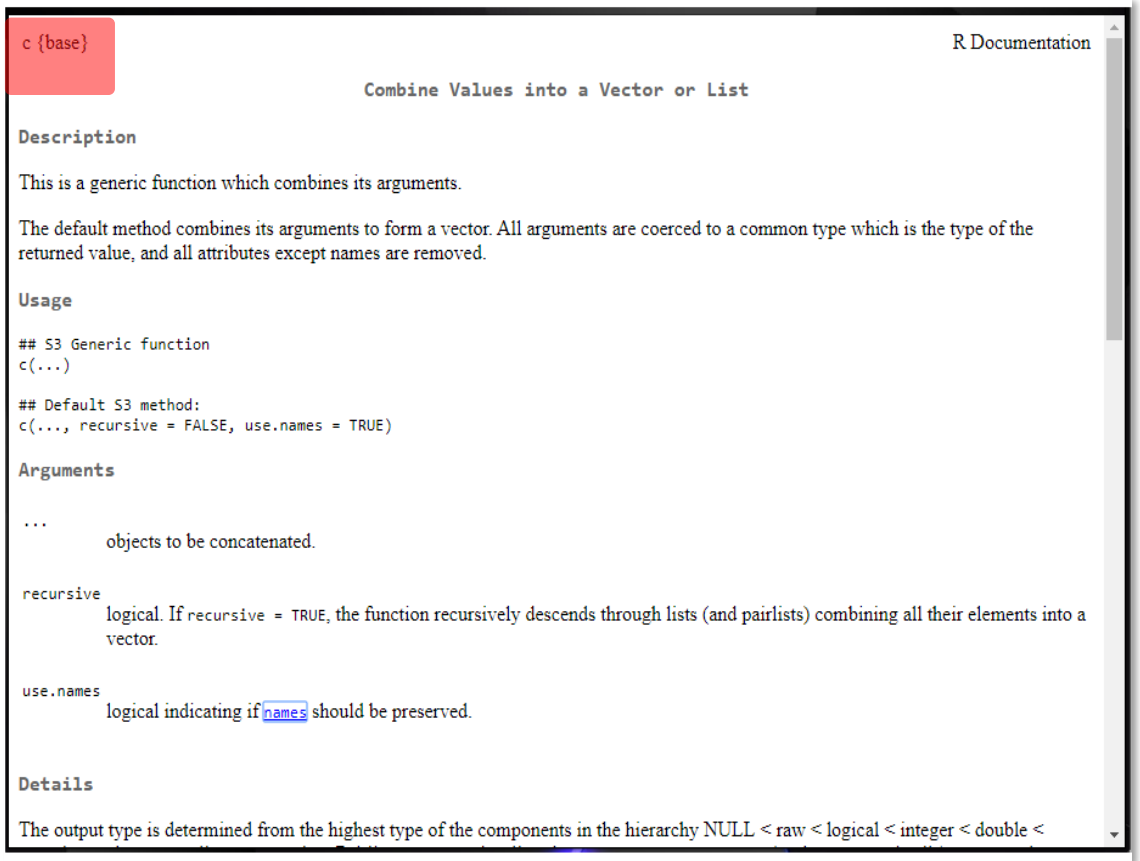


Help: la finestra essenziale.... ☺

Una incredibile **quantità di materiale informativo su R e le sue funzioni in generale** (tutorial, forum, articoli dedicati, blog, manuali), perlopiù in inglese ma non solo, è a disposizione sul **web** e accessibile con una **rapida ricerca per parole chiave** su Google o simili

Anche nella tradizionale interfaccia R c'è possibilità di avere aiuto:

Con `help(nome della funzione)` o `?(nome della funzione)` si ottengono le informazioni sugli scopi della funzione e sui modi per applicarla, che sono in realtà **parte del materiale che viene scaricato insieme alla funzione stessa**.



c {base} R Documentation

Combine Values into a Vector or List

Description

This is a generic function which combines its arguments.

The default method combines its arguments to form a vector. All arguments are coerced to a common type which is the type of the returned value, and all attributes except names are removed.

Usage

```
## S3 Generic function
c(...)

## Default S3 method:
c(..., recursive = FALSE, use.names = TRUE)
```

Arguments

`...`	objects to be concatenated.
`recursive`	logical. If `recursive = TRUE`, the function recursively descends through lists (and pairlists) combining all their elements into a vector.
`use.names`	logical indicating if `names` should be preserved.

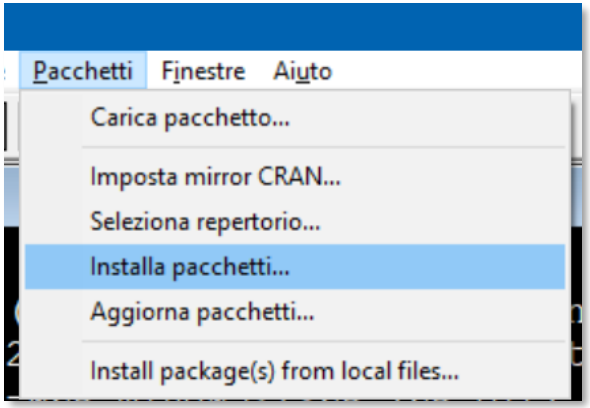
Details

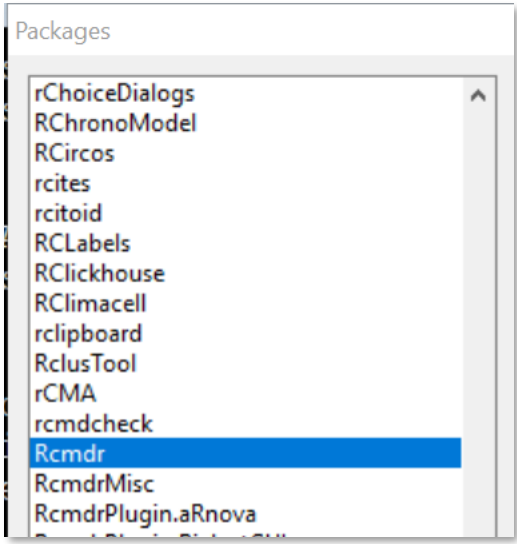
The output type is determined from the highest type of the components in the hierarchy NULL < raw < logical < integer < double <

RCommander

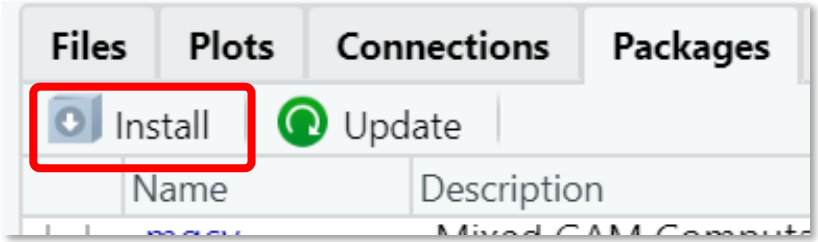
L'interfaccia RCommander è **contenuta nel package `Rcmdr`** scaricabile dal sito CRAN.

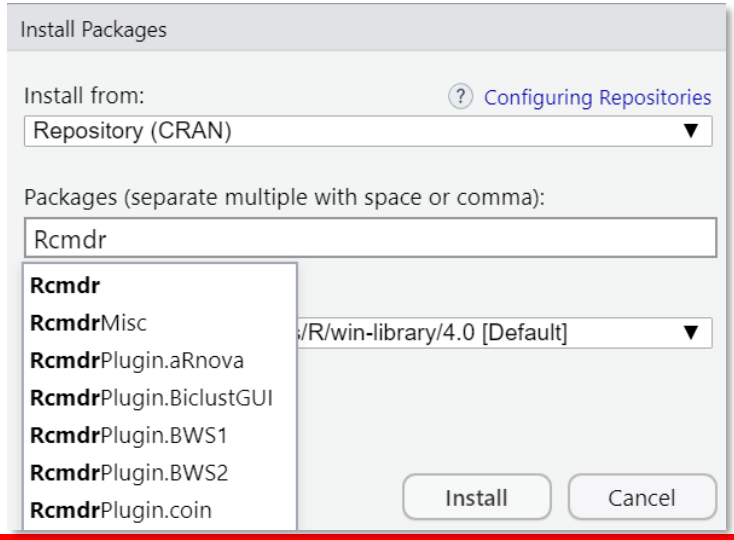
Per installarlo con **R**:





Per installarlo con **RStudio**:



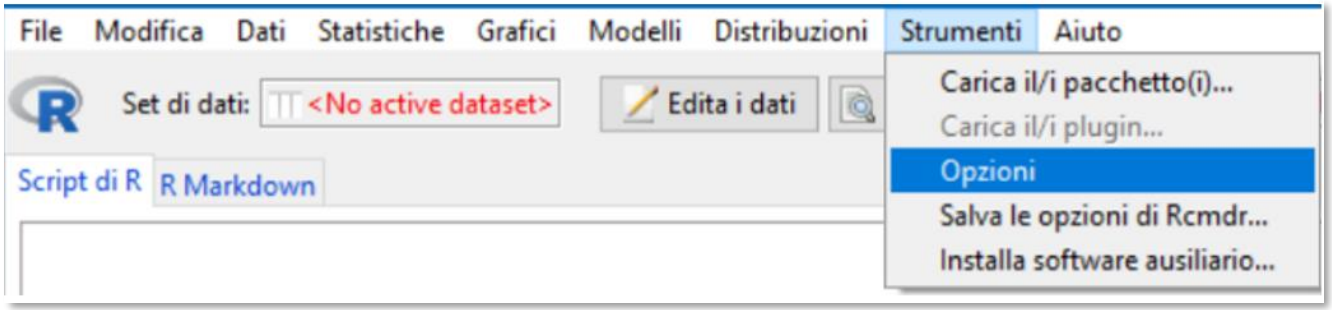


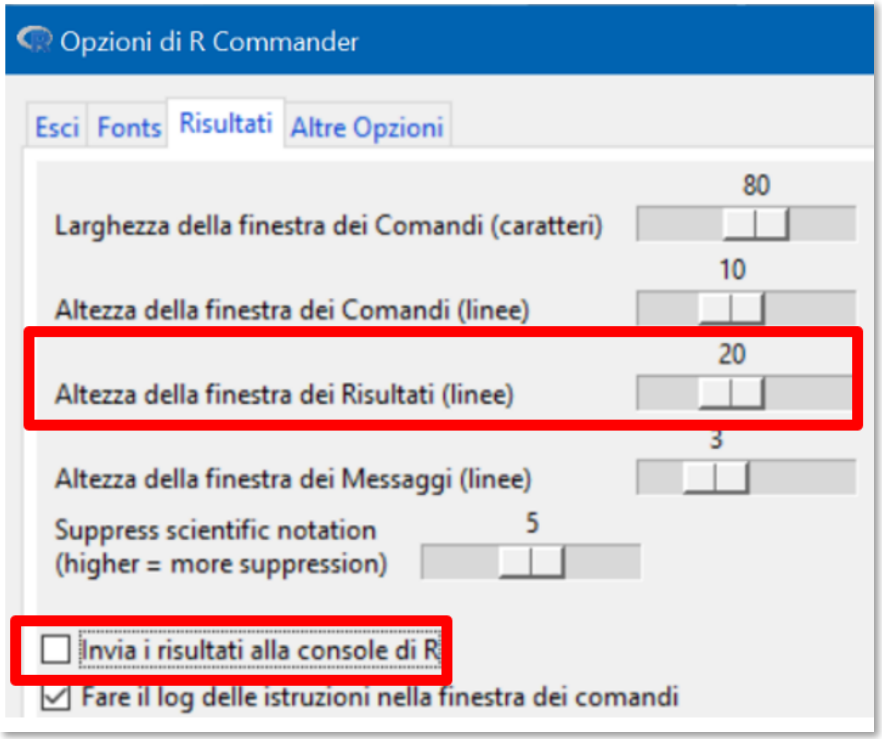
Accertatevi che sia spuntato: ☑ Install dependencies

