Pedro J. Aphalo

Learn R

As a Language

Contents

List of Figures

List of Tables

Preface

"Suppose that you want to teach the 'cat' concept to a very young child. Do you explain that a cat is a relatively small, primarily carnivorous mammal with retractible claws, a distinctive sonic output, etc.? I'll bet not. You probably show the kid a lot of different cats, saying 'kitty' each time, until it gets the idea. To put it more generally, generalizations are best made by abstraction from experience."

R. P. Boas
Can we make mathematics intelligible?, 1981

Why the title "*Learn R: As a Language*"? This book is based on exploration and practice that aims at teaching how to express various operations on data using the R language. It focuses on the language, rather than on specific types of data analysis, and exposes the reader to current usage and does not spare the quirks of the language. When we use our native language in everyday life, we do not think about grammar rules or sentence structure, except for the trickier or unfamiliar situations. My aim is for this book to help readers grow to use R in this same way, i.e., to become fluent in R. The book is structured around the elements of languages with chapter titles that highlight the parallels between natural languages like English and the R language.

Learn R: As a Language is different to other books about R in that it emphasizes learning of the language itself, rather than how to use R to address specific data analysis tasks. The aim is to enable readers to use R to implement original solutions to the data analysis and data visualization tasks they encounter. Use of quantitative methods and data analysis has become more frequent in fields with a limited long-term tradition in their use, like humanities, or, the complexity of the methods used has dramatically increased in recent years, like in Biology. Such trends can be expected to continue in the future.

Nowadays, many students of biological and environmental sciences learn R in courses about statistics or data analysis. However, frequently not in enough depth to effectively use it in scripts for automating data analyses or documenting the whole data analysis workflow to ensure reproducibility. Students in the humanities and also in other fields, may find it easier to learn the R language separately from data analysis and statistics. There are also many who are already familiar with statistical principles and wiling to switch from other software to R. *Learn R: As a*

Language is written with these readers in mind to serve both as a text book and as a reference.

A language is a system of communication. Basic concepts and operations are based on abstractions that are shared across programming languages and relevant to programs of all sizes and complexities; these abstractions are explained in the book together with their implementation in the R language. Other abstractions and programming concepts, outside the scope of this book, are relevant to large and complex pieces of software meant to be widely distributed. In other words, *Learn R: As a Language* aims at teaching and supporting *programming in the small*: the use of R to automate the drudgery of data manipulation, including the different steps spanning from data input and exploration to the production of publication-quality illustrations and their documentation.

Using a language actively is the most efficient way of learning it. By using it, I mean actually reading, writing, and running scripts or programs. *Learn R: As a Language* supports learning the R language in a way comparable to how children learn to speak: they work out what the rules are, simply by listening to people speak and trying to utter what they want to tell their parents. Of course, small children also receive guidance through feedback, but they are not taught a prescriptive set of rules like when learning a second language at school. Instead of listening, readers will read and run code, and instead of speaking, readers will write and try to execute R code statements on a computer. I do provide explanations and guidance, as understanding how R works greatly helps with its use. However, the approach I encourage in this book is for readers to play with the numerous examples and variations upon them, to find out by themselves the patterns behind the R language. Instead of parents being the sounding board for the first utterances of readers new to R, the computer will play this role. Although working through the examples in *Learn R: As a Language* in a group of peers or in class is beneficial, the book is designed to be useful also in the absence of such support.

This revised second edition reflects changes that took place in R and packages described. Very few code chunks from the first edition had stopped working but deprecations meant that some examples triggered messages or warnings, and will eventually fail. Recent (> 4.0.0) versions of R have significant enhancements such as the new pipe operator. Packages have also evolved acquiring new features. Feedback from readers and reviewers has highlighted some gaps in the contents and unclear explanations.

An additional change is in my view about some of the packages in the 'tidyverse'. This change is reflected most strongly in Chapter **??**. I have realized by my own experience and from advising other users, including students in the life sciences, that the rate of development and the frequency of code-breaking changes can make some of the *tidyverse* packages difficult for users for whom data analysis is just one aspect of their occupation. In other words, those current and future users to whom this book is targeted. It seems to me that except for packages like 'ggplot2' and 'stringr', much of the current development effort from Posit (formerly RStudio) aims at professional data analysts rather than occasional users of R. There is nothing wrong with this, but it is necessary to be aware that for occasional users learning base R can be a better investment of their time.

Re-reading myself the book after some time allowed me to think of other improvements. I have updated the book accordingly making it more accessible to readers with no previous experience in computer programming. I have added diagrams and flowcharts to facilitate comprehension of common programming abstractions. I also edited the text from the first edition to fix all errors and outdated examples or explanations known to me.

Acknowledgements

I thank Jaakko Heinonen for introducing me to the then new R. Along the way many well known and not so famous experts have answered my questions in usenet and more recently in StackOverflow. I wish to warmly thank members of my own research group, students participating in the courses I have taught, colleagues I have collaborated with, authors of the books I have read and people I have only met online or at conferences. All of them have made it possible for me to write this book. I am indebted to Tarja Lehto, Titta Kotilainen, Tautvydas Zalnierius, Fang Wang, Yan Yan, Neha Rai, Markus Laurel, Brett Cooper, Viivi Lindholm, colleagues, students and anonymous reviewers for many very helpful comments on the draft manuscript and/or the published first edition. Rob Calver, editor of both editions, provided encouragement with great patience, Lara Spieker, Vaishali Singh, Sherry Thomas, and Paul Boyd for their help with different aspects of this project.

In many ways this text owes much more to people who are not authors than to myself. However, as I am the one who has written *Learn R: As a Language* and decided what to include and exclude, I take full responsibility for any errors and inaccuracies.

Helsinki, September 3, 2023

1 Using the Book to Learn R

The important part of becoming a programmer is learning to think like a programmer. You don't need to know the details of a programming language by heart, you can just look that stuff up.

The treasure is in the structure, not the nails.

P. Burns
Tao Te Programming, 2012

1.1 Aims of this chapter

In this chapter I describe how I imagine the book can be used most effectively to learn the R language. Learning R and remembering what one has previously learnt and forgotten makes it also necessary to use this book and other sources as reference. Learning to use R effectively, also involves learning how search for information and learning how to ask questions from other users, for example, through on-line forums. Thus, I also give advice on how to find answers to R-related questions and how to use the available documentation.

1.2 Approach and structure

Depending on previous experience, reading *Learn R: As a Language* will be about exploring a new world or revisiting a familiar one. In both cases *Learn R: As a Language* aims to be a travel guide, neither a traveler's account, nor a cookbook of R recipes. It can be used as a course book, supplementary reading or for self instruction, and also as a reference.

I encourage readers to approach R like a child approaches his or her mother tongue when learning to speak: do not struggle, just play, and fool around with R! If the going gets difficult and frustrating, take a break! If you get a new insight, take a break to enjoy the victory!

In R, like in most "rich" languages, there are multiple ways of coding the same operations. I have included code examples that aim to strike a balance between execution speed and readability. One could write equivalent R books using substantially different code examples. Keep this is mind when reading the book and using R. Keep also in mind that it is impossible to remember everything about R and as a user you will frequently need to consult the documentation, even while doing the exercises in this book. The R language, in a broad sense, is vast because it can be expanded with independently developed packages. Learning to use R mainly consists of learning the basics plus developing the skill of finding your way in R, its documentation and on-line question and answer forums.

