

# Introduzione a R

Corso per imparare le basi di **R**

Psicostat

15-03-2021





# Contents

<b>Presentazione</b>	<b>5</b>
Perchè R . . . . .	5
Struttura del libro . . . . .	5
Risorse Utili . . . . .	6
Psicostat . . . . .	7
Collaborazione . . . . .	7
Riconoscimenti . . . . .	7
Licenza . . . . .	7
<b>Get Started</b>	<b>11</b>
<b>Introduzione</b>	<b>11</b>
<b>1 Installare R e RStudio</b>	<b>13</b>
1.1 Installare R . . . . .	13
1.2 Installare R Studio . . . . .	18
<b>2 Interfaccia RStudio</b>	<b>21</b>
<b>3 Primi Passi in R</b>	<b>31</b>
3.1 Operatori Matematici . . . . .	31
3.2 Operatori Relazionali e Logici . . . . .	34

<b>4 Due Compagni Inseparabili</b>	<b>39</b>
4.1 Oggetti . . . . .	39
4.2 Funzioni . . . . .	44
<b>5 Sessione di Lavoro</b>	<b>45</b>
5.1 Infobox . . . . .	45
5.2 Problema + Google = Soluzione . . . . .	47
<b>Struttura Dati</b>	<b>53</b>
<b>Introduzione</b>	<b>53</b>
<b>6 Vettori</b>	<b>55</b>
<b>7 Matrici</b>	<b>57</b>
<b>8 Dataframe</b>	<b>59</b>
<b>9 Liste</b>	<b>61</b>
<b>Algoritmi</b>	<b>65</b>
<b>Introduzione</b>	<b>65</b>
<b>10 Definizione di Funzioni</b>	<b>67</b>
<b>11 Programmazione Condizionale</b>	<b>69</b>
<b>12 Attenti al loop</b>	<b>71</b>
<b>Case study</b>	<b>75</b>
<b>Introduzione</b>	<b>75</b>
<b>13 Caso Studio I: Attaccamento</b>	<b>77</b>

# Presentazione

In questo libro impareremo le basi di *R*, uno dei migliori software per la visualizzazione e l'analisi statistica dei dati. Partiremo da zero introducendo gli aspetti fondamentali di R e i concetti alla base di ogni linguaggio di programmazione che ti permetteranno in seguito di approfondire e sviluppare le tue abilità in questo bellissimo mondo.

## Perchè R

Ci sono molte ragioni per cui scegliere R rispetto ad altri programmi usati per condurre le analisi statistiche. Innanzitutto è un linguaggio di programmazione (come ad esempio Python, Java, C++, o Julia) e non semplicemente un'interfaccia punta e clicca (come ad esempio SPSS o JASP). Questo comporta si maggiori difficoltà iniziali ma ti ricompenserà in futuro poichè avari imparato ad utilizza uno strumennto molto potente.

Inoltre, R è:

- nato per la statistica
- open-source
- ricco di pacchetti
- supportato da una grande comunità
- gratis

## Struttura del libro

Il libro è suddiviso in quattro sezioni principali:

- **Get started.** Una volta installato R ed RStudio, famiglierizzeremo con l'ambiente di lavoro introducendo alcuni aspetti generali e le funzioni principali. Verranno inoltre descritte alcune buone regole per iniziare una sessione di lavoro in R.

- **Struttura dei dati.** Impareremo gli oggetti principali che R utilizza al suo interno. Variabili, vettori, matrici, dataframe e liste non avranno più segreti e capiremo come manipolarli e utilizzarli a seconda delle varie necessità.
- **Algoritmi.** Non farti spaventare da questo nome. Ne avrai spesso sentito parlare come qualcosa di molto complicato, ma in realtà gli algoritmi sono semplicemente una serie di istruzioni che il computer segue quando deve eseguire un determinato compito. In questa sezione vedremo i principali comandi di R usati per definire degli algoritmi. Questo è il vantaggio di conoscere un linguaggio di programmazione, ci permette di creare nuovi programmi che il computer eseguirà per noi.
- **Case study.** Esegiremo passo per passo un analisi che ci permetterà di imparare come importare i dati, codificare le variabili, manipolare e preparare i dati per le analisi, condurre delle analisi descrittive e creare dei grafici.

Alla fine di questo libro probabilmente non sarete assunti da Google, ma speriamo almeno che R non vi faccia più così paura e che magari a qualcuno sia nato l'interesse di approfondire questo fantastico mondo fatto di linee di codice.

## Risorse Utili

Segnaliamo qui per il lettore interessato del materiale online (in inglese) per approfondire le conoscenze sull'uso di R.

Materiale introduttivo:

- *R for Psychological Science* di Danielle Navarro <https://psyr.djnavarro.net/index.html>
- *Hands-On Programming with R* di Garrett Grolemund <https://rstudio-education.github.io/hopr/>

Materiale intermedio:

- *R for Data Science* di Hadley Wickham e Garrett Grolemund <https://r4ds.had.co.nz/>

Materiale avanzato:

- *R Packages* di Hadley Wickham e Jennifer Bryan <https://r-pkgs.org/>
- *Advanced R* di Hadley Wickham <https://adv-r.hadley.nz/>

## Psicostat

Questo libro è stato prodotto da **Psicostat**, un gruppo di ricerca interdisciplinare dell'università di Padova che unisce la passione per la statistica e la psicologia. Se vuoi conoscere di più riguardo le nostre attività visita il nostro sito <https://psicostat.dpss.psy.unipd.it/> o aggiungiti alla nostra mailing list <https://lists.dpss.psy.unipd.it/postorius/lists/psicostat.lists.dpss.psy.unipd.it/>.

## Collaborazione

Se vuoi collaborare alla revisione e scrittura di questo libro (ovviamente è tutto in R) visita la nostra repository di Github <https://github.com/psicostat/Introduction2R>.

## Riconoscimenti

Il template di questo libro è basato su Rstudio Bookdown-demo rilasciato con licenza CC0-1.0 e rstudio4edu-book rilasciato con licenza CC BY. Nota che le illustrazioni utilizzate nelle vignette appartengono sempre a rstudio4edu-book e sono rilasciate con licenza CC BY-NC.

## Licenza

Questo libro è rilasciato sotto la Creative Commons Attribution-ShareAlike 4.0 International Public License (CC BY-SA). Le illustrazioni utilizzate nelle vignette appartengono a rstudio4edu-book e sono rilasciate con licenza CC BY-NC.



# **Get Started**



# Introduzione

In questa sezione verranno presentate le istruzioni per installare R ed RStudio. In seguito, famigerizzeremo con l'ambiente di lavoro introducendo alcuni aspetti generali e le funzioni principali. Verranno inoltre descritte alcune buone regole per iniziare una sessione di lavoro in R.

I capitoli sono così organizzati:

- **Capitolo 1 - Installare R e RStudio.** Istruzioni passo a passo per installare R e RStudio
- **Capitolo 2 - Interfaccia RStudio.** Introduzione all'interfaccia utente di RStudio.
- **Capitolo 3 - Primi Passi in R.** Operatori matematici, operatori relazionali, operatori logici.
- **Capitolo 4 - Due Compagni Inseparabili.** Introduzione dei concetti di oggetti e funzioni in R.
- **Capitolo 5 - Sessione di Lavoro.** Utilizzo degli script e introduzione dei concetti di working directory e pacchetti di R.



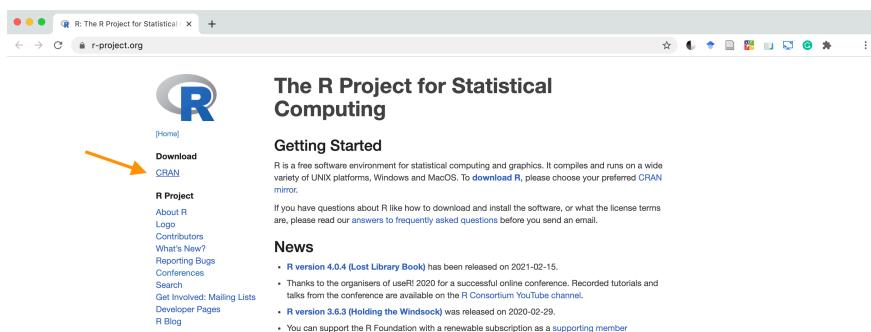
# Chapter 1

## Installare R e RStudio

R ed R-studio sono due software distinti. R è un linguaggio di programmazione usato in particolare in ambiti quali la statistica. R-studio invece è un'interfaccia *user-friendly* che permette di utilizzare R. R può essere utilizzato autonomamente tuttavia è consigliato l'utilizzo attraverso R-studio. Entrambi vanno installati separatamente e la procedura varia a seconda del proprio sistema operativo (Windows, MacOS o Linux). Riportiamo le istruzioni solo per Windows e MacOS Linux (Ubuntu). Ovviamente R è disponibile per tutte le principali distribuzioni di Linux. Le istruzioni riportate per Ubuntu (la distribuzione più diffusa) sono valide anche per le distribuzioni derivate.

