

Introduzione a R

Corso per imparare le basi di **R**

Psicostat

18-03-2021



Contents

Presentazione	7
Perchè R	7
Struttura del libro	7
Risorse Utili	8
Psicostat	9
Collaborazione	9
Riconoscimenti	9
Licenza	9
Get Started	13
Introduzione	13
1 Installare R e RStudio	15
1.1 Installare R	15
1.2 Installare R Studio	20
2 Interfaccia RStudio	23
3 Primi Passi in R	33
3.1 Operatori Matematici	33
3.2 Operatori Relazionali e Logici	36

4 Due Compagni Inseparabili	41
4.1 Oggetti	41
4.2 Funzioni	46
5 Sessione di Lavoro	53
5.1 Environment	53
5.2 Working Directory	55
5.3 R-packages	55
5.4 Errors and warnings	55
5.5 Sessione di lavoro	55
5.6 Problema + Google = Soluzione	55
Struttura Dati	61
Introduzione	61
6 Data Type	63
7 Vettori	65
7.1 Creazione di Vettori	65
7.2 Selezione Elementi di un Vettore	65
7.3 Funzioni ed Operazioni tra Vettori	66
8 Fattori	67
9 Matrici	69
9.1 Creazione di Matrici	69
9.2 Selezione di Elementi di una Matrice	70
9.3 Funzioni ed Operazioni tra Matrici	70
10 Dataframe	73
11 Liste	75

CONTENTS	5
Algoritmi	79
Introduzione	79
12 Definizione di Funzioni	81
13 Programmazione Condizionale	83
14 Attenti al loop	85
Case study	89
Introduzione	89
15 Caso Studio I: Attaccamento	91
15.1 Infobox	91

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 community
- 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, 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.

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.

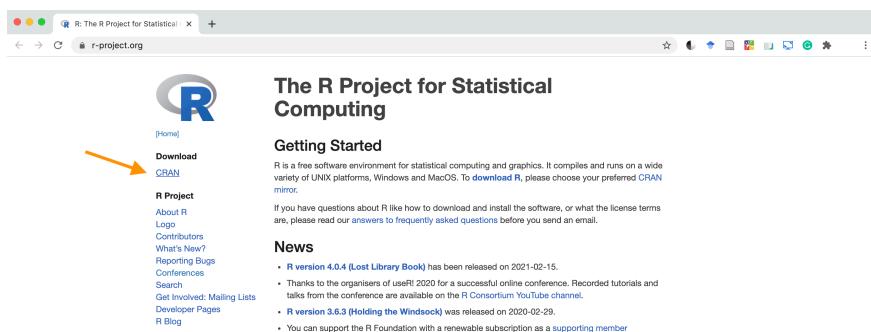
Chapter 1

Installare R e RStudio

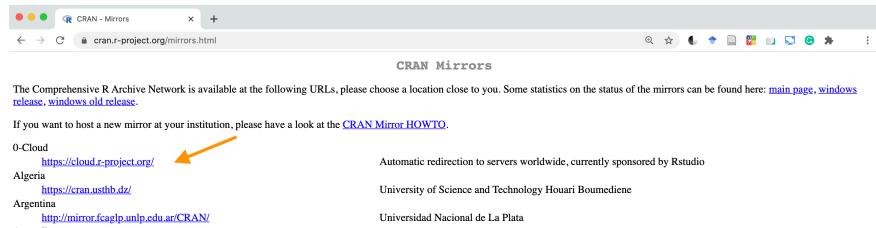
R ed R-studio sono due software distinti. R è un linguaggio di programmazione usato in particolare in ambiti quali la statistica. R-studio invece è un'interfaccia *user-friendly* che permette di utilizzare R. R può essere utilizzato autonomamente tuttavia è consigliato l'utilizzo attraverso R-studio. 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

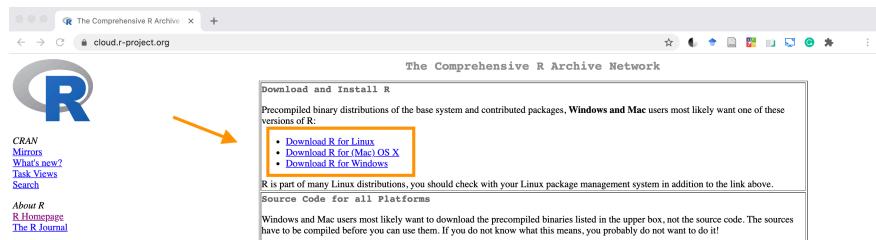
1. Accedere al sito <https://www.r-project.org>
2. Selezionare la voce **CRAN** (Comprehensive R Archive Network) dal menù di sinistra sotto **Download**



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

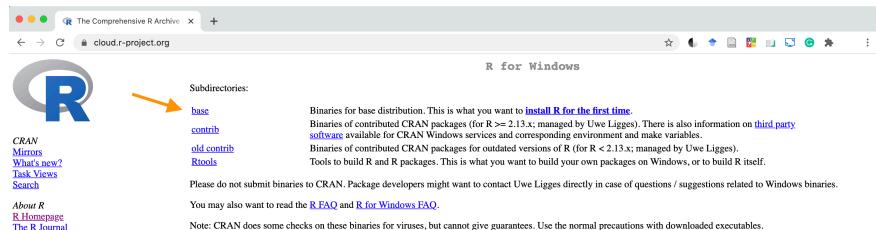


4. Selezionare il proprio sistema operativo



1.1.1 R Windows

1. Selezionare la voce **base**



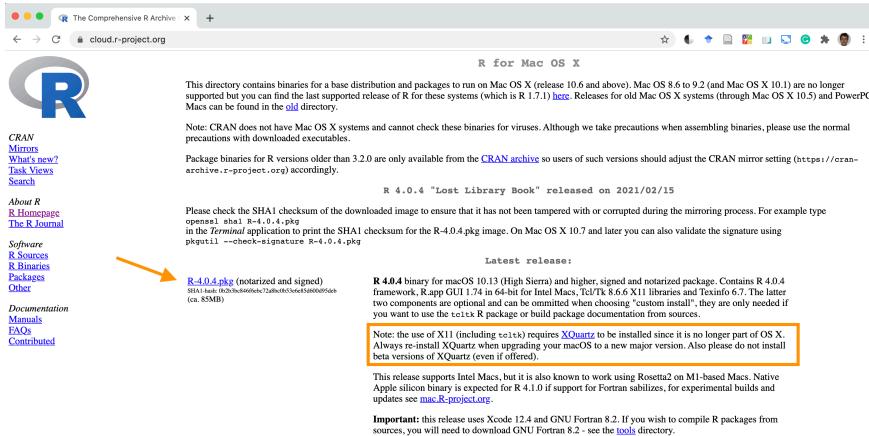
2. Selezionare la voce **Download** della versione più recente di R disponibile