Caricatelo **nella sessione di lavoro**: si apre una nuova finestra di R, a prima vista un po' deludente:

Se **vedete solo lo script**: gli output delle analisi e i grafici compaiono nella finestra di R o di RStudio, a seconda di quale abbiate aperto. Per vederli entro RCommander:

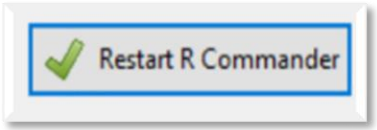
Menu
`Strumenti→Opzioni`



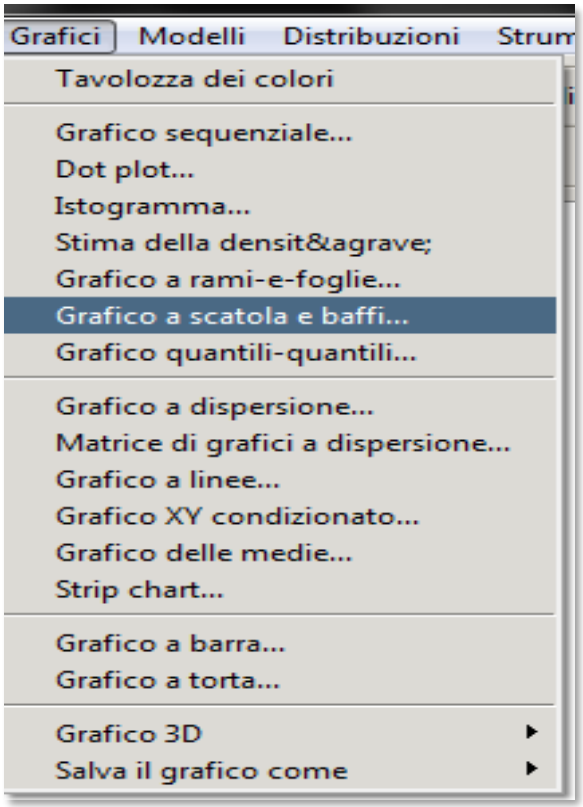


Deselezionate "`Invia i risultati alla console di R`", e definite l'altezza della finestra dei risultati, che troverete impostata a zero.

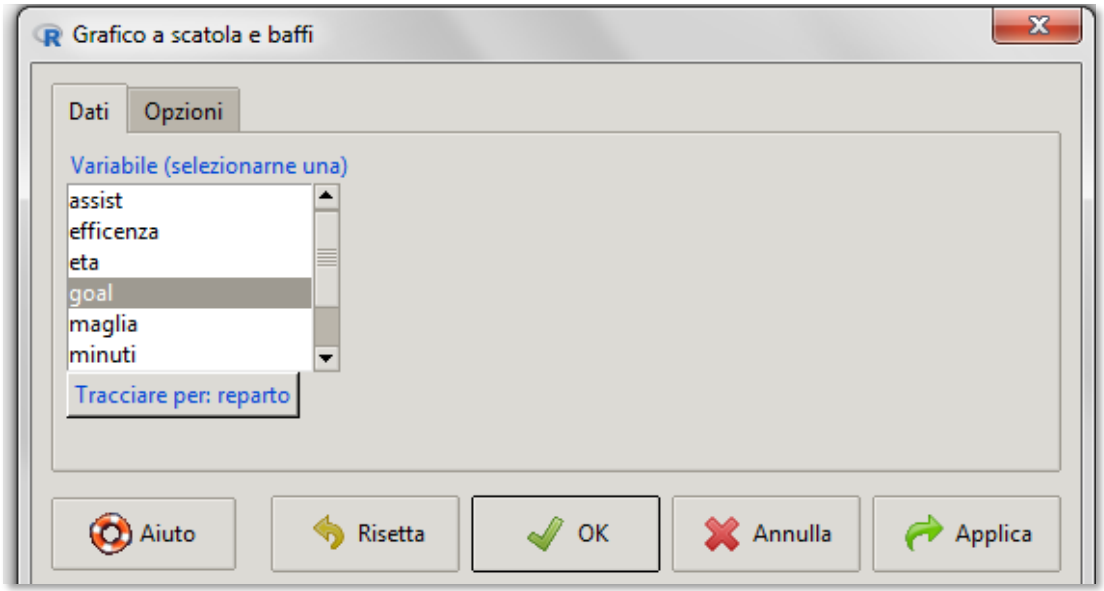
Cliccate su `Restart RCommander` in fondo alla scheda e, al riavvio, troverete la bipartizione della finestra.



Vedremo man mano che faremo operazioni su variabili e analisi come eseguirle con Rcommander.



Solo per vedere un esempio semplice: il menu **Grafici** consente di produrre praticamente tutti i grafici che useremo. Il grafico si apre in una finestra a parte e può essere salvato o copiato.



TUTTI i comandi eseguiti usando le finestre di dialogo sono stampati nella finestra degli script: ciascuno può essere salvato, modificato e arricchito a piacere nella finestra, ed eseguito (run) producendo un nuovo output.

Fare "cose" con R: oggetti e funzioni

AVVERTENZA PRELIMINARE IMPORTANTE

R è ***case sensitive***: considera una stessa lettera **maiuscola e minuscola come simboli diversi**.

Attenzione, quindi, **perché una delle situazioni più frustranti** è trovarsi di fronte a una serie di messaggi di errore in cui, nonostante voi siate convinti di scrivere il comando corretto, R vi dice che non capisce, perché avete scritto "**S**um" (fai la somma di) invece di **s**um, o "mean(**e**tà)" (fai la media della variabile età) quando il nome della variabile di cui volete la media è "**E**tà".

Tutto quello che vogliamo R faccia deve essere digitato come **comando**. I comandi sono, in genere, **strutturati in due parti**: gli **oggetti** e le **funzioni**

oggetto è **qualsiasi cosa sia creata in R**: una variabile, un modello, una serie di variabili. Può essere un singolo valore, o una serie d'informazioni (gli output delle analisi). Si distinguono per tipo (**class**).

Le **funzioni** sono le **cose che si fanno per creare o lavorare sugli oggetti**. Sono composte da **argomenti**, cioè **istruzioni**. Spesso servono più argomenti, **separati da ,** ciascuno dedicato a uno specifico aspetto.

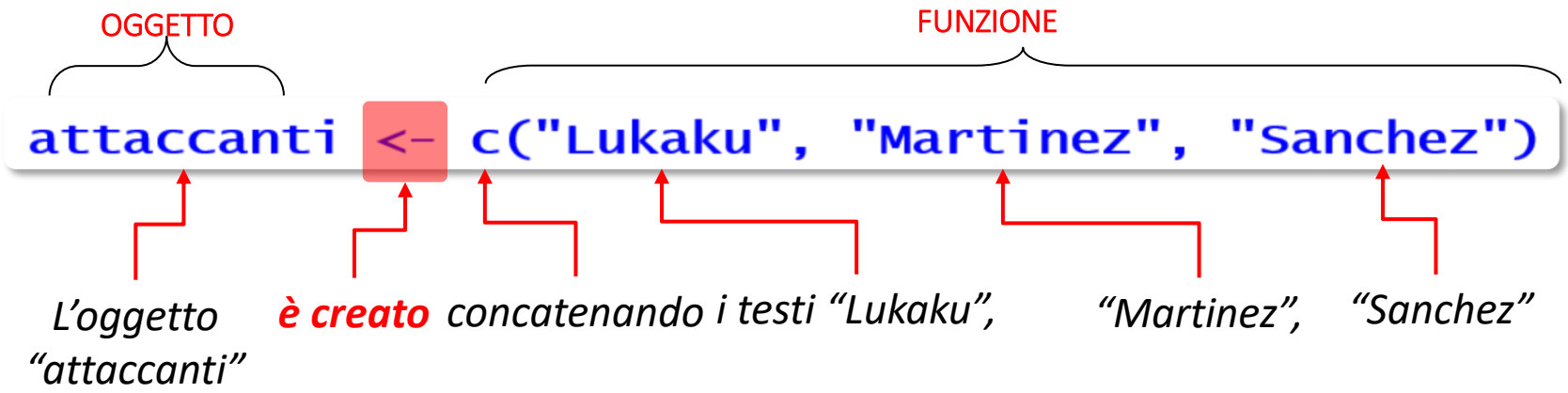
Oggetti e funzioni **sono legati <-** (**funzione di assegnazione**): "all'oggetto Y sono assegnate le proprietà della funzione X". Se preferite, potete pensarlo come **"l'oggetto è creato da" :**

oggetto <- funzione : "l'oggetto Y è creato dalla funzione X"

Vediamo un esempio usando una delle funzioni più semplici, ma più onnipresenti e importanti: la funzione **c** → *combine*

c unisce in un solo oggetto più elementi: possono essere **valori numerici** o **stringhe di testo**, purché **tutti dello stesso tipo**. La serie di elementi costituisce un **vettore**.