### 1.1 Installare R

1. Accedere al sito <https://www.r-project.org>
2. Selezionare la voce **CRAN** (Comprehensive R Archive Network) dal menù di sinistra sotto **Download**



3. Selezionare il primo link <https://cloud.r-project.org/>

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

<b>0-Cloud</b> <a href="https://cloud.r-project.org/">https://cloud.r-project.org/</a> <span style="color: orange;">(arrow)</span> Algeria <a href="https://cran.usuthb.dz/">https://cran.usuthb.dz/</a> Argentina <a href="http://mirror.fcaglp.unlp.edu.ar/CRAN/">http://mirror.fcaglp.unlp.edu.ar/CRAN/</a> ...	Automatic redirection to servers worldwide, currently sponsored by Rstudio University of Science and Technology Houari Boumediene Universidad Nacional de La Plata
--	--

#### 4. Selezionare il proprio sistema operativo

The Comprehensive R Archive Network

**Download and Install R**

Precompiled binary distributions of the base system and contributed packages. Windows and Mac users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#) (arrow)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

**Source Code for all Platforms**

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

##### 1.1.1 R Windows

###### 1. Selezionare la voce **base**

R for Windows

**Subdirectories:**

- [base](#) (arrow)
- [contrib](#)
- [old\\_contrib](#)
- [Rtools](#)

Binaries for base distribution. This is what you want to [install R for the first time](#). Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on [third party software](#) available for CRAN Windows services and corresponding environment and make variables.

Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).

Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the [R FAQ](#) and [R for Windows FAQ](#).

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

###### 2. Selezionare la voce **Download** della versione più recente di R disponibile

R-4.0.4 for Windows (32/64 bit)

**Download R 4.0.4 for Windows** (85 megabytes, 32/64 bit)

[Installation and other instructions](#)  
[New features in this version](#)

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the [md5sum](#) of the .exe to the [fingerprint](#) on the master server. You will need a version of md5Sum for windows: both [graphical](#) and [command line versions](#) are available.

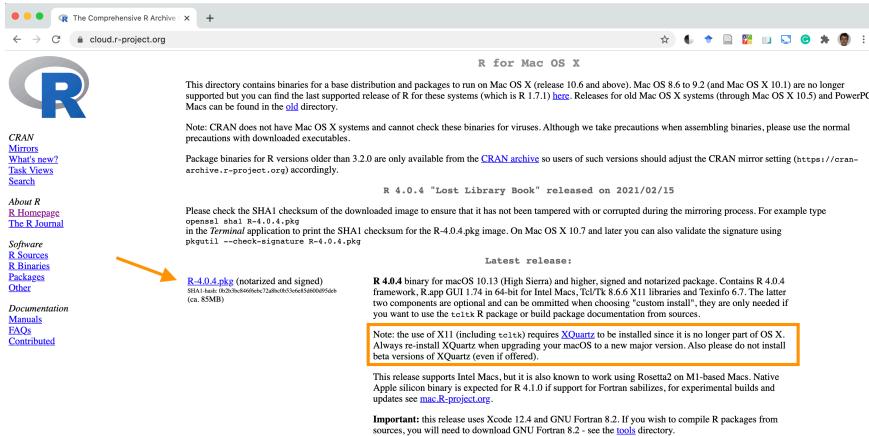
**Frequently asked questions**

- [Does R run under my version of Windows?](#)
- [How do I update packages in my previous version of R?](#)
- [Should I run 32-bit or 64-bit R?](#)

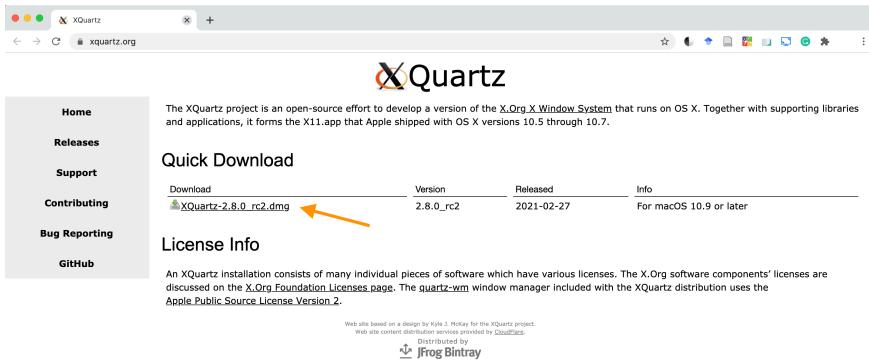
###### 3. Al termine del download, eseguire il file e seguire le istruzioni fino al termine dell'installazione

### 1.1.2 R MacOS

1. Selezionare della versione più recente di R disponibile



2. Al termine del download, eseguire il file e seguire le istruzioni fino al termine dell'installazione di R
3. Successivamente è necessario installare anche una componente aggiuntiva **XQuartz** premendo il link all'interno del riquadro arancione riportato nella figura precedente
4. Selezionare la voce Download



5. Al termine del download, eseguire il file e seguire le istruzioni fino al termine dell'installazione

### 1.1.3 R Linux

Nonostante la semplicità di installazione di pacchetti su Linux, R a volte potrebbe essere più complicato da installare per via delle diverse distribuzioni,

repository e chiavi per riconoscere la repository come sicura.

Sul **CRAN** vi è la guida ufficiale con tutti i comandi **apt** da eseguire da terminale. Seguendo questi passaggi non dovrebbero esserci problemi.

1. Andate sul CRAN
2. Cliccate [Download R for Linux](#)
3. Selezionate la vostra distribuzione (Ubuntu in questo caso)
4. Seguite le istruzioni, principalmente eseguendo i comandi da terminale suggeriti

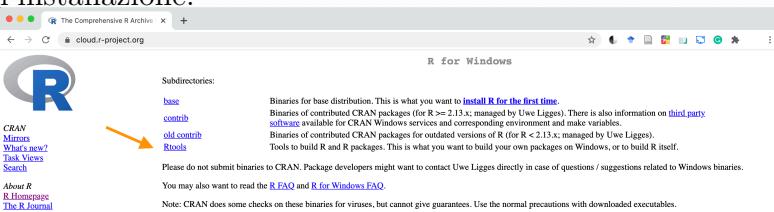
Per qualsiasi difficoltà o errore, soprattutto con il mondo Linux, una ricerca su online risolve sempre il problema.

 Approfondimento: R Tools

Utilizzi avanzati di R richiedono l'installazione di una serie ulteriore software definiti **R tools**.

### Windows

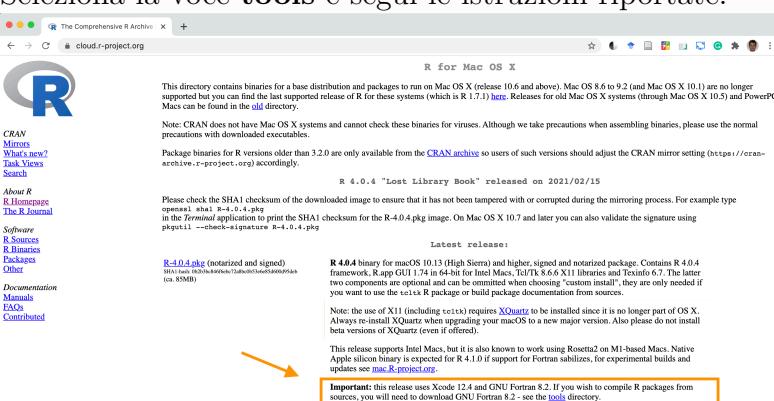
Seleziona la voce **Rtools** e segui le istruzioni per completare l'installazione.



Nota che sono richieste anche delle operazioni di configurazione affinchè tutto funzioni correttamente.

### MacOS

Seleziona la voce **tools** e segui le istruzioni riportate.