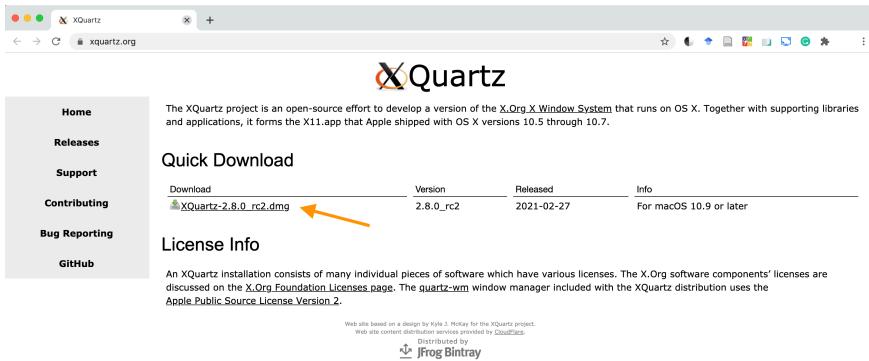
3. Al termine del download, eseguire il file e seguire le istruzioni fino al termine dell'installazione

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

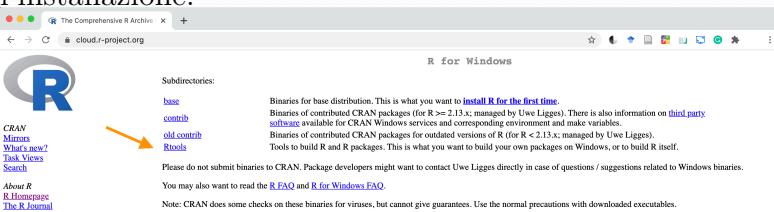
Per qualsiasi difficoltà o errore, soprattutto con il mondo Linux, una ricerca su online risolve sempre il problema.

 Approfondimento: R Tools

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

Windows

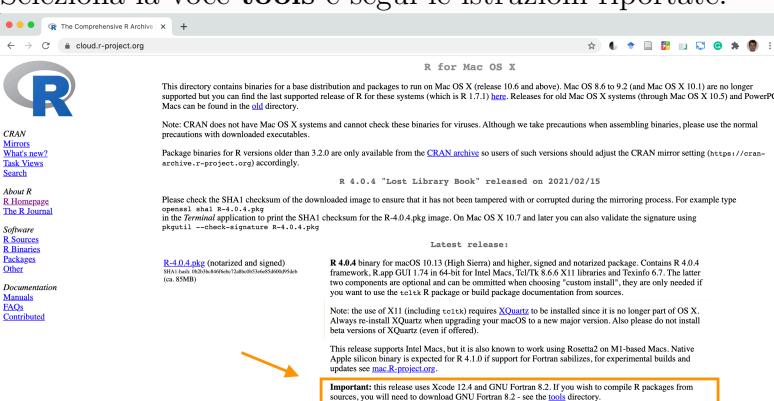
Seleziona la voce **Rtools** e segui le istruzioni per completare l'installazione.



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

MacOS

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

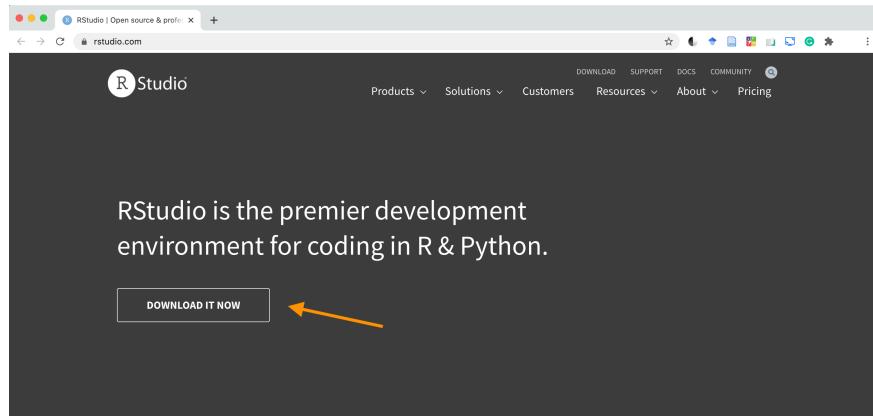


Nota in particolare che con R 4.0 le seguenti indicazioni sono riportate.

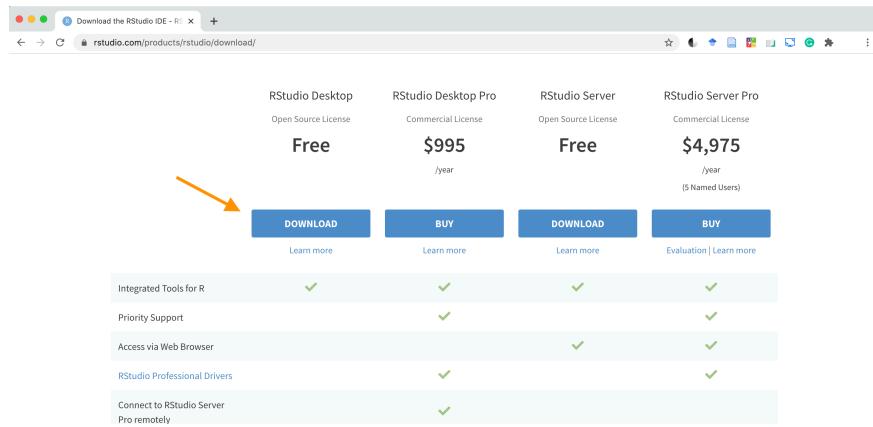


1.2 Installare R Studio

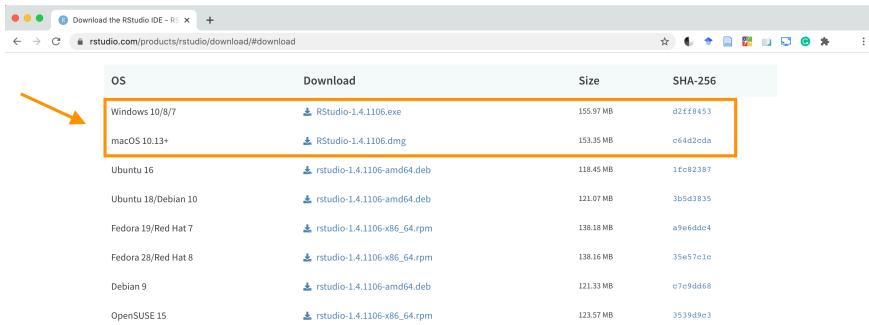
1. Accedere al sito <https://rstudio.com>
2. Selezionare la voce **DOWNLOAD IT NOW**



3. Selezionare la versione gratuita di RStudio Desktop



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



A screenshot of a web browser displaying the RStudio download page. The URL in the address bar is rstudio.com/products/rstudio/download/#download. The page lists download links for various operating systems. An orange arrow points to the 'OS' dropdown menu, which is currently set to 'Windows 10/8/7'. Other options visible in the dropdown include 'macOS 10.13+' and 'Ubuntu 16'. The table below shows the available download links for each OS.

