Advanced R
Second Edition

Hadley Wickham
To Mina, the best book writing companion.
We miss you.
Contents

Preface xiii

1 Introduction 1
  1.1 Why R? .................................................. 1
  1.2 Who should read this book .......................... 3
  1.3 What you will get out of this book .................. 4
  1.4 What you will not learn ............................. 4
  1.5 Meta-techniques ........................................... 5
  1.6 Recommended reading ............................... 5
  1.7 Getting help ............................................. 6
  1.8 Acknowledgments ....................................... 7
  1.9 Conventions ............................................. 10
  1.10 Colophon ................................................ 12

I Foundations 13

Introduction 15

2 Names and values 17
  2.1 Introduction ............................................ 17
  2.2 Binding basics ......................................... 19
  2.3 Copy-on-modify ......................................... 22
  2.4 Object size ............................................. 29
  2.5 Modify-in-place ......................................... 31
  2.6 Unbinding and the garbage collector .............. 36
  2.7 Quiz answers ............................................ 38

3 Vectors 39
  3.1 Introduction ............................................ 39
  3.2 Atomic vectors ......................................... 41
  3.3 Attributes .............................................. 45
  3.4 S3 atomic vectors ...................................... 50
  3.5 Lists .................................................... 55
  3.6 Data frames and tibbles .............................. 58
## Contents

### 14.3 Controlling access
- 332

### 14.4 Reference semantics
- 336

### 14.5 Why R6?
- 340

### 15 S4
- 341
  - 15.1 Introduction
- 341
  - 15.2 Basics
- 343
  - 15.3 Classes
- 344
  - 15.4 Generics and methods
- 349
  - 15.5 Method dispatch
- 352
  - 15.6 S4 and S3
- 359

### 16 Trade-offs
- 363
  - 16.1 Introduction
- 363
  - 16.2 S4 versus S3
- 364
  - 16.3 R6 versus S3
- 365

### IV Metaprogramming
- 371
  - Introduction
- 373

### 17 Big picture
- 375
  - 17.1 Introduction
- 375
  - 17.2 Code is data
- 376
  - 17.3 Code is a tree
- 377
  - 17.4 Code can generate code
- 378
  - 17.5 Evaluation runs code
- 380
  - 17.6 Customising evaluation with functions
- 381
  - 17.7 Customising evaluation with data
- 382
  - 17.8 Quosures
- 383

### 18 Expressions
- 385
  - 18.1 Introduction
- 385
  - 18.2 Abstract syntax trees
- 386
  - 18.3 Expressions
- 391
  - 18.4 Parsing and grammar
- 399
  - 18.5 Walking AST with recursive functions
- 404
  - 18.6 Specialised data structures
- 411

### 19 Quasiquotation
- 415
  - 19.1 Introduction
- 415
  - 19.2 Motivation
- 416
  - 19.3 Quoting
- 419
  - 19.4 Unquoting
- 424
  - 19.5 Non-quoting
- 433
  - 19.6 ... (dot-dot-dot)
# Contents

19.7 Case studies ........................................ 442  
19.8 History ........................................ 448  

20 Evaluation ........................................ 451  
  20.1 Introduction ................................ 451  
  20.2 Evaluation basics .............................. 452  
  20.3 Quosures ........................................ 457  
  20.4 Data masks .................................... 462  
  20.5 Using tidy evaluation ......................... 468  
  20.6 Base evaluation ............................... 472  

21 Translating R code ................................ 481  
  21.1 Introduction ................................ 481  
  21.2 HTML ........................................ 482  
  21.3 LaTeX ......................................... 492  

V Techniques ........................................ 501  

Introduction ........................................ 503  

22 Debugging .......................................... 505  
  22.1 Introduction ................................ 505  
  22.2 Overall approach .............................. 506  
  22.3 Locating errors ............................... 507  
  22.4 Interactive debugger ......................... 509  
  22.5 Non-interactive debugging ................... 513  
  22.6 Non-error failures ........................... 516  