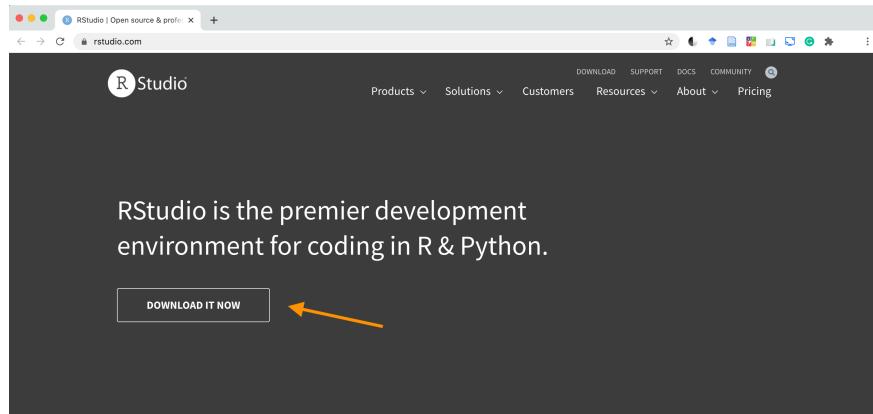


Nota in particolare che con R 4.0 le seguenti indicazioni sono riportate.

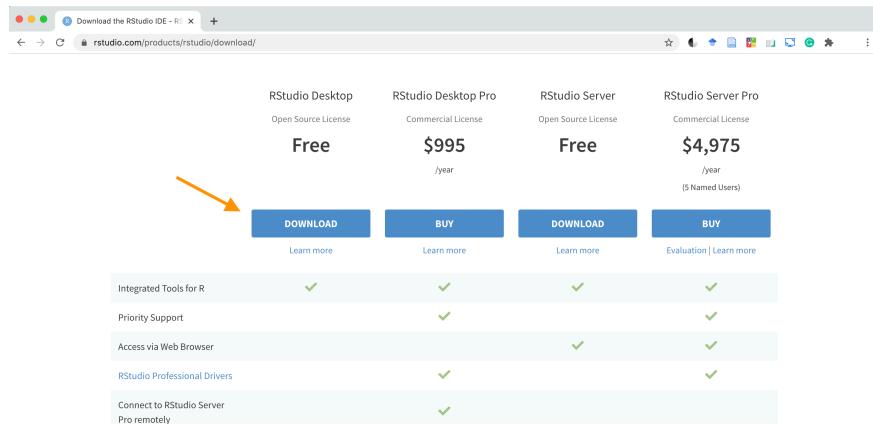


## 1.2 Installare R Studio

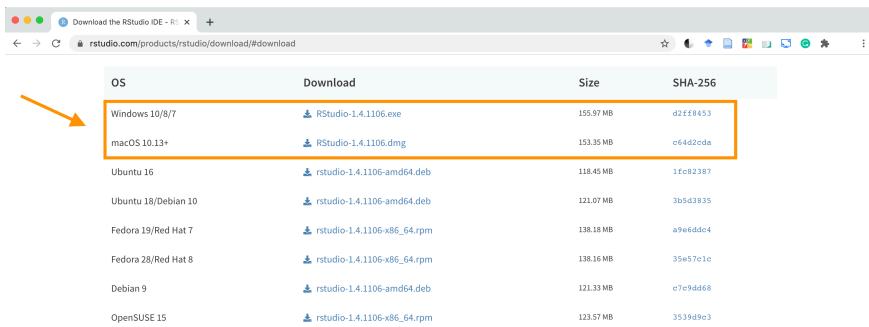
1. Accedere al sito <https://rstudio.com>
2. Selezionare la voce **DOWNLOAD IT NOW**



3. Selezionare la versione gratuita di RStudio Desktop



4. Selezionare la versione corretta a seconda del proprio sistema operativo



OS	Download	Size	SHA-256
Windows 10/8/7	<a href="#">RStudio-1.4.1106.exe</a>	155.97 MB	d2ff8453
macOS 10.13+	<a href="#">RStudio-1.4.1106.dmg</a>	153.35 MB	c64d2cda
Ubuntu 16	<a href="#">rstudio-1.4.1106-amd64.deb</a>	118.45 MB	1fc82387
Ubuntu 18/Debian 10	<a href="#">rstudio-1.4.1106-amd64.deb</a>	121.07 MB	3b5d3835
Fedora 19/Red Hat 7	<a href="#">rstudio-1.4.1106-x86_64.rpm</a>	138.18 MB	a9e6ddc4
Fedora 28/Red Hat 8	<a href="#">rstudio-1.4.1106-x86_64.rpm</a>	138.16 MB	35e57c1c
Debian 9	<a href="#">rstudio-1.4.1106-amd64.deb</a>	121.33 MB	c7e9dd68
OpenSUSE 15	<a href="#">rstudio-1.4.1106-x86_64.rpm</a>	123.57 MB	3539d9c3

5. Al termine del download, eseguire il file e seguire le istruzioni fino al termine dell'installazione

### 1.2.1 R Studio in Linux

In questo caso, come su Windows e MacOS l'installazione consiste nello scaricare ed eseguire il file corretto, in base alla distribuzione (ad esempio **.deb** per Ubuntu e derivate). Importante, nel caso di Ubuntu (ma dovrebbe valere anche per le altre distribuzioni) anche versioni successive a quella indicata (es. Ubuntu 16) sono perfettamente compatibili.



## Chapter 2

# Interfaccia RStudio

In questo capitolo presenteremo l’interfaccia utente di RStudio. Molti aspetti che introdurremo brevemente qui verranno discussi nei successivi capitoli. Adesso ci interessa solo familiarizzare con l’interfaccia del nostro strumento di lavoro principale ovvero RStudio.

Come abbiamo visto nel Capitolo 1, R è il vero “motore computazionale” che ci permette di compiere tutte le operazioni di calcolo, analisi statistiche e magie varie. Tuttavia l’interfaccia di base di R, definita **Console** (vedi Figura 2.1), è per così dire *démodé* o meglio, solo per veri intenditori.

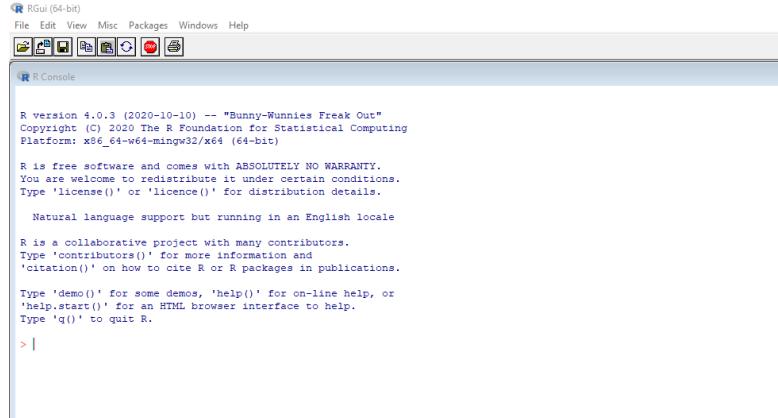


Figure 2.1: La console di R, solo per veri intenditori

In genere, per lavorare con R viene utilizzato RStudio. RStudio è un programma (IDE - Integrated Development Environment) che integra in un unica interfaccia utente (GUI - Graphical User Interface) diversi strumenti utili per la scrittura ed esecuzione di codici. L’interfaccia di RStudio è costituita da 4 pannelli principali

(vedi Figura 2.2):

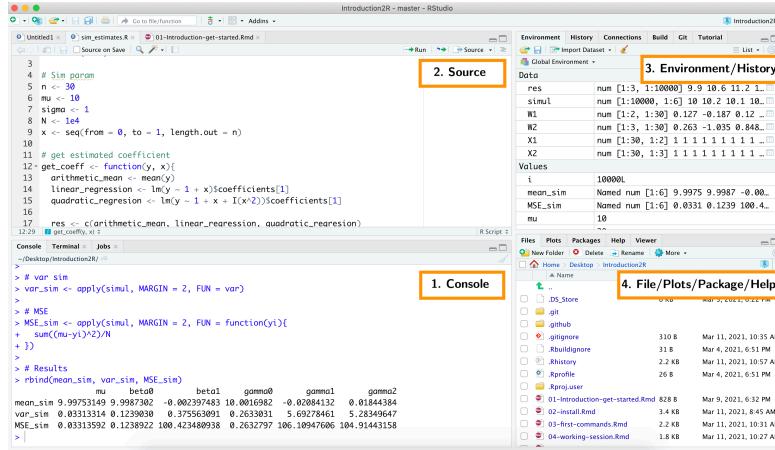


Figure 2.2: Interfaccia utente di Rstudio con i suoi 4 pannelli

### 1. Console: il cuore di R

Qui ritroviamo la *Console* di R dove vengono effettivamente eseguiti tutti i tuoi codici e comandi. Nota come nell'ultima riga della *Console* appaia il carattere **>**. Questo è definito *prompt* e ci indica che R è in attesa di nuovi comandi da eseguire.

