$MyFirstDatewithR$
$Manual$

Contents

1 Introduction **5**
- 1.1 About This Course . . . 5
- 1.2 Acknowledgements . . . 6

2 R and a bit of history **7**
- 2.1 S and a bit of history . . . 7
- 2.2 S-PLUS and a bit of history . . . 8
- 2.3 A bit of R history . . . 9
- 2.4 Why R? . . . 10

3 Online Resources **13**
- 3.1 The R-project and R Licence . . . 13
- 3.2 Other Online Resources . . . 15

4 R and R-Studio installation and configuration **17**
- 4.1 Installing and Updating R . . . 17
- 4.2 Graphical User Interfaces . . . 19
- 4.3 R Packages . . . 22

5 Your first R session **23**
- 5.1 Aritmetic with R . . . 23
- 5.2 Assignment . . . 24
- 5.3 The R Workspace . . . 24
- 5.4 R help . . . 25

6 Data Objects **27**
- 6.1 Data frames . . . 27
- 6.2 Other Data Objects . . . 30

7 Data Import from external sources **33**
- 7.1 Text files . . . 34
- 7.2 Microsoft Excel files . . . 36
- 7.3 Databases . . . 37
- 7.4 ODBC . . . 37
- 7.5 SQLlite . . . 37

8 Data Manipulation with R **41**
- 8.1 Pipe operator (`%>%`) . . . 42
- 8.2 dplyr verbs for data manipulation . . . 45
- 8.3 dplyr verbs for combining data . . . 53

9 Data Discovery with R **57**
- 9.1 Descriptive statistics with `summarise()` and `group_by()` . . . 58

9.2 Multiple operations 60

10 Data Visualization with R **61**
10.1 An overview of `ggplot2` grammar 62
10.2 Other plot types 68

11 Statistical Models with R **71**

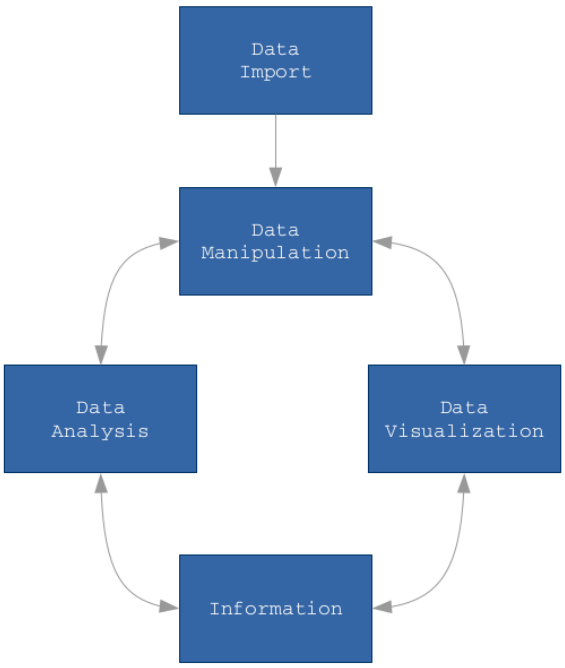
12 Data Mining with R **77**
12.1 Neural Networks 78

13 Reports with R Markdown **83**
13.1 Installation 84
13.2 Markdown Basics 84
13.3 Rendering Output 86

Chapter 1

Introduction

1.1 About This Course



This course offers the basics of R, and get an overview on methods for data import, data manipulation, data visualization and data analysis.

This manual was realized for a free R course which took place on Tuesday 13 December in Milan, organized by Microsoft in collaboaration with Quantide. For more info about the event, visit

MilanoR blog (www.milanor.net/blog/my-first-date-with-r-free-live-class-in-milano) or Facebook Page (www.facebook.com/MilanoRcommunity).

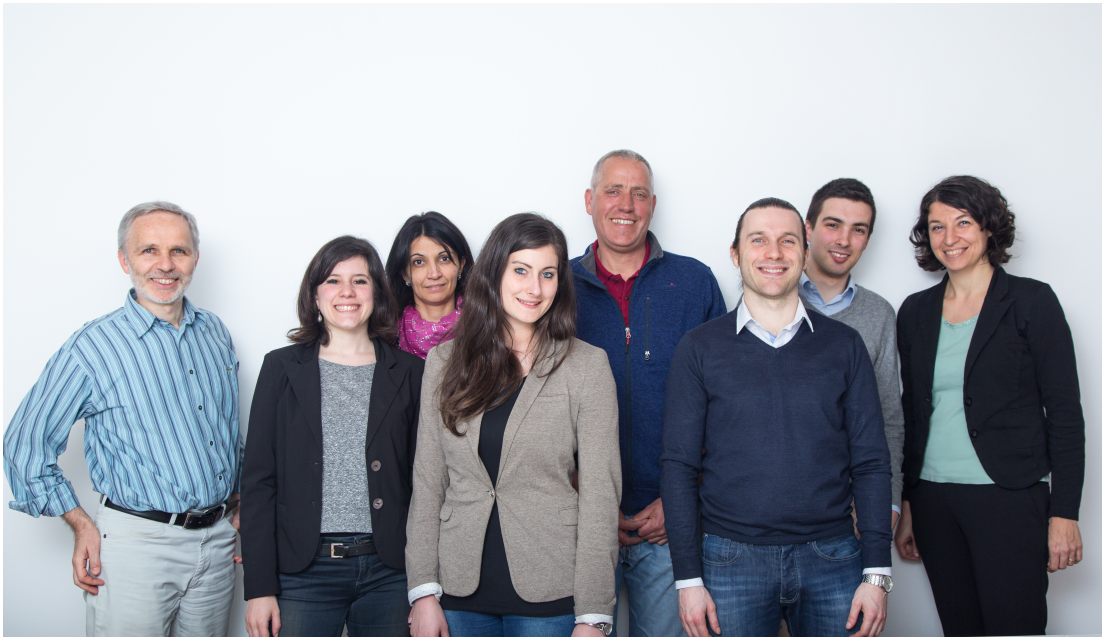
This manual is available also on a Github repository and you can download it here:

www.github.com/quantide/your-first-date-with-r/archive/master.zip

1.2 Acknowledgements

This web book is the result of several years of introductory R courses. It contains work of my colleagues:

- Daniela Manzato, who wrote the first version of this book,
- Enrico Pegoraro, who wrote chapters about statistical modeling,
- Veronica Giro and Nicola Sturaro, which reviewed and reassembled all these materials.



However, it would not have been possible without the sharing of knowledge, information, ideas, doubts and even criticisms of many individuals on the internet. I would like to extend my sincere thanks to all of them.

I am grateful to Bill Venables and John Chambers for their publications on R that provided solid foundations to my knowledge on this subject. Moreover Quick R often provides some useful ready-to-cook recipe.

I would like to express my special gratitude to Hadley Wickham for providing and sharing his research on the `R` packages: `dplyr`, `ggplot2` and `readxl`. Without his contribution, a part of this manual would never have been written.

I should like to thank also DataCamp for providing useful R courses for learning data science, from which parts of this manuals are inspired.

I finally express my sincere excuses to all researchers and R enthusiasts I have borrowed any knowledge from without mentioning them. This was not intentional, simply I had not always tracked my sources. If this is the case, please contact me directly and I will be more than happy to include any appropriate reference in this manual.

Andrea Spanò,
Quantide s.r.l.

Chapter 2

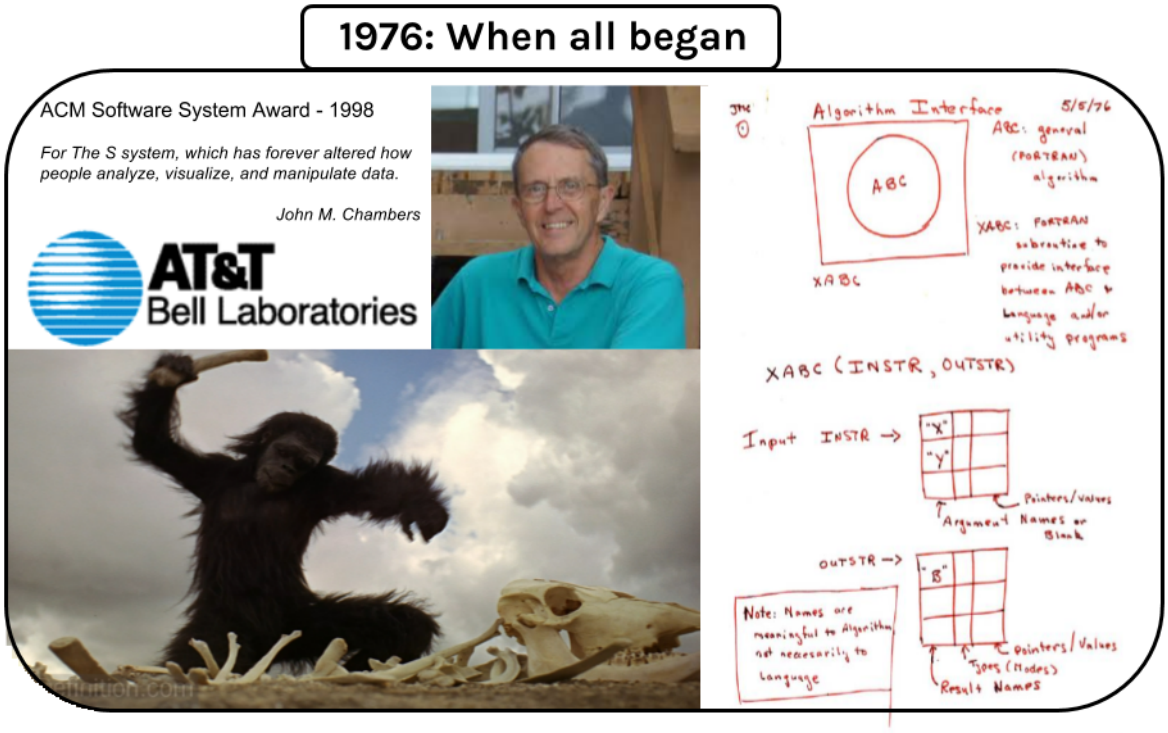
R and a bit of history

R is a programming environment for data analysis, graphics and statistical computing. The R language is widely used among statisticians for developing statistical software and data analysis. R was born as a dialect of the S language, which is a statistical programming language developed by John Chambers and others in Bell Laboratories. R is also very close to S-PLUS, which is a commercial implementation of the S programming language, sold by Insightful Corporation.

Let us see a bit of history.

2.1 S and a bit of history

S is a statistical programming language developed by John Chambers and others in Bell Laboratories. The aim of the language, as expressed by John Chambers, is "to turn ideas into software, quickly and faithfully".



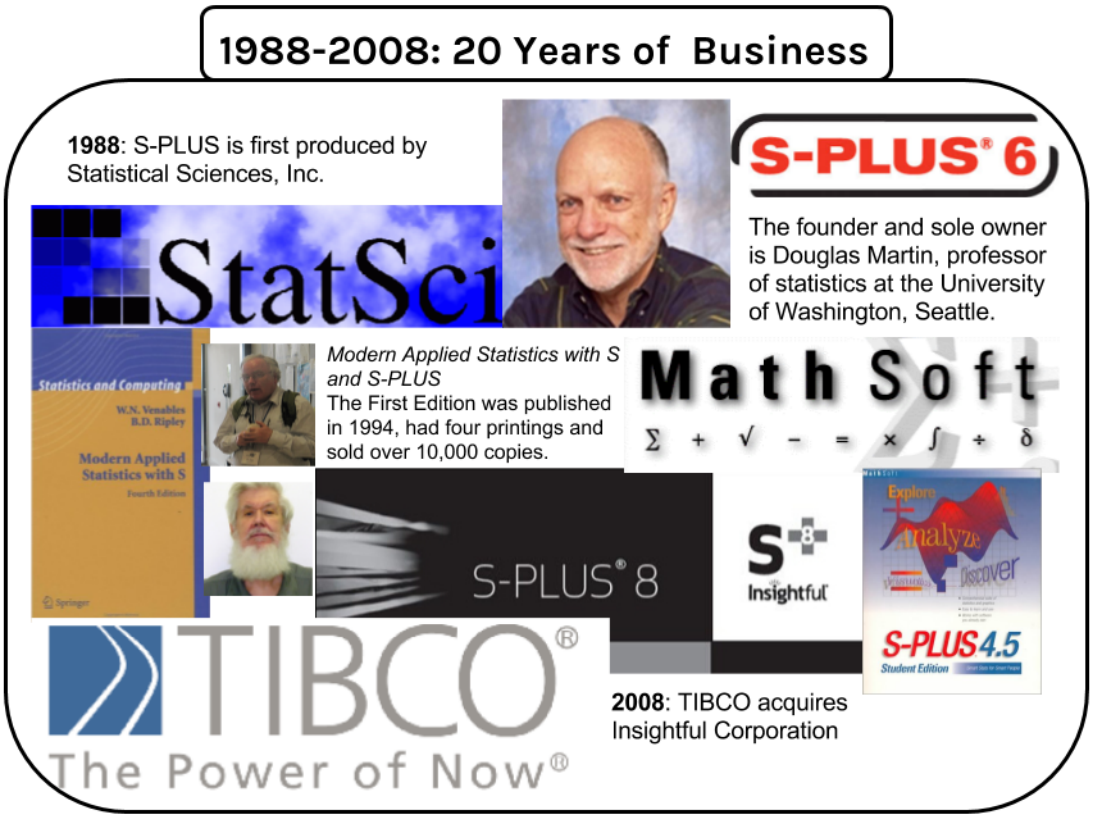
A bit of history:

- 1976: the first version of S is developed as an internal statistical analysis environment. It is originally implemented as Fortran libraries.
- 1980: the first version of S is distributed outside of Bell Laboratories. In 1981, source version is made available.
- 1984: Richard A. Becker and John M. Chambers, "S. An Interactive Environment for Data Analysis and Graphics". (Brown Book). Historical interest only.
- 1988: Richard A. Becker, John M. Chambers and Allan R. Wilks, "The New S Language". London: Chapman & Hall. (Blue Book). It introduces what is now known as S version 2. The system is rewritten in C and begins to resemble the system that we have today.
- 1992: John M. Chambers and Trevor J. Hastie, "Statistical Models in S". (White Book). It introduces S version 3, often abbreviated S3, which adds structures to facilitate statistical modeling in S.
- 1998: John M. Chambers, "Programming with Data". (Green Book). It introduces S version 4, often abbreviated S4, which provides advanced object-oriented features. S4 classes differ markedly from S3 classes.

The S language itself has not changed dramatically since 1998.

2.2 S-PLUS and a bit of history

S-PLUS is a commercial implementation of the S programming language. S-PLUS provides a number of fancy features (GUIs, mostly) on top of it, hence the "PLUS".

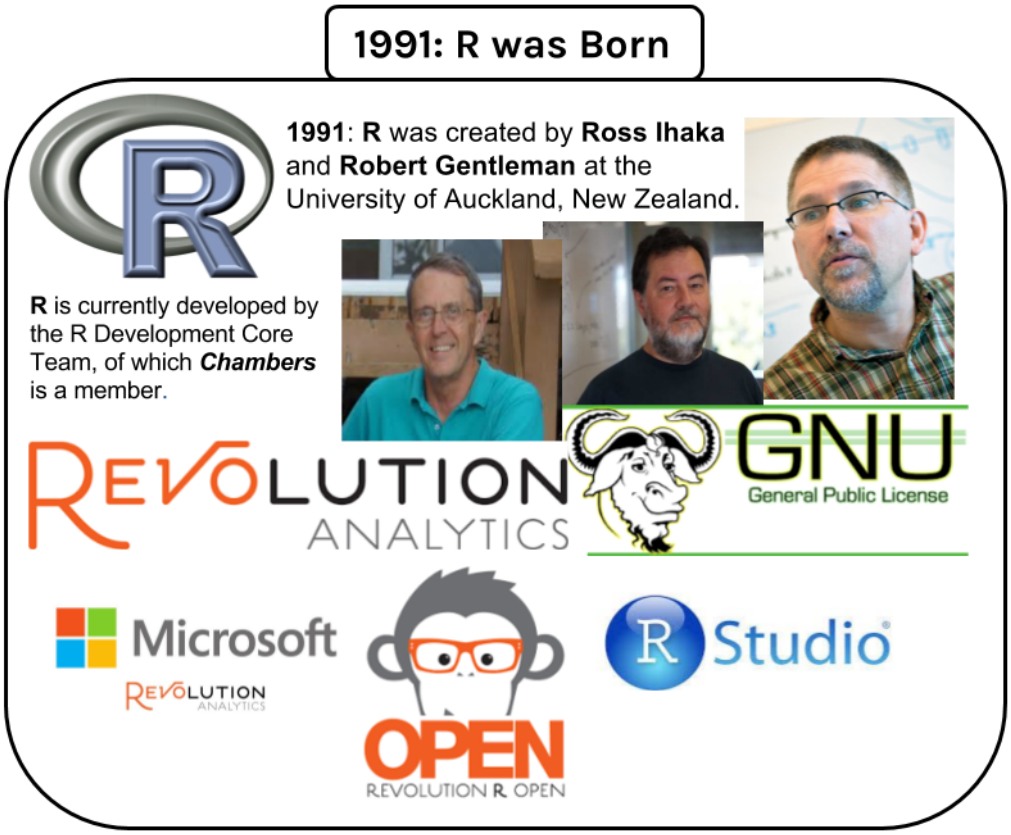


A bit of history:

- 1988: S-PLUS is first produced by a Seattle-based start-up company called Statistical Sciences, Inc. The founder and sole owner is R. Douglas Martin, professor of statistics at the University of Washington, Seattle.
- 1993: Statistical Sciences, Inc. acquires the exclusive license to distribute S and merges with MathSoft.
- 2001: MathSoft sells its Cambridge-based Engineering and Education Products Division (EEPD). It changes name to Insightful Corporation.
- 2004: Insightful purchases the S language from Lucent Technologies for $2 million.
- 2008: TIBCO acquires Insightful Corporation.

2.3 A bit of R history

R was initially developed in early 90s by Robert Gentleman and Ross Ihaka at the Department of Statistics of the University of Auckland, New Zealand. The R name is partly based on the (first) names of the first two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of S.



A bit of history:

- 1991: R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zeland.
- 1993: First announcement of R to the public.
- 1995: Martin Maechler convinces Ross Ihaka and Robert Gentleman to use the GNU General Public License to make R free software.

- 1997: The R Development Core Team is formed. The team controls the source code for R.
- 2000: R version 1.0.0 released. Developers consider R stable enough for production use.
- 2004: R version 2.0.0 released. Introduced lazy loading, which enables fast loading of data with minimal expense of system memory.
- 2013: R version 3.0.0 released. Introduced long vectors.

While R is an open source project supported by the community developing it, some companies strive to provide commercial support and/or extensions for their customers. R history is intertwined with that of its commercial support:

- 2007: Revolution Analytics was founded to provide commercial support for Revolution R, the distribution of R developed by Revolution Analytics which also includes components developed by the company.
- 2010: RStudio was founded. It is a company that develops free and open tools for the R community.
- 2014: Microsoft Corporation starts the release of Microsoft R Open, an enhanced distribution of R, formerly known as Revolution R Open (RRO).
- 2015: Microsoft Corporation completed the acquisition of Revolution Analytics.

We will deepen the commercial support tools mentioned in *Online Resources* chapter.

2.4 Why R?

In December 2016, R is in 17th place of TIOBE Programming Community Index (www.tiobe.com), that is an indicator of the popularity of programming languages. R is above SAS that is in 22th place.