Costruiamo il **reparto di attacco della squadra Internazionale F.C** 2020-2021, unendo più elementi di testo, ciascuno dei quali rappresenta un argomento di **c**. Quando gli elementi del vettore sono **stringhe di testo**, devono essere tra **" "**



La natura degli elementi che costituiscono un oggetto **definisce il tipo (class) di oggetto / vettore / variabile.**

I più frequenti per noi:

`numeric`	gli elementi sono numeri decimali $\mathbb{R}$
`integer`	i valori sono numeri interi $\mathbb{Z}$
`factor`	ogni elemento identifica un diverso gruppo di unità di osservazione **variabili di raggruppamento**
`logic`	TRUE, FALSE
`character`	gli elementi sono stringhe di testo
`date`	data (aaaa/mm/gg)
`complex`	numeri complessi $\mathbb{C}$ con parti reali e immaginarie
Altri	`data.frame, matrix, lm, htest...`

Per verificare la classe:

```
class(oggetto)
```

```
>class(attaccanti)
[1] "character"
```

Per visualizzare l'oggetto: **oggetto** + **Invio**, oppure `print(oggetto)`, oppure `(oggetto<-funzione)`

```
attaccanti
[1] "Lukaku"   "Martinez" "Sanchez"
print(attaccanti)
[1] "Lukaku"   "Martinez" "Sanchez"
(attaccanti<- c("Lukaku","Martinez","Sanchez"))
[1] "Lukaku"   "Martinez" "Sanchez"
```

Per sapere di quanti elementi è composto l'oggetto: `length(oggetto)`:

```
length(attaccanti)
[1] 3
```

Per conoscere l'elemento maggiore e l'elemento minore: `max(oggetto)` e `min(oggetto)`

```
max(attaccanti)  min(attaccanti)
[1] "Sanchez"    [1] "Lukaku"
```

*Poiché attaccanti è di classe `character`, R restituisce il primo e l'ultimo elemento in ordine **alfabetico**; se fosse `numeric`, restituirebbe il valore più grande e il più piccolo.*

Si possono digitare **più comandi in un'unica riga**, separandoli con **;**

```
max(attaccanti); min(attaccanti)
[1] "Sanchez"
[1] "Lukaku"
```

Se si applica una **funzione incompatibile con la classe della variabile**, R si rifiuta:

```
> mean(attaccanti)
[1] NA
Warning message:
In mean.default(attaccanti) : l'argomento non è numerico o logico: restituisco NA
```

NA** (Not Available): **dato mancante

Si possono **richiamare i comandi** usando i tasti **Freccia su** △↑ (o **Freccia giù** ↓▽) della tastiera.

Modificare gli oggetti

Le funzioni si applicano a oggetti per **creare altri oggetti:**

```
sanchez<-max(attaccanti)
```

```
sanchez
[1] "Sanchez"
```

Possiamo correggere `attaccanti` **eliminando l'elemento** "Lukaku":

```
attaccanti_meno_lukaku<-attaccanti[attaccanti!= "Lukaku"]
```

L'oggetto `attaccanti_meno_lukaku` — *è creato da* — *l'oggetto `attaccanti`, in cui gli elementi di `attaccanti`* — *non devono essere uguali a* — *"Lukaku"*

Cioè: "usa `attaccanti` senza l'elemento che segue !=":**!=** sta per "**non uguale a**".

```
attaccanti_meno_lukaku
[1] "Martinez" "Sanchez"
```

Possiamo **aggiungere un nuovo elemento** all'oggetto con **c**:

```
attaccanti_meno_lukaku<-c(attaccanti_meno_lukaku, "Chilosà")

attaccanti_meno_lukaku
[1] "Martinez" "Sanchez" "Chilosà"
```

Torniamo allo status quo, reintegrando "Lukaku" e togliendo "Chilosà":

```
attaccanti<-c("Lukaku", attaccanti_meno_lukaku)
attaccanti<-attaccanti[attaccanti!="Chilosà"]
[1] "Lukaku" "Martinez" "Sanchez"
```

Le [ ] si usano per comandi riferiti alla **struttura di oggetti** compositi

Salvare o eliminare gli oggetti

Ogni tipo di **oggetto** creato **può essere salvato e restare disponibile in altre** sessioni, purché ci si ricordi di **salvare il workspace**, cioè l'area di lavoro che contiene tutti gli oggetti e le funzioni: **File→ Save workspace** in un file (R Image) **.RData**.

Riaprendo R, è caricato il workspace con cui si era conclusa la precedente sessione; si può cambiare usando **File→Load workspace**.

Salvate periodicamente quello che fate durante la sessione e all'uscita: **Crl+S**

Ricordate che potete anche usare gli **script** per salvare oggetti e funzioni.

Per **cancellare** gli oggetti non più utili, usate **`rm`** (*remove*): **`rm(oggetto)`**: attenzione, l'operazione **non è reversibile**!

Per conoscere quali oggetti sono stati salvati nel workspace attivo, usate **`objects()`**: non scrivete nulla tra parentesi, e ne avrete l'elenco.

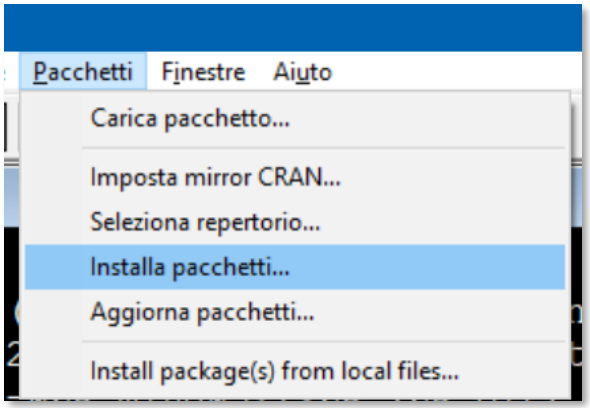
Aggiungere funzioni a quelle di base

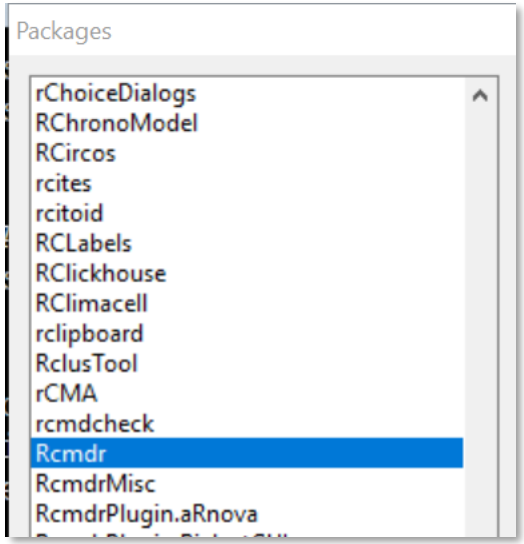
Le funzioni usate finora sono **comprese nei package di base** (`base, stats, lattice, MASS, utils`...) scaricati e installati con a R.. Per avere a disposizioni **altre** funzioni, vanno **scaricati i packages** (archivi compressi) che le contengono.

Per installarli con **R**: `install.packages("package", dependencies=TRUE)`

Oppure, abbiamo già visto:

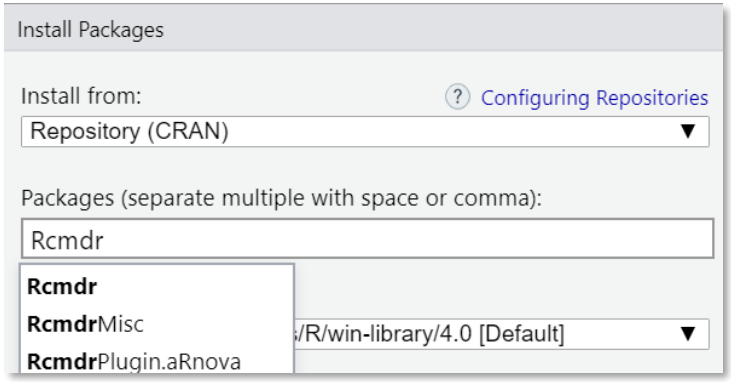
✓ In Windows: Menu **Packages** → **Install packages**: selezionate un *mirror* tra quelli elencati, poi il package (anche più d'uno) nell'elenco alfabetico



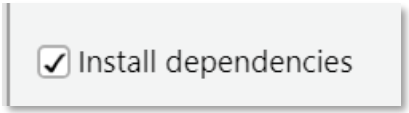


✓ In MacOS: **Packages & Data** → **Pages Installer**: cliccate su **List** e scegliete i package che vi servono, poi su **Install selected**.

Per installarli con **RStudio**:



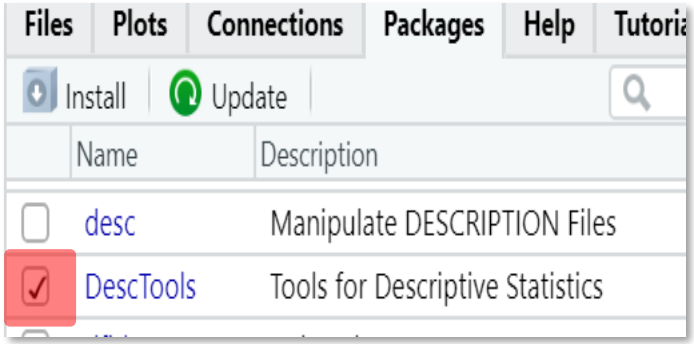
Accertatevi che sia spuntato:



Ogni package **va installato una sola volta**, ma deve essere **caricato nel workspace ogni volta** che se ne vuole usare una funzione.

Se non lo caricate, R dirà che non trova la funzione, anche se il package è correttamente installato: digitate `library(nome del package)`, oppure in Windows: **Packages**→ **Load package**, in MacOs **Packages & Data** → **Package Manager**.

In Rstudio, spuntate il nome del package nell'elenco:



Organizzare i dati: dataframe e matrici

I dati possono essere inseriti direttamente in R o importati da un fonte esterna.

Cominciamo con il vedere il primo caso, con pochi dati.

