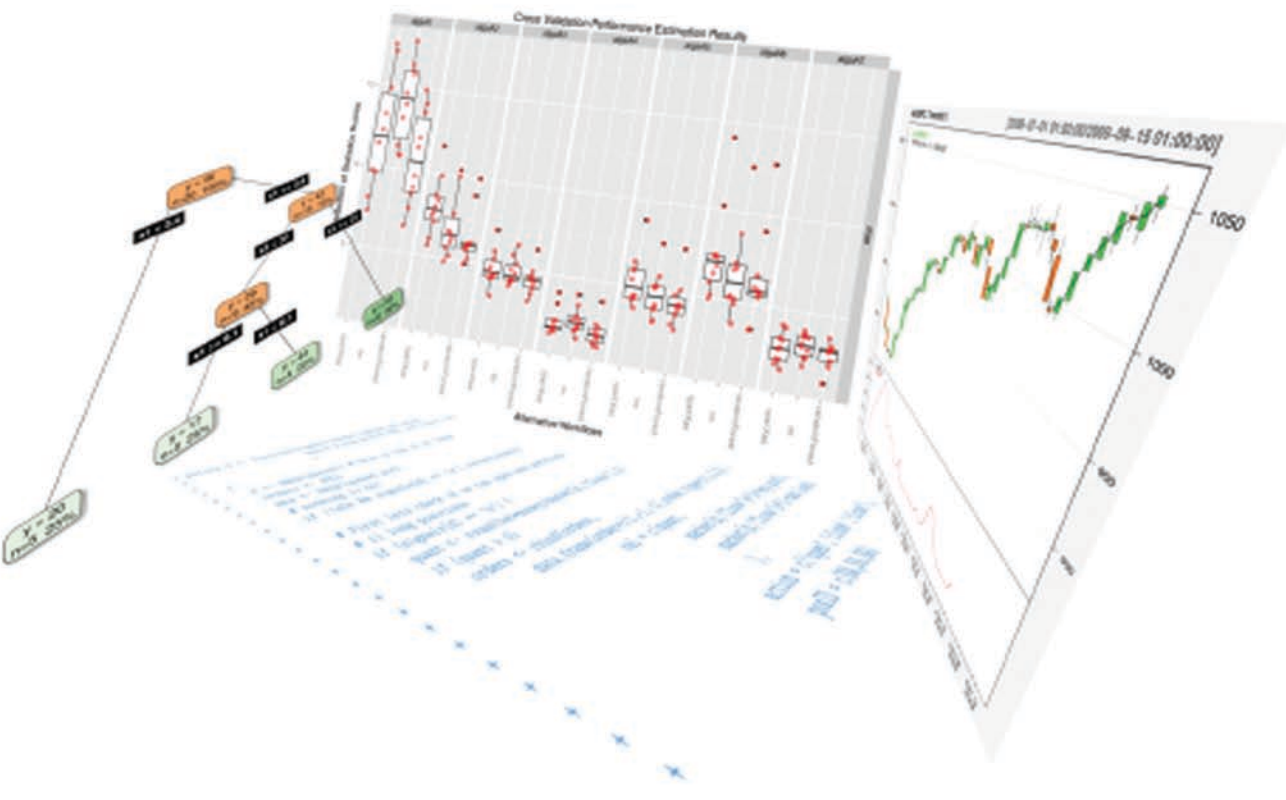
Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series

Data Mining with R

Learning with Case Studies

Second Edition



Luís Torgo

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

DATA MINING WITH R

LEARNING WITH CASE STUDIES

SECOND EDITION

Chapman & Hall/CRC
Data Mining and Knowledge Discovery Series

SERIES EDITOR

Vipin Kumar

University of Minnesota
Department of Computer Science and Engineering
Minneapolis, Minnesota, U.S.A.

AIMS AND SCOPE

This series aims to capture new developments and applications in data mining and knowledge discovery, while summarizing the computational tools and techniques useful in data analysis. This series encourages the integration of mathematical, statistical, and computational methods and techniques through the publication of a broad range of textbooks, reference works, and handbooks. The inclusion of concrete examples and applications is highly encouraged. The scope of the series includes, but is not limited to, titles in the areas of data mining and knowledge discovery methods and applications, modeling, algorithms, theory and foundations, data and knowledge visualization, data mining systems and tools, and privacy and security issues.

PUBLISHED TITLES

ACCELERATING DISCOVERY : MINING UNSTRUCTURED INFORMATION FOR HYPOTHESIS GENERATION
Scott Spangler

ADVANCES IN MACHINE LEARNING AND DATA MINING FOR ASTRONOMY
Michael J. Way, Jeffrey D. Scargle, Kamal M. Ali, and Ashok N. Srivastava

BIOLOGICAL DATA MINING
Jake Y. Chen and Stefano Lonardi

COMPUTATIONAL BUSINESS ANALYTICS
Subrata Das

COMPUTATIONAL INTELLIGENT DATA ANALYSIS FOR SUSTAINABLE DEVELOPMENT
Ting Yu, Nitesh V. Chawla, and Simeon Simoff

COMPUTATIONAL METHODS OF FEATURE SELECTION
Huan Liu and Hiroshi Motoda

CONSTRAINED CLUSTERING: ADVANCES IN ALGORITHMS, THEORY, AND APPLICATIONS
Sugato Basu, Ian Davidson, and Kiri L. Wagstaff

CONTRAST DATA MINING: CONCEPTS, ALGORITHMS, AND APPLICATIONS
Guozhu Dong and James Bailey

DATA CLASSIFICATION: ALGORITHMS AND APPLICATIONS
Charu C. Aggarawal

DATA CLUSTERING: ALGORITHMS AND APPLICATIONS
Charu C. Aggarawal and Chandan K. Reddy

DATA CLUSTERING IN C++: AN OBJECT-ORIENTED APPROACH
Guojun Gan

DATA MINING: A TUTORIAL-BASED PRIMER, SECOND EDITION
Richard J. Roiger

DATA MINING FOR DESIGN AND MARKETING
Yukio Ohsawa and Katsutoshi Yada

DATA MINING WITH R: LEARNING WITH CASE STUDIES, SECOND EDITION
Luís Torgo

EVENT MINING: ALGORITHMS AND APPLICATIONS
Tao Li

FOUNDATIONS OF PREDICTIVE ANALYTICS
James Wu and Stephen Coggeshall

GEOGRAPHIC DATA MINING AND KNOWLEDGE DISCOVERY, SECOND EDITION
Harvey J. Miller and Jiawei Han