R is the leading analytics tools for data science used by respondents to the Rexer Analytics Survey in 2015:

Moreover, the number of companies using R is grown all over the world:

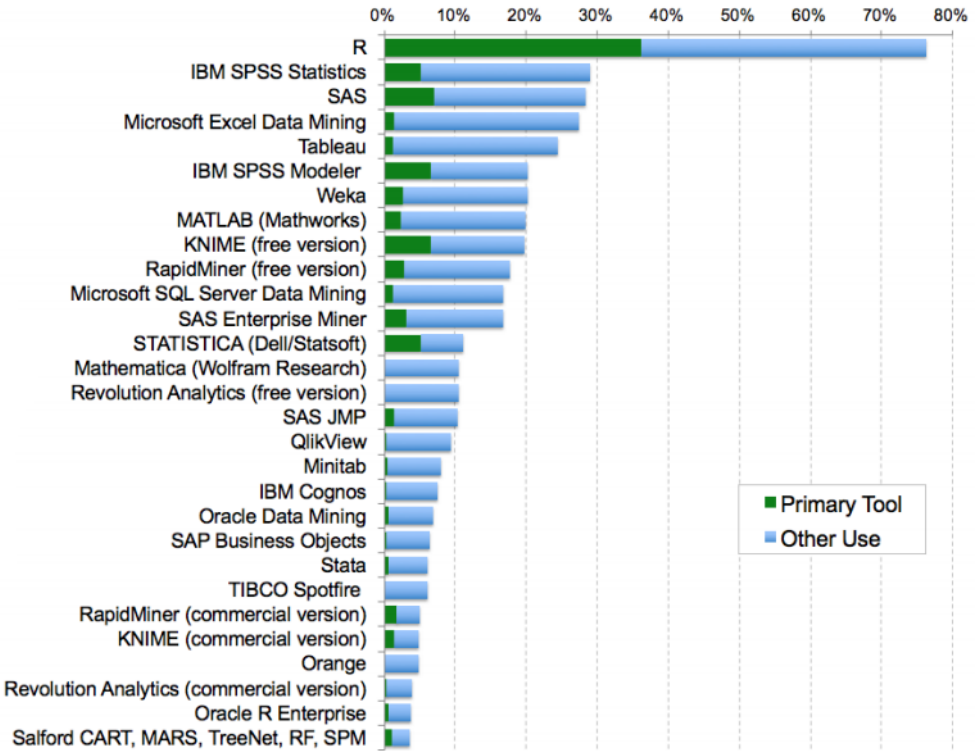


Figure 2.1: Source: r4stats.com/2015/10/13/rexer-analytics-survey-results

Figure 2.2: Source: blog.revolutionanalytics.com

Chapter 3

Online Resources

R strength is its community, which is distributed and keeps growing all over the World!

Ramnath Vaidyanathan

Thanks to its community, R can boast a wide variety of online resources, free and otherwise, to learn more about it, ranging from websites, blogs and commercial support tools.

Let us have a look at the most important.

3.1 The R-project and R Licence

R is supported by a wide community of academic users, professors, companies and developers. This community composes the so-called "R-project". The "R-project" is supported by the "R Foundation". The R Foundation is a not for profit organisation, working in the public interest.

R is an official part of the Free Software Foundation's GNU project. The R Foundation has similar goals to other open source software foundations like the Apache Foundation or the GNOME Foundation. R is free and open source software. It is released under the GPL (version 2) licence.

R is free:

- you can have R without paying for it (freeware);
- you can copy and re-use the software (free software);
- you can access source code and modify it (open source).

3.1.1 R-project Website

The R-project website (www.r-project.org) is the starting point for R materials.

The website contains:

- the software and packages;
- the search engine interface (the same queries can be submitted with the RSiteSearch ('query') function within R);
- the on-line documentation both in HTML and in PDF format. The HTML version can be accessed with the `help.start()` function within R;

Figure 3.1: Figure generated by Ramnath Vaidyanathan

- the R Journal. The R Journal is the open access, refereed journal of the R project. It features short to medium length articles covering topics that might be of interest to users or developers of R;
- the interface to the mailing list;
- the wiki, suggested books and many others.

The on-line documentation includes the following manuals. These manuals have been written by the R Development Core Team itself and contain precious information.

- *An Introduction to R* gives an introduction to the language and how to use R for doing statistical analysis and graphics.
- *Writing R Extensions* covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.
- *R Data Import/Export* describes the import and export facilities available either in R itself or via packages which are available from CRAN.
- *R Installation and Administration.*

Other manuals and tutorials provided by R users can be downloaded from the R-project website (cran.r-project.org/other-docs.html).

Mailing lists is the most important tool to contact the R community. Mailing lists can be accessed from the R-project website (www.r-project.org/mail.html).

There are five general mailing lists devoted to R:

- *R-announce*: This list is for major announcements about the development of R and the availability of new code.

- *R-packages*: This list is for announcements as well, usually on the availability of new or enhanced contributed packages (on CRAN, typically).
- *R-help*: The "main" R mailing list, for discussion about problems and solutions using R, announcements about the availability of new functionality for R and documentation of R, comparison and compatibility with S-plus, and for the posting of nice examples and benchmarks.
- *R-devel*: This list is intended for questions and discussion about code development in R.
- *R-package-devel*: This list is to get help about package development in R.

3.2 Other Online Resources

It is very difficult estimate how many sites about R are on-line. However, Google returns 224.000.000 sites searching "R stat blog". Also if only the 0.1% of these sites talk about R, it means almost 220.000 sites about R.

R-bloggers (www.r-bloggers.com) is a blog aggregator of content collected from bloggers who write about R. R-bloggers contains R news and tutorials contributed by hundreds of R bloggers.

We suggest you to visit MilanoR (www.milanor.net), which is the blog of R users in the Milan Area. Its aim is to exchange knowledge, learn and share tricks and techniques and provide R beginners with an opportunity to meet more experienced users.

Other useful websites about R are:

- Stack Overflow (www.stackoverflow.com) is a website that features questions and answers on a wide range of topics in computer programming, among which r.
- Quick-R (www.statmethods.net) is a useful on-line guide to R. It provides many examples and useful tips.
- R seek (rseek.org) uses Google to search in a selected list of websites about R.

3.2.1 R Commercial Support

While R is an open source project supported by the community developing it, some companies strive to provide commercial support and/or extensions for their customers.

3.2.1.1 Microsoft Corporation

Microsoft Corporation provides Microsoft R Open www.mran.microsoft.com/open, an enhanced distribution of R, formerly known as Revolution R Open (RRO). It is a complete open source platform for statistical analysis and data science. The current version, Microsoft R Open 3.3.2, is based on (and 100% compatible with) R-3.3.2 and is therefore fully compatibility with all packages, scripts and applications that work with that version of R. It includes additional capabilities for improved performance, reproducibility, as well as support for Windows and Linux-based platforms. Like R, Microsoft R Open is open source and free to download, use, and share. It is available for Windows, Linux and Mac Os operative systems and you can download it here.

3.2.1.2 RStudio, Inc.

RStudio, Inc. (www.rstudio.com) is a company that develops free and open tools for the R community, inspired by the innovations of R users in science, education, and industry. These include the RStudio development environment as well as the `shiny`, `ggvis`, and `dplyr` packages (among many others). RStudio

was founded around December 2010 by JJ Allaire, creator of the programming language ColdFusion. Hadley Wickham is the Chief Scientist at RStudio. RStudio has a mission to provide the most widely used open source and enterprise-ready professional software for the R statistical computing environment. These tools will further the cause of expanding the use of R and the field of data science. It also offers open source and enterprise ready tools for the R computing environment. The flagship product of the RStudio team is an Integrated Development Environment (IDE) which makes it easy for analysts, scientists, data scientists and quants to perform their analyses. It also offer Shiny: a platform that allows you to take those analyses and share them with your team/organization by creating interactive web applications.

Chapter 4

R and R-Studio installation and configuration

4.1 Installing and Updating R

4.1.1 Design of the R System

The R system is divided into two conceptual parts:

1. The "base" R system that you download from CRAN.
2. Everything else.

R functionality is divided into a number of packages. The "base" R system contains, among other things, the *base* package which is required to run R and contains the most fundamental functions. The "base" system contains also some other packages. Furthermore, every R installation contains "recommended" packages, that are not necessarily maintained by R Core.

"Everything else" point out CRAN "contributed" packages and packages that are not on CRAN. This does not mean that these packages are necessarily of lesser quality than the above, e.g., many contributed packages on CRAN are written and maintained by R Core members. The goal is simply to try to keep the base distribution as lean as possible. Beyond CRAN, many packages are avalaible in BioConductor project or in Github repositories.

Figure 4.1: Screenshot of www.r-project.org

4.1.2 Installing R

R is available for Windows, Mac Os and Linux.

The base R can be downloaded from the Comprehensive R Archive Network (CRAN) website. The CRAN is a collection of sites which carry identical material, consisting of the R distribution(s), the contributed extensions, documentation for R, and binaries.

Go to www.r-project.org. Here you can read about R and see what's new in the latest version.

Click on CRAN under Download in the left list. That'll take you to a list of servers (mirrors) in different countries where you can download R.

Now you can choose what operative system you have. Choose within the upper box with *Download and Install R*. The box contains *Precompiled binary distributions*; sounds complicated, but means that the program is ready to use, with installation program and all.

Windows users click on "base" and then click *Download R X.X.X for Windows*.

Mac users click on *R-X.X.X.pkg (latest version)*.

Linux users find download files and installation instructions for their Linux distribution in the website.

X.X.X identify the current version of R. In December 2016, the current version is the 3.3.2.

To install R, Windows and Mac users must just double click on the file and follow the installation instructions.

Linux users can download and install R using the precompiled R binary for their distribution. Alternatively, experienced Linux users can compile R from sources. Currently, precompiled R binary are available for

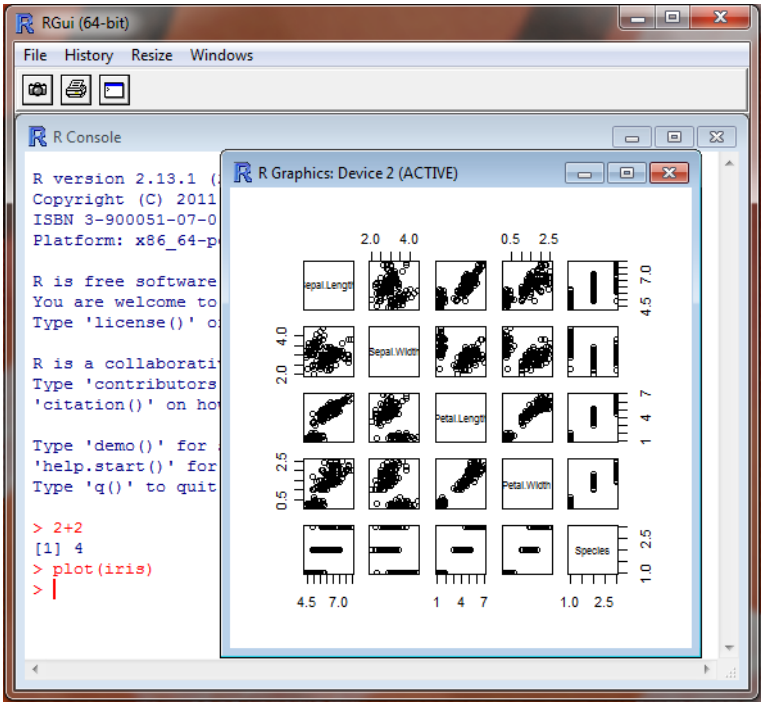


Figure 4.2: Screenshot of a first try with R

Debian, Ubuntu, Suse and RedHat.

The Figure Screenshot of a first try with R shows a first try with R, just after installation.

4.1.3 Updating R

In Windows and Mac, there is not an automatic way to update R. Two or more versions of R can coexist in the same machine, so a newer release of R can be installed beside the old release. Unfortunately, packages must be re-installed in the "new" R version. A copy-and-paste of package files from the directory containing the packages of the "old" R version to the directory containing the packages of the "new" R version may be useful when many packages are installed. Within R, the `.Library` variable shows the directory containing the packages. After the copy-and-paste, packages of the "new" R version should be updated.

In Linux, usually one R version at time can be installed. R can be updated following the instructions in the CRAN website.

4.2 Graphical User Interfaces

4.2.1 R Default Interfaces

R is provided with a Command Line Interface (CLI), which is the preferred user interface for power users because it allows direct control on calculations and it is flexible. However, good knowledge of the language is required. CLI is thus intimidating for beginners.

R is available for many different operative systems so it does not provide the same graphical interface. When you use the R program interactively, it issues a prompt when it expects input commands. The default prompt is '>'. In Windows, RGui is the default GUI. Inside RGui there is the RConsole window. In Macintosh,

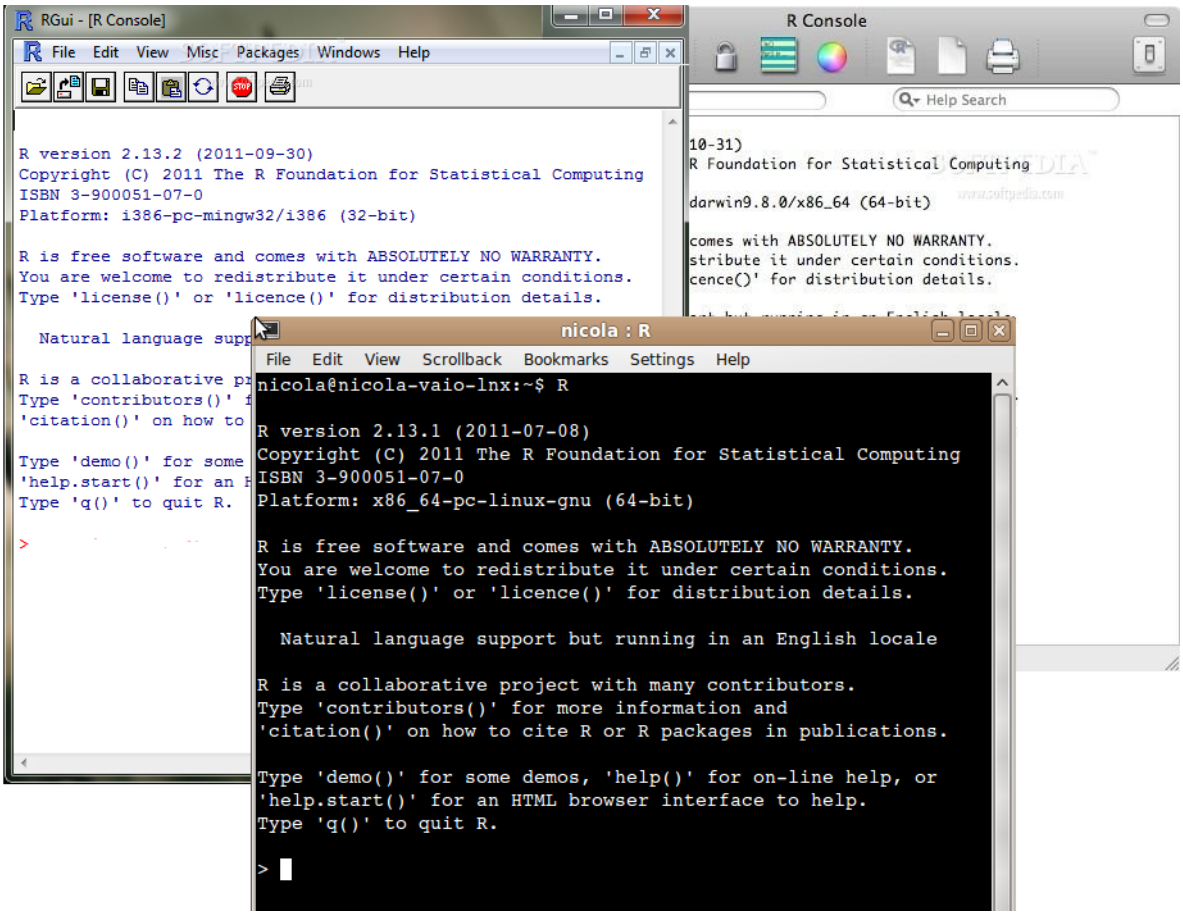


Figure 4.3: R Default Interfaces

RConsole is the default GUI. In Linux, R does not provide any graphical interface by default and it can be used through CLI.

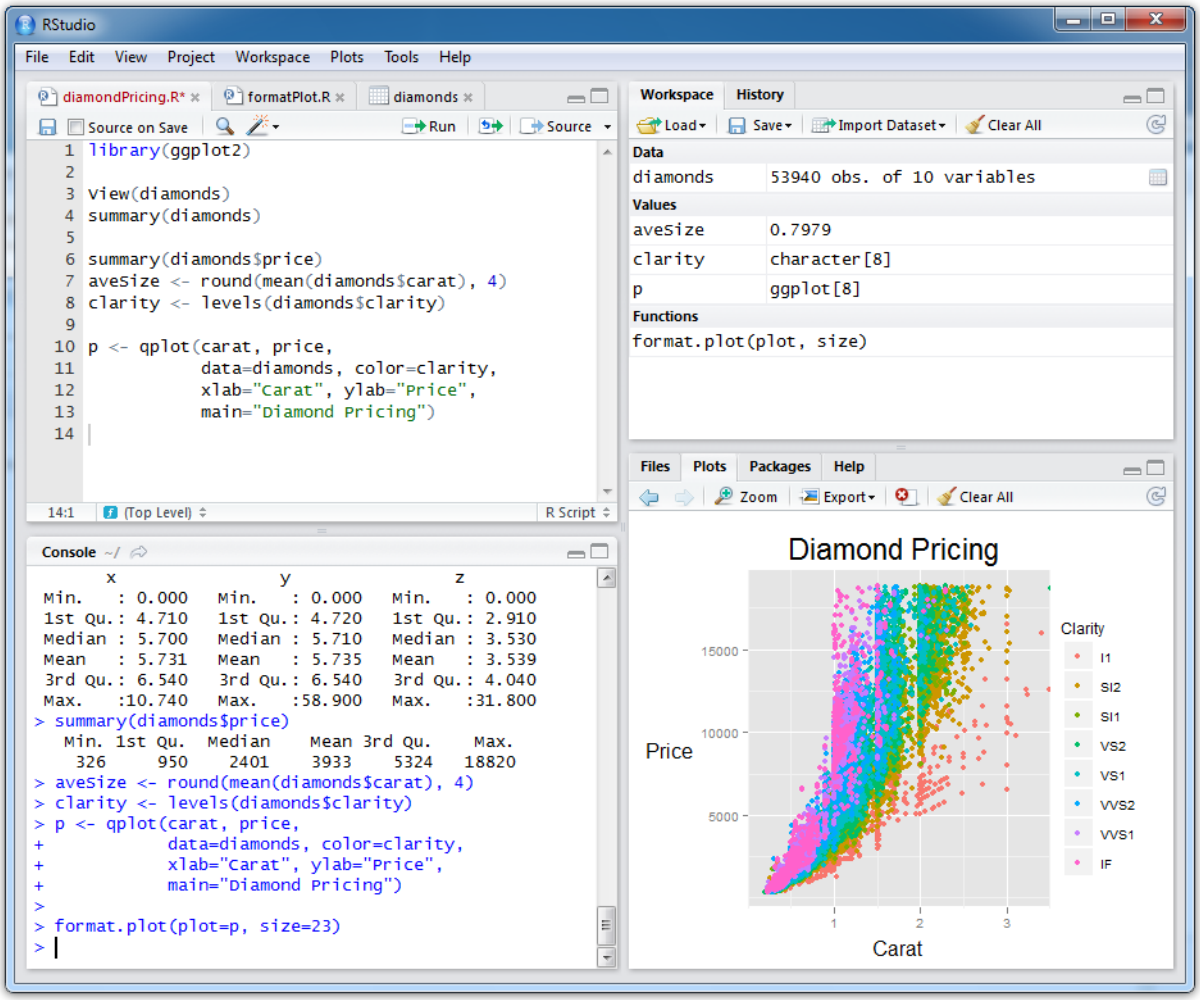


Figure 4.4: Screenshot of an RStudio session

4.2.2 RStudio

RStudio (www.rstudio.org) is a free and open source multi-platform integrated development environment (IDE) for R. It provides syntax highlighting, code completion and smart indentation. Moreover, it executes R code directly from the source editor and it manages easily multiple working directories using projects. It provides:

- workspace browser and data viewer;
- plot history, zooming, and flexible image and PDF export;
- integrated R help and documentation;
- Sweave authoring including one-click PDF preview;
- searchable command history.

