

Introduzione a R

Corso per imparare le basi di **R**

Psicostat

21-03-2021



Contents

Presentazione	7
Perchè R	7
Struttura del libro	7
Risorse Utili	8
Psicostat	8
Collaborazione	8
Riconoscimenti	8
Licenza	9
 Get Started	 13
 Intorduzione	 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	35
 4 Due Compagni Inseparabili	 41
4.1 Oggetti	41
4.2 Funzioni	45

5 Ambiente di Lavoro	51
5.1 Environment	51
5.2 Working Directory	55
5.3 R-packages	61
6 Sessione di Lavoro	67
6.1 Organizzazione Script	67
6.2 R projects	74
6.3 Messages, Warnings e Errors	74
Struttura Dati	81
Introduzione	81
7 Data Type	83
7.1 Vettori Numerici	83
7.2 Vettori logici	83
7.3 Vettori di caratteri	84
8 Vettori	85
8.1 Creazione di Vettori	85
8.2 Selezione Elementi di un Vettore	85
8.3 Funzioni ed Operazioni tra Vettori	86
9 Fattori	87
10 Matrici	89
10.1 Creazione di Matrici	89
10.2 Selezione di Elementi di una Matrice	89
10.3 Funzioni ed Operazioni tra Matrici	90
11 Dataframe	93
11.1 Creazione di DataFrames	93
11.2 Selezione di Elementi di un DataFrame	94
11.3 Funzioni con DataFrames	95
12 Liste	97
12.1 Creazione di Liste	97
12.2 Selezione di Elementi di una Lista	97

CONTENTS	5
Algoritmi	101
Introduzione	101
13 Definizione di Funzioni	103
14 Programmazione Condizionale	105
15 Attenti al loop	107
Case study	111
Introduzione	111
16 Caso Studio I: Attaccamento	113
16.1 Infobox	113

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 avrai imparato ad utilizzare uno strumento 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.** Eseguiremo 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

Intorduzione

In questa sezione verranno prima presentate le istruzioni per installare R ed RStudio. Successivamente, svolgeremo le prime operazioni in R e famiglierizzeremo con dei concetti di base della programmazione quali gli oggetti e le funzioni. Introdurremo infine altri concetti relativi alle sessioni di lavoro in R e descriveremo alcune buone regole per nell'utilizzo di 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 - Ambiente di Lavoro.** Introduzione dei concetti di Enviernement, working directory e dei pacchetti di R.
- **Capitolo 6 - Sessione di Lavoro.** Descrizione di buone pratiche nelle sessioni di lavoro e gestione degli errori.

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](#).

O-Cloud https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rstudio
Algeria https://cran.usthb.dz/	University of Science and Technology Houari Boumediene
Argentina http://mirror.fcaglp.unlp.edu.ar/CRAN/	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](#)

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
- 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

The Comprehensive R Archive

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) [here](#). Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the [old](#) directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

Package binaries for R versions older than 3.2.0 are only available from the [CRAN archive](#) so users of such versions should adjust the CRAN mirror setting (<https://cran-archive.r-project.org>) accordingly.

R 4.0.4 "Lost Library Book" released on 2021/02/15

Please check the SHA1 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type `openssl sha1 R-4.0.4.pkg` in the Terminal application to print the SHA1 checksum for the R-4.0.4.pkg image. On Mac OS X 10.7 and later you can also validate the signature using `pkutil --check-signature R-4.0.4.pkg`

Latest release:

R-4.0.4.pkg (notarized and signed)
SHA1-hash: 0b2b3b8460febc72a8c0b53afe85360085deb
(ca. 85MB)

R 4.0.4 binary for macOS 10.13 (High Sierra) and higher, signed and notarized package. Contains R 4.0.4 framework, R.app GUI 1.74 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 6.7. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `cltck` R package or build package documentation from sources.

Note: the use of X11 (including `cltck`) requires **XQuartz** to be installed since it is no longer part of OS X. Always re-install XQuartz when upgrading your macOS to a new major version. Also please do not install beta versions of XQuartz (even if offered).

This release supports Intel Macs, but it is also known to work using Rosetta2 on M1-based Macs. Native Apple silicon binary is expected for R 4.1.0 if support for Fortran stabilizes, for experimental builds and updates see [mac.R-project.org](#).

Important: this release uses Xcode 12.4 and GNU Fortran 8.2. If you wish to compile R packages from sources, you will need to download GNU Fortran 8.2 - see the [tools](#) directory.

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

XQuartz

The XQuartz project is an open-source effort to develop a version of the [X.Org X Window System](#) that runs on OS X. Together with supporting libraries and applications, it forms the X11.app that Apple shipped with OS X versions 10.5 through 10.7.

Quick Download

Download	Version	Released	Info
XQuartz-2.8.0_rc2.dmg	2.8.0_rc2	2021-02-27	For macOS 10.9 or later

License Info

An XQuartz installation consists of many individual pieces of software which have various licenses. The X.Org software components' licenses are discussed on the [X.Org Foundation Licenses page](#). The `quartz-wm` window manager included with the XQuartz distribution uses the [Apple Public Source License Version 2](#).

Web site based on a design by Kyle J. Mackay for the XQuartz project.
Web site content distribution services provided by Cloudflare.
Distributed by JFrog Bintray

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.

The screenshot shows the 'R for Windows' page. On the left, there's a sidebar with links like 'CRAN Mirrors', 'What's new?', 'Task Views', 'Search', 'About R', 'R Homepage', and 'The R Journal'. An orange arrow points from the 'Tools' link in this sidebar to the main content area. The main content area has a heading 'Subdirectories:' followed by 'base', 'contrib', 'old contrib', and 'Tools'. Below these, there's text about base binaries, contributed CRAN packages, and tools for building R packages. It also includes a note about not submitting binaries to CRAN and a warning about viruses.

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

MacOS

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

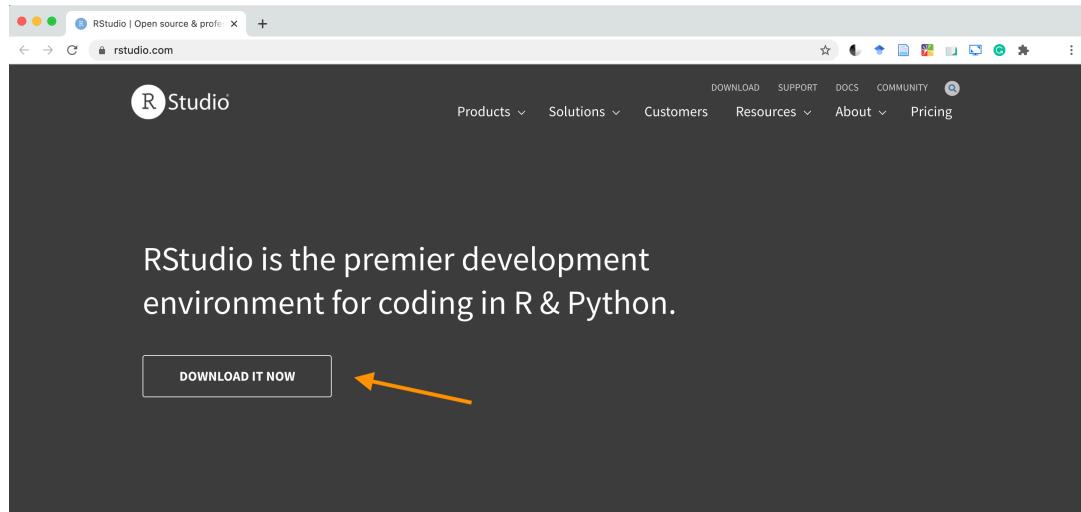
The screenshot shows the 'R for Mac OS X' page. On the left, there's a sidebar with links like 'CRAN Mirrors', 'What's new?', 'Task Views', 'Search', 'About R', 'R Homepage', and 'The R Journal'. An orange arrow points from the 'tools' link in this sidebar to the main content area. The main content area has a heading 'Latest release:' followed by 'R 4.0.4 "Lost Library Book" released on 2021/02/15'. It provides instructions for verifying the integrity of the download using SHA1 checksums and pgrepvill. A callout box highlights the note: 'Important: this release uses Xcode 12.4 and GNU Fortran 8.2. If you wish to compile R packages from sources, you will need to download GNU Fortran 8.2 - see the tools directory.'

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

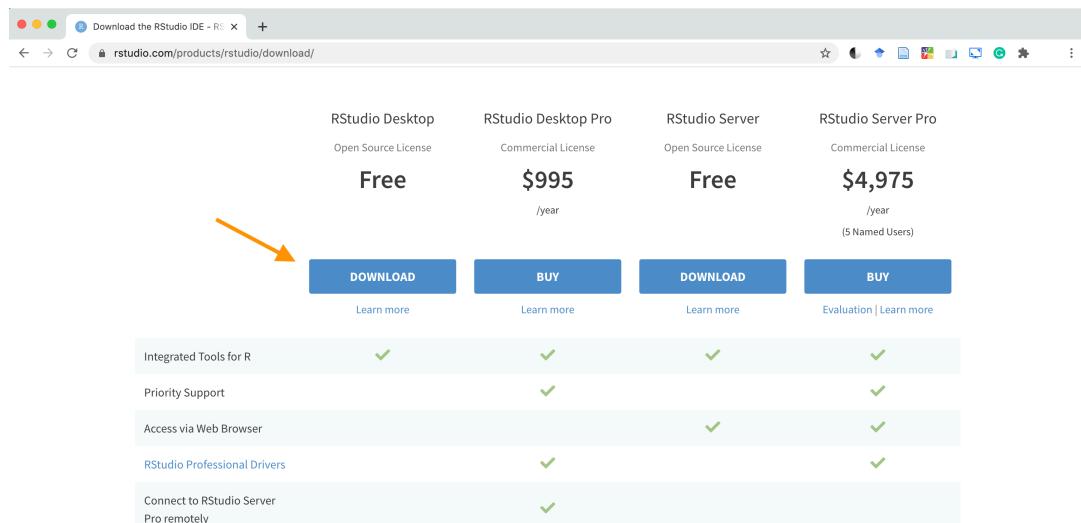
The screenshot shows the 'R for Mac OS X - Development' page. On the left, there's a sidebar with links like 'CRAN Mirrors', 'What's new?', 'Task Views', 'Search', 'About R', 'R Homepage', and 'The R Journal'. The main content area has a heading 'Development Tools and Libraries'. It contains a note: 'CRAN R 4.0.0 builds and higher no longer use any custom compilers and thus this directory is no longer relevant. We now use Apple Xcode 10.1 and GNU Fortran 8.2 from <https://github.com/xcoudert/gfortran-for-macOS/releases>. For more details on compiling R, please see also <https://mac.R-project.org/tools/>'.

1.2 Installare R Studio

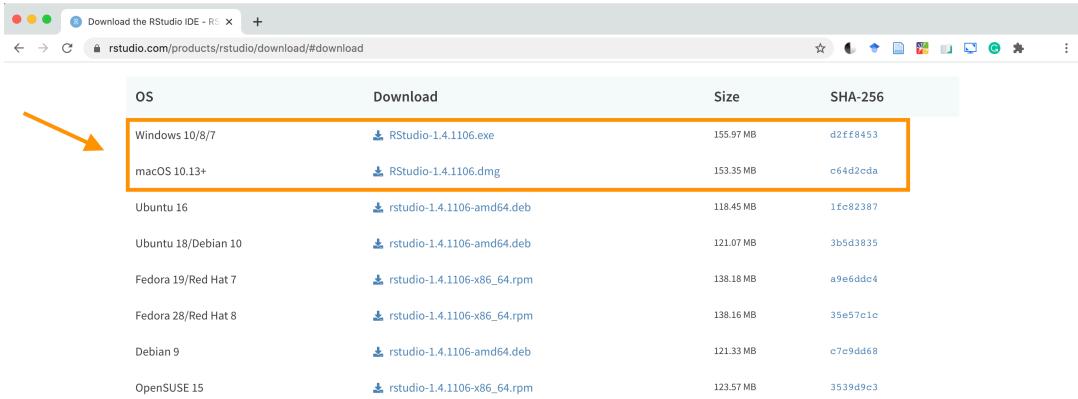
1. Accedere al sito <https://rstudio.com>
2. Selezionare la voce **DOWNLOAD IT NOW**



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	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	c7c9dd68
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.