GRAPH-BASED SOCIAL MEDIA ANALYSIS
Ioannis Pitas

HANDBOOK OF EDUCATIONAL DATA MINING
Cristóbal Romero, Sebastian Ventura, Mykola Pechenizkiy, and Ryan S.J.d. Baker

HEALTHCARE DATA ANALYTICS
Chandan K. Reddy and Charu C. Aggarwal

INFORMATION DISCOVERY ON ELECTRONIC HEALTH RECORDS
Vagelis Hristidis

INTELLIGENT TECHNOLOGIES FOR WEB APPLICATIONS
Priti Srinivas Sajja and Rajendra Akerkar

INTRODUCTION TO PRIVACY-PRESERVING DATA PUBLISHING: CONCEPTS AND TECHNIQUES
Benjamin C. M. Fung, Ke Wang, Ada Wai-Chee Fu, and Philip S. Yu

KNOWLEDGE DISCOVERY FOR COUNTERTERRORISM AND LAW ENFORCEMENT
David Skillicorn

KNOWLEDGE DISCOVERY FROM DATA STREAMS
João Gama

MACHINE LEARNING AND KNOWLEDGE DISCOVERY FOR ENGINEERING SYSTEMS HEALTH MANAGEMENT
Ashok N. Srivastava and Jiawei Han

MINING SOFTWARE SPECIFICATIONS: METHODOLOGIES AND APPLICATIONS
David Lo, Siau-Cheng Khoo, Jiawei Han, and Chao Liu

MULTIMEDIA DATA MINING: A SYSTEMATIC INTRODUCTION TO CONCEPTS AND THEORY
Zhongfei Zhang and Ruofei Zhang

MUSIC DATA MINING
Tao Li, Mitsunori Ogihara, and George Tzanetakis

NEXT GENERATION OF DATA MINING
Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar

RAPIDMINER: DATA MINING USE CASES AND BUSINESS ANALYTICS APPLICATIONS
Markus Hofmann and Ralf Klinkenberg

RELATIONAL DATA CLUSTERING: MODELS, ALGORITHMS, AND APPLICATIONS
Bo Long, Zhongfei Zhang, and Philip S. Yu

SERVICE-ORIENTED DISTRIBUTED KNOWLEDGE DISCOVERY
Domenico Talia and Paolo Trunfio

SPECTRAL FEATURE SELECTION FOR DATA MINING
Zheng Alan Zhao and Huan Liu

STATISTICAL DATA MINING USING SAS APPLICATIONS, SECOND EDITION
George Fernandez

SUPPORT VECTOR MACHINES: OPTIMIZATION BASED THEORY, ALGORITHMS, AND EXTENSIONS
Naiyang Deng, Yingjie Tian, and Chunhua Zhang

TEMPORAL DATA MINING
Theophano Mitsa

TEXT MINING: CLASSIFICATION, CLUSTERING, AND APPLICATIONS
Ashok N. Srivastava and Mehran Sahami

TEXT MINING AND VISUALIZATION: CASE STUDIES USING OPEN-SOURCE TOOLS
Markus Hofmann and Andrew Chisholm

THE TOP TEN ALGORITHMS IN DATA MINING
Xindong Wu and Vipin Kumar

UNDERSTANDING COMPLEX DATASETS: DATA MINING WITH MATRIX DECOMPOSITIONS
David Skillicorn

DATA MINING WITH R

LEARNING WITH CASE STUDIES

SECOND EDITION

Luís Torgo

University of Porto, Portugal

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20161025

International Standard Book Number-13: 978-1-4822-3489-3 (Hardback)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Preface

The main goal of this book is to introduce the reader to the use of R as a tool for data mining. R is a freely downloadable[1] language and environment for statistical computing and graphics. Its capabilities and the large set of available add-on packages make this tool an excellent alternative to many existing (and expensive!) data mining tools.

The main goal of this book is not to describe all facets of data mining processes. Many books exist that cover this scientific area. Instead we propose to introduce the reader to the power of R and data mining by means of several case studies. Obviously, these case studies do not represent all possible data mining problems that one can face in the real world. Moreover, the solutions we describe cannot be taken as complete solutions. Our goal is more to introduce the reader to the world of data mining using R through practical examples. As such, our analysis of the case studies has the goal of showing examples of knowledge extraction using R, instead of presenting complete reports of data mining case studies. They should be taken as examples of possible paths in any data mining project and can be used as the basis for developing solutions for the reader's own projects. Still, we have tried to cover a diverse set of problems posing different challenges in terms of size, type of data, goals of analysis, and the tools necessary to carry out this analysis. This hands-on approach has its costs, however. In effect, to allow for every reader to carry out our described steps on his/her computer as a form of learning with concrete case studies, we had to make some compromises. Namely, we cannot address extremely large problems as this would require computer resources that are not available to everybody. Still, we think we have covered problems that can be considered large and we have shown how to handle the problems posed by different types of data dimensionality.

This second edition strongly revises the R code of the case studies, making it more up-to-date with recent packages that have emerged in R. Moreover, we have decided to split the book into two parts: (i) a first part with introductory material, and (ii) the second part with the case studies. The first part includes a completely new chapter that provides an introduction to data mining, to complement the already existing introduction to R. The idea is to provide the reader with a kind of bird's eye view of the data mining field, describing more in depth the main topics of this research area. This information should complement the lighter descriptions that are given during the case studies analysis. Moreover, it should allow the reader to better contextualize the solutions of the case studies within the bigger picture of data mining tasks and methodologies. Finally, we hope this new chapter can serve as a kind of backup reference for the reader if more details on the methods used in the case studies are required.

We do not assume any prior knowledge about R. Readers who are new to R and data mining should be able to follow the case studies. We have tried to make the different case studies self-contained in such a way that the reader can start anywhere in the document. Still, some basic R functionalities are introduced in the first, simpler case studies, and are not repeated, which means that if you are new to R, then you should at least start with the first case studies to get acquainted with R. Moreover, as we have mentioned, the first part

[1] Download it from `http://www.R-project.org`.

of the book includes a chapter with a very short introduction to R, which should facilitate the understanding of the solutions in the following chapters. We also do not assume any familiarity with data mining or statistical techniques. Brief introductions to different data mining techniques are provided as necessary in the case studies. Still, the new chapter in the first part with the introduction to data mining includes further information on the methods we apply in the case studies as well as other methodologies commonly used in data mining. Moreover, at the end of some sections we provide "further readings" pointers that may help find more information if required. In summary, our target readers are more users of data analysis tools than researchers or developers. Still, we hope the latter also find reading this book useful as a form of entering the "world" of R and data mining.

