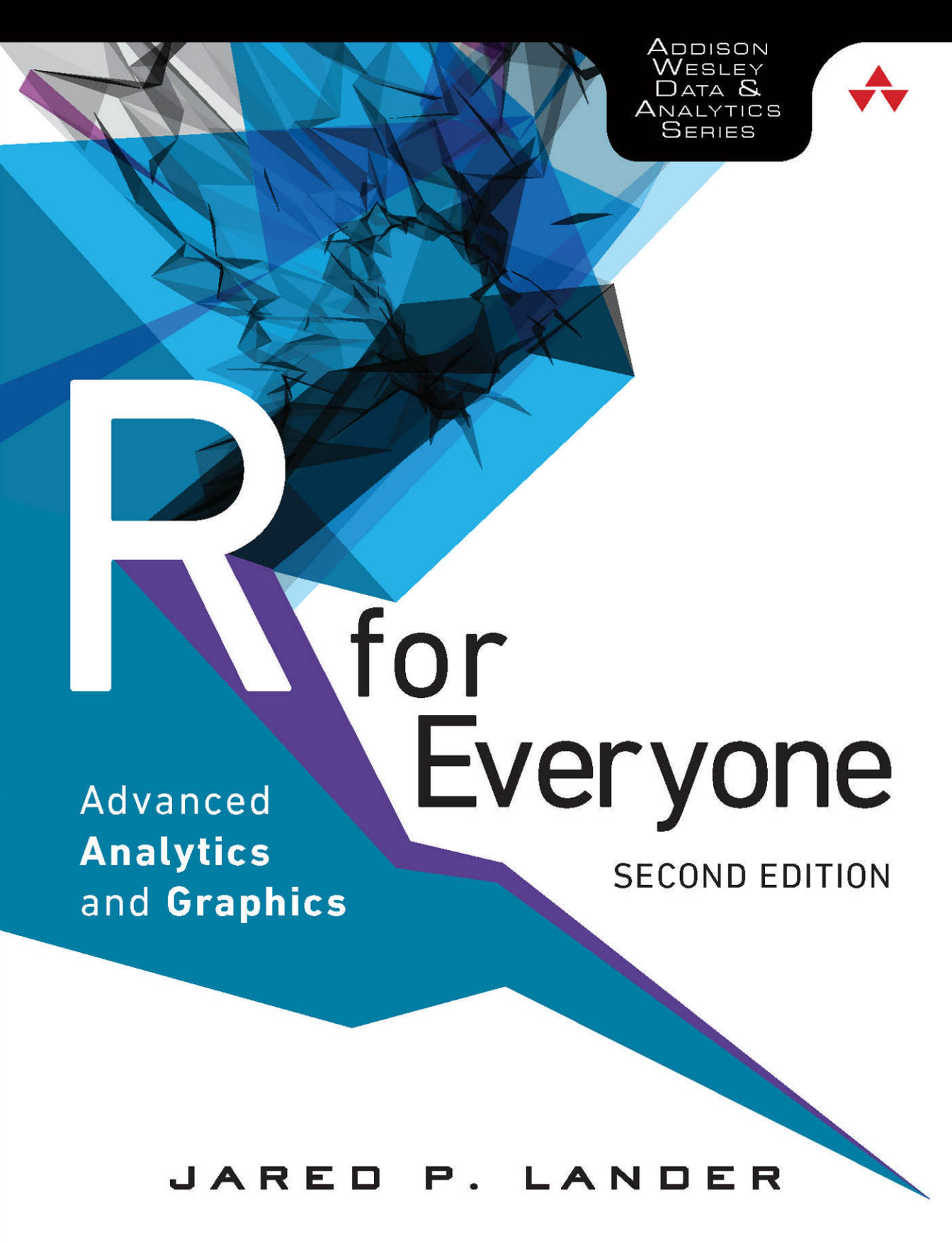
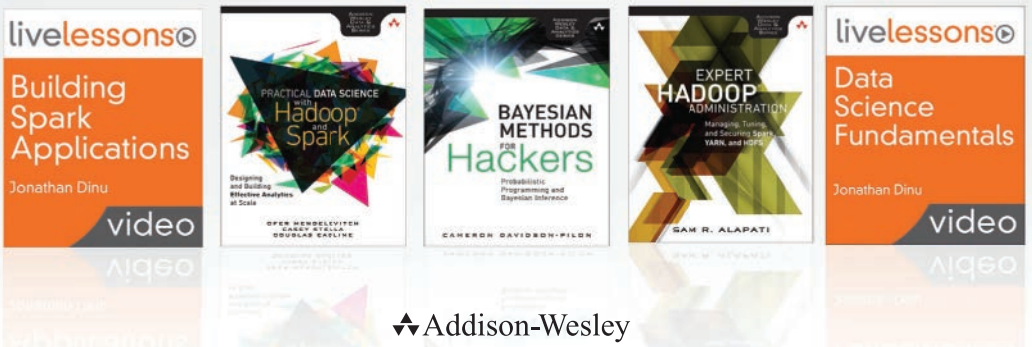

Addison Wesley Data & Analytics Series
R for Everyone
Advanced Analytics and Graphics
SECOND EDITION
JARED P. LANDER

R for Everyone

Second Edition

The Addison-Wesley Data and Analytics Series



Visit **informit.com/awdataseries** for a complete list of available publications.

The **Addison-Wesley Data and Analytics Series** provides readers with practical knowledge for solving problems and answering questions with data. Titles in this series primarily focus on three areas:

1. **Infrastructure:** how to store, move, and manage data
2. **Algorithms:** how to mine intelligence or make predictions based on data
3. **Visualizations:** how to represent data and insights in a meaningful and compelling way

The series aims to tie all three of these areas together to help the reader build end-to-end systems for fighting spam; making recommendations; building personalization; detecting trends, patterns, or problems; and gaining insight from the data exhaust of systems and user interactions.

Make sure to connect with us!
informit.com/socialconnect

Pearson

informIT.com
the trusted technology learning source

Addison-Wesley

O'Reilly Safari

R for Everyone

Advanced Analytics and Graphics

Second Edition

Jared P. Lander

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2017934582

ISBN-13: 978-0-13-454692-6
ISBN-10: 0-13-454692-X

1 17

❖

To Becky

❖

This page intentionally left blank

Contents

This page intentionally left blank

Foreword

R has had tremendous growth in popularity over the last five years. Based on that, you'd think that it was a new, up-and-coming language. But surprisingly, R has been around since 1993. Why the sudden uptick in popularity? The somewhat obvious answer seems to be the emergence of data science as a career and field of study. But the underpinnings of data science have been around for many decades. Statistics, linear algebra, operations research, artificial intelligence and machine learning all contribute parts to the tools that a modern data scientist uses. R, more than most languages, has been built to make most of these tools only a single function call away.

That's why I'm excited that Jared has chosen to revisit his bestselling first edition and provide us with this updated second edition that brings in many of the recent innovations in the R community. R is indispensable for many data science tasks. Many algorithms useful for prediction and analysis can be accessed through only a few lines of code, which makes it a great fit for solving modern data challenges. Data science as a field isn't just about math and statistics, and it isn't just about programming and infrastructure. This book provides a well-balanced introduction to the power and expressiveness of R that is aimed at a general audience.

I can't think of a better author to provide an introduction to R than Jared Lander. Jared and I first met through the NYC machine learning community in late 2009. Back then, the NYC data community was small enough to fit in a single conference room, and many of the other data meetups had yet to be formed. Over the last seven years Jared has been at the forefront of the emerging data science profession.

Through running the Open Statistical Programming Meetup, speaking at events, and teaching a course on R at Columbia University, Jared has helped grow the community by educating programmers, data scientists, journalists and statisticians alike. Jared's expertise isn't limited to teaching. As an everyday practitioner he puts these tools to use while consulting for clients big and small. In the time since the first edition of this book was published Jared has continued to do great work in the R community: from organizing the New York R Conference, to speaking at many meetups and conferences, to evaluating the 2016 NFL Draft with R.

This book provides both an introduction to programming in R and the various statistical methods and tools an everyday R programmer uses. This second edition adds new material, making it current with the latest in the R community. This includes sections on data munging with libraries from the Tidyverse, as well as new chapters on RMarkdown, Shiny and others. Examples use publicly available datasets that Jared has helpfully cleaned and made accessible through his Web site. By using real data and setting up interesting problems, this book stays engaging to the end.

—Paul Dix
Series Editor

This page intentionally left blank

Preface

With the increasing prevalence of data in our daily lives, new and better tools are needed to analyze the deluge. Traditionally there have been two ends of the spectrum: lightweight, individual analysis using tools like Excel or SPSS, and heavy duty, high-performance analysis built with C++ and the like. With the increasing strength of personal computers grew a middle ground that was both interactive and robust. Analysis done by an individual on his or her own computer in an exploratory fashion could quickly be transformed into something destined for a server, underpinning advanced business processes. This area is the domain of R, Python and other scripted languages.

R, invented by Robert Gentleman and Ross Ihaka of the University of Auckland in 1993, grew out of S, which was invented by John Chambers at Bell Labs. It is a high-level language that was originally intended to be run interactively, where the user runs a command, gets a result and then runs another command. It has since evolved into a language that can also be embedded in systems and tackle complex problems.

In addition to transforming and analyzing data, R can produce amazing graphics and reports with ease. It is now being used as a full stack for data analysis, extracting and transforming data, fitting models, drawing inferences and making predictions, plotting and reporting results.