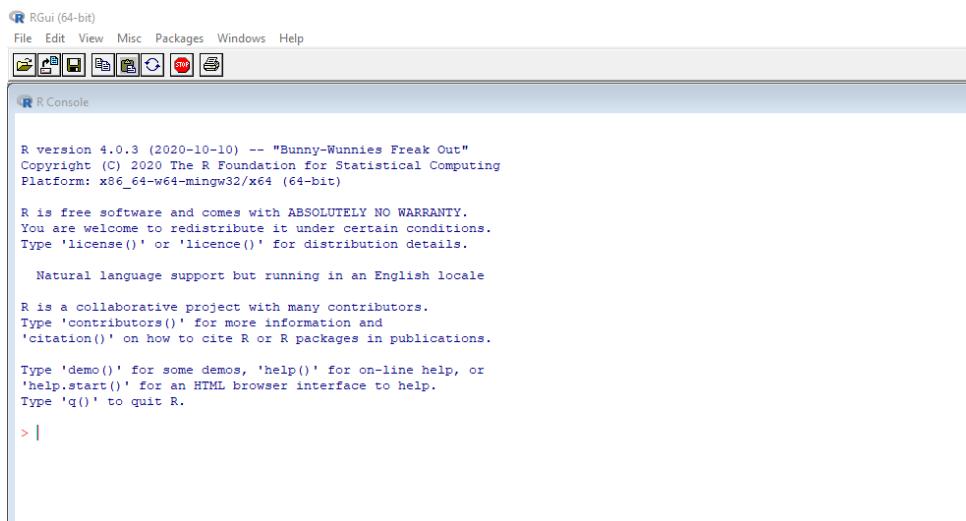


Figure 2.1: La console di R, solo per veri intenditori

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):

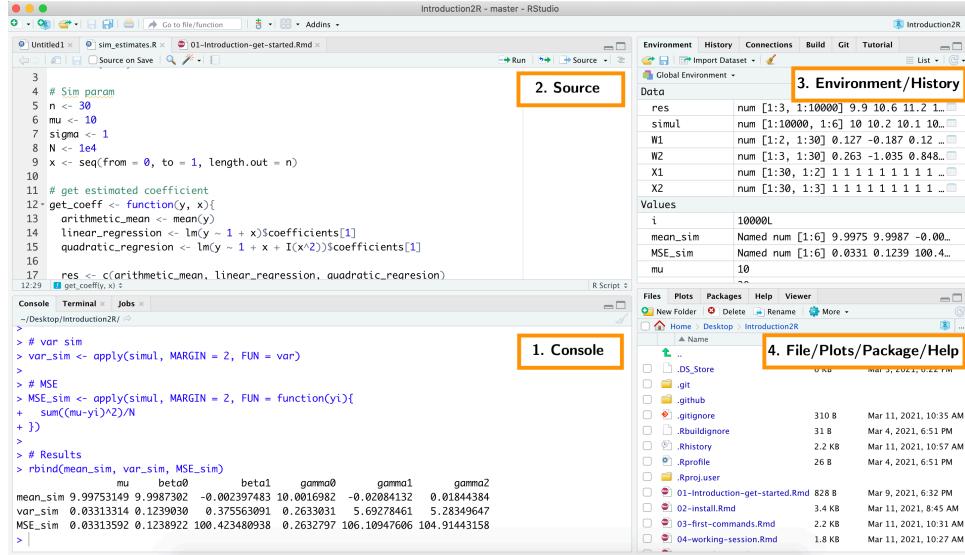


Figure 2.2: Interfaccia utente di Rstudio con i suoi 4 pannelli

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 i nostri comandi, 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).

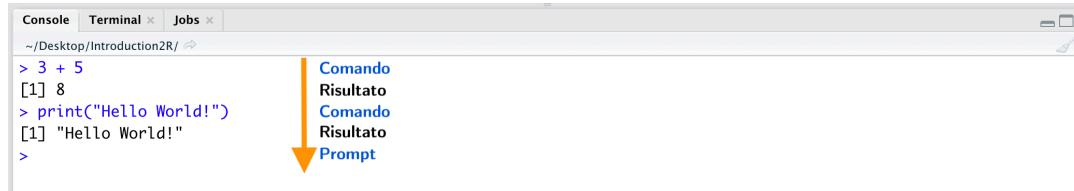
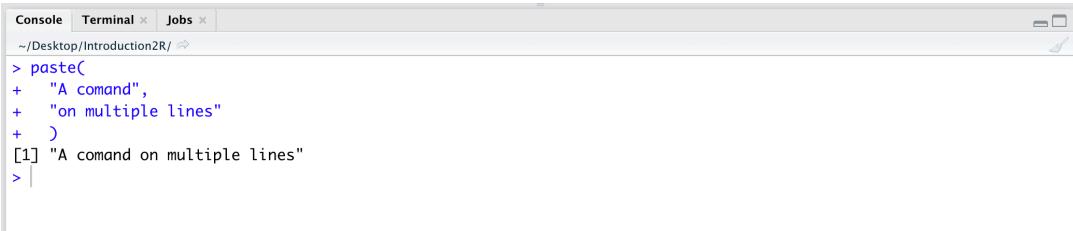


Figure 2.3: Esecuzione di comandi direttamente nella console

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.

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

A screenshot of the RStudio interface, specifically the Console tab. The title bar shows "Console", "Terminal", and "Jobs". The current working directory is "~/Desktop/Introduction2R/". In the console window, the following R code is shown:

```
> pasteC
+ "A comand",
+ "on multiple lines"
+ )
[1] "A comand on multiple lines"
>
```

The code consists of three lines starting with a plus sign (+), which are concatenated into a single string "[1] "A comand on multiple lines"".

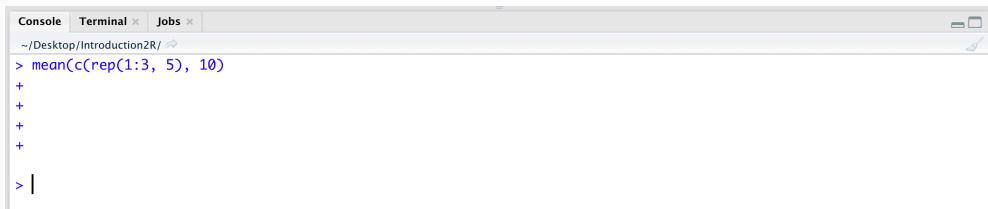
Figure 2.4: Esecuzione di un comando su più righe

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'apprire 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.

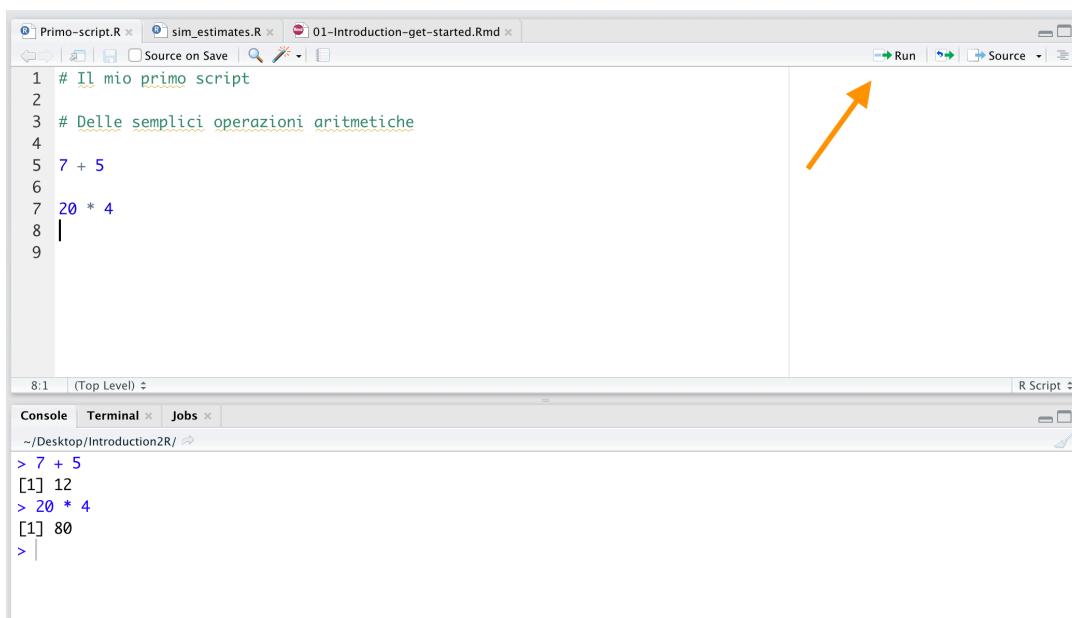


Figure 2.5: Esecuzione di un comando da script premi ‘command + invio’ (MacOs)/ ‘Ctrl+R’ (Windows) o premi il tasto indicato dalla freccia

Tip-Box: Commenti

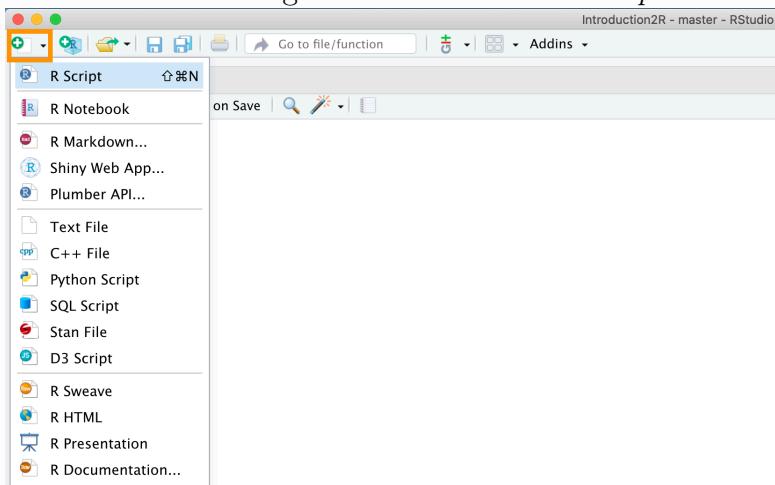
Se hai guardato con attenzione lo script rappresentato in Figura 2.5, potresti aver notato delle righe di testo verde precedute dal simbolo `#`. Questo simbolo può essere utilizzato per inserire dei *commenti* all'interno dello script. R ignorerà qualsiasi commento ed eseguirà soltanto le parti di codici.

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.

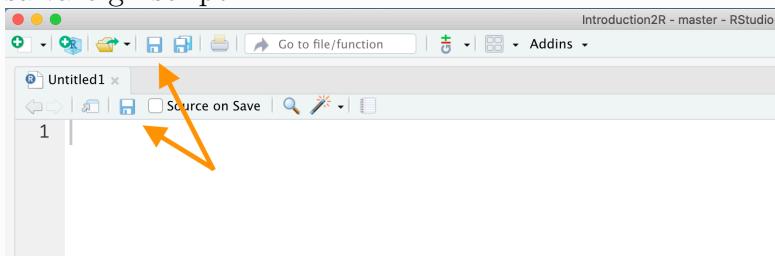


Approfondimento: Creare e Salvare uno Script

Per creare un nuovo script è sufficiente premere il pulsante in alto a sinistra come mostrato in Figura e selezionare “*R Script*”.



Un nuovo script senza nome verrà creato. Per salvare lo script premere l'icona del floppy e indicare il nome. Ricorda di usare l'estensione “**.R**” per salvare gli script.



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.

Data	
res	num [1:3, 1:10000] 10.15 10.36 10.1 9.6...
simul	num [1:10000, 1:6] 10.02 10.27 9.63 9.9...
W1	num [1:2, 1:30] 0.127 -0.187 0.12 -0.17...
X1	num [1:30, 1:2] 1 1 1 1 1 1 1 1 1 1 1 ...

Values	
i	10000L
mean_sim	Named num [1:6] 10.00019 9.99954 0.00131 ...
MSE_sim	Named num [1:6] 0.0334 0.1258 100.3463 0...

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

Name	Size	Modified
..		
.DS_Store	6 KB	Mar 3, 2021, 6:22 PM
.git		
.github		
.gitignore	310 B	Mar 11, 2021, 10:35 AM
.Rbuildignore	31 B	Mar 4, 2021, 6:51 PM
.Rhistory	3.6 KB	Mar 11, 2021, 11:03 AM
.Rprofile	26 B	Mar 4, 2021, 6:51 PM

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

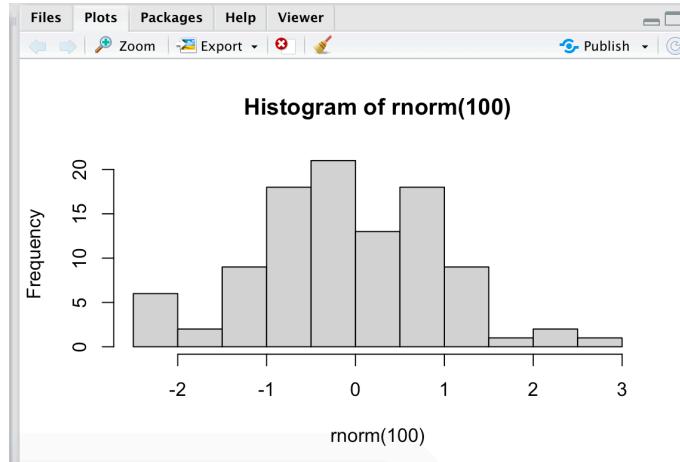


Figure 2.8: *Plots* - presentazione dei grafici

- **Plots:** pannello i cui vengono prodotti i grafici e che è possibile esportare cliccando *Export*.
- **Packages:** elenco dei pacchetti di R (questo argomento verrà approfondito nel Capitolo TODO).