23 Measuring performance ............................ 519  
  23.1 Introduction ................................ 519  
  23.2 Profiling .................................... 520  
  23.3 Microbenchmarking ............................ 526  

24 Improving performance ............................ 531  
  24.1 Introduction ................................ 531  
  24.2 Code organisation ............................. 532  
  24.3 Checking for existing solutions ............. 534  
  24.4 Doing as little as possible ................. 535  
  24.5 Vectorise .................................... 539  
  24.6 Avoiding copies ............................... 541  
  24.7 Case study: t-test ............................ 542  
  24.8 Other techniques ............................. 544  

25 Rewriting R code in C++ ........................... 547  
  25.1 Introduction ................................ 547  
  25.2 Getting started with C++ ..................... 548
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.3 Other classes</td>
<td>557</td>
</tr>
<tr>
<td>25.4 Missing values</td>
<td>559</td>
</tr>
<tr>
<td>25.5 Standard Template Library</td>
<td>563</td>
</tr>
<tr>
<td>25.6 Case studies</td>
<td>570</td>
</tr>
<tr>
<td>25.7 Using Rcpp in a package</td>
<td>574</td>
</tr>
<tr>
<td>25.8 Learning more</td>
<td>575</td>
</tr>
<tr>
<td>25.9 Acknowledgments</td>
<td>576</td>
</tr>
</tbody>
</table>

Bibliography 577

Index 581
Welcome to the second edition of *Advanced R*. I had three main goals for this edition:

- Improve coverage of important concepts that I fully understood only after the publication of the first edition.
- Reduce coverage of topics time has shown to be less useful, or that I think are really exciting but turn out not to be that practical.
- Generally make the material easier to understand with better text, clearer code, and many more diagrams.

If you’re familiar with the first edition, this preface describes the major changes so that you can focus your reading on the new areas. If you’re reading a printed version of this book you’ll notice one big change very quickly: *Advanced R* is now in colour! This has considerably improved the syntax highlighting of code chunks, and made it much easier to create helpful diagrams. I have taken advantage of this and included over 100 new diagrams throughout the book.

Another big change in this version is the use of new packages, particularly `rlang` ([http://rlang.r-lib.org](http://rlang.r-lib.org)), which provides a clean interface to low-level data structures and operations. The first edition used base R functions almost exclusively, which created some pedagogical challenges because many functions evolved independently over multiple years, making it hard to see the big underlying ideas hidden amongst the incidental variations in function names and arguments. I continue to show base equivalents in sidebars, footnotes, and where needed, in individual sections, but if you want to see the purest base R expression of the ideas in this book, I recommend reading the first edition, which you can find online at [http://adv-r.had.co.nz](http://adv-r.had.co.nz).

The foundations of R have not changed in the five years since the first edition, but my understanding of them certainly has. Thus, the overall structure of “Foundations” has remained roughly the same, but many of the individual chapters have been considerably improved:

- Chapter 2, “Names and values”, is a brand new chapter that helps you understand the difference between objects and names of objects. This helps you more accurately predict when R will make a copy of a data structure, and lays important groundwork to understand functional programming.
Preface

- Chapter 3, “Vectors” (previously called data structures), has been rewritten to focus on vector types like integers, factors, and data frames. It contains more details of important S3 vectors (like dates and date-times), discusses the data frame variation provided by the tibble package [Müller and Wickham, 2018], and generally reflects my improved understanding of vector data types.

- Chapter 4, “Subsetting”, now distinguishes between [ and [[ by their intention: [ extracts many values and [[ extracts a single value (previously they were characterised by whether they “simplified” or “preserved”). Section 4.3 draws the “train” to help you understand how [[ works with lists, and introduces new functions that provide more consistent behaviour for out-of-bounds indices.

- Chapter 5, “Control flow”, is a new chapter: somehow I previously forgot about important tools like if statements and for loops!

- Chapter 6, “Functions”, has an improved ordering, introduces the pipe (%>%), as a third way to compose functions (Section 6.3), and has considerably improved coverage of function forms (Section 6.8).