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

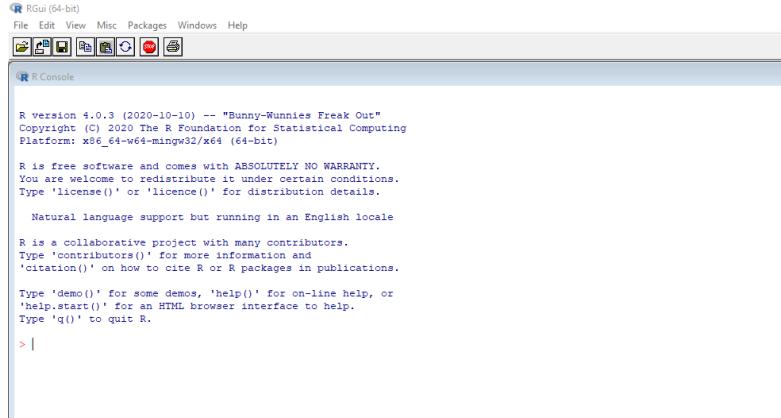


Figure 2.1: La console di R, solo per veri intenditori

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):

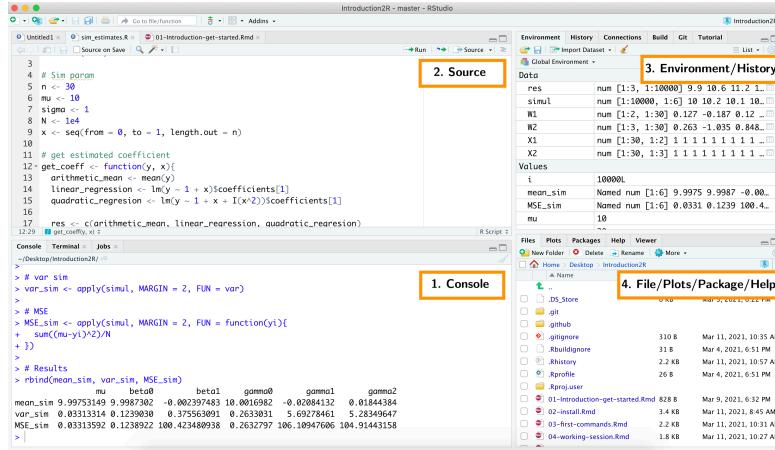


Figure 2.2: Interfaccia utente di Rstudio con i suoi 4 pannelli

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 il 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).

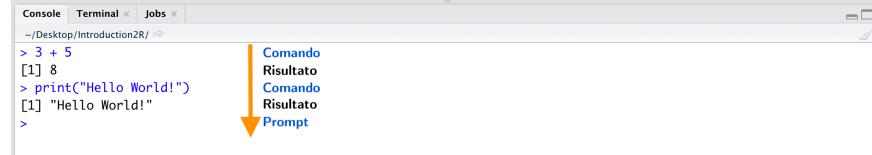
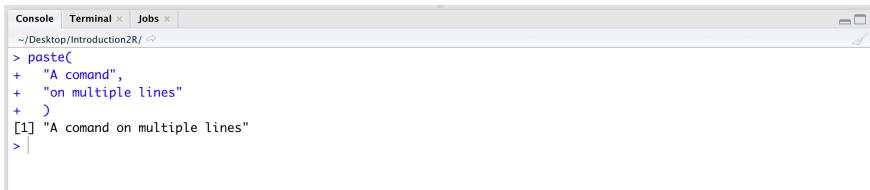


Figure 2.3: Esecuzione di comandi direttamente nella console

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"
>
```

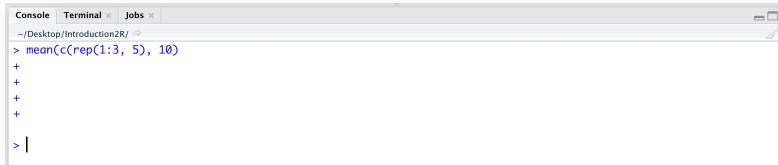
Figure 2.4: Esecuzione di un comando su più righe

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 Figure 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).



A screenshot of the RStudio Console window. The title bar says "Console Terminal Jobs". The console area shows several lines of code starting with a plus sign ('+') and ending with a greater than sign ('>'), indicating an infinite loop or a command that is taking too long to execute. The text includes:

```
-> 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 >*).



A screenshot of the RStudio Console window. The title bar says "Console Terminal Jobs". The console area shows several lines of code starting with a plus sign ('+') and ending with a greater than sign ('>'), indicating an infinite loop or a command that is taking too long to execute. The text includes:

```
-> res <- replicate(1e6, {
+   y <- rnorm(30, 10, 1)
+   mean(y)
+ })
```

An orange arrow points to the red circular "STOP" button located in the top right corner of the RStudio interface.

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.

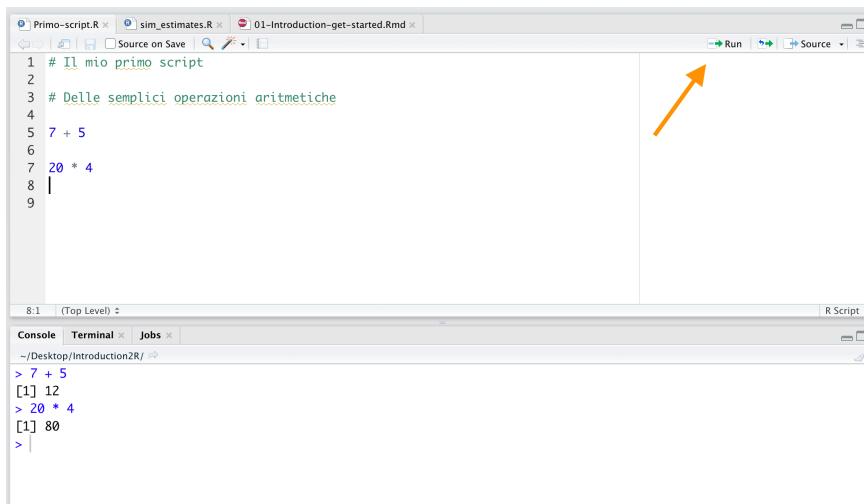


Figure 2.5: Esecuzione di un comando da script premi ‘command + invio’ (MacOs)/ ‘Ctrl+R’ (Windows) o premi il tasto indicato dalla freccia

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 4 e Capitolo TODO.

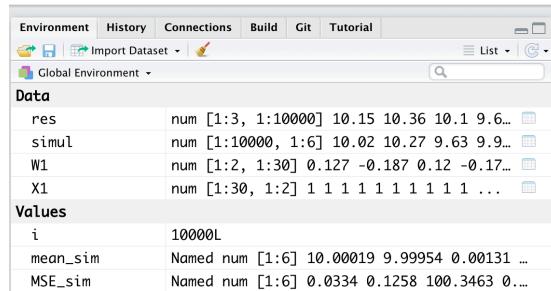


Figure 2.6: *Environment* - Elenco degli oggetti e variabili presenti nell'ambiente di lavoro

- **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

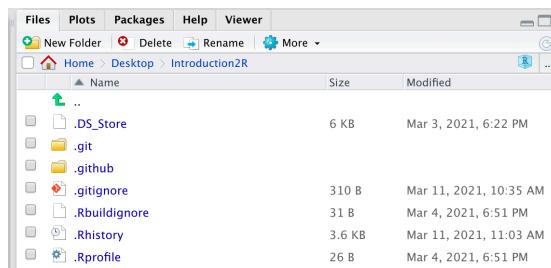


Figure 2.7: *Files* - permette di navigare tra i file del proprio computer

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

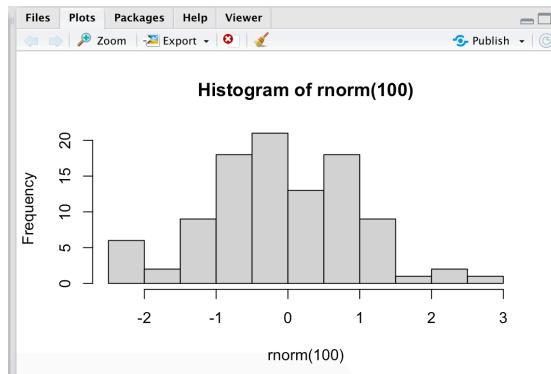


Figure 2.8: *Plots* - presentazione dei grafici

- **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).