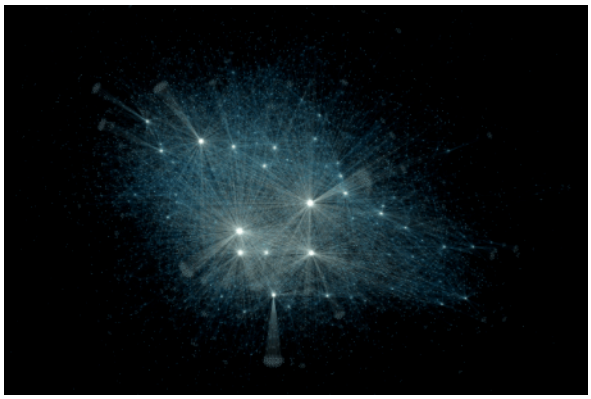


Figure 4.5: R packages universe

4.2.3 Installing RStudio

RStudio is available for Windows, Mac Os and Linux and you can install it from its website (www.rstudio.org), clicking on *Download RStudio*. Next, click *Download RStudio Desktop*. At this stage, click the link to the version of RStudio appropriate for your system in *Installers for Supported Platforms* section, which downloads RStudio to your computer. Run the installation file and RStudio will be installed on your system.

4.3 R Packages

4.3.1 Installing R Packages

R functions are collected in packages. Packages that are not contained in the "base" R systems can be downloaded from the CRAN website. A list of R packages accompanied by a brief description can be found on the CRAN website where there are more than 9500 packages available. Many of these packages are very useful; however, there are some packages in prerelease, incomplete packages, "abandoned" packages (i.e. not more updated) and/or packages containing functions with errors or compatibility troubles.

This is the R packages universe!

The simple way to install an R package is through the `install.packages()` function, directly from R. In Linux, R must be executed as administrator to install a package. Installation must be executed before the first use of a library.

When a function of a package that is not contained in the "base" R systems is required, the package must be loaded. The `require(pkg)` function load the *pkg* package.

4.3.2 Updating R Packages

R packages can be updated typing `update.packages()` within R. In Linux, R must be executed as administrator to update a package. Packages should be updated regularly.

Chapter 5

Your first R session

5.1 Aritmetic with R

Start the R system, the cursor is waiting for you to type in some R commands. For example, use R as a simple calculator:

```
6 + 3
```

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

```
5 - 9
```

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

```
4 * 6
```

```
## [1] 24
```

```
8 / 3
```

```
## [1] 2.666667
```

```
5 ^ 2
```

```
## [1] 25
```

```
(1 + 0.05)^8
```

```
## [1] 1.477455
```

```
exp(3)
```

```
## [1] 20.08554
```

```
log(14)
```

```
## [1] 2.639057
```

```
23.76 * log(8)/(23 + atan(9))
```

```
## [1] 2.01992
```

5.2 Assignment

Results of calculations can be stored in objects using the assignment operator:

```
x <- log(14)
y <- 23.76 * log(8)/(23 + atan(9))
```

To print the object just enter the name of the object or use `print()` function.

```
x
```

```
## [1] 2.639057
```

```
print(y)
```

```
## [1] 2.01992
```

These objects can then be used in other calculations.

```
z <- x + y
z
```

```
## [1] 4.658978
```

5.3 The R Workspace

The workspace is your current R working environment and includes any user-defined objects. It is also known as global environment.

5.3.1 Objects listing

Objects created during an R session are hold in memory. To list the objects in the current R session, the function `ls()` or the function `objects()` may be used.

```
ls()
```

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

```
objects()
```

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

5.3.2 Removing objects

If a value to an object that already exists is assigned then the contents of the object will be overwritten with the new value (without a warning!). The function `rm()` ought be used to remove one or more objects from your session.

```
rm(x)
ls()
```

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

5.4 R help

Within R, the following functions provide help about R itself:

- The HTML version of R's online documentation can be printed on-screen by typing `help.start()`;
- Online documentation for most of the functions and variables in R can be printed on-screen by typing `help(name)` (or `?name`), where *name* is the name of the topic help is sought for;
- Online documentation for finding help pages on a vague topic can be printed on-screen by typing `help.search('topic')`;
- A list of function containing *topic* in the name can be printed on-screen by typing `apropos('topic')`;
- A research in the website can be performed by typing `RSiteSearch('query')`, where *query* is the search query.

For example, to get help about the `mean()` function, the `help()` function can be used.

```
help(mean)
```

The `help()` function can be called using the `?`.

```
?mean
```

To get a list of functions concerning the mean, the `help.search()` function can be used.

```
help.search("mean")
```

To get a list of function containing "mean" in the name, the `apropos()` function can be used.

```
apropos("mean")
```

Chapter 6

Data Objects

6.1 Data frames

Data frames are the primary data structure in R. A data frame is a table, in which each column contains measurements on one variable, and each row contains one case. Each row may be treated as a single observation of multiple "variables".

```
df <- data.frame(
  name = c("James", "Stevie", "Otis", "Bob", "Levon", "Patti", "Karen"),
  height = c(180, 170, 175, 190, 168, 160, 165),
  graduated = c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE),
  gender = factor(c("M", "M", "M", "M", "M", "F", "F")),
  stringsAsFactors = FALSE)
df
```

```
##     name height graduated gender
## 1  James    180      TRUE      M
## 2 Stevie    170      TRUE      M
## 3   Otis    175     FALSE      M
## 4    Bob    190     FALSE      M
## 5  Levon    168     FALSE      M
## 6  Patti    160      TRUE      F
## 7  Karen    165      TRUE      F
```

Technically, in R a data frame is a list of column vectors that can be of various types, but that have to be of the same length. A **vector** is a "primitive" data object, it is a sequence of data elements of the same basic type:

- numeric
- character
- logical

```
# Numeric vector
height <- c(180, 170, 175, 190, 168, 160, 165)
height
```

```
## [1] 180 170 175 190 168 160 165
```

```
# Character vector
name <- c("James", "Stevie", "Otis", "Bob", "Levon", "Patti", "Karen")
name
```

```
## [1] "James"  "Stevie" "Otis"   "Bob"    "Levon"  "Patti"  "Karen"
```

```
# Logical vector
graduated <- c(TRUE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE)
graduated
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE
```

A data frame can include also **factors**, which are vector-like objects used to specify a discrete classification (grouping) of the components of other vectors of the same length. Factor variables are categorical variables that can be either numeric or string variables.

```
# Factor variable
gender <- factor(c("M", "M", "M", "M", "M", "F", "F"))
gender
```

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

A data frame is generated by `data.frame()` function. The vectors composing the data frame can be defined inside the `data.frame()` function itself (as seen in the first example) or can be defined outside it. In the following example, we refer to the previously defined vectors (`name`, `height`, `graduated`) and factor (`gender`) to generate a dataframe:

```
df <- data.frame(name, height, graduated, gender, stringsAsFactors = FALSE)
df
```

```
##     name height graduated gender
## 1  James    180      TRUE      M
## 2 Stevie    170      TRUE      M
## 3   Otis    175     FALSE      M
## 4    Bob    190     FALSE      M
## 5  Levon    168     FALSE      M
## 6  Patti    160      TRUE      F
## 7  Karen    165      TRUE      F
```

The management of character vectors in R requires a detailed explanation. By default, numeric vectors become part of data frames as such, whereas character vectors are transformed into factors whose levels correspond to the vector's unique values. This behaviour is surely effective when character vectors represent categorical variables. However, this behaviour is disturbing when a character vector represents a set of strings which are not necessarily referable to a definite number of modes or to numerous and/or unique modes. This behaviour is managed by the `stringsAsFactors` logical parameter. The default setting is `stringsAsFactors = TRUE` which tells R to transform character vectors into factors inside a data frame; `stringsAsFactors = FALSE` does not change character vectors.

Furthermore, data imported in R from external sources, such as text files, Excel files or databases, is saved in R as data frame-like objects.

6.1.1 `tbl_df`: the `dplyr` Data Frame Class

Sometimes data frames have large dimensions. `dplyr` package provide `tbl_df`, which is a wrapper around a data frame that will not accidentally print a lot of data to the screen; indeed tbl objects only print a few rows and all the columns that fit on one screen, describing the rest of it as text.

When the class of data object is not tbl, `tbl_df()` function should be used.
Let us consider `mtcars`, a dataset included in `datasets` package (automatically loaded at the start of an R session):

```
# Example of data frame
data("mtcars")
class(mtcars)
```

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

```
# If we do not convert it as a tbl_df, all mtcars rows
# and columns will be printed when calling mtcars
dim(mtcars)
```

```
## [1] 32 11
```

```
mtcars
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
# dplyr version of the same data frame (tbl_df conversion)
require(dplyr)
mtcars_tbl <- tbl_df(mtcars)
class(mtcars_tbl)
```

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

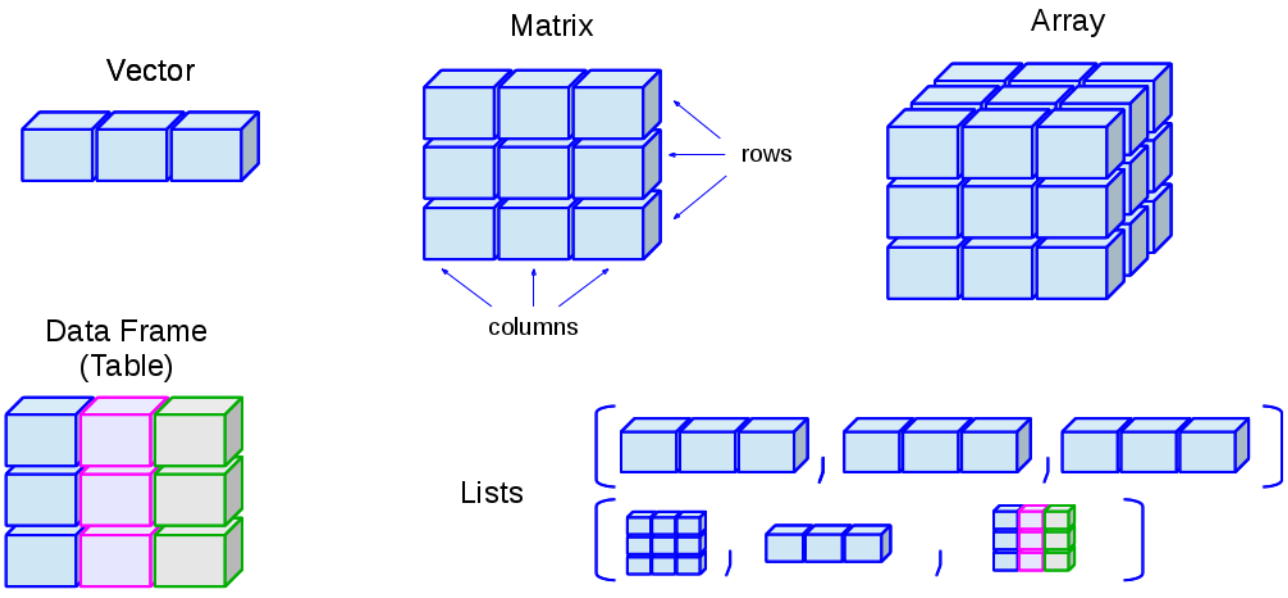


Figure 6.1: Source: venus.ifca.unican.es/Rintro/dataStruct.html

```
mtcars_tbl
```

```
## # A tibble: 32 × 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
## *  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
## 2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
## 3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
## 4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
## 5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
## 6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
## 7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
## 8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
## 9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
## 10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
## # ... with 22 more rows
```

6.2 Other Data Objects

The structures of data objects are represented in the following figure:

We have already seen data frames, vectors and factors. Let us have a look to matrices, array and lists.

6.2.1 Matrices

Matrices are generalizations of vectors. Like vectors, matrices need to contain elements of the same kind.

```
matrix(1:8, nrow = 2, ncol = 4)
```

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

6.2.2 Array

They are similar to matrices but they can be multi-dimensional (more than two dimensions)

```
z <- array(1:24, dim=c(2,3,4))
z
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

6.2.3 Lists

A list is an ordered collection of objects. Each object is a component of the list. Each element of the list can have a different structure. It can be a list itself, a vector, a matrix, an array, a factor or a data frame. A list allows you to gather a variety of (possibly unrelated) objects under one name.

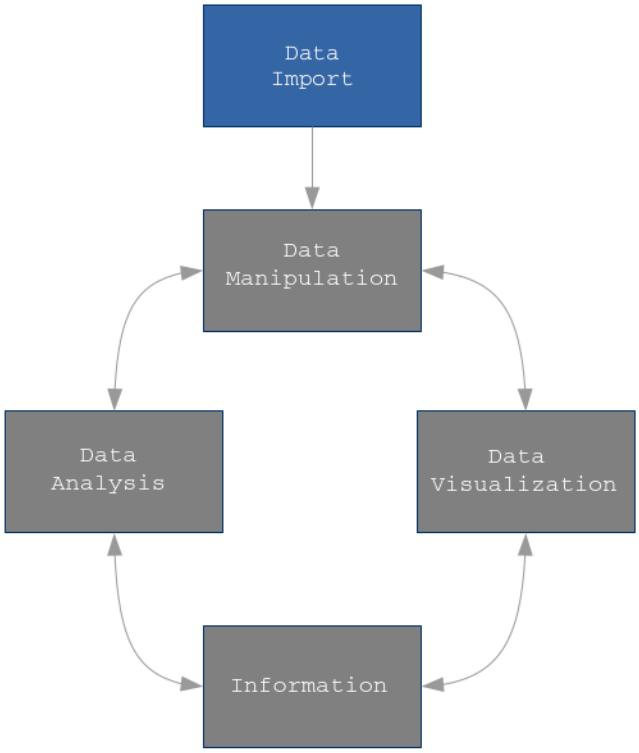
```
my_list <- list(vec = 1:7, mat = matrix(1:12, ncol = 3),
  lis = list(a = 1, b = letters[1:4]))
my_list
```

```
## $vec
## [1] 1 2 3 4 5 6 7
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
##
## $lis
## $lis$a
## [1] 1
##
## $lis$b
## [1] "a" "b" "c" "d"
```

Chapter 7

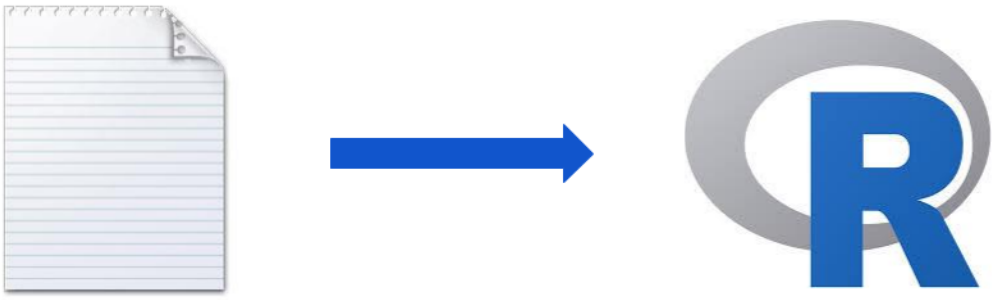
Data Import from external sources



In the following paragraphs we will explore data import methods for:

- Text files
- Microsoft Excel files
- Databases

7.1 Text files



The `read.table()` function imports a text file (ASCII) with a table structure where each row represents a case.

A full path can be provided, but it must be modified by each user, otherwise it fails:

```
df <- read.table("C:/Users/UserName/Documents/data/tennis.txt", header = TRUE, sep = "", dec = ".")
```

```
## Warning in file(file, "rt"): cannot open file 'C:/Users/UserName/Documents/
## data/tennis.txt': No such file or directory
```

```
## Error in file(file, "rt"): cannot open the connection
```

The path uses the slash ("/") as delimiting character, in the UNIX-like style. Under Windows, can be used both a slash character or a doubled backslash character ("\\").

So, it is strongly suggested to set the working directory to the directory containing the data.

`getwd()` function allows you to view the current working directory:

```
getwd()
```

```
## [1] "C:/Users/Andrea/Documents"
```

and `setwd()` function allows you to set the working directory on "data" folder, in this way:

```
setwd("./data")
```

```
## [1] "C:/Users/Andrea/Documents/data"
```

Now, the text file can be imported just providing its filename:

```
df <- read.table("tennis.txt", header = TRUE)
```

```
head(df)
```

```
##         Name First.Name Age Sex Rank Slams Won Lost Earnings Citizen
## 1    Sampras       Pete  23   M    1     2  74   11 3607.812      US
## 2     Agassi      Andre  24   M    2     1  51   13 1941.667      US
## 3     Becker      Boris  27   M    3     0  48   16 2029.756 Germany
## 4   Bruguera      Sergi  24   M    4     1  65   23 3031.874   Spain
## 5 Ivanisevic      Goran  23   M    5     0  63   26 2060.278 Croatia
## 6       Graf     Steffi  25   F    1     1  58    6 1487.980 Germany
```

```
str(df)
```

```
## 'data.frame':    10 obs. of  10 variables:
##  $ Name      : Factor w/ 10 levels "Agassi","Becker",..: 9 1 2 3 5 4 10 6 7 8
##  $ First.Name: Factor w/ 10 levels "Andre","Arantxa",..: 8 1 3 9 5 10 2 4 6 7
```

```
##  $ Age       : int  23 24 27 24 23 25 23 22 26 20
##  $ Sex       : Factor w/ 2 levels "F","M": 2 2 2 2 2 1 1 1 1 1
##  $ Rank      : int  1 2 3 4 5 1 2 3 4 5
##  $ Slams     : int  2 1 0 1 0 1 2 1 0 0
##  $ Won       : int  74 51 48 65 63 58 74 55 43 45
##  $ Lost      : int  11 13 16 23 26 6 9 15 11 18
##  $ Earnings  : num  3608 1942 2030 3032 2060 ...
##  $ Citizen   : Factor w/ 6 levels "Croatia","Czech Republic",..: 6 6 4 5 1 4 5 5 2 3
```

The `header = TRUE` option tells R that the first row of the file contains column headings and it is used to assign the name of variables. If the first row contains the first case the `header = FALSE` ought to be used and the names of the variables are automatically assigned.

The `sep` argument specifies the separator between different cases. The default value for the `read.table()` function is `sep = ""` which takes into consideration the fields delimited by a white space, be it one or more spaces or tabulations.

The `dec` argument specifies the decimal separator. The argument usually assumes the `dec = "."` (default) or `dec = ","` values.

```
df <- read.table("tennis.txt", header = TRUE, sep = "", dec = ".")
head(df)
```

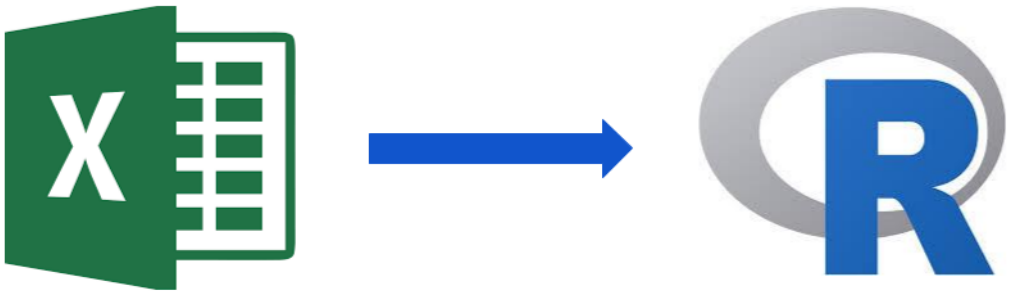
```
##         Name First.Name Age Sex Rank Slams Won Lost Earnings Citizen
## 1    Sampras       Pete  23   M    1     2  74   11 3607.812      US
## 2     Agassi      Andre  24   M    2     1  51   13 1941.667      US
## 3     Becker      Boris  27   M    3     0  48   16 2029.756 Germany
## 4   Bruguera      Sergi  24   M    4     1  65   23 3031.874   Spain
## 5 Ivanisevic      Goran  23   M    5     0  63   26 2060.278 Croatia
## 6       Graf     Steffi  25   F    1     1  58    6 1487.980 Germany
```

Variables containing text are set as character variables with the `stringsAsFactors = FALSE` option, whereas by default they are set as factors.

