Praticamente Statistica
Laboratorio di software open source

ORIENTAMENTO CONSAPEVOLE

Imparare dai dati: la Statistica come strumento della conoscenza

a.a. 2021-2022

C. Calculli
crescenza.calculli@uniba.it
Dipartimento di Economia e Finanza, UNIBA

08 marzo 2022

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

SIS Società Italiana di Statistica

Basic info

Obiettivo incontro

Percorso attraverso alcuni semplici concetti di STATISTICA ed applicazione all'analisi di dati tramite il software `R`

Laboratorio

- Introduzione al software statistico `R`
- Utilizzo di `R` per applicare concetti teorici
- Utilizzo di `R` per analizzare i propri dati

Materiale didattico(MSTeams: File/Lab Software)

- Slides
- Dati
- Descrizione dei dati
- Codice `R`

Premessa

R è un ambiente di programmazione interattivo potente e flessibile per il calcolo statistico e la ricerca. R di per se non è difficile da imparare, ma come con qualsiasi nuova lingua (parlata o al computer), la curva di apprendimento iniziale può essere un po' ripida e un po' scoraggiante. Queste slide non sono destinate a coprire tutto ciò che c'è da sapere su R - sarebbe un compito impossibile! Lo scopo principale è di aiutare a scalare la curva di apprendimento iniziale e di fornire le abilità di base, e la sicurezza, necessarie per l'approfondimento nell'uso personale di R.

Introduzione - Perchè R?

R non è semplicemente un software statistico: è un ambiente e un linguaggio di programmazione che consente di manipolare dati, eseguire calcoli matematici avanzati e scrivere funzioni

- R è distribuito liberamente sotto **licenza GNU-GPL** (**G**eneral **P**ublic **L**icense).
- R è disponibile per diverse architetture e per i più comuni sistemi operativi: Windows, MacOS, Linux, Unix

Le origini

- **1996**: viene sviluppata la prima versione da R. Gentleman e da R. Ihaka del Dipartimento di Statistica dell'Università di Auckland
- **1997-1999**: insieme ad altri ricercatori (statistici ed informatici) viene costituito il gruppo principale di sviluppo (R Development Core Team)

- **2003**: nasce la R Foundation for Statistical Computing organizzazione no-profit che si occupa di promuovere lo sviluppo e la diffusione, di fornire supporto legale per le questioni di copyright e di rappresentare un punto di riferimento per il mondo istituzionale, imprenditoriale e accademico
- **2004**: l'Austrian Association for Statistical Computing organizza la prima di una serie di conferenze internazionali dedicate alla diffusione di nuove funzionalità e pacchetti di `R` e allo scambio di nuove idee e promozione di nuovi progetti
- **2006**: nasce Use-R collana editoriale dedicata agli utenti di `R` dell'editore Springer
- **28/02/2022**: viene rilasciata l'ultima versione R-4.1.3 (One Push-Up)

Oggi la comunità di sviluppatori si è molto allargata e fornisce aggiornamenti ed implementazioni continuamente. La filosofia di `R` è quella di un ambiente di sviluppo ***aperto****, di libero utilizzo e dove chiunque può contribuire e condividere le proprie esperienze e applicazioni*

Alcuni riferimenti bibliografici

- G. Grolemund, H.Wickham. **R for Data Science**, O'Reilly Media, 2017
- R. Coccarda, F. Frascati. **Manuale interattivo di Statistica con R - 100 casi step by step**, Pearson, 2015
- M. J. Crawley. **The R Book**, Wiley, 2013
- R. Micciolo, G. Espa. **Analisi esplorativa dei dati con R**, Apogeo Education - Maggioli Editore, 2012

Materiale web

- FAQ
- Manuali
- Documentazione: http://cran.r-project.org/other-docs.html
- Journal: http://journal.r-project.org/

Caratteristiche principali

R è un interprete di comandi:

- NO: menu, finestre e mouse per eseguire istruzioni
- SI: stringhe di comandi (esecuzione in modalità batch mediante file ascii)

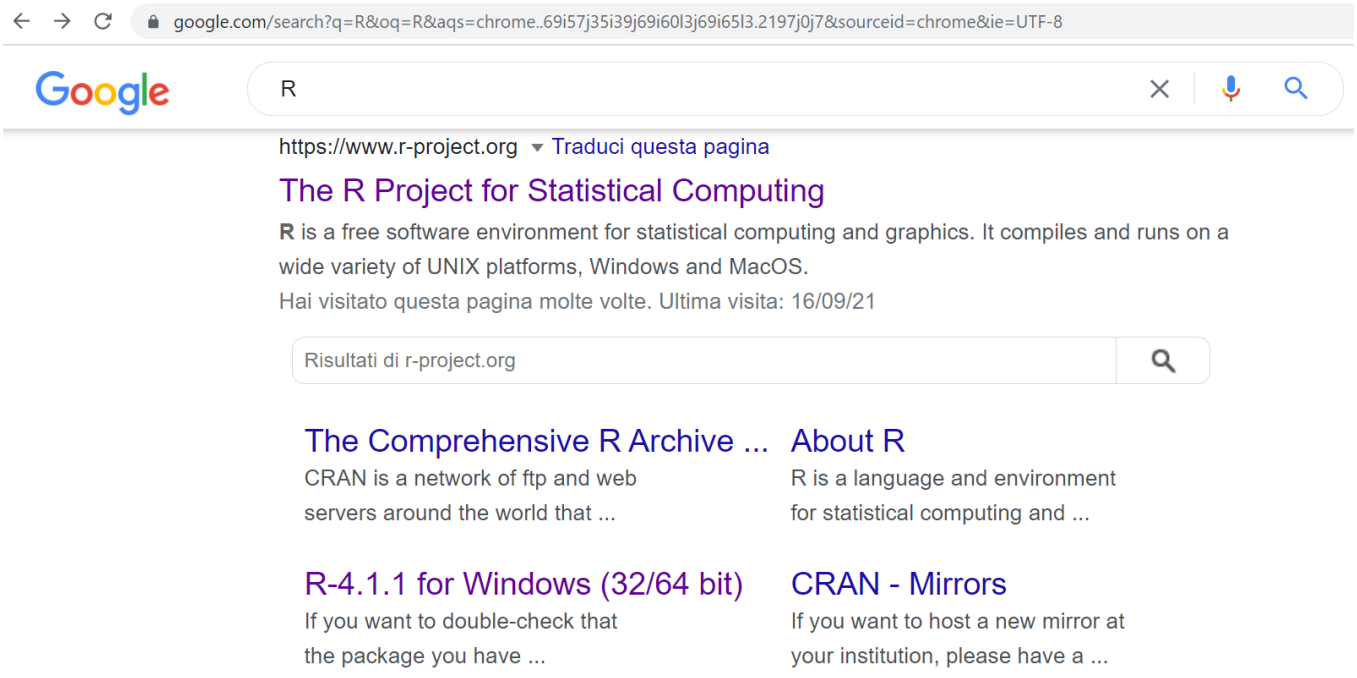
R è organizzato in modo da essere uno strumento estremamente flessibile:

- si ispira alla programmazione orientata agli oggetti
- permette di definire funzioni specifiche per le necessità dell'utente

Terminologia

- Console: finestra di lavoro che si apre all'avvio di `R`;
- Workspace: area di memoria (area di lavoro) che contiene tutti gli oggetti creati dall'utente;
- Directory di lavoro: cartella in cui salveremo il nostro lavoro e da cui importiamo i dati;
- Script: file di testo (blocco note, emacs, tex, doc ...) in cui si scrivono i comandi `R` che si eseguono in console:
  - indispensabile quando i codici sono lunghi
  - utile per eseguire le istruzioni senza scriverle nuovamente
  - chiarisce le idee

Installare R

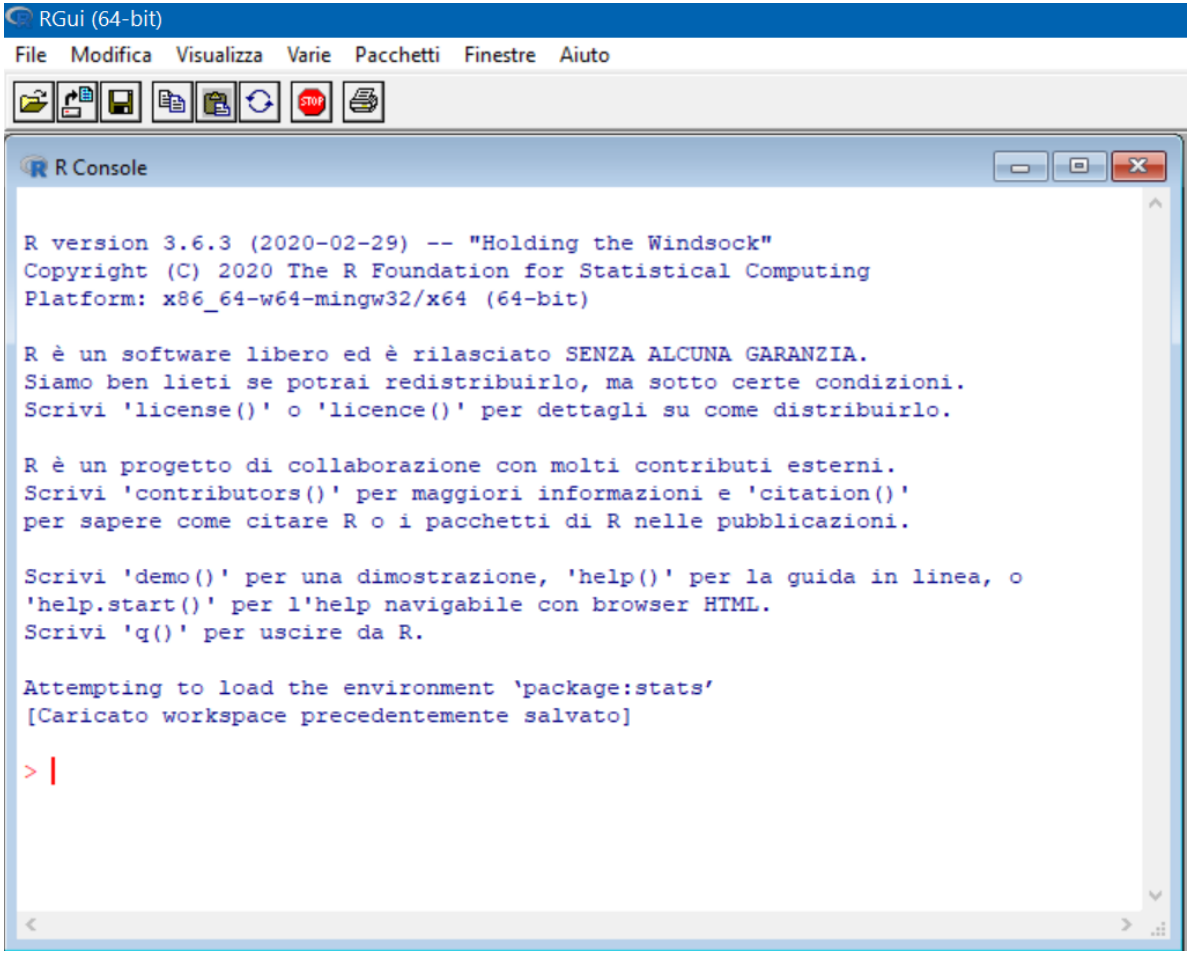


oppure direttamente

https://www.r-project.org/

1. Cliccare su **CRAN** (nel riquadro a sinistra) –> Selezionare il **Mirror** (Italy - Padova o Milano)
2. Selezionare **Download R for Windows** (o Mac o Linux a seconda dell'OS) –> Cliccare su **Base**
3. Cliccare su **Download R version (*ultima release*)**
4. Avviare il Download e installare seguendo la procedura guidata

Interfaccia utente (Graphic User Interface, GUI) di R



Il simbolo > è il *prompt* dei comandi e indica che R è pronto a ricevere l'istruzione

La linea di comando viene eseguita semplicemente premendo il tasto *Invio*

RStudio

RStudio è un ambiente di sviluppo integrato (**IDE**): è un software progettato per la realizzazione di applicazioni che aggrega strumenti di sviluppo comuni in un'unica interfaccia utente

PRO:

- facilita l'uso del linguaggio di programmazione R (ad es. attraverso le funzioni di controllo e auto-completamento)
- rende immediato l'accesso e la consultazione dell'aiuto (*help*) sulle funzioni
- consente una gestione facilitata degli oggetti presenti nell'area di lavoro e dei file di dati

ATTENZIONE!