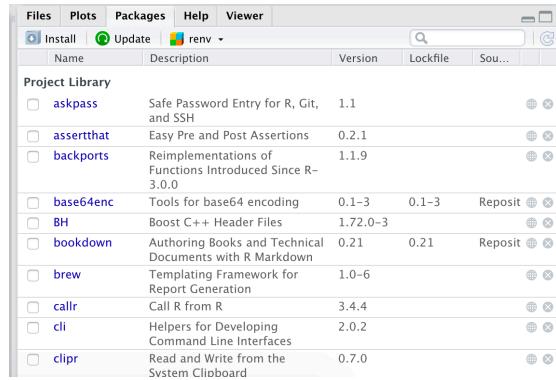


Figure 2.9: *Packages* - elenco dei pacchetti di R

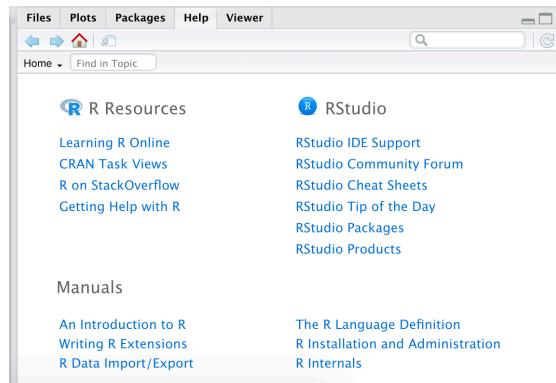
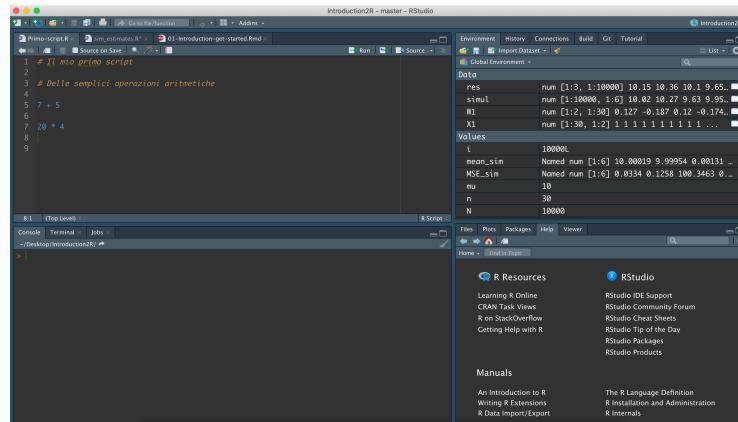


Figure 2.10: *Help* - documentazione di R

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>> 5 + 3 [1] 8</code>
<code>x - y</code>	Sottrazione	<code>> 7 - 2 [1] 5</code>
<code>x * y</code>	Moltiplicazione	<code>> 4 * 3 [1] 12</code>
<code>x / y</code>	Divisione	<code>> 8 / 3 [1] 2.666667</code>
<code>x %% y</code>	Resto della divisione	<code>> 7 %% 5 [1] 2</code>
<code>x %/% y</code>	Divisione intera	<code>> 7 %/% 5 [1] 1</code>
<code>x ^ y</code>	Potenza	<code>> 3^3 [1] 27</code>
<code>abs(x)</code>	Valore assoluto	<code>> abs(3-5^2) [1] 22</code>
<code>sign(x)</code>	Segno di un'espressione	<code>> sign(-8) [1] -1</code>
<code>sqrt(x)</code>	Radice quadrata	<code>> sqrt(25) [1] 5</code>
<code>log(x)</code>	Logaritmo naturale	<code>> log(10) [1] 2.302585</code>
<code>exp(x)</code>	Esponenziale	<code>> 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>> sin(pi/2) [1] 1 > cos(pi/2) [1] 6.123234e-17</code>
<code>factorial(x)</code>	Fattoriale	<code>> factorial(6) [1] 720</code>
<code>choose(n, k)</code>	Coefficiente binomiale	<code>> 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 nel Capitolo 4.2.

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. \wedge (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 (soluzioni):

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

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

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 π 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

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).

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”.

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.



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 3.2: Operatori Relazionali

Funzione	Nome	Esempio
<code>x == y</code>	Uguale	<code>> 5 == 3 [1] FALSE</code>
<code>x != y</code>	Diverso	<code>> 7 != 2 [1] TRUE</code>
<code>x > y</code>	Maggiore	<code>> 4 > 3 [1] TRUE</code>
<code>x >= y</code>	Maggiore o uguale	<code>> -2 >= 3 [1] FALSE</code>
<code>x < y</code>	Minore	<code>> 7 < 5 [1] FALSE</code>
<code>x <= y</code>	Minore o uguale	<code>> 7 <= 7 [1] TRUE</code>
<code>x %in% y</code>	inclusione	<code>> 5 %in% c(3, 5, 8) [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 frantendimenti. Infatti mentre TRUE e FALSE sono parole riservate (vedi Capitolo TODO) T a F non lo sono.

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

3.2.2 Operatori Logici

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.

Table 3.3: Operatori Logici

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

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 (vedi Tabella 3.4).

Table 3.4: Congiunzione '&'

x	y	<code>x & y</code>
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- 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 (vedi Tabella 3.5).

Table 3.5: Disgiunzione inclusiva '|'

x	y	$x \mid y$
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE



Approfondimento: Disgiunzione esclusiva

Per completezza ricordiamo che tra gli operatori logici esiste anche la **disgiunzione esclusiva**. La proposizione sarà valutata falsa se entrambe le relazioni sono vere oppure false. Affinchè la proposizione sia valutata vera una sola delle relazioni deve essere vera mentre l'altra deve essere falsa.

In R la disgiunzione esclusiva tra due relazioni (x e y) è indicata con la funzione `xor(x, y)`. Tuttavia tale funzione è raramente usata.