The book is accompanied by a set of freely available R source files that can be obtained at the book's Web site.[2] These files include all the code used in the case studies. They facilitate the "do-it-yourself" approach followed in this book. We strongly recommend that readers install R and try the code as they read the book. All data used in the case studies is available at the book's Web site as well. Moreover, we have created an R package called `DMwR2` that contains several functions used in the book as well as the datasets already in R format. You should install and load this package to follow the code in the book (details on how to do this are given in the first chapter).

[2] `http://ltorgo.github.io/DMwR2`

Acknowledgments

I would like to thank my family for all the support they give me. Without them I would have found it difficult to embrace this project. Their presence, love, and caring provided the necessary comfort to overcome the ups and downs of writing a book. The same kind of comfort was given by my dear friends who were always ready for an extra beer when necessary. Thank you all, and now I hope I will have more time to share with you.

I am also grateful for all the support of my research colleagues and to LIAAD/INESC Tec LA as a whole. Thanks also to the University of Porto for supporting my research, and also to my colleagues at the Department of Computer Science of the Faculty of Sciences of the same University for providing such an enjoyable working environment. Part of the writing of this book was financially supported by a sabbatical grant (SFRH/BSAB/113896/2015) of FCT.

Finally, thanks to all students and colleagues who helped improving the first edition with their feedback, as well as in proofreading drafts of the current edition. In particular, I would like to thank to my students of Data Mining at the Masters on Computer Science of the Faculty of Sciences of the University of Porto, and also my students of the Data Mining with R subject at the Masters of Science on Business Analytics of Stern Business School of NYU — their involvement and feedback on my teaching material is strongly reflected on this new edition of the book.

Luis Torgo
Porto, Portugal

List of Figures

List of Tables

Chapter 1

Introduction

R[1] is a programming language and an environment for statistical computing (R Core Team, 2015b). It is similar to the S language developed at AT&T Bell Laboratories by Rick Becker, John Chambers and Allan Wilks. There are versions of R for the Unix, Windows and MacOS families of operating systems. Moreover, R runs on different computer architectures like Intel, PowerPC, Alpha systems and Sparc systems. R was initially developed by Ihaka and Gentleman (1996), both from the University of Auckland, New Zealand. The current development of R is carried out by a core team of a dozen people from different institutions around the world and it is supported by the R Foundation. R development takes advantage of a growing community that cooperates in its development due to its open source philosophy. In effect, the source code of every R component is freely available for inspection and/or adaptation. This fact allows you to check and test the reliability of anything you use in R and this ability may be crucial in many critical application domains. There are many critics of the open source model. Most of them mention the lack of support as one of the main drawbacks of open source software. It is certainly not the case with R! There are many excellent documents, books and sites that provide free information on R. Moreover, the excellent R-help mailing list is a source of invaluable advice and information. There are also searchable mailing list archives that you can (and should!) use before posting a question. More information on these mailing lists can be obtained at the R Web site in the section "Mailing Lists".

Data mining has to do with the discovery of useful, valid, unexpected, and understandable knowledge from data. These general objectives are obviously shared by other disciplines like statistics, machine learning, or pattern recognition. One of the most important distinguishing issues in data mining is size. With the widespread use of computer technology and information systems, the amount of data available for exploration has increased exponentially. This poses difficult challenges for the standard data analysis disciplines: One has to consider issues like computational efficiency, limited memory resources, interfaces to databases, etc. Other key distinguishing features are the diversity of data sources that one frequently encounters in data mining projects, as well as the diversity of data types (text, sound, video, etc.). All these issues turn data mining into a highly interdisciplinary subject involving not only typical data analysts but also people working with databases, data visualization on high dimensions, etc.

R has limitations with handling enormous datasets because all computation is carried out in the main memory of the computer. This does not mean that you will not be able to handle these problems. Taking advantage of the highly flexible database interfaces available in R, you will be able to perform data mining on large problems. Moreover, the awareness of the R community of this constant increase in dataset sizes has lead to the development of many new R packages designed to work with large data or to provide interfaces to other infrastructures better suited to heavy computation tasks. More information on this relevant work can be found on the High-Performance and Parallel Computing in R task view[2].

[1] `http://www.r-project.org`
[2] `http://cran.at.r-project.org/web/views/HighPerformanceComputing.html`

In summary, we hope that at the end of reading this book you are convinced that you can do data mining on large problems without having to spend any money at all! That is only possible due to the generous and invaluable contribution of lots of people who build such wonderful tools as R.

1.1 How to Read This Book

The main spirit behind the book is

Learn by doing it!

The first part of the book provides you with some basic information on both R and Data Mining. The second part of the book is organized as a set of case studies. The "solutions" to these case studies are obtained using R. All the necessary steps to reach the solutions are described. Using the book Web site[3] and the book-associated R package (DMwR2), you can get all of the code included in the document, as well as all data of the case studies. This should facilitate trying them out by yourself. Ideally, you should read this document beside your computer and try every step as it is presented to you in the book. R code and its respective output is shown in the book using the following font:

```
> citation()


To cite R in publications use:

  R Core Team (2016). R: A language and environment for
  statistical computing. R Foundation for Statistical Computing,
  Vienna, Austria. URL https://www.R-project.org/.

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {R: A Language and Environment for Statistical Computing},
    author = {{R Core Team}},
    organization = {R Foundation for Statistical Computing},
    address = {Vienna, Austria},
    year = {2016},
    url = {https://www.R-project.org/},
  }

We have invested a lot of time and effort in creating R, please
cite it when using it for data analysis. See also
'citation("pkgname")' for citing R packages.
```

R commands are entered at R command prompt, ">" in an interactive fashion. Whenever you see this prompt you can interpret it as R waiting for you to enter a command. You type in the commands at the prompt and then press the ENTER key to ask R to execute them. This may or may not produce some form of output (the result of the command) and then a new prompt appears. At the prompt you may use the arrow keys to browse and edit

---

[3] http://ltorgo.github.io/DMwR2

previously entered commands. This is handy when you want to type commands similar to what you have done before as you avoid typing them again.