R's popularity has skyrocketed since the late 2000s as it has stepped out of academia and into banking, marketing, pharmaceuticals, politics, genomics and many other fields. Its new users are often shifting from low-level, compiled languages like C++, other statistical packages such as SAS or SPSS and from the 800-pound gorilla, Excel. This time period also saw a rapid surge in the number of add-on packages, libraries of prewritten code that extend R's functionality.

While R can sometimes be intimidating to beginners, especially for those without programming experience, I find that programming analysis, instead of pointing and clicking, soon becomes much easier, more convenient and more reliable. It is my goal to make that learning process easier and quicker.

This book lays out information in a way I wish I were taught when learning R in graduate school. Coming full circle, the content of this book was developed in conjunction with the data science course I teach at Columbia University. It is not meant to cover every minute detail of R but rather the 20% of functionality needed to accomplish 80% of the work. The content is organized into self-contained chapters as follows.

The second edition has been updated to cover many tools that have been developed or improved since the publication of the first edition. Primary among the new additions are **dplyr**, **tidyr** and **purrr** from the Tidyverse for munging data. Model fitting gained more attention with discussion of boosted trees and **caret** for parameter tuning. The **knitr** chapter was split in two, with one covering **knitr** and LaTeX and the other devoted to RMarkdown, which has been significantly improved in the past few years, including the

creation of **htmlwidgets** that allow for the inclusion of JavaScript into documents. An entire chapter is dedicated to Shiny, a new tool for creating interactive Web-based dashboards in R. The chapter on writing R packages has been updated to include code testing, and the chapter on reading data has been updated to cover new ways of reading data, including using **readr**, **readxl** and **jsonlite**. The new content reflects many of the new practices in the R community.

Chapter 1, "Getting R," covers where to download R and how to install it. This deals with the various operating systems and 32-bit versus 64-bit versions. It also gives advice on where to install R.

Chapter 2, "The R Environment," provides an overview of using R, particularly from within RStudio. RStudio projects and Git integration are covered, as is customizing and navigating RStudio.

Chapter 3, "Packages," is concerned with how to locate, install and load R packages.

Chapter 4, "Basics of R," is about using R for math. Variable types such as `numeric`, `character` and `Date` are detailed as are `vectors`. There is a brief introduction to calling functions and finding documentation on functions.

Chapter 5, "Advanced Data Structures," is about the most powerful and commonly used data structure, `data.frames`, along with `matrices` and `lists`, are introduced.

Chapter 6, "Reading Data into R," is about getting data into R. Before data can be analyzed, it must be read into R. There are numerous ways to ingest data, including reading from CSVs and databases.

Chapter 7, "Statistical Graphics," makes it clear why graphics are a crucial part of preliminary data analysis and communicating results. R can make beautiful plots using its powerful plotting utilities. Base graphics and **ggplot2** are introduced and detailed here.

Chapter 8, "Writing R Functions," shows that repeatable analysis is often made easier with user defined functions. The structure, arguments and return rules are discussed.

Chapter 9, "Control Statements," covers controlling the flow of programs using **if**, **ifelse** and complex checks.

Chapter 10, "Loops, the Un-R Way to Iterate," introduces iterating using **for** and **while** loops. While these are generally discouraged, they are important to know.

Chapter 11, "Group Manipulations," provides a better alternative to loops—vectorization. Vectorization does not quite iterate through data so much as operate on all elements at once. This is more efficient and is primarily performed with the **apply** family of functions and **plyr** package.

Chapter 12, "Faster Group Manipulation with `dplyr`," covers the next evolution in group manipulation, **dplyr**. This new package has been optimized to work with `data.frames` and takes advantage of pipes for efficient coding that is easier to read.

Chapter 13, "Iterating with `purrr`," provides another alternative to loops with `purrr`, for iterating over `lists` and `vectors`. This represents a return to the functional roots of R.

Chapter 14, "Data Reshaping," is about the fact that combining multiple datasets, whether by stacking or joining, is commonly necessary as is changing the shape of data. The **plyr** and **reshape2** packages offer good functions for accomplishing this in addition to base tools such as **rbind**, **cbind** and **merge**.

Chapter 15, "Reshaping Data in the Tidyverse," showcases another example of package evolution as **dplyr** and **tidyr** replace **plyr** and **reshape2** for combining, reshaping and joining data.

Chapter 16, "Manipulating Strings," is about text. Most people do not associate character data with statistics, but it is an important form of data. R provides numerous facilities for working with strings, including combining them and extracting information from within. Regular expressions are also detailed.

Chapter 17, "Probability Distributions," provides a thorough look at the normal, binomial and Poisson distributions. The formulas and functions for many distributions are noted.

Chapter 18, "Basic Statistics," covers the first statistics most people are taught, such as mean, standard deviation and t-tests.

Chapter 19, "Linear Models," extensively details the most powerful and common tool in statistics—linear models.

Chapter 20, "Generalized Linear Models," shows how linear models are extended to include logistic and Poisson regression. Survival analysis is also covered.

Chapter 21, "Model Diagnostics," establishes the methods for determining the quality of models and variable selection using residuals, AIC, cross-validation, the bootstrap and stepwise variable selection.

Chapter 22, "Regularization and Shrinkage," covers prevention of overfitting using the Elastic Net and Bayesian methods.

Chapter 23, "Nonlinear Models," covers those cases where linear models are inappropriate and nonlinear models are a good solution. Nonlinear least squares, splines, generalized additive models, decision trees, boosted trees and random forests are discussed.

Chapter 24, "Time Series and Autocorrelation," covers methods for the analysis of univariate and multivariate time series data.

Chapter 25, "Clustering," shows how clustering, the grouping of data, is accomplished by various methods such as K-means and hierarchical clustering.

Chapter 26, "Model Fitting with Caret," introduces the **caret** package for automatic model tuning. The package also provides a uniform interface for hundreds of models, easing the analysis process.

Chapter 27, "Reproducibility and Reports with `knitr`," gets into integrating R code and results into reports from within R. This is made easy with **knitr** and LaTeX.

Chapter 28, "Rich Documents with RMarkdown," showcases how to generate reproducible reports, slide shows and Web pages from within R with RMarkdown. Interactivity is accomplished using **htmlwidgets** such as **leaflet** and **dygraphs**.

Chapter 29, "Interactive Dashboards with Shiny," introduces interactive dashboards using Shiny which can generate Web-based dashboards with the full power of R as a backend.

Chapter 30, "Building R Packages," is about how R packages are great for portable, reusable code. Building these packages has been made incredibly easy with the advent of **devtools** and **Rcpp**.

Appendix A, "Real-Life Resources," is a listing of our favorite resources for learning more about R and interacting with the community.

Appendix B, "Glossary," is a glossary of terms used throughout this book.

A good deal of the text in this book is either R code or the results of running code. Code and results are most often in a separate block of text and set in a distinctive font, as shown in the following example. The different parts of code also have different colors. Lines of code start with >, and if code is continued from one line to another, the continued line begins with +.

```
> # this is a comment
>
> # now basic math
> 10 * 10

[1] 100

> # calling a function
> sqrt(4)

[1] 2
```

Certain Kindle devices do not display color, so the digital edition of this book will be viewed in grayscale on those devices.

There are occasions where code is shown inline and looks like `sqrt(4)`.

In the few places where math is necessary, the equations are indented from the margin and are numbered.

$$e^{i\pi} + 1 = 0 \tag{1}$$

Within equations, normal variables appear as italic text (x), vectors are bold lowercase letters ($\mathbf{x}$) and matrices are bold uppercase letters ($\mathbf{X}$). Greek letters, such as α and β, follow the same convention.

Function names are written as **join** and package names as **plyr**. Objects generated in code that are referenced in text are written as `object1`.

Learning R is a gratifying experience that makes life so much easier for so many tasks. I hope you enjoy learning with me.

Acknowledgments

Acknowledgments for the Second Edition

First and foremost, I am most appreciative of my wife-to-be, Rebecca Martin. Writing this second edition meant playing in R for hours at a time, which is fun on its own, but was greatly enhanced by her presence. She is amazing in so many ways, not least of which is that she uses R. She even indulged my delusions of writing like Orwell and Kipling while cruising up the Irrawaddy on the road to Mandalay.

As before, my family has always supported me in so many ways. My parents, Gail and Howard Lander, encouraged me on this path to math and data science. When this book was first published they said it would be too boring for them to enjoy and have since kept their promise of never reading it. It sits similarly unread, yet proudly displayed, in the homes of my grandmother and all my aunts and uncles. My sister and brother-in-law, Aimee and Eric Schechterman, always humor my antics with their kids, Noah and Lila, whom I am beginning to teach to program.

There are many people in the open-source community, particularly those who attend and contribute to the New York Open Statistical Computing Meetup, whose work and encouragement have been great motivators. Principal among them is Drew Conway, the early leader of the meetup who provided a place for my love of R to grow and eventually turned the meetup over to my stewardship. The friendship of Paul Puglia, Saar Golde, Jay Emerson, Adam Hogan, John Mount, Nina Zumel, Kirk Mettler, Max Kuhn, Bryan Lewis, David Smith, Dirk Eddelbuettel, JD Long, Ramnath Vaidyanathan, Hilary Parker and David Robinson has made the experience incredibly entertaining. I even enjoy my time with Python users Wes McKinney, Ben Lerner and James Powell.