```
df <- read.table("tennis.txt", header = TRUE, sep = "", dec = ".", stringsAsFactors = FALSE)
str(df)
```

```
## 'data.frame':    10 obs. of  10 variables:
##  $ Name      : chr  "Sampras" "Agassi" "Becker" "Bruguera" ...
##  $ First.Name: chr  "Pete" "Andre" "Boris" "Sergi" ...
##  $ Age       : int  23 24 27 24 23 25 23 22 26 20
##  $ Sex       : chr  "M" "M" "M" "M" ...
##  $ Rank      : int  1 2 3 4 5 1 2 3 4 5
##  $ Slams     : int  2 1 0 1 0 1 2 1 0 0
##  $ Won       : int  74 51 48 65 63 58 74 55 43 45
##  $ Lost      : int  11 13 16 23 26 6 9 15 11 18
##  $ Earnings  : num  3608 1942 2030 3032 2060 ...
##  $ Citizen   : chr  "US" "US" "Germany" "Spain" ...
```

7.2 Microsoft Excel files



One of the most important R package for importing excel files into R is `readxl`. Let us see how it works.

```
require(readxl)
```

To read an existing excel file, the syntax is:

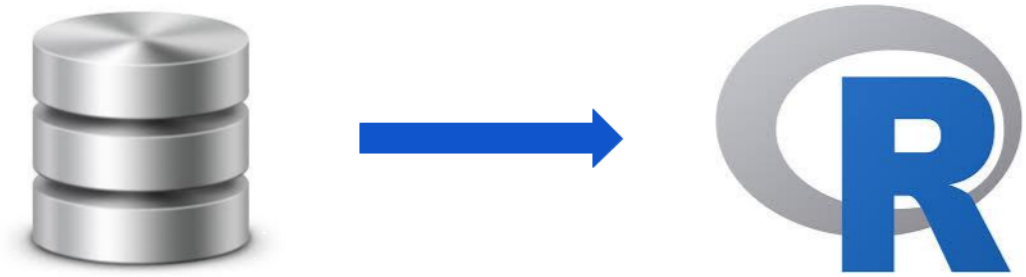
```
# Excel file (with path) to be loaded into R
file_xls <- "./xlsx/newFile.xlsx"
```

```
ds <-read_excel(path = file_xls, sheet = 'Airquality', col_names = TRUE)
head(ds)
```

```
## # A tibble: 6 × 7
##   Ozone Solar.R  Wind  Temp Month   Day isCurrent
##   <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl>     <dbl>
## 1    41     190   7.4    67     5     1         0
## 2    36     118   8.0    72     5     2         0
## 3    12     149  12.6    74     5     3         0
## 4    18     313  11.5    62     5     4         0
## 5    NA      NA  14.3    56     5     5         0
## 6    28      NA  14.9    66     5     6         0
```

`read_excel()` function allows us to read xls and xlsx files, specified in `path` argument. `sheet` argument specifies the sheet to read and `col_names` indicates if the first row has to be used as column names (set as `TRUE`).

7.3 Databases



7.4 ODBC

Open Database Connectivity (ODBC) is a standard programming language interface for accessing database management systems (DBMS). ODBC is independent from database systems and operating systems. An application can use ODBC to query data from a DBMS, regardless of the operating system or DBMS it uses. ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS.

With the `RODBC` package R enables the use of ODBC for interacting with databases. This solution is particularly useful when data occupies much space, is frequently updated or shared by two or more users. In this case, data is kept in the database. With R it is possible to make a query in the database, load data in the R workspace and carry out analyses.

The following code shows some examples of how to use ODBC in a MySQL database. For the following examples to work, it is necessary to modify the following functions with the parameters related to the available MySQL database.

The `odbcConnect()` function establishes the connection to the MySQL database. Its main arguments are: `dsn`, a string containing the name of the data source, `uid` and `pwd`, i.e. the user name and the password for the login.
The `sqlQuery()` function performs queries to the MySQL database. The use of single and double quotation marks require attention. In the following example the query is contained in a string and is delimited by double quotation marks. The strings belonging to the query are delimited by single quotation marks.

Finally, the `odbcClose()` function closes the connection to the database.

```
# RODBC driver ought be configured to work properly
require(RODBC)
conn <- odbcConnect(dsn = "test", uid = "user", pwd = "pass")
sqlQuery(conn, "select * from tbl where gender = 'F'")
odbcClose(conn)
```

7.5 SQLlite

`RSQLite` package embeds the SQLite database engine in R, providing a DBI-compliant interface. SQLite is a public-domain, single-user, very light-weight database engine that implements a decent subset of the SQL 92 standard, including the core table creation, updating, insertion, and selection operations, plus transaction management.

```
require(RSQLite)
```

The function `dbConnect` connect to a SQLite database, or creates it if it doesn't exist:

```
con <- dbConnect(RSQLite::SQLite(), "mtcars.sqlite")
```

To see a list of available SQLite tables:

```
dbListTables(con)
```

```
## [1] "mtcars"
```

or a list of fields in specified table:

```
dbListFields(con, "mtcars")
```

```
##  [1] "row_names" "mpg"       "cyl"       "disp"      "hp"
##  [6] "drat"      "wt"        "qsec"      "vs"        "am"
## [11] "gear"      "carb"
```

The next function mimic its R/S-Plus counterpart `get`, except that it generates code that gets remotely executed in a database engine:

```
dbReadTable(con, "mtcars")
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

The function `dbGetQuery` send query, retrieve results and then clear result set:

```
dbGetQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
```

```
##         row_names  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1      Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 2       Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 3        Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 4        Fiat 128 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 5     Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 6  Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 7   Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 8       Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 9   Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 10   Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 11     Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

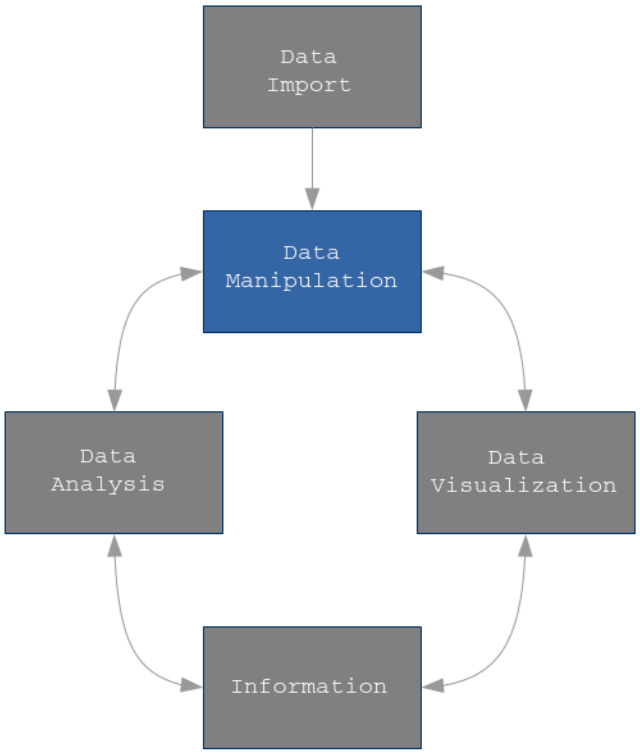
And finally disconnect from the database:

```
dbDisconnect(con)
```

```
## [1] TRUE
```

Chapter 8

Data Manipulation with R



The `dplyr` package for `R` is very powerful for data management since:

- it simplifies how you can think about common data manipulation tasks
- it provides simple "verbs", functions that correspond to the most common data manipulation tasks
- it uses efficient data storage backends, so you spend less time waiting for the computer

```
require(dplyr)
```

The examples of this chapter will refer to `bank` data set which contains information about a direct marketing campaigns of a Portuguese banking institution based on phone calls.

```
require(qdata)
data(bank)
```

```
bank
```

```
## # A tibble: 45,211 × 20
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 45,201 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

In the following paragraphs, we will explore the innovations introduced by `dplyr` to make our lifes easier when dealing with dataframes manipulation tasks.
In particular:

- pipe operator (`%>%`)
- dplyr verbs for data manipulation
- dplyr verbs for combining data

8.1 Pipe operator (%>%)

`dplyr` pipe operator (`%>%`) allows us to pipe the output from one function to the input of another function. The idea of piping is to read the functions from left to right. It is particularly useful with nested functions (reading from the inside to the outside) or with multiple operations.

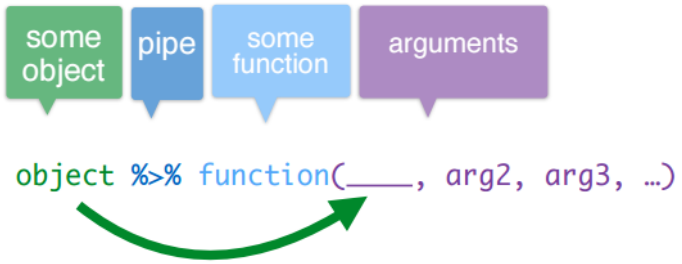


Figure 8.1: Source: www.datacamp.com

Pipes can work with nearly any functions (`dplyr` and not-`dplyr` functions), let us see an example.

Suppose we want to visualize the first rows of `bank` dataframe, by using `head()` function.

Usually we write:

```
head(bank)
```

```
## # A tibble: 6 × 20
##      id   age          job marital education default balance housing
##   <int> <int>       <fctr>  <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1     1    58   management married  tertiary      no    2143     yes
## 2     2    44   technician  single secondary      no      29     yes
## 3     3    33 entrepreneur married secondary      no       2     yes
## 4     4    47  blue-collar married   unknown      no    1506     yes
## 5     5    33      unknown  single   unknown      no       1      no
## 6     6    35   management married  tertiary      no     231     yes
## # ... with 12 more variables: loan <fctr>, contact <fctr>, day <int>,
## #   month <fctr>, year <int>, date <dttm>, duration <int>, campaign <int>,
## #   pdays <int>, previous <int>, poutcome <fctr>, y <fctr>
```

By using %>%, the code becomes:

```
bank %>% head()
```

```
## # A tibble: 6 × 20
##      id   age          job marital education default balance housing
##   <int> <int>       <fctr>  <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1     1    58   management married  tertiary      no    2143     yes
## 2     2    44   technician  single secondary      no      29     yes
## 3     3    33 entrepreneur married secondary      no       2     yes
## 4     4    47  blue-collar married   unknown      no    1506     yes
## 5     5    33      unknown  single   unknown      no       1      no
## 6     6    35   management married  tertiary      no     231     yes
## # ... with 12 more variables: loan <fctr>, contact <fctr>, day <int>,
## #   month <fctr>, year <int>, date <dttm>, duration <int>, campaign <int>,
## #   pdays <int>, previous <int>, poutcome <fctr>, y <fctr>
```

Pipe takes the argument on the left (`bank`) and passes it to the function on the right (`head()`). So you don't need to write the first argument of the function.

Other arguments of the function must be added to the function itself, as usually done. By default `head()` prints the first 6 rows of the dataframe. Suppose we want to print 10 rows, by setting `n` argument to 10:

```
bank %>% head(n=10)
```

```
## # A tibble: 10 × 20
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 12 more variables: loan <fctr>, contact <fctr>, day <int>,
## #   month <fctr>, year <int>, date <dttm>, duration <int>, campaign <int>,
## #   pdays <int>, previous <int>, poutcome <fctr>, y <fctr>
```

8.2 dplyr verbs for data manipulation

`dplyr` aims to provide a function for each basic verb of data manipulation and data discovery.

All these functions are very similar:

- the first argument is a data frame;
- the subsequent arguments describe what to do with it, and you can refer to columns in the data frame directly without using $. Note that the column names must be unquoted;
- the result is a new data frame.

Let us have a look to `dplyr` verbs for data manipulation:

- `select()`: select variables of interest
- `filter()`: filter records of interest
- `arrange()`: reorder the rows
- `mutate()`: add new variables that are functions of existing ones

8.2.1 `select()`

Often you work with large datasets with many columns where only a few are actually of interest to you.

`select()` allows you to rapidly zoom in on a useful subset of columns.

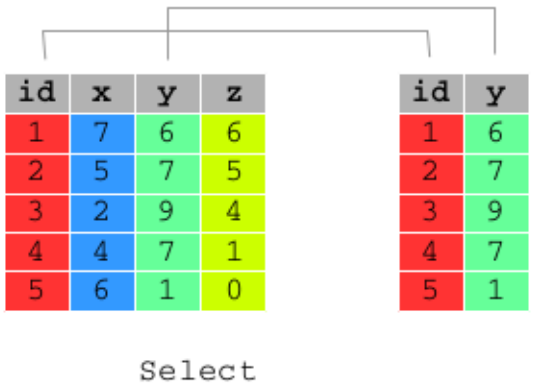


Figure 8.2: select scheme

```
# Select columns: year, month and day of bank data frame
select(bank, year, month, day)
```

```
## # A tibble: 45,211 × 3
##     year  month   day
##    <int> <fctr> <int>
## 1   2008    may     5
## 2   2008    may     5
## 3   2008    may     5
```

```
## 4   2008    may     5
## 5   2008    may     5
## 6   2008    may     5
## 7   2008    may     5
## 8   2008    may     5
## 9   2008    may     5
## 10  2008    may     5
## # ... with 45,201 more rows
```

```
# Select columns: year, month and day of bank data frame
bank %>% select(year:day)
```

```
## # A tibble: 45,211 × 3
##     year  month   day
##    <int> <fctr> <int>
## 1   2008    may     5
## 2   2008    may     5
## 3   2008    may     5
## 4   2008    may     5
## 5   2008    may     5
## 6   2008    may     5
## 7   2008    may     5
## 8   2008    may     5
## 9   2008    may     5
## 10  2008    may     5
## # ... with 45,201 more rows
```

```
# Select all columns of bank data frame apart from: year, month and day
bank %>% select(-(year:day))
```

```
## # A tibble: 45,211 × 17
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 45,201 more rows, and 9 more variables: loan <fctr>,
## #   contact <fctr>, date <dttm>, duration <int>, campaign <int>,
## #   pdays <int>, previous <int>, poutcome <fctr>, y <fctr>
```

You can rename variables with `select()` by using named arguments:

```
# Rename id variable as ID
bank %>% select(ID = id)
```

```
## # A tibble: 45,211 × 1
##       ID
##    <int>
## 1      1
## 2      2
```

```
## 3      3
## 4      4
## 5      5
## 6      6
## 7      7
## 8      8
## 9      9
## 10    10
## # ... with 45,201 more rows
```

8.2.2 filter()

`filter()` allows you to select a subset of the rows of a data frame.

Figure 8.3: filter scheme

```
# filter all calls made to students with balance above 20,000 in bank data frame
filter(bank, job == "student", balance > 20000)
```

```
## # A tibble: 3 × 20
##      id   age     job marital education default balance housing   loan
##   <int> <int>  <fctr>  <fctr>    <fctr>  <fctr>   <int>  <fctr> <fctr>
## 1 31125    24 student  single secondary      no   23878      no     no
## 2 39536    24 student  single secondary      no   23878      no     no
## 3 41923    27 student  single  tertiary      no   24025      no     no
## # ... with 11 more variables: contact <fctr>, day <int>, month <fctr>,
## #   year <int>, date <dttm>, duration <int>, campaign <int>, pdays <int>,
## #   previous <int>, poutcome <fctr>, y <fctr>
```

`filter()` allows you to give it any number of filtering conditions which are joined together with `&` and/or the other operators.

```
# Filter all calls made to student of 18 years in bank data frame
bank %>% filter(age == 18 & job == "student")
```

```
## # A tibble: 12 × 20
```

```
##       id   age     job marital education default balance housing   loan
##    <int> <int>  <fctr>  <fctr>    <fctr>  <fctr>   <int>  <fctr> <fctr>
## 1  40737    18 student  single   primary      no    1944      no     no
## 2  40745    18 student  single   unknown      no     108      no     no
## 3  40888    18 student  single   primary      no     608      no     no
## 4  41223    18 student  single   unknown      no      35      no     no
## 5  41253    18 student  single secondary      no       5      no     no
## 6  41274    18 student  single   unknown      no       3      no     no
## 7  41488    18 student  single   unknown      no     108      no     no
## 8  42147    18 student  single secondary      no     156      no     no
## 9  42275    18 student  single   primary      no     608      no     no
## 10 42955    18 student  single   unknown      no     108      no     no
## 11 43638    18 student  single   unknown      no     348      no     no
## 12 44645    18 student  single   unknown      no     438      no     no
## # ... with 11 more variables: contact <fctr>, day <int>, month <fctr>,
## #   year <int>, date <dttm>, duration <int>, campaign <int>, pdays <int>,
## #   previous <int>, poutcome <fctr>, y <fctr>
```

```
# Filter all calls made to people of 18 or 95 years in bank data frame
bank %>% filter(age == 18 | age == 95)
```

```
## # A tibble: 14 × 20
##       id   age     job  marital education default balance housing   loan
##    <int> <int>  <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr> <fctr>
## 1  33700    95 retired divorced   primary      no    2282      no     no
## 2  40737    18 student   single   primary      no    1944      no     no
## 3  40745    18 student   single   unknown      no     108      no     no
## 4  40888    18 student   single   primary      no     608      no     no
## 5  41223    18 student   single   unknown      no      35      no     no
## 6  41253    18 student   single secondary      no       5      no     no
## 7  41274    18 student   single   unknown      no       3      no     no
## 8  41488    18 student   single   unknown      no     108      no     no
## 9  41664    95 retired  married secondary      no       0      no     no
## 10 42147    18 student   single secondary      no     156      no     no
## 11 42275    18 student   single   primary      no     608      no     no
## 12 42955    18 student   single   unknown      no     108      no     no
## 13 43638    18 student   single   unknown      no     348      no     no
## 14 44645    18 student   single   unknown      no     438      no     no
## # ... with 11 more variables: contact <fctr>, day <int>, month <fctr>,
## #   year <int>, date <dttm>, duration <int>, campaign <int>, pdays <int>,
## #   previous <int>, poutcome <fctr>, y <fctr>
```

`filter()` can be used also with `%in%` to establish conditions under which filter:

```
# Filter all calls made to people of 18 or 95 years in bank data frame
bank %>% filter(age %in% c(18,95))
```

```
## # A tibble: 14 × 20
##       id   age     job  marital education default balance housing   loan
##    <int> <int>  <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr> <fctr>
## 1  33700    95 retired divorced   primary      no    2282      no     no
## 2  40737    18 student   single   primary      no    1944      no     no
## 3  40745    18 student   single   unknown      no     108      no     no
## 4  40888    18 student   single   primary      no     608      no     no
## 5  41223    18 student   single   unknown      no      35      no     no
## 6  41253    18 student   single secondary      no       5      no     no
```

```
## 7  41274    18 student   single   unknown      no       3      no     no
## 8  41488    18 student   single   unknown      no     108      no     no
## 9  41664    95 retired  married secondary      no       0      no     no
## 10 42147    18 student   single secondary      no     156      no     no
## 11 42275    18 student   single   primary      no     608      no     no
## 12 42955    18 student   single   unknown      no     108      no     no
## 13 43638    18 student   single   unknown      no     348      no     no
## 14 44645    18 student   single   unknown      no     438      no     no
## # ... with 11 more variables: contact <fctr>, day <int>, month <fctr>,
## #   year <int>, date <dttm>, duration <int>, campaign <int>, pdays <int>,
## #   previous <int>, poutcome <fctr>, y <fctr>
```

```
# Filter all calls made to people whose job is admin. or technician in bank data frame
bank %>% filter(job %in% c("admin.","technician"))
```

```
## # A tibble: 12,768 × 20
##       id   age        job  marital education default balance housing
##    <int> <int>     <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      2    44 technician   single secondary      no      29     yes
## 2     10    43 technician   single secondary      no     593     yes
## 3     11    41     admin. divorced secondary      no     270     yes
## 4     12    29     admin.   single secondary      no     390     yes
## 5     13    53 technician  married secondary      no       6     yes
## 6     14    58 technician  married   unknown      no      71     yes
## 7     17    45     admin.   single   unknown      no      13     yes
## 8     26    44     admin.  married secondary      no    -372     yes
## 9     30    36 technician   single secondary      no     265     yes
## 10    31    57 technician  married secondary      no     839      no
## # ... with 12,758 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