I dataframe

Arricchiamo le informazioni sul reparto di attacco dell'Inter 20-21 con la **variabile numerica** **Numero di goal** in campionato segnati dai tre attaccanti. **Niente virgolette**, sono numeri:

```
goal<-c(24,17,7)
```

```
class(goal)
[1] "numeric"
```

```
attaccanti
[1] "Lukaku" "Martinez" "Sanchez"
```

```
min(goal); max(goal); mean(goal)
[1] 7
[1] 24
[1] 16
```

```
> range(goal)
[1] 7 24
```

Attenti all'ordine, se vogliamo associarli ai rispettivi autori

Possiamo **combinare attaccanti e goal in una struttura** chiamata **dataframe**: un dataframe è un *array*, una struttura tabellare che può contenere **variabili anche di classe diversa.**

Usiamo `data.frame`: i suoi argomenti sono il **nome** scelto per identificare la variabile nel dataframe e il **vettore** che ne contiene i dati, **uniti da =,** per tutte le variabili.

```
oggetto_dataframe <- data.frame(nome variabile 1= vettore1, nome variabile 2 = vettore2,...)
```

Aggiungiamo un **argomento opzionale** che indica: "quando inserisci un vettore di classe `string` nel dataframe, **convertilo** in un oggetto di classe **`factor`**": `stringsAsFactors= TRUE / FALSE`. Di default, `stringsAsFactors` è `= FALSE`.

```
due_variabili<-data.frame(attaccante=attaccanti, goal=goal, stringsAsFactors = TRUE)
```

argomento `logic` : se `TRUE` esegue l'argomento, se `FALSE` non lo esegue.

```
due_variabili
  attaccante goal
1    Lukaku   24
2  Martinez   17
3   Sanchez    7
```

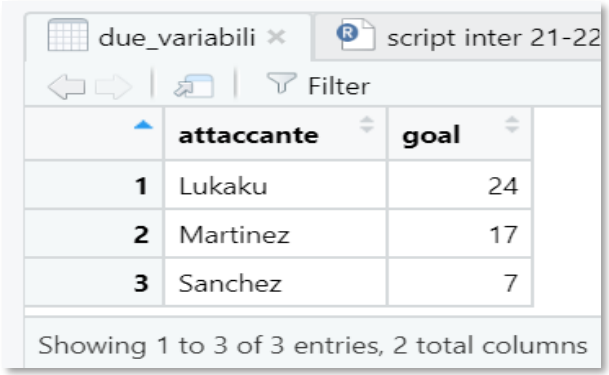
```
class(due_variabili)
[1] "data.frame
```

Ogni **riga** contiene le **osservazioni relative a uno specifico caso** e ogni **colonna** i **dati relativi a una sola variabile**: **wide format.**

*Useremo in Tecniche di analisi di dati II una diversa struttura in cui ogni riga corrisponde a una singola misura (**long format**)*

Per visualizzare il dataframe in una **finestra separata**:

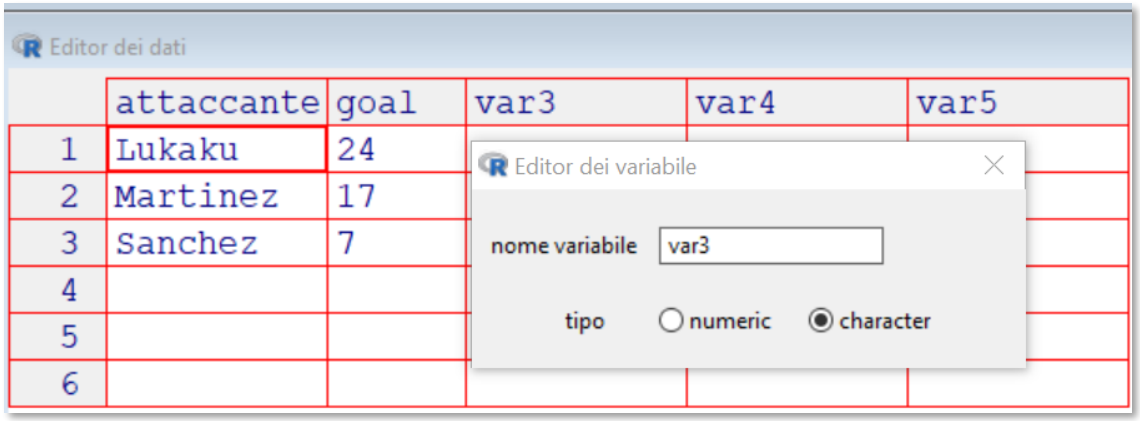
`View(dataframe)` .



Nelle versione di R precedenti alla 4.0.0, di default `stringAsFactors=TRUE`. Quindi, se state usando una versione vecchia di R, è inutile specificare l'argomento – ma aggiornate R

Modificare i dataframe

`fix(dataframe)` apre una **finestra di editing** in cui possono essere corretti i dati nelle singole celle, modificati i nomi e corretto il tipo delle variabili, aggiunte variabili.



Aggiungiamo la nuova variabile **`presenze`** in campionato: digitiamo il nome, specifichiamo il tipo e scriviamo i dati nelle cella corrispondenti, ottenendo:

```
due_variabili
  campione goal presenze
1   Lukaku   24       36
2 Martinez   17       38
3  Sanchez    7       30
```

Vogliamo **completare il dataframe** aggiungendo (dati ufficiali):

```
minuti<-c(2886,2576,1137)
```

```
assist<-c(14,6,5)
```

```
maglia<-c(9,10,7)
```

Non possiamo unire le tre variabili a `due_variabili` con **`c`**, che accetta solo **oggetti dello stesso tipo**, perché abbiamo:

```
class(due_variabili)
 [1] "data.frame"
```

```
class(minuti);class(assist); class(maglia)
[1] "numeric"
[1] "numeric"
[1] "numeric"
```

Invece **`data.frame`** non ha problemi a unire oggetti di classe diversa:

```
attacco<-data.frame(due_variabili,minuti,assist,maglia)
```

Data: attacco

	attaccante	goal	presenze	minuti	assist	maglia
1	Lukaku	24	36	2886	14	9
2	Martinez	17	38	2576	6	10
3	Sanchez	7	30	1137	5	7

Quando i vettori sono uniti in una struttura, oltre al **nome** acquistano un **"cognome", cioè il nome del dataframe**: "**cognome e nome**" sono separati da $:

Nome del dataframe

```
> mean(attacco$presenze)
[1] 27.5
```

Nome della variabile

I vettori uniti nel dataframe non sono stati eliminati, ed R li considera oggetti a sé:

un cambiamento in `attacco$presenze` non cambierà `presenze`

Cbind e rbind

cbind(vettore1,vettore2,…) e **rbind**(vettore1,vettore2,…) creano dataframe da vettori, matrici o altri dataframe, disponendoli **per colonna** o **per riga**. Gli elementi possono essere di classe diversa, purché abbiano lo stesso numero di colonne o di righe

*Creiamo i vettori A, B e C, composti da **numeri interi consecutivi**: A da 1 a 5, B da 10 a 14, C da 20 a 24. Invece di c, usiamo i **:** che indicano a R "**usa tutti i valori compresi tra… e…**":*

```
A<-(1:5)
B<-(10:14)
C<-(20:24)
```

```
cbind(A, B, C)
     A  B  C
[1,] 1 10 20
[2,] 2 11 21
[3,] 3 12 22
[4,] 4 13 23
[5,] 5 14 24
```

```
rbind(A, B, C)
  [,1] [,2] [,3] [,4] [,5]
A    1    2    3    4    5
B   10   11   12   13   14
C   20   21   22   23   24
```

```
attacco<-cbind(due_variabili,minuti, assist, maglia)
attacco
attaccante goal presenze minuti assist maglia
    Lukaku   24       36   2886     14      9
  Martinez   17       38   2576      6     10
   Sanchez    7       30   1137      5      7
```

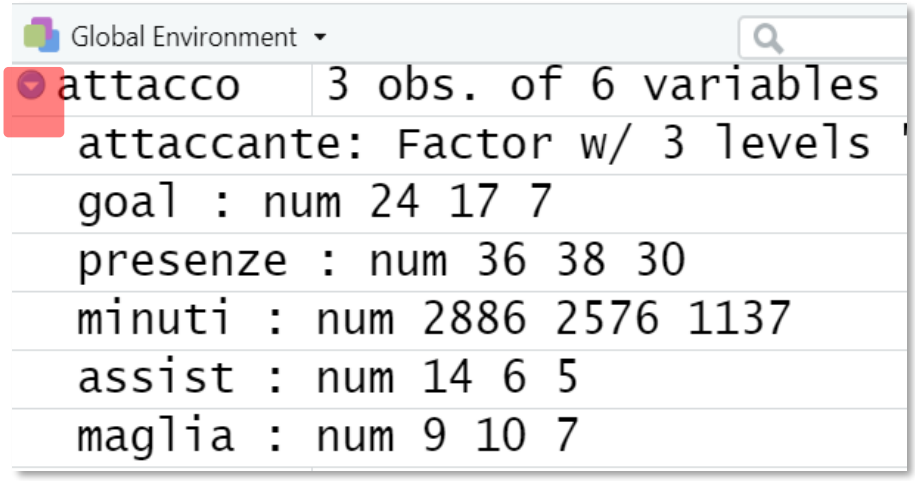
```
class(attacco)
[1] "data.frame"
```

Descrivere la struttura

`str(oggetto)` si applica a **più classi di oggetto**. Per un dataframe, `str` dà il **numero** di osservazioni e variabili, la loro **classe**, i **primi casi** di ogni variabile. In Rstudio, l'informazione di `str` è in **Global Environment**

```
str(attacco)
'data.frame':   3 obs. of  6 variables:
 $ attaccante: Factor w/ 3 levels "Lukaku","Martinez",..: 1 2 3
 $ goal      : num  24 17 7
 $ presenze  : num  36 38 30
 $ minuti    : num  2886 2576 1137
 $ assist    : num  14 6 5
 $ maglia    : num  9 10 7
```

Lukaku Sanchez Martinez



`$attaccante` **non deve essere** `factor` e `$maglia` non è veramente `numeric`: le correggeremo. Osserviamo che **l'ordine** dei livelli del `factor` corrisponde **all'ordine alfabetico** (Lukaku-Martinez-Sanchez→1-2-3)**.**

In generale, R **identifica i livelli secondo il loro ordine alfanumerico crescente**.

Descrivere con summary

`summary(oggetto)` è una funzione molto **versatile**: i suoi oggetti sono vettori, modelli statistici, ecc.

```
summary(attacco)
     attaccante          goal          presenze        minuti          assist          maglia
 Lukaku   :1     Min.   : 7.0   Min.   :30.00   Min.   :1137   Min.   : 5.000   Min.   : 7.000
 Martinez:1     1st Qu.:12.0   1st Qu.:33.00   1st Qu.:1856   1st Qu.: 5.500   1st Qu.: 8.000
 Sanchez :1     Median :17.0   Median :36.00   Median :2576   Median : 6.000   Median : 9.000
                Mean   :16.0   Mean   :34.67   Mean   :2200   Mean   : 8.333   Mean   : 8.667
                3rd Qu.:20.5   3rd Qu.:37.00   3rd Qu.:2731   3rd Qu.:10.000   3rd Qu.: 9.500
                Max.   :24.0   Max.   :38.00   Max.   :2886   Max.   :14.000   Max.   :10.000
```

Frequenza

Nei dataframe, `summary` dà le **frequenze** per variabili `character` o `factor`, e **statistiche** per le variabili `numeric`: **minimo** e **massimo**, il **1° e il 3° quartile**, **mediana** e **media**. Ripasseremo a breve quartili, mediane e medie.