Table 3.6: Disgiunzione esclusiva ‘xor(x, y)’

x	y	<code>xor(x, y)</code>
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

3.2.3 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

Esegui i seguenti esercizi utilizzando gli operatori relazionali e logici (soluzioni):

1. Definisici 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 riguardanti l'utilizzo degli oggetti e delle funzioni che saranno successivamente approfonditi rispettivamente nel corso della seconda e della 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:

```
# Aggiornare un valore
x <- x*10
x
## [1] 50

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

Nel primo caso, abbiamo 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 `=` (attenti 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 la comprensione dei codici.

Ci sono alcune regole che discriminano 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 da R con un significato speciale (e.g., TRUE, FALSE, etc.; esegui il comando `?reserved` per la lista di tutte le parole riservate in R).



Warning-Box: CaSe-SeNsItIvE

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 il gender dei partecipanti del nostro studio).
- **della giusta lunghezza:** non deve essere ne troppo breve (evitare sigle incomprensibili) ne 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 Dati e Strutture Dati

Per lavorare in modo ottimale in R, è fondamentale conoscere bene e distinguere chiaramente quali sono le tipologie di dati e le strutture degli oggetti usati.

In R abbiamo 4 principali tipologie di dati, ovvero tipologie di valori che possono essere utilizzati:

- **character** - *Stringhe di caratteri* i cui valori alfannumerici vengono delimitati dalle doppie virgolette "Hello world!" o virgolette singole 'Hello world!'.

- **double** - *Valori numerici* con o senza cifre decimali ad esempio 27 o 93.46.
- **integer** - *Valori interi* definiti apponendo la lettera L al numero desiderato, ad esempio 58L.
- **logical** - *Valori logici* TRUE e FALSE usati nelle operazioni logiche.

```
typeof("Psicostat")
## [1] "character"
typeof(24.04)
## [1] "double"
typeof(1993L)
## [1] "integer"
typeof(TRUE)
## [1] "logical"
```

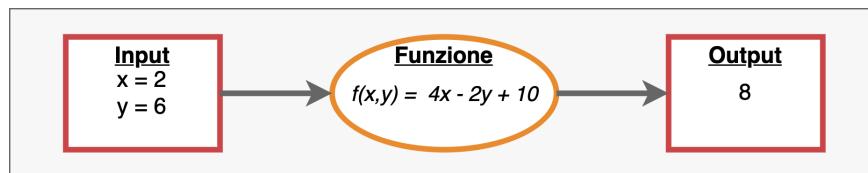
In R abbiamo inoltre differenti tipologie di oggetti, ovvero diverse strutture in cui possono essere organizzati i dati:

- **Vettori**
- **Matrici**
- **Dataframe**
- **Liste**

Approfondiremo la loro definizione, le loro caratteristiche ed il loro utilizzo nel corso di tutta la seconda sezione di questo libro TODO.

4.2 Funzioni

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



Abbiamo già incontrato le nostre prime funzioni per eseguire specifiche operazioni matematiche nel Capitolo 3.1 come ad esempio `sqrt()` o `abs()` usate per ottenere ripetutamente la radice quadrata o il valore assoluto di un numero. Ovviamente le funzioni in R non sono limitate ai soli calcoli matematici ma possono eseguire qualsiasi genere di compito come ad esempio creare grafici e tavole o manipolare dei dati o dei file. Tuttavia il concetto rimane lo stesso:

ricevuti degli oggetti in input, le funzioni compiono determinate azioni e restituiscono dei nuovi oggetti in output.

In realtà incontreremo delle funzioni che non richiedono input o non producono output. Ad esempio `getwd()` non richiede input oppure la funzione `rm()` non produce output. Tuttavia questo accade nella minoranza dei casi.

Per eseguire una funzione in R è necessario digitare il nome della funzione ed indicare tra parentesi i valori che vogliamo assegnare agli **argomenti** della funzione, ovvero i nostri input, separati da virgole. Generalmente si utilizza quindi la seguente sintassi:

```
<nome-funzione>(<nome-arg1> = <valore-arg1>, <nome-arg2> = <valore-arg2>, ...)
```

Ad esempio per creare una sequenza di valori con incrementi di 1 posso usare la funzione `seq()`, i cui argomenti sono `from` e `to` ed indicano rispettivamente il valore iniziale ed il valore massimo della sequenza.

```
# creo una sequenza di valori da 0 a 10 con incrementi di 1
seq(from = 0, to = 10)
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

4.2.1 Argomenti di una Funzione

Nel definire gli argomenti di una funzione non è necessario specificare il nome degli argomenti. Ad esempio il comando precedente può essere eseguito anche specificando solamente i valori.

```
# creo una sequenza di valori da 0 a 10 con incrementi di 1
seq(0, 10)
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

Tuttavia, questo rende più difficile la lettura e la comprensione del codice poichè non è chiaro a quali argomenti si riferiscono i valori. L'ordine con cui vengono definiti i valori in questo caso è importante, poichè R assume rispetti l'ordine prestabilito degli argomenti. Osserva come invertendo invertendo i valori ovviamente otteniamo risultati differenti da quelli precedenti, ma questo non avviene quando il nome dell'argomento è specificato.

```
# inverti i valori senza i nomi degli argomenti
seq(10, 0)
## [1] 10 9 8 7 6 5 4 3 2 1 0

# inverti i valori con i nomi degli argomenti
seq(to = 10, from = 0)
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

Vediamo inoltre come le funzioni possano avere molteplici argomenti, ma che non sia necessario specificare il valore per ognuno di essi. Molti argomenti, infatti, hanno già dei valori prestabiliti di *default* e non richiedono quindi di essere specificati almeno che ovviamente non si vogliono utilizzare impostazioni diverse da quelle di *default*. Oppure lo specificare un dato argomento rispetto ad un altro può definire il comportamento stesso della funzione.

Ad esempio la funzione `seq()` possiede anche gli argomenti `by` e `length.out` che prima non erano stati specificati. `by` permette di definire l'incremento per ogni elemento successivo della sequenza mentre `length.out` permette di definire il numero di elementi della sequenza. Vediamo come allo specificare dell'uno o dell'altro argomento (o di entrambi) il comportamento della funzione `vari`.

```
seq(from = 0, to = 10, by = 5)
## [1] 0 5 10
seq(from = 0, to = 10, length.out = 5)
## [1] 0.0 2.5 5.0 7.5 10.0
seq(from = 0, to = 10, length.out = 5, by = 4)
## Error in seq.default(from = 0, to = 10, length.out = 5, by = 4): too many arguments
```

E' pertanto consigliabile esplicitare sempre gli argomenti di una funzione per rendere chiaro a che cosa si riferiscono i valori indicati. Questo è utile anche per evitare eventuali comportamenti non voluti delle funzioni ad individuare più facilmente possibili errori.

Gli argomenti di una funzione, inoltre, richiedono specifiche tipologie e strutture di dati e sta a noi assicuraci che i dati siano forniti nel modo corretto. Vediamo ad esempio come la funzione `mean()` che calcola la media di un insieme di valori, richieda come input un vettore di valori numerici. Approfondiremo il concetto di vettori nel Capitolo TODO, al momento ci basta sapere che possiamo usare la funzione `c()` per combinare più valori in un unico vettore.

```
# Calcolo la media dei seguenti valori (numerici)
mean(c(10, 6, 8, 12)) # c() combina più valori in un unico vettore
## [1] 9

mean(10, 6, 8, 12)
## [1] 10
```

Notiamo come nel primo caso il risultato sia corretto mentre nel secondo è sbagliato. Questo perché `mean()` richiede come primo argomento il vettore su cui calcolare la media. Nel primo caso abbiamo correttamente specificato il vettore di valori usando la funzione `c()`. Nel secondo caso invece, il primo argomento risulta essere solo il valore 10 ed R calcola la media di 10 ovvero 10. Gli altri valori sono passati ad altri argomenti che non alterano il comportamento ma neppure ci segnalano di questo importante errore.