Still, you can take advantage of the code provided at the book Web site to copy and paste between your browser or editor and the R console, thus avoiding having to type all the commands described in the book. This will surely facilitate your learning experience and improve your understanding of its potential.

1.2 Reproducibility

One of the main goals of this book is to provide you with illustrative examples of how to address several data mining tasks using the tools made available by R. For this to be possible we have worked hard to make sure all cases we describe are reproducible by our readers on their own computers. This means that if you follow all steps we describe in the book you should get the same results we describe.

There are two essencial components of this reproducibility goal: (i) the used R code; and (ii) the data of the case studies. Accompanying this book we provide two other means of facilitating your access to the code and data: (i) the book Web page; and (ii) the book R package. Together with the descriptions included in this book, the Web page and the package should allow you to easily replicate what we describe and also re-use and/or adapt it to your own application domains.

The book Web page[4] provides access to all code used in the book in a copy/paste-friendly manner, so that you can easily copy it from your browser into your R session. The code is organized by chapters and sections to facilitate the task of finding it.

The Web page also contains other useful information like the list of packages we use, or the data sets, as well as other files containing some of the objects created in the book, particularly when these can take considerable time to compute on more average desktop computers.

R is a very dynamic "ecosystem". This means that when you read this book most probably some of the packages we use (or even R itself) already have new versions out. Although this will most probably not create any problem, in the sense that the code we show will still work with these new versions, we can not be sure of this. If something stops working due to these new versions we will try to quickly post solutions in the "Errata" section of the book Web page. The book and the R code in it was created and tested in the following R version:

```
> R.version


               _
platform       x86_64-apple-darwin13.4.0
arch           x86_64
os             darwin13.4.0
system         x86_64, darwin13.4.0
status
major          3
minor          3.1
year           2016
month          06
day            21
```

[4]http://ltorgo.github.io/DMwR2

```
svn rev        70800
language       R
version.string R version 3.3.1 (2016-06-21)
nickname       Bug in Your Hair
```

At the book Web page you will also find the information on the versions of all used packages in our R system when the code was executed.

The book R package is another key element for allowing reproducibility. This package contains several of the functions we describe and/or use in the book, as well as the datasets of the case studies (which as we have mentioned above are also available in the book Web page). This package is available and installable from the usual sources, i.e. the R central repository (CRAN). It is possible that the package evolves to new versions if any bug is found in the code we provide. These corrections will tend to follow a slow pace as recommended by CRAN policies. In this context, for more up-to-date versions of the package, which may include not yet so well tested solutions (so use it at your own risk), you may wish to download and install the development version of the package from its Web page:
`https://github.com/ltorgo/DMwR2`

Part I

A Short Introduction to R and Data Mining

Chapter 2

Introduction to R

This chapter provides a very short introduction to the main features of the R language. We do not assume any familiarity with computer programming. Readers should be able to easily follow the examples presented in this chapter. Still, if you feel some lack of motivation to continue reading this introductory material, do not worry. You may proceed to the case studies and then return to this introduction as you get more motivated by the concrete applications.

The material in this chapter should serve as a quick tutorial for those that are not familiar with the basics of the R language. Some other more specific aspects of R will also appear in the next chapter when we introduce the reader to some concepts of Data Mining. Finally, further learning will also take place when presenting the case studies in the second part of the book. Still, some basic knowledge of R is necessary to start addressing these case studies and this chapter should provide that in case you do not have it.

2.1 Starting with R

R is a functional language for statistical computation and graphics. It can be seen as a dialect of the S language (developed at AT&T) for which John Chambers was awarded the 1998 Association for Computing Machinery (ACM) Software award that mentioned that this language "forever altered how people analyze, visualize and manipulate data".

R can be quite useful just by using it in an interactive fashion at its command line. Still, more advanced uses of the system will lead the user to develop his own functions to systematize repetitive tasks, or even to add or change some functionalities of the existing add-on packages, taking advantage of being open source.

The easiest way to install R in your system is to obtain a binary distribution from the R Web site[1] where you can follow the link that takes you to the CRAN (Comprehensive R Archive Network) site to obtain, among other things, the binary distribution for your particular operating system/architecture. If you prefer to build R directly from the sources, you can get instructions on how to do it from the CRAN but most of the times that is not necessary at all.

After downloading the binary distribution for your operating system you just need to follow the instructions that come with it. In the case of the Windows version, you simply execute the downloaded file (`R-3.3.1-win.exe`)[2] and select the options you want in the following menus. In some operating systems you may need to contact your system administrator to fulfill the installation task due to lack of permissions to install software.

To run R in Windows you simply double-click the appropriate icon on your desktop,

[1] `http://www.R-project.org`.
[2] The actual name of the file changes with newer versions. This is the name for version 3.3.1

while in Unix versions you should type R at the operating system prompt. Both will bring up the R console with its prompt ">".

If you want to quit R you can issue the command `q()` at the prompt. You will be asked if you want to save the current workspace. You should answer yes only if you want to resume your current analysis at the point you are leaving it, later on.

A frequently used alternative way to interact with R is through RStudio[3]. This free software can be downloaded and installed for the most common setups (e.g. Linux, Windows or Mac OS X). It is an integrated development environment that includes on the same graphical user interface several important elements of R, like its console where you can interact with R, a script editor where you can write more complex programs/solutions to your problems, an interface to browse the help pages of R, and many other useful facilities. I strongly recommend its usage, particularly if you are starting with R.[4]

Although the set of tools that comes with R is by itself quite powerful, it is natural that you will end up wanting to install some of the large (and growing) set of add-on packages available for R at CRAN. In the Windows version this is easily done through the "`Packages`" menu. After connecting your computer to the Internet you should select the "`Install package from CRAN...`" option from this menu. This option will present a list of the packages available at CRAN. You select the one(s) you want, and R will download the package(s) and self-install it(them) on your system. In Unix versions, things may be slightly different depending on the graphical capabilities of your R installation. Still, even without selection from menus, the operation is simple.[5] Suppose you want to download the package that provides functions to connect to MySQL databases. This package name is **RMySQL**.[6] You just need to type the following command at R prompt:

```
> install.packages("RMySQL")
```

The `install.packages()` function has many parameters, among which there is the `repos` argument that allows you to indicate the nearest CRAN mirror.[7] Still, the first time you run the function in an R session, it will prompt you for the repository you wish to use.