- Chapter 7, “Environments”, has a reorganised treatment of special environments (Section 7.4), and a much improved discussion of the call stack (Section 7.5).

- Chapter 8, “Conditions”, contains material previously in “Exceptions and debugging”, and much new content on how R’s condition system works. It also shows you how to create your own custom condition classes (Section 8.5).

The chapters following Part I, Foundations, have been re-organised around the three most important programming paradigms in R: functional programming, object-oriented programming, and metaprogramming.

- Functional programming is now more cleanly divided into the three main techniques: “Functionals” (Chapter 9), “Function factories” (Chapter 10), and “Function operators” (Chapter 11). I’ve focussed in on ideas that have practical applications in data science and reduced the amount of pure theory.

These chapters now use functions provided by the purrr package [Henry and Wickham, 2018a], which allow me to focus more on the underlying ideas and less on the incidental details. This led to a considerable simplification of the function operators chapter since a major use was to work around the absence of ellipses (…) in base functionals.

- Object-oriented programming (OOP) now forms a major section of the book with completely new chapters on base types (Chapter 12), S3 (Chapter 13), S4 (Chapter 15), R6 (Chapter 14), and the tradeoffs between the systems (Chapter 16).
These chapters focus on how the different object systems work, not how to use them effectively. This is unfortunate, but necessary, because many of the technical details are not described elsewhere, and effective use of OOP needs a whole book of its own.

- Metaprogramming (previously called “computing on the language”) describes the suite of tools that you can use to generate code with code. Compared to the first edition this material has been substantially expanded and now focusses on “tidy evaluation”, a set of ideas and theory that make metaprogramming safe, well-principled, and accessible to many more R programmers. Chapter 17, “Big picture” coarsely lays out how all the pieces fit together; Chapter 18, “Expressions”, describes the underlying data structures; Chapter 19, “Quasiquotation”, covers quoting and unquoting; Chapter 20, “Evaluation”, explains evaluation of code in special environments; and Chapter 21, “Translations”, pulls all the themes together to show how you might translate from one (programming) language to another.

The final section of the book pulls together the chapters on programming techniques: profiling, measuring and improving performance, and Rcpp. The contents are very similar to the first edition, although the organisation is a little different. I have made light updates throughout these chapters particularly to use newer packages (microbenchmark -> bench, lineprof -> profvis), but the majority of the text is the same.

While the second edition has mostly expanded coverage of existing material, five chapters have been removed:

- The vocabulary chapter has been removed because it was always a bit of an odd duck, and there are more effective ways to present vocabulary lists than in a book chapter.

- The style chapter has been replaced with an online style guide, http://style.tidyverse.org/. The style guide is paired with the new styler package [Müller and Walthert, 2018] which can automatically apply many of the rules.

- The C chapter has been moved to https://github.com/hadley/r-internals, which, over time, will provide a guide to writing C code that works with R’s data structures.

- The memory chapter has been removed. Much of the material has been integrated into Chapter 2 and the remainder felt excessively technical and not that important to understand.

- The chapter on R’s performance as a language was removed. It delivered few actionable insights, and became dated as R changed.
I have now been programming in R for over 15 years, and have been doing it full-time for the last five years. This has given me the luxury of time to examine how the language works. This book is my attempt to pass on what I've learned so that you can understand the intricacies of R as quickly and painlessly as possible. Reading it will help you avoid the mistakes I've made and dead ends I've gone down, and will teach you useful tools, techniques, and idioms that can help you to attack many types of problems. In the process, I hope to show that, despite its sometimes frustrating quirks, R is, at its heart, an elegant and beautiful language, well tailored for data science.

1.1 Why R?

If you are new to R, you might wonder what makes learning such a quirky language worthwhile. To me, some of the best features are:

- It’s free, open source, and available on every major platform. As a result, if you do your analysis in R, anyone can easily replicate it, regardless of where they live or how much money they earn.