Readers should not aim at remembering all the details presented in the book, this is impossible for most of us. Later use of this and other books, and documentation effectively as references, depends on a good grasp of a broad picture of how R works and on learning how to navigate the documentation; i.e., it is more important to remember abstractions and in what situations they are used, and function names, than the details of how to use them. Developing a sense of when one needs to be careful not to fall in a "language trap" is also important.

The book are can be used both as a text book for learning R and as a reference. It starts with simple concepts and language elements progressing towards more complex language structures and uses. Along the way readers will find, in each chapter, descriptions and examples for the common (usual) cases and the exceptions. Some books hide the exceptions and counterintuitive features from learners to make the learning easier, I instead have included these but marked them using icons and marginal bars. There are two reasons for choosing this approach. First, the boundary between boringly easy and frustratingly challenging is different for each of us, and varies depending on the subject dealt with. So, I hope the marks will help readers predict what to expect, how much effort to put in each section and even what to read and what to skip. Second, if I had hidden the tricky bits of the R language, I would have made reader's later use of R more difficult. It would have also made the book less useful as a reference.

The book contains many code examples as well as exercises. I expect readers will run code examples and try as many variations of them as needed to develop an understanding of the "rules" of the R language, e.g., how the function or feature exemplified works. This is what long-time users of R do when facing an unfamiliar feature or a gap in their understanding.

Readers new to R should read at least chapters 2 to **??** sequentially. Possibly, skipping parts of the text and exercises marked as advanced. However, I expect to be most useful to these readers, not to completely skip the description of unusual features and special cases, but rather to skim enough from them so as to get an idea of what special situations they may face as R users. Exercises should not be skipped, as they are a key component of the didactic approach used.

Readers already familiar with R will be able to read the chapters in the book in any order, as the need arises. Marginal bars and icons, and the backwards and forward cross references among sections, make possible for readers to *select suitable path* within the book both when learning R and when using the book as a reference.

I expect *Learn R: As a Language* to remain useful as a reference to those readers who use it to learn R. It will be also useful as a reference to readers already familiar with R. To support the use of the book as a reference, I have been thorough with indexing, including many carefully chosen terms, their synonyms and the names of all R objects and constructs discussed, collecting them in three alphabetical indexes: *General index*, *Index of R names by category*, and *Alphabetic index of R names* starting at pages 26, 29 and 27, respectively. I have also included back and forward cross references linking related sections throughout the whole book.

1.3 Diagrams and Typography

Marginal bars and icons are used in the book to inform about what content is advanced or included with a specific aim. The following icons and colours are used.

Signals ancillary information.

Signals in-depth explanations of specific R features or general programming concepts, which can be skipped on first reading, but to which you should return without hurry, preferably with a cup of coffee or tea.

Signals important bits of information that must be remembered when using R—i.e., explanations of some unusual, but important, feature of the language or concepts that in my experience are easily missed by those new to R.

Signals *playground* sections which contain open-ended exercises—ideas and pieces of R code to play with at the R console. I expect readers to run these examples both as is and after creating variations by editing the code, studying the output, or diagnosis messages, returned by R in each case.

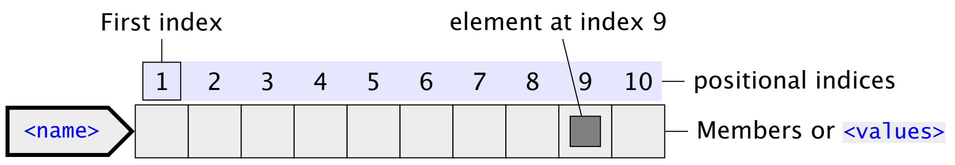
Signals *advanced playground* sections which will require more time to play with before grasping concepts than regular *playground* sections.

Question
Signals a frequently asked question and my answer to it.

Small sections of program code interspersed within the main text, receive the name of *code chunks*. In this book R code chunks are typeset in a typewriter font, using colour to highlight the different elements of the syntax, such as variables, functions, constant values, etc. R code elements embedded in the text are similarly typeset but always black. For example in the code chunk below `mean()` and `print()` are functions; 1, 5 and 3 are constant numeric values, and `z` is the name of a variable where the result of the computation done in the first line of code is stored. The line starting with `##` shows what is printed or shown when executing the second statement: `[1] 1`. In the book `##` is used as a marker to signal output from R, it is not part of the output.

```
z <- mean(1, 5, 3)
print(z)
## [1] 1
```

To describe data objects I use diagrams inspired in Joseph N. Hall's PEGS (Perl Graphical Structures) (Hall and Schwartz 1997). I use colour fill to highlight the type of the stored objects. I use the "signal" sign for the names of whole objects and of their component members, the former with a thicker border.



For code structure I use diagrams based on boxes and arrows, and to describe the flow of code execution, the usual flow charts.

In the different diagrams, I use the notation `<value>`, `<statement>`, `<name>`, etc., as generic placeholders indicating *any valid value*, *any valid R statement*, *any valid R name*, etc.

1.4 Findings answers to problems

1.4.1 What are the options

First of all do not panic! Every programmer, even those with decades of experience, get stuck with problems from time to time, and can run out of ideas for a while. This is normal, and happens to all of us.

It is important to learn how to find answers as part of the routine of using R. First of all one can read the documentation of the function or object that one is trying to use, which in many cases also includes use examples. R's help pages tell how to use individual functions or objects. In contrast, R's manual *An Introduction to R* and books describe what functions or overall approaches to use for different tasks.

Reading the documentation and books not always helps. Sometimes one can become blind to the obvious, by being too familiar with a piece of code, as it also happens when writing in a natural language like English. A second useful step is, thus, looking at the code with "different eyes", those of a friend or workmate, or your own eyes a day or a week later.

One can also seek help in specialized on-line forums or from peers or "local experts". If searching in forums for existing questions and answers fails to yield a useful answer, one can write a new question in a forum.

When searching for answers, asking for advice or reading books, one can be confronted with different ways of approaching the same tasks. Do not allow this to overwhelm you; in most cases it will not matter which approach you use as many computations can be done in R, as in any computer language, in several different ways, still obtaining the same result. Use the alternative that you find easier to understand.

1.4.2 R's built-in help

Every object available in base R or exported by an R extension package (functions, methods, classes, data) is documented in R's help system. Sometimes a single help page documents several R objects. Not only help pages are always available, but they are structured consistently with a title, short description, and frequently also a detailed description. In the case of functions, parameter names, their purpose and expected arguments are always described, as well as the returned value. Usually at the bottom of help pages, several examples of the use of the objects or functions are given. How to access R help is described in section 2.6 on page 15.

In addition to help pages, R's distribution includes useful manuals as PDF or HTML files. These manuals are also available at `https://rstudio.github.io/r-manuals/` restyled for easier reading in web browsers. In addition to help pages, many packages, contain *vignettes* such as User Guides or articles describing the algorithms used and/or containing use case examples. In the case of some packages, a web site with documentation in HTML format is also available. Package documentation can be also accessed through CRAN. The DESCRIPTION of packages provides contact information for the maintainer, links to web sites, and instructions on how to report bugs. Similar information plus a short description are frequently also available in a README file.

