

# 1

---

## *Introduction*

---

With more than 10 years experience programming in R, I've had the luxury of being able to spend a lot of time trying to figure out and understand how the language works. This book is my attempt to pass on what I've learned so that you can quickly become an effective R programmer. 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 frustrating quirks, R is, at its heart, an elegant and beautiful language, well tailored for data analysis and statistics.

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.
- 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. At a minimum, you can learn from their efforts.
- 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 subsetting.
- A fantastic community. It is easy to get help from experts on the R-help mailing list (<https://stat.ethz.ch/mailman/listinfo/r-help>), stackoverflow (<http://stackoverflow.com/questions/tagged/r>), or subject-specific mailing lists like R-SIG-mixed-models (<https://>

[//stat.ethz.ch/mailman/listinfo/r-sig-mixed-models](http://stat.ethz.ch/mailman/listinfo/r-sig-mixed-models)) or `ggplot2` (<https://groups.google.com/forum/#!forum/ggplot2>). You can also connect with other R learners via twitter (<https://twitter.com/search?q=%23rstats>), linkedin (<http://www.linkedin.com/groups/R-Project-Statistical-Computing-77616>), and through many local user groups (<http://blog.revolutionanalytics.com/local-r-groups.html>).

- Powerful tools for communicating your results. R packages make it easy to produce html or pdf reports (<http://yihui.name/knitr/>), or create interactive websites (<http://www.rstudio.com/shiny/>).
- A strong foundation in functional programming. The ideas of functional programming are well suited to solving many of the challenges of data analysis. R provides a powerful and flexible toolkit which allows you to write concise yet descriptive code.
- An IDE (<http://www.rstudio.com/ide/>) tailored to the needs of interactive data analysis and statistical programming.
- Powerful metaprogramming facilities. R is not just a programming language, it is also an environment for interactive data analysis. Its metaprogramming capabilities allow you to write magically succinct and concise functions and provide an excellent environment for designing domain-specific languages.
- Designed to connect to high-performance programming languages like C, Fortran, and C++.

Of course, R is not perfect. R's biggest challenge 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 tends to be more focussed on results instead of processes. Knowledge of software engineering best practices is patchy: for instance, 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.

- Inconsistency is rife across contributed packages, even within base R. You are confronted with over 20 years of evolution every time you use R. Learning R can be tough because there are 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.

---

## 1.1 Who should read this book

This book is aimed at two complementary audiences:

- Intermediate R programmers who want to dive deeper into R 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 might not know all the details, but you should be familiar with how functions work in R and although you may currently struggle to use them effectively, you should be familiar with the `apply` family (like `apply()` and `lapply()`).

---

## 1.2 What you will get out of this book

This book describes the skills I think an advanced R programmer should have: the ability to produce quality code that can be used in a wide variety of circumstances.

After reading this book, you will:

- Be familiar with the fundamentals 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, and be able to recognise and use the four object systems in R.
- Understand what functional programming means, and why it is a useful tool for data analysis. You'll be able to quickly learn how to use existing tools, and have the knowledge to create your own functional tools when needed.
- Appreciate the double-edged sword of metaprogramming. You'll be able to create functions that use non-standard evaluation in a principled way, saving typing and creating elegant code to express important operations. You'll also understand the dangers of metaprogramming and why you should be careful about its use.
- 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.
- Be comfortable reading and understanding the majority of R code. You'll recognise common idioms (even if you wouldn't use them yourself) and be able to critique others' code.

---

### 1.3 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, 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. When I clearly describe a problem to someone else (the art of creating a reproducible example (<http://stackoverflow.com/questions/5963269>)), I often figure out the solution myself.

---

## 1.4 Recommended reading

R is still a relatively young language, and the resources to help you understand it are still maturing. 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* (<http://mitpress.mit.edu/sicp/full-text/book/book.html>) (SICP) by Harold Abelson and Gerald Jay Sussman, particularly helpful. It's a concise but deep book. 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 generic function style of OO common in R. It helped me understand its strengths and weaknesses. SICP also talks a lot about functional programming, and how to create simple functions which become powerful when combined.

To understand the trade-offs that R has made compared to other programming languages, I found *Concepts, Techniques and Models of Computer Programming* (<http://amzn.com/0262220695?tag=devtools-20>) by Peter van Roy and Sef Haridi 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* (<http://amzn.com/020161622X?tag=devtools-20>) by Andrew Hunt and David Thomas. This book is language agnostic, and provides great advice for how to be a better programmer.

---

## 1.5 Getting help

Currently, there are two main venues to get help when you're stuck and can't figure out what's causing the problem: stackoverflow (<http://stackoverflow.com>) and the R-help mailing list. You can get fantastic help in both venues, 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 reproducible example (<http://stackoverflow.com/questions/5963269>). This is often a useful process in its own right, because in the course of making the problem reproducible you often figure out what's causing the problem.
- Look for related problems before posting. If someone has already asked your question and it has been answered, it's much faster for everyone if you use the existing answer.

---

## 1.6 Acknowledgments

I would like to thank the tireless contributors to R-help and, more recently, stackoverflow (<http://stackoverflow.com/questions/tagged/r>). There are too many to name individually, but I'd particularly like to thank Luke Tierney, John Chambers, Dirk Eddelbuettel, 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 Peter Li, who read the book from cover-to-cover and provided many fixes. Other outstanding contributors were Aaron Schumacher, @crtahlin, Lingbing Feng, @juancentro, and @johnbaums.