- R has a diverse and welcoming community, both online (e.g. the #rstats twitter community (https://twitter.com/search?q=%23rstats)) and in person (like the many R meetups (https://www.meetup.com/topics/r-programming-language/)). Two particularly inspiring community groups are rweekly newsletter (https://rweekly.org) which makes it easy to keep up to date with R, and R-Ladies (http://r-ladies.org) which has made a wonderfully welcoming community for women and other minority genders.

- A massive set of packages for statistical modelling, machine learning, visualisation, and importing and manipulating data. Whatever model or graphic you’re trying to do, chances are that someone has already tried to do it and you can learn from their efforts.

- Powerful tools for communicating your results. RMarkdown (https://rmarkdown.rstudio.com) makes it easy to turn your results into HTML
files, PDFs, Word documents, PowerPoint presentations, dashboards and more. Shiny (http://shiny.rstudio.com) allows you to make beautiful interactive apps without any knowledge of HTML or javascript.

- RStudio, the IDE (http://www.rstudio.com/ide/), provides an integrated development environment, tailored to the needs of data science, interactive data analysis, and statistical programming.

- Cutting edge tools. Researchers in statistics and machine learning will often publish an R package to accompany their articles. This means immediate access to the very latest statistical techniques and implementations.

- Deep-seated language support for data analysis. This includes features like missing values, data frames, and vectorisation.

- A strong foundation of functional programming. The ideas of functional programming are well suited to the challenges of data science, and the R language is functional at heart, and provides many primitives needed for effective functional programming.

- RStudio, the company (https://www.rstudio.com), which makes money by selling professional products to teams of R users, and turns around and invests much of that money back into the open source community (over 50% of software engineers at RStudio work on open source projects). I work for RStudio because I fundamentally believe in its mission.

- Powerful metaprogramming facilities. R’s metaprogramming capabilities allow you to write magically succinct and concise functions and provide an excellent environment for designing domain-specific languages like ggplot2, dplyr, data.table, and more.

- The ease with which R can connect to high-performance programming languages like C, Fortran, and C++.

Of course, R is not perfect. R’s biggest challenge (and opportunity!) is that most R users are not programmers. This means that:

- Much of the R code you’ll see in the wild is written in haste to solve a pressing problem. As a result, code is not very elegant, fast, or easy to understand. Most users do not revise their code to address these shortcomings.

- Compared to other programming languages, the R community is more focussed on results than processes. Knowledge of software engineering best practices is patchy. For example, not enough R programmers use source code control or automated testing.

- Metaprogramming is a double-edged sword. Too many R functions use tricks to reduce the amount of typing at the cost of making code that is hard to understand and that can fail in unexpected ways.
1.3 Who should read this book

- Inconsistency is rife across contributed packages, and even within base R. You are confronted with over 25 years of evolution every time you use R, and this can make learning R tough because there are so many special cases to remember.

- R is not a particularly fast programming language, and poorly written R code can be terribly slow. R is also a profligate user of memory.

Personally, I think these challenges create a great opportunity for experienced programmers to have a profound positive impact on R and the R community. R users do care about writing high quality code, particularly for reproducible research, but they don’t yet have the skills to do so. I hope this book will not only help more R users to become R programmers, but also encourage programmers from other languages to contribute to R.

1.2 Who should read this book

This book is aimed at two complementary audiences:

- Intermediate R programmers who want to dive deeper into R, understand how the language works, and learn new strategies for solving diverse problems.

- Programmers from other languages who are learning R and want to understand why R works the way it does.

To get the most out of this book, you’ll need to have written a decent amount of code in R or another programming language. You should be familiar with the basics of data analysis (i.e. data import, manipulation, and visualisation), have written a number of functions, and be familiar with the installation and use of CRAN packages.

This book walks the narrow line between being a reference book (primarily used for lookup), and being linearly readable. This involves some tradeoffs, because it’s difficult to linearise material while still keeping related materials together, and some concepts are much easier to explain if you’re already familiar with specific technical vocabulary. I’ve tried to use footnotes and cross-references to make sure you can still make sense even if you just dip your toes in a chapter.
1.3 What you will get out of this book

This book delivers the knowledge that I think an advanced R programmer should possess: a deep understanding of the fundamentals coupled with a broad vocabulary that means that you can tactically learn more about a topic when needed.