Name	Description	Version	Lockfile	Source
Project Library				
<input type="checkbox"/> askpass	Safe Password Entry for R, Git, and SSH	1.1		@ X
<input type="checkbox"/> assertthat	Easy Pre and Post Assertions	0.2.1		@ X
<input type="checkbox"/> backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.9		@ X
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3	0.1-3	Reposit @ X
<input type="checkbox"/> BH	Boost C++ Header Files	1.72.0-3		@ X
<input type="checkbox"/> bookdown	Authoring Books and Technical Documents with R Markdown	0.21	0.21	Reposit @ X
<input type="checkbox"/> brew	Templating Framework for Report Generation	1.0-6		@ X
<input type="checkbox"/> callr	Call R from R	3.4.4		@ X
<input type="checkbox"/> cli	Helpers for Developing Command Line Interfaces	2.0.2		@ X
<input type="checkbox"/> clipr	Read and Write from the System Clipboard	0.7.0		@ X

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

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

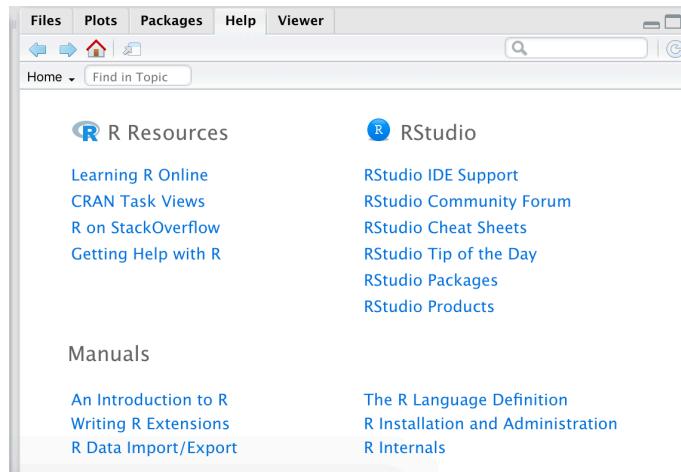
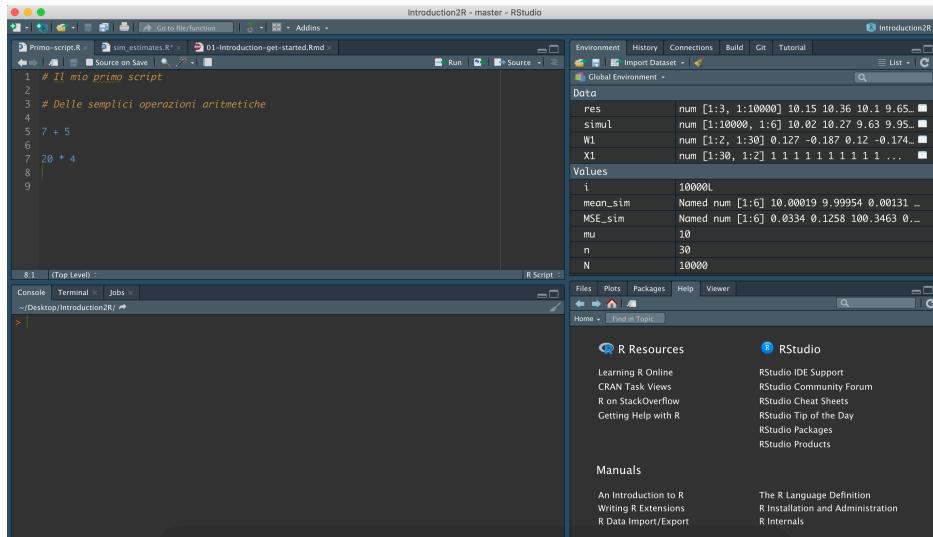


Figure 2.10: *Help* - documentazione di R

Tip-Box: Personalizza tema e layout

RStudio permette un ampio grado di personalizzazione dell'interfaccia 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 strumento 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 poiché si incontrano per la prima volta molti comandi e concetti di programmazione. Tuttavia, una volta familiarizzato con gli appetiti 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.

💡 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:

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>

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 (soluzioni):

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

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

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

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

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

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 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 rivelano 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”.

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>

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



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. 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	$x \setminus \& y$
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- Nel caso della **disgiunzione inclusiva logica \mid** , 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 ' \mid '

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., \wedge , $*$, $/$, $+$, $-$, etc.)
2. operatori relazionali (e.g., $<$, $>$, \leq , \geq , $==$, $!=$)
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 comando 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) ma 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"
```

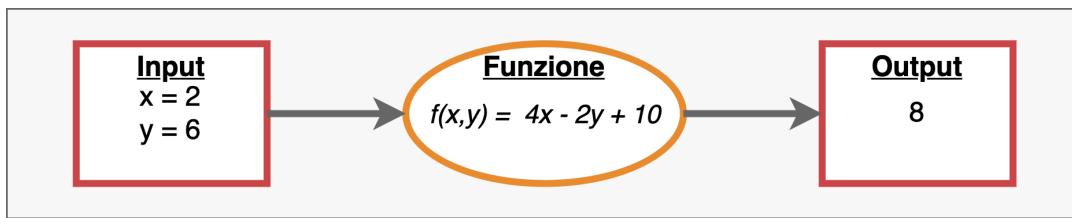
In R abbiamo inoltre differenti tipologie di oggetti, ovvero diverse strutture in cui possono essere organizzati i dati:

- **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 tabelle 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 produrrenno 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 virgolet. 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.

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

# inverto 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

seq {base}
R Documentation

Sequence Generation

Description

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.

Usage

```
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)
```

Arguments

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

Details

Numerical inputs should all be [finite](#) (that is, not infinite, [NaN](#) or [NA](#)).

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

- **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
- **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 ed 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

Ricerca per Parola

Quando non si conosce esattamente il nome di una funzione o si vuole cercare tutte le funzioni e pagine che includono una certa parola, è possibile utilizzare il comando `??<parola>` oppure `help.search(<parola>)`.

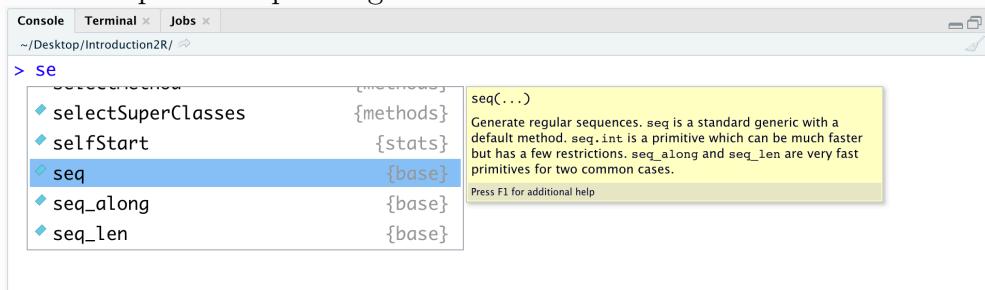
R eseguirà una ricerca tra tutta la documentazione disponibile e fornirà un elenco delle pagine che contengono la parola desiderata nel titolo o tra le keywords.



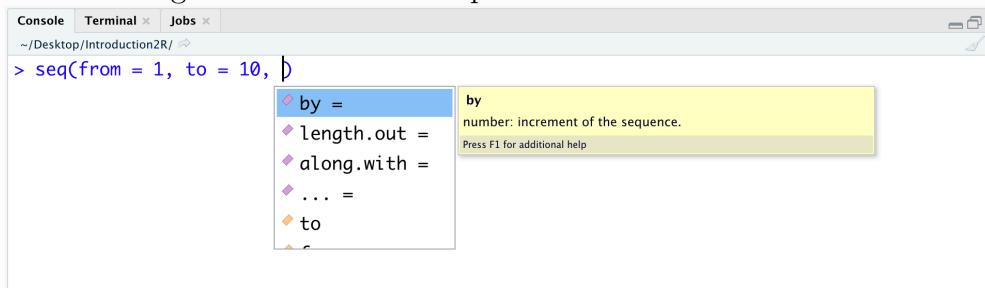
Trick-Box: Autocompletamento with 'Tab'

La natura dei programmatori 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.

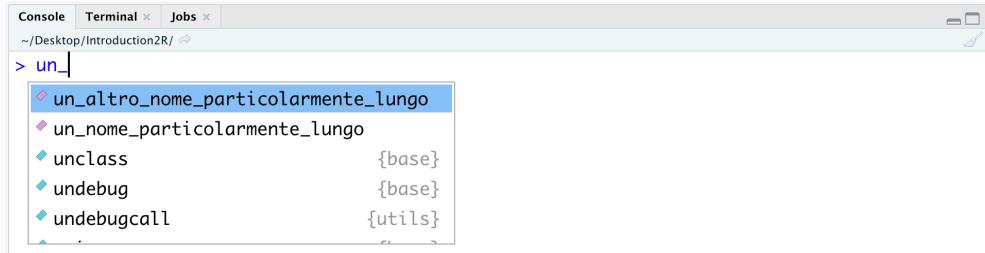
In Rstudio, i suggerimenti compaino automaticamente durante la scrittura di un comando oppure possono essere richiamati premendo il tasto **Tab** in alto a sinistra della tastiera (). Comparirà una finestra con possibili soluzioni di autocompletamento del nome della funzione. Utilizzando le frecce della 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

Ambiente di Lavoro

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

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

The screenshot shows the RStudio interface with the 'Environment' tab selected in the top navigation bar. Below the tabs, there are buttons for 'Import Dataset' and 'Global Environment'. A search bar is also present. The main area is divided into two sections: 'Data' and 'Values'. The 'Data' section lists four objects: 'res', 'simul', 'W1', and 'X1', each with a preview of its contents. The 'Values' section lists three objects: 'i', 'mean_sim', and 'MSE_sim', each with a preview of its contents. The entire interface has a light gray background with dark gray borders for the panels.

Data	
res	num [1:3, 1:10000] 10.15 10.36 10.1 9.6...
simul	num [1:10000, 1:6] 10.02 10.27 9.63 9.9...
W1	num [1:2, 1:30] 0.127 -0.187 0.12 -0.17...
X1	num [1:30, 1:2] 1 1 1 1 1 1 1 1 1 1 1 ...

Values	
i	10000L
mean_sim	Named num [1:6] 10.00019 9.99954 0.00131 ...
MSE_sim	Named num [1:6] 0.0334 0.1258 100.3463 0...

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

All'inizio della sessione di lavoro il nostro Environment sarà vuoto (vedi Figura ??). Il comando `ls()` non restituirà alcun oggetto ma per indicare l'assenza di oggetti userà la risposta `character(0)`, ovvero un vettore di tipo caratteri di lunghezza zero (vedi Capitolo TODO).



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

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

5.1.1 Aggiungere Oggetti all'Environment

Una volta creati degli oggetti, questi saranno presenti nel nostro Environment e il comando `ls()` restituirà un vettore di caratteri in cui vengono elencati tutti i loro nomi.

```
# Creo oggetti
x <- c(2,4,6,8)
y <- 27
word <- "Hello Word!"

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

Nel pannello in alto a destra (vedi Figura 5.3), possiamo trovare un elenco degli oggetti attualmente presenti nel nostro Environment. Insieme al nome vengono riportate anche alcune utili informazioni a seconda del tipo di oggetto. Vediamo come nel nostro esempio, nel caso di variabili con un singolo valore (e.g., `word` e `y`) vengano presentati direttamente gli stessi valori. Mentre, nel caso di vettori (e.g., `x`) vengano fornite anche informazioni riguardanti la tipologia di vettore e la sua dimensione (vedi Capitolo TODO), nell'esempio abbiamo un vettore numerico (`num`) di 4 elementi (`[1:4]`).

Values	
word	"Hello Word!"
x	num [1:4] 2 4 6 8
y	27

Figure 5.3: *Environment* contenente gli oggetti creati

5.1.2 Rimuovere Oggetti dall'Environment

Per rimuovere un oggetto dal proprio environment è possibile utilizzare il comando `remove()` oppure la sua abbreviazione `rm()`, indicando tra parentesi il nome dell'oggetto che si intende

rimuovere. E' possibile indicare più di un oggetto separando i loro nomi con la virgola.

```
# Rimuovo un oggetto
rm(word)
ls()
## [1] "x" "y"

# Rimuovo più oggetti contemporaneamente
rm(x,y)
ls()
## character(0)
```



Trick-Box: rm(list=ls())

Qualora fosse necessario eliminare tutti gli oggetti attualmente presenti nel nostro ambiente di lavoro è possibile ricorrere alla formula `rm(list=ls())`. In questo modo si avrà la certezza di pulire l'ambiente da ogni oggetto e di ripristinarlo alle condizioni iniziali della sessione.