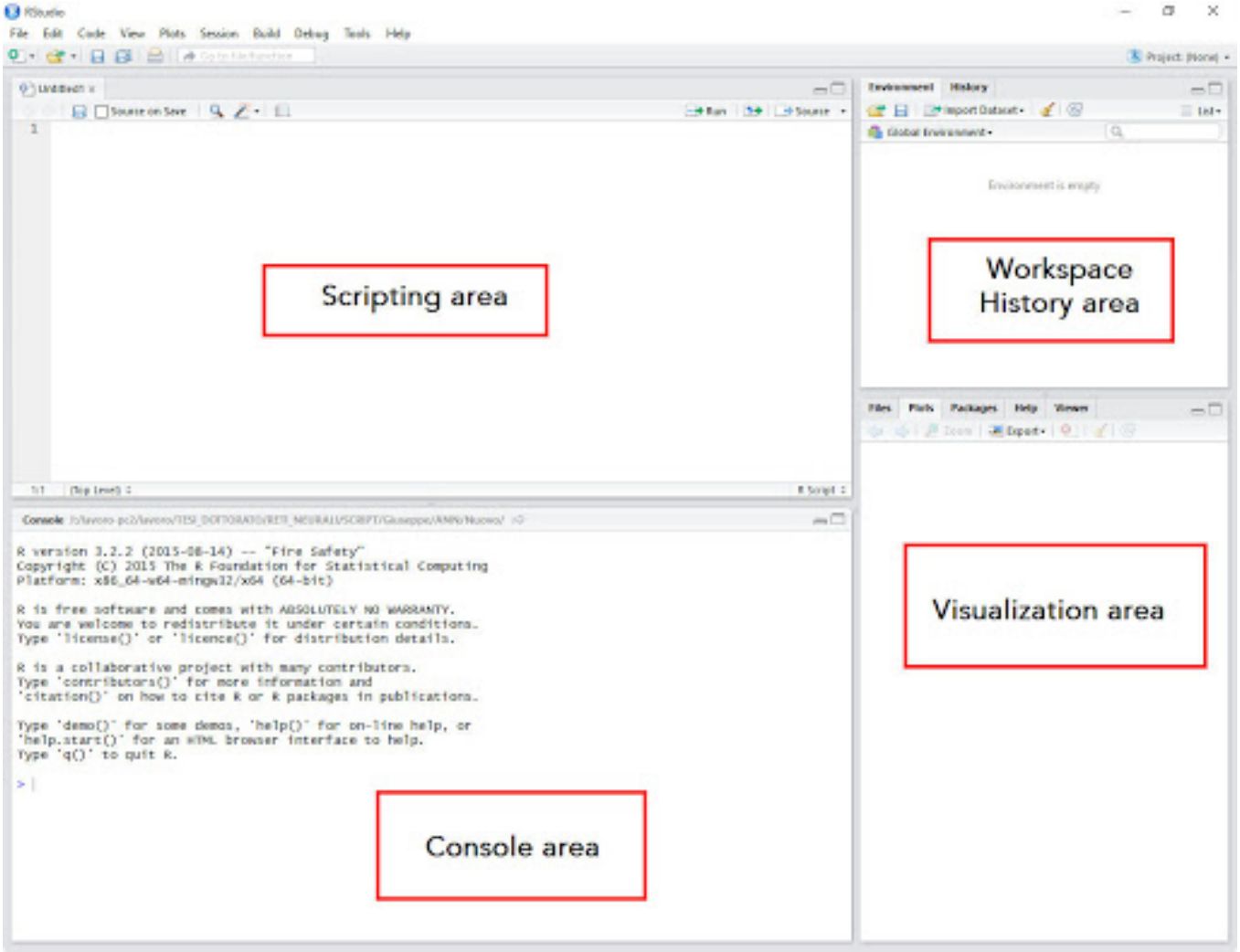
`RStudio` non è un software alternativo ad `R`! Prima di installare `RStudio` è necessario installare l'ultima versione di `R` - chiaramente, se non si conosce il linguaggio di programmazione di `R`, è impossibile usare sia `R` che `RStudio` come ambienti per la programmazione

Per scaricare `RStudio`

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

1. In corrispondenza di **RStudio Desktop** (FREE), cliccare su Download
2. Scegliere la versione di **Rstudio-*ultima relaese*.exe** in base all'OS e avviare il Download
3. Seguire la procedura automatica per l'installazione

Layout di RStudio



Anche in RStudio tutte le istruzioni possono essere eseguite dalla linea di comando nella *Console area*

Usare R come calcolatrice

Nel suo utilizzo più semplice, R può essere usato come calcolatrice

(Alcune) operazioni elementari

```
> 2+30 # somma
```

```
## [1] 32
```

```
> 20*5 # prodotto
```

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

```
> 2^3  # elevamento a potenza
```

```
## [1] 8
```

```
> (10/4)*(3^2) # espressione (valgono le comuni regole algebriche)
```

```
## [1] 22.5
```

Operatori più usati

Operatori matematici e funzioni elementari	
addizione	+
sottrazione	-
moltiplicazione	*
divisione o rapporto	/
elevamento a potenza	^ or **
divisione intera	%/%
resto della divisione intera (modulo)	%%
radice quadrata	sqrt()
esponenziale	exp()
logaritmo	ln()

Operatori logici e strutture di controllo	
minore	<
maggiore	>
minore uguale (maggiore uguale)	<= (>=)
AND	&
OR	|
NOT	!
diverso	!=
uguale	==

R concetti di base

R è un ambiente di programmazione **Object-Oriented** (OOP). Ogni entità (dato semplice, struttura di dati, risultato, ecc.) in R è un **oggetto**.

In un linguaggio di programmazione, il *tipo* di un dato è l'insieme dei valori assumibili da quel dato, ad es. interi, stringhe, booleani (detti tipi primitivi).

Questo vale anche in un linguaggio come R: essendo OO ha il concetto di oggetto e di classe. Gli oggetti sono dati strutturati identificabili con una *classe*. La classe è un tipo di dato derivato dai tipi primitivi. Ogni oggetto di una classe è definito da attributi, ognuno dei quali può essere un dato di tipo primitivo, e metodi applicabili agli attributi e agli oggetti stessi

Le operazioni sono disponibili solo tra oggetti dello stesso tipo. Ad es. una stringa non verrà trattata come un oggetto numerico. Se si tenterà di applicare una funzione impropria all'oggetto (ad es. una funzione algebrica), verrà restituito un messaggio di errore

Per costruire gli oggetti in R si utilizza il comando di **assegnamento** ("<-" o "->" o "=")

```
> x<-4+2
> x
```

```
## [1] 6
```

```
> y=log(5)*sqrt(36)
> y
```

```
## [1] 9.656627
```

```
> "statistica"->z
> z
```

```
## [1] "statistica"
```

In questo modo, gli oggetti x, y e z possono essere riutilizzati o modificati successivamente

Tutti gli oggetti creati risiedono nel Workspace; la funzione **ls()** permette di accedere all'elenco di questi oggetti

```
> ls()
```

```
## [1] "x" "y" "z"
```

Un oggetto può essere rimosso dal Workspace usando la funzione **rm()**

```
> rm(z)          # elimina z dal workspace
> rm(list=ls()) # elimina tutti gli oggetti presenti nel Workspace
```

Le caratteristiche degli oggetti possono essere ottenute utilizzando ad es. le funzioni **class()**, **typeof()** e **str()** (i.e. structure)

- quando il tipo di dato è semplice (stringa, numero, booleano) è equiparabile alla classe, quindi **class()**, **typeof()** restituiranno lo stesso risultato

```
> class(z)
```

```
## [1] "character"
```

```
> typeof(z)
```

```
## [1] "character"
```

- quando il tipo di dato è strutturato attraverso **class()** otteniamo la classe dell'oggetto (ad es. *dataframe*) mentre attraverso **typeof()** otteniamo la caratteristica più a basso livello dell'oggetto (ad es. *list*). Attraverso invece **str()** si ottiene la composizione dell'oggetto strutturato e i valori contenuti negli attributi

```
> # solo a titolo di esempio creiamo un dataframe
> # (vedremo in dettaglio in seguito questa funzione)
> X<-data.frame(a=c(2,5,6),b=c(3,4,10))
> class(X)
```

```
## [1] "data.frame"
```

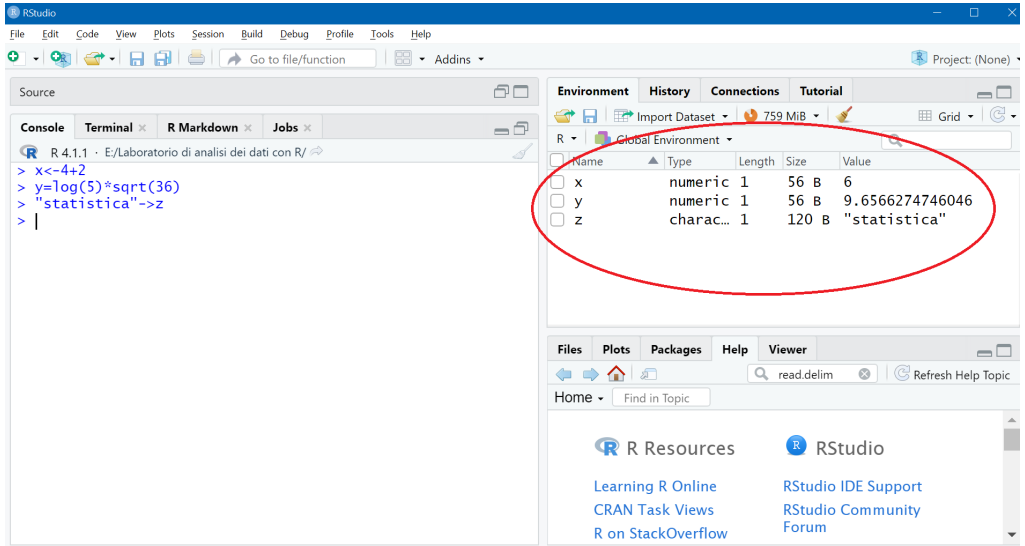
```
> typeof(X)
```

```
## [1] "list"
```

```
> str(X)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ a: num  2 5 6
##  $ b: num  3 4 10
```

In `Rstudio`, tutti gli oggetti presenti nel Workspace sono elencati anche nel pannello **Environment** insieme alle loro caratteristiche (nome, tipo, ecc.)



Il carattere "#" è il simbolo di commento e tutto ciò che segue viene ignorato dall'ambiente

```
> #  x è il primo oggetto creato
> #  y è il secondo oggetto
```

Attenzione! Se si creano oggetti che hanno gli stessi nomi di oggetti già presenti nel Workspace, il nuovo oggetto sovrascriverà il precedente e quest'ultimo non potrà essere recuperato

```
> x<-6^3+10
> x     # x non è più 8 ma 226
```

```
## [1] 226
```

Nella scelta dei nomi degli oggetti bisogna rispettare alcune semplici regole

- x è diverso da X - `R` è **Case sensitive**

```
> X
```

```
##   a  b
## 1 2  3
## 2 5  4
## 3 6 10
```

- non si possono iniziare i nomi degli oggetti con un numero

```
> 2oggetto<-3+sqrt(8)
```

```
## Error: <text>:1:2: simbolo inatteso
## 1: 2oggetto
##      ^
```

- non si possono usare nomi di oggetti che contegono spazi, ";" o ","

```
> oggetto 2<-3/5
```

```
## Error: <text>:1:9: costante numerica inattesa
## 1: oggetto 2
##            ^
```

- l'uso del "_" o dei "." invece è ammesso

```
> oggetto_2<-3/5
> oggetto_2
```

```
## [1] 0.6
```

- non si possono usare le parole riservate *FALSE*, *TRUE*, *Inf*, *NA*, *NaN*, *NULL*, *break*, *else*, *for*, *function*, *if*, *in*, *next*, *repeat* e *while* come nome da assegnare ad un oggetto

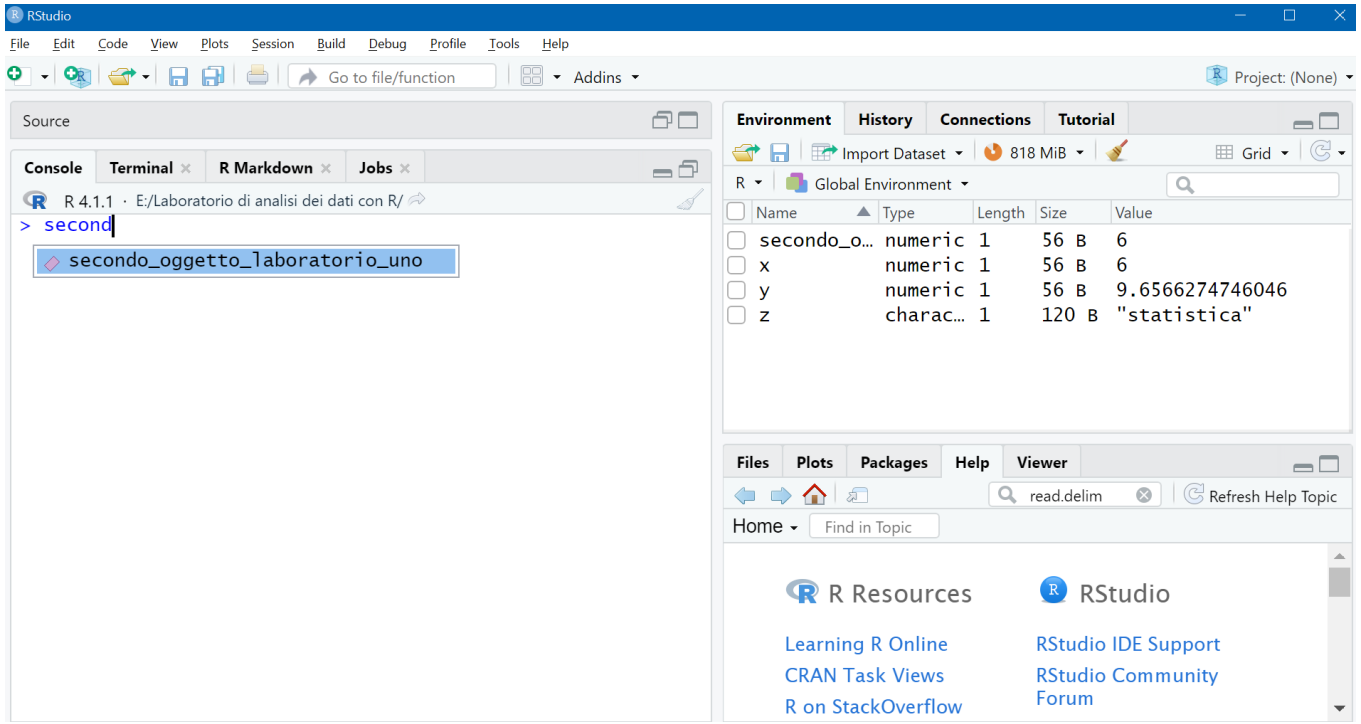
```
> TRUE<-2
```

```
## Error in TRUE <- 2: membro di sinistra dell'assegnazione (do_set) non valido
```