Error messages tend to be terse in R, and may require some lateral thinking and/or "experimentation" to understand the real cause behind problems. Learning to interpret error messages is necessary to become a proficient user of R, so forcing errors and warnings with purposely written "bad" code is a useful exercise.

1.4.3 Online forums

Netiquette

When posting requests for help, one needs to abide by what is usually described as "netiquette", which in many respects also applies to asking in person or by e-mail help from a peer or local expert. Preference among sources of information depends on what one finds easier to use. Consideration towards others' time is necessary but has to be sbalanced against wasting too much of one's own time.

In most internet forums, a certain behavior is expected from those asking and answering questions. Some types of misbehavior, like use of offensive or inappropriate language, will usually result in the user losing writing rights in a forum. Occasional minor misbehavior, usually results in the original question not being answered and instead the problem highlighted in a comment. In general following the steps listed below will greatly increase your chances of getting a detailed and useful answer.

- Do your homework: first search for existing answers to your question, both online and in the documentation. (Do mention that you attempted this without success when you post your question.)

- Provide a clear explanation of the problem, and all the relevant information. The version of R, operating system, and any packages loaded and their versions can be important.
- If at all possible, provide a simplified and short, but self-contained, code example that reproduces the problem (sometimes called a *reprex*).
- Be polite.
- Contribute to the forum by answering other users' questions when you know the answer.

Carefully preparing a reproducible example ("reprex") is crucial. A *reprex* is a self-contained and as simple as possible piece of computer code that triggers (and so demonstrates) a problem. If possible, when data are needed, a data set included in base R or artificial data generated within the reprex code should be used. If the problem can only be reproduced with one's own data, then one needs to provide a minimal subset of it that still triggers the problem.

While preparing a *reprex* one has to simplify the code, and sometimes this step makes clear the nature of the problem. Always, before posting a reprex online, check it with the latest versions of R and any package being used. If sharing data, be careful about confidential information and either remove or mangle it.

I must say that about two out of three times I prepare a *reprex*, it allows me to find the root of the problem and a solution or a work-around on my own. Preparing a *reprex* takes some effort but it is worthwhile even if it ends up not being posted on-line.

R package 'reprex' and its RStudio add-in simplify the creation of reproducible code examples, by creating and copying to the clipboard a reprex encoded in Markdown and ready to be pasted into a question at StackOverflow or into an issue at GitHub. See `https://reprex.tidyverse.org/` for details.

StackOverflow

Nowadays, StackOverflow (`http://stackoverflow.com/`) is the best question-and-answer (Q&A) support site for R. Within this site there is an R collective. In most cases, searching for existing questions and their answers, will be all that you need to do. If asking a question, make sure that it is really a new question. If there is some question that looks similar, make clear how your question is different.

StackOverflow has a user-rights system based on reputation, and questions and answers can be up- and down-voted. Questions with the most up-votes are listed at the top of searches, and the most voted answers to each question are also displayed first. Who asks a question is expected to accept correct answers. If the questions or answers one writes are up-voted one gains reputation (expressed as number). As one accumulates reputation, gets badges and additional rights, such as editing other users' questions and answers or later on, even deleting wrong answers or off-topic questions from the system. This sounds complicated, but works extremely well at ensuring that the base of questions and answers is relevant and correct, without relying heavily on nominated *moderators*. When using StackOverflow, do contribute by accepting correct answers, up-voting questions and answers that you find useful, down-voting those you consider poor, and flagging or correcting errors you may discover.

Being careful in the preparation of a reproducible example is important both when asking a question at StackOverflow and when reporting a bug to the maintainer of any piece of software. For the question to be reliably answered or the problem to be fixed, the person answering a question, needs to be able to reproduce the problem, and after modifying the code, needs to be able to test if the problem has been solved or not. However, even if you are facing a problem caused by your misunderstanding of how R works, the simpler the example, the more likely that someone will quickly realize what your intention was when writing the code that produces a result different from what you expected. Even when it is not possible to create a reprex, one needs to ask clearly only one thing per question.

Contacting the author

The best way to get in contacting with me about this book is by rasing an issue at `https://github.com/aphalo/learnr-book-crc/issues`. Issues can be used both to ask for support questions related to the book, report mistakes and suggest changes to the text, diagrams and/or example code. Edits to the manuscript of this book can be submitted as pull requests.

Issues are raised by filling-in an on-line form, at a web page that also contains brief instructions. Git issues are a very efficient way of keeping track of corrections that need to be done. As support questions usually reveal unclear explanations or other problems, raising issues to ask them facilitates the tasks of improving and keeping the book up-to-date.

1.5 Further reading

At the end of each chapter a section like this one gives suggestions for further reading. To understand what programming as an activity is, do read *Tao Te Programming* (Burns 2012), it will make easier the learning of programming in R, both practically and emotionally. In Burns's words "This is a book about what goes on in the minds of programmers".

2

R and Data Analysis

In a world of … relentless pressure for more of everything, one can lose sight of the basic principles—simplicity, clarity, generality—that form the bedrock of good software.

Brian W. Kernighan and Rob Pike
The Practice of Programming, 1999

2.1 Aims of this chapter

First, I will describe the steps in a typical scientific or technical study, the data analysis work flow and the roles that R can play in it. Some facts about the history and design aims behind the R language will provide you a better vantage point to grasp the logic behind R's features, making it easier understand and remember them.

You will learn how to use R in practice when sitting at a computer. You will learn the difference between typing commands interactively, reading each partial response from R on the screen as you type versus using R scripts to execute a "job" which saves results for later inspection by the user.

I will consider the advantages and disadvantages of textual command languages such as R compared to menu-driven user interfaces as frequently used in other statistics software and occasionally also with R. I will discuss the role of textual languages in the very important question of reproducibility of data analyses.

Finally you will learn about the different types and sources of help available to R users, and how to best make use of them.

2.2 The research process

Statistics are not only important after the fact but also crucial at the design stage of a study. Rather frequently, we deal with existing data from the real world or from model simulations already at the planning stage of an experiment or survey. Statistics gives support to data analysis and data visualization, like grammar and vocabulary give support to textual communication. Statistics is both a tool that supports data analysis and decision-making based on evidence, but also a means of communication. R can be used profitably throughout all stages of the research process, from study design to communication of the results.

In research, the path from data acquisition to conclusions and their communication can be described as a work flow, usually not linear, as repeated attempts at extracting information from a set of observations are in most cases beneficial. In modern data analysis data visualization plays a central role both for "quality control" and in the discovery of interesting features, in addition to communication or results. Although, most research experiments are unique, the overall design patterns can repeat, while in monitoring, consistency over time is crucial. In both cases it is important to document all steps in detail.

2.3 Reproducible data analysis

Under any situation where accountability is important, from scientific research to decision making in commercial enterprises, industrial quality control and safety and environmental impact assessments, being able to reproduce a data analysis reaching the same conclusions from the same data is crucial. Most approaches to reproducible data analysis are based on automating report generation and including, as part of the report, all the computer commands used to generate the results presented.

A fundamental requirement for reproducibility is a reliable record of what commands have been run on which data. Such a record is especially difficult to keep when issuing commands through menus and dialogue boxes in a graphical user interface or interactively at a console. Even working interactively at the R console using copy and paste to include commands and results in a report is error prone, and laborious.