The Work-Bench family, my coorganizers for the New York R Conference, are fantastic people. Jon Lehr, Jess Lin, Stephanie Manning, Kelley Mak, Vipin Chamakkala, Laurel Woerner, Michael Yamnitsky and Mickey Graham (despite his obsession with the Oxford comma) are great to be around.

As my business has grown in recent years, many people have helped, either as employees and clients or by providing valuable advice. Among these are Joseph Sherman, Jeff Horner, Lee Medoff, Jeroen Janssens, Jonathan Hersh, Matt Sheridan, Omar De La Cruz Cabrera, Benjamin De Groot, Vinny Saulys, Rick Spielman, Scott Kuhn, Mike Band, Nate Shea-Han, Greg Fuller, Mark Barry and Lenn Robbins. The teachings of Andrew Gelman, David Madigan and Richard Garfield have stayed with me far beyond the university.

This book is largely possible due to the tireless efforts of the RStudio team. The efforts of JJ Allaire, Winston Chang, Joe Cheng, Garrett Grolemund, Hadley Wickham and Yihui Xie provide the tools that make this book, and much of what we do in R technically feasible. Tareef Kawaf, Pete Knast, Sean Lopp, Roger Oberg, Joe Rickert,

Nathan Stephens, Jim Clemens, Anne Carome, Bill Carney and many others support and spur the growth of the R community.

The material for this book was largely drawn from the class I taught at Columbia University with Rachel Schutt, Introduction to Data Science. The students in that class largely shaped the material and tone of the book, including how it was presented. Vivian Peng, Dan Chen, Michael Piccirilli, Adam Obeng, Eurry Kim and Kaz Sakamoto all inspired my writing.

Numerous people helped with the writing, validating, proofreading and editing of this book. Michael Beigelmacher ensured the code works while Chris Zahn did the same with the prose. Paul Dix introduced me to Pearson, enabling the whole process. My editor, Debra Williams Cauley, has now gone through two editions and three videos of my work patterns and is the driving force that has made the book succeed. Without her, this would not exist.

This second edition is built upon all the help of those mentioned in the acknowledgments from my original book, who are still very dear to me.

Acknowledgments for the First Edition

To start, I must thank my mother, Gail Lander, for encouraging me to become a math major. Without that I would never have followed the path that led me to statistics and data science. In a similar vein I have to thank my father, Howard Lander, for paying all those tuition bills. He has been a valuable source of advice and guidance throughout my life and someone I have aspired to emulate in many ways. While they both insist they do not understand what I do, they love that I do it and have helped me all along the way. Staying with family, I should thank my sister and brother-in-law, Aimee and Eric Schechterman, for letting me teach math to Noah, their five-year-old son.

There are many teachers that have helped shape me over the years. The first is Rochelle Lecke who tutored me in middle school math even when my teacher told me I did not have worthwhile math skills.

Then there is Beth Edmondson, my precalc teacher at Princeton Day School. After wasting the first half of high school as a mediocre student she told me I had "some nerve signing up for next year's AP Calc given my grades." She agreed to let me in AP Calc if I went from a C to an A+ in her class, never thinking I stood a chance. Three months later she stood in stood in disbelief as I not only got the A+ but turned around my entire academic career and became an excellent student. She changed my life. Without her I do not know where I would be. I am forever grateful that she was my teacher.

For the first two years at Muhlenberg College I was determined to be a Business and Communications major yet took math classes because they just came naturally to me. Penny Dunham, Bill Dunham and Linda McGuire all convinced me to become a math major, a decision that has certainly improved my life. Greg Cicconetti gave me my first glimpse of rigorous statistics, my first research opportunity and planted the idea in my head that I should go to grad school for statistics. Fortunately, I eventually listened to him.

My time at Columbia University was spent surrounded by brilliant minds in statistics and programming. David Madigan opened my eyes to modern machine learning and Bodhi Sen got thinking about statistical programming. I had the privilege to do research with Andrew Gelman whose insights have been immeasurably important to me. Richard

Garfield showed me how to use statistics to help people in disaster and war zones. His advice and friendship over the years have been dear to me. Jingchen Liu allowed and encouraged me to write my thesis on New York City pizza,[1] which has brought me an inordinate amount of attention.

While at Columbia University I met my good friend—and one time TA—Ivor Cribben who filled in so many gaps in my knowledge. Through him I met Rachel Schutt, who was a source of great advice in grad school and who I am now honored to teach with at Columbia University.

Grad school might never have happened without the encouragement and support of Shanna Lee. She took good care of me and helped maintain my sanity while I was incredibly overcommited to two jobs, classes and Columbia University's hockey team. I am not sure I would have made it through without her.

Steve Czetty gave me my first job in analytics at Sky IT Group and taught me about databases while letting me run wild, programming anything I wanted. This sparked my interest in statistics and data. Joe DeSiena, Philip DuPlessis and Ed Bobrin at the Bardess Group are some of the finest people I have ever had the pleasure to work with and the work they gave me helped put me through grad school. I am proud to be able to do statistics work with them to this day. Mike Minelli, Rich Kittler, Mark Barry, David Smith, Joseph Rickert, Norman Nie, James Peruvankal, Neera Talbert and Dave Rich at Revolution Analytics let me do one of the best jobs I could possibly imagine: Explaining to people in industry why they should be using R. Kirk Mettler, Richard Schultz, Bryan Lewis and Jim Winfield at Big Computing encourage me to have fun, tackling interesting problems in R. Vinny Saulys and Saar Golde were a lot of fun to work with at Goldman Sachs and also very educational.

Throughout the course of writing this book so many people helped, or rather put up with, me. First and foremost is Yin Cheung who saw all the stress I constantly felt. There were many nights and days ruined when I had to work or write and she suffered through those.

My editor, Debra Williams, knew just how to handle me when I was churning out pages, and more frequently, when I was letting time slip by. Her guiding hand has been invaluable. Paul Dix, the series editor and friend of mine, is the person who suggested I write this book, so without him none of this would have happened. Thanks to Caroline Senay and Andrea Fox I realized quite how many mistakes I made as a writer. Without them, this book would not be nearly as well put together. Robert Mauriello's technical review was incredibly useful in honing the presentation of the included material. The folks at RStudio, particularly JJ Allaire and Josh Paulson, make an amazing product, which made the writing process far easier than it would have been otherwise. Yihui Xie, the author of the `knitr` package, put up with a long series of personal feature requests that I desperately needed to write this book. His software, and his speed at implementing my requests, allowed me to make this book look and feel just the way I felt was right.

Numerous people have looked over parts of this book and given me valuable feedback, including some of those already mentioned. Others who have greatly helped me are

1. http://slice.seriouseats.com/archives/2010/03/the-moneyball-of-pizza-statistician-uses-statistics-to-find-nyc-best-pizza.html

Chris Bethel, Dirk Eddelbuettel, Ramnath Vaidyanathan, Eran Bellin, Avi Fisher, Brian Ezra, Paul Puglia, Nicholas Galasinao, Aaron Schumaker, Adam Hogan, Jeffrey Arnold and John Houston.

Last fall was my first time teaching and I am thankful to the students from the Fall 2012 Introduction to Data Science class at Columbia University for being the guinea pigs for the material that ultimately ended up in this book.

There are many people who have helped me along the way and I am grateful to them all.

About the Author

Jared P. Lander is the Chief Data Scientist of Lander Analytics, a New York-based data science firm that specializes in statistical consulting and training services, the Organizer of the New York Open Statistical Programming Meetup—the world's largest R meetup—and the New York R Conference and an Adjunct Professor of Statistics at Columbia University. He is also a tour guide for Scott's Pizza Tours. With a masters from Columbia University in statistics and a bachelors from Muhlenberg College in mathematics, he has experience in both academic research and industry. Very active in the data community, Jared is a frequent speaker at conferences such as Strata and the MIT Sloan Sports Analytics Conference, universities and meetups around the world. His writings on statistics can be found at jaredlander.com, and his work has been featured in many outlets, in particular CBS and the *Wall Street Journal*.

This page intentionally left blank

1

Getting R

R is a wonderful tool for statistical analysis, visualization and reporting. Its usefulness is best seen in the wide variety of fields where it is used. We alone have used R for projects with banks, political campaigns, tech startups, food startups, international development and aid organizations, hospitals and real estate developers. Other areas where we have seen it used are online advertising, insurance, ecology, genetics and pharmaceuticals. R is used by statisticians with advanced machine learning training and by programmers familiar with other languages and also by people who are not necessarily trained in advanced data analysis but are tired of using Excel.

Before it can be used it needs to be downloaded and installed, a process that is no more complicated than installing any other program.

1.1 Downloading R

The first step in using R is getting it on the computer. Unlike with languages such as C++, R must be installed in order to run.[1] The program is easily obtainable from the Comprehensive R Archive Network (CRAN), the maintainer of R, at `http://cran.r-project.org/`. At the top of the page are links to download R for Windows, Mac OS X and Linux.

There are prebuilt installations available for Windows and Mac OS X, while those for Linux usually compile from source. Installing R on any of these platforms is just like installing any other program.

Windows users should click the link "Download R for Windows," then "base" and then "Download R 3.x.x for Windows"; the x's indicate the version of R. This changes periodically as improvements are made.

Similarly, Mac users should click "Download R for (Mac) OS X" and then "R-3.x.x.pkg"; again, the x's indicate the current version of R. This will also install both 32- and 64-bit versions.

Linux users should download R using their standard distribution mechanism whether that is apt-get (Ubuntu and Debian), yum (Red Hat), zypper (SUSE) or another source. This will also build and install R.