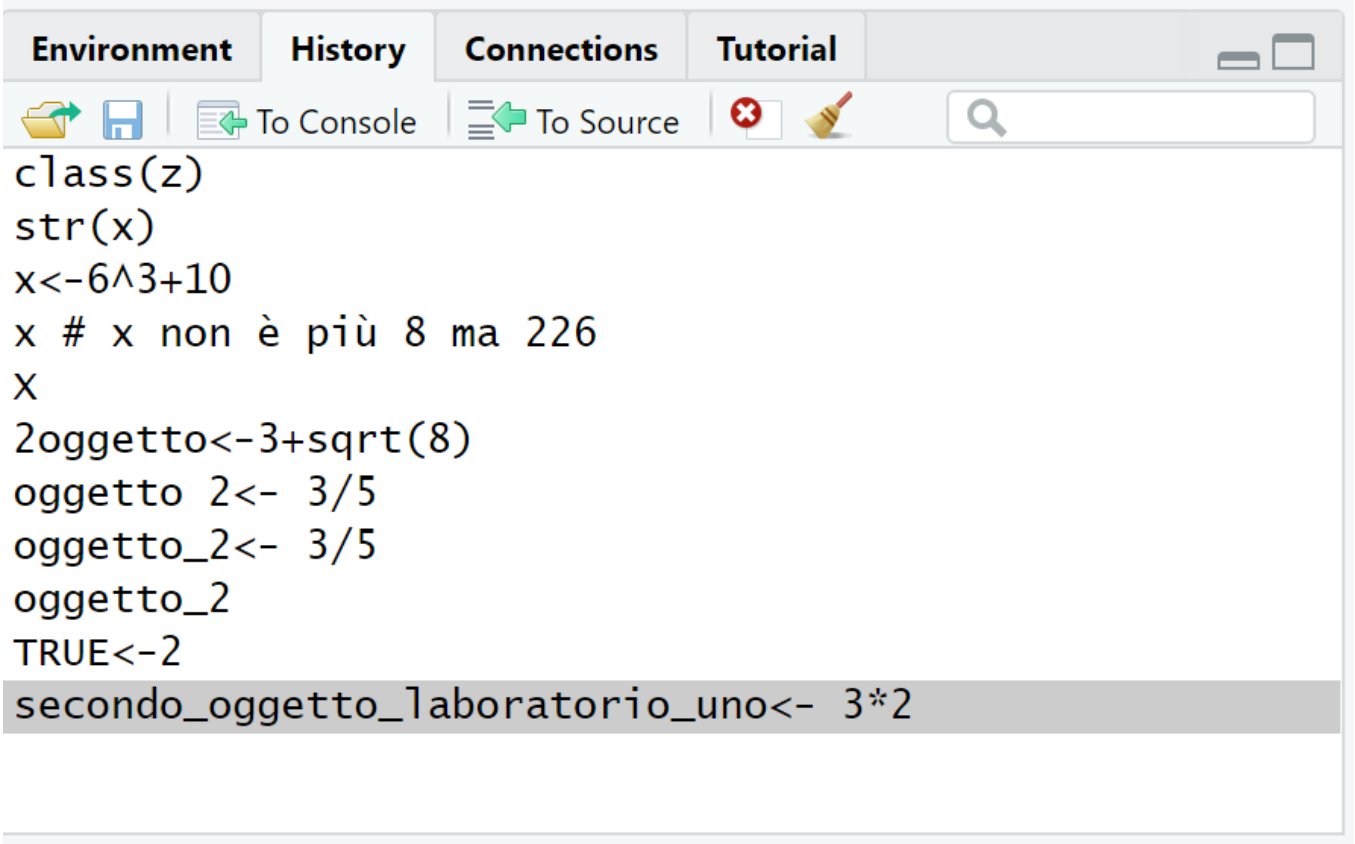
- è buona pratica non assegnare agli oggetti nomi troppo lunghi o complessi

```
> secondo_oggetto_laboratorio_uno<- 3*2 # sconsigliato!
```

Posizionandosi nella Console, è possibile richiamare un oggetto iniziando a digitarne il nome e, utilizzando la funzione di autocompletamento di `RStudio`, selezionarlo premendo "Enter"



Per risalire invece all'istruzione con cui è stato generato un oggetto, bisogna accedere alla **Command history** o digitando il tasto "↑" (cliccando più volte si può risalire all'istruzione desiderata) o direttamente dal pannello **History** di **RStudio**. In questo pannello vengono conservate tutte le operazioni eseguite fino a quel momento



Le funzioni in R

In R, una *funzione* può essere definita come un insieme compatto e organizzato di istruzioni che permette di ottenere un risultato in maniera veloce considerando uno o più valori impostati

Una generica funzione viene richiamata utilizzando la sintassi *nome_funzione(argomenti)*

A differenza di altri linguaggi, in R le funzioni sono degli oggetti primitivi, al pari di interi, booleani e stringhe. Per tale ragione è possibile assegnare funzioni ad oggetti, passare come argomento ad una funzione un'altra funzione, restituire una funzione come risultato

Esistono una molteplicità di funzioni predefinite dal linguaggio

A titolo di esempio, consideriamo l'istruzione necessaria per calcolare la media aritmetica dei primi 5 numeri interi

```
> x=(1+2+3+4+5)/5
> x
```

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

In modo più efficente potremmo calcolare la stessa media utilizzando la funzione **mean()** di R

```
> x=mean(1:5)
> x
```

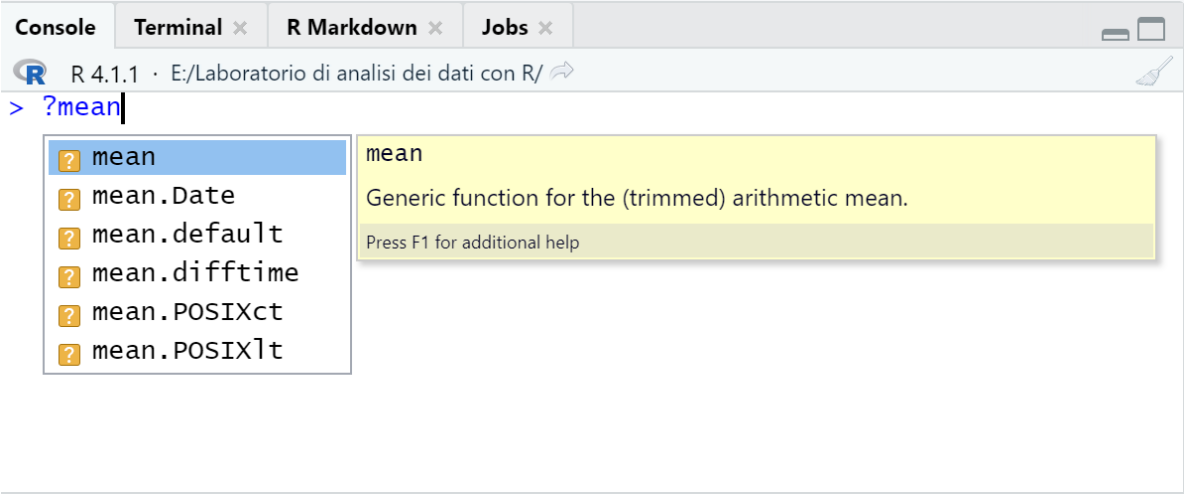
```
## [1] 3
```

Le informazioni riferite alle funzioni predefinite sono contenute nell'**Help page** (in `RStudio`, il pannello in basso a destra)

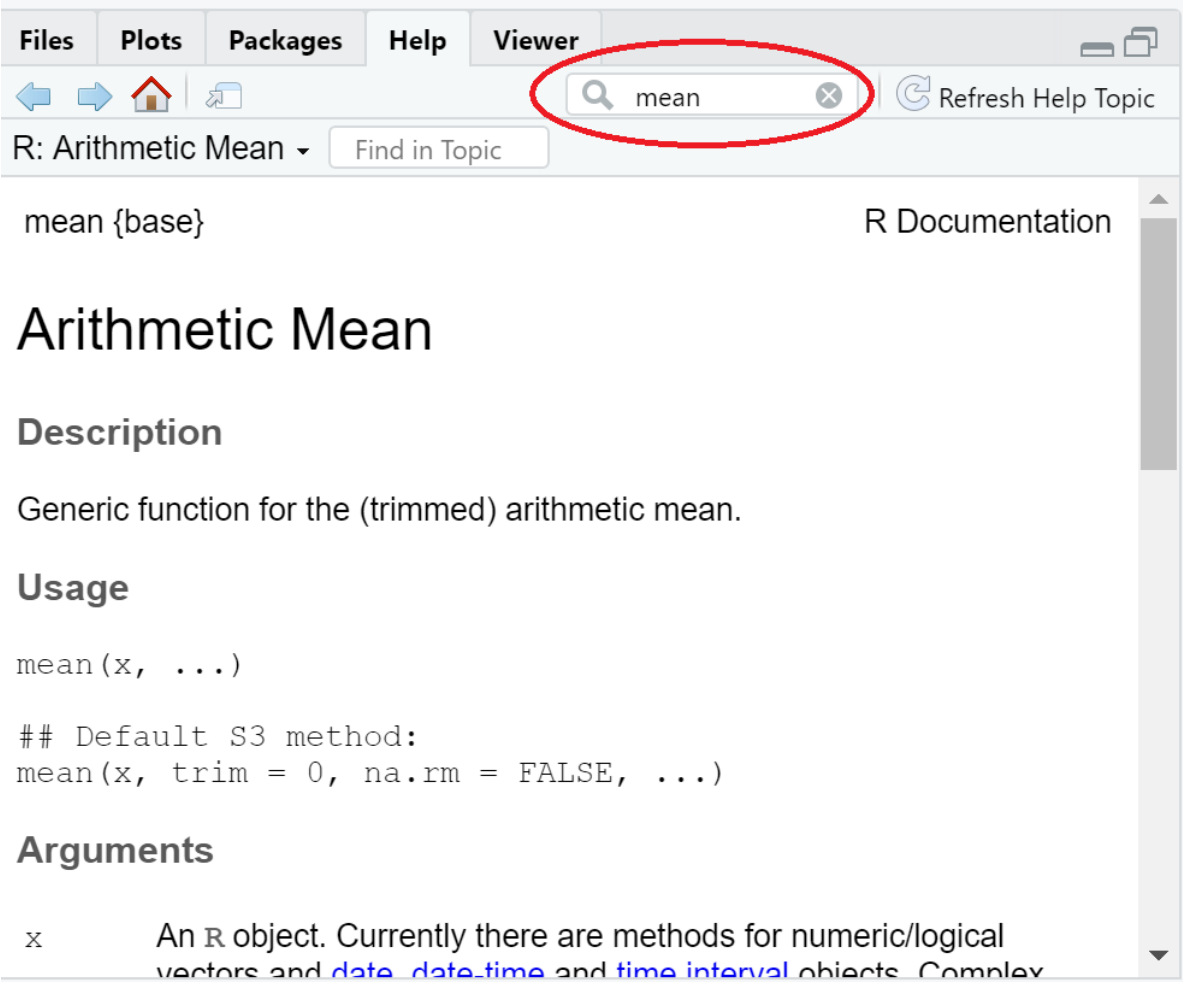
Per accedere all'help delle funzioni basta digitare nella Console *?nome_funzione* oppure *help(nome_funzione)*

```
> ?mean
> help(mean)
```

Anche in questo caso verranno proposte, grazie all'autocompletamento, tutte le funzioni predefinite che contengono la stringa digitata



Alternativamente in `RStudio` si può richiamare l'help di una funzione digitando il nome della funzione nella barra di ricerca



Il contenuto dell'Help page è molto ricco e le informazioni in esso contenute sono

- in alto a sinistra figura il nome della funzione; tra parentesi graffe è riporatato il nome della libreria che contiene la funzione. La funzione **mean()** fa parte dei comandi base di R quindi troviamo {base}
- **Description**: è una breve descrizione di cosa fa la funzione
- **Usage**: mostra le impostazioni di default della funzione
- **Arguments**: descrive uno ad uno, tutti gli argomenti che possono essere utilizzati nella funzione
- **Value**: descrive l'output della funzione
- **Reference**: ogni funzione di R è associata ad uno o più riferimenti bibliografici
- **See Also**: suggerisce altre voci dell'Help che sono in qualche modo legate alla funzione cercata
- **Examples**: fornisce alcuni esempi per l'utilizzo della funzione

I Pacchetti

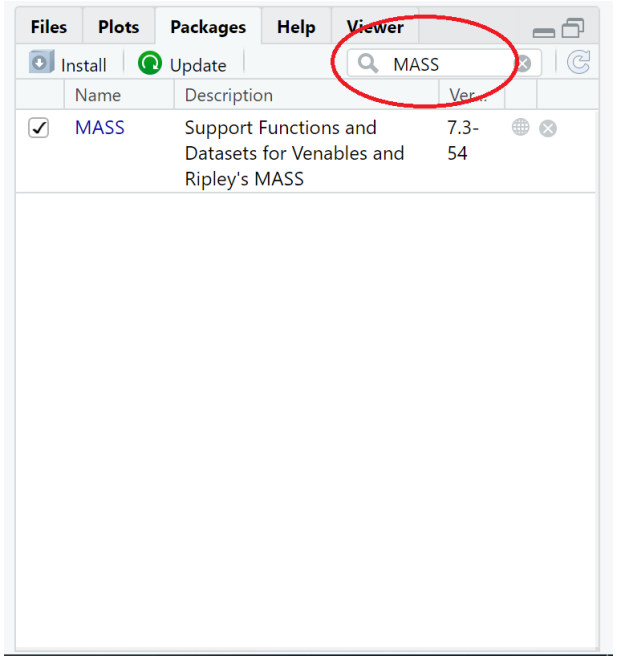
R propone un set fondamentale di funzioni (base) accessibile a tutti gli utenti e la possibilità di istallare o caricare delle **estensioni** per svolgere ulteriori compiti particolari

In R queste estensioni sono i **pacchetti** (packages o libraries). Si può accedere alla lista dei pacchetti installati usando il comando **library()**, senza indicare l'argomento oppure dal panello **Packages** (in RStudio, in basso a destra). Se si indica come argomento il nome di un pacchetto, R caricherà il pacchetto ed il suo contenuto sarà disponibile per l'utente

Ad es., possiamo caricare il pacchetto **MASS** con il comando **library(MASS)** (o anche **require(MASS)**)

```
> library(MASS)
```

Sfruttando i vantaggi di RStudio, lo stesso pacchetto può essere caricato spuntando nella lista dei pacchetti installati il pacchetto desiderato