```
# Filter all calls made to people whose job is admin. or technician in bank data frame
bank %>% filter(job == "admin." | job == "technician")
```

```
## # A tibble: 12,768 × 20
##       id   age        job  marital education default balance housing
##    <int> <int>     <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      2    44 technician   single secondary      no      29     yes
## 2     10    43 technician   single secondary      no     593     yes
## 3     11    41     admin. divorced secondary      no     270     yes
## 4     12    29     admin.   single secondary      no     390     yes
## 5     13    53 technician  married secondary      no       6     yes
## 6     14    58 technician  married   unknown      no      71     yes
## 7     17    45     admin.   single   unknown      no      13     yes
## 8     26    44     admin.  married secondary      no    -372     yes
## 9     30    36 technician   single secondary      no     265     yes
## 10    31    57 technician  married secondary      no     839      no
## # ... with 12,758 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

8.2.3 `arrange()`

Function `arrange()` reorders a data frame by one or more variables. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

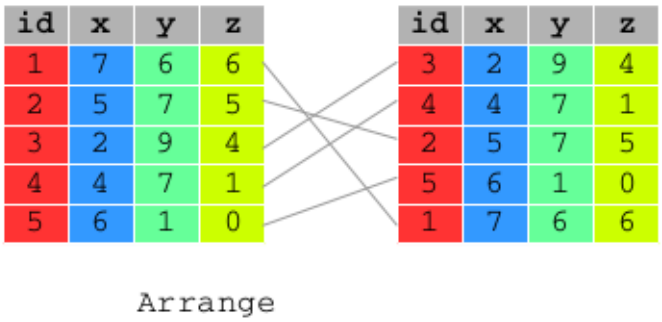


Figure 8.4: arrange scheme

```
# Order `bank` data frame by the balance of the account in ascending order
arrange(bank, balance)
```

```
## # A tibble: 45,211 × 20
##       id   age           job  marital education default balance housing
##    <int> <int>        <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1  12910    26   blue-collar   single secondary     yes   -8019      no
## 2  15683    49    management  married  tertiary     yes   -6847      no
## 3  38737    60    management divorced  tertiary      no   -4057     yes
## 4   7414    43    management  married  tertiary     yes   -3372     yes
## 5   1897    57 self-employed  married  tertiary     yes   -3313     yes
## 6  32714    39 self-employed  married  tertiary      no   -3058     yes
## 7  18574    40    technician  married  tertiary     yes   -2827     yes
## 8  31510    52    management  married  tertiary      no   -2712     yes
## 9  25120    49   blue-collar   single   primary     yes   -2604     yes
## 10 14435    51    management divorced  tertiary      no   -2282     yes
## # ... with 45,201 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

```
# Order `bank` data frame by the balance of the account in descending order
bank %>% arrange(desc(balance))
```

```
## # A tibble: 45,211 × 20
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1  39990    51   management   single  tertiary      no  102127      no
## 2  26228    59   management  married  tertiary      no   98417      no
## 3  42559    84      retired  married secondary      no   81204      no
## 4  43394    84      retired  married secondary      no   81204      no
```

```
## 5  41694    60      retired  married   primary      no   71188      no
## 6  19786    56   management divorced  tertiary      no   66721      no
## 7  21193    52  blue-collar  married   primary      no   66653      no
## 8  19421    59       admin.  married   unknown      no   64343      no
## 9  41375    32 entrepreneur   single  tertiary      no   59649      no
## 10 12927    56  blue-collar  married secondary      no   58932      no
## # ... with 45,201 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

```
# Order `bank` data frame by age of the client and by the balance of the account in descending order
bank %>% arrange(age, desc(balance))
```

```
## # A tibble: 45,211 × 20
##       id   age     job marital education default balance housing   loan
##    <int> <int>  <fctr>  <fctr>    <fctr>  <fctr>   <int>  <fctr> <fctr>
## 1  40737    18 student  single   primary      no    1944      no     no
## 2  40888    18 student  single   primary      no     608      no     no
## 3  42275    18 student  single   primary      no     608      no     no
## 4  44645    18 student  single   unknown      no     438      no     no
## 5  43638    18 student  single   unknown      no     348      no     no
## 6  42147    18 student  single secondary      no     156      no     no
## 7  40745    18 student  single   unknown      no     108      no     no
## 8  41488    18 student  single   unknown      no     108      no     no
## 9  42955    18 student  single   unknown      no     108      no     no
## 10 41223    18 student  single   unknown      no      35      no     no
## # ... with 45,201 more rows, and 11 more variables: contact <fctr>,
## #   day <int>, month <fctr>, year <int>, date <dttm>, duration <int>,
## #   campaign <int>, pdays <int>, previous <int>, poutcome <fctr>, y <fctr>
```

8.2.4 mutate()

As well as selecting from the set of existing columns, it's often useful to add new columns that are functions of existing ones. This is the job of `mutate()`:

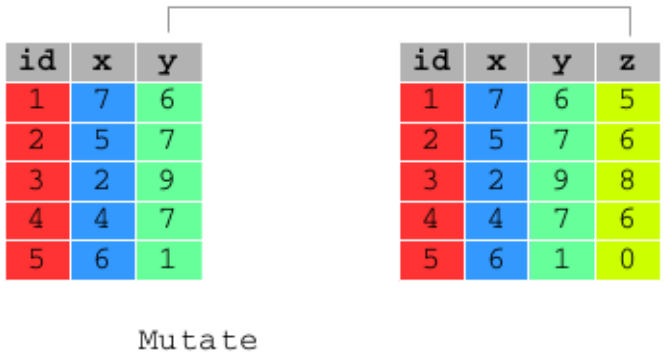


Figure 8.5: mutate scheme

```
# generate a variable indicating the total number of times each person has been contacted
# during this campaign and during the previous ones
mutate(bank, contacts_n = campaign + previous)
```

```
## # A tibble: 45,211 × 21
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 45,201 more rows, and 13 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>, contacts_n <int>
```

`mutate()` allows you to refer to columns that you just created:

```
# generate two variable: one indicating the year of birth and one the year of birth without century
bank %>% mutate(year_of_birth = year - age, year_of_birth_no_century = year_of_birth - 1900)
```

```
## # A tibble: 45,211 × 22
##       id   age          job  marital education default balance housing
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
```

```
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 45,201 more rows, and 14 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>, year_of_birth <int>,
## #   year_of_birth_no_century <dbl>
```

8.3 dplyr verbs for combining data

Very often you will have to deal with many tables that contribute to the analysis you are performing and you need flexible tools to combine them. Supposing that the two tables are already in a tidy form: the rows are observations and the columns are variables, `dplyr` provides **mutating joins**, which add new variables to one table from matching rows in another.

There are four types of mutating join, which differ in their behaviour when a match is not found.

- `inner_join(x, y)`
- `left_join(x, y)`
- `right_join(x, y)`
- `outer_join(x, y)`

All these verbs work similarly:

- the first two arguments, `x` and `y`, provide the tables to combine
- the output is always a new table with the same type as `x`

For the next examples we will consider these two small data frames:

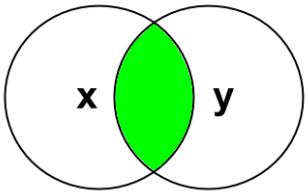
```
df1 <- data.frame(id = 1:4, x1 = letters[1:4])
df1
```

```
##   id x1
## 1  1  a
## 2  2  b
## 3  3  c
## 4  4  d
```

```
df2 <- data.frame(id = 3:5, x2 = letters[3:5])
df2
```

```
##   id x2
## 1  3  c
## 2  4  d
## 3  5  e
```

8.3.1 inner_join(x, y)



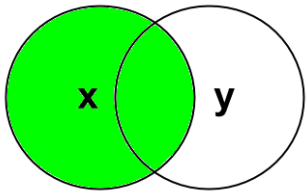
`inner_join(x, y)` only includes observations that match in both x and y:

```
inner_join(df1, df2)
```

```
## Joining, by = "id"
```

```
##   id x1 x2
## 1  3  c  c
## 2  4  d  d
```

8.3.2 left_join(x, y)



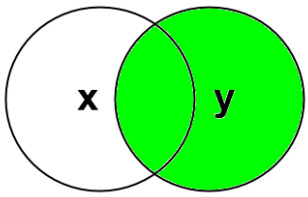
`left_join(x, y)` includes all observations in `x`, regardless of whether they match or not. This is the most commonly used join because it ensures that you don't lose observations from your primary table:

```
left_join(df1, df2)
```

```
## Joining, by = "id"
```

```
##   id x1   x2
## 1  1  a <NA>
## 2  2  b <NA>
## 3  3  c    c
## 4  4  d    d
```

8.3.3 right_join(x, y)



`right_join(x, y)` includes all observations in `y`. It's equivalent to `left_join(y, x)`, but the columns will be ordered differently:

```
right_join(df1, df2)
```

```
## Joining, by = "id"
```

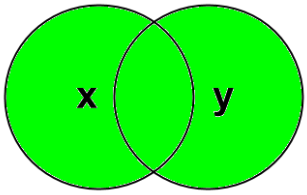
```
##   id   x1 x2
## 1  3    c  c
## 2  4    d  d
## 3  5 <NA>  e
```

```
left_join(df2, df1)
```

```
## Joining, by = "id"
```

```
##   id x2   x1
## 1  3  c    c
## 2  4  d    d
## 3  5  e <NA>
```

8.3.4 full_join()



`full_join()` includes all observations from `x` and `y`:

```
full_join(df1, df2)
```

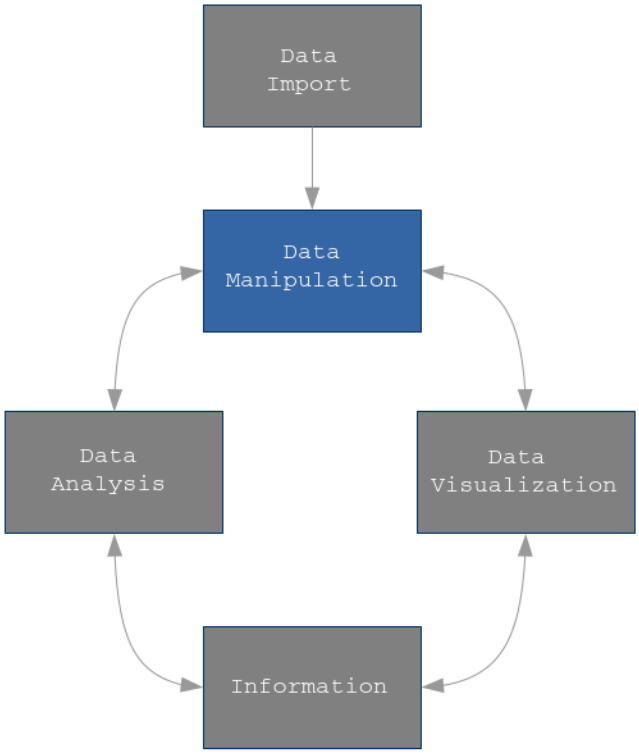
```
## Joining, by = "id"
```

```
##   id   x1   x2
## 1  1    a <NA>
## 2  2    b <NA>
## 3  3    c    c
## 4  4    d    d
## 5  5 <NA>    e
```

The left, right and full joins are collectively know as outer joins. When a row doesn't match in an outer join, the new variables are filled in with missing values.

Chapter 9

Data Discovery with R



`dplyr` provides also the tools for discovering your data. We are talking about `summarise()` verb, which collapse many values into a summary and about other best practices that allows us to combine together multiple operations.

```
require(dplyr)
```

In the following examples, we will refer to `bank` data set.

```
require(qdata)
data(bank)
bank
```

```
## # A tibble: 45,211 × 20
##       id   age          job  marital education default balance housing
```

```
##    <int> <int>       <fctr>   <fctr>    <fctr>  <fctr>   <int>  <fctr>
## 1      1    58   management  married  tertiary      no    2143     yes
## 2      2    44   technician   single secondary      no      29     yes
## 3      3    33 entrepreneur  married secondary      no       2     yes
## 4      4    47  blue-collar  married   unknown      no    1506     yes
## 5      5    33      unknown   single   unknown      no       1      no
## 6      6    35   management  married  tertiary      no     231     yes
## 7      7    28   management   single  tertiary      no     447     yes
## 8      8    42 entrepreneur divorced  tertiary     yes       2     yes
## 9      9    58      retired  married   primary      no     121     yes
## 10    10    43   technician   single secondary      no     593     yes
## # ... with 45,201 more rows, and 12 more variables: loan <fctr>,
## #   contact <fctr>, day <int>, month <fctr>, year <int>, date <dttm>,
## #   duration <int>, campaign <int>, pdays <int>, previous <int>,
## #   poutcome <fctr>, y <fctr>
```

9.1 Descriptive statistics with `summarise()` and `group_by()`

`summarise()` collapses a data frame to a single row and allows us to compute descriptive statistics. Let us see some examples:

```
# Compute the mean of balance of the accounts
summarise(bank, mean_balance = mean(balance, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   mean_balance
##          <dbl>
## 1     1362.272
```

```
# Compute the sum of balance of the accounts
bank %>% summarise(max_balance = sum(balance, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##   max_balance
##         <int>
## 1    61589682
```

```
# Compute the minimum and the maximum balance of the accounts
bank %>% summarise(max_balance = max(balance, na.rm = TRUE), min_balance = min(balance, na.rm = TRUE))
```

```
## # A tibble: 1 × 2
##   max_balance min_balance
##         <int>       <int>
## 1      102127       -8019
```

```
# Compute the summary (number of obs, minimum, first quartile, median, mean, third quartile,
# maximum and standard deviation) of balance of the accounts
bank %>% summarise(n_obs = n(),
                   min=min(balance, na.rm = TRUE),
                   first_q=quantile(balance, prob = 0.25, na.rm = TRUE),
                   median=median(balance, na.rm = TRUE),
                   mean=mean(balance, na.rm =TRUE),
                   third_q=quantile(balance, prob = 0.75, na.rm = TRUE),
                   max=max(balance, na.rm = TRUE),
                   sd=sd(balance, na.rm =TRUE))
```

```
## # A tibble: 1 × 8
##   n_obs   min first_q median     mean third_q    max       sd
##   <int> <int>   <dbl>  <int>    <dbl>   <dbl>  <int>    <dbl>
## 1 45211 -8019      72    448 1362.272    1428 102127 3044.766
```

Descriptive statistics can be computed also by groups, by using `group_by()`, which is a `dplyr` function that allows us to group observations according to the levels of a variable/s.

```
# Compute the mean of balance of the accounts by job
bank %>%
  group_by(job) %>%
  summarise(mean_balance = mean(balance, na.rm = TRUE))
```

```
## # A tibble: 12 × 2
##              job mean_balance
##           <fctr>        <dbl>
## 1         admin.    1135.8389
## 2    blue-collar    1078.8267
## 3   entrepreneur    1521.4701
## 4      housemaid    1392.3952
## 5     management    1763.6168
## 6        retired    1984.2151
## 7  self-employed    1647.9709
## 8       services     997.0881
## 9        student    1388.0608
## 10    technician    1252.6321
## 11    unemployed    1521.7460
## 12       unknown    1772.3576
```

```
# Compute the summary (number of obs, minimum, first quartile, median, mean, third quantile,
# maximum and standard deviation) of balance of the accounts by job
bank %>%
  group_by(job) %>%
  summarise(n_obs = n(),
            min=min(balance, na.rm = TRUE),
            first_q=quantile(balance, prob = 0.25, na.rm = TRUE),
            median=median(balance, na.rm = TRUE),
            mean=mean(balance, na.rm =TRUE),
            third_q=quantile(balance, prob = 0.75, na.rm = TRUE),
            max=max(balance, na.rm = TRUE),
            sd=sd(balance, na.rm =TRUE))
```

```
## # A tibble: 12 × 9
##              job n_obs   min first_q median      mean third_q    max
##           <fctr> <int> <int>   <dbl>  <dbl>     <dbl>   <dbl>  <int>
## 1         admin.  5171 -1601   63.00  396.0 1135.8389 1203.00  64343
## 2    blue-collar  9732 -8019   55.00  388.0 1078.8267 1203.00  66653
## 3   entrepreneur  1487 -2082   44.50  352.0 1521.4701 1341.00  59649
## 4      housemaid  1240 -1941   57.75  406.0 1392.3952 1382.75  45141
## 5     management  9458 -6847   98.00  572.0 1763.6168 1825.00 102127
## 6        retired  2264 -1598  164.50  787.0 1984.2151 2309.00  81204
## 7  self-employed  1579 -3313  120.00  526.0 1647.9709 1603.50  52587
## 8       services  4154 -2122   35.00  339.5  997.0881 1071.75  57435
## 9        student   938  -679  148.25  502.0 1388.0608 1579.75  24025
## 10    technician  7597 -2827   61.00  421.0 1252.6321 1327.00  45248
## 11    unemployed  1303 -1270   94.00  529.0 1521.7460 1603.50  44134
```

```
## 12       unknown   288  -295  170.75  677.0 1772.3576 2165.50  19706
## # ... with 1 more variables: sd <dbl>
```

9.2 Multiple operations

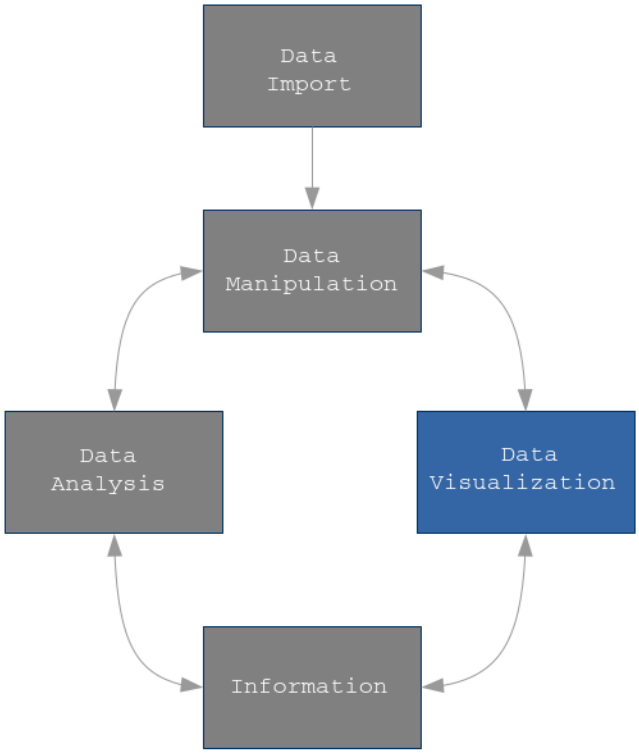
You can also chain together multiple operations to achieve a complex result. Let us have a look!

```
# Compute the mean of balance of the accounts and the number of obs by job for people not older than 40
# sort the result in ascending order
bank %>%
  filter(age < 40) %>%
  group_by(job) %>%
  summarise(n_obs =n(),
            mean_balance = mean(balance, na.rm = TRUE)) %>%
  arrange(mean_balance)
```

```
## # A tibble: 12 × 3
##              job n_obs mean_balance
##           <fctr> <int>        <dbl>
## 1        retired    25     445.8800
## 2       services  2419     827.6701
## 3         admin.  2924     936.2401
## 4    blue-collar  5064     951.9682
## 5   entrepreneur   647    1153.1453
## 6     technician  4376    1198.5599
## 7      housemaid   363    1346.0992
## 8     unemployed   631    1355.1886
## 9  self-employed   820    1379.8793
## 10       student   924    1385.0801
## 11    management  5101    1548.9508
## 12       unknown    68    1777.5882
```

Chapter 10

Data Visualization with R



Visualizing data is crucial in today's world. Without powerful visualizations, it is almost impossible to create and narrate stories on data. These stories help us build strategies and make intelligent business decisions.