La *Console* di R è un'interfaccia a linea di comando. A differenza di altri programmi “*punta e clicca*”, in R è necessario digitare i comandi utilizzando la tastiera. Per eseguire dei comandi possiamo direttamente scrivere nella *Console* le operazioni da eseguire e premere **invio**. R eseguirà immediatamente i nostro comando, riporterà il risultato e nella linea successiva apparirà nuovamente il *prompt* indicando che R è pronto ad eseguire un altro comando (vedi Figura 2.3).

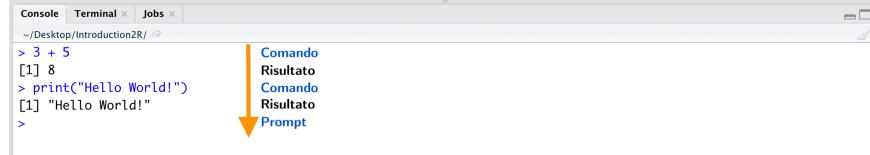
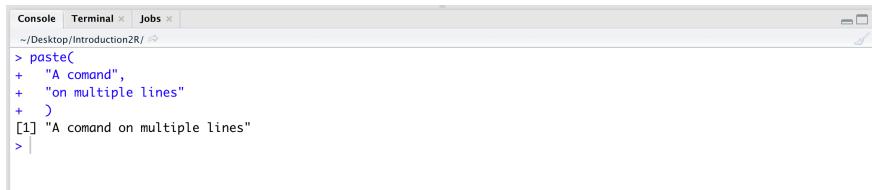


Figure 2.3: Esecuzione di comandi direttamente nella console

Nel caso di comandi scritti su più righe, vedi l'esempio di Figura 2.4, è possibile notare come venga mostrato il simbolo `+` come *prompt*. Questo indica che R è in attesa che l'intero comando venga digitato prima che esso venga eseguito.



The screenshot shows a terminal window with three tabs: 'Console', 'Terminal', and 'Jobs'. The 'Console' tab is active. The window title is 'Console' and the path is '/Desktop/Introduction2R/'. The text area contains the following command:

```
> paste<
+ "A command",
+ "on multiple lines"
+ )
[1] "A command on multiple lines"
>
```

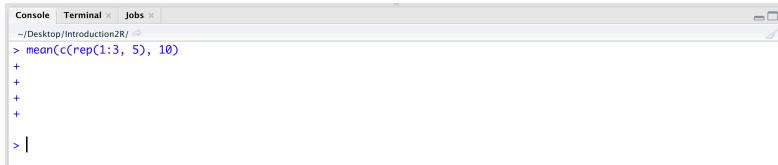
Figure 2.4: Esecuzione di un comando su più righe

Come avrai notato facendo alcune prove, i comandi digitati nella *Console* vengono eseguiti immediatamente ma non sono salvati. Per rieseguire un comando, possiamo navigare tra quelli precedentemente eseguiti usando le frecce della tastiera  $\uparrow\downarrow$ . Tuttavia, in caso di errori dovremmo riscrivere e rieseguire tutti i comandi. Siccome scrivere codici è un continuo “*try and error*”, lavorare unicamente dalla *Console* diventa presto caotico. Abbiamo bisogno quindi di una soluzione che ci permetta di lavorare più comodamente sui nostri codici e di poter salvare i nostri comandi da eseguire all’occorrenza con il giusto ordine. La soluzione sono gli *Scripts* che introdurremo vedremo nella prossima sezione.

 Tip-Box: Interrompere un comando

Potrebbe accadere che per qualche errore nel digitare un comando o perchè sono richiesti lunghi tempi computazionali, la *Console* di R diventi non responsiva. In questo caso è necessario interrompere la scrittura o l'esecuzione di un comando. Vediamo due situazioni comuni:

1. **Continua a comparire il prompt +.** Specialmente nel caso di utilizzo di parentesi e lunghi comandi, accade che una volta premuto **invio** R non esegua alcun comando ma resta in attesa mostrando il *prompt +* (vedi Figura seguente). Questo è in genere dato da un errore nella sintassi del comando (e.g., un errore nell'uso delle parentesi o delle virgole). Per riprendere la sessione è necessario premere il tasto **esc** della tastiera. L'appriore del *prompt >*, indica che R è nuovamente in ascolto pronto per eseguire un nuovo comando ma attento a non ripetere lo stesso errore, la sintassi dei comandi è importante (vedi Capitolo TODO).



```
Console Terminal Jobs
~/Desktop/Introduction2R >
> mean(c(rep(1:3, 5), 10)
+
+
+
+
> |
```

2. **R non risponde.** Alcuni calcoli potrebbero richiedere molto tempo o semplicemente un qualche problema ha mandato in loop la tua sessione di lavoro. In questa situazione la *Console* di R diventa non responsiva. Nel caso fosse necessario interrompere i processi attualmente in esecuzione devi premere il pulsante *STOP* come indicato nella Figura seguente. R si fermerà e ritornerà in attesa di nuovi comandi (*prompt >*).



```
Console Terminal Jobs
~/Desktop/Introduction2R >
> res <- replicate(1e6, {
+   y <- rnorm(30, 10, 1)
+   mean(y)
+ })
```

### Trick-Box: Force Quit

In alcuni casi estremi in cui R sembra non rispondere, usa i comandi **Ctrl-C** per forzare R a interrompere il processo in esecuzione.

Come ultima soluzione ricorda uno dei principi base dell'informatica “*spegni e riaccendi*” (a volte potrebbe bastare chiudere e riaprire RStudio).

## 2. Source: il tuo blocco appunti

In questa parte vengono mostrati i tuoi *Scripts*. Questi non sono altro che degli speciali documenti (con estensione “**.R**”) in cui sono salvati i tuoi codici e comandi che potrai eseguire quando necessario in R. Gli *Scripts* ti permetteranno di lavorare comodamente sui tuoi codici, scrivere i comandi, correggerli, organizzarli, aggiungere dei commenti e soprattutto salvarli.

Dopo aver terminato di scrivere i comandi, posiziona il cursore sulla stessa linea del comando che desideri eseguire e premi **command + invio** (MacOs) o **Ctrl+R** (Windows). Automaticamente il comando verrà copiato nella *Console* ed eseguito. In alternativa potrai premere il tasto **Run** indicato dalla freccia in Figura 2.5.

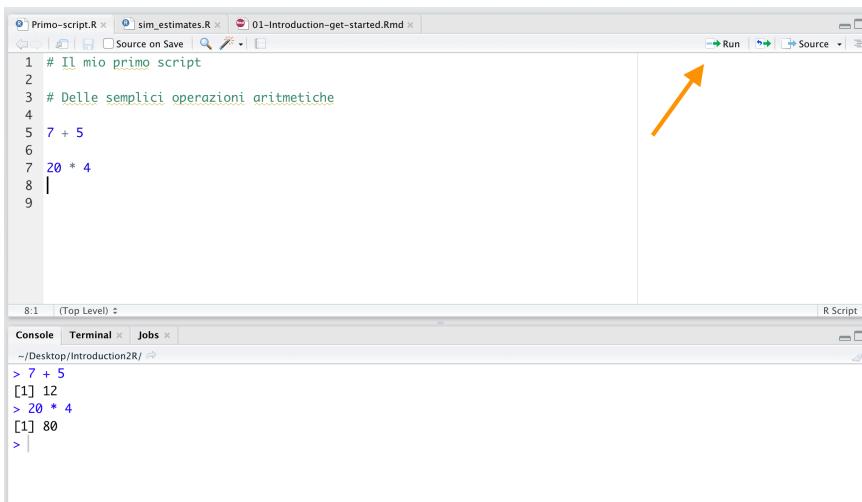


Figure 2.5: Esecuzione di un comando da script premi ‘command + invio’ (MacOs)/ ‘Ctrl+R’ (Windows) o premi il tasto indicato dalla freccia

### Tip-Box: Commenti

Se hai guardato con attenzione lo script rappresentato in Figura 2.5, potresti aver notato delle righe di testo verde precedute dal simbolo `#`. Questo simbolo può essere utilizzato per inserire dei *commenti* all'interno dello script. R ignorerà qualsiasi commento ed eseguirà soltanto le parti di codici. L'utilizzo dei commenti è molto importante nel caso di script complessi poiché ci permette di spiegare e documentare il codice che viene eseguito. Nel Capitolo TODO approfondiremo il loro utilizzo.