Descrivere parti della struttura

Nella struttura un elemento è identificato dalla cella che occupa: **numero di riga** (*row*) **incrociato** con il **numero della colonna** (*column*). I comandi che riguardano strutture usano le parentesi quadre: all'interno delle **[ ]**, il **primo elemento indica la riga**, il **secondo la colonna**.

qual è il dato nella riga 3 ← → *incrociata con la colonna 2?*

Nel dataframe `attacco`,

```
attacco[3, 2]
[1] 7
```

	attaccante	goal	presenze	minuti	assist	maglia
1	Lukaku	24	36	2886	14	9
2	Martinez	17	38	2576	6	10
3	Sanchez	7	30	1137	5	7

Per informazioni relative a **più righe o colonne consecutive**, si usano i **:**

In `attacco`, *quali sono i dati delle righe comprese tra 1 e 2*

```
attacco[1:2, 2]
[1] 24 17
```

nella colonna 2?

In `attacco`, *quali sono i dati della riga 3*

```
attacco[3, 2:3]
  goal presenze
3    7       30
```

nelle colonne comprese tra 2 e 3?

Per dati in righe e/o colonne **non consecutive**, si usa **c:**

```
attacco[3, c (2, 3, 5)]
  goal presenze assist
3    7        30      5
```

Per operazioni su **tutte le righe** e/o su **tutte le colonne**, si **lascia libero lo spazio prima della virgola** ("tutte le righe") **o dopo la virgola** ("tutte le colonne"): "applica quanto richiesto a tutte le righe / a tutte le colonne".

Mostra i goal di tutti gli attaccanti

```
attacco[ ,6]
[1] 9 10 7
```

Mostra tutte le caratteristiche del solo Lukaku

```
attacco[1, ]
  attaccante goal presenze minuti assist maglia
1     Lukaku   24        36   2886     14      9
```

Per visualizzare tutti gli elementi di una colonna **tranne** quelli corrispondenti a una o più righe, oppure gli elementi di una riga **tranne** quelli corrispondenti a una o più colonne, o gli elementi dell'intera struttura, **tranne** gli elementi corrispondenti a una più righe e colonne, si antepone – al numero di riga / colonna da omettere.

```
attacco[-3, 4:5]
  minuti assist
1   2886     14
2   2576      6
```

```
attacco[c(-2,-3), 3:4]
  presenze minuti
1       36   2886
```

Liste e matrici

Altri tipi di oggetti che "contengono" vari altri oggetti sono:

- **liste**: create da `list(oggetto)`, possono contenere oggetti di classi diversa;
- **matrici**: create da `matrix(variabili, numero di righe, numero di colonne)`, devono essere composte da **variabili o solo numeriche o solo testuali**.

```
rendimento <- matrix(data=c(goal, assist), nrow=3, ncol=2)
```

rendimento è creato da una matrice che concatena i vettori *goal* e *assist*, fatta da 3 righe e 2 colonne

Possiamo aggiungere i **nomi delle variabili** con `fix`, oppure con `colnames(colonna)`; per le righe, useremmo `rownames(riga)`.

	V1	V2
1	24	14
2	17	6
3	7	5

```
colnames(rendimento) <- c("goal", "assist")
```

Come nomi delle colonne — *di rendimento* — *assegna* — *la concatenazione di "goal" e "assist"*

	goal	assist
1	24	14
2	17	6
3	7	5

*Per cambiare i nomi delle variabili di un **dataframe** si usa `names(oggetto)`, lo vedremo tra poco*

Se nella matrice si inseriscono erroneamente variabili di testo e numeriche, R **cambia tutte le variabili `numeric` in `character`**

```
A<-1:5
D<-c("uno","due","tre","quattro","cinque")
class(A);class(D)
[1] "integer"
[1] "character"
```

```
mista<-matrix(c(A,D),5,2)
mista
     [,1] [,2]
[1,] "1"  "uno"
[2,] "2"  "due"
[3,] "3"  "tre"
[4,] "4"  "quattro"
[5,] "5"  "cinque"
```

Il `tibble`, più recente, è usato in package aggiuntivi, ad esempio `tibble`: è un dataframe che perde alcune caratteristiche (per esempio, non si cambiano nome e classe delle variabili) e ne aggiunge altre che possono renderlo più "appetibile" per calcoli complessi.

```
ADX<-as_tibble(data.frame(A, D, X))
ADX
# A tibble: 5 x 3
      A D      X
  <int> <chr>  <date>
1     1 uno    2021-01-01
2     2 due    2021-02-02
3     3 tre    2021-03-03
4     4 quattro 2021-04-04
5     5 cinque 2021-05-05
```

Variabile di classe Date, che impareremo tra poco

I tibble mostrano solo le prime dieci righe e le colonne che si adattano alla dimensione della finestra; viene sempre mostrata la loro classe

Vedremo esempi di tibble in qualche funzione di Tecniche di Analisi di Dati II

Le differenze tra dataframe e matrici sono più profonde, ma per il livello richiesto nell'esame, ci fermiamo qui.

Lavorare su e con le variabili della struttura

Vedremo come cambiare il nome e la classe della variabile, le peculiarità delle variabili date e factor, la gestione dei dati mancanti e dei dati scorretti, le operazioni tra le variabili.

Rinominare le variabili

Possiamo cambiare i **nomi delle variabili** con `fix` o con **`names`**. **Da sola**, `names` restituisce i nomi degli elementi dell'oggetto; **associata a** ←, assegna i nomi agli elementi dell'oggetto.

*Per **conoscere** i nomi delle variabili di `attacco`:*

```
names(attacco)
[1] "campione" "goal"     "presenze" "minuti"   "assist"   "maglia"
```

Il nome `$attaccante` è ridondante rispetto al nome del dataframe: sono tutti attaccanti.

***Rinominiamo** questa variabile in "**campione**". Per **cambiare** i nomi delle variabili:*

```
names(attacco)<-c("campione","goal","presenze","minuti","assist","maglia")
```

Se si cambia il nome di **una** sola variabile, può essere comodo usare la **struttura** :

```
names(attacco)[1] <- "polpetta"
```

Come nome del primo elemento della struttura attacco

assegna

L'etichetta di testo "polpetta"

… ma torniamo a definizioni più consone:

```
names(attacco)<-
c("campione","goal","presenze","minuti","assist","maglia")
```

Cambiare la classe delle variabili

$attaccante e **$maglia** non sono della classe che ci aspetteremmo:

```
str(attacco)
'data.frame':   3 obs. of  6 variables:
 $ campione: Factor w/ 3 levels "Lukaku",
 …
 …
 $ maglia  : num  9 10 7
```

Per cambiare `factor` in `character` **dopo** aver creato il dataframe, usiamo `fix` oppure assegniamo alla variabile `as.*` : * è **sostituibile da qualsiasi classe**, come `as.numeric`, `as.character`, `as.factor`, `as.Date`, …

```
attacco$campione<-as.character(attacco$campione)
attacco$maglia <- as.character(attacco$maglia)
```

La variabile `$maglia` è creata trasformando in classe `character` la variabile `$maglia`

Ora:

```
str(attacco)
'data.frame':   3 obs. of  6 vari
 $ campione: chr  "Lukaku" "Marti
 $ goal    : num  24 17 7
 $ presenze: num  36 38 30
 $ minuti  : num  2886 2576 1137
 $ assist  : num  14 6 5
 $ maglia  : chr  "9" "10" "7"
```

Correggere dati errati

Per **correggere** **valori di cella** errati, si può usare `fix`, oppure assegnare il dato corretto indicando il valore da sostituire con le coordinate di riga – colonna.

Per esempio, se mai venisse finalmente riconosciuto a Lukaku quel goal che **non** era in fuorigioco, correggeremmo i suoi 24 goal scrivendo:

```
attacco[1,2]<-25
```

	campione	goal
1	Lukaku	25
2	Martinez	17
3	Sanchez	7

La classe `Date`

Se si devono fare **operazioni su date** (ad esempio, calcolare l'età dall'anno di nascita), è essenziale che la variabile sia correttamente codificata. Gli elementi di un vettore `Date` devono essere di tipo **aaaa/mm/gg**, e sono concatenati combinando **`as.Date`** e **`c`**:

```
nascita <- as.Date (c("1993/05/13", "1997/08/22", "1988/12/19"))
```

La variabile `nascita` è creata — *trasformando in classe `Date`* — *la concatenazione di questi elementi*

```
class(nascita)
[1] "Date"
```

Aggiungiamola al dataframe: `attacco$nascita<-nascita`

```
str(attacco)
'data.frame':   3 obs. of  7 variables:
 $ campione: chr  "Lukaku" "Martinez" "San
 $ goal    : num  24 17 7
 $ presenze: num  36 38 30
 $ minuti  : num  2886 2576 1137
 $ assist  : num  14 6 5
 $ maglia  : chr  "9" "10" "7"
 $ nascita : Date, format: "1993-05-13" …
```

Gestione dei dati mancanti NA

Quando una **cella non contiene un dato valido**, R lo sostituisce con **NA** (not available).

Se conoscessimo il numero di assist di tutti gli attaccanti **tranne Sanchez**, scriveremmo:

```
assist_mancanti<-c(14,6,NA)
```

Alcune funzioni gestiscono gli NA da sole, altre richiedono **istruzioni ad hoc**. Per esempio, chiedendo la **media degli assist con NA (`mean`)** senza specificare, otteniamo :

```
mean(assist_mancanti)
[1] NA
```

In `mean` va aggiunta l'istruzione di **togliere i dati mancanti prima di eseguire il calcolo**: si usa l'argomento `logic` **`na.rm =TRUE`** *[NA remove]*, in cui `TRUE` dice a R di rimuovere gli NA

```
mean(assist_mancanti, na.rm=TRUE)
[1] 10
```

Vedremo man mano come le diverse funzioni gestiscono i dati mancanti, ove necessario.