1. Technically C++ cannot be set up on its own without a compiler, so something would still need to be installed anyway.

1.2 R Version

As of this writing, R is at version 3.4.0 and has seen a lot of improvements since the first edition of this book when the version was 3.0.1. CRAN follows a one-year release cycle where each major version change increases the middle of the three numbers in the version. For instance, version 3.2.0 was released in 2015. In 2016 the version was incremented to 3.3.0 with 3.4.0 released in 2017. The last number in the version is for minor updates to the current major version.

Most R functionality is usually backward compatible with previous versions.

1.3 32-bit vs. 64-bit

The choice between using 32-bit and using 64-bit comes down to whether the computer supports 64-bit—most new machines do—and the size of the data to be worked with. The 64-bit versions can address arbitrarily large amounts of memory (or RAM), so it might as well be used.

This is especially important starting with version 3.0.0, as that adds support for 64-bit integers, meaning far greater amounts of data can be stored in R objects.

In the past, certain packages required the 32-bit version of R, but that is exceedingly rare these days. The only reason for installing the 32-bit version now is to support some legacy analysis or for use on a machine with a 32-bit processor such as Intel's low-power Atom chip.

1.4 Installing

Installing R on Windows and Mac is just like installing any other program.

1.4.1 Installing on Windows

Find the appropriate installer where it was downloaded. For Windows users, it will look like Figure 1.1.

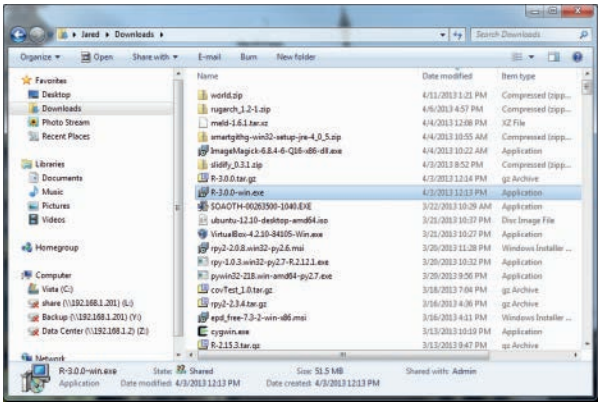


Figure 1.1 Location of R installer.

R should be installed using administrator privileges. This means right-clicking the installer and then selecting Run as Administrator. This brings up a prompt where the administrator password should be entered.

The first dialog, shown in Figure 1.2, offers a choice of language, defaulted at English. Choose the appropriate language and click OK.

Figure 1.2 Language selection for Windows.

Next, the caution shown in Figure 1.3 recommends that all other programs be closed. This advice is rarely followed or necessary anymore, so clicking Next is appropriate.

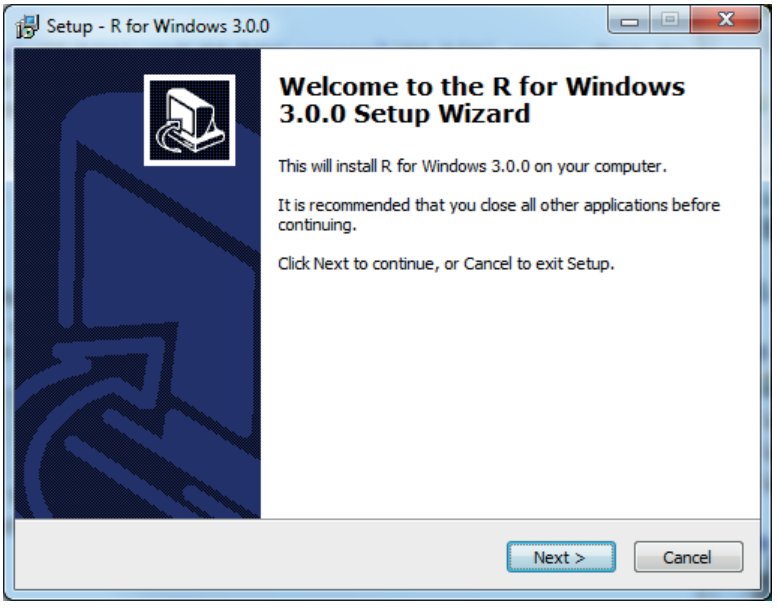


Figure 1.3 With modern versions of Windows, this suggestion can be safely ignored.

The software license is then displayed, as in Figure 1.4. R cannot be used without agreeing to this (important) license, so the only recourse is to click Next.

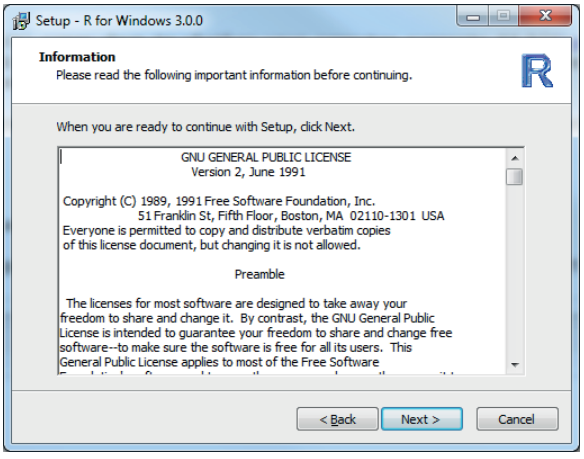


Figure 1.4 The license agreement must be acknowledged to use R.

The installer then asks for a destination location. Even though the official advice from CRAN is that R should be installed in a directory with no spaces in the name, half the time the default installation directory is `Program Files\R`, which causes trouble if we try to build packages that require compiled code such as C++ for FORTRAN. Figure 1.5 shows this dialog. It is important to choose a directory with no spaces, even if the default installation says otherwise.

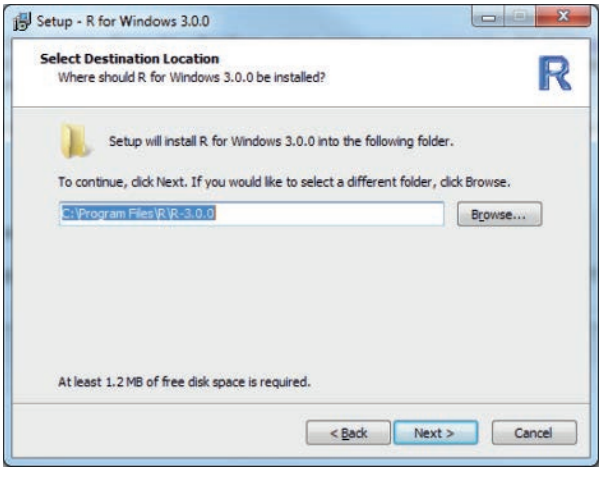


Figure 1.5 It is important to choose a destination folder with no spaces in the name.

If that is the case, click the Browse button to bring up folder options like the ones shown in Figure 1.6.

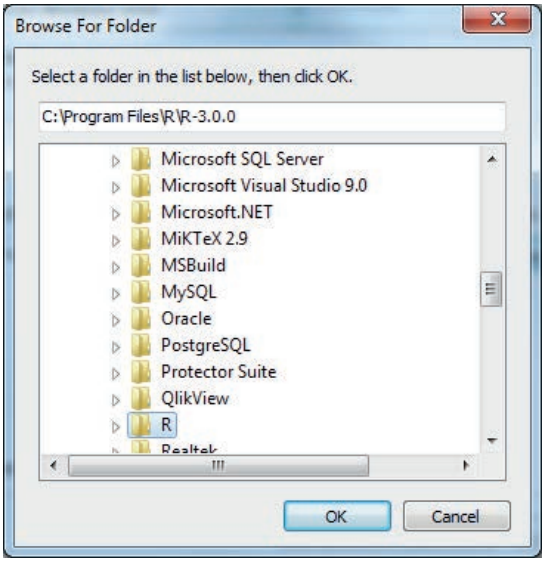


Figure 1.6 This dialog is used to choose the destination folder.

It is best to choose a destination folder that is on the `C:` drive (or another hard disk drive) or inside `My Documents`, which despite that user-friendly name is actually located at `C:\Users\UserName\Documents`, which contains no spaces. Figure 1.7 shows a proper destination for the installation.

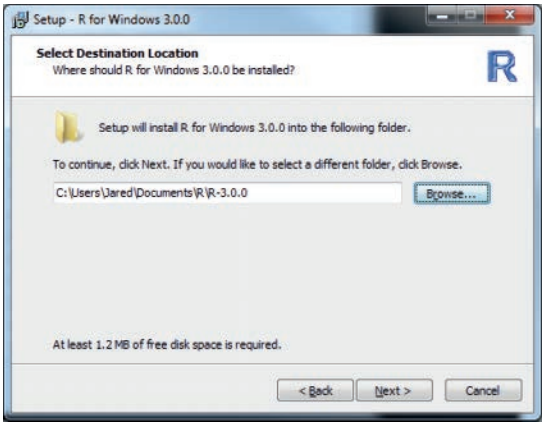


Figure 1.7 This is a proper destination, with no spaces in the name.

Next, Figure 1.8, shows a list of components to install. Unless there is a specific need for 32-bit files, that option can be unchecked. Everything else should be selected.