Approfondimento: Mantenere Ordinato l'Environment

Avere cura di mantenre il proprio Environment ordinato ed essere consapevoli degli oggetti attualmente presenti è importante. Questo ci permette di evitare di compiere due errori comuni.

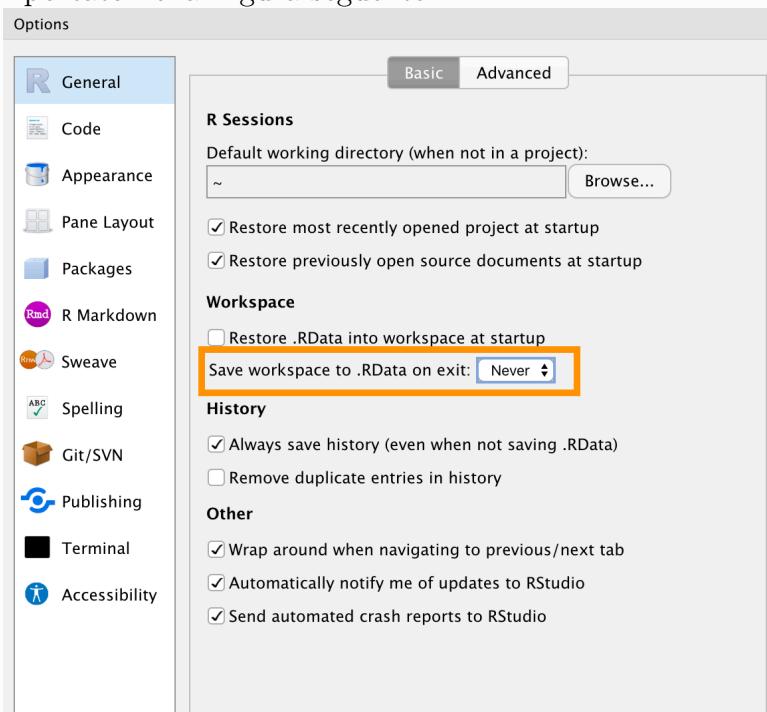
- **Utilizzare oggetti non ancora creati.** In questo caso l'errore è facilmente individuabile dato che sarà lo stesso R ad avvisarci che “*object ‘not found’*”. In questo caso dovremmo semplicemente eseguire il comando per creare l'oggetto richiesto.

```
oggetto_non_esistente
```

```
## Error in eval(expr, envir, enclos): object 'oggetto_non_esistente' not found
```

- **Utilizzare oggetti con “vecchi” valori.** Se non si ha cura di mantenere ordinato il proprio ambiente di lavoro potrebbe accadere che diversi oggetti vengano creati durante successive sessioni di lavoro. A questo punto si corre il rischio di perdere il controllo rispetto al vero contenuto degli oggetti e potremmo quindi utilizzare degli oggetti pensando che contengano un certo valore, quando invece si riferiscono a tutt'altro. Questo comporta che qualsiasi nostro risultato perda di significato. Bisogna prestare molta attenzione perché R non potrà avvisarci di questo errore (per lui sono solo numeri), siamo noi che dobbiamo essere consapevoli del fatto che i comandi eseguiti abbiano senso oppure no.

Per mantenere un Environment ordinato vi consigliamo innanzitutto di non salvare automaticamente il vostro *workspace* quando terminate una sessione di lavoro. È possibile settare tale opzione nelle impostazioni generali di R selezionando “*Never*” alla voce “*save workspace to .RData on exit*” come riportato nella Figura seguente.



5.2 Working Directory

Il concetto di *working directory* è molto importante ma spesso poco conosciuto. La *working directory* è la posizione all'interno del computer in cui ci troviamo durante la nostra sessione di lavoro e da cui eseguiamo i nostri comandi.

5.2.1 Organizzazione Computer

L'idea intuitiva che abbiamo comunemente del funzionamento del computer è fuorviante. Spesso si pensa che il Desktop rispecchi l'organizzazione del nostro intero computer e che tutte le azioni siano gestite attraverso l'interfaccia punta-e-clicca a cui ormai siamo abituati dai moderni sistemi operativi.

Senza entrare nel dettaglio, è più corretto pensare all'organizzazione del computer come ad un insieme di cartelle e sottocartelle che contengono tutti i nostri file e al funzionamento del computer come ad un insieme di processi (o comandi) che vengono eseguiti. Gli stessi programmi che installiamo non sono altro che delle cartelle in cui sono contenuti tutti gli script che determinano il loro funzionamento. Anche il Desktop non è altro che una semplice cartella mentre quello che vediamo noi è un programma definito dal sistema operativo che visualizza il contenuto di quella cartella sul nostro schermo e ci permette di interfacciarsi con il mouse.

Tutto quello che è presente nel nostro computer, compresi i nostri file, i programmi e lo stesso sistema operativo in uso, tutto è organizzato in un articolato sistema di cartelle e sottocartelle. Approssimativamente possiamo pensare all'organizzazione del nostro computer in modo simile alla Figura 5.4 (da: https://en.wikipedia.org/wiki/Operating_system).



Figure 5.4: Organizzazione Computer (da Wikipedia vedi link nel testo)

Ai livelli più bassi troviamo tutti i file di sistema ai quali gli utenti possono accedere solo con speciali autorizzazioni. Al livello superiore troviamo tutte i file riguardanti i programmi e applicazioni installati che in genere sono utilizzabili da più utenti sullo stesso computer. Infine troviamo tutte le cartelle e file che riguardano lo specifico utente.

5.2.2 Absolute Path e Relative Path

Questo ampio preambolo riguardante l'organizzazione in cartelle e sottocartelle, ci serve perchè è la struttura che il computer utilizza per orientarsi tra tutti file quando esegue dei comandi attraverso un'interfaccia a riga di comando (e.g., R). Se vogliamo ad esempio caricare dei dati da uno specifico file in R devo fornire il *path* (o indirizzo) corretto che mi indichi esattamente la posizione del file all'interno della struttura di cartelle del computer. Ad esempio, immaginiamo di avere dei dati *My-data.Rda* salvato nella cartella *Introduction2R* nel proprio Desktop.

```
Desktop
|
|- Introduction2R
|   |
|   |- Dati
|   |   |- My-data.Rda
```

Per indicare la posizione del File potrei utilizzare un:

- **absolute path** - la posizione “*assoluta*” del file rispetto alla *root directory* del sistema ovvero la cartella principale dell'intero computer.

```
# Mac
"/Users/<username>/Desktop/Introduction2R/Dati/My-data.Rda"

# Windows Vista
"c:\Users\<username>\Desktop\Introduction2R\Dati\My-data.Rda"
```

- **relative path** - la posizione del file rispetto alla nostra attuale posizione nel computer da cui stiamo eseguendo il comando, ovvero rispetto alla **working directory** della nostra sessione di lavoro. In questo riprendendo il precedente esempio se la nostra working directory fosse la cartella *Desktop/Introduction2R* avremmo i seguenti relative path:

```
# Mac
"Dati/My-data.Rda"

# Windows Vista
"Dati\My-data.Rda"
```

Nota come sia preferibile l'utilizzo dei relative path poichè gli absolute path sono unici per il singolo computer di riferimento e non possono essere quindi utilizzati su altri computer.



Warning-Box: "Error: No such file or directory"

Qualora si utilizzasse un relative path per indicare la posizione di un file, è importante che la working directory attualmente in uso sia effettivamente quella prevista. Se ci trovassimo in una diversa cartella, ovviamente il “relative path” indicato non sarebbe più valido e R ci mostrerebbe un messaggio di errore.

Riprendendo l'esempio precedente, supponiamo che la nostra attuale working directory sia Desktop invece di Desktop/Introduction2R. Eseguendo il comando `load()` per caricare i dati utilizzando il relative path ora non più valido ottengo:

```
load("Dati/My-data.Rda")
## Warning in readChar(con, 5L, useBytes = TRUE): cannot open compressed file
## 'Dati/My-data.Rda', probable reason 'No such file or directory'
## Error in readChar(con, 5L, useBytes = TRUE): cannot open the connection
```

Il messaggio di errore mi indica che R non è stato in grado di trovare il file seguendo le mie indicazioni. E' come se chiedessi al computer di aprire il frigo ma attualmente si trovasse in camera, devo prima dargli le indicazioni per raggiungere la cucina altrimenti mi risponderebbe “*frigo non trovato*”. Risulta pertanto fondamentale essere sempre consapevoli di quale sia l'attuale working directory in cui si sta svolgendo la sessione di lavoro.

Ovviamente otterrei lo stesso errore anche usando un absolute path se questo contenesse degli errori.



Approfondimento: The Garden of Forking Paths

Come avrai notato dagli esempi precedenti, sia la struttura in cui vengono organizzati i file nel computer sia la sintassi utilizzata per indicare i path è differente in base al sistema operativo utilizzato.

Mac OS e Linux

- Il carattere utilizzato per separare la cartelle nella definizione del path è "/":

```
"Introduction2R/Dati/My-data.Rda"
```

- Iniziando il path con il carattere "/" si indica la root-directory:

```
"/Users/<username>/Desktop/Introduction2R/Dati/My-data.Rda"
```

- Iniziando il path con il carattere "~" si indica la cartella *home* dell'utente ovvero /Users/<username>/:

```
"~/Desktop/Introduction2R/Dati/My-data.Rda"
```

Windows

- Il carattere utilizzato per separare la cartelle nella definizione del path è "\":

```
"Introduction2R\Dados\My-data.Rda"
```

5.2.3 Working Directory in R

Vediamo ora i comandi utilizzati in R per valutare e cambiare la working directory nella propria sessione di lavoro.

Attuale Working Directory

In R è possibile valutare l'attuale working directory utilizzando il comando `getwd()` che restituirà l'absolute path dell'attuale posizione.

```
getwd()
## [1] "/Users/<username>/Desktop/Introduction2R"
```

In alternativa, l'attuale working directory è anche riportata in alto a sinistra della Console come mostrato in Figura 5.5.



Figure 5.5: Workig directory dell'attuale sessione di lavoro

Premendo la freccia al suo fianco il pannello *Files* in basso a destra sarà reindirizzato direttamente alla workig directory dell'attuale sessione di lavoro. In questo modo sarà facile navigare tra i file e cartelle presenti al suo interno (vedi Figura 5.6).

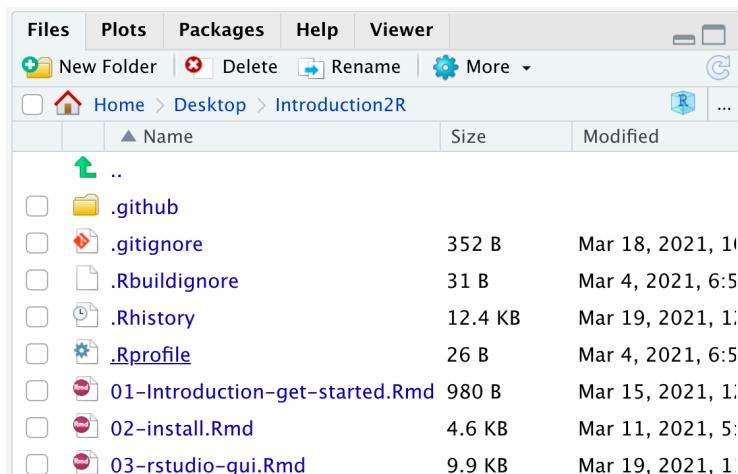


Figure 5.6: Workig directory dell'attuale sessione di lavoro

Cambiare Working Directory

Per cambiare la working directory è possibile utilizzare il comando `setwd()` indicando il path (absolute o relative) della nuova working directory. Nota come, nel caso in cui venga indicato un relative path, questo dovrà indicare la posizione della nuova working directory rispetto alla vecchia working directory.

```
getwd()
## [1] "/Users/<username>/Desktop/Introduction2R"

setwd("Dati/")
```

```
getwd()
## [1] "/Users/<username>/Desktop/Introduction2R/Dati"
```

In alternativa è possibile selezionare l'opzione “*Choose Directory*” dal menù “*Session*” > “*Set Working Directory*” come mostrato in Figura 5.7. Verrà quindi richiesto di selezionare la working directory desiderata e preme “*Open*”.