È possibile **convertire un dato errato in NA**: `dataframe[dataframe== valore] ← NA`

*Crediamo che solo 5 assist vincenti siano un risultato impossibile per un giocatore di serie A: potrebbe essere un **errore di digitazione**. Impossibilitati a recuperare il vero valore, lo consideriamo missing.*

attacco

campione	goal	presenze	minuti	assist
Lukaku	24	36	2886	14
Martinez	17	38	2576	6
Sanchez	7	30	1137	5

Verifichiamo quanti "5" e quanti "NA" siano presenti in `$assist` con `table(oggetto)`, che dà la **frequenza assoluta** degli elementi; con l'argomento opzionale **exclude=NULL**, R non ometterà alcun dato.

```
table(attacco$assist, exclude=NULL)
5  6 14
```

Usiamo `which(criterio)` *[chi è?..]* per sapere qual è la riga con assist == 5

```
which(attacco$assist==5)
[1] 3
```

Sostituiamo il dato della riga 3 colonna 55 con NA:

```
attacco[3,5]<-NA
```

Controlliamo: per sapere quali siano i casi con NA, si usa `which` associata alla funzione `is.na`, che **identifica gli NA**: `which(is.na(dataframe$variabile))`.

```
which(is.na(attacco$assist))
[1] 3
```

Possiamo anche riprovare `table`:

```
table(attacco$assist, exclude=NULL)
 6  14 <NA>
 1   1    1
```

```
table(attacco$assist)
 6 14
 1  1
```

Prima di proseguire con la teoria, un esercizio:

Create il dataframe "centrocampo" usando le seguenti informazioni:

*I **centrocampisti** sono: Hakimi, Perisic, Vidal, Barella e Brozovic; i loro **numeri di maglia** sono: 2, 14, 22, 23, 77. Hanno fatto rispettivamente 7, 4, 1, 3 e 2 **goal**. Hanno registrato 37, 32, 23, 36 e 33 **presenze** in campionato. Hanno giocato rispettivamente per 2672, 1800, 1142, 2900 e 2584 **minuti**, facendo 8, 4, 1, 7, e 6 **assist vincenti**. Sono **nati** nei seguenti giorni: 4/11/1998, 2/2/1998, 22/5/1987, 7/2/1997, 16/11/1992.*

Nel dataframe, la maglia di gioco e il nome del giocatore devono essere stringhe di testo.

*Nel crearlo, considerate che proseguiremo **unendo i due dataframe attacco e centrocampo** in un unico dataframe: questa informazione guiderà in qualche maniera i criteri che seguirete?*

Nelle slide successive illustreremo come fare questa unione tra dataframe, ma in realtà avete già tutti gli strumenti per sapere come si fa, senza andare a leggere: potete provare da soli?

Lo script per eseguire tutto quanto richiesto è in fondo alla dispensa, ma è inutile andarlo a vedere senza almeno provarci (e riprovarci, e riprovarci ☺)

Fare operazioni con le variabili

Avete creato il nuovo dataframe **centrocampo**: da qui lavoreremo sul dataframe **avanti**, composto da attacco e centrocampo uniti da **rbind:**

```
avanti<-rbind(attacco, centrocampo)
```

Data: attacco

	campione	goal	presenze	minuti	assist	maglia	nascita
1	Lukaku	24	36	2886	14	9	1993-05-13
2	Martinez	17	38	2576	6	10	1997-08-22
3	Sanchez	7	30	1137	NA	7	1988-12-19

Data: centrocampo

	campione	goal	presenze	minuti	assist	maglia	nascita
1	Hakimi	7	37	2672	8	2	1998-11-04
2	Perisic	4	32	1800	4	14	1998-02-02
3	Vidal	1	23	1142	1	22	1987-05-22
4	Barella	3	36	2900	7	23	1997-02-07
5	Brozovic	2	33	2584	6	77	1992-11-16

Data: avanti

	campione	goal	presenze	minuti	assist	maglia	nascita
1	Lukaku	24	36	2886	14	9	1993-05-13
2	Martinez	17	38	2576	6	10	1997-08-22
3	Sanchez	7	30	1137	NA	7	1988-12-19
4	Hakimi	7	37	2672	8	2	1998-11-04
5	Perisic	4	32	1800	4	14	1998-02-02
6	Vidal	1	23	1142	1	22	1987-05-22
7	Barella	3	36	2900	7	23	1997-02-07
8	Brozovic	2	33	2584	6	77	1992-11-16

```
str(avanti)
'data.frame':   8 obs. of  7 variables:
 $ campione: chr  "Lukaku" "Martinez" "Sanchez" "Hak
 $ goal    : num  24 17 7 7 4 1 3 2
 $ presenze: num  36 38 30 37 32 23 36 33
 $ minuti  : num  2886 2576 1137 2672 1800 ...
 $ assist  : num  14 6 NA 8 4 1 7 6
 $ maglia  : chr  "9" "10" "7" "2" ...
 $ nascita : Date, format: "1993-05-13" "1997-08-22"
```

Va da sé che per funzionare le variabili dei due dataframe devono essere coerenti tra loro….

Creare variabili factor

Attaccanti e centrocampisti vanno assegnati al **proprio reparto**: creiamo una **variabile** `factor`, con due livelli corrispondenti ai reparti. I livelli dei `factor` sono **numeri**, cui possiamo assegnare **etichette**. Potremmo **generare la sequenza**, **correttamente ordinata**, delle **etichette dei livelli** del fattore usando `c`: 3 "attacco" e 5 "centrocampo".

```
reparto<-c("attacco","attacco","attacco","centrocampo","centrocampo","centrocampo",
 "centrocampo","centrocampo")
```

Ma è meglio: `rep(x= cosa da replicare, times= numero di repliche)`:

```
reparto<-c(rep("attacco",3),rep("centrocampo",5))
```

```
class(reparto)
[1] "character"
```

```
reparto<-as.factor(reparto)
```

Più rapidamente:

```
reparto <- as.factor(c(rep("attacco", 3), rep("centrocampo", 5)))
```

```
str(reparto)
 Factor w/ 2 levels "attacco","centrocampo": 1 1 1 2 2 2 2 2
```

D'ora in poi **creeremo l'oggetto direttamente come variabile** del dataframe:

```
avanti$reparto<- as.factor(c(rep("attacco", 3), rep("centrocampo", 5)))
```

Creare o modificare variabili da variabili esistenti

Ogni vettore può essere trasformato **associandolo a un altro vettore**, o a un **unico valore scalare**, tramite **operatori matematici (+, -, *, /,** ecc.) e/o gli **operatori logici**.

Operatore	Cosa fa	Operatore	Cosa fa
+	Somma oggetti	>=	Maggiore o uguale a
-	Sottrae oggetti	==	**Esattamente uguale a**
*	Moltiplica oggetti	!=	**Non uguale a**
/	Divide oggetti	!x	**Non *x***
^ (o **)	Esponente	x | y	**O** / **VEL**: l'operazione fornisce i valori che soddisfano la caratteristica y o la caratteristica y, ma anche entrambe,
<	Minore di	x & y	**E** / **AND**: l'operazione fornisce i valori che soddisfano sia la caratteristica X sia la caratteristica Y, ma non quelli che hanno una sola delle due.
<=	Minore o uguale a	isTRUE(x)	Verifica se x è TRUE
>	Maggiore di		

Creiamo **$rendimento**, **somma** delle azioni finalizzate con successo: **goal + assist vincenti**.

```
avanti$rendimento <- avanti$goal + avanti$assist
avanti$rendimento
[1] 38 23 12 15  8  2 10  8
```

Con molte colonne da sommare, si usa `rowSums(dataframe[ ,colonne da sommare])`:

```
avanti$rendimento <- rowSums(avanti[,c(2,5)])
[1] 38 23 12 15  8  2 10  8
```

Visualizziamo **solo i primi** - `head(variabile, n=)` e **gli ultimi** - `tail(variabile, n=)`

casi; con `n=` specifichiamo quanti (di default **n=6**):

```
head(avanti$goal,n= 3);head(avanti$assist,3);head(avanti$rendimento,3)
[1] 24  17  7 ← goal
[1] 14   6  5 ← assist
[1] 38  23 12 ← efficienza
```

```
tail(avanti$goal,n=3);tail(avanti$assist,3);tail(avanti$rendimento,3)
[1] 1  3  2
[1] 1  7  6
[1] 2 10  8
```

Facciamo un'operazione **tra un vettore e uno scalare**, passando da minuti a **ore di gioco**:

```
avanti$ore_gioco<-avanti$minuti / 60
head(avanti$minuti,3); head(avanti$ore_gioco,3)
[1] 2886 2576 1137                ← minuti
[1] 48.10000 42.93333 18.95000    ← ore gioco
```

Rapportiamo il rendimento alle ore di gioco, creando `$efficienza`

```
avanti$efficienza<-avanti$rendimento / avanti$ore_gioco
```

Arrotondiamo a due decimali: **`round(x=oggetto, digits=numero di decimali)`**:

```
round(head(avanti$efficienza,3),2)
[1] 0.79 0.54 0.63
```

Attenti alle parentesi.....

```
max(avanti$efficienza); min(avanti$efficienza)
[1] 0.7900208
[1] 0.1050788
```

```
which(avanti$efficienza>.79); which(avanti$efficienza<.11)
[1] 1
[1] 6
```

```
avanti[c(1,6),1]
[1] "Lukaku" "Vidal"
```

Usiamo `$nascita` per ottenere **l'età**, calcolabile come **quantità di giorni trascorsi da quello natale alla data di fine stagione**, che creiamo; dividiamo $età per 365, per averla in anni:

```
data_fine_stagione<-as.Date("2021/05/30")
```

```
avanti$eta<-(data_fine_stagione-avanti$nascita)/365
```

```
round(avanti$eta)
Time differences in days
[1] 28 24 32 23 23 34 24 29
```

Il default per `round` è `digits= 0`

`$eta` è di classe `difftime`, differenza tra epoche, convertiamola in `numeric`:

```
avanti$eta<-as.numeric(avanti$eta)
```

Selezionare parti della struttura

Soprattutto in dataframe con molti soggetti e/o molte variabili sarà necessario **estrarre parti di un dataframe** per lavorare solo su alcuni casi o su alcune variabili. Si può fare in vari modi.

Si può usare la struttura `[riga,colonna]`:

```
nuovo_dataframe ← vecchio_dataframe[righe, colonne]
```

```
centro <-  avanti [4:8 , ]
```

Il datafame `centro` *è creato dal dataframe* `avanti` *di cui seleziona le righe da 4 a 8* ***e tieni tutte le colonne***

Se non viene specificato nulla dove R aspetta istruzioni su righe o colonne, R le **importa tutte** nel nuovo oggetto.