After reading this book, you will:

- Be familiar with the foundations of R. You will understand complex data types and the best ways to perform operations on them. You will have a deep understanding of how functions work, you’ll know what environments are, and how to make use of the condition system.
- Understand what functional programming means, and why it is a useful tool for data science. You’ll be able to quickly learn how to use existing tools, and have the knowledge to create your own functional tools when needed.
- Know about R’s rich variety of object-oriented systems. You’ll be most familiar with S3, but you’ll know of S4 and R6 and where to look for more information when needed.
- Appreciate the double-edged sword of metaprogramming. You’ll be able to create functions that use tidy evaluation, saving typing and creating elegant code to express important operations. You’ll also understand the dangers and when to avoid it.
- Have a good intuition for which operations in R are slow or use a lot of memory. You’ll know how to use profiling to pinpoint performance bottlenecks, and you’ll know enough C++ to convert slow R functions to fast C++ equivalents.

1.4 What you will not learn

This book is about R the programming language, not R the data analysis tool. If you are looking to improve your data science skills, I instead recommend that you learn about the tidyverse (https://www.tidyverse.org/), a collection of consistent packages developed by me and my colleagues. In this book you’ll learn the techniques used to develop the tidyverse packages; if you want to instead learn how to use them, I recommend R for Data Science (http://r4ds.had.co.nz/).
1.6 Meta-techniques

If you want to share your R code with others, you will need to make an R package. This allows you to bundle code along with documentation and unit tests, and easily distribute it via CRAN. In my opinion, the easiest way to develop packages is with devtools (http://devtools.r-lib.org), roxygen2 (http://klutometis.github.io/roxygen/), testthat (http://testthat.r-lib.org), and usethis (http://usethis.r-lib.org). You can learn about using these packages to make your own package in R packages (http://r-pkgs.had.co.nz/).

1.5 Meta-techniques

There are two meta-techniques that are tremendously helpful for improving your skills as an R programmer: reading source code and adopting a scientific mindset.

Reading source code is important because it will help you write better code. A great place to start developing this skill is to look at the source code of the functions and packages you use most often. You’ll find things that are worth emulating in your own code and you’ll develop a sense of taste for what makes good R code. You will also see things that you don’t like, either because its virtues are not obvious or it offends your sensibilities. Such code is nonetheless valuable, because it helps make concrete your opinions on good and bad code.

A scientific mindset is extremely helpful when learning R. If you don’t understand how something works, you should develop a hypothesis, design some experiments, run them, and record the results. This exercise is extremely useful since if you can’t figure something out and need to get help, you can easily show others what you tried. Also, when you learn the right answer, you’ll be mentally prepared to update your world view.

1.6 Recommended reading

Because the R community mostly consists of data scientists, not computer scientists, there are relatively few books that go deep in the technical underpinnings of R. In my personal journey to understand R, I’ve found it particularly helpful to use resources from other programming languages. R has aspects of both functional and object-oriented (OO) programming languages. Learning how these concepts are expressed in R will help you leverage your existing knowledge of other programming languages, and will help you identify areas where you can improve.
To understand why R’s object systems work the way they do, I found *The Structure and Interpretation of Computer Programs*\(^1\) [Abelson et al., 1996] (SICP) to be particularly helpful. It’s a concise but deep book, and after reading it, I felt for the first time that I could actually design my own object-oriented system. The book was my first introduction to the encapsulated paradigm of object-oriented programming found in R, and it helped me understand the strengths and weaknesses of this system. SICP also teaches the functional mindset where you create functions that are simple individually, and which become powerful when composed together.