`ggplot2` is a data visualization package which has become a synonym for data visualization in R.

```
require(ggplot2)
```

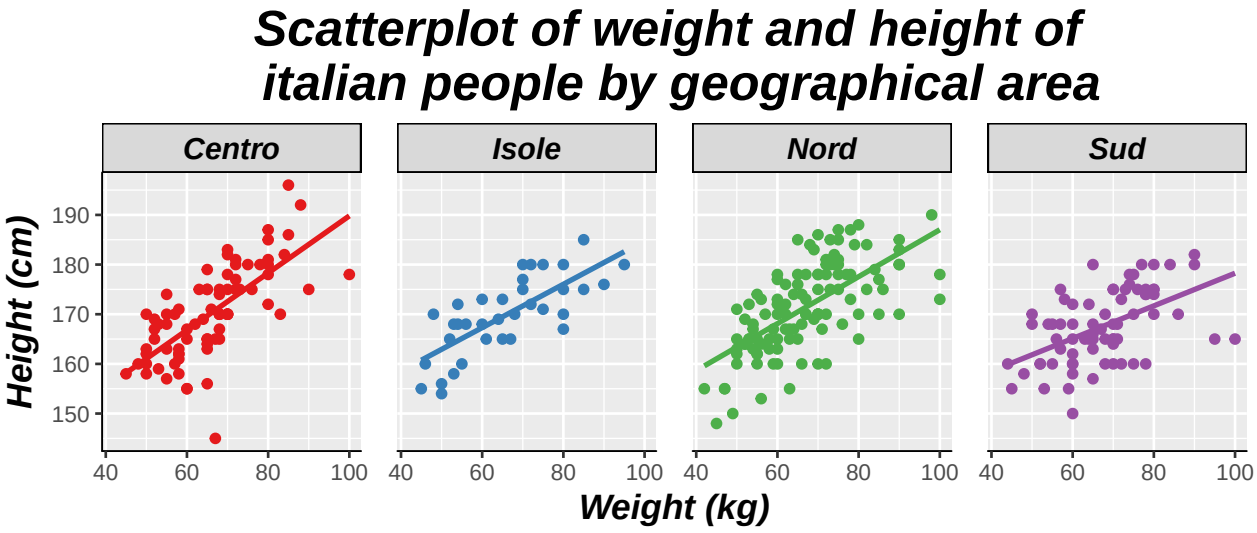
Created by Hadley Wickham in 2005, `ggplot2` is an implementation of Leland Wilkinson's Grammar of Graphics, a general scheme for data visualization which breaks up graphs into semantic independent components, such as scales and layers, that can be composed in many different ways. This makes `ggplot2` very powerful, because there are no limitations due to a set of pre-specified graphics, so it is possible to create new graphics that are precisely tailored for the problem in analysis.

10.1 An overview of ggplot2 grammar

Let us consider `people` dataset, included in `qdata` package. `people` dataset contains informations about weight, height, gender and geographical area of 300 italian people.

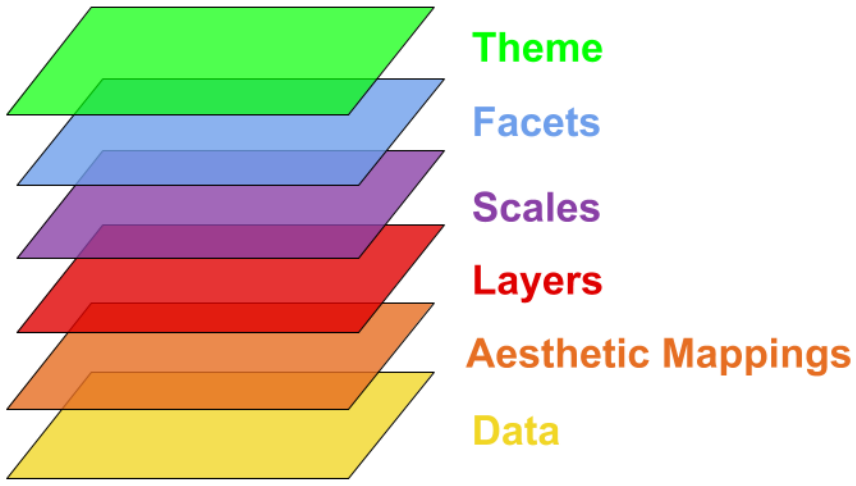
```
require(qdata)
```

Suppose we want to visualize the relationship between weight and height of italian people according to the different geographical area, by using a **scatterplot**:



The previous plot is composed of building blocks that are added to the plot one after the other.

The complete scheme of the most important building blocks of `ggplot2` grammar is displayed in the following figure:



The scheme must be read from bottom to top. Starting from bottom, the first three building blocks (Data , Aestethic Mappings and Layers) are fundamental to build a simple plot, indeed they are called **"key"** building blocks. The remaining building blocks (Scales , Coordinates , Facets and Themes) allow us to build a complex plot and to customize it; their use and order is not compulsory.

Let us briefly describe the task of each element of the scheme and how it helps build the previous plot:

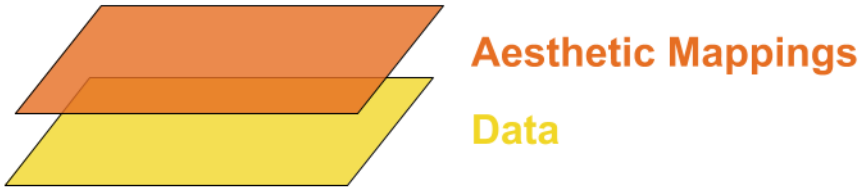
1. Data : the dataset that we want to visualize


```
# people dataset
data(people)
```

```
head(people)
```

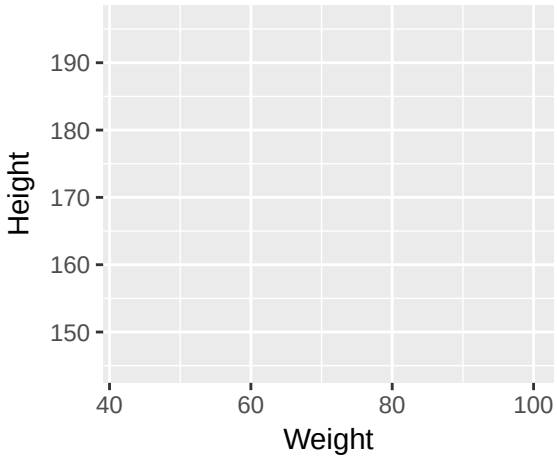
```
## # A tibble: 6 × 4
##   Gender   Area Weight Height
##   <fctr> <fctr>  <int>  <int>
## 1 Female  Isole     54    168
## 2 Female   Nord     61    171
## 3   Male    Sud     68    170
## 4 Female   Nord     52    164
## 5   Male   Nord     75    181
## 6   Male   Nord     77    178
```

2. Aestethic Mappings : describes how variables in the data are mapped to aestethic attributes that you can perceive



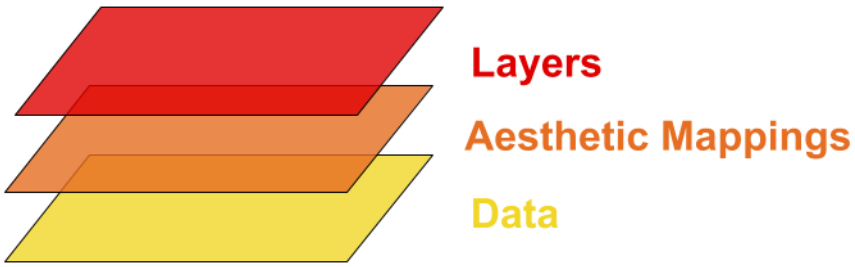
Gender	Area	Weight	Height
		x	y

```
ggplot(data = people, aes(x = Weight, y = Height))
```

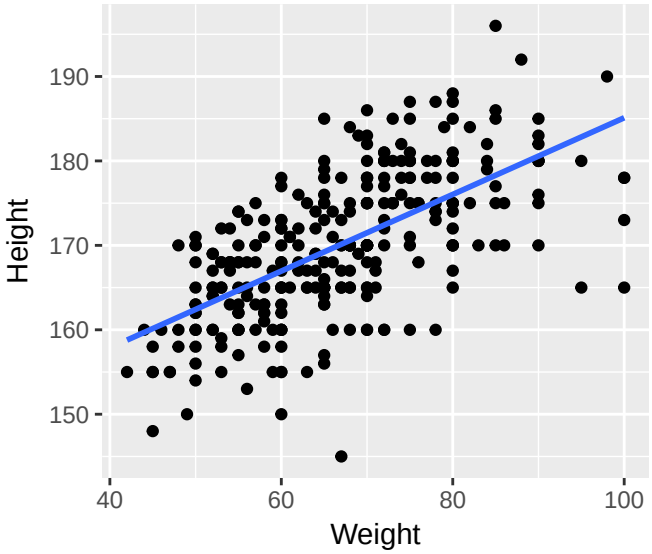


3. Layers : are made up by geometric elements and statistical transformations. In details, geometric objects (geoms) represent what we actually see on the plot: points, lines, polygons, etc. Statistical

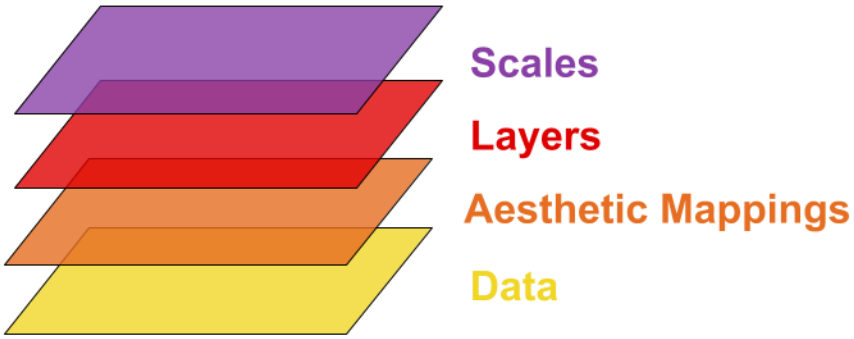
transformations (`stats`) summarise data in many useful ways. For example, binning and counting observations to create an histogram, or summarising a 2d relationship with a linear model.



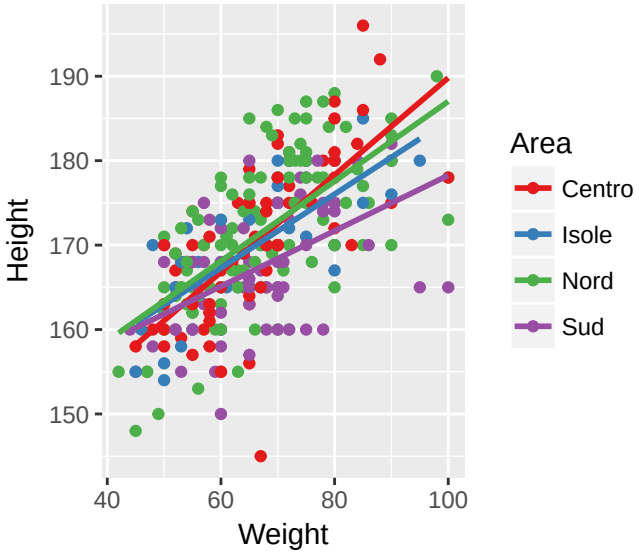

```r
# Scatterplot of the relationship between weight and height with regression line
ggplot(people, aes(x = Weight, y = Height)) +
  geom_point() + # layer 1 (draw points)
  stat_smooth(method = "lm", se = FALSE) # layer 2 (draw regression line)
```



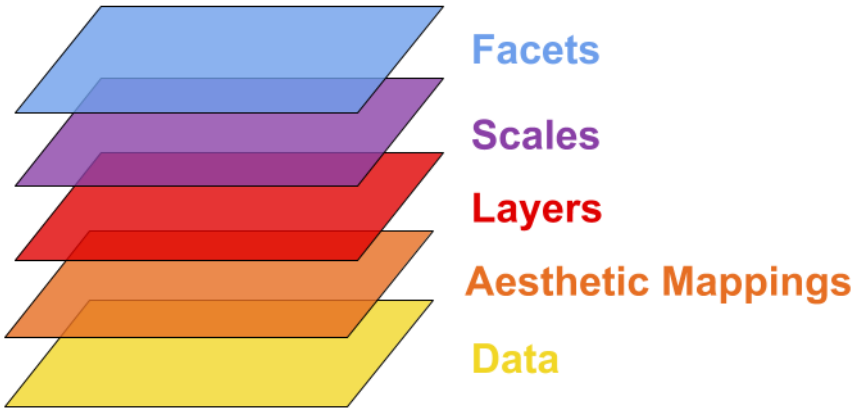
4. Scales : map values in the data space to values in an aesthetic space, whether it be colour, or size or shape. Scales draw a legend on axes, which provide an inverse mapping to make it possible to read the original data values from the graph. Scales are closely related to aestethics mapped.



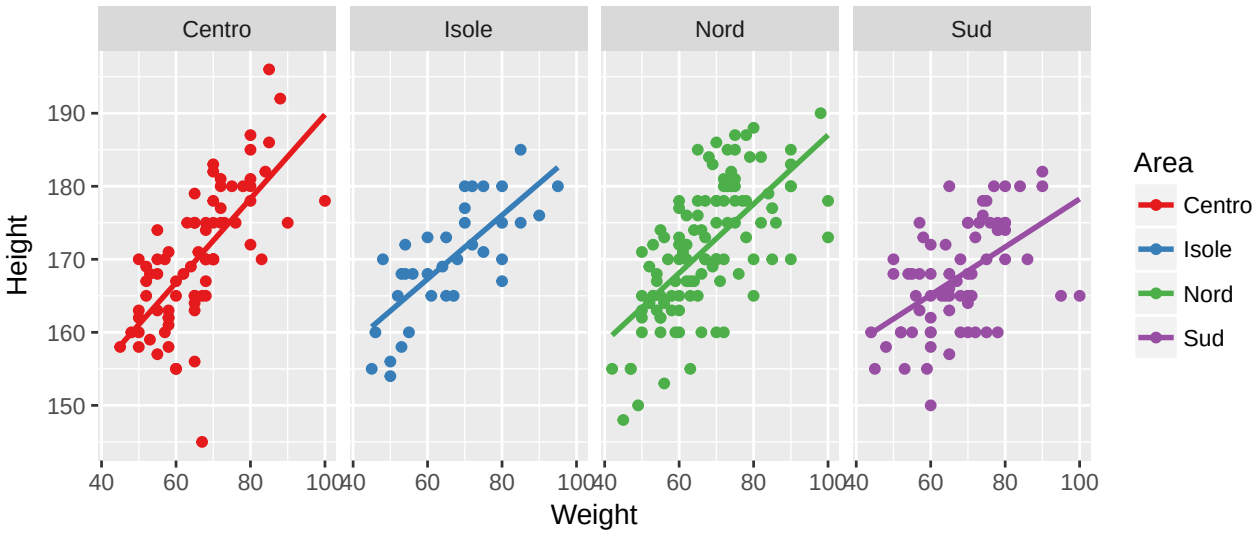
```
# map Area to colour in aes() and change the default colours of colour scale
ggplot(people, aes(x = Weight, y = Height, colour = Area)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  scale_colour_brewer(palette="Set1")
```



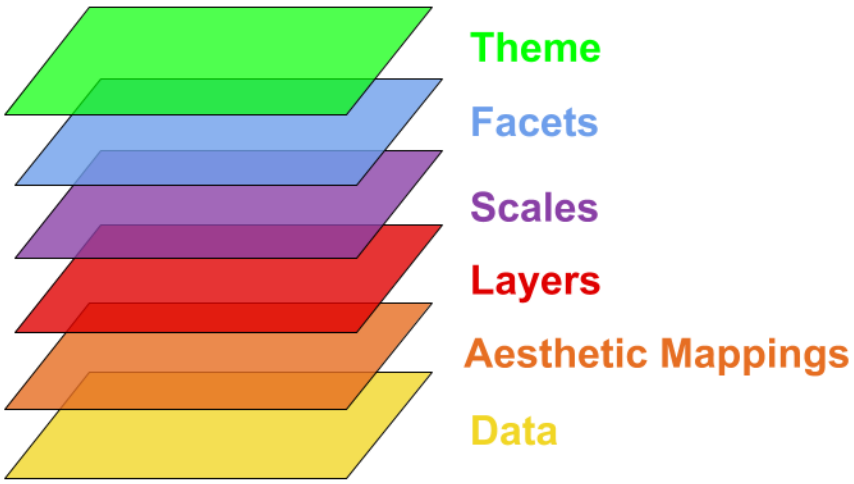
5. Facets : describes how to break up the data into subset and how to display those subsets as small multiples.



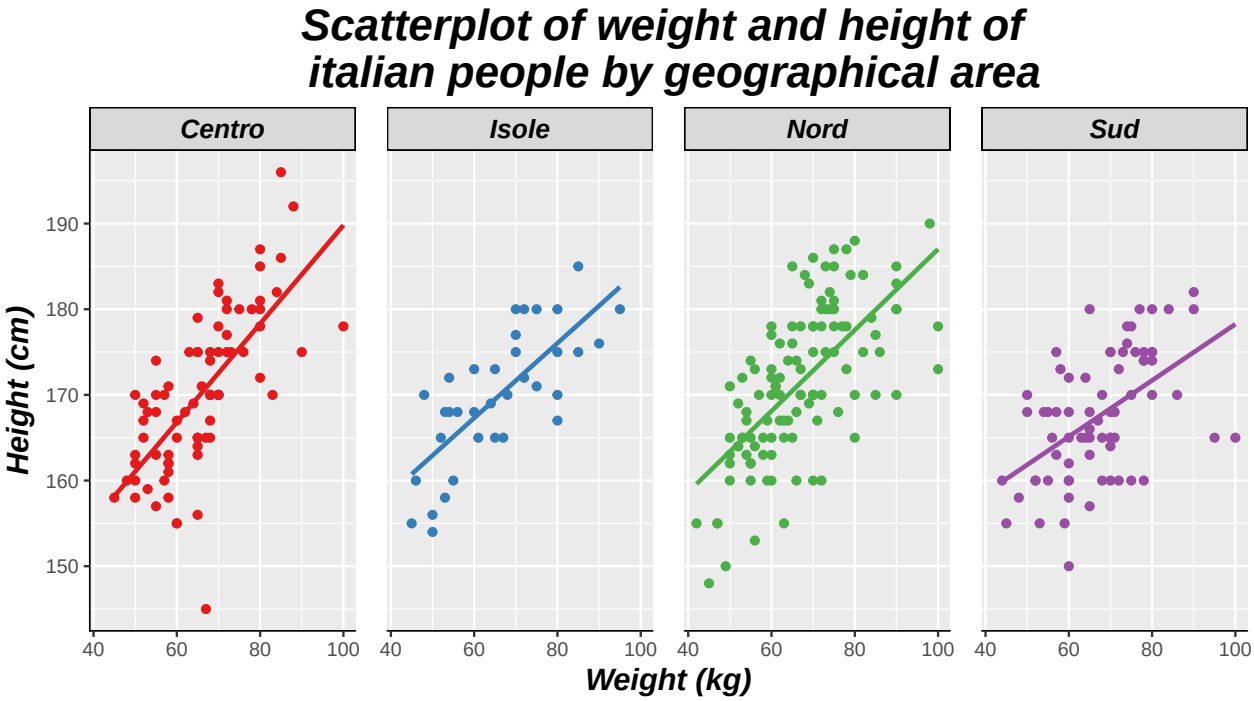

```
# Generate a plot for each geographical area
ggplot(people, aes(x = Weight, y = Height, colour = Area)) +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE) +
  scale_colour_brewer(palette="Set1") +
  facet_grid(. ~ Area)
```



6. Themes : controls all non-data elements of the plot, like the font size, background colour, etc.




```
# Customize the appearance of the plot
ggplot(people, aes(x = Weight, y = Height, colour = Area)) +
  geom_point() +
  stat_smooth(method = "lm", se = F) +
  scale_colour_brewer(palette="Set1") +
  facet_grid(. ~ Area) +
  ggtitle("Scatterplot of weight and height of \n italian people by geographical area") +
    xlab("Weight (kg)") +
    ylab("Height (cm)") +
  theme(plot.background = element_blank(),
    axis.line.x = element_line(colour = "black"),
    axis.line.y = element_line(colour = "black"),
    axis.title = element_text(colour = "black", size = 14, face = "bold.italic"),
    strip.background = element_rect(colour = "black"),
    strip.text = element_text(colour = "black", face = "bold.italic", size = 12),
    plot.title = element_text(colour = "black", size = 20, face = "bold.italic", hjust = 0.5),
    panel.spacing = unit(1, "lines"),
    legend.position="none")
```

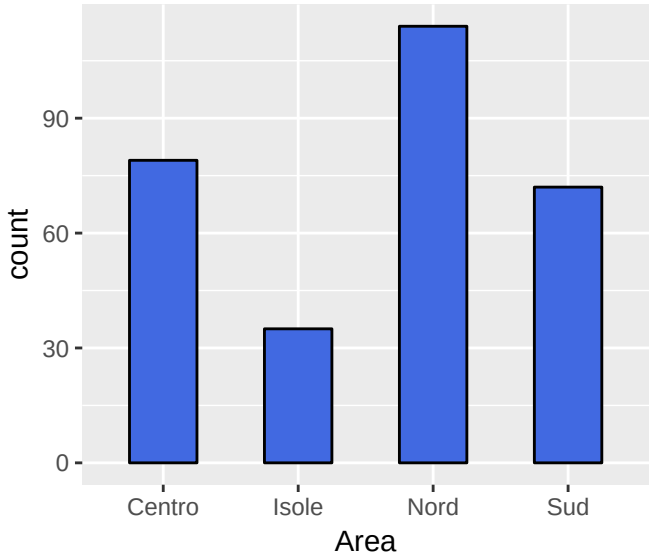

Scatterplot of weight and height of italian people by geographical area
Centro
Isole
Nord
Sud
Height (cm)
190
180
170
160
150
40
60
80
100
Weight (kg)

10.2 Other plot types

The building block scheme previously seen can help us understanding how to build other plot types. We will focus on the first three building blocks of the scheme (data, aestethic mappings and layers).