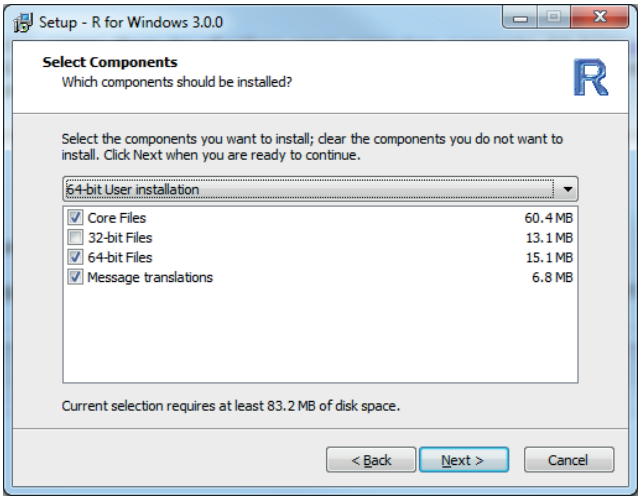


Figure 1.8 It is best to select everything except 32-bit components.

The startup options should be left at the default, No, as in Figure 1.9, because there are not a lot of options and we recommend using RStudio as the front end anyway.

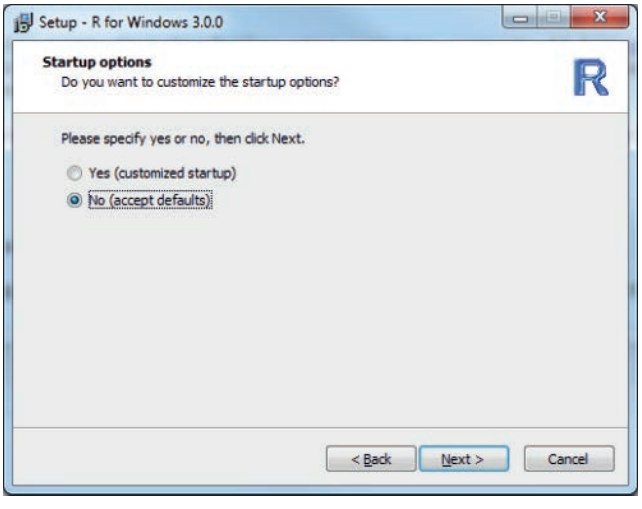


Figure 1.9 Accept the default startup options, as we recommend using RStudio as the front end, and these will not be important.

Next, choose where to put the start menu shortcuts. We recommend simply using R and putting every version in there as shown in Figure 1.10.

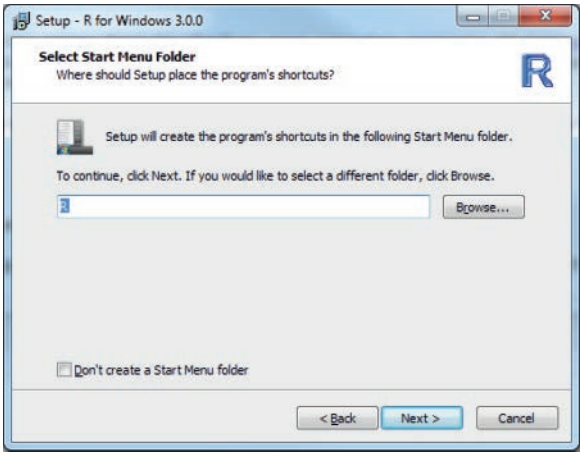


Figure 1.10 Choose the Start Menu folder where the shortcuts will be installed.

We have many versions of R, all inside the same Start Menu folder, which allows code to be tested in different versions. This is illustrated in Figure 1.11.

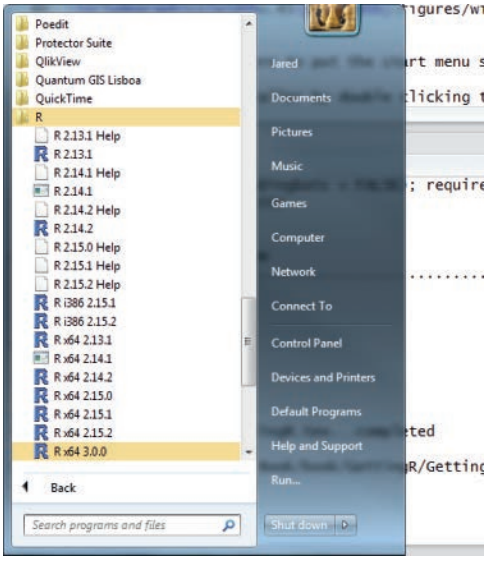


Figure 1.11 We have multiple versions of R installed to allow development and testing with different versions.

The last option is choosing whether to complete some additional tasks such as creating a desktop icon (not too useful if using RStudio). We highly recommend saving the version number in the registry and associating R with `RData` files. These options are shown in Figure 1.12.

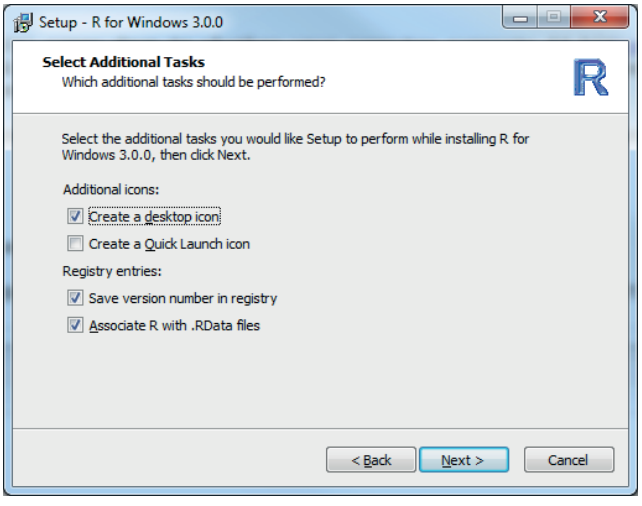


Figure 1.12 We recommend saving the version number in the registry and associating R with `RData` files.

Clicking Next begins installation and displays a progress bar, as shown in Figure 1.13.

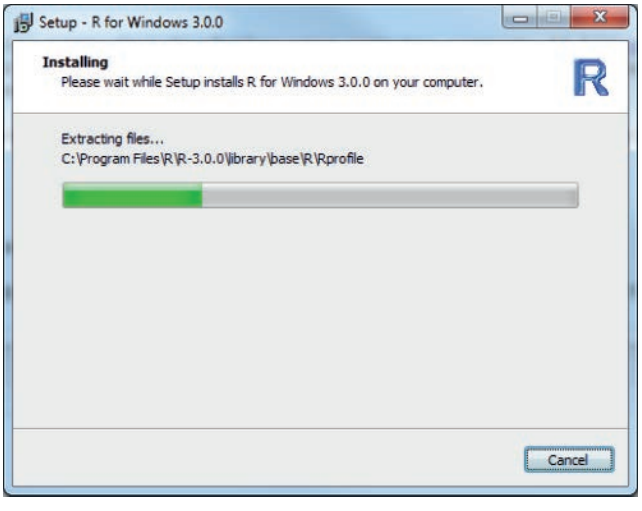


Figure 1.13 A progress bar is displayed during installation.

The last step, shown in Figure 1.14, is to click Finish, confirming the installation is complete.

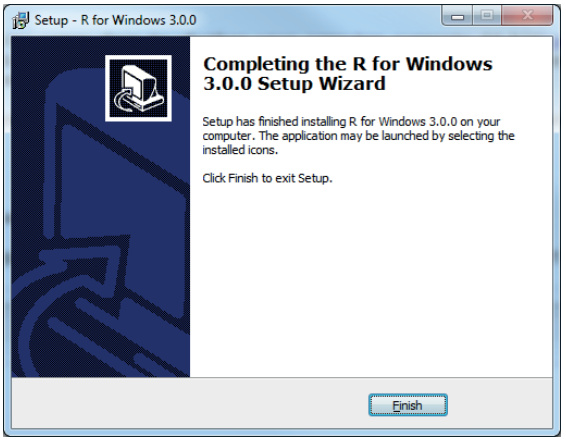


Figure 1.14 Confirmation that installation is complete.

1.4.2 Installing on Mac OS X

Find the appropriate installer, which ends in `.pkg`, and launch it by double-clicking. This brings up the introduction, shown in Figure 1.15. Click Continue to begin the installation process.

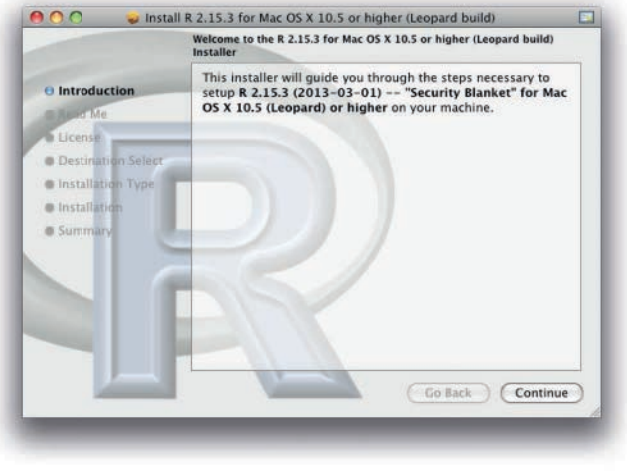


Figure 1.15 Introductory screen for installation on a Mac.

This brings up some information about the version of R being installed. There is nothing to do except click Continue, as shown in Figure 1.16.