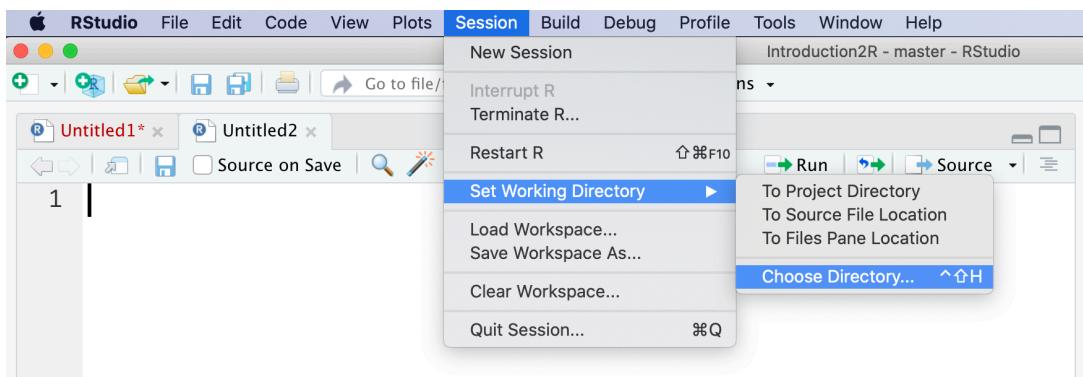
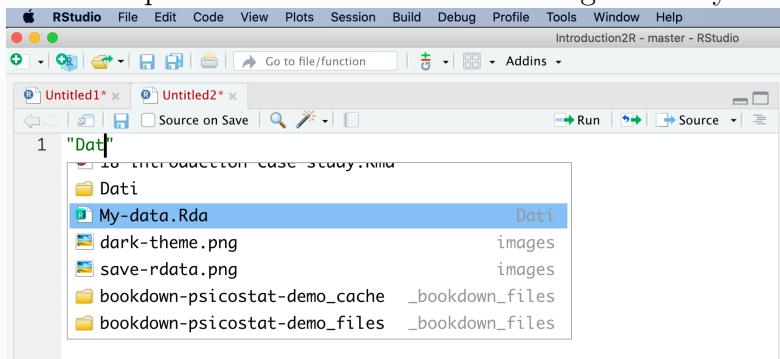


Figure 5.7: Definire la working directory



Trick-Box: Show me the Path

Nota come sia possibile nel digitare il path sfruttare l'autocompletamento. All'interno delle virgolette "" premi il tasto Tab per visualizzare i suggerimenti dei path relativi alla attuale working directory.



E' possibile inoltre utilizzare i caratteri speciali "./" e "../" per indicare rispettivamente l'attuale working directory e la cartella del livello superiore (i.e., *parent folder*) che include l'attuale working directory. "../" ci permette quindi di navigare a ritroso dalla nostra attuale posizione tra le cartelle del computer.

```
getwd()
## [1] "/Users/<username>/Desktop/Introduction2R"

setwd("../")

getwd()
## [1] "/Users/<username>/Desktop/"
```

5.3 R-packages

Uno dei grandi punti di forza di R è quella di poter estendere le proprie funzioni di base in modo semplice ed intuitivo utilizzando nuovi pacchetti. Al momento esistono oltre **17'000** pacchetti disponibili gratuitamente sul CRAN (la repository ufficiale di R). Questi pacchetti sono stati sviluppati dall'immensa community di R per svolgere ogni sorta di compito. Si potrebbe dire quindi che in R ogni cosa sia possibile, basta trovare il giusto pacchetto (oppure crearlo!).

Quando abbiamo installato R in automatico sono stati installati una serie di pacchetti che costituiscono la **system library**, ovvero tutti quei pacchetti di base che permettono il fuzionamento di R. Tuttavia, gli altri pacchetti non sono disponibili da subito. Per utilizzare le funzioni di altri pacchetti, è necessario seguire una procedura in due step come rappresentato in Figura 5.8:

1. **Scaricare ed installare i pacchetti sul nostro computer.** I pacchetti sono disponibili

gratuitamente online nella repository del CRAN, una sorta di archivio. Vengono quindi scaricati ed installati nella nostra *library*, ovvero la raccolta di tutti i pacchetti di R disponibili sul nostro computer.

- Caricare il pacchetto nella sessione di lavoro.** Anche se il pacchetto è installato nella nostra library non siamo ancora pronti per utilizzare le sue funzioni. Sarà necessario prima caricare il pacchetto nella nostra sessione di lavoro. Solo ora le funzioni del pacchetto saranno effettivamente disponibili per essere usate.

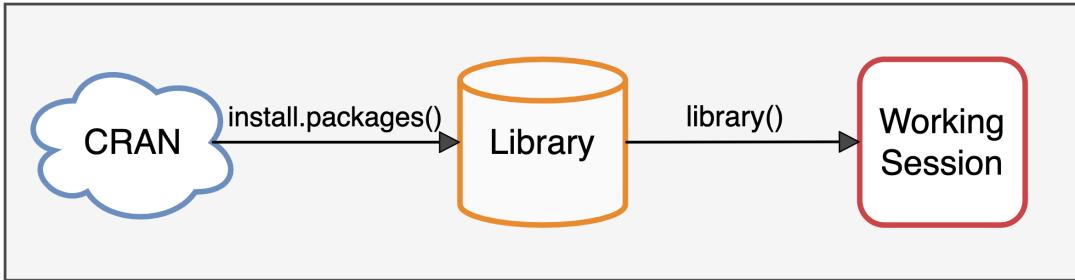


Figure 5.8: Utilizzare i pacchetti in R

Questo procedimento in due step potrebbe sembrare poco intuitivo. “Perchè dover caricare qualcosa che è già installato?” La risposta è molto semplice ci serve per mantenere efficiente e sotto controllo la nostra sessione di lavoro. Infatti non avremo mai bisogno di tutti i pacchetti installati ma a seconda dei compiti da eseguire utilizzeremo di volta in volta solo alcuni pacchetti specifici. Se tutti i pacchetti fossero caricati automaticamente ogni volta sarebbe un inutile spreco di memoria e si creerebbero facilmente dei conflitti. Ovvero, alcune funzioni di diversi pacchetti potrebbero avere lo stesso nome ma scopi diversi. Sarebbe quindi molto facile ottenere errori o comunque risultati non validi.

Vediamo ora come eseguire queste operazioni in R.

5.3.1 `install.packages()`

Per installare dei pacchetti dal CRAN nella nostra library è possibile eseguire il comando `install.packages()` indicando tra parentesi il nome del pacchetto desiderato.

```

# Un ottimo pacchetto per le analisi statistiche di John Fox
# un grandissimo statistico...per gli amici Jonny la volpe ;
install.packages("car")
  
```

In alternativa è possibile utilizzare il pulsante “Install” nella barra in alto a sinistra del pannello Packages (vedi Figura 5.9), indicando successivamente il nome del pacchetto desiderato.

Nota come installare un pacchetto potrebbe comportare l’installazione di più pacchetti. Questo perchè verranno automaticamente installate anche le *dependencies* del pacchetto, ovvero, tutti i pacchetti usati internamente dal pacchetto di interesse che quindi necessari per il suo corretto funzionamento (come in un gioco di matrioske).

Una volta installato il pacchetto, questo comparirà nella library ovvero la lista dei pacchetti disponibili mostrata nel pannello Packages (vedi Figura 5.10).

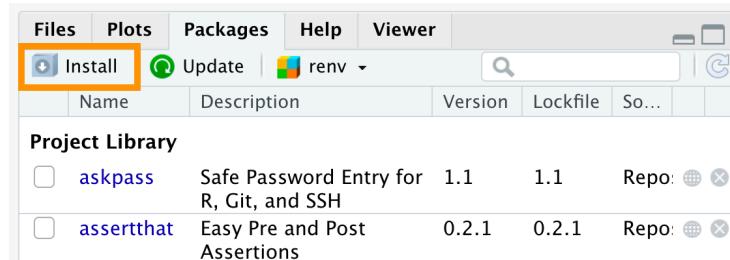


Figure 5.9: Installare paccektti tramite interfacci RStudio

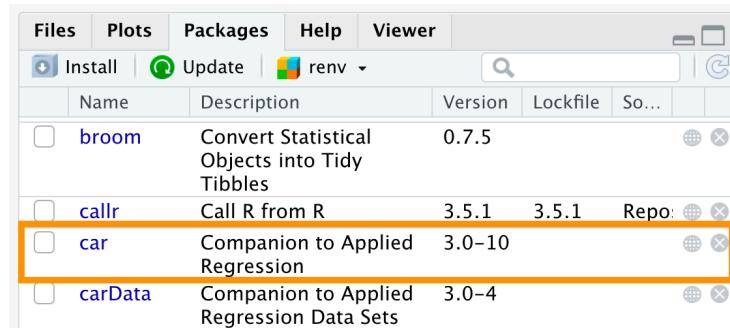


Figure 5.10: Il pacchetto car è ora disponibile nella library

5.3.2 library()

Per utilizzare le funzioni di un pacchetto già presente nella nostra library, dobbiamo ora caricarlo nella nostra sessione di lavoro. Per fare ciò, possiamo utilizzare il comando `library()` indicando tra parentesi il nome del pacchetto richiesto.

```
library(car)
```

In alternativa è possibile spuntare il riquadro alla sinistra del nome del pacchetto dal pannello Packages come mostrato in Figura 5.11. Nota tuttavia come questa procedura sia sconsigliata. Infatti, ogni azione punta-e-clicca dovrebbe essere eseguita ad ogni sessione mentre l'utilizzo di comandi inclusi nello script garantisce la loro esecuzione automatica.

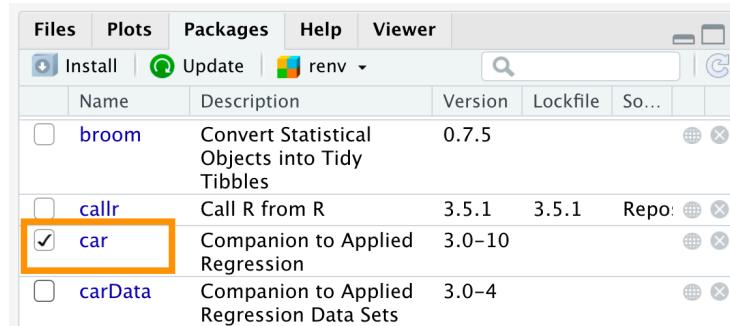


Figure 5.11: Caricare un pacchetto nella sessione di lavoro

Ora siamo finalmente pronti per utilizzare le funzioni del pacchetto nella nostra sessione di lavoro.

Trick-Box: package::function()

Esiste un piccolo trucco per utilizzare la funzione di uno specifico pacchetto senza dover caricare il pacchetto nella propria sessione. Per fare questo è possibile usare la sintassi:

```
<nome-pacchetto>::<nome-funzione>()
```

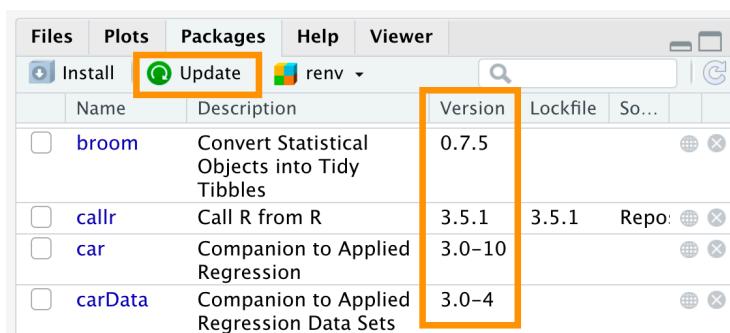
```
# Esempio con la funzione Anova del pacchetto car
car::Anova()
```

L'utilizzo dei :: ci permette di richiamare direttamente la funzione desiderata. La differenza tra l'uso di `library()` e l'uso di :: riguarda aspetti abbastanza avanzati di R (per un approfondimento vedi <https://r-pkgs.org/namespace.html>). In estrema sintesi, possiamo dire che in alcuni casi è preferibile non caricare un'intero pacchetto se di questo abbiamo bisogno di un'unica funzione.

5.3.3 Aggiornare e Rimuovere Pacchetti

Anche i pacchetti come ogni altro software vengono aggiornati nel corso del tempo fornendo nuove funzionalità e risolvendo eventuali problemi. Per aggiornare i pacchetti alla versione più recente è possibile eseguire il comando `update.packages()` senza inidare nulla tra le parentesi.

In alternativa è possibile premere il pulsante “Update” nella barra in alto a sinistra del pannello Packages (vedi Figura 5.12), indicando successivamente i pacchetti che si desidera aggiornare. Nota come nella lista dei pacchetti venga riportata l'attuale versione alla voce “Version”.