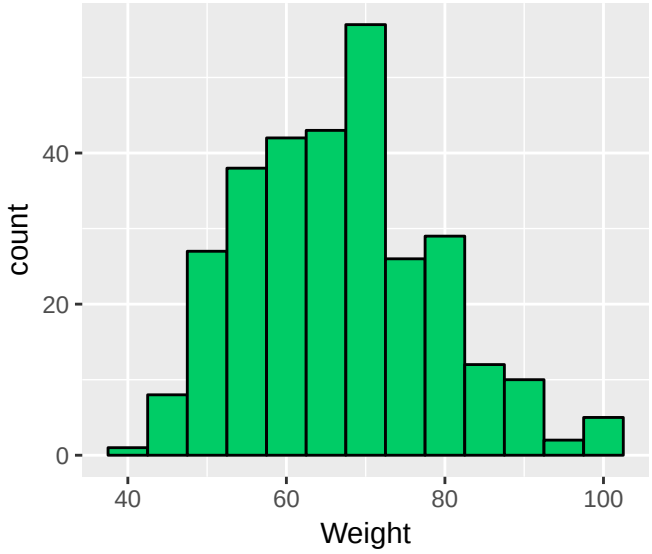
- **Barplot**, which is used to show the count of each case of a categorial variable. Suppose we want to analyze the count of people by geographical area:

```
# base plot: key building blocks (data, aes, layer)
ggplot(data = people, mapping = aes(x = Area)) +
  geom_bar(fill = "royalblue", colour = "black", width = 0.5)
```



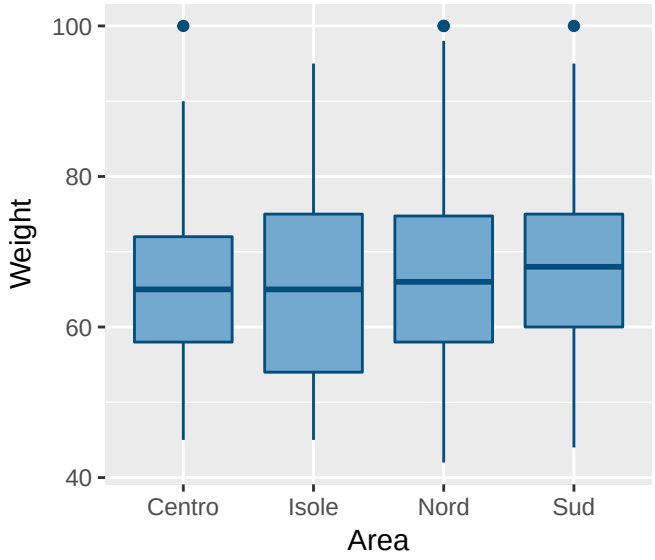
- **Histogram**, which is used to summarize a continuous variable into classes. Suppose we want to analyze the distribution of people weight:

```
# base plot: key building blocks (data, aes, layer)
ggplot(data=people, mapping=aes(x=Weight)) +
  geom_histogram(fill="#00cc66", colour= "#000000", binwidth=5)
```



- **Boxplot**, which is used to draw a data distribution. Supposing you are interested in the differences of weight accordingly to geographical area:

```
# base plot: key building blocks (data, aes, layer)
ggplot(data=people, aes(x=Area, y=Weight)) +
  geom_boxplot(fill="#74a9cf", colour="#034e7b")
```



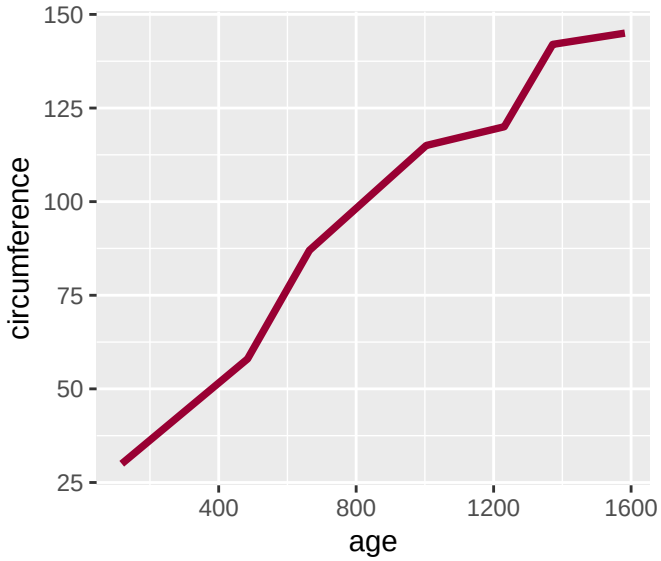
- **Lineplot**, used to display how one continuous variable, on the y-axis, changes in relation to another continuous variable, on the x-axis. For this example we consider `orange` data, included in `qdata` package. `orange` contains information about the growth of 5 Orange Trees, according to their trunk circumferences.

```
data(orange)
head(orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

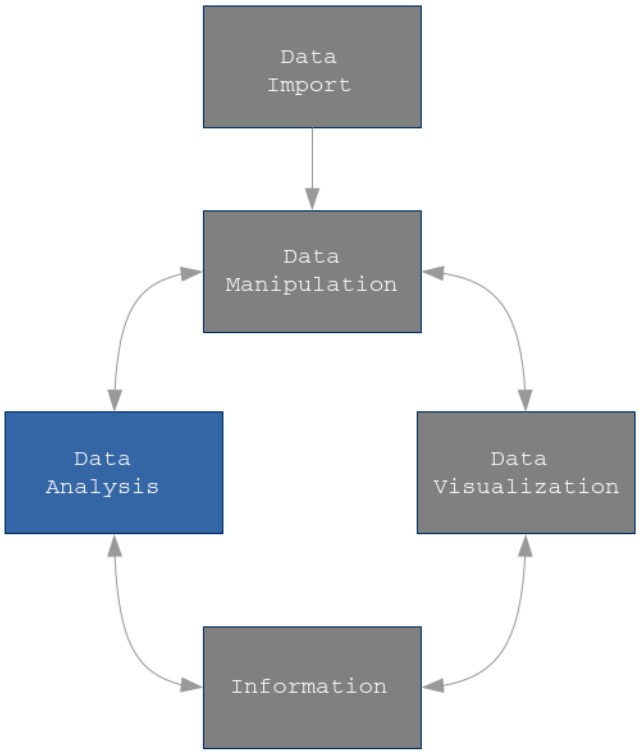
Suppose we want to represent the growth of one tree:

```
require(dplyr)
# base plot of 1 tree: key components(data, aes, layer)
ggplot(data=orange %>% filter(Tree==1), mapping=aes(x=age, y=circumference)) +
  geom_line(colour= "#990033", size=1.3)
```


150
125
100
75
50
25
circumference
400
800
1200
1600
age

Chapter 11

Statistical Models with R



Data can be analysed by regression models with R. Regression is an analysis that attempts to determine the strength of the relationship between one dependent variable, y, and a series of other changing variables, x.

There are lots of different types of regression models in statistics but R mantains a coherent syntax for the estimation of all of them. Indeed, the common interface to fit a model in R is made of a call to the corresponding function with arguments `formula` and `data`.

The `lm` and `aov` functions are used in R to fit respectively linear regression and analysis of variance model and their syntax is:

```
linear_model <- lm(formula, data)
anova_model <- aov(formula, data)
```

`formula` argument is a symbolic description of the model to be fitted, which has the form: `response`

`variable predictor variables`. The variables involved in `formula` should be columns of a dataframe specified in `data` argument.

The resulting object (`linear_model` or `anova_model`) is a list of elements containing information about regression results. This information can be investigated by the following functions:

Expression	Description
`coef(obj)`	regression coefficients
`resid(obj)`	residuals
`fitted(obj)`	fitted values
`summary(obj)`	analysis summary
`predict(obj, newdata = ndat)`	predict for newdata
`deviance(obj)`	residual sum of squares

Let us see an example of regression analysis.

11.0.1 Drug Dosage and Reaction Time

In an experiment to investigate the effect of a depressant drug, the reaction times of ten males rats to a certain stimulus were measured after a specified dose of the drug had been administer to each rat. The results were as follows:

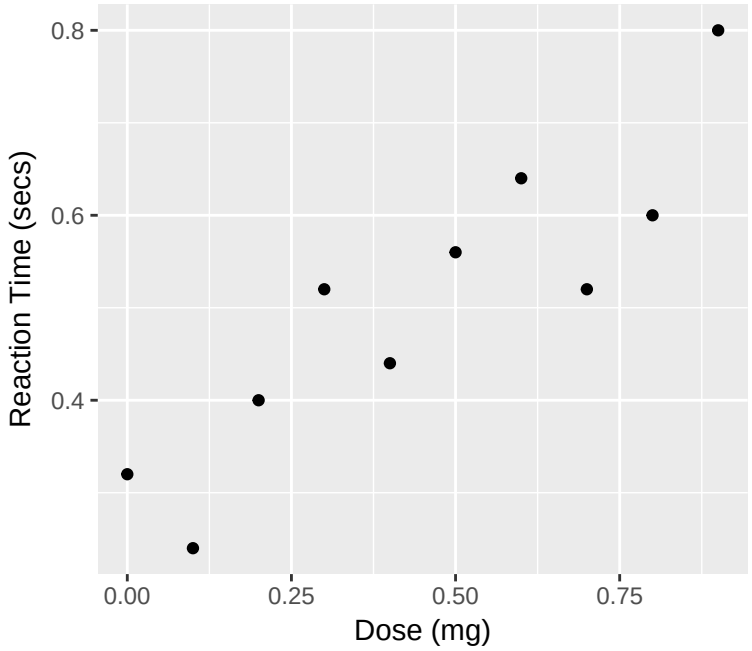
```
require(qdata)
data(drug)
str(drug)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  3 variables:
##  $ rat : int  1 2 3 4 5 6 7 8 9 10
##  $ dose: num  0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
##  $ time: num  0.32 0.24 0.4 0.52 0.44 0.56 0.64 0.52 0.6 0.8
```

Basic graphical data exploration may be achieved with:

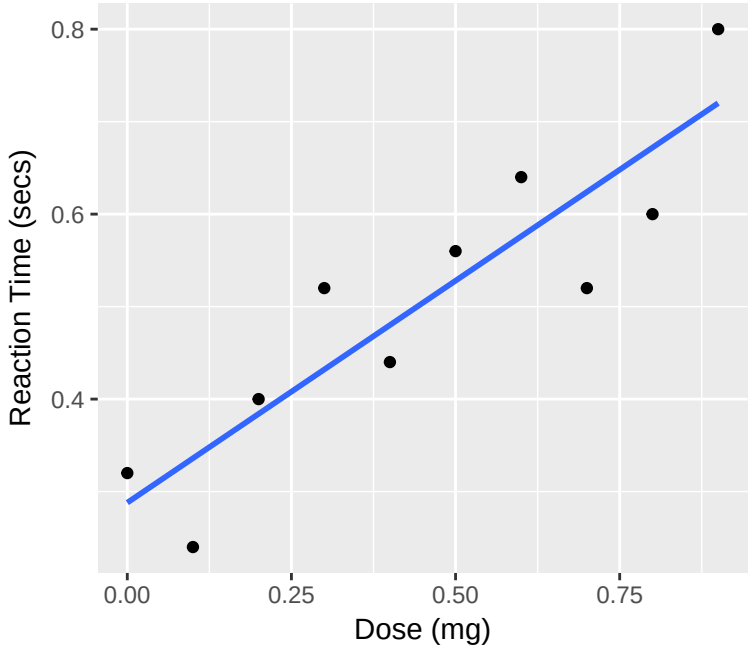
```
require(ggplot2)
pl <- ggplot(data = drug, mapping = aes(x = dose, y=time)) +
  geom_point() +
  xlab(label = "Dose (mg)") +
  ylab(label = "Reaction Time (secs)")

pl
```



A simple model for these data might be a straight line, which can be easy superposed to the data scatterplot by:

```
pl + geom_smooth(method="lm", se=FALSE)
```



The R command to fit a simple linear model is:

```
fm <- lm(formula = time ~ dose, data = drug)
```

As we said, `fm` object is a list of elements containing information about regression results.

Regression results can be investigated by generic function `summary()`, which includes the most important elements for model interpretation.

```
summary(fm)
```

```
##
## Call:
## lm(formula = time ~ dose, data = drug)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -0.104 -0.064  0.024  0.056  0.088
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.28800    0.04522   6.368 0.000216 ***
## dose         0.48000    0.08471   5.666 0.000472 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.07694 on 8 degrees of freedom
## Multiple R-squared:  0.8005, Adjusted R-squared:  0.7756
## F-statistic: 32.11 on 1 and 8 DF,  p-value: 0.0004724
```

Small p-values for t-test on coefficients lead to consider both coefficients as significant. Adjust R-squared equal to 0.78 shows an acceptable portion of total variation explained by the regression model.

Small p-value for F-statistic confirm fitted model significance when compared with null model (model only with intercept).

You can also visualize single elements of `fm` list.
For example:

```
# regression coefficients
coef(fm)
```

```
## (Intercept)        dose
##       0.288       0.480
```

```
# residuals
resid(fm)
```

```
##      1      2      3      4      5      6      7      8      9     10
##  0.032 -0.096  0.016  0.088 -0.040  0.032  0.064 -0.104 -0.072  0.080
```

```
# fitted values
fitted(fm)
```

```
##     1     2     3     4     5     6     7     8     9    10
## 0.288 0.336 0.384 0.432 0.480 0.528 0.576 0.624 0.672 0.720
```

Prediction for response variable at specific values of the explanatories variable can be gained using `predict()` function with the following sintax:

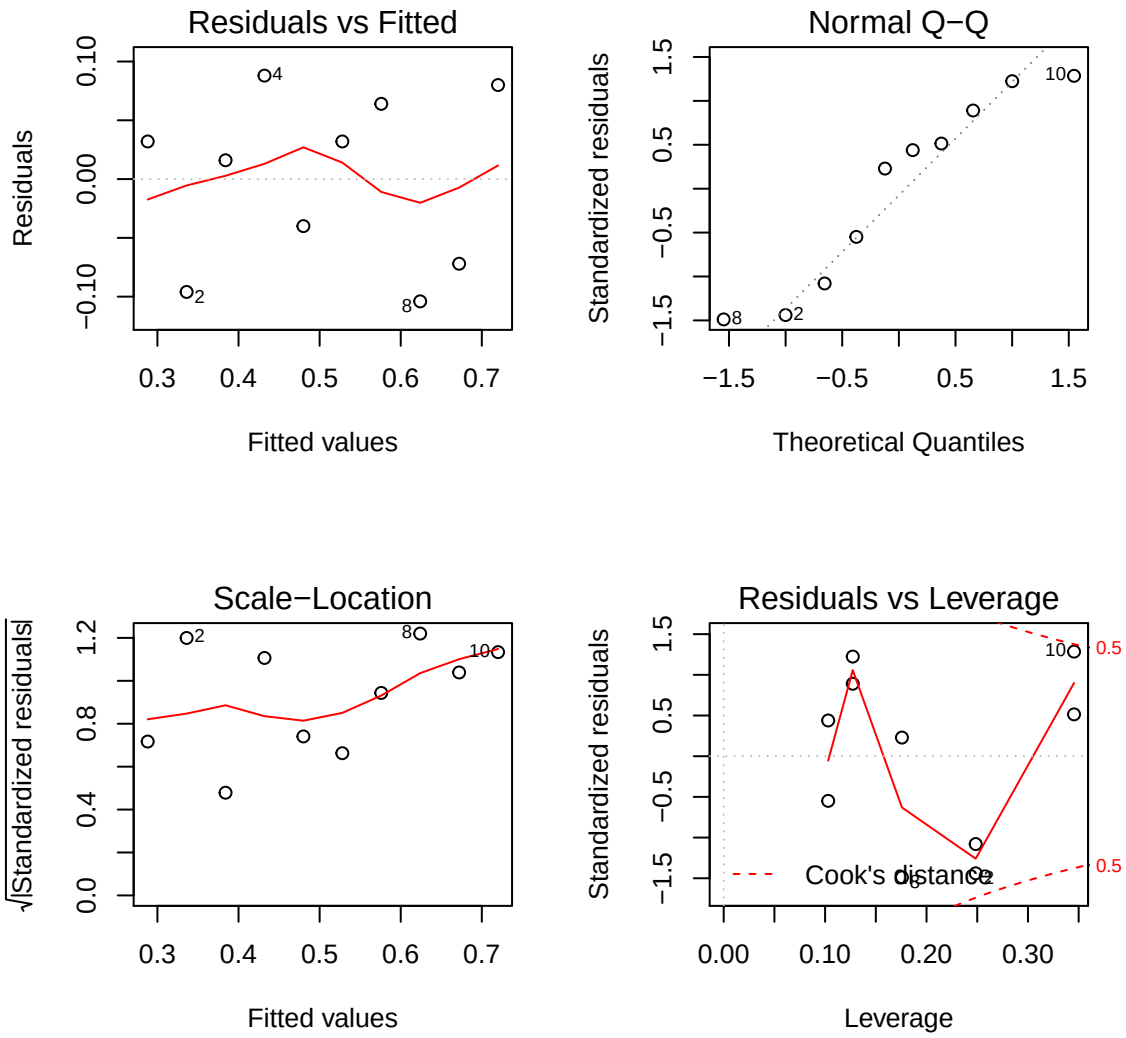
```
newdata <- data.frame(dose = c(0.2, 0.4, 0.6 ))
predict (fm, newdata = newdata)
```