### 3. Environment e History: la sessione di lavoro

Qui sono presentati una serie di pannelli utili per valutare informazioni inerenti alla propria sessione di lavoro. I pannelli principali sono *Environment* e *History* (gli altri pannelli presenti in Figura 2.6 riguardano funzioni avanzate di RStudio).

- **Environment:** elenco tutti gli oggetti e variabili attualmente presenti nell'ambiente di lavoro. Approfondiremo i concetti di variabili e di ambiente di lavoro rispettivamente nel Capitolo TODO e Capitolo TODO.

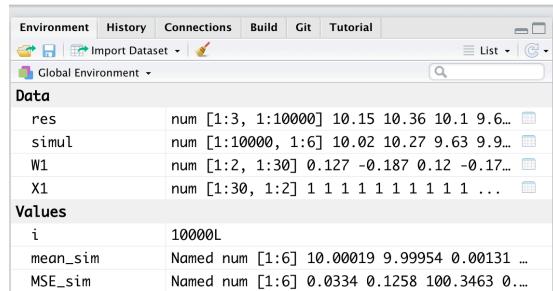


Figure 2.6: \*Environment\* - Elenco degli oggetti e variabili presenti nell'ambiente di lavoro

- **History:** elenco di tutti i comandi precedentemente eseguiti nella console. Nota che questo non equivale ad uno script, anzi, è semplicemente un elenco non modificabile (e quasi mai usato).

#### 4. File, Plots, Package, Help: system management

In questa parte sono raccolti una serie di pannelli utilizzati per interfacciarsi con ulteriori risorse del sistema (e.g., file e pacchetti) o produrre output quali grafici e tabelle.

- **Files:** pannello da cui è possibile navigare tra tutti i file del proprio computer

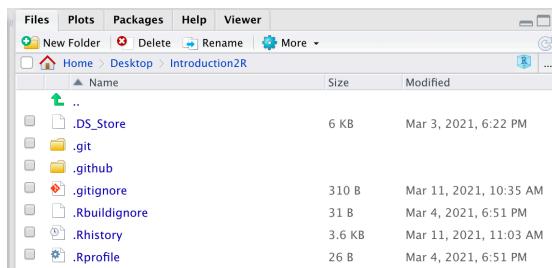


Figure 2.7: \*Files\* - permette di navigare tra i file del proprio computer

- **Plots:** pannello i cui vengono prodotti i grafici e che è possibile esportare cliccando *Export*.

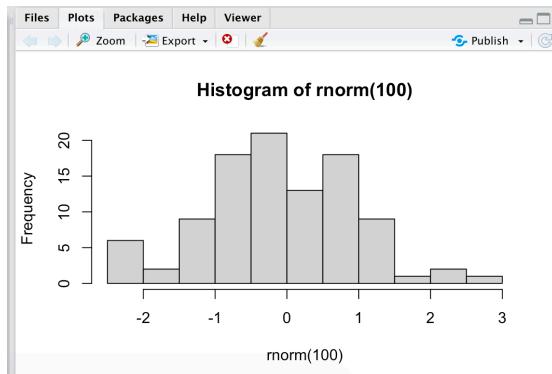


Figure 2.8: \*Plots\* - presentazione dei grafici

- **Packages:** elenco dei pacchetti di R (questo argomento verrà approfondito nel Capitolo TODO).
- **Help:** utilizzato per navigare la documentazione interna di R (questo argomento verrà approfondito nel Capitolo TODO).

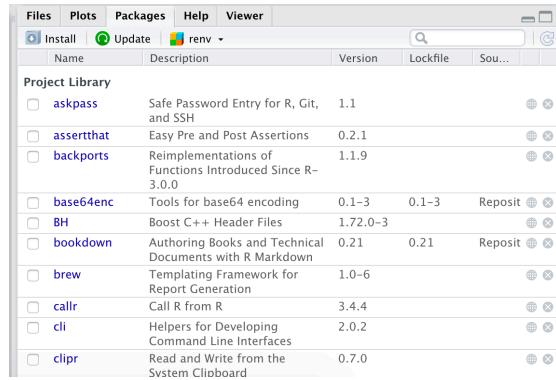


Figure 2.9: \*Packages\* - elenco dei pacchetti di R

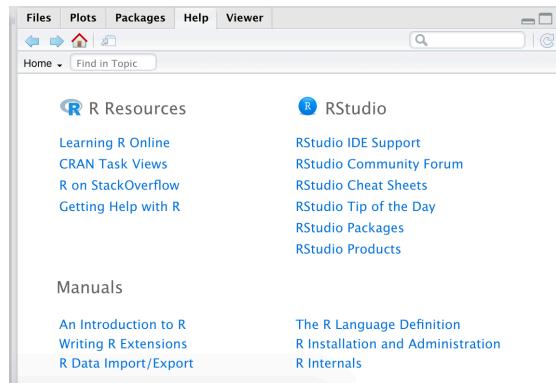


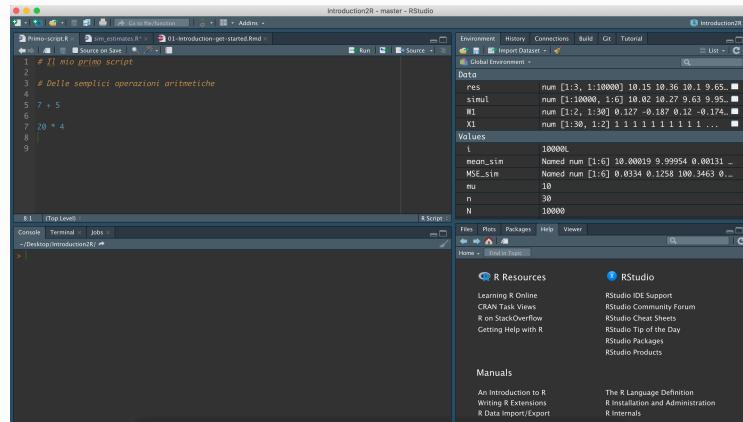
Figure 2.10: \*Help\* - documentazione di R



### Tip-Box: Personalizza tema e layout

RStudio permette un ampio grado di personalizzazione dell'intrafaccia grafica utilizzata. E' possibile cambiare tema, font e disposizione dei pannelli a seconda dei tuoi gusti ed esigenze.

Prova a cambiare il tema dell'editor in *Idle Fingers* per utilizzare un background scuro che affatichi meno la vista (vedi Figura seguente). Clicca su RStudio > Preferenze > Appearance (MacOS) o Tools > Options > Appearance (Windows).





# Chapter 3

## Primi Passi in R

Ora che abbiamo iniziato a familiarizzare con il nostro stumento di lavoro possiamo finalmente dare fuoco alle polveri e concentraci sulla scrittura di codici!

In questo capitolo muoveremo i primi passi in R. Inizieremo vedendo come utilizzare operatori matematici, relazionali e logici per compiere semplici operazioni in R. Imparare R è un lungo percorso (scoop: questo percorso non termina mai dato che R è sempre in continuo sviluppo). Soprattutto all'inizio può sembrare eccessivamente difficile poichè è si incontrano per la prima volta molti comandi e concetti di programmazione. Tuttavia, una volta familiarizzato con gli apetti di base, la progressione diventa sempre più veloce (inarrestabile direi!).

In questo capitolo introdurremo per la prima volta molti elementi che saranno poi ripresi e approfonditi nei seguenti capitoli. Quindi non preoccuparti se non tutto ti sarà chiaro fin da subito. Imparare il tuo primo linguaggio di programmazione è difficile ma da qualche parte bisogna pure iniziare. Pronto per le tue prime linee di codice? Let's become a useR!

### 3.1 Operatori Matematici

R è un'ottima calcolatrice. Nella Tabella 3.1 sono elencati i principali operatori matematici e funzioni usate in R.