Nel seguente esempio, possiamo vedere come `mean()` richieda che i valori siano numerici. Seppur "1" "2", e "3" siano dei numeri, l'utilizzo delle doppie virgolette li rende delle stringhe di caratteri e non dei valori numerici e giustamente R non può eseguire una media su dei caratteri.

```
# Calcolo la media dei seguenti valori (caratteri)
mean(c("1", "2", "3"))
## Warning in mean.default(c("1", "2", "3")): argument is not numeric or logical:
## returning NA
## [1] NA
```

Capiamo quindi che per usare correttamente le funzioni è fondamentale conoscerne gli argomenti e rispettare le tipologie e strutture di dati richieste.

4.2.2 Help! I need Somebody...Help!

Conoscere tutte le funzioni e tutti i loro argomenti è impossibile. Per fortuna R ci viene in soccorso fornendoci per ogni funzione la sua documentazione. Qui vengono fornite tutte le informazioni riguardanti la finalità della funzione, la descrizione dei suoi argomenti, i dettagli riguardanti i suoi possibili utilizzi.

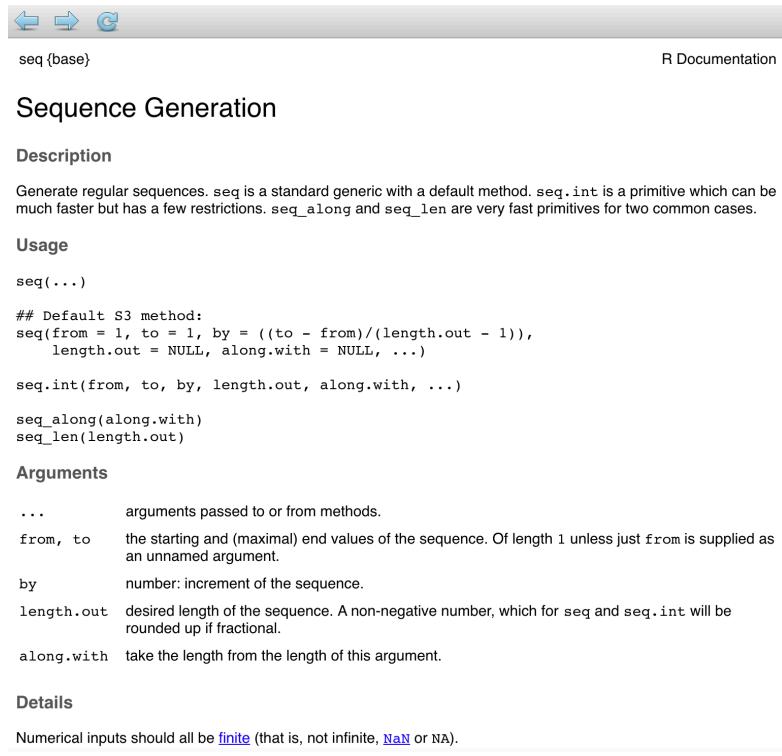
Per accedere alla documentazione possiamo utilizzare il comando `?<nome-funzione>` oppure `help(<nome-funzione>)`. Ad esempio:

```
?seq
help(seq)
```

Una pagina si aprirà nel pannello “Help” in basso a destra con la documentazione della funzione in modo simile a quanto rappresentato in Figura 4.1.

Il formato e le informazioni presenti nella pagina seguono delle norme comuni ma non obbligatorie. Infatti, non necessariamente vengono usati sempre tutti i campi e comunque all'autore delle funzioni è lasciato un certo grado di libertà nel personalizzare la documentazione. Tra i campi principali e più comunemente usati abbiamo:

- **Tiolo** - Titolo esplicativo della finalità della funzione
- **Description** - Descrizione concisa della funzione
- **Usage** - Viene mostrata la struttura della funzione con i suoi argomenti e valori di default
- **Arguments** - Elenco con la descrizione dettagliata di tutti gli argomenti. Qui troviamo per ogni argomento sia le opzioni utilizzabili ed il loro effetto, che la tipologia di valori richiesti



The screenshot shows the R Documentation page for the `seq` function. At the top, there are navigation icons for back, forward, and search. Below them, the package name `seq {base}` is displayed, along with a link to the R Documentation. The main title is **Sequence Generation**. Under the **Description** section, it says: "Generate regular sequences. `seq` is a standard generic with a default method. `seq.int` is a primitive which can be much faster but has a few restrictions. `seq_along` and `seq_len` are very fast primitives for two common cases." The **Usage** section contains the R code for the function:

```
seq(...)

## Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...)

seq.int(from, to, by, length.out, along.with, ...)

seq_along(along.with)
seq_len(length.out)
```

The **Arguments** section lists the arguments and their descriptions:

- `...` arguments passed to or from methods.
- `from, to` the starting and (maximal) end values of the sequence. Of length 1 unless just `from` is supplied as an unnamed argument.
- `by` number: increment of the sequence.
- `length.out` desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.
- `along.with` take the length from the length of this argument.

The **Details** section notes: "Numerical inputs should all be `finite` (that is, not infinite, `NaN` or `NA`).

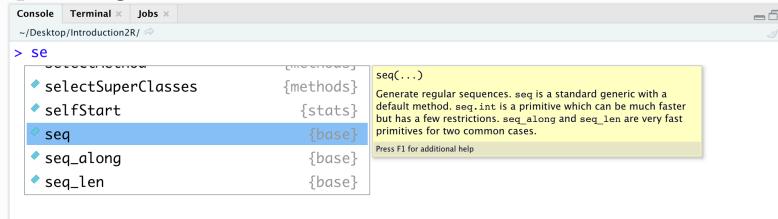
Figure 4.1: Help-page della funzione `seq()`

- **Details** - Descrizione dettagliata della funzione considerando i casi di utilizzo ed eventuali note tecniche
- **Value** - Descrizione dell'output dalla funzione. Qui troviamo sia la descrizione della struttura dei dati dell'output che la descrizione dei suoi elementi utile per interpretare e utilizzare i risultati ottenuti
- **See Also** - Eventuali link ad altre funzioni simili o in relazione con la nostra funzione
- **Examples** - Esempi di uso della funzione

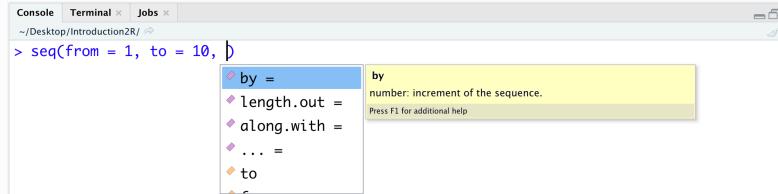
Trick-Box: Autocompletamento with 'Tab'

La natura dei programmati è essere pigri e smemorati. Per fortuna ogni *code editor* che si rispetti (i.e., programma per la scrittura di codici) possiede delle utili funzioni di autocompletamento e suggerimento dei comandi che semplificano la scrittura di codici.