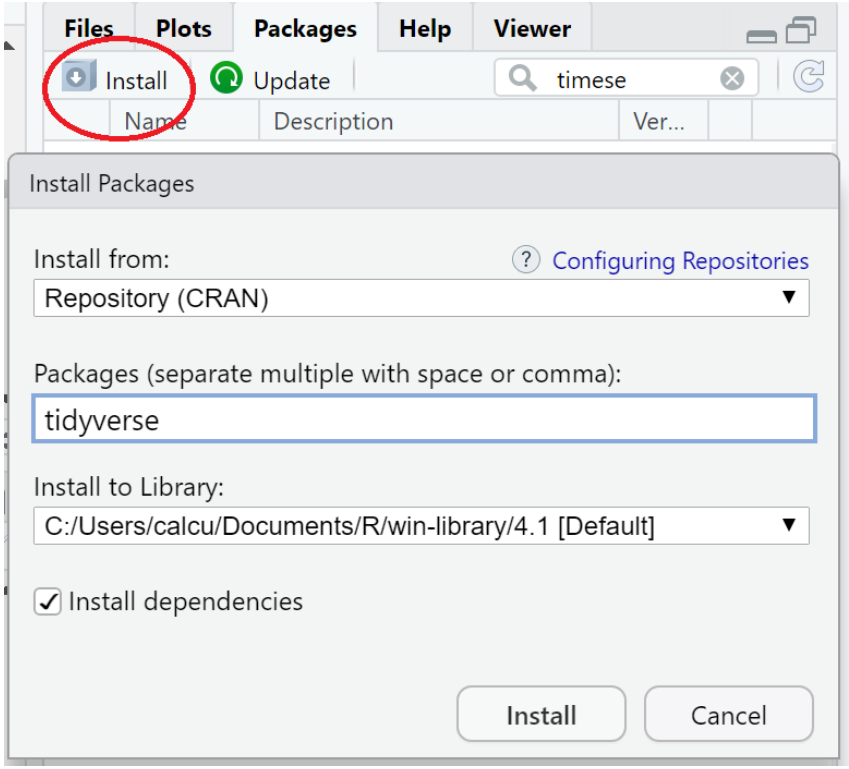


Prima di caricare un pacchetto è sempre necessario installarlo! Non tutti i pacchetti sono già installati in R, ad esempio il pacchetto **tidyverse** non lo è

Per installare un pacchetto si può usare la funzione
install.packages("*nome_del_pacchetto*")

```
> install.packages("tidyverse")
> library(tidyverse)    # ora il pacchetto è caricato!
```

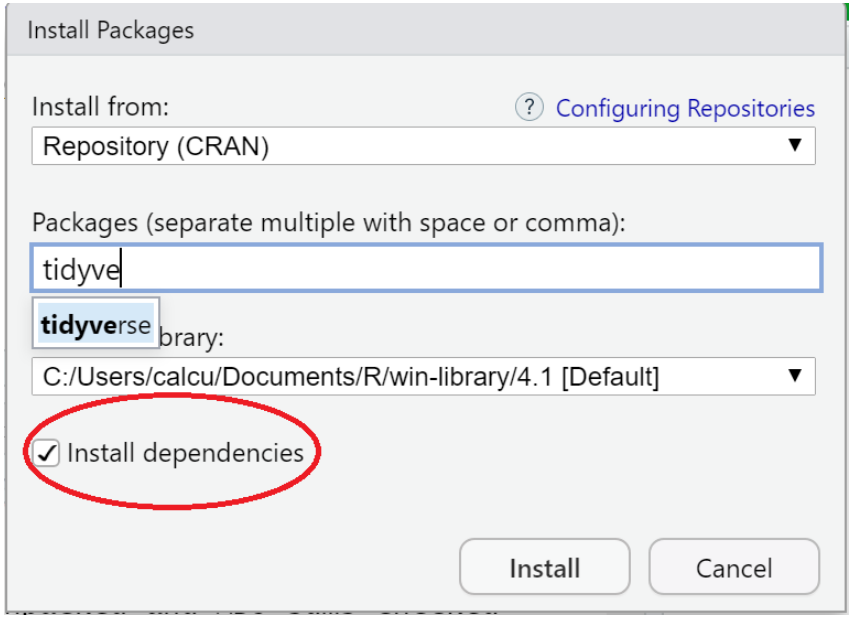
Alternativamente, si può utilizzare la procedura guidata dal pannello **Packages**



Alcuni pacchetti potrebbero dipendere da altri pacchetti non installati per funzionare. Per indicare ad R di installare anche le dipendenze, basta inserire nel comando, dopo il nome del pacchetto l'istruzione **install.packages(nome_pacchetto , dependencies = TRUE)**

```
> install.packages("tidyverse", dependencies = TRUE)
```

Con la procedura guidata basta spuntare il riquadro delle dipendenze



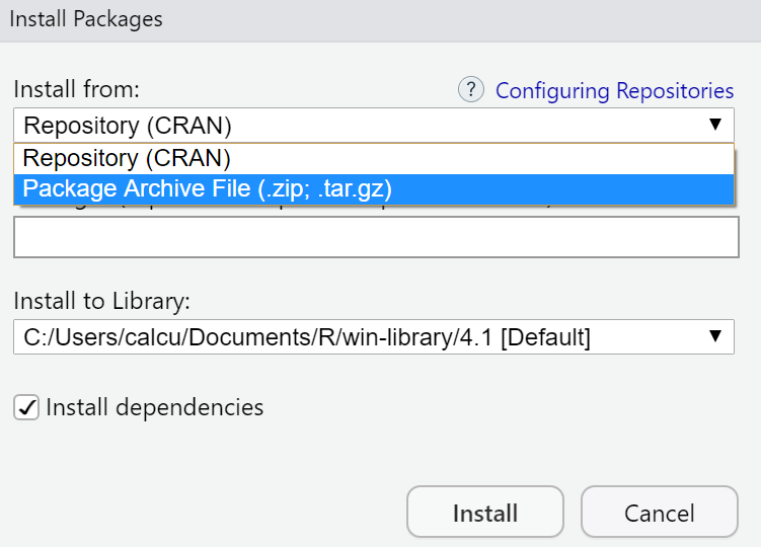
Il comando **search()** elenca i pacchetti caricati nella sessione di lavoro

I pacchetti possono essere prodotti da qualsiasi utente e solitamente sono reperibili usando i comandi visti sopra. Di default `RStudio` si appoggia al sito **CRAN** per reperire i pacchetti

Qualora il pacchetto desiderato non fosse presente nel **CRAN**, è possibile indicare un server diverso. Ad es., il comando seguente permette di installare un pacchetto presente nei server di Bioconductor

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("RnaSeqTutorial")
```

Infine, in `RStudio` è possibile installare i pacchetti manualmente anche da locale dal menu Tools| Install Packages... | Install from: | Package Archive File



Funzioni definite dall'utente

Indipendentemente dalle funzioni presenti nei pacchetti, l'utente può scrivere una propria funzione seguendo delle regole di programmazione. Per definire una funzione in R si usa il comando **function**. La sintassi di base è
function (lista parametri) espressione
Ad es., **function (x) x+2** è la funzione che prende un oggetto x in input e restituisce x+2. La funzione in se non ha un nome, perché una funzione è un valore come un intero o un booleano (è appunto un tipo primitivo!). Se vogliamo assegnare una funzione ad un oggetto "funz", scriveremo
funz<-function()

```
> function (x) x+2
```

```
## function (x) x+2
```

```
> funz=function (x) x+2
> funz
```

```
## function (x) x+2
```

Possiamo ora invocare la funzione con dei parametri opportuni, causando l'esecuzione dell'espressione che la definisce. Si può invocare sia una funzione assegnata ad un oggetto ma anche una funzione "anonima"

```
> funz(10)
```

```
## [1] 12
```

```
> funz(c(3,5))   # c() definisce un vettore
```

```
## [1] 5 7
```

```
> (function(x) x+2)(c(3,5))  # funzione anonima
```

```
## [1] 5 7
```

Quando si crea una funzione è possibile specificare dei parametri di default, semplicemente scrivendo *parametro=espressione* nella lista dei parametri

```
> funz2=function (x,y=sqrt(4)) x+y
> funz2(3)
```

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

Nella definizione di una funzione, è possibile inserire anche più espressioni. In questo caso, queste espressioni vanno racchiuse tra parentesi graffe. Il valore restituito dalla funzione è il valore dell'ultima espressione. Una delle espressioni all'interno della funzione può essere anche un assegnamento ad un oggetto. Si tratta in questo caso di un oggetto locale che ha validità solo all'interno della funzione stessa

```
> funz2 = function (x,y=sqrt(4)) {
+     t = y*y # è un oggetto locale!
+     x + t
+ }
> funz2(3)
```

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

Gli oggetti di R

I principali oggetti con cui si lavora in R sono

- Vettori
- Matrici e Array
- Liste
- Data Frames

R - Vettori

R gestisce i vettori come un tipo primitivo, al pari dei numeri, dei booleani e delle stringhe. Anzi, il vettore è il tipo di R più semplice possibile: un numero per R non è altro che un vettore di numeri di lunghezza 1

Per creare un vettore sia usa la funzione **c(*input1*, *input2*, ...)** (**c** sta per *concatenate*, concatenare)

```
> x<-c(1,5,7,8,30)
> x
```

```
## [1]  1  5  7  8 30
```

Si possono creare vettori anche utilizzando altri comandi

- **from:to**

```
> x<-5:20     # x è un vettore di interi compresi tra 5 e 20
> x
```

```
##  [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

- **seq(*from=...,to=...,by=...*)**

```
> x<-seq(5,20,5)    # x è un vettore di numeri compresi tra 5 e 20,
>                   # a passo di lunghezza 5
> x
```

```
## [1]  5 10 15 20
```

- **rep(*input,times*)**

```
> xx<-rep(x,2)     # xx è un vettore ottenuto replicando
>                  # il vettore x per 2
> xx
```

```
## [1]  5 10 15 20  5 10 15 20
```

Se il vettore generato è troppo lungo per essere visualizzato su una sola riga, viene visualizzato in più righe consecutive. All'inizio di ogni riga, il simbolo **[i]** (che fin'ora abbiamo sempre visto come [1]) indica quale posizione del vettore occupa il primo elemento visualizzato nella riga

```
> x<-5:34
> x     # 5 e 30 sono rispettivamente il 1° e il 26° elemento di x
```

```
##  [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 30 31 32 33 34
```

Subsetting di vettori

Utilizzando le parentesi quadre (“**[]**”) si può effettuare la selezione degli elementi di un vettore

```
> y<-c(1,3,-4,6,0,-10,8,5)
> y[2]        # seleziona l'elemento di y alla posizione 2
```

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

```
> y[c(2,4)]  # seleziona gli elementi di y alla posizione 2 e 4
```

```
## [1] 3 6
```

```
> y[-c(1,3)] # esclude gli elementi di y nelle posizioni 1 e 3
```

```
## [1]   3   6   0 -10   8   5
```

```
> y[y>0]      # seleziona gli elementi positivi di y
```

```
## [1] 1 3 6 8 5
```

```
> y[!(y<=0)] # esclude  elementi non strettamente positivi
```

```
## [1] 1 3 6 8 5
```

```
> y[y>0]-1   # tutti gli elementi positivi di y -1
```

```
## [1] 0 2 5 7 4
```

```
> y[y>0][3]  # seleziona l'elemento di y in posizione 3 tra i positivi
```

```
## [1] 6
```

Le espressioni sui vettori posso essere composte da operazioni annidate.

Vengono eseguite proprio come le espressioni sui numeri: prima si eseguono le operazioni all'interno delle parentesi più interne, poi si eseguono le operazioni più esterne

```
> yy<-rep(c(1,4),3)*2 # prima si crea il vettore c(),
>                     # questo viene ripetuto 3 volte e
>                     # infine ogni elemento viene
>                     # moltiplicato per 2
> yy
```

```
## [1] 2 8 2 8 2 8
```

Per conoscere la posizione degli elementi di un vettore che soddisfano una particolare condizione si può utilizzare la funzione **which()**, che richiede come argomento un vettore di tipo logico

```
> which(y>0)   # posizione assunta dagli elementi positivi
```

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

```
> which.min(y) # posizioni del valore minimo
```

```
## [1] 6
```

```
> which.max(y) # posizioni del valore massimo
```

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

La funzione **length()** se applicata ad un vettore, ne restituisce la dimensione:

```
> length(y) # vettore di lunghezza 8
```

```
## [1] 8
```

Per ordinare gli elementi di un vettore si possono usare le funzioni **sort()** oppure **order()**

```
> sort(y)                    # in ordine crescente
```

```
## [1] -10  -4   0   1   3   5   6   8
```

```
> sort(y, decreasing=T) # in ordine decrescente
```

```
## [1]   8   6   5   3   1   0  -4 -10
```

Fin qui abbiamo considerato vettori numerici, ovvero vettori su cui è possibile effettuare operazioni aritmetiche. R consente la costruzione di vettori non-numerici, sui cui si possono effettuare solo alcune operazioni (ad es. subsetting)

```
> z<-c("Foggia","BAT","Bari","Brindisi","Taranto","Lecce")
> z
```

```
## [1] "Foggia"   "BAT"      "Bari"     "Brindisi" "Taranto"  "Lecce"
```

```
> z[c(1,2)]  # Le province più a nord della Puglia
```

```
## [1] "Foggia" "BAT"
```

```
> provB<-z[grep("^[B].*", z)]  # la funzione grep() cattura tutti
>                               # gli elementi che iniziano con "B"
>
> provB
```

```
## [1] "BAT"      "Bari"     "Brindisi"
```

Anche se è possibile formare vettori che comprendono sia stringhe sia numeri, i numeri inseriti in vettori "misti" vengono comunque trasformati in stringhe

```
> v<-c("Bari",5, 10)
> v
```

```
## [1] "Bari" "5"    "10"
```

Le virgolette indicano che tutti gli elementi di v sono stringhe; più avanti vedremo come creare oggetti "misti" (i.e. dataframe)

R - Matrici e Array

Matrici

Le matrici sono collezioni di dati dello stesso tipo, individuate dall'indice di riga e colonna

Per creare una matrice sia usa la funzione **matrix(*data*, *nrow* *ncol*)**

```
> X<-matrix(data=c(10,20,30,40),nrow=2,ncol=2)
> X  # matrice di dimensione 2X2
```

```
##      [,1] [,2]
## [1,]   10   30
## [2,]   20   40
```

Di default R popola le matrici per colonna. Se si vuole procedere inserendo gli elementi per riga si utilizza l'argomento ***byrow=T***

```
> X<-matrix(data=c(10,20,30,40),2,2, byrow=T)
> X
```

```
##      [,1] [,2]
## [1,]   10   20
## [2,]   30   40
```

Attenzione! Il numero di righe e colonne deve essere coerente con l'argomento *data* della funzione matrix

```
> X1<-matrix(1:10,nrow=4,ncol=3)
> X1
```

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

Il vettore 1:10 viene riutilizzato (vengono usati i primi due elementi) per ottenere il risultato; inoltre, viene stampato un *warning*...provate!