A further requirement is to be able to preserve a link between the output of the R commands to the input. If the script saves the output to separate files, then the user will need to take care that the script saved or shared as a record of the data analysis was the one actually used for obtaining the reported results and conclusions. This is another error-prone stage in the reporting of data analysis. To solve this problem an approach was developed, inspired in what is called *literate programming* (Knuth 1984). The idea is that running the script will produce a document that includes the listing of the R code used, the results of running this code and any explanatory text needed to understand and interpret the analysis.

Although a system capable of producing such reports with R, called 'Sweave' (Leisch 2002), has been available for a couple decades, it was rather limited and not supported by an IDE, making its use rather tedious. A more recently developed system called 'knitr' (Xie 2013) together with its integration into RStudio has made the use of this type of reports very easy. A further development called R *notebooks* created within RStudio can create a readable report as an HTML file from an ordinary R script. This HTML file shows the code used interspersed with the results within the viewable file as in earlier approaches. However, this newer approach goes even further: the actual source script used to generate the report is embedded in the HTML file of the report and can be extracted and run very easily and consequently re-used. This means that anyone who gets access to the output of the analysis in human readable form also gets access to the code used to generate the report, in computer executable format.

Package 'knitr' supports the writing of reports with the text marked using Markdown or LaTeX. The recently released Quarto (see https://quarto.org/) is an enhancement of R markdown (see https://rmarkdown.rstudio.com/), mainly improving typesetting and styling, but also providing a single system capable of generating a broad selection of outputs.

Because of these recent developments, R is an ideal language to use when the goal of reproducibility is important. During recent years the problem of the lack of reproducibility in scientific research has been broadly discussed and analysed (Gandrud 2015). One of the problems faced when attempting to reproduce experimental work, is reproducing the data analysis. R together with these modern tools can help in avoiding this source of lack of reproducibility.

How powerful are these tools and how flexible? They are powerful and flexible enough to write whole books, such as this very book you are now reading, produced with R, 'knitr' and LaTeX. All pages in the book are generated directly, all figures are generated by R and included automatically, except for the figures in this chapter that have been manually captured from the computer screen. Why am I using this approach? First because I want to make sure that every bit of code as you will see printed, runs without error. In addition, I want to make sure that the output that you will see below every line or chunk of R language code is exactly what R returns. Furthermore, it saves a lot of work for me as author, as I can just update R and all the packages used to their latest version, and build the book again, to keep it up to date and free of errors. By using these tools and markup in plain text files, the indices, cross-references, citations and list of references are all generated automatically.

Although the use of these tools is important, they are outside the scope of this book and well described in other books (Gandrud 2015; Xie 2013).

2.4 Computer programming

As with natural language writing or the creation of any new device, we can distinguish two phases in the development of a computer program or script. The design phase is the initial step: deciding what the computer should do and what algorithms will be used. Coding is the second phase, and consists in translating a design into a given computer language, such as R. The distinction is not absolutely clear cut, as usually when programming one re-uses available code. In a language like C++ we use libraries of routines, classes and templates. In R we use *packages* that provide extensions to the language (see section **??** on page **??**). So, in most cases, the design stage for a data-analysis script in R centres, once the statistical procedure to use has been decided, in selecting what package, if any, to use, and the identification of the steps needed to import the data, possibly validate them, pass them to the functions in the packages used, and reporting the results either graphically or as text. In fact, most of the ad-hoc data-analysis code users write in their scripts is to transfer data among ready made “black boxes” and display the results. In contrast, writing new packages, requires much more effort towards design, of both computations and the interface to users’ code. In this book, we focus on the design of scripts and their coding.

Abstraction plays a central role in designing solutions suitable for families of similar problems. According to Wirth (1974) “Our most important mental tool for coping with complexity is abstraction. Therefore, a complex problem should not be regarded immediately in terms of computer instructions … but rather in terms and entities natural to the problem itself, abstracted in some suitable sense.” Zimmer (1985) adds “Abstraction is the way we carry out a divide-and-conquer approach to the solution of complex problems.” A simple example of an abstraction centred on objects is the concept of *fruit* that describes properties shared among apples, oranges, pears, and many other fruits. An example of an abstraction centred on an action is the verb *show*, which depending on the context may signify different actions that share similar aims or purposes.

New concepts and styles of programming have appeared since Wirth and Zimmer wrote the texts quoted above and new terms are in use, but the role of abstraction remains as important. An important distinction is in the focus of the abstractions: actions vs. objects. These, oversimplifying things, give rise to procedural and object-oriented approaches (or paradigms) to computer programming, respectively. Which approach yields the most useful abstraction of a problem depends on the nature of the problem (see Coplien 1999). As we will see through the book, the R language is eclectic in this respect, and supports multiple approaches and their combined use. R itself relies quite heavily on a rather simple approach to object-oriented programming. When writing scripts, it is unusual to define new classes of objects, but in almost every script we make use of classes of objects and their corresponding methods, both defined in R and in extension packages. R also supports functional programming because functions are treated similarly to other objects and can be saved and operated upon. It is even possible in R to write functions that accept other functions as arguments and/or dynamically construct new functions and return them.

2.5 What is R?

In the case of languages like C++, C, Pascal and FORTRAN multiple software implementations exist (different compilers and interpreters), some free and some commercial. So in addition to different flavours of each language stemming from different definitions, e.g., versions of international standards, different implementations of the same standard may have, usually small, unintentional and intentional differences.

Most people think of R as a computer program, similar to SAS or SPPS. R is indeed a computer program—a piece of software— but it is also a computer language, implemented in the R program. At the moment, differently to with most other computer languages, this difference is not important as the R program is the only widely used implementation of the R language. Furthermore, while SAS or SPPS support their own scripting languages, these programs are frequently used through a menu-based interface.

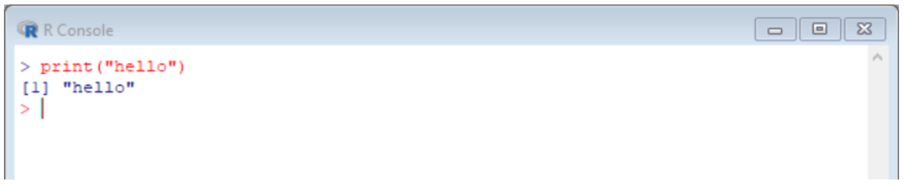
2.5.1 R as a language

R is a computer language designed for data analysis and data visualization, however, in contrast to some other scripting languages, it is, from the point of view of computer programming, a complete language—it is not missing any important feature. In other words, no fundamental operations or data types are lacking (Chambers 2016). I attribute much of its success to the fact that its design achieves a very good balance between simplicity, clarity and generality. R excels at generality thanks to its extensibility at the cost of only a moderate loss of simplicity, while clarity is ensured by enforced documentation of extensions and support for both object-oriented and functional approaches to programming. The same three principles can be also easily followed by user code written in R.