Table 3.1: Operatori Matematici

Funzione	Nome	Esempio
<code>x + y</code>	Addizione	<code>&gt; 5 + 3 [1] 8</code>
<code>x - y</code>	Sottrazione	<code>&gt; 7 - 2 [1] 5</code>
<code>x * y</code>	Moltiplicazione	<code>&gt; 4 * 3 [1] 12</code>
<code>x / y</code>	Divisione	<code>&gt; 8 / 3 [1] 2.666667</code>
<code>x %% y</code>	Resto della divisione	<code>&gt; 7 %% 5 [1] 2</code>
<code>x %/% y</code>	Divisione intera	<code>&gt; 7 %/% 5 [1] 1</code>
<code>x ^ y</code>	Potenza	<code>&gt; 3^3 [1] 27</code>
<code>abs(x)</code>	Valore assoluto	<code>&gt; abs(3-5^2) [1] 22</code>
<code>sign(x)</code>	Segno di un'espressione	<code>&gt; sign(-8) [1] -1</code>
<code>sqrt(x)</code>	Radice quadrata	<code>&gt; sqrt(25) [1] 5</code>
<code>log(x)</code>	Logaritmo naturale	<code>&gt; log(10) [1] 2.302585</code>
<code>exp(x)</code>	Esponenziale	<code>&gt; exp(1) [1] 2.718282</code>
<code>sin(x)</code> <code>cos(x)</code> <code>tan(x)</code> <code>asin(x)</code> <code>acos(x)</code> <code>atan(x)</code>	Funzioni trigonometriche	<code>&gt; sin(pi/2) [1] 1 &gt; cos(pi/2) [1] 6.123234e-17</code>
<code>factorial(x)</code>	Fattoriale	<code>&gt; factorial(6) [1] 720</code>
<code>choose(n, k)</code>	Coefficiente binomiale	<code>&gt; choose(5,3) [1] 10</code>



### Tip-Box: Le prime funzioni

Nota come per svolgere operazioni come la radice quadrata o il valore assoluto vengono utilizzate delle specifiche funzioni. In R le funzioni sono richiamate digitando `<nome-funzione>()` (e.g., `sqrt(25)`) indicando all'interno delle parentesi tonde gli argomenti della funzione. Approfondiremo le funzioni nella Sezione TODO.

### 3.1.1 Ordine Operazioni

Nello svolgere le operazioni, R segue lo stesso ordine usato nelle normali espressioni matematiche. Quindi l'ordine di precedenza degli operatori è:

1. `^` (potenza)
2. `%%` (resto della divisione) e `%/%` (divisione intera)
3. `*` (moltiplicazione) e `/` (divisione)
4. `+` (addizione) e `-` (sottrazione)

Nota che in presenza di funzioni (e.g., `abs()`, `sin()`), R per prima cosa sostituisca le funzioni con il loro risultato per poi procedere con l'esecuzione delle operazioni nell'ordine indicato precedentemente.

L'ordine di esecuzione delle operazioni può essere controllato attraverso l'uso delle **parentesi tondone** `()`. R eseguirà tutte le operazioni incluse nelle parentesi seguendo lo stesso ordine indicato sopra. Utilizzando più gruppi di parentesi possiamo ottenere i risultati desiderati.



### Warning-Box: Le parentesi

Nota che in R solo le **parentesi tonde** `()` sono utilizzate per gestire l'ordine con cui sono eseguite le operazioni.

**Parentesi quadre** `[]` e **parentesi graffe** `{}` sono invece speciali operatori utilizzati in R per altre ragioni come la selezione di elementi e la definizione di blocchi di codici. Argomenti che approfondiremo rispettivamente nel Capitolo TODO e Capitolo TODO.

## Esercizi

Calcola il risultato delle seguenti operazioni utilizzando R:

$$1. \frac{(45+21)^3 + \frac{3}{4}}{\sqrt{32 - \frac{12}{17}}}$$

$$2. \frac{\sqrt{7-\pi}}{3(45-34)}$$

$$3. \sqrt[3]{12 - e^2} + \ln(10\pi)$$

$$4. \frac{\sin(\frac{3}{4}\pi)^2 + \cos(\frac{3}{2}\pi)}{\log_7 e^{\frac{3}{2}}}$$

$$5. \frac{\sum_{n=1}^{10} n}{10}$$

Note per la risoluzione degli esercizi:

- In R la radice quadrata si ottiene con la funzione `sqrt()` mentre per radici di indici diversi si utilizza la notazione esponenziale ( $\sqrt[3]{x}$  è dato da `x^(1/3)`).
- Il valore di  $\pi$  si ottiene con `pi`.
- Il valore di  $e$  si ottiene con `exp(1)`.
- In R per i logaritmi si usa la funzione `log(x, base=a)`, di base viene considerato il logaritmo naturale.

## 3.2 Operatori Relazionali e Logici

### 3.2.1 Operatori Relazionali

In R è possibile valutare se una data relazione è vera o falsa. Ad esempio, posiamo valutare se “2 è minore di 10” o se “4 numero è un numero pari”. Queste operazioni al momento potrebbero sembrare non particolarmente interessanti ma si riveleranno molto utili nei capitoli successivi ad esempio per la selezione di elementi (vedi Capitolo TODO) o la definizione di algoritmi (vedi Capitolo TODO).

R valuterà le proposizioni e ci restituirà il valore `TRUE` se la proposizione è vera oppure `FALSE` se la proposizione è falsa. Nella Tabella 3.2 sono elencati gli operatori relazionali.

Table 3.2: Operatori Relazionali

Funzione	Nome	Esempio
<code>x == y</code>	Uguale	<code>&gt; 5 == 3 [1] FALSE</code>
<code>x != y</code>	Diverso	<code>&gt; 7 != 2 [1] TRUE</code>
<code>x &gt; y</code>	Maggiore	<code>&gt; 4 &gt; 3 [1] TRUE</code>
<code>x &gt;= y</code>	Maggiore o uguale	<code>&gt; -2 &gt;= 3 [1] FALSE</code>
<code>x &lt; y</code>	Minore	<code>&gt; 7 &lt; 5 [1] FALSE</code>
<code>x &lt;= y</code>	Minore o uguale	<code>&gt; 7 &lt;= 7 [1] TRUE</code>
<code>x %in% y</code>	inclusione	<code>&gt; 5 %in% c(3, 5, 8) [1] TRUE</code>



Warning-Box: '`==`' non è uguale a '`=`'

Attenzione che per valutare l'uguaglianza tra due valori non bisogna utilizzare `=` ma `==`. Questo è un'errore molto comune che si commette in continuazione.

L'operatore `=` è utilizzato in R per assegnare un valore ad una variabile. Argomento che vederemo nella Sezione TODO

Table 3.3: Operatori Logici

Funzione	Nome	Esempio
<code>!x</code>	Negazione	<code>&gt; !TRUE [1] FALSE</code>
<code>x &amp; y</code>	Congiunzione	<code>&gt; TRUE &amp; FALSE [1] FALSE</code>
<code>x   y</code>	Disgiunzione Inclusiva	<code>&gt; TRUE   FALSE [1] TRUE</code>



## Tip-Box: TRUE-T-1; FALSE-F-0

Nota che in qualsiasi linguaggio di Programmazione, ai valori TRUE e FALSE sono associati rispettivamente i valori numerici 1 e 0. Questi sono definiti valori booleani.

```
TRUE == 1    # TRUE
TRUE == 2    # FALSE
TRUE == 0    # FALSE
FALSE == 0   # TRUE
FALSE == 1   # FALSE
```

In R è possibile anche abbreviare TRUE e FALSE rispettivamente in T e F, sebbene sia una pratica non consigliata poichè potrebbe non essere chiara e creare fraintendimenti.

```
T == 1      # TRUE
T == TRUE   # TRUE
F == 0      # TRUE
F == FALSE  # TRUE
```

In R è possibile congiungere più relazioni per valutare una desiderata proposizione. Ad esempio potremmo valutare se “17 è maggiore di 10 e minore di 20”. Per unire più relazioni in un’unica proposizione che R valuterà come TRUE o FALSE, vengono utilizzati gli operatori logici riportati in Tabella 3.3.

Questi operatori sono anche definiti operatori booleani e seguono le comuni definizioni degli operatori logici. In particolare abbiamo che:

- Nel caso della **congiunzione logica** `&`, affinchè la proposizione sia vera è necessario che entrambe le relazioni siano vere. Negli altri casi la proposizione sarà valutata falsa.
- Nel caso della **disgiunzione inclusiva logica** `|`, affinchè la proposizione sia vera è necessario che almeno una relazione sia vera. La proposizione sarà valutata falsa solo quando entrambe le relazioni sono false.