To understand the trade-offs that R has made compared to other programming languages, I found *Concepts, Techniques and Models of Computer Programming* [Van-Roy and Haridi, 2004] extremely helpful. It helped me understand that R’s copy-on-modify semantics make it substantially easier to reason about code, and that while its current implementation is not particularly efficient, it is a solvable problem.

If you want to learn to be a better programmer, there’s no place better to turn than *The Pragmatic Programmer* [Hunt and Thomas, 1990]. This book is language agnostic, and provides great advice for how to be a better programmer.

### 1.7 Getting help

Currently, there are three main venues to get help when you’re stuck and can’t figure out what’s causing the problem: RStudio Community (https://community.rstudio.com/), StackOverflow (http://stackoverflow.com) and the R-help mailing list (https://stat.ethz.ch/mailman/listinfo/r-help). You can get fantastic help in each venue, but they do have their own cultures and expectations. It’s usually a good idea to spend a little time lurking, learning about community expectations, before you put up your first post.

Some good general advice:

- Make sure you have the latest version of R and of the package (or packages) you are having problems with. It may be that your problem is the result of a recently fixed bug.

- Spend some time creating a reproduceable example, or reprex. This will help others help you, and often leads to a solution without asking others, because in the course of making the problem reproducible you often figure

\(^1\)You can read it online for free at https://mitpress.mit.edu/sites/default/files/sicp/full-text/book/book.html
out the root cause. I highly recommend learning and using the reprex (https://reprex.tidyverse.org/) package.

If you are looking for specific help solving the exercises in this book, solutions from Malte Grosser and Henning Bumann are available at https://advanced-r-solutions.rbind.io.

1.8 Acknowledgments

I would like to thank the many contributors to R-devel and R-help and, more recently, Stack Overflow and RStudio Community. There are too many to name individually, but I'd particularly like to thank Luke Tierney, John Chambers, JJ Allaire, and Brian Ripley for generously giving their time and correcting my countless misunderstandings.

This book was written in the open (https://github.com/hadley/adv-r/), and chapters were advertised on twitter (https://twitter.com/hadleywickham) when complete. It is truly a community effort: many people read drafts, fixed typos, suggested improvements, and contributed content. Without those contributors, the book wouldn’t be nearly as good as it is, and I’m deeply grateful for their help. Special thanks go to Jeff Hammerbacher, Peter Li, Duncan Murdoch, and Greg Wilson, who all read the book from cover-to-cover and provided many fixes and suggestions.

A big thank you to all 386 contributors (in alphabetical order by username): Aaron Wolen (@aaronwolen), @absolutelyNoWarranty, Adam Hunt (@adamphunt), @agrabovsky, Alexander Grueneberg (@agrueneberg), Anthony Damico (@ajdamico), James Manton (@ajdm), Aaron Schumacher (@ajschumacher), Alan Dipert (@alandipert), Alex Brown (@alexbbrown), @alexperrone, Alex Whitworth (@alexWhitworth), Alexandros Kokkalis (@alko989), @amarchin, Amelia McNamara (@AmeliaMN), Bryce Mecum (@amoeba), Andrew Laucius (@andrewla), Andrew Bray (@andrewpbray), Andrie de Vries (@andrie), Angela Li (@angela-li), @aranluner, Ari Lamstein (@arilandstein), @asnr, Andy Teucher (@ateucher), Albert Vilella (@avilella), Baptiste (@baptiste), Brian G. Barkley (@BarkleyBG), Mara Averick (@batpigandme), Byron (@bcaeger), Brandon Greenwell (@bgreenwell), Brandon Hurr (@bhive01), Jason Knight (@binarybana), Brett Klamer (@bklamer), Jesse Anderson (@blindjesse), Brian Mayer (@blmayer), Benjamin L. Moore (@blmoore), Brian Diggs (@BrianDiggs), Brian S. Yandell (@byandell), @carey1024, Chip Hogg (@chiphogg), Chris Muir (@ChrisMuir), Christopher Gandrud (@christophergandrud), Clay Ford (@clayford), Colin Fay (@ColinFay), @cortinah, Cameron Plouffe (@cplouffe), Carson Sievert (@cpsievert), Craig Citro (@craigcitro), Craig Grabowski (@craiggrabowski),
1.8 Acknowledgments