R started as a partial implementation of the then relatively new S language (Becker and Chambers 1984; Becker et al. 1988). When designed, S, developed at Bell Labs, in the U.S.A. provided a novel way of carrying out data analyses. S evolved into S-Plus (Becker et al. 1988). S-Plus was available as a commercial program, most recently from TIBCO, U.S.A. R started as a poor man's home-brewed implementation of S, for use in teaching, developed by Robert Gentleman and Ross Ihaka at the University of Auckland, in New Zealand. Initially R, the program, implemented a subset of the S language. The R program evolved until only relatively few differences between S and R remained. These remaining differences are intentional—thought of as significant improvements. In more recent times R overtook S-Plus in popularity. The R language is not standardised, and no formal definition of its grammar exists. Consequently, the R language is defined by the behavior of its implementation in the R program.

What makes R different from SPSS, SAS, etc., is that S was designed from the start as a computer programming language. This may look unimportant for someone not actually needing or willing to write software for data analysis. However, in reality it makes a huge difference because R is easily extensible. By this we mean that new functionality can be easily added, and easily shared. In other words, instead of having to switch between different pieces of software to do different types of analyses or plots, one can usually find a package that will make seamlessly available within R the needed tools. The name "base R " is used to distinguish R itself, as in the R executable included in the R distribution, from R in a broader sense, which includes packages. A few packages are included in the R distribution, but most R packages are independently developed extensions and separately distributed.

The most important advantage of using a language like R is that instructions to the computer are given as text. This makes it easy to repeat or *reproduce* a data analysis. Textual instructions serve to communicate to other people what



```
> print("hello")
[1] "hello"
>
```

FIGURE 2.1
The R console where the user can type textual commands one by one. Here the user has typed `print("Hello")` and *entered* it by ending the line of text by pressing the "enter" key. The result of running the command is displayed below the command. The character at the head of the input line, a ">" in this case, is called the command prompt, signaling where a command can be typed in. Commands entered by the user are displayed in red, while results returned by R are displayed in blue. "`[1]`" can be ignored here, its meaning is explained on page **??**

.

has been done in a way that is unambiguous. Sharing the instructions themselves avoids a translation from a set of instructions to the computer into text readable to humans—say the materials and methods section of a paper. In the case of an attempt to reproduce the data analysis, a further translation back into computer instructions is also avoided, together with the ambiguities usually creeping in.

Readers with programming experience, will notice that some features of R differ from those in other programming languages. R does not have the strict type checks of Pascal or C++. It has operators that can take vectors and matrices as operands. Reliable and fast R code, tends to rely on different *idioms* than well-written Pascal or C++ code.

2.5.2 R as a computer program

The R program itself is open-source, i.e., its source code is available for anybody to inspect, modify and use. A small fraction of users will directly contribute improvements to the R program itself, but it is possible, and those contributions are important in making R extremely reliable. The executable, the R program we actually use, can be built for different operating systems and computer hardware. The members of the R developing team aim to keep the results obtained from calculations done on all the different builds and computer architectures as consistent as possible. The idea is to ensure that computations return consistent results not only across updates to R but also across different operating systems like Linux, Unix (including OS X), and MS-Windows, and computer hardware.

The R program does not have a graphical user interface (GUI), or menus from which to start different types of analyses. Instead, the user types the commands at the R console and the result is displayed starting on the next line (Figure 2.1). The same textual commands can also be saved into a text file, line by line, and such a file, called a "script" can substitute repeated typing of the same sequence of commands. When we work at the console, typing in commands one by one, we use R *interactively*. When we run a script, we may say that we run a "batch job." The two approaches described above are available in the R program itself.

As R is essentially a command-line application, it can be used on what nowadays are frugal computing resources, equivalent to a personal computer of three decades ago. R can run even on the Raspberry Pi, a micro-controller board with the processing power of a modest smart phone (see `https://r4pi.org/`). At the other end of the spectrum, on really powerful servers, R can be used for the analysis of big data sets with millions of observations. How powerful a computer is needed for a given data analysis task depends on the size of the data sets, on how patient one is, on the ability to select efficient algorithms and on writing "good" code.

2.6 Using R

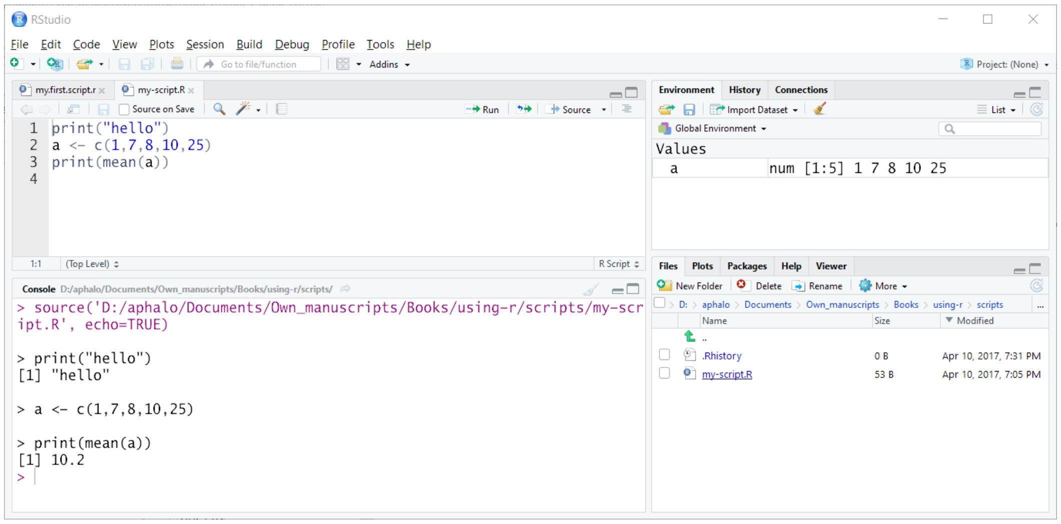
2.6.1 Editors and IDEs

Integrated Development Environments (IDEs) are used when developing computer programs. IDEs provide a centralized user interface from within which the different tools used to create and test a computer program can be accessed and used in coordination. Most IDEs include a dedicated editor capable of syntax highlighting, and even report some mistakes, related to the programming language in use. One could describe such an editor as the equivalent of a word processor with spelling and grammar checking, that can alert about spelling and syntax errors for a computer language like R instead of for a natural language like English.

It is nowadays very common to use an IDE as a front-end or middleman between the user and the R program. Computations are still done the R program, which is *not* built-in in the IDE. Of the available IDEs for R, RStudio is currently the most popular by a wide margin. Recent versions of RStudio support Python in addition to R.

Readers with programming experience may be already familiar with Microsoft's free Visual Studio Code or the open-source Eclipse IDE's for which add-ons supporting R are available.