In Rstudio, i suggerimenti compaiono automaticamente durante la scrittura di un comando oppure possono essere richiamati premendo il tasto **Tab** in alto a sinistra della tastiera (). Comparirà una finestra con possibili soluzioni di autocompletamento del nome della funzione. Utilizzando le frecce della tastiera possiamo evidenziare la funzione voluta e premere **Invio** per autocompletare il comando. Nota come accanto al nome della funzione appare anche un piccolo riquadro giallo con la descrizione della funzione.



Per inserire gli argomenti della funzione possiamo fare affidamento nuovamente ai suggerimenti e alla funzione di autocompletamento. Sarà sufficiente premere nuovamente il tasto **Tab** e questa volta comparirà una lista degli argomenti con la relativa descrizione. Sarà quindi sufficiente selezionare con le frecce l'argomento desiderato e premere **Invio**.



Noteate come la funzione di autocompletamento non sia utilizzata solo per le funzioni ma anche per i nomi degli oggetti. Questo ci consentirà di richiamare velocemente oggetti precedentemente creati evitando di digitare l'intero nome.



Chapter 5

Sessione di Lavoro

In questo capitolo introdurremo alcuni concetti molto importanti che riguardano le sessioni di lavoro in R o RStudio. In particolare parleremo dell'*environment*, della *working directory* e dell'utilizzo di pacchetti.

Infine, disuteremo di alcuni aspetti generali della programmazione quali la gestione dei messaggi di errore o *warnings* e vedremo alcune buone norme riguardanti l'organizzazione degli scripts e l'uso degli *RStudio Projects* per essere ordinati ed efficaci nelle proprie sessioni di lavoro.

5.1 Environment

Nel Capitolo 4.1, abbiamo visto come sia possibile assegnare dei valori a degli oggetti. Questi oggetti vengono creati nel nostro ambiente di lavoro (o meglio *Environment*) e potranno essere utilizzati in seguito.

Il nostro Environment raccoglie quindi tutti gli oggetti che vengono creati durante la nostra sessione di lavoro. E' possibile valutare gli oggetti attualmente presenti osservando il pannello *Envrionmen* in alto a destra (vedi Figura 5.1) oppure utilizzandno il comando `ls()`, ovvero *list objects*.

All'inizio della sessione di lavoro il nostro Environment sarà vuoto () .

```
# Environment vuoto
ls()
## character(0)

# Creo oggetti
x <- c(2,4,6,8)
y <- 27
```

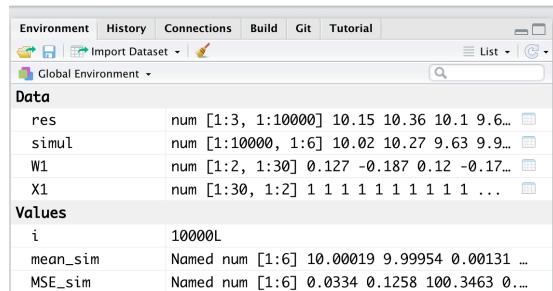


Figure 5.1: *Environment* - Elenco degli oggetti e variabili presenti nel'ambiente di lavoro



Figure 5.2: *Environment* vuoto ad inizio sessione di lavoro

```
word <- "Hello Word!"

# Lista nomi oggetti nell'Environment
ls()
## [1] "word"  "x"      "y"
```

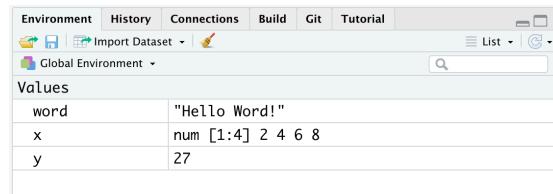


Figure 5.3: *Environment* contenente gli oggetti creati

- concetto di ambiente di lavoro
- ls()
- rm()
- tip box aspetto transitorio (vedremo successivamente come salvare caricare dati)

5.2 Working Directory

- che cosa è un path
- home directory ~ (non mi ricordo se per windows lavora)
- getwd();
- setwd(); altri modi per settare al working directory
- abbsolute/relative path (?)

5.3 R-packages

- scaricare
- library
- uso funzioni
- aggiornare i pacchetti
- box tip per l'uso di ::

5.4 Errors and warnings

5.5 Sessione di lavoro

- pulizia script
- commenti
- sezioni script
- settings
- sintassi (gli spazi e gli indent corretti alineamenti)
- idea di organizzare in vari script, cartelle

5.5.1 R projects

5.6 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 programmatore 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 with the following details:

- Search Query:** Error in : could not find function ""
- Results Count:** About 1,740,000,000 results (0.63 seconds)
- Top Result:** [Error: could not find function ... in R - Stack Overflow](#)
- Sub-Header:** 10 answers
- Text Preview:** 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 ...
- Related Questions:**
 - Error: could not find function "%>%" - Stack Overflow
 - Could not find function "%<%>" with dplyr loaded ...
 - dplyr error - could not find function "%>%" - Stack ...
 - "could not find function" error though function is in ...
- Dates:** Jun 4, 2017; Mar 27, 2019; May 29, 2019; Jul 26, 2018
- Link:** More results from stackoverflow.com

5.6.0.1 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 generale¹:

- Titolo: un super riassunto del problema
- Contesto: linguaggio (es. R), quale sistema operativo (es. Windows)

¹Fonte: Writing the perfect question - Jon Skeet

- 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`.

Struttura Dati

Introduzione

Working in progress.

Chapter 6

Data Type

Working in progress.

Chapter 7

Vettori

Working in progress

7.1 Creazione di Vettori

In R per definire un vettore si utilizza il comando `<nome-vettore> <- c(<oggetti>)`. Ricorda che gli elementi devono essere separati da una virgola.

Esercizi

1. Crea il vettore `x` contenente i numeri 4, 6, 12, 34, 8
2. Crea il vettore `y` contenente tutti i numeri pari compresi tra 1 e 25 (`?seq()`)
3. Crea il vettore `z` contenente tutti i primi 10 multipli di 7 partendo da 13 (`?seq()`)
4. Crea il vettore `s` in cui le lettere "A", "B" e "C" vengono ripetute nel medesimo ordine 4 volte (`?rep()`).
5. Crea il vettore `t` in cui le lettere "A", "B" e "C" vengono ripetute ognuna 4 volte (`?rep()`).

7.2 Selezione Elementi di un Vettore

In R per selezionare gli elementi di un vettore si deve indicare all'interno delle parentesi quadre la **posizione degli elementi** da selezionare, non il valore dell'elemento stesso:

`<nome-vettore>[<indice-posizione>]` In alternativa si può definire la condizione logica che gli elementi che si vogliono selezionare devono rispettare.

Per *\textbf{eliminare degli elementi} da un vettore si utilizza all'interno delle parentesi quadre l'operatore “-” insieme agli indici di posizione degli elementi da eliminare (esempio: `x[c(-2,-4)]` oppure `x[-c(2,4)]`).

Esercizi

1. Del vettore `x` seleziona il 2°, 3° e 5° elemento
2. Del vettore `y` seleziona tutti i valori minori di 13 o maggiori di 19
3. Del vettore `z` seleziona tutti i valori compresi tra 24 e 50
4. Elimina dal vettore `z` i valori 28 e 42
5. Del vettore `s` seleziona tutti gli elementi uguali ad “A”
6. Del vettore `t` seleziona tutti gli elementi diversi da “B”.