One thing that you surely should do is to install the package associated with this book, named **DMwR2**. This package will give you access to several functions used throughout the book as well as the datasets. You install the package as any other package available on CRAN, i.e by issuing the following command at your R prompt (or using the respective menu if using RStudio),

```
> install.packages("DMwR2")
```

Once this procedure is finished you may use the book package when necessary by loading it as any other package,

```
> library(DMwR2)
```

The function `installed.packages()` allows you to know the packages currently installed in your computer,

---

[3] `https://www.rstudio.com/`

[4] Other alternatives include for instance the excellent Emacs package called ESS (`http://ess.r-project.org/`), in case you prefer Emacs as your editor.

[5] Please note that the following code also works in other versions, although you may find the use of the menus more practical.

[6] You can get an idea of the functionalities of each of the R packages in the R FAQ (frequently asked questions) at CRAN.

[7] The list of available mirrors can be found at `http://cran.r-project.org/mirrors.html`.

```
> installed.packages()
```

This produces a long output with each line containing a package, its version information, the packages it depends on, and so on. A more user-friendly, although less complete, list of the installed packages can be obtained by issuing

```
> library()
```

The following command can be very useful as it allows you to check whether there are newer versions of your installed packages at CRAN:

```
> old.packages()
```

Moreover, you can use the following command to update all your installed packages:

```
> update.packages()
```

R has an integrated help system that you can use to know more about the system and its functionalities. Moreover, you can find extra documentation at the R site. R comes with a set of HTML files that can be read using a Web browser[8]. On Windows and Mac OS X versions of R, these pages are accessible through the HELP menu. Alternatively, you can issue `help.start()` at the prompt to launch a browser showing the HTML help pages. Another form of getting help is to use the `help()` function. For instance, if you want some help on the `plot()` function, you can enter the command "`help(plot)`" (or alternatively, `?plot`). A quite powerful alternative, provided you are connected to the Internet, is to use the `RSiteSearch()` function that searches for key words or phrases in the mailing list archives, R manuals, and help pages; for example,

```
> RSiteSearch('neural networks')
```

Finally, there are several places on the Web that provide help on several facets of R, such as the sites `http://www.rseek.org/` or `http://www.rdocumentation.org/`. For more direct questions related to R, stack overflow is a "must"[9].

2.2 Basic Interaction with the R Console

The R console is the place where you carry out most of the interaction with R. This allows for easy interactive exploration of ideas that may solve your data analysis problems. Frequently, after this exploration phase one tends to dump the sequence of R commands that lead to the solution we have found into an R script file. These script files can then be reused, for instance by asking R to execute all commands contained in the script file in sequence.

The interaction with the R console consists of typing some instruction followed by the ENTER key, and receiving back the result of this command. The simplest example of this usage would be to ask R to carry out some calculation:

[8]Obviously if you are using RStudio it is even easier to browse the help pages.

[9]`http://stackoverflow.com/questions/tagged/r`

```
> 4 + 3 / 5^2

[1] 4.12
```

The rather cryptic "[1]" in front of the output can be read as "this output line is showing values starting from the first element of the object". This is particularly useful for results containing many values, as these may be spread over several lines of output. For now we can simply ignore the "[1]" as we will return to this issue later.

More interesting usages of R typically involve some of its many functions, as shown in the following simple examples:

```
> rnorm(4, mean = 10, sd = 2)

[1] 10.257398 10.552028  9.677471  4.615118

> mean(sample(1:10, 5))

[1] 6
```

The first of these instructions randomly generates 4 numbers from a normal distribution with mean 10 and standard deviation 2, while the second calculates the mean of 5 random numbers generated from the interval of integers from 1 to 10. This last instruction is also an example of something we see frequently in R- function composition. This mathematical concept involves applying a function to the result of another function, in this case calculating the mean of the result of the call to the function `sample()`.

Another frequent task we will carry out at the R prompt is to generate some statistical graph of a dataset. For instance, in Figure 2.1 we see a scatter plot containing 5 points whose coordinates were randomly generated in the interval 1 to 10. The code to obtain such a graph is the following:[10]

```
> plot(x=sample(1:10,5),y=sample(1:10,5),
+      main="Five random points",xlab="X values",ylab="Y values")
```

These are just a few short examples of the typical interaction with R. In the next sections we will learn about the main concepts behind the R language that will allow us to carry out useful data analysis tasks with this tool.

2.3 R Objects and Variables

Everything in R is stored as an object. An object is most of the time associated with a variable name that allows us to refer to its content. We can think of a variable as referring to some storage location in the computer memory that holds some content (an object) that can range from a simple number to a complex model.

R objects may store diverse types of information. The simplest content is some value of

[10] The "+" sign you see is the continuation prompt. It appears any time you type ENTER before you finish some statement as a way of R reminding you that there is something missing till it can execute your order. You should remember that these prompt characters are not to be entered by you! They are automatically printed by R (as with the normal prompt ">").

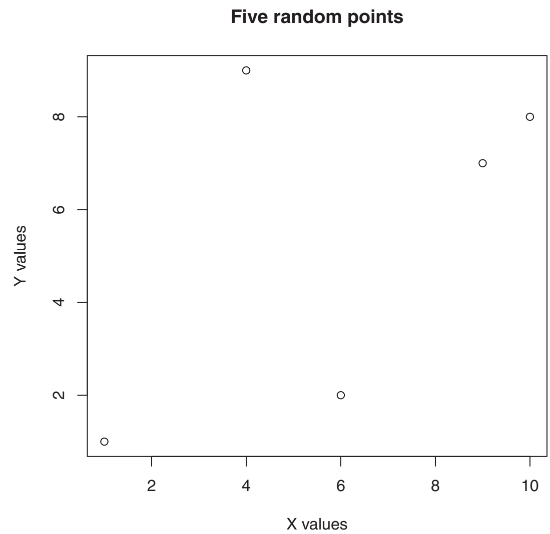


FIGURE 2.1: A simple scatter plot.

one of R basic data types : *numeric*, *character*, or *logical* values[11]. Character values in R are strings of characters[12] enclosed by either single or double quotes (e.g. `"hello"` or `'today'`), while the logical values are either TRUE or FALSE.[13] Please be aware that R is case-sensitive so true and false must be in capital letters!

Other more complex data types may also be stored in objects. We will see examples of this in the following sections.