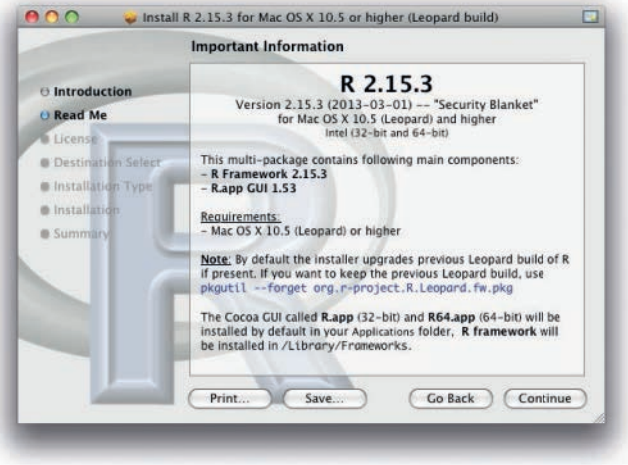


Figure 1.16 Version selection.

Then the license information is displayed, as in Figure 1.17. Click Continue to proceed, the only viable option in order to use R.

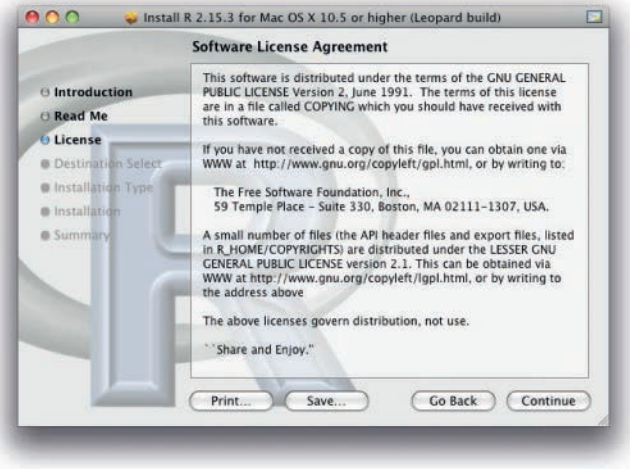


Figure 1.17 The license agreement, which must be acknowledged to use R.

Click Agree to confirm that the license is agreed to, which is mandatory to use R, as is evidenced in Figure 1.18.

Figure 1.18 The license agreement must also be agreed to.

To install R for all users, click Install; otherwise, click Change Install Location to pick a different location. This is shown in Figure 1.19.

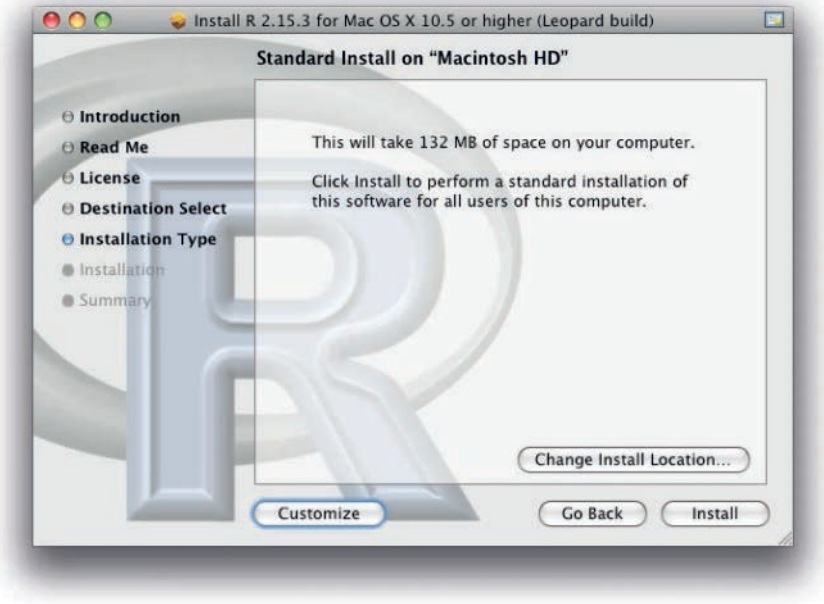


Figure 1.19 By default R is installed for all users, although there is the option to choose a specific location.

If prompted, enter the necessary password as shown in Figure 1.20.

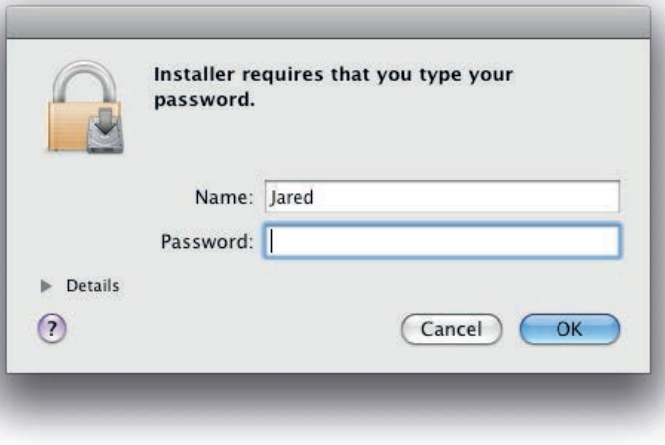


Figure 1.20 The administrator password might be required for installation.

This starts the installation process, which displays a progress bar as shown in Figure 1.21.

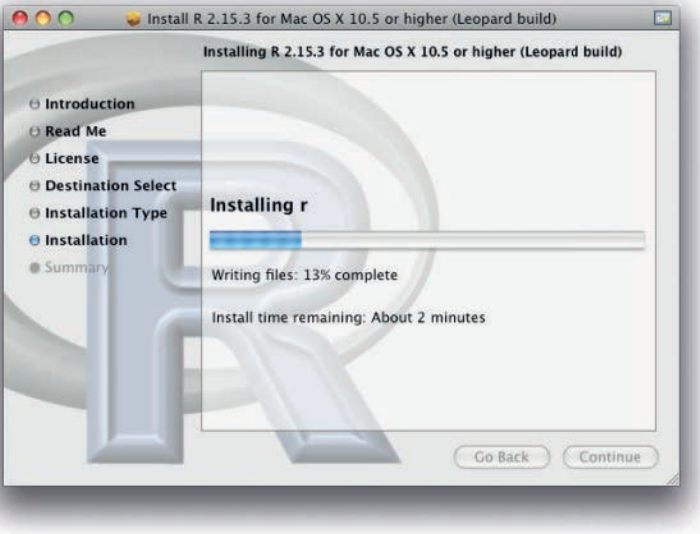


Figure 1.21 A progress bar is displayed during installation.

When done, the installer signals success as Figure 1.22 shows. Click Close to finish the installation.

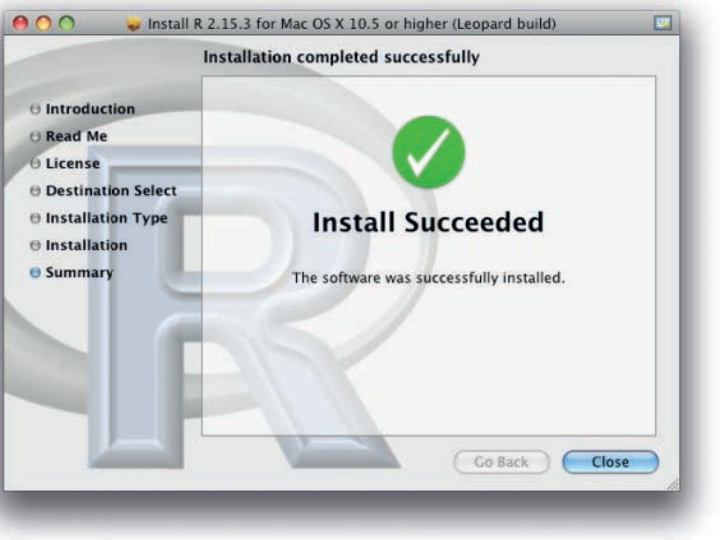


Figure 1.22 This signals a successful installation.

1.4.3 Installing on Linux

Retrieving R from its standard distribution mechanism will download, build and install R in one step. We will focus on Ubuntu, which uses apt-get.

The first step is to edit the file `/etc/apt/sources.list`, which lists package sources. Two pieces of information need to be included: the CRAN mirror and the version of Ubuntu or Debian.

Any CRAN mirror can be used, so we choose the RStudio mirror at `http://cran.rstudio.com/bin/linux/ubuntu`.

The supported versions of Ubuntu, as of early 2017, are Yakkety Yak (16.10), Xenial Xerus (16.04), Wily Werewolf (15.10), Vivid Vervet (15.04), Trusty Tahr (14.04; LTS) and Precise Pangolin (12.04; LTS).[2]

To install R from the RStudio CRAN mirror on Ubuntu 16.04, we need to add the line

```
deb http://cran.rstudio.com/bin/linux/ubuntu xenial/
```

to `/etc/apt/sources.list`. This can be done manually or by running the following command in the terminal.

```
sudo sh -c \
'echo "deb http://cran.rstudio.com/bin/linux/ubuntu xenial/" \
>> /etc/apt/sources.list'
```

2. According to `https://cran.r-project.org/bin/linux/ubuntu/README`

Then we add a public key to authenticate the packages.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com
    --recv-keys E084DAB9
```

Now we can update apt-get and install R. We install both R base and R devel so we can build packages from source or build our own.