bins (@jtir13), Juan Manuel Truppi (@juacentro), @juan@gomezduaso, Kevin Markham (@justmarkham), John Verzani (@jverzani), Michael Kane (@kaneplusplus), Bart Kastermans (@kasterma), Kevin D’Auria (@kdau- ria), Karandeep Singh (@kdpsingh), Ken Williams (@kenahoo), Kendon Bell (@kendonB), Kent Johnson (@kent37), Kevin Ushey (@kevinushey), 電线杆 (@kfun123), Karl Forner (@kforner), Kirill Sevastyanenko (@kirillseva), Brian Knaus (@knausb), Kirill Müller (@krlmhr), Kriti Sen Sharma (@ksens), Kai Tang (@kttang), Kevin Wright (@kwstat), suo.lawrence.liu@gmail.com (mailto:suo.lawrence.liu@gmail.com) (@Lawrence-Liu), @ldfnrails, Kevin Kainan Li (@legendre6891), Rachel Severson (@leightseverson), Laurent Gatto (@gatto), C. Jason Liang (@liangcj), Steve Lianoglou (@lianos), Yongfu Liao (@liao96120), Likan (@likanzhan), @lindbrook, Lingbing Feng (@ling- bing), Marcel Ramos (@liink-NY), Zhongpeng Lin (@linzhp), Lionel Henry (@lione-), Llius (@llrs), myq (@lrcg), Luke W Johnston (@lwjohnst86), Kevin Lynagh (@lynagkh), @MajoroMask, Malcolm Barrett (@malcolmbarrett), @mannyishere, @mascaretti, Matt (@mattbaggott), Matthew Gro- gan (@mattgorgan), @matthewhillary, Matthieu Gomez (@matthieugomez), Matt Malin (@mattmalin), Mauro Lepore (@maurolepore), Max Ghenis (@maxGhenis), Maximilian Held (@maxheld83), Michal Bojanowski (@mbo- jan), Mark Rosenstei (@mborosna), Michael Sumner (@mdsumner), Jun Mei (@meijunj), merkliopas (@merkliopas), mfrasco (@mfrasco), Michael Bach (@michaelbach), Michael Bishop (@MichaelMBishop), Michael Buck- ley (@michaelmikebuckley), Michael Quinn (@michaelquinn32), @miguel- morin, Michael (@mikekaminsky), Mine Cetinkaya-Rundel (@mine-cetinkayarundel), @mj$duncan, Mamoun Benghezal (@MoBeng), Matt Pettis (@mpe- ttit), Martin Morgan (@mtnmorgan), Guy Dawson (@Mullefa), Nacho Caballero (@nachocab), Natalya Rapstine (@natalya-patrikeeva), Nick Carchedi (@ncarch- edi), Pascal Burkhard (@Penuil), Noah Greifer (@ngreifer), Nicholas Vasile (@nicky9), Nikos Ignotiadiis (@ignatiadiis), Nina Munikhoit Jakobsen (@nm- jakobsen), Xavier Laviron (@norical), Nick Pullen (@nsthp), Oge Nnadi (@ogennadi), Oliver Paisley (@oliverpaisley), Parikesht Nanda (@onsai), Øystein Sørensen (@osorensen), Paul (@otepoti), Otho Mantegazza (@otho- mantegazza), Dewey Dunnington (@paleolimbott), Paola Corrales (@paocor- rales), Parker Abergrombrie (@parkerabergrombie), Patrick Hausmann (@patperu), Patrick Miller (@pattrickmn), Patrick Werkmeister (@Patrick01), @paulpontin, @pbdb61, Tom Crockett (@pelotom), @pengyu, Jeremiah (@perryjer1), Peter Hickey (@PeteHairch), Phil Chalmers (@philchalmers), Jose Antonio Magaña Mesa (@picaras), Pierre Casadebaig (@picasa), Antonio Picco- lboni (@picolbo), Pierre Roudier (@pierreroudier), Poor Yorick (@pooryorick), Marie-Helene Burle (@prosoitos), Peter Schulam (@pschulam), John (@quantbo), Quyu Kong (@qykon), Ramiro Magno (@ramiromagno), Ram- nath Vaidyanathan (@ramnathv), Kun Ren (@renkun-ken), Richard Reeve (@richardreeve), Richard Cotton (@richierocks), Robert M Flight (@rm- flight), R. Mark Sharp (@rmsharp), Robert Krzyzanowski (@robertzk), robiRagan, Romain François (@romanfrancois), Ross Holmberg (@rossholm-
1.9 Conventions