Content (i.e. objects) may be stored in a variable using the assignment operator. This operator is denoted by an angle bracket followed by a minus sign (`<-`):[14]

```
> vat <- 0.2
```

The effect of the previous instruction is thus to store the number 0.2 on a variable named **vat**. By simply entering the name of a variable at the R prompt one can see its contents:[15]

```
> vat

[1] 0.2
```

Below you will find other examples of assignment statements. These examples should make it clear that this is a destructive operation, as any variable can only have a single content at any time t. This means that by assigning some new content to an existing variable, you in effect lose its previous content:

[11] Things are in effect slightly more complex, as R is also able to distinguish between floating point and integer numbers. Still, this is seldom required, unless you are heavily concerned with memory usage and CPU speed. Moreover, R also has complex numbers as another base data type but again this is not frequently used.

[12] This means the character type is in effect a set of characters, which are usually known as strings in some programming languages, and not a single character as you might expect.

[13] You may actually also use T or F.

[14] You may also use the `=` sign but I would not recommend it as it may be confused with testing for equality.

[15] Or an error message if we type the name incorrectly, a rather frequent error!

```
> y <- 39
> y

[1] 39

> y <- 43
> y

[1] 43
```

You can also assign numerical expressions to a variable. In this case the variable will store the result of the evaluation of the expression, not the expression:

```
> z <- 5
> w <- z^2
> w

[1] 25

> i <- (z * 2 + 45)/2
> i

[1] 27.5
```

This means that we can think of the assignment operation as "evaluate whatever is given on the right side of the operator, and assign (store) the result (an object of some type) of this evaluation in the variable whose name is given on the left side".

Every object you create will stay in the computer memory until you delete it (or you exit R). You may list the objects currently in the memory by issuing the `ls()` or `objects()` command at the prompt. If you do not need an object, you may free some memory space by removing it using the function `rm()`:

```
> ls()

[1] "i"   "vat" "w"   "y"   "z"

> rm(vat,y,z,w,i)
```

Variable names may consist of any upper- and lower-case letters, the digits 0 to 9 (except in the beginning of the name), and also the period, ".", which behaves like a letter. Once again we remind that names in R are *case sensitive*, meaning that `Color` and `color` are two distinct variables with potentially very different content. This is in effect a frequent cause of frustration for beginners who keep getting "object not found" errors. If you face this type of error, start by checking the correctness of the name of the object causing the error.

2.4 R Functions

R functions are a special type of R object designed to carry out some operation. R functions, like mathematical functions, are applied to some set of arguments and produce a result. In R, both the arguments that we provide when we call the function and the result of the function execution are R objects whose type will depend on the function. R functions range from simple objects implementing some standard calculation, e.g. calculating the square root of a number, to more complex functions that can obtain some model of a dataset, e.g. a neural network. R already comes with an overwhelming set of functions available for us to use, but as we will see, the user can also create new functions.

In terms of notation, a function has a name and can have zero or more parameters. When we call (execute) the function we use its name followed by the arguments between parentheses separated by commas[16],

```
> max(4, 5, 6, 12, -4)

[1] 12
```

In the above example we are calling a function named `max()` that as the name suggests returns the maximum value of the arguments supplied by the user when calling the function.

In R we frequently tend to use function composition that, as mentioned before, consists of applying functions to the result of other functions, as shown in this example where we obtain the maximum of a random sample of 30 integers in the interval 1 to 100[17]:

```
> max(sample(1:100, 30))

[1] 99
```

R allows the user to create new functions. This is a useful feature, particularly when you want to automate certain tasks that you have to repeat over and over. Instead of typing the instructions that perform this task every time you want to execute it, you encapsulate them in a new function and then simply use it whenever necessary.

R functions are objects that can be stored in a variable. The contents of these objects are the statements that, when executed, carry out the task for which the function was designed. These variables where we store the content of a function will act as the function name. Thus to create a new function we use the assignment operator to store the contents of the function in a variable (whose name will be the name of the function).

Let us start with a simple example. Suppose you often want to calculate the standard error of a mean associated with a set of values. By definition, the standard error of a sample mean is given by

$$\text{standard error} = \sqrt{\frac{s^2}{n}}$$

where s^2 is the sample variance and n the sample size.

Given a set of numbers, we want a function to calculate the respective standard error of the mean of these numbers. Let us decide to call this function `se`. Before proceeding to create the function we should check whether there is already a function with this name in R. If that is the case, then it would be better to use another name, not to "hide" the other R function from the user.[18] We can check the existence of some object with a certain name using the function `exists()`,

[16] Note that even if the function takes no arguments we need to call it with the parentheses, e.g. `f()`.

[17] Due to the random nature of the `sample()` function you may get a different maximum if you run this code.

[18] You do not have to worry about overriding the definition of the R function. It will continue to exist, although your new function with the same name will be on top of the search path of R, thus "hiding" the other standard function.

```
> exists("se")

[1] FALSE
```

The fact that R answered FALSE means that there is no object with the name `se` and thus we are safe to create a function with that name. The following is a possible way to create our function:

```
> se <- function(x) {
+     v <- var(x)
+     n <- length(x)
+     return(sqrt(v/n))
+ }
```

Thus, to create a function object, you assign to its name something with the general form

`function(<set of parameters>) { <set of R instructions> }`

A set of R instructions (a block) is delimited by curly braces and it is formed by each instruction on its own line. This means that in our example we have decided that to calculate the standard error of the sample mean of a set of numbers it would be sufficient to execute the above 3 statements. The first of these calls the function `var()` with the content of the variable `x`. This variable is a parameter of the function. Parameters are special variables that will hold the values supplied in the arguments of the function when the user calls it. This means that whenever some user calls our `se` function he will have to supply a set of values in the first (and only) argument of this function. These values will be assigned by R to the parameter (variable) `x`. The function `var()` is an R function that returns the variance of a set of values, that we decided to store in the variable `v`. The second statement uses function `length()` to obtain the number of values in `x`, that we store in another variable named `n`. Having these two quantities we are ready to calculate the standard error, by simply calculating the square root (function `sqrt()`) of the quotient of `v` by `n`. The result of this calculation is then returned back to the user by using the function `return()`.

After creating this function, we could use it as follows:

```
> mySample <- rnorm(100, mean=20, sd=4)
> se(mySample)

[1] 0.3550299
```

In the above code we have used the function `rnorm()` to obtain a random sample of 100 numbers from a normal distribution with mean 20 and standard deviation 4. We have then called our function with this set of numbers. Please note that due to the random nature of the function `rnorm()` you may get a different result.