```
sudo apt-get update
sudo apt-get install r-base
sudo apt-get install r-base-dev
```

R is also natively supported on Debian, Red Hat and SuSE.

1.5 Microsoft R Open

Microsoft, which purchased Revolution Analytics, offers a community version of its build of R, called Microsoft R Open, featuring an Integrated Development Environment based on Visual Studio and built with the Intel Matrix Kernel Library (MKL), enabling faster matrix computations. It is available for free at `https://mran.microsoft.com/download/`. They also offer a paid version—Microsoft R Server—that provides specialized algorithms to work on large data and greater operability with Microsoft SQL Server and Hadoop. More information is available at `https://www.microsoft.com/en-us/server-cloud/products/r-server/`.

1.6 Conclusion

At this point R is fully usable and comes with a crude GUI. However, it is best to install RStudio and use its interface, which is detailed in Section 2.2. The process involves downloading and launching an installer, just as with any other program.

2

The R Environment

Now that R is downloaded and installed, it is time to get familiar with how to use R. The basic R interface on Windows is fairly Spartan, as seen in Figure 2.1. The Mac interface (Figure 2.2) has some extra features and Linux has far fewer, being just a terminal.

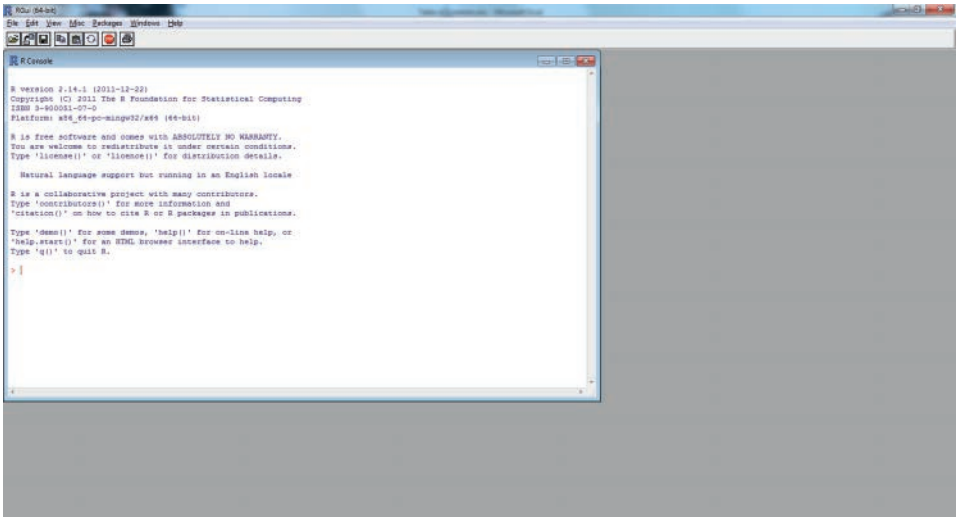


Figure 2.1 The standard R interface in Windows.

Unlike other languages, R is very interactive. That is, results can be seen one command at a time. Languages such as C++ require that an entire section of code be written, compiled and run in order to see results. The state of objects and results can be seen at any point in R. This interactivity is one of the most amazing aspects of working with R.

There have been numerous Integrated Development Environments (IDEs) built for R. For the purposes of this book we will assume that RStudio is being used, which is discussed in Section 2.2.

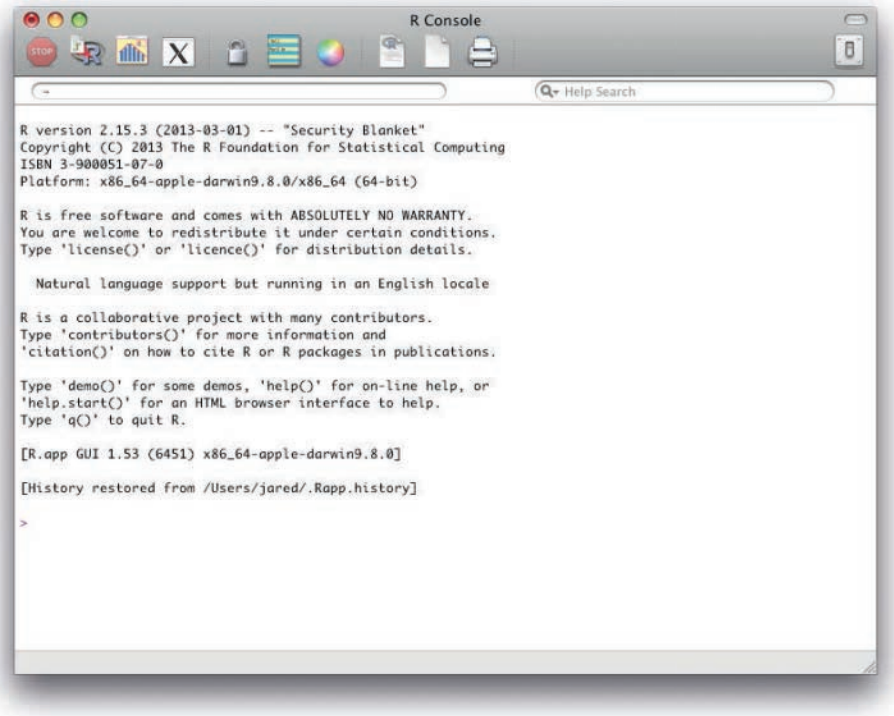


Figure 2.2 The standard R interface on Mac OS X.

2.1 Command Line Interface

The command line interface is what makes R so powerful, and also frustrating to learn. There have been attempts to build point-and-click interfaces for R, such as Rcmdr, but none have truly taken off. This is a testament to how typing in commands is much better than using a mouse. That might be hard to believe, especially for those coming from Excel, but over time it becomes easier and less error prone.

For instance, fitting a regression in Excel takes at least seven mouse clicks, often more: `Data >> Data Analysis >> Regression >> OK >> Input Y Range >> Input X Range >> OK`. Then it may need to be done all over again to make one little tweak or because there are new data. Even harder is walking a colleague through those steps via email. In contrast, the same command is just one line in R, which can easily be repeated and copied and pasted. This may be hard to believe initially, but after some time the command line makes life much easier.

To run a command in R, type it into the console next to the > symbol and press the Enter key. Entries can be as simple as the number 2 or complex functions, such as those seen in Chapter 8.

To repeat a line of code, simply press the Up Arrow key and hit Enter again. All previous commands are saved and can be accessed by repeatedly using the Up and Down Arrow keys to cycle through them.

Interrupting a command is done with `Esc` in Windows and Mac and `Ctrl-C` in Linux.

Often when working on a large analysis it is good to have a file of the code used. Until a few years ago, the most common way to handle this was to use a text editor[1] such as Sublime Text or Notepad++ to write code and then copy and paste it into the R console. While this worked, it was sloppy and led to a lot of switching between programs. Thankfully, there is now RStudio, which is a game changer and detailed in Section 2.2.

2.2 RStudio

While there are a number of IDEs available, the best right now is RStudio, created by a team led by JJ Allaire whose previous products include ColdFusion and Windows Live Writer. It is available for Windows, Mac and Linux and looks identical in all of them. Even more impressive is the RStudio Server, which runs an R instance on a Linux server and enables the user to run commands through the standard RStudio interface in a Web browser. It works with any version of R (greater than 2.11.1), including Microsoft R Open and Microsoft R Server from Microsoft. RStudio has so many options that it can be a bit overwhelming. We cover some of the most useful or frequently used features.