Thanks go to all contributors in alphabetical order: Aaron Schumacher, Aaron Wolen, @aaronwolen, @absolutelyNoWarranty, Adam Hunt, @agrabovsky, @ajdm, @alexbbrown, @alko989, @allegretto, @AmeliaMN, @andrewla, Andy Teucher, Anthony Damico, Anton Antonov, @aranlunzer, @arilamstein, @avilella, @baptiste, @blindjesse, @blmoore, @bnjmn, Brandon Hurr, @BrianDiggs, @Bryce, C. Jason Liang, @Carson, @cdrv, Ching Boon, @chiphogg, Christopher Brown, @christophergandrud, Clay Ford, @cornelius1729, @cplouffe, Craig Citro, @crossfitAL, @crowding, Crt Ahlin, @crtahlin, @cscheid, @cs-gillespie, @cusanovich, @cwarden, @cwickham, Daniel Lee, @darrkj, @Dasonk, David Hajage, David LeBauer, @dchudz, dennis feehan, @dfeehan, Dirk Eddelbuettel, @dkahle, @dlebauer, @dlschweizer, @dmontaner, @dougmitarotonda, @dpatschke, @duncandonutz, @EdFineOKL, @EDiLD, @eipi10, @elegrand, @EmilRehnberg, Eric C. Anderson, @etb, @fabian-s, Facundo Mu  soz, @flammy0530, @fpepin, Frank Farach, @freezby, @fyears, Garrett Grolemond, @garrettgman, @gavinsimpson, @gggtest, G     gen Eraslan, Gregg Whitworth, @gregorp, @gsee, @gsk3, @gthb, @hassaad85, @i, Iain Dillingham, @ijlyttle, Ilan Man, @immanuelcostigan, @initdch, Jason Asher, Jason Knight, @jasondavies, @jastingo, @jcborras, Jeff Allen, @jeharmse, @jentjr, @JestonBlu, @JimInNashville, @jinlong25, JJ Allaire, Jochen Van de Velde, Johann Hirschman, John Blischak, John Verzani, @johnbaums, @johnjosephhorton, Joris Muller, Joseph Casillas, @juancentro, @kdauria, @kenahoo, @kent37, Kevin Markham, Kevin Ushey, @kforner, Kirill M  ijler, Kun Ren, Laurent Gatto, @Lawrence-Liu, @ldfmrails, @lgatto, @liangcj, Lingbing Feng, @lynaghk, Maarten Kruijver, Mamoun Benghezal, @mannyishere, Matt Pettis, @mattbaggott, Matthew Grogan, @mattmalin, Michael Kane, @michaelbach, @mjdsduncan, @Mullefa, @myqlarson, Nacho Caballero, Nick Carchedi, @nstjhp, @ogennadi, Oliver Keyes, @otepoti, Parker Abercrombie, @patperu,

Patrick Miller, @pdb61, @pengyu, Peter F Schulam, Peter Lindbrook, Peter Meilstrup, @philchalmers, @picasa, @piccolbo, @pierrerroudier, @pooryorick, R. Mark Sharp, Ramnath Vaidyanathan, @ramnathv, @Rappster, Ricardo Pietrobon, Richard Cotton, @richardreeve, @rmflight, @rmsharp, Robert M Flight, @RobertZK, @robiRagan, Romain François, @rrunner, @rubenfcasal, @sailingwave, @sarunasmerkliopas, @sbgraves237, Scott Ritchie, @scottko, @scottl, Sean Anderson, Sean Carmody, Sean Wilkinson, @sebastian-c, Sebastien Vigneau, @shabbychef, Shannon Rush, Simon O'Hanlon, Simon Potter, @SplashDance, @ste-fan, Stefan Widgren, @stephens999, Steven Pav, @strongh, @stuttgartur, @surmann, @swnydick, @taekyunk, Tal Galili, @talgalili, @tdenes, @Thomas, @thomasherbig, @thomaszumbrunn, Tim Cole, @tjmahr, Tom Buckley, Tom Crockett, @ttriche, @twjacobs, @tyhenkaline, @tylerritchie, @ulrichatz, @varun729, @victorkryukov, @vijaybarve, @vzemlys, @wchi144, @wibeasley, @WilCrofter, William Doane, Winston Chang, @wmc3, @wordnerd, Yoni Ben-Meshulam, @zackham, @zerokarmaleft, Zhongpeng Lin.

---

## 1.7 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., <http://adv-r.had.co.nz>, you can easily copy and paste examples into R. Output comments look like `#>` to distinguish them from regular comments.

---

## 1.8 Colophon

This book was written in Rmarkdown (<http://rmarkdown.rstudio.com/>) inside Rstudio (<http://www.rstudio.com/ide/>). knitr (<http://yihui.name/knitr/>) and pandoc (<http://johnmacfarlane.net/pandoc/>) converted the raw Rmarkdown to html and pdf. The website (<http://adv-r.had.co.nz>) was made with jekyll (<http://jekyllrb.com/>), styled with bootstrap (<http://getbootstrap.com/>), and automatically



published to Amazon's S3 (<http://aws.amazon.com/s3/>) by travis-ci (<https://travis-ci.org/>). The complete source is available from github (<https://github.com/hadley/adv-r>).

Code is set in inconsolata (<http://levien.com/type/myfonts/inconsolata.html>).