Sometimes we want to create functions that may have some parameters that have default values. For instance, we could create a function to convert a value in meters to other units of length. This function could take as a first argument the value in meters and as a second argument the target unit. However, we could allow the user to omit this second argument by setting a default value when we create the function. The following is an illustration of this:

```
> convMeters <- function(val, to="inch") {
+     mult <- switch(to,inch=39.3701,foot=3.28084,yard=1.09361,mile=0.000621371,NA)
+     if (is.na(mult)) stop("Unknown target unit of length.")
+     else return(val*mult)
+ }
> convMeters(23,"foot")

[1] 75.45932

> convMeters(40,"inch")

[1] 1574.804

> convMeters(40)

[1] 1574.804

> convMeters(2.4,"km")

Error in convMeters(2.4, "km"): Unknown target unit of length.
```

The above function is able to convert meters to inches, feet, yards, and miles. As seen in the example calls, the user may omit the second argument as this has a default value ("inch"). This default value was established at the function creation by telling R not only the name of the parameter (`to`), but also a value that the parameter should take in case the user does not supply another value. Note that this value will always be overridden by any value the user supplies when calling the function.

The code of the function also illustrates a few other functions available in R. Function `switch()` for instance, allows us to compare the contents of a variable (`to` in the above code), against a set of options. For each option we can supply the value that will be the result of the function `switch()`. In the above example, if the variable `to` has the value "inch" the value assigned to the variable `mult` will be 39.3701. The function also allows to supply a return value in case the variable does not match any of the alternatives. In this case we are returning the special value NA. The goal here is to foresee situations where the user supplies a target unit that is unknown to this function. The following statement is another conditional statement. The `if` statement allows us to have conditional execution of other statements. In this case if the value assigned to variable `mult` was NA (which is checked by a call to the function `is.na()`), then we want to stop the execution of the function with some sort of error message (using function `stop()`) because the user has supplied an unknown target unit. Otherwise we simply carry out the conversion calculation and return it as the result of the function execution.

The way we call functions (either existing or the ones we create) can also have some variations, namely in terms of the way we supply the values for the parameters of the functions. The most frequent setup is when we simply supply a value for each parameter, e.g.:

```
> convMeters(56.2,"yard")

[1] 61.46088
```

Calling the function this way we are supplying the values for the parameters "by position", i.e. the value in the first argument (56.2) is assigned by R to the first parameter of the function (`val`), and the value in the second argument ("yard") is assigned to the second parameter (to). We may also supply the parameter values "by name". We could get the same exact result with the following call:

```
> convMeters(to="yard",val=56.2)

[1] 61.46088
```

In effect, we can even mix both forms of calling a function,

```
> convMeters(56.2,to="yard")

[1] 61.46088
```

Calling by name is particularly useful with functions with a lot of parameters, most of which with default values. Say we have a function named f with 20 parameters, all but the two first having default values. Suppose we want to call the function but we want to supply a value different from the default for the tenth parameter named tol. With the possibility of calling by name we could do something like:

```
> f(10,43.2,tol=0.25)
```

This avoids having to supply all the values till the tenth argument in order to be able to use a value different from the default for this parameter.

2.5 Vectors

The most basic data object in R is a vector. Even when you assign a single number to a variable (like in x <- 45.3), you are creating a vector containing a single element. A vector is an object that can store a set of values of the same base data type. Thus you may have for instance vectors of strings, logical values, or numbers. The length of a vector object is the number of elements in it, and can be obtained with the function length().

Most of the time you will be using vectors with length larger than 1. You can create a vector in R, using the c() function, which combines its arguments to form a vector:

```
> v <- c(4, 7, 23.5, 76.2, 80)
> v

[1]  4.0  7.0 23.5 76.2 80.0

> length(v)

[1] 5

> mode(v)

[1] "numeric"
```

The mode() function returns the base data type of the values stored in an object. All elements of a vector must belong to the same base data type. If that is not true, R will force it by type coercion. The following is an example of this:

```
> v <- c(4, 7, 23.5, 76.2, 80, "rrt")
> v

[1] "4"    "7"    "23.5" "76.2" "80"   "rrt"

> mode(v)

[1] "character"
```

All elements of the vector have been converted to the character type, i.e. strings.

All vectors may contain a special value called NA. This represents a missing value:

```
> u <- c(4, 6, NA, 2)
> u

[1]  4  6 NA  2

> k <- c(TRUE, FALSE, FALSE, NA, TRUE)
> k

[1]  TRUE FALSE FALSE    NA  TRUE
```

You can access a particular element of a vector through an index between square brackets:

```
> u[2]

[1] 6
```

The example above gives you the second element of the vector **u**. In Section 2.9 we will explore more powerful indexing schemes.

You can also change the value of one particular vector element by using the same indexing strategies:

```
> k[4] <- TRUE
> k

[1]  TRUE FALSE FALSE  TRUE  TRUE
```

R allows you to create empty vectors like this:

```
> x <- vector()
```

The length of a vector can be changed by simply adding more elements to it using a previously nonexistent index. For instance, after creating the empty vector `x`, you could type

```
> x[3] <- 45
> x

[1] NA NA 45
```

Notice how the first two elements have a missing value, NA. This sort of flexibility comes with a cost. Contrary to other programming languages, in R you will not get an error if you use a position of a vector that does not exist:

```
> length(x)

[1] 3

> x[10]

[1] NA

> x[5] <- 4
> x

[1] NA NA 45 NA  4
```

To shrink the size of a vector, you can take advantage of the fact that the assignment operation is destructive, as we have mentioned before. For instance,

```
> v <- c(45, 243, 78, 343, 445, 44, 56, 77)
> v

[1]  45 243  78 343 445  44  56  77

> v <- c(v[5], v[7])
> v

[1] 445  56
```

Through the use of more powerful indexing schemes to be explored in Section 2.9, you will be able delete particular elements of a vector in an easier way.

2.6 Vectorization

One of the most powerful aspects of the R language is the vectorization of several of its available functions. These functions can be applied directly to a vector of values producing an equal-sized vector of results. For instance,

```
> v <- c(4, 7, 23.5, 76.2, 80)
> sqrt(v)

[1] 2.000000 2.645751 4.847680 8.729261 8.944272
```

The function `sqrt()` calculates the square root of its argument. In this case we have used a vector of numbers as its argument. Vectorization makes the function produce a vector of the same length, with each element resulting from applying the function to the respective element of the original vector.