Ulteriori funzioni per la costruzioni di matrici sono **cbind(*input1*, *input2*, ...)** e **rbind(*input1*, *input2*, ...)**

```
> X2<-cbind(3:4,c(2,-3),c(NA,9))
> X2 # dispone i tre vettori per colonna
```

```
##      [,1] [,2] [,3]
## [1,]    3    2   NA
## [2,]    4   -3    9
```

```
> X3<-rbind(3:4,c(2,-3),c(NA,9))
> X3 # dispone i tre vettori per riga
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    2   -3
## [3,]   NA    9
```

Alcune funzioni utili per lavorare con le matrici sono

```
> diag(X)          # estrae la diagonale principale di X
```

```
## [1] 10 40
```

```
> diagX<-diag(1:3) # costruisce la matrice diagonale di elementi 1:3
> diagX
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

```
> diagI<-diag(3)  # se k è uno scalare, crea una matrice
>                 # identità di dimensione kXk
> diagI
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Subsetting di matrici

Come per i vettori, l'utilizzo di "[ ]" è utile per selezionare gli elementi della matrice: la "," serve per separare gli indici di riga e di colonna

```
> X2[2,]    # seleziona la riga 2 di X2
```

```
## [1]  4 -3  9
```

```
> X2[,1]    # seleziona la colonna 1 di X2
```

```
## [1] 3 4
```

```
> X2[2,3]   # seleziona l'elemento alla riga 2 e alla colonna 3
```

```
## [1] 9
```

Si possono selezionare anche più righe e colonne insieme, fornendo i vettori come indici

```
> X2[c(1:2),c(2:3)]
```

```
##      [,1] [,2]
## [1,]    2   NA
## [2,]   -3    9
```

```
> X2[,-c(1:2)]      # seleziona tutte le colonne tranne la 1 e la 2
```

```
## [1] NA  9
```

In una matrice si può sostituire una riga o una colonna, ammesso che le dimensioni corrispondano

```
> X2[1,]<-rep(3,3)  # sostituisce la riga 1 di X2 con il nuovo vettore
```

La dimensione di una matrice è associata ad una coppia di numeri: la funzione **dim()** viene utilizzata per accedere al numero di righe e colonne

```
> dim(X2)     # dimensione di X2
```

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

```
> dim(X2)[1] # numero di righe di X2
```

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

```
> dim(X2)[2] # numero di colonne di X2
```

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

La funzione **matrix()** utilizzata senza il suo primo argomento permette di creare una matrice di valori mancanti, NA

```
> emptym=matrix(,nrow=2,ncol=2)  # matrice vuota
> emptym
```

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

La matrice *emptym* può essere popolata ad es. utilizzando un vettore numerico di dimensione pari al numero di elementi della matrice

```
> emptym[]<-1:4
```

Infine si fa notare come si può "forzare" una matrice a diventare un vettore con la funzione **as.vector()**

```
> as.vector(emptym)
```

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

Array

Gli Array sono oggetti multidimensionale e costituiscono una estensione delle matrici. Ogni elemento di un array è individuato da un vettore di indici (così come i vettori e matrici sono individuati rispettivamente da uno e due indici) Un array tridimensionale è caraterizzato dalla terna (i_1, i_2, i_3) e può essere creato attraverso la funzione **array()**

```r
> a<-array(1:36, dim=c(3,4,3))  # crea un array
> a
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   25   28   31   34
## [2,]   26   29   32   35
## [3,]   27   30   33   36
```

```
> dim(a)
```

```
## [1] 3 4 3
```

L'array *a* di dimensione 3×4×3 può essere pensato come 3 matrici 3 × 4 una dietro l'altra

Subsetting di array

L'uso delle parentesi quadre, alla stregua di vettori e matrici, ha il fine di selezionare sottoinsiemi dell'array

```
> a[1,,]  # seleziona le righe 1 delle 3 matrici disponendole per colonna
```

```
##      [,1] [,2] [,3]
## [1,]    1   13   25
## [2,]    4   16   28
## [3,]    7   19   31
## [4,]   10   22   34
```

```
> a[,,3]  # seleziona la terza matrice
> a[,2,]  # seleziona le colonne 2 delle 3 matrici
```

R - Liste

In R una lista è un oggetto che raccoglie altri oggetti, anche differenti tra loro, compreso altre liste. Abbiamo visto che questo non vale per i vettori e le matrici, i cui elementi invece sono dello stesso tipo

Una lista viene creata usando il comando **list()** ed il numero degli oggetti che la costituiscono definisce la dimensione della lista, mentre le sue componenti sono individuate con semplici [] o doppie [[ ]] parentesi quadre

```
> lista<-list(rep(0,3), matrix(1:4,nrow=2),c("Bari","Lecce"))
> # questa lista contiene un vettore di 0 di lunghezza 3,
> # una matrice 2X2 e
> # un vettore di strighe di lunghezza 2
```

```r
> lista
```

```
## [[1]]
## [1] 0 0 0
##
## [[2]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## [[3]]
## [1] "Bari"  "Lecce"
```

```r
> length(lista) # restituisce la dimensione della lista
```

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

```r
> lista[1]   # seleziona il primo elemento della lista
```

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

oppure

```
> lista[[3]] # seleziona il terzo elemento della lista
```

```
## [1] "Bari"  "Lecce"
```

```
> lista[[2]]+2  # esegue un'operazione sul secondo elemento (matrice)
```

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

```
> lista[[2]][2,2]  # estrae l'elemento alla riga 2  e colonna 2
```

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

Ogni singola sub-componente della lista può essere trattata come un oggetto separato e indipendente su cui poter effettuare delle operazioni. Se per estrarre gli oggetti della lista si usano le singole (piuttosto che le doppie) parentesi quadre, l'oggetto verrà comunque estratto ma sarà ancora una lista; la funzione **unlist()** viene usata per rimuovere dall'oggetto la struttura della lista

```
> str(lista[1])   # il primo elemento della lista è ancora una lista
```

```
## List of 1
##  $ : num [1:3] 0 0 0
```

```
> prov<- unlist(lista[1])  # il primo elemento della lista viene
>                          # salvato come vettore
> str(prov)
```

```
##  num [1:3] 0 0 0
```

E' possibile assegnare ad ogni elemento della lista un nome in due modi

1. direttamente nella funzione **list()**

```
> lista<-list(primo=rep(0,3),secondo= matrix(1:4,nrow=2),terzo=c("Bari","Lecce"))
```

2. attraverso la funzione **names()** che restituisce NULL se la lista è stata creata senza specificare i nomi

```
> names(lista)<-c("primo","secondo","terzo")
```

Quando i nomi sono stati assegnati è possibile estrarre ogni elemento della lista invocando direttamente il proprio nome, attraverso il simbolo **"$"** o con le parentesi quadre, la differenza sta nel tipo di oggetto restituito

```
> lista$secondo  # estrae il secondo elemento
```

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

```
> str(lista$secondo) # è una matrice
```

```
##  int [1:2, 1:2] 1 2 3 4
```

```
> lista["secondo"] # estrae il secondo elemento, ma...
```

```
## $secondo
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
> str(lista["secondo"]) # è una lista
```

```
## List of 1
##  $ secondo: int [1:2, 1:2] 1 2 3 4
```

R - Dataframes

Il dataframe è (probabilmente) l'oggetto più utilizzato di tutto l'ambiente R, almeno in una sua ottica di **gestione** e **analisi dei dati**

Un dataframe è una matrice di dati in cui ad ogni riga corrisponde una osservazione (unità statitica) e ad ogni colonna una variabile statistica, e nell'ambiente viene trattato come una lista

Ogni elemento di tale lista rappresenta una variabile statistica, per cui **length()** restituisce il numero delle variabili, mentre **names()** i rispettivi nomi

Tra le diverse opzioni disponibili, è possibile costruire un dataframe direttamente con la funzione **data.frame()**

```
> df<-data.frame(a=1:3, sesso=c("F","M","F")) # crea un dataframe
> df
```

```
##   a sesso
## 1 1     F
## 2 2     M
## 3 3     F
```

```
> length(df)  # n. variabili
```

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

```
> names(df)   # nome delle variabili
```

```
## [1] "a"     "sesso"
```

```
> dim(df)     # restituisce la dimensione (n. osserv e n. variabili)
```

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

```
> str(df)   # restituisce informazioni circa la struttura di df
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ a    : int  1 2 3
##  $ sesso: chr  "F" "M" "F"
```

Possiamo aggiungere altre variabili (colonne) al dataframe assegnado direttamente un nome alla variabile

```
> df$eta<-c(21,19,20) # aggiunge una variabile di nome "eta"
> df
```

```
##   a sesso eta
## 1 1     F  21
## 2 2     M  19
## 3 3     F  20
```

Subsetting di dataframe

Il dataframe creato è definito da 3 casi e 3 variabili (due numeriche ed una categoriale) che possono essere selezionate in modi diversi

- utilizzando il nome della variabile (ad es., **df$sesso** o **df[,"sesso"]**)
- utilizzando il numero di colonna che la variabile occupa (ad es., **df[,2]**)

Il risultato è comunque un vettore. Provate!

Nelle applicazioni è utile selezionare soltanto una parte del dataframe iniziale, ad es. solo alcune variabili o soltanto alcuni casi che soddisfano determinati criteriati

```
> # selezione dell'età delle sole femmine
> df[df$sesso=="F","eta"]
```

```
## [1] 21 20
```

```
> # selezione delle sole femmine con eta<=20
> # (si fa notare l'uso dell'operatore logico "&"
> # affinchè siano soddisfatte entrambe le condizioni)
> df[df$sesso=="F" & df$eta<=20,]
```

```
##   a sesso eta
## 3 3     F  20
```

```
> #selezione dei valori di "sesso" con eta<20 o eta>20
> # (si fa notare l'uso dell'operatore logico "|"
> # affinchè siano soddisfatte alternativamente le due condizioni)
> df[df$eta<20 | df$eta>20, "sesso"]
```

```
## [1] "F" "M"
```

Alternativamente è possibile selezionare sottoinsiemi di dati utilzzando la funzione **subset(*data*,*condizione*,*selezione*)**

```
> # selezione dei casi con età <=20
> subset(df,eta<=20)
```

```
##   a sesso eta
## 2 2     M  19
## 3 3     F  20
```

```
> # selezione dei valori della variabile sesso per i casi con età <=20
> subset(df,eta<=20, select="sesso")
```

```
##   sesso
## 2     M
## 3     F
```

```
> # selezione dei casi con 19 e 21 anni
> # (si fa notare l'uso dell'operatore logico %in%
> # che permette di specificare una lista di valori
> # come clausole di ricerca in un'unica query)
> subset(df,eta %in% c(19,21))
```

```
##   a sesso eta
## 1 1     F  21
## 2 2     M  19
```

Importazione dei dati

La funzione **data.frame()** vista in precedenza non esaurisce le diverse possibilità per affrontare il problema dell'inserimento e gestione di dati. Ad es. la funzione **as.data.frame()** può essere utilizzata per forzare una matrice di dati ad un dataframe

```
> # Richiamiamo la matrice X1
> str(X1)
```

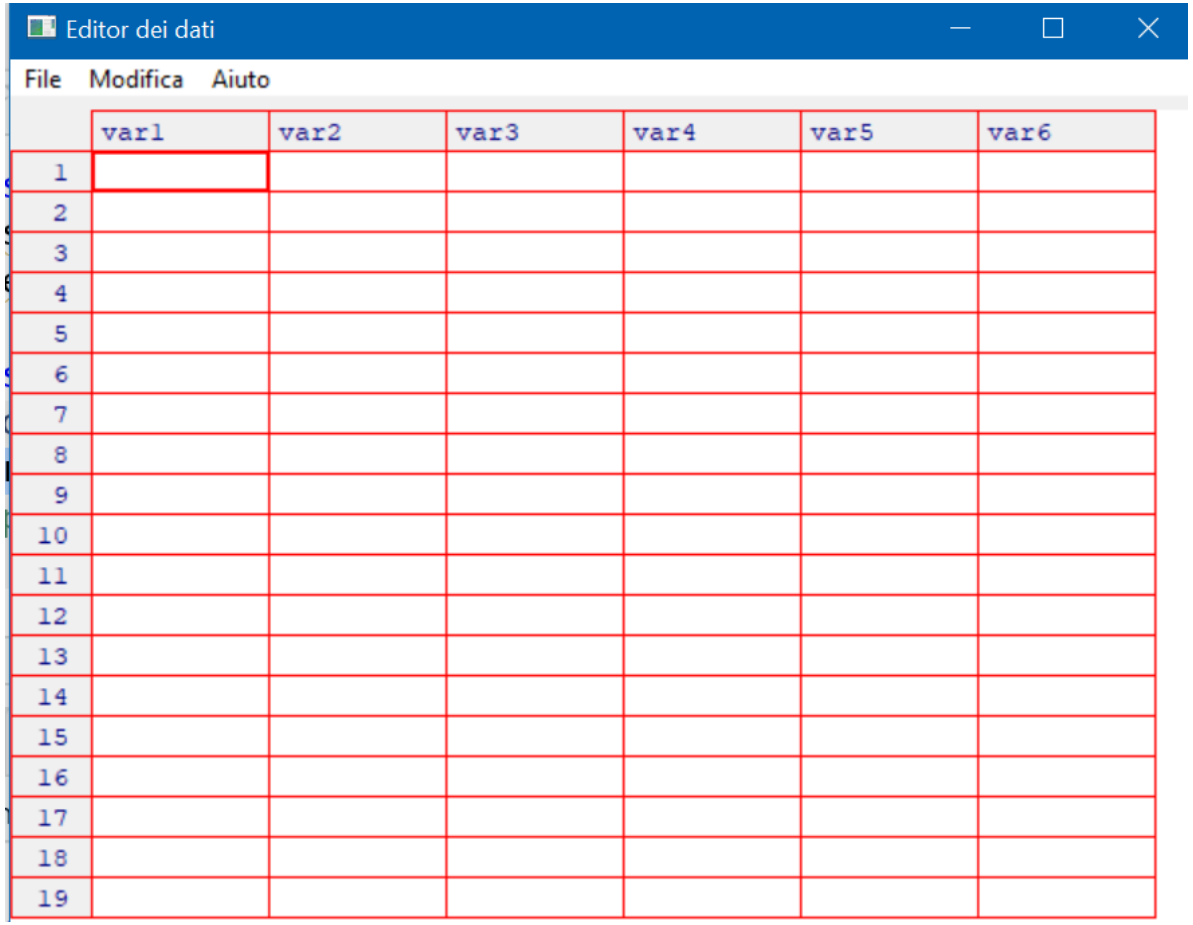
```
##  int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
```

```
> df1<-as.data.frame(X1)
> str(df1)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ V1: int  1 2 3 4
##  $ V2: int  5 6 7 8
##  $ V3: int  9 10 1 2
```