RStudio and other IDEs provide a more comfortable user interface (UI) to R by making easier to edit and run code scripts. They also allow easier access to R help. Many menu entries and dialogue boxes in RStudio and other IDEs give access to various tools, frequently by calling behind the scenes R functions. It is important to keep this in mind, as R at least when working with small data sets needs much less computing power and memory resources than ÍDEs like RStudio. R scripts created in IDEs run unchanged in the different IDEs and in their absence, as IDEs call the R behind the scenes. A simpler option, requiring less memory and speed in the computer, is to use a text editor to edit the scripts and then run the scripts in R from within the editor, or directly.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
my.first.script.r
my-script.R
Source on Save
Run
Source
(Top Level)
R Script
Environment History Connections
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
New Folder Delete Rename More

```
print("hello")
a <- c(1,7,8,10,25)
print(mean(a))
```

```
Console D:/aphalo/Documents/Own_manuscripts/Books/using-r/scripts/
> source('D:/aphalo/Documents/Own_manuscripts/Books/using-r/scripts/my-scr
ipt.R', echo=TRUE)

> print("hello")
[1] "hello"

> a <- c(1,7,8,10,25)

> print(mean(a))
[1] 10.2
>
```

```
Values
  a           num [1:5] 1 7 8 10 25
```

D: › aphalo › Documents › Own_manuscripts › Books › using-r › scripts

Name	Size	Modified
..		
.Rhistory	0 B	Apr 10, 2017, 7:31 PM
my-script.R	53 B	Apr 10, 2017, 7:05 PM

FIGURE 2.2
The RStudio interface just after running the same script. Here we used the "Source" button to run the script. In this case, R prints the results to the R console in the lower left pane.

This book provides only a minimum of guidance on the use of RStudio, and no guidance for other IDEs. Additional instructions on the use of RStudio are available through the Resources menu entry at the book website at `https://www.learnr-book.info/`. To learn more about RStudio, please, read the documentation available through RStudio's help menu and keep at hand a printed copy of the RStudio cheat sheet while learning how to use it. This and other R-related cheatsheets can be downloaded at `https://posit.co/resources/cheatsheets/`.

The main window of IDEs is in most cases divided into windows or panes, possibly with tabs. In RStudio one has access to the R console, a text editor, a file-system browser, a pane for graphical output, and access to several additional tools such as for installing and updating extension packages. Although RStudio supports very well the development of large scripts and packages, it is currently, in my opinion, also the best possible way of using R at the console as it has the R help system very well integrated both in the editor and R console. Figure 2.2 shows the main window displayed by RStudio after running the same script as shown above at the R console (Figure 2.5) and at the operating system command prompt (Figure 2.6). We can see by comparing these three figures how RStudio is really a layer between the user and an unmodified R executable. The script was sourced by pressing the "Source" button at the top of the editor pane. RStudio, in response to this, generated the code needed to source the file and "entered" it at the console, the same console, where we can directly type in these same R commands if we wish.

When a script is run, if an error is triggered, RStudio automatically finds the location of the error. Some features are beyond what one needs for simple everyday data analysis and aimed at package development and report generation.

Integration of debugging tools, trace-back on errors, code profiling, bench marking of code and unit tests, make it possible to analyze and improve performance as well help with quality assurance and certification. It also integrates support for file version control, which is not only useful for package development, but also for keeping track of the progress or work together with collaborators in the analysis of data.

The "desktop" version of RStudio that one installs locally, runs on most modern operating systems, such as Linux, Unix, OS X, and MS-Windows. There is also a server version that runs on Linux, as well as a cloud service (https://posit.cloud/) providing shared access to such a server. The RStudio server is used remotely through in a web browser. The user interface is almost the same in all cases. Desktop and server versions are both distributed as unsupported free software and supported commercial software.

RStudio and other IDEs support saving of their state and settings per working folder under the name of *project*, so that work on a data analysis can be interrupted and later continued, even on a different computer. As mentioned in section 2.6.3 on page 18, when working with R we keep related files in a folder.

⚠ RStudio projects are implemented as a folder with a name ending in `.Rprj`, located under the same folder where scripts, data, `.Rdata` and `.Rhistory` files are stored. This folder is managed by RStudio and should be not modified or deleted by the user. Only in the very rare case of its corruption, it should be deleted, and the project created again from scratch.

2.6.2 Installation

Installation of R varies depending on the operating system and computer hardware, and is in general similar to that of other software under a given operating system distribution. For most types of computer hardware the current version of R is available through the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/. Especially in the case of Linux distributions, R can frequently be installed as a component of the operating system distribution. There are some exceptions, such as the *R4Pi* distribution of R for the Raspberry Pi, which is maintained independently (https://r4pi.org/).

Installers for Linux, Windows and MacOS are available through CRAN (https://cran.r-project.org/) together with brief but up-to-date installation instructions. RStudio installers are available at Posit's web site (https://posit.co/products/open-source/rstudio/) of which the free version is suitable for running the code examples and exercises in the book. In many cases the IT staff at your employer or school will install them, or they may be already included in the default computer setup.

An alternative, that is very well suited for courses or learning as part of a group is the RStudio cloud service, recently renamed Posit cloud (https://posit.co/products/cloud/cloud/). For individual use a free account is in many cases enough and for groups a low cost teacher's account works very well.

Code examples make use of some freely available R extension packages, which can be installed from CRAN. (How to install and use packages is described in section **??** on page **??**.) One package, 'learnrbook', also available through CRAN, contains data sets and files specific to this book. Package 'learnrbook' contains installation instructions and saved lists of the names of all other packages used in the book. The package also contains one script file per chapter with the R code for all the examples and exercises.

Please, visit `https://www.learnr-book.info/` for up-to-date instructions, as these may change after the publication of the book.

2.6.3 R sessions and workspaces

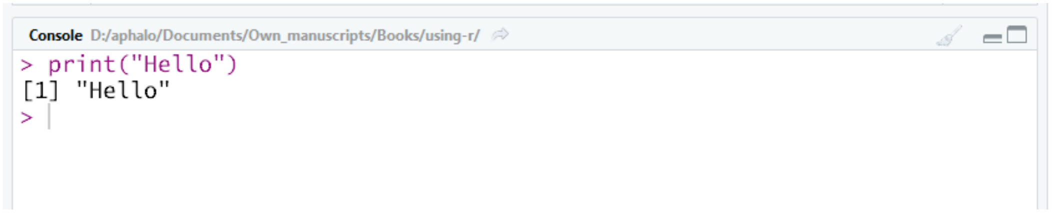
We use *session* to describe the interactive execution from start to finish of one running instance or R. We use *workspace* to name the imaginary space were all objects currently available in a session are stored. In R the whole workspace can be stored in file on disk at the end or during a session and restored later into another session, possibly on a different computer. Usually when working with R we dedicate a folder in disk storage to store all files from a given data analysis project. We normally keep in this folder files with data to read in, scripts, a file storing the whole contents of the workspace, named by default `.Rdata` and a text file with the history of commands entered interactively, named by default `.Rhistory`. The user's files within this folder can be located in nested folders. There are no strict rules on how the files should be organised or on their number. The recommended practice is to avoid crowded folders and folders containing unrelated files. It is a good idea to keep in a given folder and workspace the work in progress for a single data-analysis project or experiment, so that the workspace can be saved and restored easily between sessions and work continued from where one left it independently of work done on other workspaces. The folder where files are currently read and saved is in R documentation called the *current working directory*. When opening an `.Rdata` file the current working directory is automatically set to the folder where the `.Rdata` file was read from.

As described below, if we use a front-end program, it will save additional files, possibly a whole hierarchy of folders, to keep track of its own state and local settings between sessions. If we use a program like git to track our edits to R scripts and changes in other files in the folder, additional folders and files will be kept within the same folder.