### 3.2.2 Ordine valutazione relazioni

Nel valutare le veridicità delle proposizioni R esegue le operazioni nel seguente ordine:

1. operatori matematici (e.g., `^`, `*`, `/`, `+`, `-`, etc.)
2. operatori relazionali (e.g., `<`, `>`, `<=`, `>=`, `==`, `!=`)
3. operatori logici (e.g., `!`, `&`, `|`)

La lista completa dell'ordine di esecuzione delle operazioni è riportata al seguente link <https://stat.ethz.ch/R-manual/R-devel/library/base/html/Syntax.html>. Ricordiamo che, in caso di dubbi riguardanti l'ordine di esecuzione delle operazioni, la cosa migliore è utilizzare le parentesi tonde () per disambiguare ogni possibile fraintendimento.



#### Warning-Box: L'operatore `%in%`

Nota che l'operatore `%in%` che abbiamo precedentemente indicato tra gli operatori relazionali in realtà è un operatore speciale. In particolare, non segue le stesse regole degli altri operatori relazionali per quanto riguarda l'ordine di esecuzione.

La soluzione migliore? Usa le parentesi!

## Esercizi

1. Definisce due relazioni false e due vere che ti permettano di valutare i risultati di tutti i possibili incroci che puoi ottenere con gli operatori logici `&` e `|`.
2. Definisci una proposizione che ti permetta di valutare se un numero è pari. Definisci un'altra proposizione per i numeri dispari (tip: cosa ti ricorda `%%?`).

3. Definisci una proposizione per valutare la seguente condizione (ricordati di testare tutti i possibili scenari) “*x è un numero compreso tra -4 e -2 oppure è un numero compreso tra 2 e 4*”.
4. Esegui le seguenti operazioni `4 ^ 3 %in% c(2,3,4)` e `4 * 3 %in% c(2,3,4)`. Cosa osservi nell’ordine di esecuzione degli operatori?

## Chapter 4

# Due Compagni Inseparabili

In questo capitolo introdurremmo i concetti di oggetti e funzioni, due elementi alla base di R (e di ogni linguaggio di programmazione). Potremmo pensare agli oggetti in R come a delle variabili che ci permettono di mantenere in memoria dei valori (e.g., i risultati dei nostri calcoli o i nostri dati). Le funzioni in R, invece, sono analoghe a delle funzioni matematiche che, ricevuti degli oggetti in input, compiono delle azioni e restituiscono dei nuovi oggetti in output.

Questa è una iper-semplificazione (e pure tecnicamente non corretta) che ci permette però di capire come, partendo dai nostri dati o valori iniziali, possiamo manipolarli applicando delle funzioni per ottenere attraverso differenti step i risultati desiderati (e.g., analisi statistiche o grafici e tabelle).

Qui valuteremo gli aspetti fondamentali che riguardano l'utilizzo degli oggetti e delle funzioni mentre saranno approfonditi rispettivamente in tutta la seconda e terza sezione del libro (TODO).

### 4.1 Oggetti

Quando eseguiamo un commando in R, il risultato ottenuto viene immediatamente mostrato in *Console*. Tale risultato, tuttavia, non viene salvato in memoria e quindi non potrà essere riutilizzato in nessuna operazione futura. Condurre delle analisi in questo modo sarebbe estremamente complicato ed inefficiente. La soluzione più ovvia è quella di salvare in memoria i nostri risultati intermedi per poterli poi riutilizzare nel corso delle nostre analisi. Si definisce questo processo come *assegnare* un valore ad un oggetto.

### 4.1.1 Assegnare e Richiamare un oggetto

Per assegnare il valore numerico 5 all'oggetto `x` è necessario eseguire il seguente comando:

```
x <- 5
```

La funzione `<-` ci permette di assegnare i valori che si trovano alla sua destra all'oggetto il cui nome è definito alla sinistra. Abbiamo pertanto il seguente pattern: `<nome-oggetto> <- <valore-assegnato>`. Notate come in *Console* appaia solo il comando appena eseguito ma non venga mostrato alcun risultato.

Per utilizzare il valore contenuto nell'oggetto sarà ora sufficiente richiamare nel proprio codice il nome dell'oggetto desiderato.

```
x + 3
## [1] 8
```

E' inoltre possibile "aggiornare" o "sostituire" il valore contenuto in un oggetto. Ad esempio:

```
x <- x*10
x
## [1] 50

x <- "Hello World!"
x
## [1] "Hello World!"
```

Nel primo caso abbiamo prima utilizzato il vecchio valore contenuto in `x` per calcolare il nuovo risultato che è stato assegnato a `x`. Nel secondo caso abbiamo sostituito il vecchio valore di `x` con un nuovo valore (nell'esempio una stringa di caratteri).



### Approfondimento: Assegnare valori '<->' vs '='

Esistono due operatori principali che sono usati per assegnare un valore ad un oggetto: l'operatore `<-` e l'operatore `=`. Entrambi sono validi e spesso la scelta tra i due diventa solo una questione di stile personale.

```
x_1 <- 45
x_2 = 45

x_1 == x_2
## [1] TRUE
```

Esistono, tuttavia, alcune buone ragioni per preferire l'uso di `<-` rispetto a `=` (attewnti a non confonderlo con l'operatore relazionale `==`). L'operazione di assegnazione è un'operazione che implica una direzionalità, il chè è reso esplicito dal simbolo `<-` mentre il simbolo `=` non evidenzia questo aspetto e anzi richiama la relazione di uguaglianza in matematica.

La decisione su quale operatore adottare è comunque libera, ma ricorda che una buona norma nella programmazione riguarda la *consistenza*: una volta presa una decisione è bene mantenerla per facilitare la comprensione del codice.

#### 4.1.2 Nomi degli oggetti

La scelta dei nomi degli oggetti sembra un aspetto secondario ma invece ha una grande importanza per facilitare la chiarezza e comprensione dei codici.

Ci sono alcune regole che discriminano i nomi validi da nomi non validi. Il nome di un oggetto

- deve iniziare con una lettera e può contenere lettere, numeri, underscore (`_`), o punti (`.`).
- potrebbe anche iniziare con un punto (`.`) ma in tal caso non può essere seguito da un numero.
- non deve contenere caratteri speciali come `#`, `&`, `$`, `?`, etc.

- non deve essere una parola riservata ovvero quelle parole che sono utilizzate internamente da R con un significato speciale (e.g, TRUE, FALSE etc.).

Nota come R sia **Case-Sensitive**, ovvero distingua tra lettere minuscole e maiuscole. Nel seguente esempio i due nomi sono considerate diversi e pertanto non avviene una sovrascrizione ma due differenti oggetti sono creati:

```
My_name <- "Monty"
my_name <- "Python"

My_name
## [1] "Monty"
my_name
## [1] "Python"
```

Inoltre, il nome ideale di un oggetto dovrebbe essere:

- **auto-descrittivo**: dal solo nome dovrebbe essere possibile intuire il contenuto dell'oggetto. Un nome generico quale x o y ci sarebbero di poco aiuto poiché potrebbero contenere qualsiasi informazione. Invece un nome come weight o gender ci suggerirebbe chiaramente il contenuto dell'oggetto (e.g., il peso o gender dei partecipanti del nostro studio).
- **dell giusta lunghezza**: non deve essere ne troppo breve (evitare sigle incomprensibili) e neppure troppo lunghi. La lunghezza corretta è quella che permette al nome di essere sufficientemente informativo senza aggiungere inutili dettagli. In genere sono sufficienti 2 o 3 parole.



### Approfondimento: CamelCase vs snake\_case

Spesso più parole sono usate per ottenere un nome sufficientemente chiaro. Dato che però non è possibile includere spazi in un nome, nasce il problema di come unire più parole senza che il nome diventi incomprensibile, ad esempio `mediatestcontrollo`.

Esistono diverse convenzioni tra cui:

- **CamelCase.** L'inizio di una nuova parola viene indicata con l'uso della prima lettera maiuscola. Ad esempio `mediaTestControllo`.
- **snake\_case.** L'inizio di una nuova parola viene indicata con l'uso carattere `_`. Ad esempio `media_test_controllo`.
- una variante al classico `snake_case` riguarda l'uso del `.`, ad esempio `media.test.controllo`. Questo approccio in genere è evitato poiché in molti linguaggi di programmazione (ed anche in R in alcune condizioni) il carattere `.` è un carattere speciale.

In genere viene raccomandato di seguire la convenzione `snake_case`. Tuttavia, la decisione su quale convenzione adottare è libera, ma ricorda ancora che una buona norma nella programmazione riguarda la *consistenza*: una volta presa una decisione è bene mantenerla per facilitare la comprensione del codice.