I dati possono essere passati ad **R** anche attraverso l'inserimento manualmente utilizzando un foglio elettronico

```
> X<-data.frame()
> fix(X) # apre un Editor di dati
>        # che permette l'inserimento dei dati
>        # cella per cella
```



Un'altra possibilità (probabilmente la più frequente nelle applicazioni), è quella di importare i dati da un file ASCII (.txt, .asc o .csv). A tale scopo viene utilizzata la funzione **read.table()** o **read.delim()**

```
> stud<-read.table(file.choose(),
+    header=TRUE,
+    sep="\t",
+    na.strings = "NA",
+    dec=",")
```

```
> str(stud)
```

```
## 'data.frame':    20 obs. of  7 variables:
##  $ id    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Sex   : chr  "Female" "Male" "Male" "Male" ...
##  $ Age   : num  18.2 17.6 16.9 20.3 23.7 ...
##  $ Height: num  173 178 NA 160 165 ...
##  $ WHnd  : chr  "Right" "Left" "Right" "Right" ...
##  $ Pulse : int  92 104 87 NA 35 64 83 74 72 90 ...
##  $ Smoke : chr  "Never" "Regul" "Occas" "Never" ...
```

In questo caso, il file *"Students.txt"* viene importato e automaticamente convertito nel dataframe *stud*

Alcuni argomenti della funzione sono

- il **path** (o percorso del file)
  - può essere specificato e scritto come negli ambienti Unix (o Linux) con "/" (ad es. *"E:/Laboratorio di analisi dei dati con R/ Data/studenti.txt"*)
  - il percorso viene dinamicamente scelto attraverso la funzione **file.choose()** che apre la finestra di navigazione
- **header**=TRUE specifica che la prima linea del file contiene le intestazioni delle colonne
- **sep**="\t" indica che i diversi campi sono separati da un tab (altre opzioni posso essere ad es. ";",",")
- **na.strings**="NA" è utile se nel file sono presenti valori mancanti, in questo caso individuati con NA
- **dec**="," specifica il tipo di carattere utilizzato nel file per separare i decimali, in questo caso la virgola

Se il file di input è in formato ".csv" si utilizza la funzione **read.csv()** con la stessa logica vista in precedenza

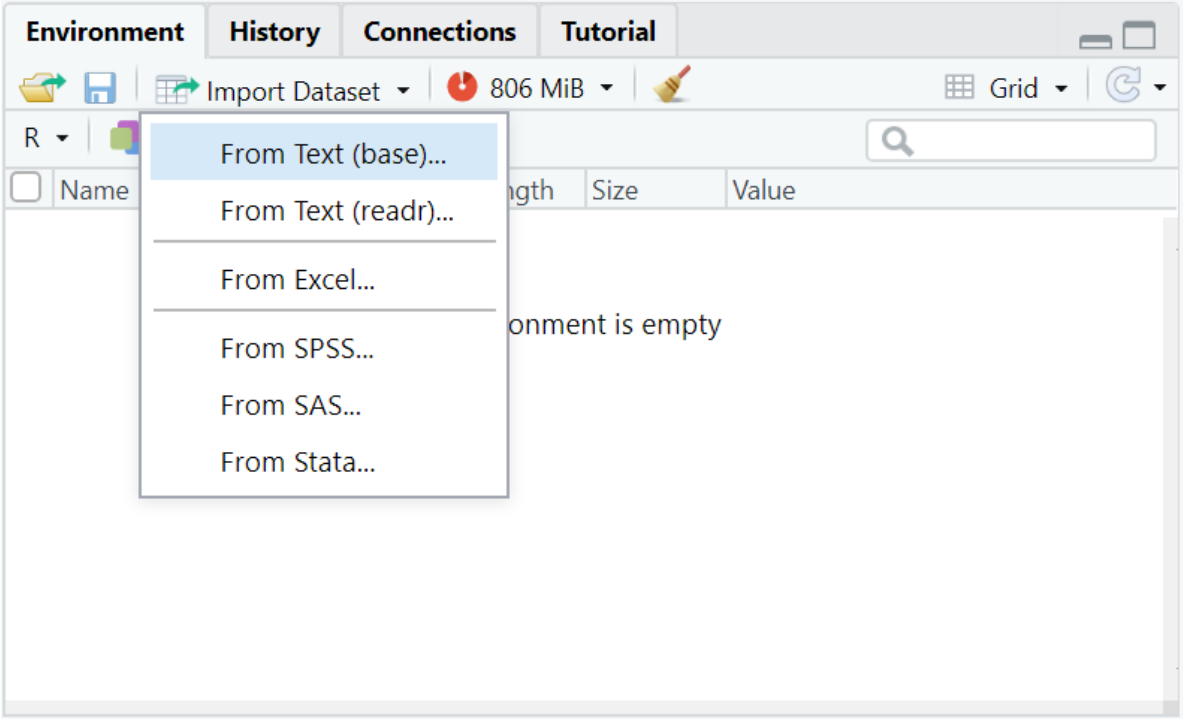
```
> salary<-read.csv(file.choose(),
+    header=TRUE,
+    sep=",")
```

```
> str(salary)
```

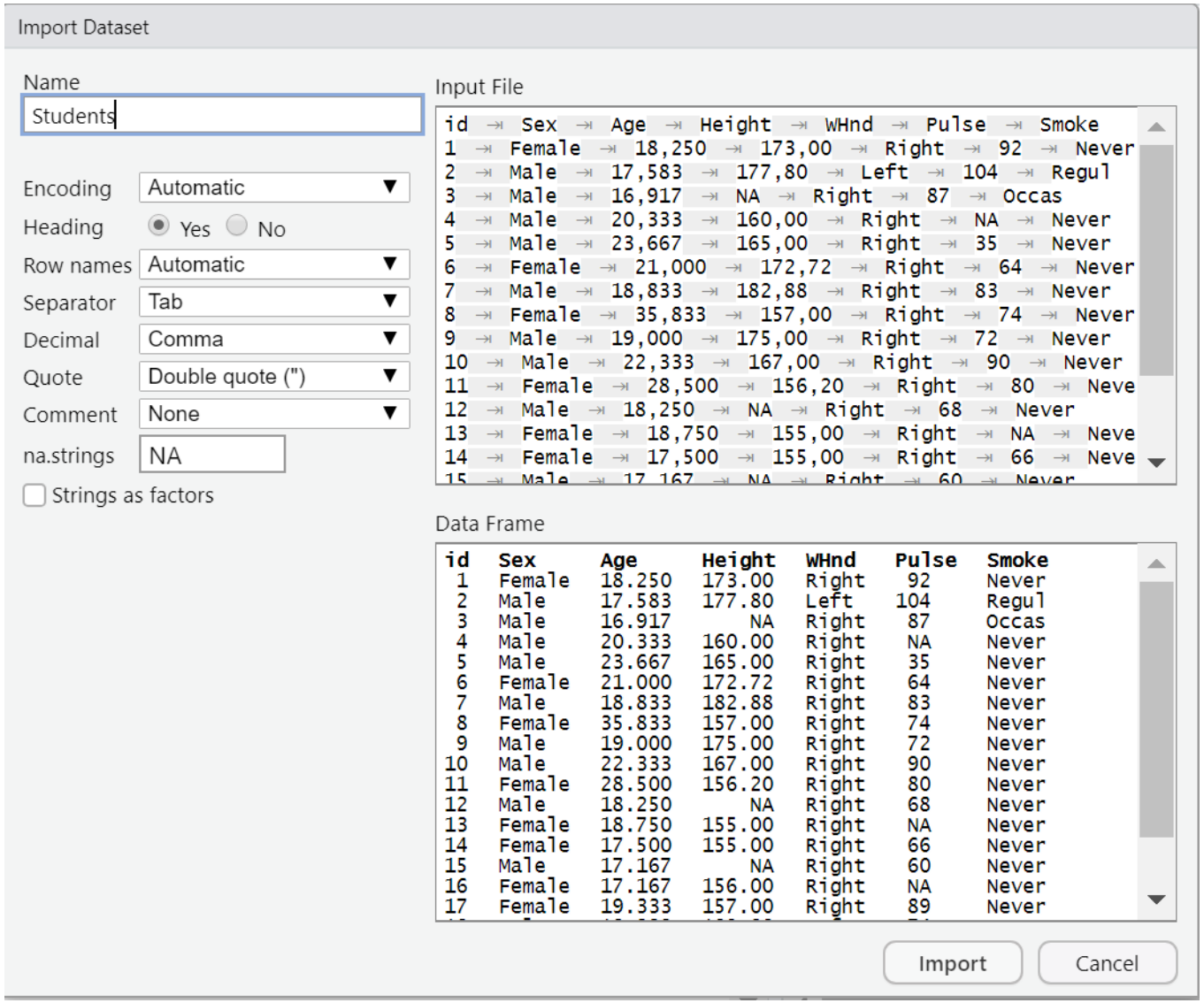
```
## 'data.frame':    8 obs. of  5 variables:
##  $ id        : int  1 2 3 4 5 6 7 8
##  $ name      : chr  "Rick" "Dan" "Michelle" "Ryan" ...
##  $ salary    : num  623 515 611 729 843 ...
##  $ start_date: chr  "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
##  $ dept      : chr  "IT" "Operations" "IT" "HR" ...
```

`RStudio` permette di importare i file di input attraverso una procedura guidata. È possibile accedere alle funzionalità di importazione dei dati dal pannello **Environment**. Gli importatori sono raggruppati in 3 categorie: dati di testo, dati di Excel e dati statistici

Per accedere a questa funzione, si utilizza il menu a tendina "Import Dataset" dal panello **Environment**



L'importazione tramite "From text (base)" consente di importare file di testo (anche .csv) utilizzando il pacchetto di **base**

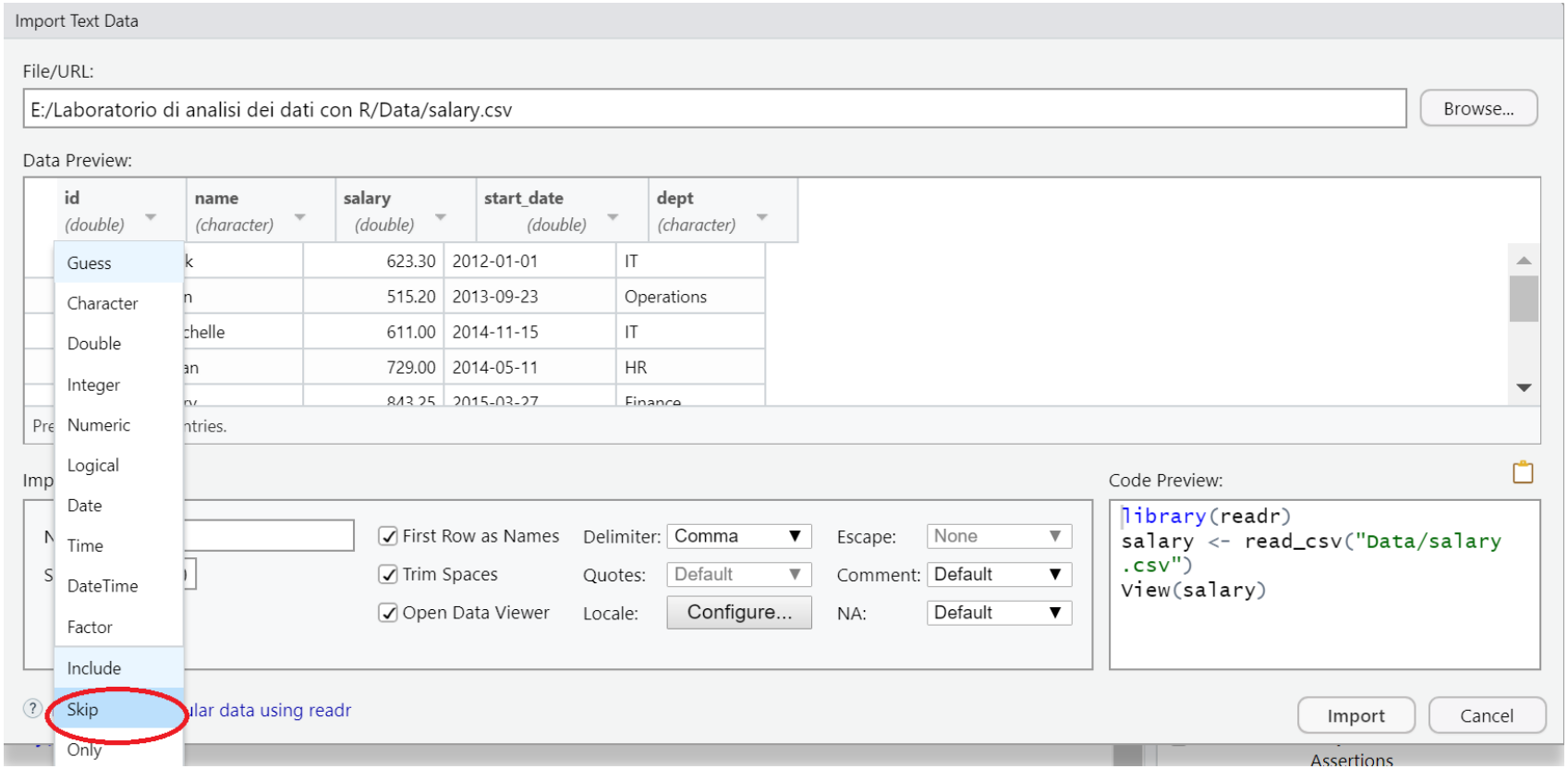


L'importazione di file "From text (readr)" consente di importare file .csv e in generale file delimitati da caratteri, utilizzando il pacchetto **readr**. Questo importatore di testi fornisce supporto per

- importare i dati dal file system o da un URL
- modificare il tipo di dati della colonna
- saltare o includere solo alcune colonne
- rinominare il set di dati
- saltare le prime N righe
- usare la riga di intestazione per i nomi delle colonne
- tagliare gli spazi nei nomi
- cambiare il delimitatore di colonna
- selezionare citazioni, escape, commenti e identificatori NA