7.3 Funzioni ed Operazioni tra Vettori

Per compiere operazioni tra vettori è necessario che essi abbiano identica lunghezza.

Table 7.1: Operazioni con vettori

Operazione	Nome
<code><nuovo-vettore> <- c(<vettore1>, <vettore2>)</code>	Per unire più vettori in un unico vettore
<code>length(<nome-vettore>)</code>	Per valutare il numero di elementi contenuti in
<code>vettore1 + vettore2</code>	Somma di due vettori
<code>vettore1 - vettore2</code>	Differenza tra due vettori
<code>vettore1 * vettore2</code>	Prodotto tra due vettori
<code>vettore1 / vettore2</code>	Rapporto tra due vettori

Nota: In R il prodotto e rapporto tra vettori sono eseguiti elemento per elemento (al contrario di molti altri software).

Esercizi

1. Crea il vettore `j` unendo i vettori `x` ed `z`.
2. Elimina gli ultimi tre elementi del vettore `j` e controlla che i vettori `j` e `y` abbiano la stessa lunghezza.
3. Calcola la somma tra i vettori `j` e `y`.
4. Moltiplica il vettore `z` per una costante `k=3`.
5. Calcola il prodotto tra i primi 10 elementi del vettore `y` ed il vettore `z`.

Chapter 8

Fattori

Working in progress.

Chapter 9

Matrici

Working in progress.

9.1 Creazione di Matrici

```
<nome-matrice> <- matrix(data, nrow=n, ncol=s, byrow=FALSE)
```

Nota: Di default R riempie la matrice per colonne, impostando `byrow = TRUE` si riempie per righe.

Esercizi

1. Crea la matrice A così definita:

$$\begin{matrix} 2 & 34 & 12 & 7 \\ 46 & 93 & 27 & 99 \\ 23 & 38 & 7 & 04 \end{matrix}$$

2. Crea la matrice B contenente tutti i primi 12 numeri dispari disposti su 4 righe e 3 colonne.
3. Crea la matrice C contenente i primi 12 multipli di 9 disposti su 3 righe e 4 colonne.
4. Crea la matrice D formata da 3 colonne in cui le lettere "A","B" e "C" vengano ripetute 4 volte ciascuna rispettivamente nella prima, seconda e terza colonna.
5. Crea la matrice E formata da 3 righe in cui le lettere "A","B" e "C" vengano ripetute 4 volte ciascuna rispettivamente nella prima, seconda e terza riga.

9.2 Selezione di Elementi di una Matrice

In R per selezionare gli elementi di matrice si deve indicare all'interno delle parentesi quadre l'indice di riga e l'indice di colonna (**separati da virgola**) degli elementi da selezionare oppure la condizione logica che devono rispettare.

```
<nome-matrice>[<indice-riga>, <indice-colonna>]
```

Nota: per selezionare tutti gli elementi di una data riga o di una data colonna basta lasciare vuoto rispettivamente l'indice di riga o l'indice di colonna.

Esercizi

1. Utilizzando gli indici di riga e di colonna seleziona il numero 27 della matrice A
2. Seleziona gli elementi compresi tra la seconda e quarta riga, seconda e terza colonna della matrice B
3. Seleziona solo gli elementi pari della matrice A (Nota: utilizza l'operazione resto %%)
4. Elimina dalla matrice C la terza riga e la terza colonna
5. Seleziona tutti gli elementi della seconda e terza riga della matrice B
6. Seleziona tutti gli elementi diversi da “B” appartenenti alla matrice D

9.3 Funzioni ed Operazioni tra Matrici

Note:

- Per il significato di determinante di una matrice considera: <https://it.wikipedia.org/wiki/Determinante>
- Per il significato di matrice inversa considera: https://it.wikipedia.org/wiki/Matrice_invertibile
- Per compiere operazioni elemento per elemento tra due matrici, esse devono avere la stessa dimensione
- Per compiere il prodotto matriciale il numero di colonne della prima matrice deve essere uguale al numero di righe della seconda matrice (vedi https://it.wikipedia.org/wiki/Moltiplicazione_di_matrici).
- E' possibile assegnare nomi alle colonne e righe di una matrice rispettivamente attraverso i comandi:

```
colnames(<nome-matrice>) <- c("nome-1", ..., "nome-s")}
```

```
rownames(<nome-matrice>) <- c("nome-1", ..., "nome-n")}
```

Table 9.1: Operazioni con matrici

Operazione	Nome
<code><nuova-matrice> <- cbind(<matrice1>, <matrice2>)</code>	Per unire due matrici creando nuove colonne (le matrici devono avere la stessa dimensione di righe)
<code><nuova-matrice> <- rbind(<matrice1>, <matrice2>)</code>	Per unire due matrici creando nuove righe (le matrici devono avere la stessa dimensione di colonne)
<code>nrow(<nome-matrice>)</code>	Per valutare il numero di righe della matrice
<code>ncol(<nome-matrice>)</code>	Per valutare il numero di colonne della matrice
<code>dim(<nome-matrice>)</code>	Per valutare la dimensione della matrice (righe e colonne)
<code>t(<nome-matrice>)</code>	Per ottenere la trasposta della matrice
<code>diag(<nome-matrice>)</code>	Ottenere un vettore con gli elementi della diagonale della matrice
<code>det(<nome-matrice>)</code>	Ottenere il determinante della matrice (la matrice deve essere quadrata)
<code>solve(<nome-matrice>)</code>	Ottenere l'inversa della matrice
<code>colnames(<nome-matrice>)</code>	Nomi delle colonne della matrice
<code>rownames(<nome-matrice>)</code>	Nomi delle righe della matrice
<code>matrice1 + matrice2</code>	Somma elemento per elemento di due matrici
<code>matrice1 - matrice2</code>	Differenza elemento per elemento tra due matrici
<code>matrice1 * matrice2</code>	Prodotto elemento per elemento tra due matrici
<code>matrice1 / matrice2</code>	Rapporto elemento per elemento tra due matrici
<code>matrice1 %*% matrice2</code>	Prodotto matriciale

Esercizi

1. Crea la matrice **G** unendo alla matrice **A** le prime due colonne della matrice **C**
2. Crea la matrice **H** unendo alla matrice **C** le prime due righe della matrice trasposta di **B**
3. Ridefinisci la matrice **A** eliminando la seconda colonna. Ridefinisci la matrice **B** eliminando la prima riga. Verifica che le matrici così ottenute abbiano la stessa dimensione.
4. Commenta i differenti risultati che otteniamo nelle operazioni **A*B**, **B*A**, **A%*%B** e **B%*%A**.
5. Assegna i seguenti nomi alle colonne e alle righe della matrice **C**: "col_1", "col_2", "col_3", "col_4", "row_1", "row_2", "row_3".

Chapter 10

Dataframe

Working in progress.

Chapter 11

Liste

Working in progress.

Algoritmi

Introduzione

Working in progress.

Chapter 12

Definizione di Funzioni

Working in progress.

Chapter 13

Programmazione Condizionale

Working in progress.

Chapter 14

Attenti al loop

Working in progress.

Case study

Introduzione

Working in progress.

Chapter 15

Caso Studio I: Attaccamento

Working in progress.

15.1 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!