You can also use this feature of R to carry out vector arithmetic:

```
> v1 <- c(4, 6, 87)
> v2 <- c(34, 32.4, 12)
> v1 + v2

[1] 38.0 38.4 99.0
```

What if the vectors do not have the same length? R will use a *recycling rule* by repeating the shorter vector until it reaches the size of the larger vector. For example,

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2)
> v1 + v2

[1] 14  8 18 26
```

It is just as if the vector `c(10,2)` was `c(10,2,10,2)`. If the lengths are not multiples, then a warning is issued, but the recycling still takes place (it is a warning, not an error):

```
> v1 <- c(4, 6, 8, 24)
> v2 <- c(10, 2, 4)
> v1 + v2

Warning in v1 + v2: longer object length is not a multiple of shorter object length

[1] 14  8 12 34
```

As mentioned before, single numbers are represented in R as vectors of length 1. Together with the recycling rule this is very handy for operations like the one shown below:

```
> v1 <- c(4, 6, 8, 24)
> 2 * v1

[1]  8 12 16 48
```

Notice how the number 2 (actually the vector `c(2)`!) was recycled, resulting in multiplying all elements of `v1` by 2. As we will see, this recycling rule is also applied with other objects, such as arrays and matrices.

2.7 Factors

Factors provide an easy and compact form of handling categorical (nominal) data. Factors have *levels* that are the possible values they can take. Factors are particularly useful in datasets where you have nominal variables with a fixed number of possible values. Several graphical and summarization functions that we will explore in the following chapters take advantage of this type of information. Factors allow you to use and show the values of your nominal variables as they are, which is clearly more interpretable for the user, while internally R stores these values as numeric codes that are considerably more memory efficient (but this is transparent to the user).

Let us see how to create factors in R. Suppose you have a vector with the sex of ten individuals:

```
> g <- c("f", "m", "m", "m", "f", "m", "f", "m", "f", "f")
> g

 [1] "f" "m" "m" "m" "f" "m" "f" "m" "f" "f"
```

You can transform this vector into a factor by:

```
> g <- factor(g)
> g

 [1] f m m m f m f m f f
Levels: f m
```

Notice that you do not have a character vector anymore. Actually, as mentioned above, factors are represented internally as numeric vectors.[19] In this example, we have two levels, 'f' and 'm', which are represented internally as 1 and 2, respectively. Still, you do not need to bother about this as you can use the "original" character values, and R will also use them when showing you the factors. So the coding translation, motivated by efficiency reasons, is transparent to you, as you can confirm in the following example:

```
> g[3]

[1] m
Levels: f m

> g[3] == "m"

[1] TRUE
```

In the above example we asked R to compare the third element of vector `g` with the character value `"m"`, and the answer `TRUE`, which means that R internally translated this character value into the respective code of the factor `g`. Note that if you tried to do `g[3] == m` you would get an error... why?

Suppose you have five extra individuals whose sex information you want to store in another factor object. Suppose that they are all males. If you still want the factor object to have the same two levels as object `g`, you must use the following:

```
> other.g <- factor(c("m", "m", "m", "m", "m"), levels = c("f","m"))
> other.g

[1] m m m m m
Levels: f m
```

Without the `levels` argument the factor `other.g` would have a single level ("m").

One of the many things you can do with factors is to count the occurrence of each possible value. Try this:

```
> table(g)

g
f m
5 5

> table(other.g)

other.g
f m
0 5
```

The `table()` function can also be used to obtain cross-tabulation of several factors.

[19] You can confirm it by typing `mode(g)`.

Suppose that we have in another vector the age category of the ten individuals stored in vector `g`. You could cross-tabulate these two factors as follows:

```
> a <- factor(c('adult','adult','juvenile','juvenile','adult',
+               'adult','adult','juvenile','adult','juvenile'))
> table(a, g)

          g
a          f m
  adult    4 2
  juvenile 1 3
```

Sometimes we wish to calculate the marginal and relative frequencies for this type of contingency table. The following gives you the totals for both the sex and the age factors of this dataset:

```
> t <- table(a, g)
> margin.table(t, 1)

a
   adult juvenile
       6        4

> margin.table(t, 2)

g
f m
5 5
```

The "1" and "2" in the function calls represent the first and second dimensions of the table, that is, the rows and columns of `t`.

For relative frequencies with respect to each margin and overall, we do

```
> prop.table(t, 1)

          g
a                  f         m
  adult    0.6666667 0.3333333
  juvenile 0.2500000 0.7500000

> prop.table(t, 2)

          g
a            f   m
  adult    0.8 0.4
  juvenile 0.2 0.6

> prop.table(t)

          g
a            f   m
  adult    0.4 0.2
  juvenile 0.1 0.3
```

Notice that if we wanted percentages instead, we could simply multiply these function calls by 100 making use of the concept of vectorization we have mentioned before.

2.8 Generating Sequences

R has several facilities to generate different types of sequences. For instance, if you want to create a vector containing the integers between 1 and 100, you can simply type

```
> x <- 1:100
```

which creates a vector called `x` containing 100 elements—the integers from 1 to 100.

You should be careful with the precedence of the operator ":". The following examples illustrate this danger:

```
> 10:15 - 1

[1]  9 10 11 12 13 14

> 10:(15 - 1)

[1] 10 11 12 13 14
```

Please make sure you understand what happened in the first command (remember the recycling rule!).

You may also generate decreasing sequences such as the following:

```
> 5:0

[1] 5 4 3 2 1 0
```

To generate sequences of real numbers, you can use the function `seq()`,

```
> seq(-4, 1, 0.5)

 [1] -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5  0.0  0.5  1.0
```

This instruction generates a sequence of real numbers between -4 and 1 in increments of 0.5. Here are a few other examples of the use of the function `seq()`:[20]

```
> seq(from = 1, to = 5, length = 4)

[1] 1.000000 2.333333 3.666667 5.000000

> seq(from = 1, to = 5, length = 2)

[1] 1 5

> seq(length = 10, from = -2, by = 0.2)

 [1] -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2
```

Another very useful function to generate sequences with a certain pattern is the function `rep()`:

[20] You may want to have a look at the help page of the function (typing, for instance, '`?seq`'), to better understand its arguments and variants.