Sempre dall'Import data si può selezionare l'opzione "From text (readr)" che apre la finestra dove è possibile specificare le modalità di importazione del file di dati

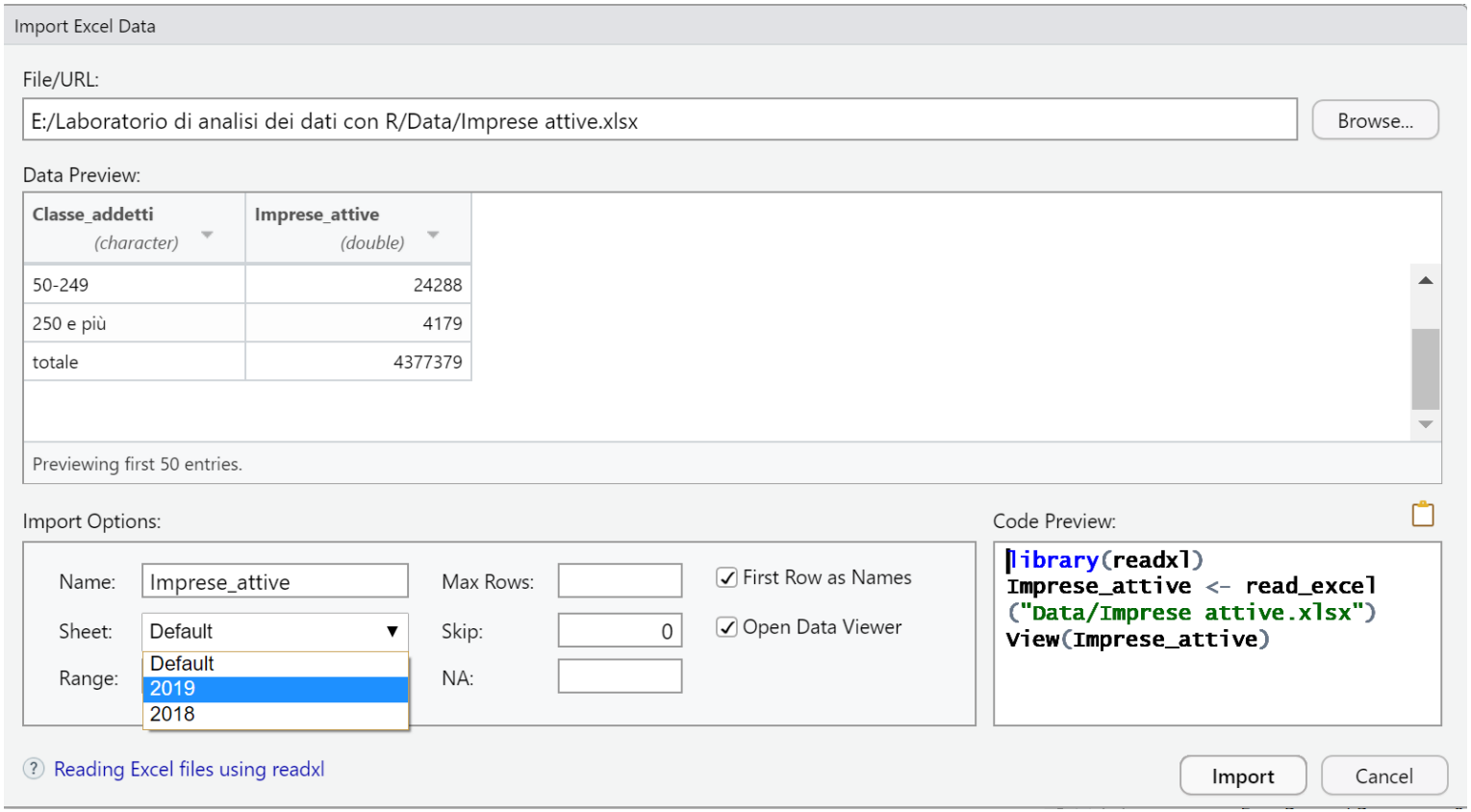
Nell'esempio, si vuole importare il file "salary.csv" escludendo la prima colonna



L'importazione tramite "From Excel (base)" consente di importare file con estension ".xlsx" utilizzando il pacchetto **readxl**

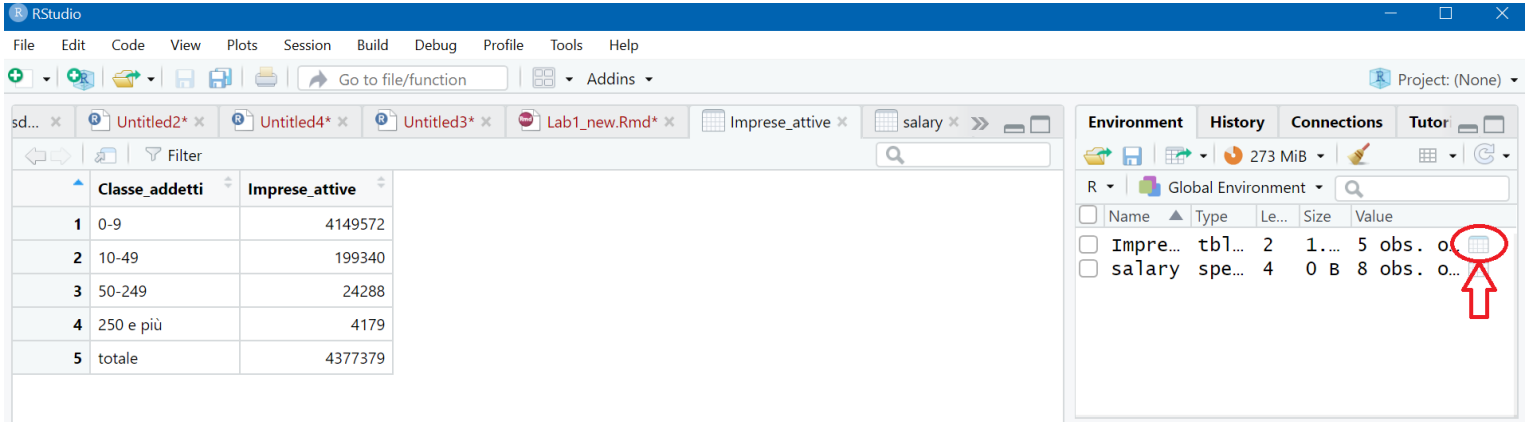
Anche in questo caso l'acquisizione dei dati avviene inserendo nella finestra di importazione, le opzioni per l'importazione del file di dati

Nell'esempio, si importano i dati contenuti nel file "Imprese attive.xlsx", scegliendo lo spreadsheet "2019"



N.B. Poichè non interagiremo in questo incontro con altri software statistici, la trattazione degli imporatori di dati statistici (da SPSS, SAS, Stata) è rimandata all'approfondimento personale

I dati importati in `RStudio`, possono essere visulalizzati scrivendo nella Console il comando **view(*nome_dataset*)** oppure cliccando nell'Environment sulla griglia accanto al dataset



Accanto alle precendenti modalità di acquisizione dei dati, esiste la possibilità di utilizzare dataset contenuti nei pacchetti di R. Ad es., consideriamo il pacchetto "datasets" di R

```
> library(datasets)
> library(help = "datasets")  # apre la lista dei dataset contenuti
>                             # nel pacchetto
```

Per accedere ad uno dei dataset, si utilizza la funzione **data(*nome_dataset*)**

```
> data(cars)
> cars
> ?cars    # descrizione del dataset
```

Questi dati possono quindi essere esplorati, analizzati e manipolati attraverso alcune funzioni (di base) quali

```
> str(cars)  # struttura del dataset
> dim(ca)    # dimensione del dataset
> head(cars) # prime righe
> cars$speed # seleziona la variabile speed
> cars[20:30,]# seleziona le osservazioni da 20 a 30
```

Organizzazione del lavoro

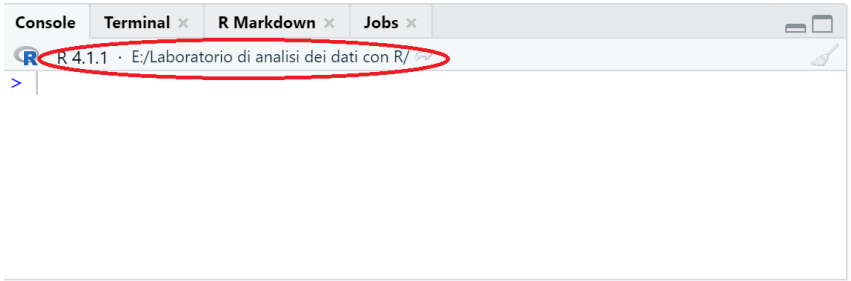
Quando si lavora in `RStudio` è utile specificare la directory di lavoro, ovvero la cartella dove salvare i file relativi ad una sessione di lavoro. Con il comando **getwd()** viene restituito il **path** dell'attuale directory di lavoro che è possibile cambiare attraverso **setwd()**

```
> getwd()  # questa è l'attuale directory di lavoro
```

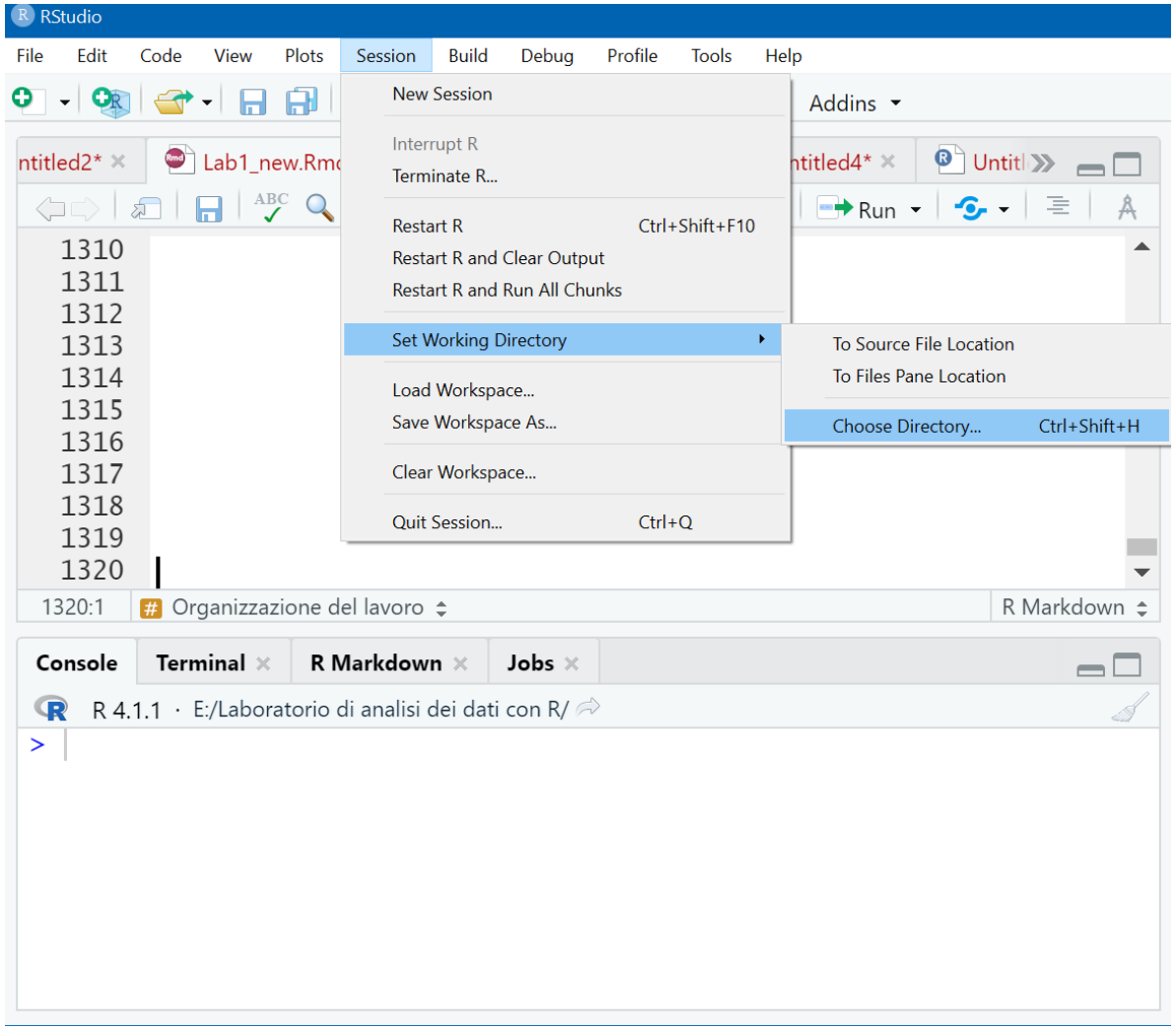
```
## [1] "E:/Orientamento consapevole/Orientamento consapevole 2022/LEZ I
```

```
> setwd("E:/Varie/")  # questa è la nuova directory di lavoro
```

L'attuale directory di lavoro è anche riportata nella Console



per cambiarla è possibile utilizzare anche il menu a tendina **Session | Set working directory | Choose Directory | la mia cartella**



Una volta impostata la directory di lavoro, possiamo iniziare creare degli **script** nella stessa interfaccia grafica

Lo **script** non è altro che un file di testo che contiene tutti i comandi (di `R`) che è necessario far girare, passando per la Console di `R`. Questo file viene salvato con l'estensione "*.R*", e può essere modificato e riutilizzato anche in un diverso progetto di analisi dei dati

E' buona pratica lavorare sempre con uno script, evitando di eseguire i comandi nella Console dopo il prompt, ma riportarli nello script e poi eseguirli. Le ragioni sono fondalmente due

- il codice, in questo modo, può essere immediatamente corretto, modificato e rieseguito, nel caso produca risultati diversi da quelli attesi o siano presenti errori di battitura
- il file può essere salvato, riutilizzato e condiviso