2.6.4 Using R interactively

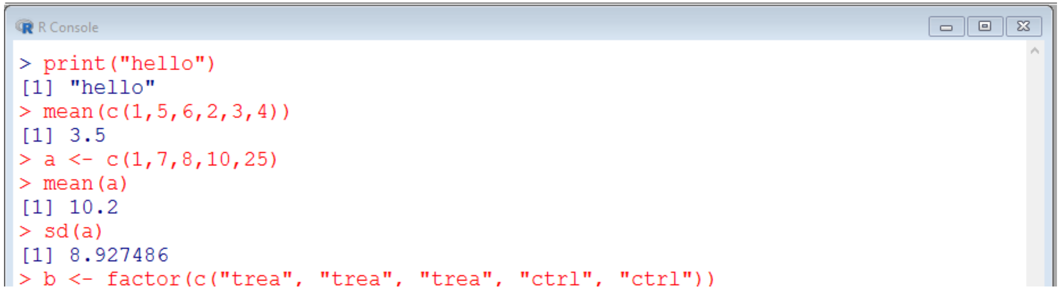
Decades ago users communicated with computers through a physical terminal (keyboard plus text-only screen) that was frequently called a *console*. A text-only interface to a computer program, in most cases a window or a pane within a graphical user interface, is still called a console. In our case, the R console (Figure 2.1). This is the native user interface of R.

Typing commands at the R console is useful when one is playing around, rather aimlessly exploring things, or trying to understand how an R function or operator we are not familiar with works. Once we want to keep track of what we are doing, there are better ways of using R, which allow us to keep a record of how an analysis has been carried out. The different ways of using R are not exclusive of each other,



```
> print("Hello")
[1] "Hello"
>
```

FIGURE 2.3
The R console embedded in RStudio. The same commands have been typed in as in Figure 2.1. Commands entered by the user are displayed in purple, while results returned by R are displayed in black.



```
> print("hello")
[1] "hello"
> mean(c(1,5,6,2,3,4))
[1] 3.5
> a <- c(1,7,8,10,25)
> mean(a)
[1] 10.2
> sd(a)
[1] 8.927486
> b <- factor(c("trea", "trea", "trea", "ctrl", "ctrl"))
```

FIGURE 2.4
The R console after several commands have been entered. Commands entered by the user are displayed in red, while results returned by R are displayed in blue.

so most users will use the R console to test individual commands and plot data during the first stages of exploration. As soon as we decide how we want to plot or analyze the data, it is best to start using scripts. This is not enforced in any way by R, but scripts are what really brings to light the most important advantages of using a programming language for data analysis. In Figure 2.1 we can see how the R console looks. The text in red has been typed in by the user, except for the prompt >, and the text in blue is what R has displayed in response. It is essentially a dialogue between user and R. The console can *look* different when displayed within an IDE like RStudio, but the only difference is in the appearance of the text rather than in the text itself (cf. Figures 2.1 and 2.3).

The two previous figures showed the result of entering a single command. Figure 2.4 shows how the console looks after the user has entered several commands, each as a separate line of text.

The examples in this book require only the console window for user input. Menu-driven programs are not necessarily bad, they are just unsuitable when there is a need to set very many options and choose from many different actions. They are also difficult to maintain when extensibility is desired, and when independently developed modules of very different characteristics need to be integrated. Textual languages also have the advantage, to be addressed in later chapters, that command sequences can be stored in human- and computer-readable text files. Such files constitute a record of all the steps used, and in most cases, makes it trivial to manually reproduce the same steps at a later time. Scripts are a very simple and handy way of communicating to other users how a given data analysis has been done or can be done.

In the console one types commands at the `>` prompt. When one ends a line by pressing the return or enter key, if the line can be interpreted as an R command, the result will be printed at the console, followed by a new `>` prompt. If the command is incomplete, a `+` continuation prompt will be shown, and you will be able to type in the rest of the command. For example if the whole calculation that you would like to do is $1 + 2 + 3$, if you enter in the console `1 + 2 +` in one line, you will get a continuation prompt where you will be able to type `3`. However, if you type `1 + 2`, the result will be calculated, and printed.

For example, one can search for a help page at the R console.

Below is the first code example in the book. To run this example you first have to start the R program and then type the code shown below at the command prompt.

```
help("sum")
?sum
```

Look at help for some other functions like `mean()`, `var()`, `plot()` and, why not, `help()` itself!

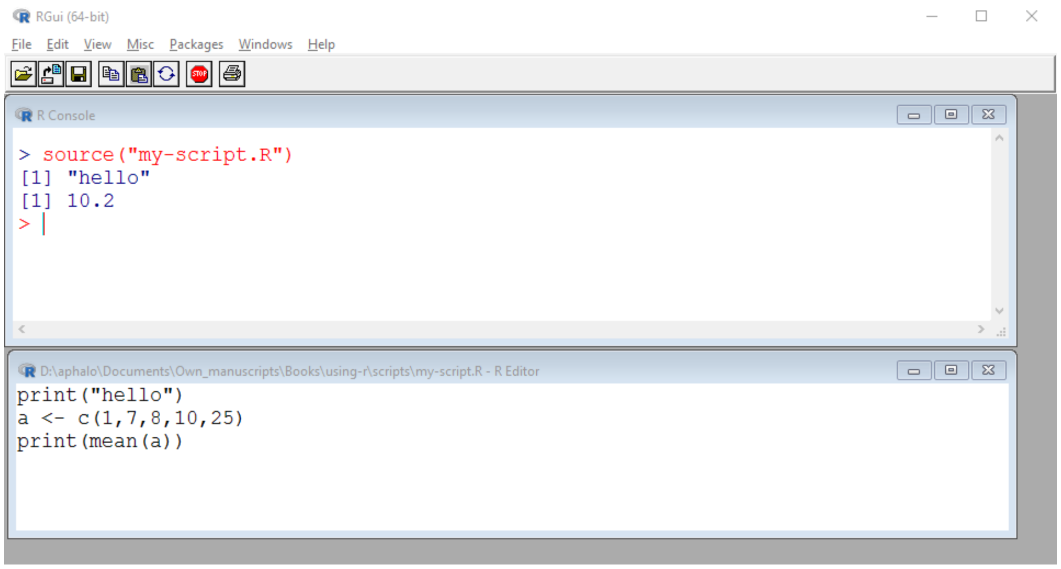
```
help(help)
```

When trying to access help related to R extension packages trough R's built in help, make sure the package is loaded into the current R session, as described on page ??, before calling `help()`.

When using RStudio there are easier ways of navigating to a help page than calling function `help()` by typing its name, for example, with the cursor on the name of a function in the editor or console, pressing the F1 key opens the corresponding help page in the help pane. Letting the cursor hover for a few seconds over the name of a function at the R console will open "bubble help" for it. If the function is defined in a script or another file that is open in the editor pane, one can directly navigate from the line where the function is called to where it is defined. In RStudio one can also search for help through the graphical interface. The R manuals can also be accessed most easily through the Help menu in RStudio or RGUI.

2.6.5 Using R in a "batch job"

To run a script we need first to prepare a script in a text editor. Figure 2.5 shows the console immediately after running the script file shown in the text editor. As before, red text, the command `source("my-script.R")`, was typed by the user, and the blue text in the console is what was displayed by R as a result of this action.