	Name	Description	Version	Lockfile	So...
<input type="checkbox"/>	broom	Convert Statistical Objects into Tidy Tibbles	0.7.5		
<input type="checkbox"/>	callr	Call R from R	3.5.1	3.5.1	Repo:
<input type="checkbox"/>	car	Companion to Applied Regression	3.0-10		
<input type="checkbox"/>	carData	Companion to Applied Regression Data Sets	3.0-4		

Figure 5.12: Aggiornare i pacchetti

Nel caso in cui si vogli invece rimuover uno specifico pacchetto, è possibile eseguire il comando `remove.packages()` indicando tra le parentesi il nome del pacchetto.

In alternativa è possibile premere il pulsante **x** alla destra del pacchetto nel pannello Packages come mostrato in Figura 5.13.

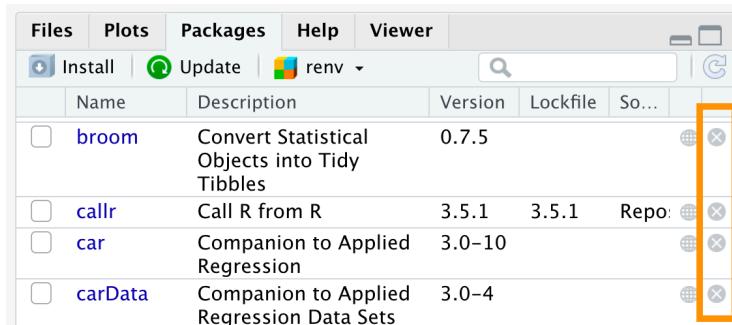


Figure 5.13: Rimuovere un pacchetto

5.3.4 Documentazione Pacchetti

Ogni pacchetto include la documentazione delle proprie funzioni e delle *vignette* ovvero dei brevi tutorial che mostrano degli esempi di applicazione e utilizzo del pacchetto.

- **Documentazione funzione** - Per accedere alla documentazione di una funzione è sufficiente utilizzare il comando `?<nome-funzione>` oppure `help(<nome-funzione>)`. Ricorda è necessario avere prima caricato il pacchetto altrimenti la funzione non risulta ancora disponibile. In alternativa si potrebbe estendere la ricerca utilizzando il comando `??`.
- **Vignette** - Per ottenere la lista di tutte le vignette di un determinato pacchetto è possibile utilizzare il comando `browseVignettes(package = <nome-pacchetto>)`. Mentre, per accedere ad una specifica vignetta, si utilizza il comando `vignette("<name-vignetta>")`.
- **Documentazione intero pacchetto** - Premendo il nome del pacchetto dal pannello Packages in basso a destra, è possibile accedere alla lista di tutte le informazioni relative al pacchetto come riportato in Figura 5.14. Vengono prima forniti i link per le vignette ed altri file relativi alle caratteristiche del pacchetto. Successivamente sono presentate in ordine alfabetico tutte le funzioni.

Ricordate tuttavia che in ogni caso la più grande risorsa di informazioni è come sempre google. Spesso i pacchetti più importanti hanno addirittura un proprio sito in cui raccolgono molto materiale utile. Ma comunque in ogni caso in internet sono sempre disponibili moltissimi tutorial ed esempi.

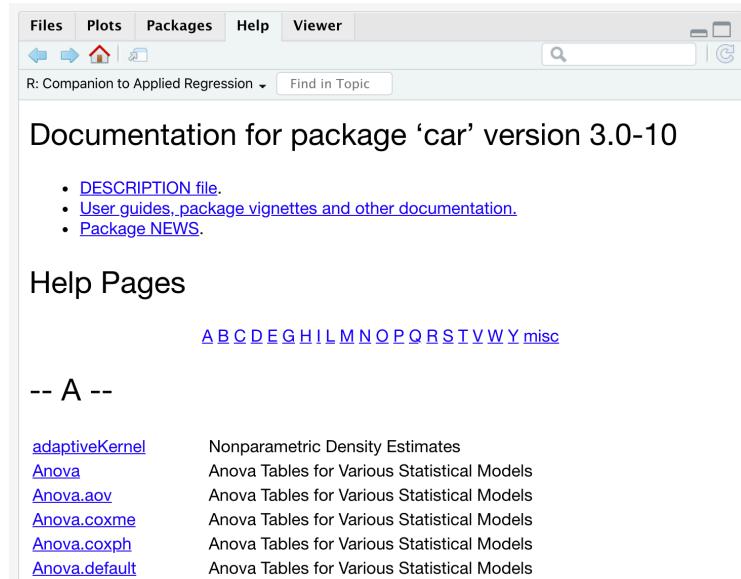


Figure 5.14: Documentazione del pacchetto car



Approfondimento: Github

Il CRAN non è l'unica risorsa da cui è possibile installare dei pacchetti di R tuttavia è quella ufficiale e garantisce un certo standard e stabilità dei pacchetti presenti. In internet esistono molte altre repository che raccolgono pacchetti di R (e software in generale) tra cui una delle più popolari è certamente GitHub (<https://github.com/>).

Github viene utilizzato come piattaforma di sviluppo per molti pacchetti di R ed è quindi possibile trovarve le ultime versioni di sviluppo dei pacchetti con gli aggiornamenti più recenti o anche nuovi pacchetti non ancora disponibili sul CRAN. Va sottolineato tuttavia, come queste siano appunto delle versioni di sviluppo e quindi potrebbero presentare maggiori problemi. Inoltre per installare i pacchetti in questo modo, è richiesta l'installazione di **R tools** (vedi “*Approfondimento: R Tools*” nel Capitolo 1.1).

Per installare un pacchetto direttamente da Github è possibile utilizzare il comando `install_github()` del pacchetto `devtools`, indicando tra parentesi la l'url della repository desiderata.

```
install.packages("devtools")

# ggplot2 il miglior pacchetto per grafici
devtools::install_github("https://github.com/tidyverse/ggplot2")
```

Chapter 6

Sessione di Lavoro

In questo capitolo, discuteremo di alcuni aspetti generali delle sessioni di lavoro in R. Descriveremo delle buone abitudini riguardanti l'organizzazione degli scripts e l'uso degli *RStudio Projects* per essere ordinati ed efficaci nel proprio lavoro. Infine approfondiremo l'uso dei messaggi di R ed in particolare come comportarsi in caso di errori.

6.1 Organizzazione Script

Abbiamo visto che idealmente tutti i passaggi delle nostre analisi devono essere raccolti in modo ordinato all'interno di uno script. Eseguendo in ordine linea per linea i comandi, dovrebbe essere possibile svolgere tutte le analisi fino ad ottenere i risultati desiderati.

Vediamo ora una serie di buone regole per organizzare in modo ordinato il codice all'interno di uno script e facilitare la sua lettura.

6.1.1 Creare delle Sezioni

Per mantenere chiara l'organizzazione degli script e facilitare la sua comprensione, è utile suddividere il codice in sezioni dove vengono eseguiti i diversi step delle analisi. In RStudio è possibile creare una sezione aggiungendo al termine di una linea di commento i caratteri ##### o -----. Il testo del commento verrà considerato il titolo della sezione e comparirà una piccola freccia a lato del numero di riga. È possibile utilizzare a piacere i caratteri # o - per creare lo stile desiderato, l'importante è che la linea si concluda con almeno quattro caratteri identici.

```
# Sezione 1 #####
# Sezione 2 -----
#----  Sezione 3   ----
#####  Sezione non valida  --##
```

A titolo del tutto esemplificativo prendiamo in esempio la divisione in sezioni utilizzata nello script in Figura 6.1.

The screenshot shows the RStudio Source Editor window titled "Introduction2R - master - RStudio Source Editor". The file is named "Analisi-Tesi.R". The code is divided into several sections, each starting with a line of code ending in a hash symbol (#). The sections are:

- #===== Analisi Esperimento Tesi =====#
- # In questo script vengono analizzate i dati relativi alla tesi
- Settings -----
- rm(list = ls())
- setwd("~/Desktop/Analisi_Tesi/")
- library(tidyverse)
- library(lme4)
- library(car)
- library(effects)
- Caricare e Pulire i Dati -----
- # Codici relativi all'importazione e pulizia dei dati
- Codifica e Scoring dei Dati -----
- # Codici relativi alla codifica e scoring dei dati
- Analisi Descrittive -----
- # Codici relativi alle analisi descrittive
- Analisi Inferenziali -----
- # Codici relativi alle analisi inferenziali

The status bar at the bottom shows "17:1" and "R Script".

Figure 6.1: Esempio di suddivisione in sezioni di uno script

- **Titolo** - Un titolo esplicativo del contenuto dello script. E' possibile utilizzare altri caratteri all'interno dei commenti per creare l'effetto desiderato.
- **Introduzione** - Descrizione e utili informazioni che riguardano sia l'obbiettivo del lavoro che l'esecuzione del codice (e.g., dove sono disponibili i dati, eventuali specifiche tecniche). Potrebbe essere utile anche indicare l'autore e la data del lavoro.
- **Setting** - Sezione fondamentale in cui si predispone l'ambiente di lavoro. Le operazioni da svolgere sono:
 1. `rm(list = ls())` per pulire l'Environment da eventuali oggetti in modo da eseguire lo script partendo da un ambiente vuoto (vedi Capitolo 5.1).
 2. `setwd()` per settare la working directory per assicurarsi che i comandi siano eseguiti dalla corretta posizione nel nostro computer (vedi Capitolo 5.2).
 3. `library()` per caricare i pacchetti utilizzati nel corso delle analisi (vedi Capitolo 5.3).
- **Caricare e Pulire i Dati** - Generica sezione in cui eseguire l'importazione e pulizia dei dati.
- **Codifica e Scoring dei Dati** - Generica sezione in cui eseguire la codifica ed eventuale scoring dei dati.
- **Analisi Descrittive** - Generica sezione in cui eseguire le analisi descrittive.
- **Analisi Inferenziali** - Generica sezione in cui eseguire le analisi inferenziali

Oltre che a mantenere ordinato e chiaro il codice, suddividere il proprio script in sezioni ci permette anche di navigare facilmente tra le diverse parti del codice. Possiamo infatti sfruttare l'indice che automaticamente viene creato. L'indice è consultabile premendo il tasto in alto a destra dello script da cui successivamente selezionare la sezione desiderata (vedi Figura 6.2).



```

Introduction2R - master - RStudio Source Editor
Analisi-Tesi.R
Source on Save Run Source
1 #=====
2 === Analisi Esperimento Tesi ===#
3 =====#
4
5 # In questo script vengono analizzate i dati relativi alla tesi
6
7 #---- Settings ----
8
9 rm(list = ls())
10
11 setwd("~/Desktop/Analisi_Tesi/")
12
13 library(tidyverse)
14 library(lme4)

```

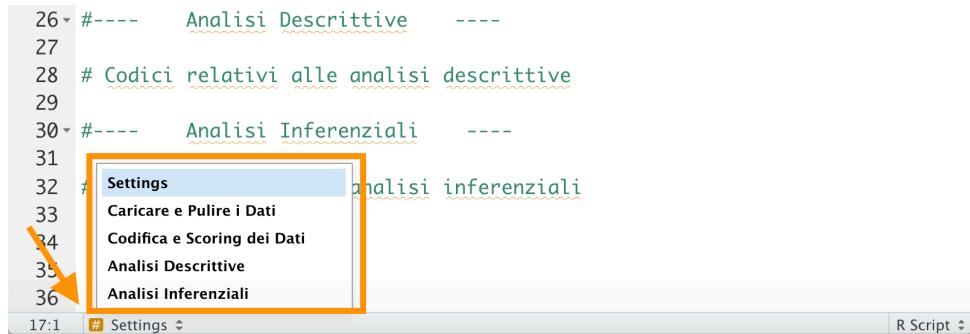
Figure 6.2: Indice in alto per navigazione sezioni

In alternativa, è possibile utilizzare il menù in basso a sinistra dello script (vedi Figura 6.3).

Infine, un altro vantaggio è quello di poter compattare o espandere le sezioni di codice all'interno dell'editor, utilizzando le frecce a lato del numero di riga (vedi Figura 6.4).

6.1.2 Sintassi

Elenchiamo qui altre buone norme nella scrittura del codice che ne facilitano la comprensione.