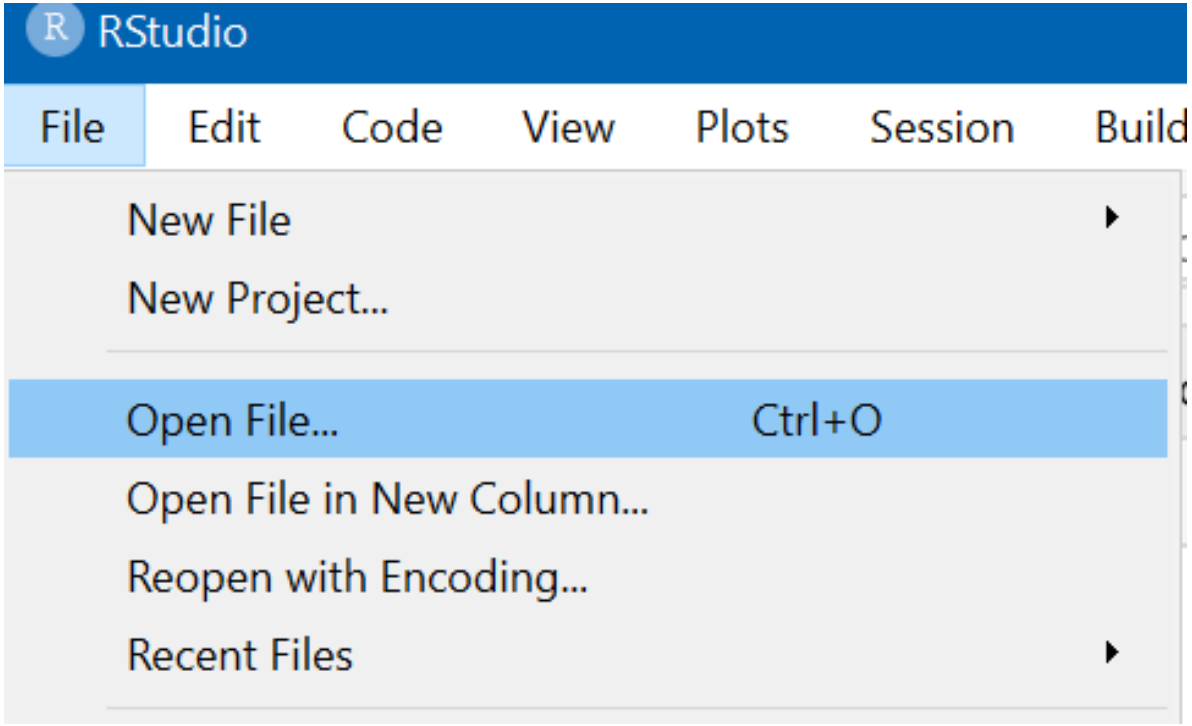
Selezionando **File|New File|R Script** apriamo in `RStudio` una nuova finestra che permette di editare (scrivere) uno script

RStudio

File | Edit | Code | View | Plots | Session | Build | Debug | Profile | Tools | Help

- New File ▸
  - R Script — Ctrl+Shift+N
  - R Notebook
  - R Markdown...
  - Shiny Web App...
  - Plumber API...
  - C File
  - C++ File
  - Header File
  - Markdown File
  - HTML File
  - CSS File
  - JavaScript File
  - D3 Script
  - Python Script
  - Shell Script
  - SQL Script
  - Stan File
  - Text File
  - R Sweave
  - R HTML
  - R Presentation
  - R Documentation...
- New Project...
- Open File... — Ctrl+O
- Open File in New Column...
- Reopen with Encoding...
- Recent Files ▸
- Open Project...
- Open Project in New Session...
- Recent Projects ▸
- Import Dataset ▸
- Save — Ctrl+S
- Save As...
- Save with Encoding...
- Save All — Ctrl+Alt+S
- Knit Document — Ctrl+Shift+K
- Compile Report...
- Print...
- Close — Ctrl+W
- Close All — Ctrl+Shift+W
- Close All Except Current — Ctrl+Alt+Shift+W
- Close Project

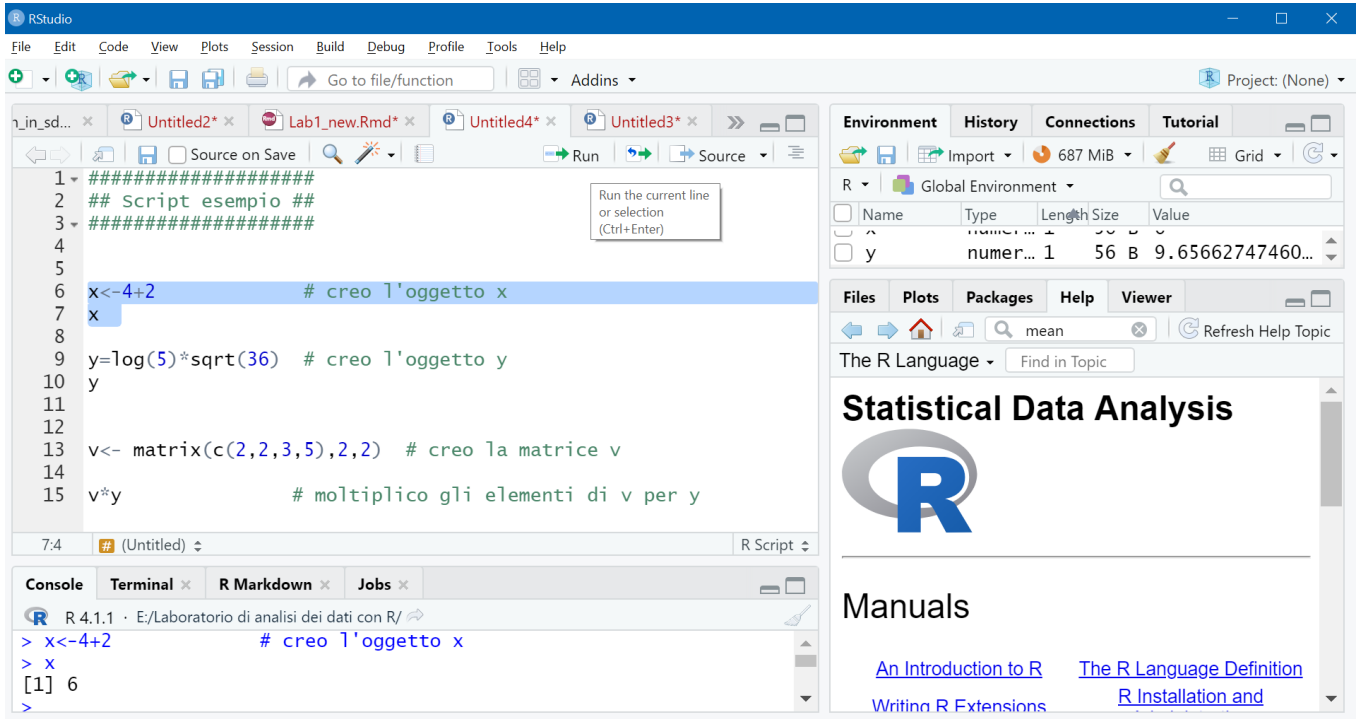
Se invece si vuole accedere ad uno script già salvato selezionando **File | Open File** si aprirà la finestra di navigazione che permette di visualizzare (e scegliere) il file desiderato



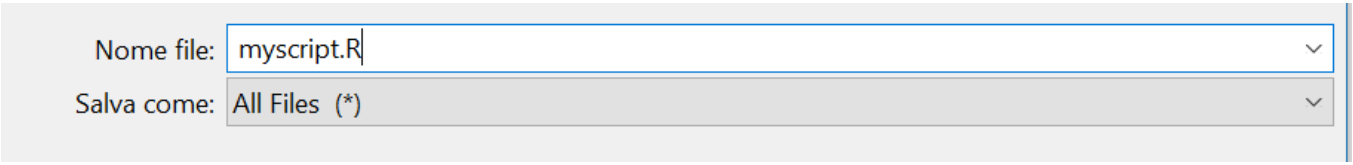
Lo script appare in `RStudio` nel riguardo in alto a sinistra

E' possibile ora iniziare a scrivere i proprio comandi o modificare quelli già salvati

Per eseguire un comando, basta posizionarsi con il cursore sul comando (o su una selezione di più comandi) e premere "Run" oppure Crtl+Enter



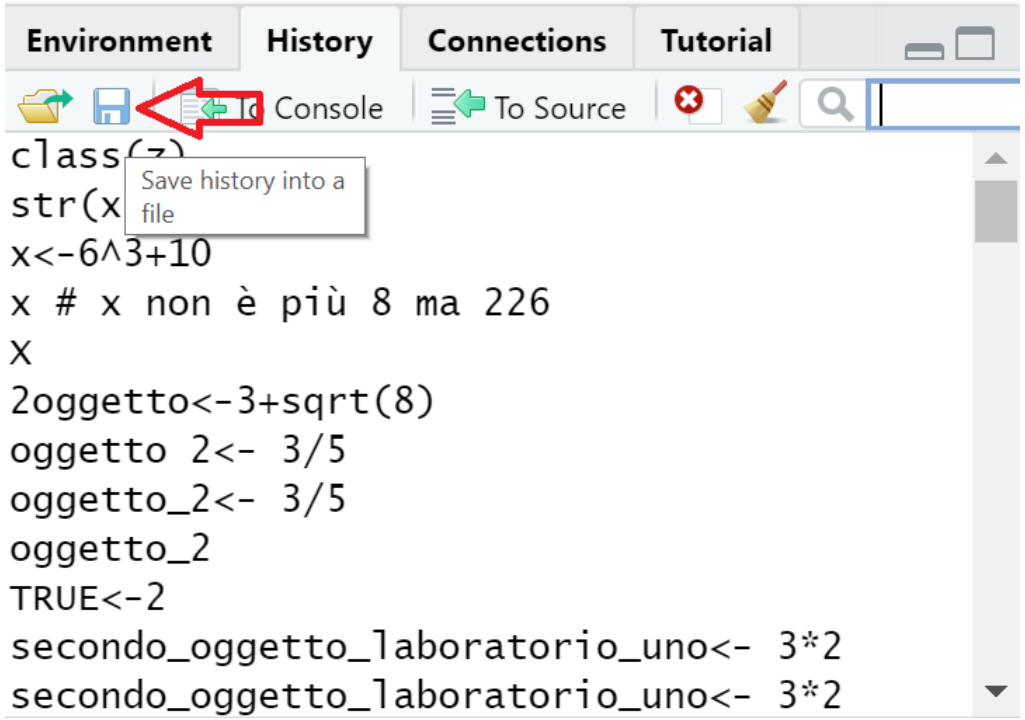
Si ribadisce che lo script va salvato con estensione ".R" selezionando dal menu a tendina **File** | **Save as** | **"myscript.R"**



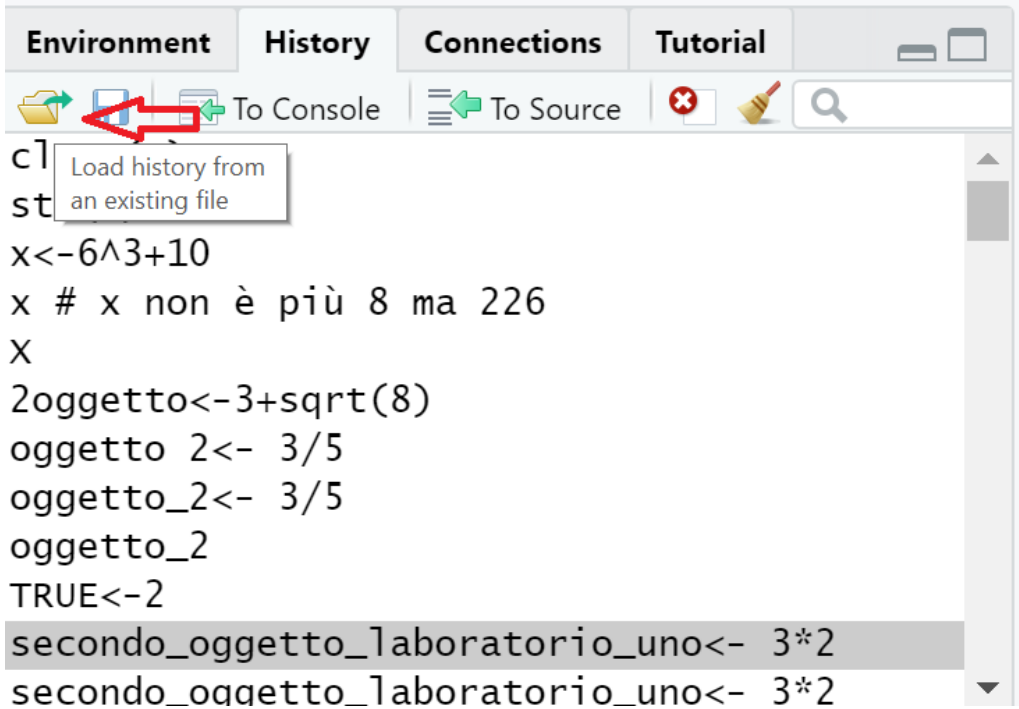
Prima di chiudere la sessione di lavoro, può essere necessario salvare sia la Command history (o semplicemente History) che il Workspace

- la Command history viene salvata con estensione ".Rhistory"

Questo file viene creato automaticamente al momento della chiusura della sessione quando quest'ultima viene salvata attraverso il comando **q("yes")** , oppure può essere esplicitamente creato durante la sessione di lavoro selezionando, dal pannello **History**, **Save history in to a file** o utilizzando la funzione **savehistory(file="myHistory.Rhistory")**

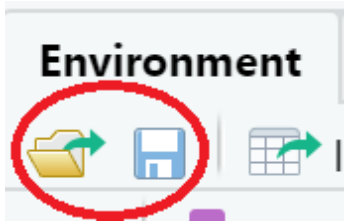


Per richiamare la History così creata durante una nuova sessione di lavoro si può usare **loadhistory(file="myHistory.Rhistory")** oppure importare lo stesso file selezionando loadhistory dal panello History



- il workspace viene salvato con estensione ".Rdata"

Il file può essere salvato in qualsiasi momento durante la sessione di lavoro utilizzando la funzione **save.image(file="myWorkSpace.RData")** e può essere caricato in una nuova sessione di lavoro utilizzando **load("myWorkSpace.RData")**; alternativamente, si possono eseguire le stesse operazioni utilizzando i comandi veloci nel pannello **Environment** (così come per la History)



Ogni sessione di lavoro può essere chiusa dal pulsante di chiusura "x" oppure digitando il comando **q()** nella Console. In entrambi i casi il programma chiederà se si vuole salvare il Workspace. In caso affermativo, insieme al file ".RData" viene salvato in automatico anche il file ".RHistory"