Throughout this book I use \( f() \) to refer to functions, \( g \) to refer to variables and function parameters, and \( h/ \) to paths.

Larger code blocks intermingle input and output. Output is commented \( \#> \) so that if you have an electronic version of the book, e.g., https://adv-r.hadley.nz/, you can easily copy and paste examples into R.
Many examples use random numbers. These are made reproducible by \texttt{set.seed(1014)}, which is executed automatically at the start of each chapter.
1.10 Colophon


This version of the book was built with R version 3.5.2 (2018-12-20) and the following packages.

<table>
<thead>
<tr>
<th>package</th>
<th>version</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>bench</td>
<td>1.0.1</td>
<td>Github (r-lib/bench@97844d5)</td>
</tr>
<tr>
<td>bookdown</td>
<td>0.9</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>dbplyr</td>
<td>1.3.0.9000</td>
<td>local</td>
</tr>
<tr>
<td>desc</td>
<td>1.2.0</td>
<td>Github (r-lib/desc@42b9578)</td>
</tr>
<tr>
<td>emo</td>
<td>0.0.0.9000</td>
<td>Github (hadley/emo@02a5206)</td>
</tr>
<tr>
<td>ggbeeswarm</td>
<td>0.6.0</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>ggplot2</td>
<td>3.0.0</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>knitr</td>
<td>1.20</td>
<td>standard (@1.20)</td>
</tr>
<tr>
<td>lobstr</td>
<td>1.0.1</td>
<td>CRAN (R 3.5.1)</td>
</tr>
<tr>
<td>memoise</td>
<td>1.1.0.9000</td>
<td>Github (hadley/memoise@1650ad7)</td>
</tr>
<tr>
<td>png</td>
<td>0.1-7</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>profvis</td>
<td>0.3.5</td>
<td>CRAN (R 3.5.1)</td>
</tr>
<tr>
<td>Rcpp</td>
<td>1.0.0.1</td>
<td>Github (RcppCore/Rcpp@0c9f683)</td>
</tr>
<tr>
<td>rlang</td>
<td>0.3.1.9000</td>
<td>Github (r-lib/rlang@7243e6d)</td>
</tr>
<tr>
<td>rmarkdown</td>
<td>1.11</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>RSQLite</td>
<td>2.1.1.9002</td>
<td>Github (r-dbi/RSQLite@0db36af)</td>
</tr>
<tr>
<td>scales</td>
<td>1.0.0</td>
<td>CRAN (R 3.5.0)</td>
</tr>
<tr>
<td>sessioninfo</td>
<td>1.1.1</td>
<td>CRAN (R 3.5.1)</td>
</tr>
<tr>
<td>sloop</td>
<td>1.0.0.9000</td>
<td>local</td>
</tr>
<tr>
<td>testthat</td>
<td>2.0.1.9000</td>
<td>local</td>
</tr>
<tr>
<td>tidyr</td>
<td>0.8.3.9000</td>
<td>local</td>
</tr>
<tr>
<td>vctrs</td>
<td>0.1.0.9002</td>
<td>Github (r-lib/vctrs@098154c)</td>
</tr>
<tr>
<td>zealot</td>
<td>0.1.0</td>
<td>CRAN (R 3.5.0)</td>
</tr>
</tbody>
</table>
Bibliography