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
D:\aphalo\Documents\Own_manuscripts\Books\using-r\scripts\my-script.R - R Editor

```
> source("my-script.R")
[1] "hello"
[1] 10.2
> |
```

```
print("hello")
a <- c(1,7,8,10,25)
print(mean(a))
```

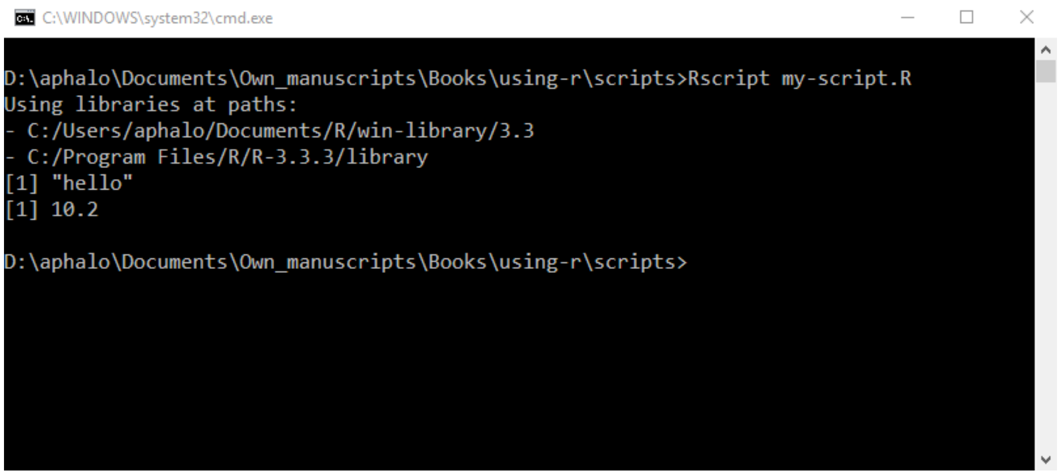
FIGURE 2.5
Screen capture of the R console and editor just after running a script. The upper pane shows the R console, and the lower pane, the script file in an editor.

The title bar of the console, shows “R-console,” while the title bar of the editor shows the *path* to the script file that is open and ready to be edited followed by “R-editor.”

⚠ When working at the command prompt, most results are printed by default. However, within scripts one needs to use function `print()` explicitly when a result is to be displayed.

A true “batch job” is not run at the R console but at the operating system command prompt, or shell. The shell is the console of the operating system—Linux, Unix, OS X, or MS-Windows. Figure 2.6 shows how running a script at the Windows command prompt looks. A script can be run at the operating system prompt to do time-consuming calculations with the output saved to a file. One may use this approach on a server, say, to leave a large data analysis job running overnight or even for several days.

Within RStudio desktop it is possible to access the operating system shell through the tab named “Terminal” and through the menu. It is also possible to run jobs in the background in tab “Background jobs”, i.e., while simultaneously using the R console. This is made possible by concurrently running two or more instances of the R program.



```
D:\aphalo\Documents\Own_manuscripts\Books\using-r\scripts>Rscript my-script.R
Using libraries at paths:
- C:/Users/aphalo/Documents/R/win-library/3.3
- C:/Program Files/R/R-3.3.3/library
[1] "hello"
[1] 10.2

D:\aphalo\Documents\Own_manuscripts\Books\using-r\scripts>
```

FIGURE 2.6
Screen capture of the MS-Windows command console just after running the same script. Here we use `Rscript` to run the script; the exact syntax will depend on the operating system in use. In this case, R prints the results at the operating system console or shell, rather than in its own R console.

2.7 Further reading

Suggestions for further reading are dependent on how you plan to use R. If you envision yourself running batch jobs under Linux or Unix, you would profit from learning to write shell scripts. Because bash is widely used nowadays, *Learning the bash Shell* (Newham and Rosenblatt 2005) can be recommended. If you aim at writing R code that is going to be reused, and have some familiarity with C, C++ or Java, reading *The Practice of Programming* (Kernighan and Pike 1999) will provide a mostly language-independent view of programming as an activity and help you master the all-important tricks of the trade. The history of R is best told by some of those who witnessed or were involved at early stages of its development such as (Chambers 2016), (**Ihaka1998**) `https://www.stat.auckland.ac.nz/~ihaka/downloads/Interface98.pdf`.

Bibliography

Becker, R. A. and J. M. Chambers (1984). *S: An Interactive Environment for Data Analysis and Graphics*. Chapman and Hall/CRC. ISBN: 0-534-03313-X (cit. on p. 13).

Becker, R. A., J. M. Chambers, and A. R. Wilks (1988). *The New S Language: A Programming Environment for Data Analysis and Graphics*. Chapman & Hall. ISBN: 0-534-09192-X (cit. on p. 13).

Boas, R. P. (1981). "Can we make mathematics intelligible?" In: *The American Mathematical Monthly* 88.10, pp. 727–731.

Burns, P. (2012). *Tao Te Programming*. Lulu. ISBN: 9781291130454 (cit. on p. 7).

Chambers, J. M. (2016). *Extending R*. The R Series. Chapman and Hall/CRC. ISBN: 1498775713 (cit. on pp. 13, 22).

Coplien, J. O. (1999). *Multi-paradigm design for C++*. Addison-Wesley, p. 280. ISBN: 0201824671 (cit. on p. 12).

Gandrud, C. (2015). *Reproducible Research with R and R Studio*. 2nd ed. Chapman & Hall/CRC The R Series). Chapman and Hall/CRC. 323 pp. ISBN: 1498715370 (cit. on p. 11).

Hall, J. N. and R. L. Schwartz (1997). *Effective Perl Programming. Writing Better Programs with Perl*. Addison-Wesley. 288 pp. ISBN: 9780201419757 (cit. on p. 4).

Kernighan, B. W. and R. Pike (1999). *The Practice of Programming*. Addison Wesley. 288 pp. ISBN: 020161586X (cit. on p. 22).

Knuth, D. E. (1984). "Literate programming". In: *The Computer Journal* 27.2, pp. 97–111 (cit. on p. 10).

Leisch, F. (2002). "Dynamic generation of statistical reports using literate data analysis". In: *Proceedings in Computational Statistics*. Compstat 2002. Ed. by W. Härdle and B. Rönz. Heidelberg, Germany: Physika Verlag, pp. 575–580. ISBN: 3-7908-1517-9 (cit. on p. 10).

Newham, C. and B. Rosenblatt (June 1, 2005). *Learning the bash Shell*. O'Reilly UK Ltd. 352 pp. ISBN: 0596009658 (cit. on p. 22).

Wirth, N. (1974). "On the Composition of Well-Structured Programs". In: *Computing Surveys* 6.4, pp. 247–259. DOI: `10.1145/356635.356639` (cit. on p. 12).

Xie, Y. (2013). *Dynamic Documents with R and knitr*. The R Series. Chapman and Hall/CRC, p. 216. ISBN: 1482203537 (cit. on pp. 10, 11).

Zimmer, J. A. (1985). *Abstraction for Programmers*. New York: McGraw-Hill, p. 251. ISBN: 0070728321 (cit. on p. 12).

General Index

Alphabetic Index of R Names

Index of R Names by Category

R names and symbols grouped into the categories ‘classes and modes’, ‘constant and special values’, ‘control of execution’, ‘data objects’, ‘functions and methods’, ‘names and their scope’, and ‘operators’.