A screenshot of the RStudio interface showing an R script editor. A context menu is open over line 36 of the code, listing options: Settings, Caricare e Pulire i Dati, Codifica e Scoring dei Dati, Analisi Descrittive, and Analisi Inferenziali. The entire menu is highlighted with an orange border. An orange arrow points from the bottom left towards the menu.

```

26 #---- Analisi Descrittive ----
27
28 # Codici relativi alle analisi descrittive
29
30 #---- Analisi Inferenziali ----
31
32 #---- analisi inferenziali
33
34
35
36

```

Figure 6.3: Menù in basso per navigazione sezioni



A screenshot of the RStudio interface showing an R script editor. Lines 7 through 30 are highlighted with an orange border, indicating they are collapsed. Line 7 shows the expanded state with the word "Settings". Lines 18 and 22 show the collapsed state with the words "Caricare e Pulire i Dati" and "Codifica e Scoring dei Dati". Lines 24, 26, and 30 show the collapsed state with the words "# Codici relativi alla codifica e scoring dei dati", "# Analisi Descrittive", and "# Analisi Inferenziali".

```

1 ######
2 === Analisi Esperimento Tesi ===#
3 #####
4
5 # In questo script vengono analizzate i dati relativi alla test
6
7 #---- Settings
8 #---- Caricare e Pulire i Dati
22 #---- Codifica e Scoring dei Dati ----
23
24 # Codici relativi alla codifica e scoring dei dati
25
26 #---- Analisi Descrittive
30 #---- Analisi Inferenziali ----
31
32 # Codici relativi alle analisi inferenziali
33
34
35

```

Figure 6.4: Compattare ed espandere le sezioni di codice

Commenti

L'uso dei commenti è molto importante, ci permette di documentare le varie parti del codice e chiarire eventuali comandi difficili da capire. Tuttavia, non è necessario commentare ogni singola riga di codice ed anzi è meglio evitare di commentare laddove i comandi sono facilmente interpretabili semplicemente leggendo il codice.

La capacità di scrivere commenti utili ed evitare quelli rindondanti si impara con l'esperienza. In generale un commento non dovrebbe indicare “*che cosa*” ma piuttosto il “*perchè*” di quella parte di codice. Infatti il cosa è facilmente interpretabile dal codice stesso mentre il perchè potrebbe essere meno ovvio e soprattutto più utile per la comprensione dell'intero script. Ad esempio:

```
x <- 10 # assegno a x il valore 10
x <- 10 # definisco massimo numero risposte
```

Il primo commento è inutile poichè è facilmente comprensibile dal codice stesso, mentre il secondo commento è molto utile perché chiarisce il significato della variabile e mi faciliterà nella comprensione del codice.

Nomi Oggetti

Abbiamo visto nel Capitolo 4.1.2 le regole che discriminano nomi validi da nomi non validi e le convenzioni da seguire nella definizione di un nome. Ricordiamo qui le caratteristiche che un nome deve avere per facilitare la comprensione del codice. Il nome di un oggetto deve essere:

- **auto-descrittivo** - Dal solo nome dovrebbe essere possibile intuire il contenuto dell'oggetto. E' meglio quindi evitare nomi generici (quali x o y) ed utilizzare invece nomi che chiaramente descrivano il contenuto dell'oggetto.
- **della giusta lunghezza** - Non deve essere né troppo breve (evitare sigle incomprensibili) ma neppure troppo lunghi. In genere sono sufficienti 2 o 3 parole per descrivere chiaramente un oggetto.

E' inoltre importante essere **consistenti** nella scelta dello stile con cui si nominano le variabili. In genere è preferibile usare lo **snake_case** rispetto al **CamelCase**, ma la scelta è comunque libera. Tuttavia, una volta presa una decisione, è bene mantenerla per facilitare la comprensione del codice.

Esplicitare Argomenti

Abbiamo visto nel Capitolo 4.2.1 l'importanza di esplicitare il nome degli argomenti quando vengono utilizzati nelle funzioni. Specificando a che cosa si riferiscono i vari valori facilitiamo la lettura e la comprensione del codice. Ad esempio:

```
seq(0, 10, 2)
```

Potrebbe non essere chiaro se intendiamo una sequenza tra 0 e 10 di lunghezza 2 o a intervalli di 2. Specificando gli argomenti evitiamo incomprensioni e possibili errori.

```
seq(from = 0, to = 10, by = 2)
seq(from = 0, to = 10, length.out = 2)
```

Spazi, Indentazione ed allineamento

Al contrario di molti altri software, R non impone regole severe nell'utilizzo di spazi, indentazioni ed allineamenti ed in genere è molto permissivo per quanto riguarda la sintassi del codice. Tuttavia è importante ricordare che:

Good coding style is like using correct punctuation. You can manage without it, but it sure makes things easier to read. Hadley Wickham

Prendiamo ad esempio le seguenti linee di codice, che includono delle funzioni avanzate di R:

```
# Stile 1
k=10;if(k<5){x<-5:15}else{x<-seq(0,16,4)};y=7*2-12;mean(x/y)
## [1] 4

# Stile 2
k <- 10

if (k < 5){
  x <- 5:15
} else {
  x<-seq(from = 0, to = 16, by = 4)
}

y <- 7 * 2 - 12

mean(x / y)
## [1] 4
```

Come puoi notare otteniamo in entrambi i casi gli stessi risultati, per R non c'è alcuna differenza. Tuttavia, l'uso di spazi, una corretta indentazione ed un appropriato allineamento facilita la lettura e comprensione del codice.

In genere sono valide le seguenti norme:

- Aggiungi degli **spazi** intorno agli operatori (+, -, <-, etc.) per separargli dagli argomenti ad eccezione di :.

```
# Good
35 / 5 + 7
x <- 0:10

# Bad
35/5+7
x<-0 : 10
```

- Nelle funzioni aggiungi degli **spazi** intorno al simbolo = che separa il nome degli argomenti e il loro valore. Aggiungi uno spazio dopo ogni virgola ma non separare il nome della funzione dalla parentesi sinistra.

```
# Good
seq(from = 0, to = 10, by = 2)

# Bad
seq (from=0,to=10,by=2)
```

- Usa la corretta **indentazione** per i blocchi di codice posti all'interno delle parentesi graffe. Il livello di indentazione deve rispecchiare la struttura di annidamento del codice.

```
# Good
for (...) {      # loop più esterno
...
  for (...) {    # loop interno
...
    if (...) {    # istruzione condizionale
...
  }
}
}

# Bad
for (...) {      # loop più esterno
...
  for (...) {    # loop interno
...
    if (...) {    # istruzione condizionale
...
  }
}
}
```

- Allinea gli argomenti di una funzione se questi spaziano più righe.

```
# Good
data.frame(id = ...,
           name = ...,
           age = ...,
           sex = ...)

# Bad
data.frame(id = ..., name = ...,
           age = ..., sex = ...)
```



Approfondimento: Tutta una Questione di Stile

Potente trovare ulteriori norme e consigli riguardo lo stile nella scrittura del codice al seguente link <https://irudnyts.github.io/r-coding-style-guide/>

6.2 R projects

TODO

6.2.1 Più di uno script

- idea di organizzare in vari script, cartelle
- working directory tutto in funzione alla cartella

6.3 Messages, Warnings e Errors

R utilizza la console per comunicare con noi durante le nostre sessioni di lavoro. Oltre a fornirci i risultati dei nostri comandi, R ci segnala anche altre utili informazioni attraverso diverse tipologie di messaggi. In particolare abbiamo:

- **Messages:** dei semplici messaggi che ci possono aggiornare ad esempio sullo stato di avanzamento di un dato compito oppure fornire suggerimenti sull'uso di una determinata funzione o pacchetto (spesso vengono mostrati quando viene caricato un pacchetto).
- **Warnings:** questi messaggi sono utilizzati da R per dirci che c'è stato qualche cosa di strano che ha messo in allerta R. R ci avvisa che, sebbene il comando sia stato eseguito ed abbiamo ottenuto un risultato, ci sono stati dei comportamenti inusuali o magari eventuali correzioni apportate in automatico. Nel caso di warnings non ci dobbiamo allarmare, è importante controllare che i comandi siano corretti e che abbiano effettivamente ottenuto il risultato desiderato. Una volta sicuri dei risultati possiamo procedere tranquillamente.
- **Errors:** R ci avvisa di eventuali errori e problemi che non permettono di eseguire il comando. In questo caso non otterremo nessun risultato ma sarà necessario capire e risolvere il problema per poi rieseguire nuovamente il comando e procedere.

Notiamo quindi come non tutti i messaggi che R ci manda sono dei messaggi di errore. E' quindi importante non spaventarsi ma leggere con attenzione i messaggi, molte volte si tratta semplicemente di avvertimenti o suggerimenti.

Tuttavia gli errori rappresentano sempre il maggiore dei problemi perché non è possibile procedere nel lavoro senza averli prima risolti. E' importante ricordare che i messaggi di errore non sono delle critiche che R ci rivolge perché sbagliamo. Al contrario, sono delle richieste di aiuto fatte da R perché non sa come comportarsi. Per quanto super potente, R è un semplice programma che non può interpretare le nostre richieste ma si basa sull'uso dei comandi che seguono una rigida sintassi. A volte è sufficiente una virgola mancante o un carattere al posto di un numero per mandare in confusione R e richiedere il nostro intervento risolutore.

6.3.1 Risolvere gli Errori

Quando si approccia la scrittura di codice, anche molto semplice, la cosa che sicuramente capiterà più spesso sarà riscontrare messaggi di **errore** 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

Leggere con attenzione i messaggi di errore è molto importante. 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"**.

Tuttavia, in altre situazioni i messaggi potrebbero non essere altrettanto chiari. Seppur esplicito R è anche molto sintetico e quindi l'utilizzo di un linguaggio molto specifico (e almeno inizialmente poco familiare), potrebbe rendere difficile capire il loro significato o addirittura renderli del tutto incomprensibili. Man mano che diventerete più esperti in R, diventerà sempre più semplice ed immediato capire quale sia il problema e anche come risolverlo. Ma nel caso non si conosca la soluzione è necessario cercarla in altro modo.

Problema + Google = Soluzione

In qualsiasi situazione 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.



Tip-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:

Error in : could not find function ""

All Videos News Shopping Maps More Settings Tools

About 1,740,000,000 results (0.63 seconds)

stackoverflow.com › questions › error-could-not-find-f... ▾

[Error: could not find function ... in R - Stack Overflow](#)

10 answers

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 ...

[Error: could not find function "%>%" - Stack Overflow](#) 5 answers Jun 4, 2017

[Could not find function "%<>%" with dplyr loaded ...](#) 1 answer Mar 27, 2019

[dplyr error - could not find function "%>%" - Stack ...](#) 1 answer May 29, 2019

["could not find function" error though function is in ...](#) 1 answer Jul 26, 2018

[More results from stackoverflow.com](#)

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. È 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. È 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)
- 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). È un tema leggermente più avanzato ma l'idea di fondo è quella di

¹Fonente: Writing the perfect question - Jon Skeet

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?”



Trick-Box: reprex

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 7

Data Type

Working in progress.

Tipi di vettori

In R ci sono 4 tipi differenti di vettori: numerici, logici, caratteri e fattori.

7.1 Vettori Numerici

I vettori numerici sono utilizzati per compiere operazioni aritmetiche, in R sono indicati come `num`. In R ci sono è possibile specificare se i numeri contenuti nel vettore sono numeri interi, avremmo quindi un vettore di valori interi (indicato in R come `int`). Per fare ciò è possibile aggiungere `L` ad ogni valore numerico nel definire il vettore oppure usare la funzione `as.integer()` per trasformare un vettore numerico in un vettore intero.

Esempio:

```
x <- c(4L, 6L, 12L, 34L, 8L)  
x <- as.integer(c(4, 6, 12, 34, 8))
```

Nota: per trasformare un vettore intero in un vettore numerico è possibile usare la funzione `as.numeric()`.

7.2 Vettori logici

I vettori logici sono formati dai valori `TRUE` e `FALSE`, che possono essere abbreviati rispettivamente in `T` e `F`. In R i vettori logici sono indicati come `logi`. In genere, i vettori logici sono il risultato delle operazioni in cui viene chiesto ad R di valutare la condizione logica di una proposizione.

```
x>10  
## [1] FALSE FALSE TRUE TRUE FALSE
```

Nota: in R, come in molti altri software di programmazione, `TRUE` assume il valore numerico `1` e `FALSE` assume il valore `0`.

```
sum(x>10)
## [1] 2
```

E' possibile trasformare un vettore numerico in un vettore logico attraverso la funzione `as.logical()`, gli 0 assumeranno il valore FALSE mentre qualsiasi altro numero assumerà il valore TRUE.

```
as.logical(c(1,0,.034,-1,0,8))
## [1] TRUE FALSE TRUE TRUE FALSE TRUE
```

7.3 Vettori di caratteri

I vettori di caratteri contengono stringhe di caratteri e sono indicati in R con 'chr}'. Non è possibile eseguire operazioni aritmetiche con vettori di caratteri ma solo valutare se due stringhe sono uguali o differenti.

```
j<-c("Hello","World","hello","world")
j=="hello"
## [1] FALSE FALSE TRUE FALSE
```

Per trasformare un vettore qualsiasi in una vettore di caratteri è possibile usare la funzione `as.character()`.

```
as.character(x)
## [1] "4"   "6"   "12"  "34"  "8"
as.character(x>10)
## [1] "FALSE" "FALSE" "TRUE"  "TRUE" "FALSE"
```

Chapter 8

Vettori

Working in progress

8.1 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()`).