Solo gli attaccanti:

```
att<-avanti[avanti$reparto=="attacco",]
```

Goal e presenze dei centrocampisti:

```
goal_presenze_centrocampo<-avanti[4:8,2:3]
```

Si può usare **subset: nuovo dataframe← subset(x=vecchio dataframe, subset= casi da esportare, select = c (colonna/e da esportare)).**

`subset=` estrae le righe, `select=` (opzionale) le colonne: di default, sono esportate tutte. I casi si selezionano con gli **operatori logici**. In `subset=` e `select=` si può omettere il nome del dataframe prima della variabile, perché è già indicato in `x=`.

Nomi e goal di chi ha segnato almeno 5 goal

```
solo_5_goal<-subset(x= avanti, subset= goal>=5,
    select=c(campione, goal))
```

Solo i centrocampisti con più attitudine al goal

```
centrocampisti_pungenti<-subset(x= avanti, subset= reparto
  == "centrocampo"&goal>=5,select=c(campione,goal,reparto)
```

I più giovani o quelli che hanno segnato poco

```
giovani_o_scarsi<-subset(x= avanti, subset= eta<25|goal<=4,
  select=c(campione, goal, eta))
```

Solo gli attaccanti, per esclusione

```
attaccanti_in_negativo<-subset(x=avanti,subset=
  reparto!="centrocampo", select=c(campione, reparto))
```

Data: solo_5_goal

	campione	goal
1	Lukaku	24
2	Martinez	17
3	Sanchez	7
4	Hakimi	7

Data: centrocampisti_pungenti

	row.names	campione	goal
1	4	Hakimi	7

Data: attaccanti_in_nega...

	campione	reparto
1	Lukaku	attacco
2	Martinez	attacco
3	Sanchez	attacco

Data: giovani_o_scarsi

	row.names	campione	goal	eta
1	2	Martinez	17	23.78630 days
2	4	Hakimi	7	22.58356 days
3	5	Perisic	4	23.33699 days
4	6	Vidal	1	34.04658 days
5	7	Barella	3	24.32329 days
6	8	Brozovic	2	28.55342 days

Per qualsiasi funzione

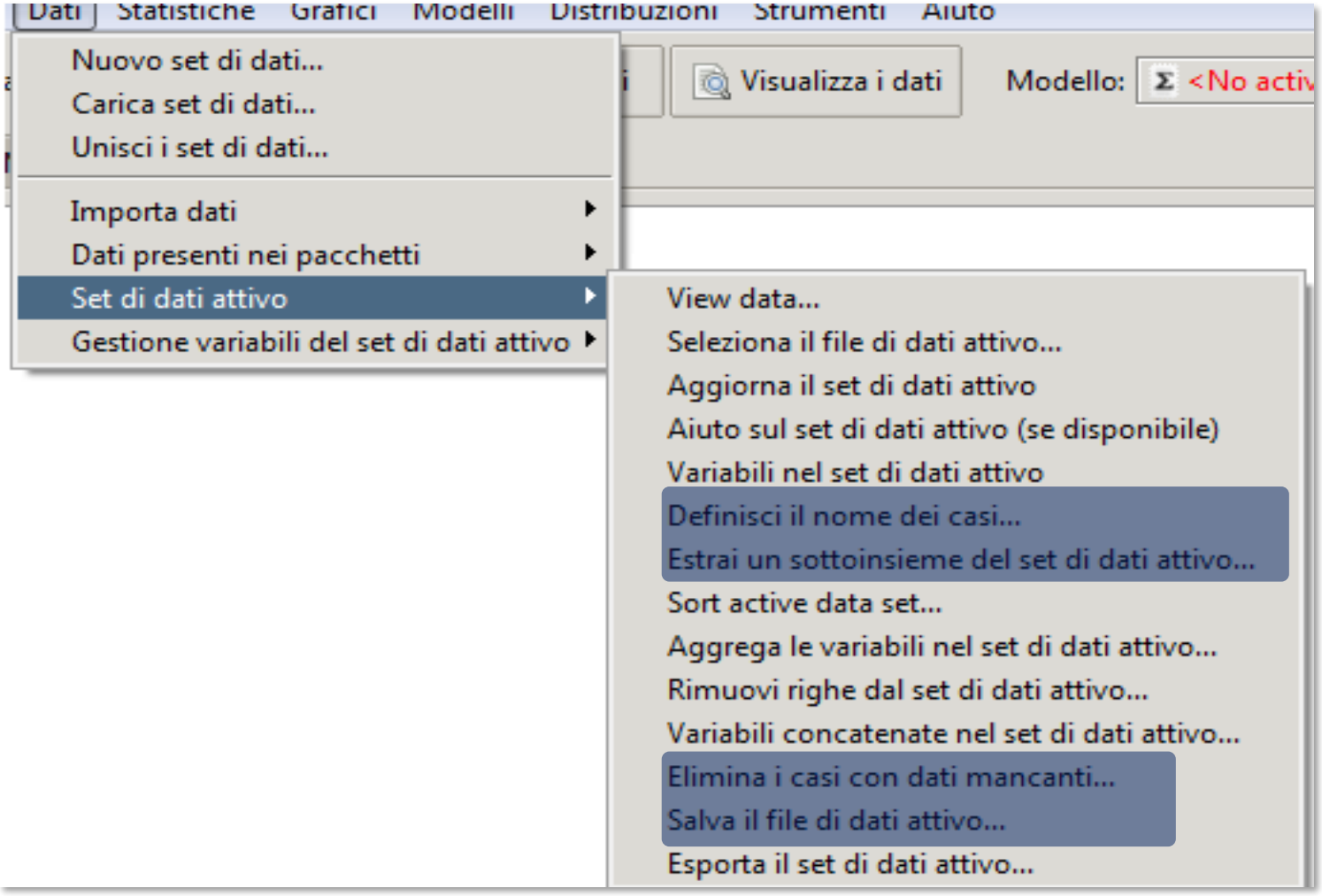
Se omettete il nome dell'argomento (x=, subset=, ecc.), R si aspetta che gli **argomenti si succedano nell'ordine previsto dalla funzione** (nel caso di subset, prima il dataframe, poi il criterio di selezione delle righe, infine il criterio di selezione delle colonne).

Se gli argomenti non rispettano quest'ordine, R non trova quello che si aspetta e restituisce un errore o un output insensato.

Invece, **se indicate il nome dell'argomento, potete inserirli nella funzione in qualsiasi ordine.**

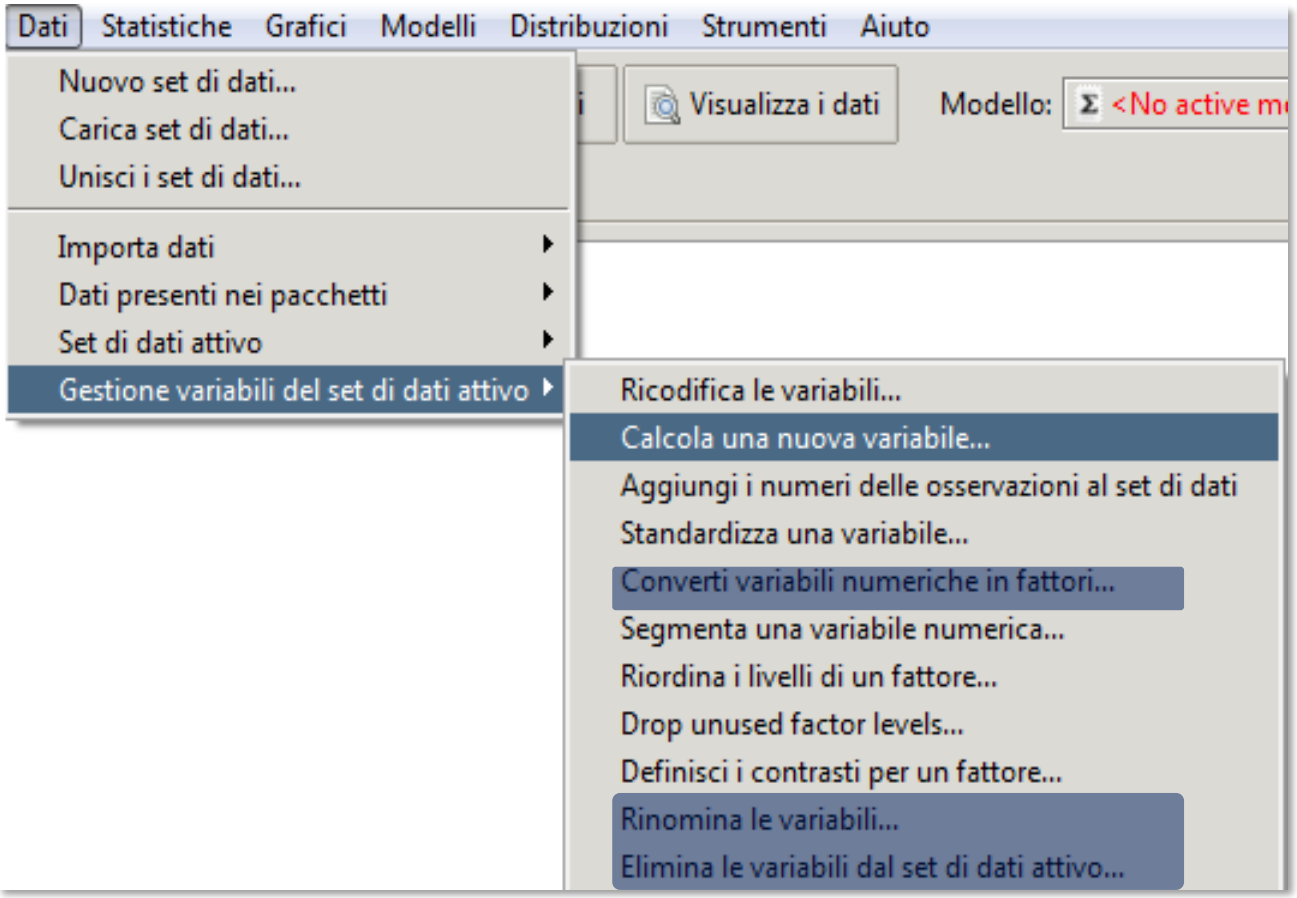
Lavorare sulla struttura con Rcommander

Il menu **`Dati → Set di dati attivo`** **offre una gran quantità di possibilità di azioni**



Lavorare sulle variabili con Rcommander

Il menu **`Dati → Gestione variabili del set di dati attivo`** offre altrettante possibilità di agire sulle singole variabili che lo compongono:



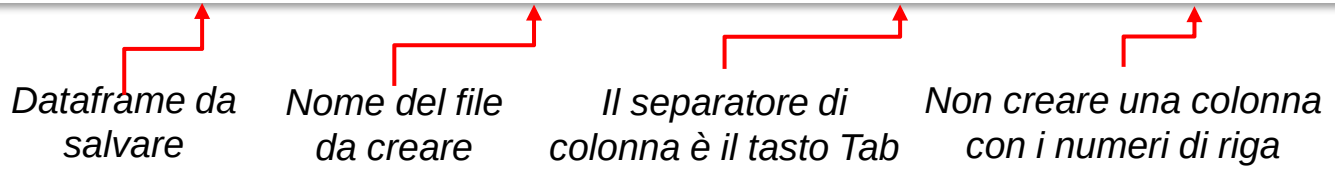
Salvare il dataframe come file

Non ci servirà per l'esame, ma i dataframe creati in R si possono salvare in formati diversi.