#### 4.1.3 Tipologie di Oggetti

In R abbiamo differenti tipologie di oggetti (o meglio di strutture dati):

- Vettori
- Matrici
- Dataframe
- Liste

La loro definizione, le loro caratteristiche ed il loro utilizzo sarà discusso nella seconda sezione di questo libro TODO

## 4.2 Funzioni

- arguments input
- output
- nomi riservati ?reserved
- trick tab
- help function

### 4.2.1

# Chapter 5

## Sessione di Lavoro

Working in progress.

### 5.1 Infobox

Illustrations included in `images/` are retrieved from rstudio4edu-book under CC-BY-NC. Remember to include an *Attributions* section in the book and repository's README file.



#### Tip-Box: My title

Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudian-dae consequuntur voluptatum laborum numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium optio, eaque rerum!

 Warning-Box: My title

Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium optio, eaque rerum!

 Definition-Box: My title

Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium optio, eaque rerum!

 Approfondimento: My title

Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium optio, eaque rerum!

 Trick-Box: My title

Lorem ipsum dolor sit amet consectetur adipisicing elit. Maxime mollitia, molestiae quas vel sint commodi repudiandae consequuntur voluptatum laborum numquam blanditiis harum quisquam eius sed odit fugiat iusto fuga praesentium optio, eaque rerum!

## 5.2 Problema + Google = Soluzione

Quando si approccia la scrittura di codice, anche molto semplice la cosa che sicuramente capiterà più spesso sarà riscontrare **errori** e quindi trovare il modo per risolverli.

Qualche programmatore esperto direbbe che l'essenza stessa di programmare è in realtà risolvere gli errori che il codice produce.

L'**errore** non è quindi un difetto o un imprevisto, ma parte integrante della scrittura del codice. L'importante è capire come gestirlo.

Abbiamo tutti le immagini in testa di programmatori da film che scrivono codice alla velocità della luce, quando nella realtà dobbiamo spesso affrontare **bug**, **errori di output** o altri problemi vari. Una serie di skills utili da imparare sono:

- Comprendere a fondo gli **errori** (non banale)
- Sapere **come e dove cercare una soluzione** (ancora meno banale)
- In caso non si trovi una soluzione direttamente, chiedere aiuto in modo efficace

### Comprendere gli errori

Rispetto agli errori, R è solitamente abbastanza esplicito nel farci capire il problema. Ad esempio usare una funzione di un pacchetto che non è stato caricato di solito fornisce un messaggio del tipo **Error in funzione : could not find function "funzione"**.

### Ricercare soluzioni

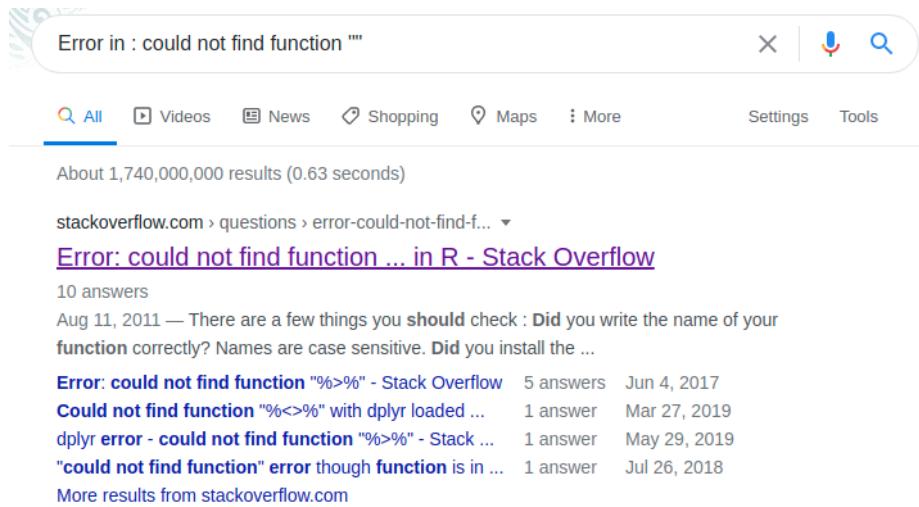
Altre situazioni o messaggi potrebbero non essere altrettanto immediati, in quel caso Google è il nostro miglior amico.

Cercando infatti il messaggio di errore/warning su Google, al 99% avremo altre persone che hanno avuto lo stesso problema e probabilmente anche una soluzione.

### Trick-Box: Ricerca su Google

Il modo migliore per cercare è copiare e incollare su Google direttamente l'output di errore di R come ad esempio **Error in funzione : could not find function "funzione"** piuttosto che descrivere a parole il problema. I messaggi di errore sono standard per tutti, la tua descrizione invece no.

Cercando in questo modo vedrete che molti dei risultati saranno esattamente riferiti al vostro errore:



The screenshot shows a Google search results page. The search bar contains the query "Error in : could not find function". Below the search bar, there are tabs for All, Videos, News, Shopping, Maps, More, Settings, and Tools. The "All" tab is selected. It displays approximately 1,740,000,000 results found in 0.63 seconds. The top result is a link to "Error: could not find function ... in R - Stack Overflow". Below the link, it says "10 answers". A snippet of the answer from Aug 11, 2011, reads: "Aug 11, 2011 — There are a few things you **should** check : Did you write the name of your function correctly? Names are case sensitive. Did you install the ...". Below this, there are several other search results for related errors like "%>%", "%<%", and "dplyr error". At the bottom of the results, there is a link to "More results from stackoverflow.com".

#### 5.2.0.1 Chiedere una soluzione

Se invece il vostro problema non è un messaggio di errore ma un utilizzo specifico di R allora il consiglio è di usare una ricerca del tipo: **argomento + breve descrizione problema + R**. Nelle sezioni successive vedrete nel dettaglio altri aspetti della programmazione ma se volete ad esempio calcolare la **media** in R potrete scrivere **compute mean in R**. Mi raccomando, fate tutte le ricerche in **inglese** perchè le possibilità di trovare una soluzione sono molto più alte.

Dopo qualche ricerca, vi renderete conto che il sito che vedrete più spesso si chiama **Stack Overflow**. Questo è una manna dal cielo per tutti i programmati, a qualsiasi livello di expertise. E' una community dove tramite domande e risposte, si impara a risolvere i vari problemi ed anche a trovare nuovi modi di fare la stessa cosa. E' veramente utile oltre che un ottimo modo per imparare.

L'ultimo punto di questa piccola guida alla ricerca di soluzioni, riguarda il fatto di dover non solo cercare ma anche chiedere. Dopo aver cercato vari post di persone che richiedevano aiuto per un problema noterete che le domande e le risposte hanno sempre una struttura simile. Questo non è solo un fatto stilistico ma anzi è molto utile per uniformare e rendere chiara la domanda ma soprattutto la risposta, in uno spirito di condivisione. C'è anche una guida dedicata per scrivere la domanda perfetta.

In generale<sup>1</sup>:

- Titolo: un super riassunto del problema
- Contesto: linguaggio (es. R), quale sistema operativo (es. Windows)
- Descrizione del problema/richiesta: in modo chiaro e semplice ma non troppo generico
- Codice ed eventuali dati per capire il problema

L'ultimo punto di questa lista è forse il più importante e si chiama in gergo tecnico **REPREX** (**R**eproducible **E**xample). E' un tema leggermente più avanzato ma l'idea di fondo è quella di fornire tutte le informazioni possibili per poter riprodurre (e quindi eventualmente trovare una soluzione) il codice di qualcuno nel proprio computer.

Se vi dico "R non mi fa creare un nuovo oggetto, quale è l'errore?" è diverso da dire "il comando oggetto -> 10 mi da questo errore **Error in 10 <- oggetto : invalid (do\_set) left-hand side to assignment**, come posso risolvere?"



#### Tip-Box: My title

Ci sono anche diversi pacchetti in R che rendono automatico creare questi esempi di codice da poter condividere, come il pacchetto **reprex**.

---

<sup>1</sup>Fonte: Writing the perfect question - Jon Skeet



# **Struttura Dati**



# Introduzione

Working in progress.



# **Chapter 6**

## **Vettori**

Working in progress.



# **Chapter 7**

## **Matrici**

Working in progress.



# **Chapter 8**

# **Dataframe**

Working in progress.



# **Chapter 9**

## **Liste**

Working in progress.



# **Algoritmi**



# Introduzione

Working in progress.



## Chapter 10

# Definizione di Funzioni

Working in progress.



## Chapter 11

# Programmazione Condizionale

Working in progress.



## **Chapter 12**

# **Attenti al loop**

Working in progress.



# Case study



# Introduzione

Working in progress.



## **Chapter 13**

# **Caso Studio I: Attaccamento**

Working in progress.