8.2 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”.

8.3 Funzioni ed Operazioni tra Vettori

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

Table 8.1: 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 un vettore
<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 9

Fattori

Working in progress.

I fattori sono utilizzati per definire delle variabili categoriali, sono indicati in R con **Factor**. Per creare una variabile categoriale in R si utilizza la funzione:

```
nome_variabile<-factor(c(..., data, ...), levels=c(...))
```

L'opzione **levels=c(...)** è usata per specificare quali sono i possibili livelli della variabile categoriale. E' possibile modificare o aggiungere nuovi livelli della variabile anche in un secondo momento utilizzando la funzione:

```
levels(nome_fattore)<- c(..., nuovi_livelli, ...)
```

Nota: nel creare un fattore R associa ad ogni livello un valore in ordine crescente e assegna agli elementi del vettore il loro valore numerico a seconda del proprio livello. Pertanto se un fattore è trasformato in un vettore numerico vengono restituiti tali valori numerici e non i livelli anche nel caso fossero dei numeri. Prendiamo per esempio la variabile **anni_istruzione**:

```
anni_istruzione<-factor(c(11,8,4,8,11,4,11,8))
anni_istruzione
## [1] 11 8 4 8 11 4 11 8
## Levels: 4 8 11
as.numeric(anni_istruzione)
## [1] 3 2 1 2 3 1 3 2
```

Per riottenere gli estti valori numerici è necessario eseguire:

```
as.numeric(as.character(anni_istruzione))
## [1] 11 8 4 8 11 4 11 8
```

Esercizi

1. Crea la variabile categoriale **sex** così definita:

```
## [1] M F M F M F F F M
## Levels: F M
```

2. Rinomina i livelli della variabile `sex` rispettivamente in "donne" e "uomini".
3. Crea la variabile categoriale `intervento` così definita:

```
## [1] CBT          Psicanalisi CBT          Psicanalisi CBT          Psicanalisi  
## [7] Controllo    Controllo    CBT          Levels: CBT Controllo Psicanalisi
```

4. Correggi nella variabile `intervento` la 7° e 8° osservazione con la voce `Farmaci`.
5. Aggiungi alla variabile `intervento` le seguenti nuove osservazioni:

```
## [1] "Farmaci"    "Controllo"   "Farmaci"
```

Chapter 10

Matrici

Working in progress.

10.1 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:

2	34	12	7
46	93	27	99
23	38	7	04

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.

10.2 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. Selziona 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

10.3 Funzioni ed Operazioni tra Matrici

Table 10.1: Operazioni con matrici

Operazione	Nome
<code><nuova-matrice> <- cbind(<matrice1>, <matrice2>)</code>	Per unire due matrici creando nuove colonne (le matrici d
<code><nuova-matrice> <- rbind(<matrice1>, <matrice2>)</code>	Per unire due matrici creando nuove righe (le matrici d
<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 colonn
<code>t(<nome-matrice>)</code>	Per ottenere la trasposta della matrice
<code>diag(<nome-matrice>)</code>	Ottenerne un vettore con gli elementi della diagonale de
<code>det(<nome-matrice>)</code>	Ottenerne il determinante della matrice (la matrice deve
<code>solve(<nome-matrice>)</code>	Ottenerne 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

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")}
```

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 11

Dataframe

Working in progress.

11.1 Creazione di DataFrames

Uno degli oggetti più utilizzati in R sono i DataFrames. I DataFrames permettono di raccogliere all'interno di uno stesso oggetto vettori di diverso tipo (i.e., vettori numerici, logici, fattori o stringhe di caratteri). Per questo motivo, i DataFrames sono utili per riportare tutti i dati riguardanti le diverse variabili misurate in un esperimento.

In genere ogni riga di un DataFrames rappresenta una singola osservazione e nelle colonne sono riportate i vari valori delle variabili misurate.

Esistono due formati principali di DataFrames:

- **Wide:** ogni singola riga rappresenta un soggetto e ogni sua risposta o variabile misurata sarà riportata in una diversa colonna.

```
data_wide<-data.frame(  
  Id=c("subj_1","subj_2","subj_3"),  
  age=c(21,23,19),  
  sex=c("F","M","F"),  
  item_1=c(2,1,1),  
  item_2=c(0,2,1),  
  item_3=c(2,0,1)  
)  
  
data_wide  
##      Id age sex item_1 item_2 item_3  
## 1 subj_1  21   F     2     0     2  
## 2 subj_2  23   M     1     2     0  
## 3 subj_3  19   F     1     1     1
```

- **Long:** ogni singola riga rappresenta una singola osservazione. Quindi i dati di ogni soggetto saranno riportati su più righe e le variabili che non cambiano tra le osservazioni saranno ripetute.

```

data_long<-data.frame(Id=rep(c("subj_1","subj_2","subj_3"),each=3),
                      age=rep(c(21,23,19),each=3),
                      sex=rep(c("F","M","F"),each=3),
                      item=rep(1:3,3),
                      response=c(2,1,1,0,2,1,2,0,1))

data_long
##      Id age sex item response
## 1 subj_1 21   F    1      2
## 2 subj_1 21   F    2      1
## 3 subj_1 21   F    3      1
## 4 subj_2 23   M    1      0
## 5 subj_2 23   M    2      2
## 6 subj_2 23   M    3      1
## 7 subj_3 19   F    1      2
## 8 subj_3 19   F    2      0
## 9 subj_3 19   F    3      1

```

In R per definire un DataFrame si utilizza il comando:

```
<nome-DataFrame> <- data.frame(variabile_1=c(...), ..., variabile_s=c(...))
```

All'interno vanno riportate le variabili che si vogliono inserire separate da virgole. Ogni variabile deve avere la **stessa lunghezza**.

Nota: di default R considera una variabile stringa all'interno di un DataFrame come una variabile categoriale. E' possibile cambiare questa opzione specificando `stringsAsFactors=FALSE`.

Esercizi

1. Crea il dataframe `data_wide` riportato precedentemente
2. Crea il dataframe `data_long` riportato precedentemente

11.2 Selezione di Elementi di un DataFrame

In R per selezionare gli elementi di un DataFrame si può, analogamente alle matrici, indicare all'interno delle parentesi quadre l'indice di riga e l'indice di colonna (**separati da virgola**).

```
<nome-DataFrame>[indice_riga , indice_colonna]
```

Per accedere ad una specifica variabile del DataFrame è possibile utilizzare l'operatore “\$”:

```
<nome-DataFrame>$<nome-variabile>
```

Per quanto riguarda l'indice di riga è possibile definire una condizione logica rispetto ad una variabile, mentre per l'indice di colonna si può indicare il nome delle variabili:

```
<nome-DataFrame>[condizione_logica , c("variabile_1", ..., "variabile_s")]
```

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

Esempio: `data_wide[data_wide$sex=="F", c("Id","age")]`

Esercizi

- Utilizzando gli **indici numerici** di riga e di colonna seleziona i dati del soggetto `subj_2` riguardanti le variabili `item` e `response` dal DataFrame `data_long`.
- Compi la stessa selezione dell'esercizio precedente usando però questa volta una condizione logica per gli indici di riga e indicando direttamente il nome delle variabili per gli indici di colonna.
- Considerando il DataFrame `data_wide` seleziona le variabili `Id` e `sex` dei soggetti che hanno risposto 1 alla variabile `item_1`.
- Considerando il DataFrame `data_long` seleziona solamente i dati riguardanti le ragazze con età superiore ai 20 anni.
- Elimina dal DataFrame `data_long` le osservazioni riguardanti il soggetto `subj_2` e la variabile "sex".

11.3 Funzioni con DataFrames

Table 11.1: Operazioni con matrici

Operazione	Nome
<code>nome_DataFrame <- cbind(nome_DataFrame, nuova_variabile)</code>	Per aggiungere una nuova variabile al DataFrame
<code>nome_DataFrame\$nome_variabile <- dati</code>	
<code>nome_DataFrame <- rbind(nome_DataFrame, nuova_variabile)</code>	Per aggiungere delle osservazioni (i nuovi dati)
<code>nrow(nome_DataFrame)</code>	Per valutare il numero di osservazioni del DataFrame
<code>ncol(nome_DataFrame)</code>	Per valutare il numero di variabili del DataFrame
<code>holder(nome_DataFrame)</code>	Nomi delle colonne del DataFrame
<code>rownames(nome_DataFrame)</code>	Nomi delle righe del DataFrame

Nota: E' possibile assegnare nomi alle colonne e righe di un DataFrame allo stesso modo delle matrici, attraverso i comandi

```
colnames(nome_DataFrame)<-c("nome_1",...,"nome_s")
names(nome_DataFrame)<-c("nome_1",...,"nome_s")
rownames(nome_DataFrame)<-c("nome_1",...,"nome_n")
```

Esercizi

- Aggiungi sia al DataFrame `data_wide` che `data_long` la variabile numerica "memory_pre".

```
data.frame(Id=c("subj_1","subj_2","subj_3"),
           memory_pre=c(3,2,1))
```

- Aggiungi sia al DataFrame `data_wide` che `data_long` la variabile categoriale "gruppo".

```
data.frame(Id=c("subj_1","subj_2","subj_3"),
           gruppo=c("trattamento","trattement","controllo"))
```

- Aggiungi al DataFrame `data_wide` i dati del soggetto `subj_4` e `subj_5`.

```
data.frame(Id=c("subj_4","subj_5"),
           age=c(25,22),
           sex=c("F","M"),
           item_1=c(1,1),
           item_2=c(0,1),
           item_3=c(2,0),
           memory_pre=c(1,3),
           gruppo=c("trattamento","controllo"))
```

4. Considerando il DataFrame `datawide` calcola la variabile "memory_post" data dalla somma degli item.
5. Considerando il DataFrame `data_wide` cambia i nomi delle variabili `item_1`, `item_2` e `item_3` rispettivamente in `problem_1`, `problem_2` e `problem_3`.

Chapter 12

Liste

Working in progress.

12.1 Creazione di Liste

Le liste sono degli speciali oggi in R che permettono di contenere al loro interno altri oggetti indipendentemente dalla loro tipologia. Possiamo quindi avere nella stessa lista sia vettori, sia matrici sia DataFrames.

In R per definire una lista si utilizza il comando:

```
<- list(nome_oggetto_1 = oggetto_1, ..., nome_oggetto_n = oggetto_n)
```

All'interno si possono riportare vari oggettive si vogliono inserire con i relativi nomi, separati da virgole.

Esercizi

1. Crea la lista `esperimento_1` contenente:

- DataFrame `data_wide`
- la matrice `A`
- il vettore `x`
- la variabile `info = "Prima raccolta dati"`

2. Crea la lista `esperimento_2` contenente:

- DataFrame `data_long`
- la matrice `C`
- il vettore `y`
- la variabile `info = "Seconda raccolta dati"`

12.2 Selezione di Elementi di una Lista

In R per selezionare gli elementi di una lista si possono usare le doppie parentesi quadre indicando l'indice della posizione dell'oggetto che si vuole selezionare:

`nome_lista[[indice_posizione]]` In alternativa, se i nomi degli oggetti sono stati specificati, è possibile utilizzare l'operatore “\$” e il nome dell'oggetto da selezionare all'interno della lista:

`nome_lista$nome_oggetto` In seguito per accedere a specifici elementi all'interno degli oggetti si utilizzano le stesse norme precedentemente presentate a seconda del tipo di oggetto.

Esempio: - `esperimento_1[[2]][,2]` - `esperimento_1$data_wide$age`

Nota: per definire o cambiare i nomi degli oggetti contenuti in una lista è possibile utilizzare la funzione: `names(nome_lista) <- c(nome_oggetto_1, ..., nome_oggetto_n)`

Esercizi

1. Utilizzando gli **indici numerici** di posizione seleziona i dati dei soggetti `subj_1` e `subj_4` riguardanti le variabili `age`, `sex` e `gruppo` dal DataFrame `data_wide` contenuto nella lista `esperimento_1`.
2. Compi la stessa selezione dell'esercizio precedente usando però questa volta il nome dell'oggetto per selezionare il DataFrame dalla lista.
3. Considerando la lista `esperimento_2` seleziona gli oggetti `data_long`, `y` e `info`
4. Cambia i nomi degli oggetti contenuti nella lista `esperimento_2` rispettivamente in `"dati_esperimento"`, `"matrice_VCV"`, `"codici_Id"` e `"note"`

Algoritmi

Introduzione

Working in progress.

Chapter 13

Definizione di Funzioni

Working in progress.

Chapter 14

Programmazione Condizionale

Working in progress.

Chapter 15

Attenti al loop

Working in progress.

Case study

Introduzione

Working in progress.

Chapter 16

Caso Studio I: Attaccamento

Working in progress.

16.1 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 repudiandae 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!