Per il formato con valori delimitati da tabulazioni (**tab-delimited file**) **.txt**, si usa:

```
dati<-write.table(dataframe, "nome del file.txt", sep="\t",
row.names=FALSE):
```

```
avanti_tabulato <- write.table(avanti, "avanti_tabulato.txt", sep="\t", row.names=FALSE)
```



Per salvare il file con valori separati da virgole (**CSV**: **comma-separated values**) **.csv** si usa:

```
dati <- write.csv(dataframe, "nome del file.csv")
```

```
avanti_virgole <- write.csv(avanti, "avanti con virgole.csv", row.names= FALSE)
```

È salvato nella **cartella predefinita**, che potete conoscere chiedere con `getwd()` - *get working directory*; non scrivete nulla tra `()`. Per cambiarla, usate `setwd()` - *set working directory* -specificando tra le parentesi il nuovo percorso, o `setwd(choose.dir())`

Importare dati in R

Lavoreremo molto più spesso in questa modalità

Aprire archivi di dati esterni

Spesso si usano **fogli di calcolo in wide format**, con dati inseriti a mano o esportati da un altro programma**. R importa dati da file di molti formati diversi**, ma noi lavoreremo con dati da file in **formato .txt** (o formato **.csv**).

Per aprire un file in formato .txt si usa **`read.delim()`**.

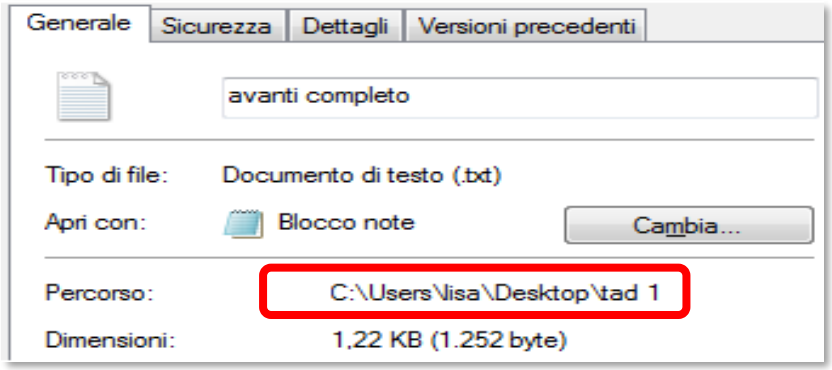
Se non diversamente specificato, R si **indirizza alla cartella in cui è stato installato**:

```
avanti_tabulato <-read.delim("C:/Users/lisa/Desktop/tad1.txt",header=TRUE)
```

Il dataframe `avanti_tabulato` — *è creato* — *leggendo il file in formato delimitato da tabulazioni* — *che è così identificato nel computer* — *e in cui la prima riga costituisce l'intestazione delle colonne*

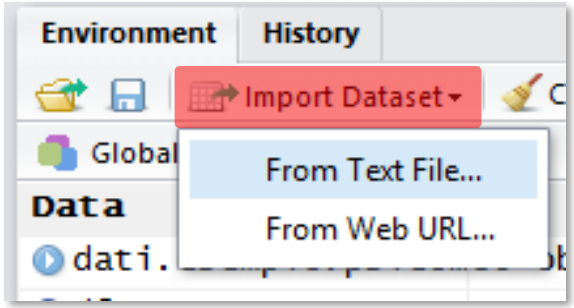
Trucco per evitare il percorso del file:

```
dataframe<-read.delim(file.choose(),
header=TRUE)
```



Analogamente, per **aprire un file .CSV**, useremo la funzione **`read.csv()`**.

Importare dati con RStudio

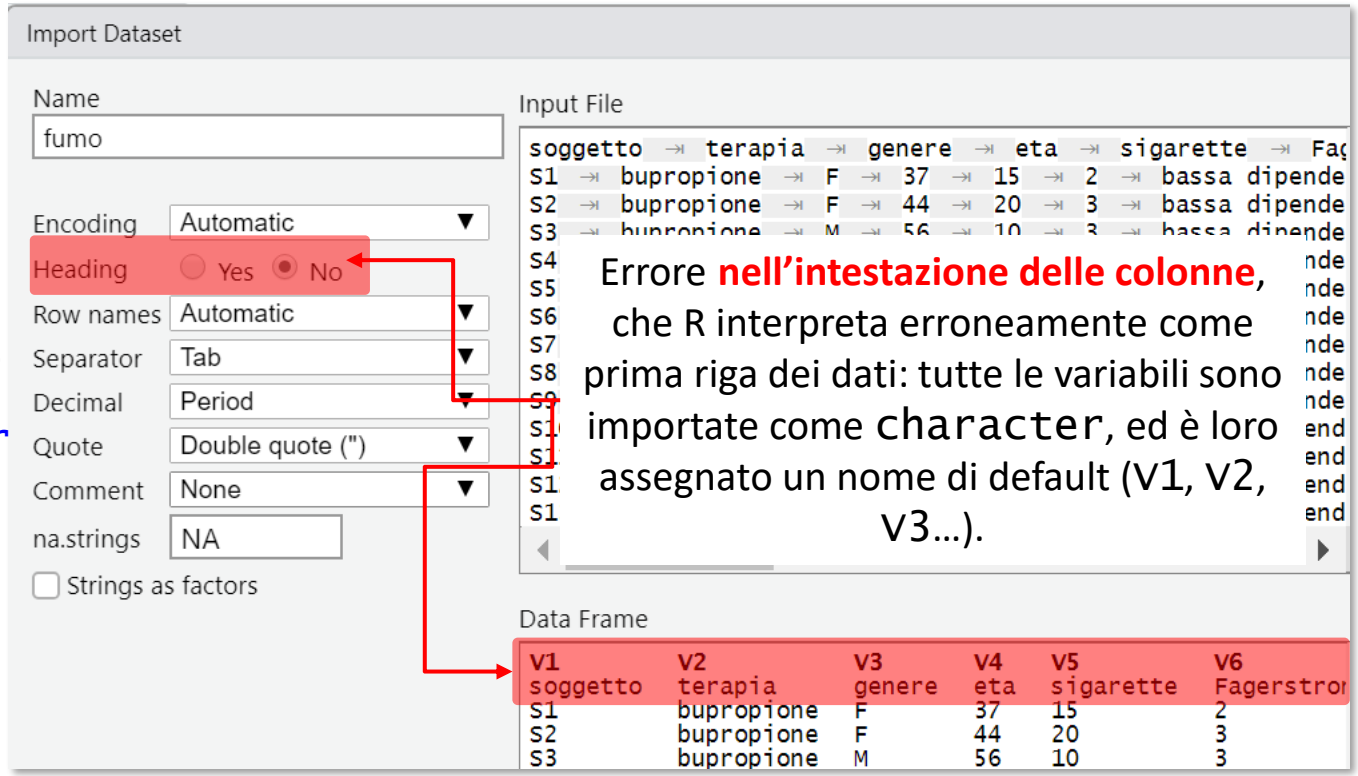


Dalla **Data** scegliete **Import Dataset** → **From Text File**

e poi usate le finestre per individuare il file da importare.

Una finestra di **Preview** consente di individuare eventuali errori nell'importazione.

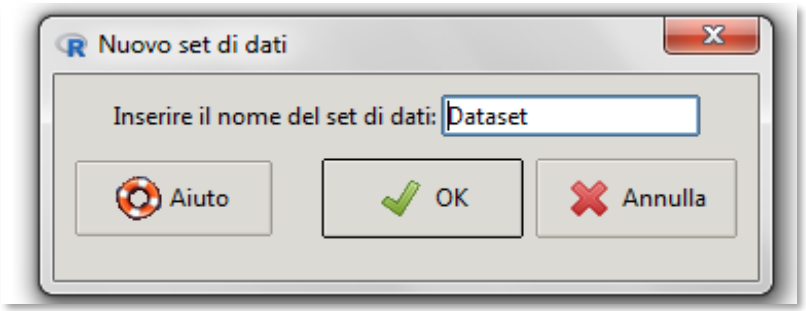
Controllate sempre **Heading: Yes** eche **Strings as factor** sia spuntato,

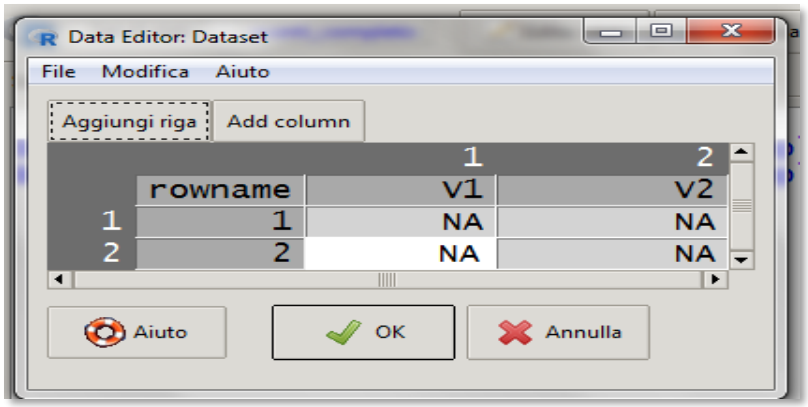


Il corrispondente comando viene stampato in **Console**; nella finestra **Data** compare il nome del dataframe e si apre la finestra **Visualizza**.

Modificare e creare dataframe con RCommander

Per creare variabili e dataframe, si può usare il menu **Dati** → **Nuovo set di dati**;





Per lavorare su dataframe esistenti, nel menu **Dati** si sceglie **Importa dati**, specificandone il formato (.txt, .xls). Se è un dataframe su cui si è già lavorato, si sceglie più rapidamente utilizzando l'icona **Set di dati**, scegliendo poi di visualizzarlo (**Visualizza** i dati) o di modificarlo (**Edita** i dati):

Questo è l'essenziale per iniziare a lavorare

Vedremo diverse altre operazioni con o sulle variabili e i loro elementi quando ne avremo necessità, nelle prossime lezioni