```
##     1     2     3
## 0.384 0.480 0.576
```

Note how the `newdata` argument within `predict()` requires an object of class `data.frame`, whose column names have to be equal to the explicative variables names of the model.

For a visual inspection of the suite of residuals plots of the model, the sintax is:

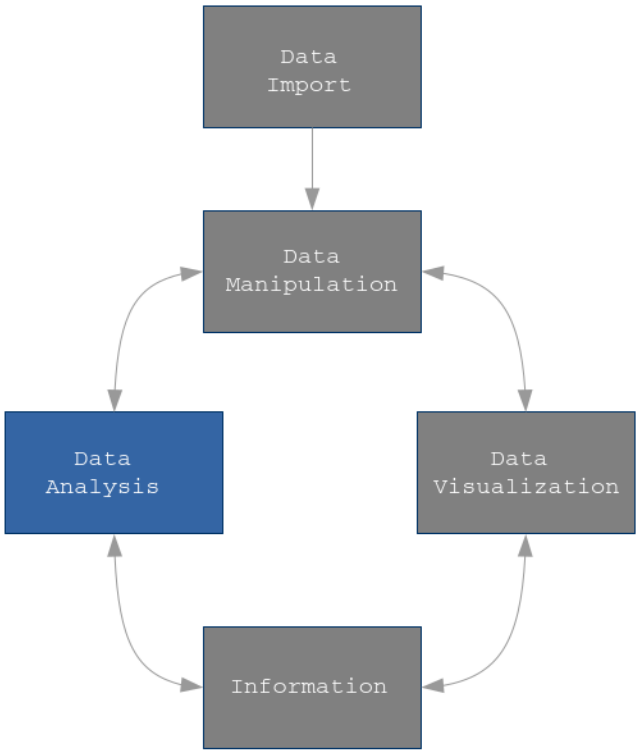
```
par(mfrow = c(2,2))
plot(fm)
```



The "Residuals vs Fitted" plot does not show, in this example, any particular pattern. The presence of patterns may suggest that the model is inadequate. The "Normal Q-Q" plot shows points close to the straight line. If the normal assumption of residuals is not satisfied, points are far from the straight line. The "Normal Q-Q plot" is less reliable on the distribution tails, i.e. points ought be very far from the straight line to suggest that residuals follow a non-normal distribution. The "Scale location" is similar to the residuals versus fitted values plot, but it uses the square root of the standardized residuals in order to diminish skewness. Like the first plot, there should be no discernable pattern to the plot. The "Residuals vs Leverage" shows leverage points. Leverage points are those observations, if any, made at extreme or outlying values of the independent variables such that the lack of neighboring observations means that the fitted regression model will pass close to that particular observation. Leverage points fall out the dotted lines.

Chapter 12

Data Mining with R



Data Mining is the process to discover interesting, previously unknown and potentially useful information from large amounts of data. It is an interdisciplinary field with contributions from many areas, such as statistics, machine learning, information retrieval, pattern recognition and bioinformatics. Data mining is widely used in many domains, such as retail, finance, telecommunication and social media.

R well supports data mining research and projects, providing lots of packages for data minig techniques.

In this chapter we will analize Neural Networks, which is an advanced data mining method.

12.1 Neural Networks

Neural networks tries to artificially simulate the organization and functioning of the human brain structures. A neural network can be seen as a system capable of giving an answer to a question or provide an output in response to an input. In particular, it takes a set of inputs (explanatory variables), transforms and weights these within a set of hidden units and hidden layers to produce a set of outputs or predictions (that are also transformed).

Usually, neural networks are made up by three layers:

- input layer, that capture of the network input signals
- hidden layer, that transform the inputs into something that the output layer can use
- output layer, that produces output signals

Next figure is an example of a feed forward neural network consisting of four inputs, a hidden layer that contains three units and an output layer that contains two outputs. The outputs of nodes in one layer are inputs to the next layer. The inputs to each node are combined using a weighted linear combination. The result is then usually modified by a nonlinear function before being output.

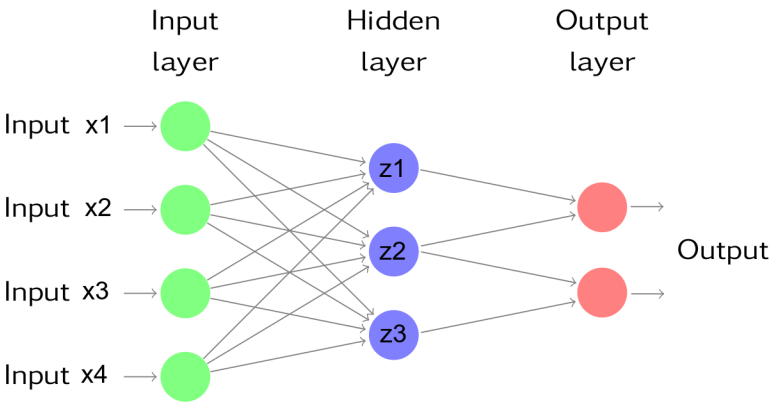


Figure 12.1: Neural Network scheme

The R `nnet` package provides `nnet()` function, which fits a single-hidden-layer neural network to data.

```
require(nnet)
```

`nnet()` function syntax is very similar to the one used for linear models:

```
nn_mod <- nnet(formula, data, size)
```

`nnet()` function requires `formula` argument, which is a symbolic description of the model to be fitted, which has the form: `response variable  predictor variables`. The variables involved in `formula` should be columns of a data frame specified in `data` argument. Unlike linear model definiton, you have to define `size` argument, which represents the number of units in the hidden layer.

The resulting object is of `nnet` or `nnet.formula` class and it can be investigated by most of the functions seen for linear model like `summary()`, `predict()`, `coef()` and `resid()`.

12.1.1 Titanic

Let us see how Neural Network works, considering `titanic` data, included in `qdata` package. `titanic` data contains information about surviving to Titanic wreck along with personal and travel information of the single passengers.

```
require(qdata)
data(titanic)
head(titanic)
```

```
## # A tibble: 6 × 4
##    Class Gender   Age   Status
##   <fctr> <fctr> <int>   <fctr>
## 1  Coach Female    20 Survived
## 2  Coach Female    21 Survived
## 3  Coach Female    26 Survived
## 4  Coach Female    26     Died
## 5  Coach Female    36 Survived
## 6  Coach Female    41 Survived
```

The aim of study is to find a prediction model to assess the probability of die for each passenger based on its `Age`, `Gender`, and `Class` of accomodation.

Some tables could help in describing the relations between die probability and explicative variables (`Age`, `Gender`, and `Class`):

```
require(dplyr)
# frequency table of the counts and percentage of Died and Survived people
titanic %>%
  group_by(Status) %>%
  summarise(n = n()) %>%
  mutate(freq = paste(round(n/sum(n)* 100, 2), "%"))
```

```
## # A tibble: 2 × 3
##     Status     n   freq
##     <fctr> <int>  <chr>
## 1     Died  1490 67.7 %
## 2 Survived   711 32.3 %
```

```
# frequency table of the counts and percentage of Died and Survived people by Class
titanic %>%
  group_by(Status, Class) %>%
  summarise(n = n()) %>%
  mutate(freq = paste(round(n/sum(n)* 100, 2), "%"))
```

```
## Source: local data frame [4 x 4]
## Groups: Status [2]
##
##     Status  Class     n    freq
##     <fctr> <fctr> <int>   <chr>
## 1     Died  Coach  1368 91.81 %
## 2     Died  First   122  8.19 %
## 3 Survived  Coach   508 71.45 %
## 4 Survived  First   203 28.55 %
```

```
# frequency table of the counts and percentage of Died and Survived people by Gender
titanic %>%
  group_by(Status, Gender) %>%
  summarise(n= n()) %>%
  mutate(freq = paste(round(n/sum(n)* 100, 2), "%"))
```

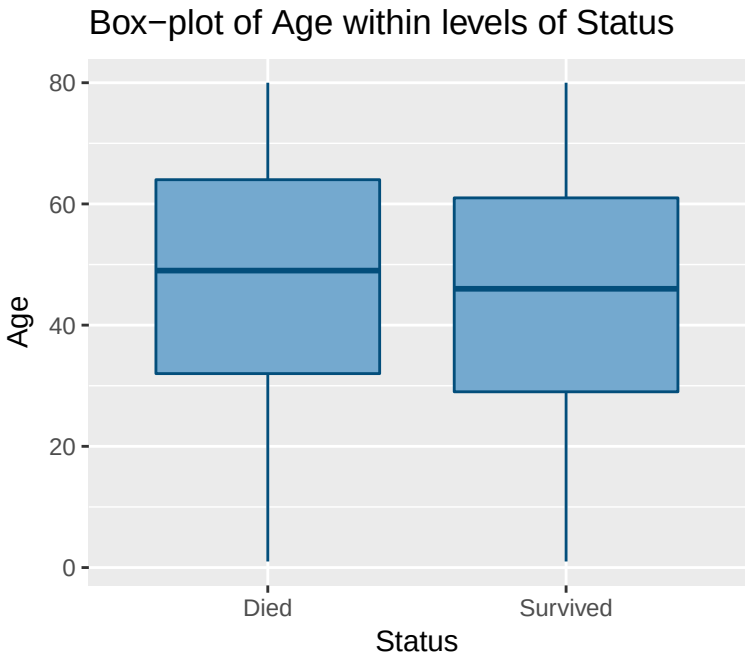
```
## Source: local data frame [4 x 4]
## Groups: Status [2]
```

```
##
##     Status Gender     n    freq
##     <fctr> <fctr> <int>   <chr>
## 1     Died Female   126  8.46 %
## 2     Died   Male  1364 91.54 %
## 3 Survived Female   344 48.38 %
## 4 Survived   Male   367 51.62 %
```

The above tables show counts and percentages of died and survived for each combination of Sex and Class factors. Some relations appear, but its interpretation is not really simple.

The following graph shows if some relations exist between Status and Age:

```
require(ggplot2)
ggplot(data=titanic, mapping=aes(x=Status, y=Age)) +
  geom_boxplot(fill="#74a9cf", colour="#034e7b")+
  ggtitle("Box-plot of Age within levels of Status")
```



Apparently, a slightly lower age is in survived passengers.

Neural Network method allows us to assess the probability of die for each passenger based on its `Age`, `Gender` and `Class` of accomodation.

Let us divide the data in train and test samples in order to estimate the model on train sample and to test the results on test sample.

```
# generate train and test sample
train <- titanic %>% sample_frac(0.7)
test <- titanic %>% slice(-as.numeric(rownames(train)))
```

Neural Network model estimate on train sample:

```
nn_mod <- nnet(Status ~ Class + Gender + Age, data = train, size = 3)
```

```
## # weights:  16
## initial  value 997.297049
## iter  10 value 924.823475
```

```
## iter  20 value 792.467557
## iter  30 value 787.742979
## iter  40 value 786.919231
## iter  50 value 785.340304
## iter  60 value 781.830746
## iter  70 value 779.632855
## iter  80 value 778.798058
## iter  90 value 778.208582
## iter 100 value 777.123373
## final  value 777.123373
## stopped after 100 iterations
```

The predictions on test sample can be gained using `predict()` function:

```
pr <- predict(object = nn_mod, newdata = test)
head(pr)
```

```
##        [,1]
## 1 0.6471217
## 2 0.6471217
## 3 0.6471217
## 4 0.6471217
## 5 0.6471217
## 6 0.6466071
```

The predictions included the probability of survive in the Titanic wreck according to `Gender`, `Age` and `Class`. We define that probabilities below 0.5 represent `Died` (`TRUE`) and probabilities over 0.5 represent `Survived` (`FALSE`).

```
test <- test %>% mutate(pr_mod = pr < .5)
```

Let us see how the performance of the fitted model on the test sample:

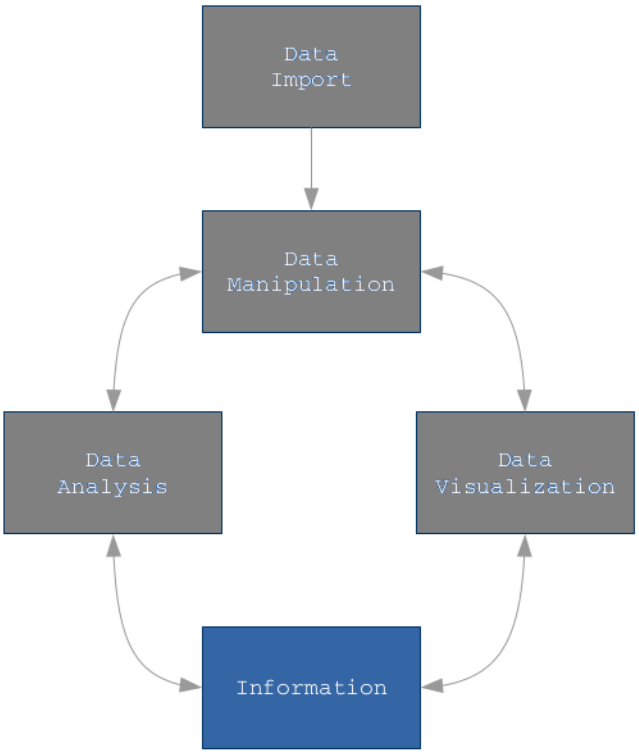
```
test %>%
  group_by(Status, pr_mod) %>%
  summarise(n = n()) %>%
  mutate(freq = paste(round(n/sum(n)* 100, 2), "%"))
```

```
## Source: local data frame [4 x 4]
## Groups: Status [2]
##
##     Status pr_mod     n    freq
##     <fctr>  <lgl> <int>   <chr>
## 1     Died  FALSE    81  16.1 %
## 2     Died   TRUE   422  83.9 %
## 3 Survived  FALSE    69 43.95 %
## 4 Survived   TRUE    88 56.05 %
```

We conclude that the model has quite good performance in the prediction of the probability of die in the Titanic wreck according to `Gender`, `Age` and `Class`.

Chapter 13

Reports with R Markdown



R Markdown provides an authoring framework for data science. R Markdown allows us to turn our analysis into high quality documents, reports, presentations and dashboards.

We can use a single R Markdown file to both:

- save and execute code
- generate high quality reports that can be shared with an audience

R Markdown documents are fully reproducible and support multiple programming languages like: R, Python, and SQL. It support also dozens of static and dynamic output formats, including HTML, PDF, MS Word, Beamer, HTML5 slides, Tufte-style handouts, books, dashboards, shiny applications, scientific articles, websites, and more.

13.1 Installation

You can install the R Markdown package from CRAN as follows:

```
install.packages("rmarkdown")
```

13.2 Markdown Basics

This is an R Markdown file, a plain text file that has the extension `.Rmd`:

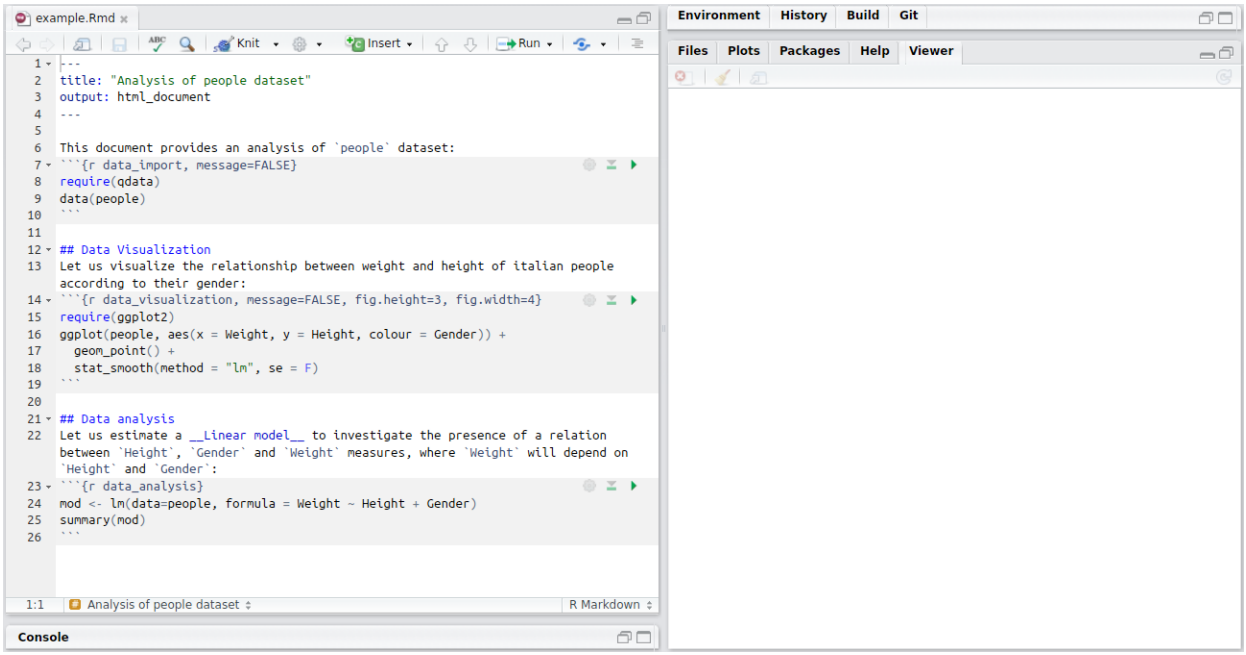


Figure 13.1: R Markdown File

Notice that the file contains three types of content:

- An (optional) YAML header surrounded by `---`
- R code chunks surrounded by ```
- text mixed with simple text formatting

Markdown is a simple formatting language designed to make authoring content easy for everyone. Rather than writing complex markup code (e.g. HTML or LaTeX), Markdown enables the use of a syntax much more like plain-text email.

13.2.1 Basic rules for text

This section provides quick references to the most commonly used R Markdown syntax.

Figure 13.2: Source: R Markdown Cheat Sheet

13.3 Rendering Output

To generate a report from the file, run the `render` command:

```
require(rmarkdown)
render("example.Rmd")
```

Otherwise, use the "Knit" button in the RStudio IDE to render the file and preview the output with a single click or the keyboard shortcut *Ctrl + Shift + K*.

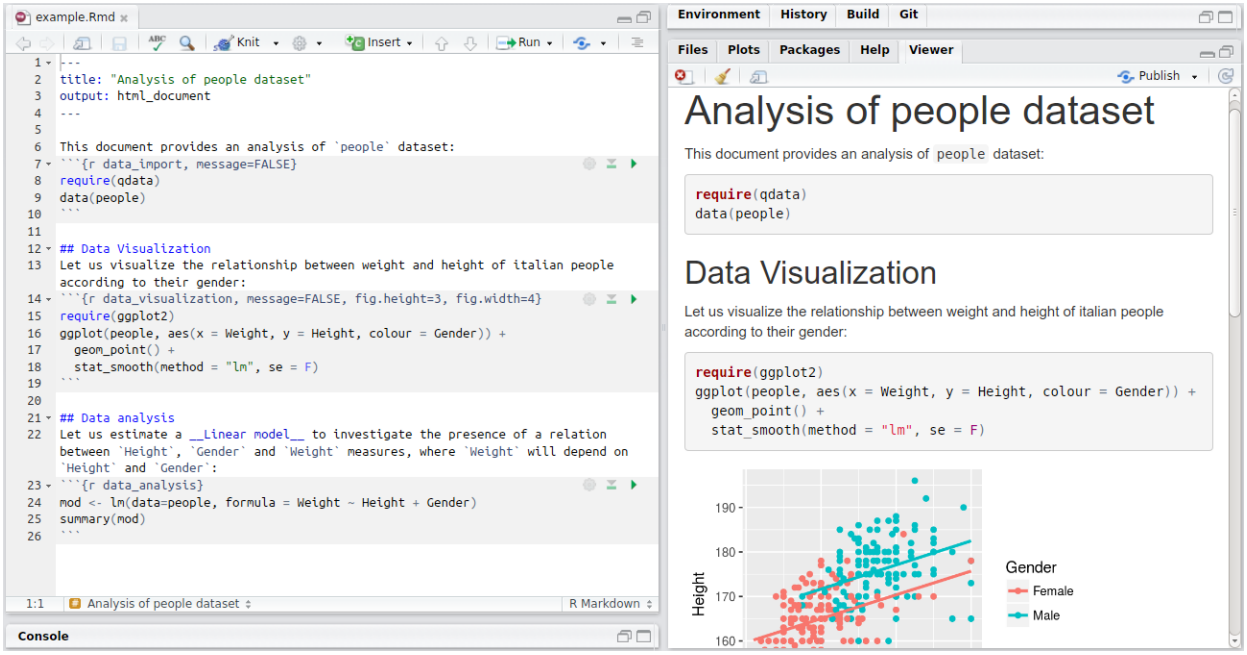


Figure 13.3: R Markdown Output

R Markdown generates a new file that contains selected text, code, and results from the .Rmd file. The new file can be a finished web page, PDF, MS Word document, slide show, notebook, handout, book, dashboard, package vignette or other format.