RStudio is highly customizable, but the basic interface looks roughly like Figure 2.3. In this case the lower left pane is the R console, which can be used just like the standard R console. The upper left pane takes the place of a text editor but is far more powerful. The upper right pane holds information about the workspace, command history, files in the current folder and Git version control. The lower right pane displays plots, package information and help files.

There are a number of ways to send and execute commands from the editor to the console. To send one line, place the cursor at the desired line and press `Ctrl+Enter` (`Command+Enter` on Mac). To insert a selection, simply highlight the selection and press `Ctrl+Enter`. To run an entire file of code, press `Ctrl+Shift+S`.

When typing code, such as an object name or function name, hitting `Tab` will autocomplete the code. If more than one object or function matches the letters typed so far, a dialog will pop up giving the matching options, as shown in Figure 2.4.

Typing `Ctrl+1` moves the cursor to the text editor area and `Ctrl+2` moves it to the console. To move to the previous tab in the text editor, press `Ctrl+Alt+Left` on Windows, `Ctrl+PageUp` in Linux and `Ctrl+Option+Left` on Mac. To move to the next tab in the text editor, press `Ctrl+Alt+Right` in Windows, `Ctrl+PageDown` in Linux and `Ctrl+Option+Right` on Mac. On some Windows machines these shortcuts can cause the screen to rotate, so `Ctrl+F11` and `Ctrl+F12` also move between tabs as does `Ctrl+Alt+Left` and `Ctrl+Alt+Right`, though only in the desktop client. For an almost-complete list of shortcuts, click `Help >> Keyboard Shortcuts` or use the keyboard shortcut `Alt+Shift+K` on Windows and Linux and `Option+Shift+K` on

1. This means a programming text editor as opposed to a word processor such as Microsoft Word. A text editor preserves the structure of the text, whereas word processors may add formatting that makes it unsuitable for insertion into the console.

Mac. A more complete list is available at `https://support.rstudio.com/hc/en-us/articles/200711853-Keyboard-Shortcuts`.

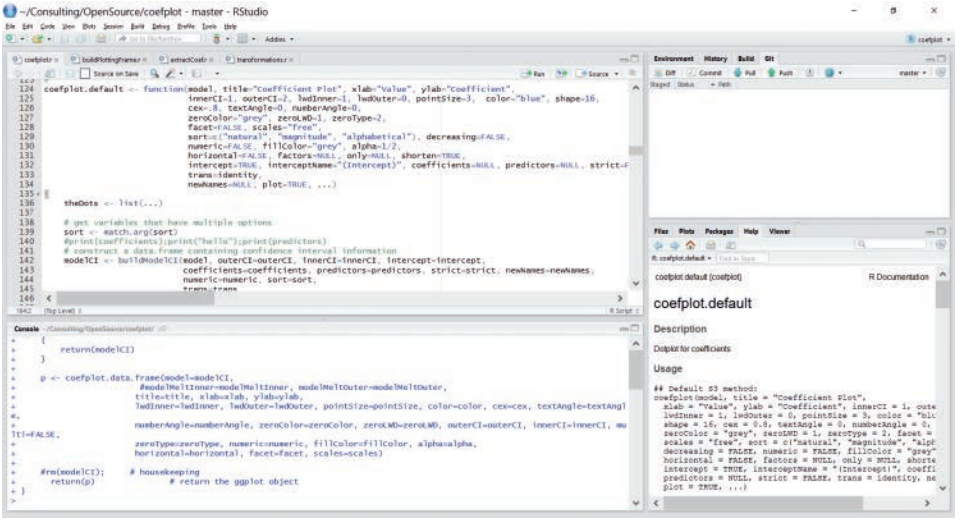


Figure 2.3 The general layout of RStudio.

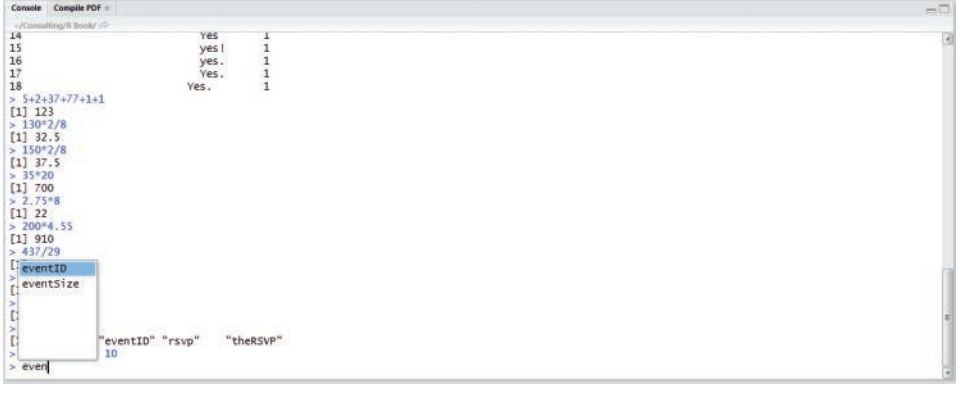


Figure 2.4 Object Name Autocomplete in RStudio.

2.2.1 RStudio Projects

A primary feature of RStudio is projects. A project is a collection of files—and possibly data, results and graphs—that are all related to each other.[2] Each package even has its own working directory. This is a great way to keep organized.

The simplest way to start a new project is to click `File >> New Project`, as in Figure 2.5.

Figure 2.5 Clicking `File >> New Project` begins the project creation process.

Three options are available, shown in Figure 2.6: starting a new project in a new directory, associating a project with an existing directory or checking out a project from a version control repository such as Git or SVN[3]. In all three cases a `.Rproj` file is put into the resulting directory and keeps track of the project.

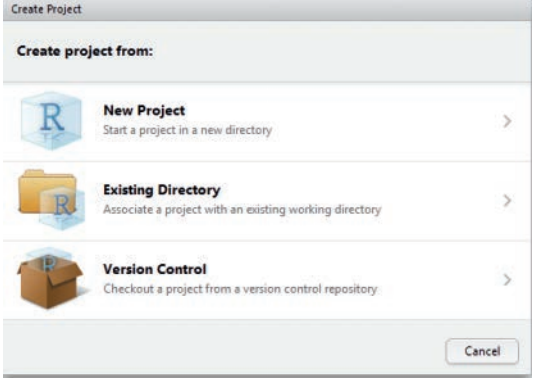


Figure 2.6 Three options are available to start a new project: a new directory, associating a project with an existing directory or checking out a project from a version control repository.

2. This is different from an R session, which is all the objects and work done in R and kept in memory for the current usage period, and usually resets upon restarting R.
3. Using version control requires that the version control program is installed on the computer.

Choosing to create a new directory brings up a dialog, shown in Figure 2.7, that requests a project name and where to create a new directory.

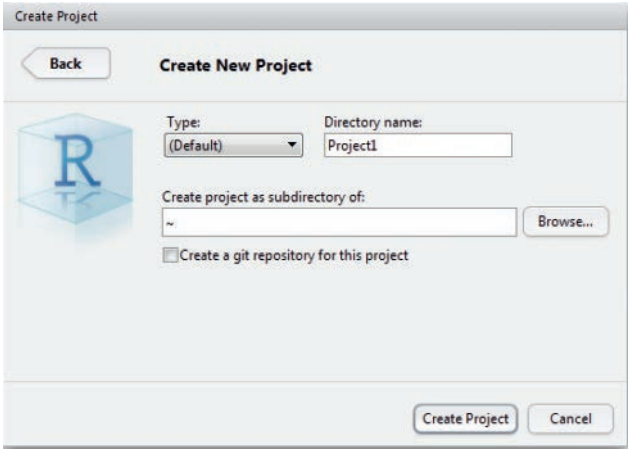


Figure 2.7 Dialog to choose the location of a new project directory.

Choosing an existing directory asks for the name of the directory, as shown in Figure 2.8.

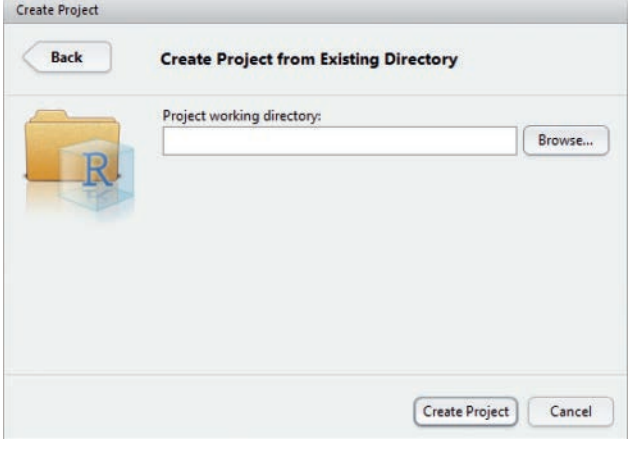


Figure 2.8 Dialog to choose an existing directory in which to start a project.

Choosing to use version control (we prefer Git) firsts asks whether to use Git or SVN as in Figure 2.9.

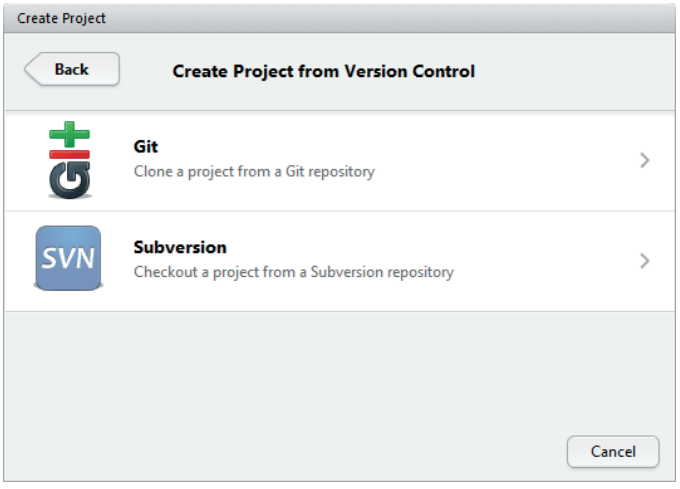


Figure 2.9 Here is the option to choose which type of repository to start a new project from.

Selecting Git asks for a repository URL, such as `git@github.com:jaredlander/coefplot.git`, which will then fill in the project directory name, as shown in Figure 2.10. As with creating a new directory, this will ask where to put this new directory.

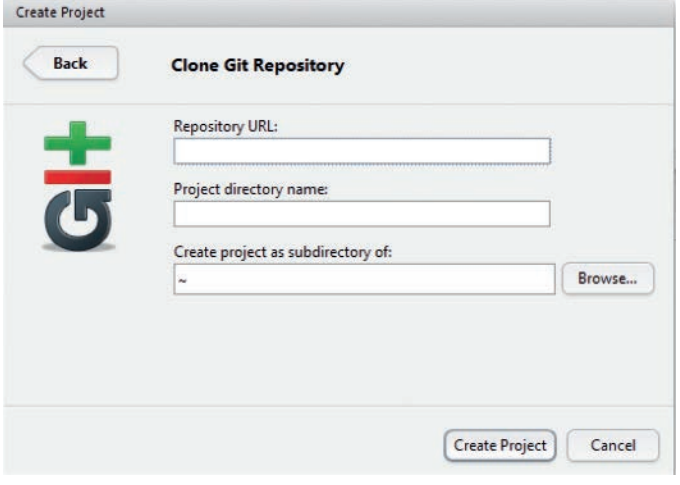


Figure 2.10 Enter the URL for a Git repository, as well as the folder where this should be cloned to.

2.2.2 RStudio Tools

RStudio is highly customizable with a lot of options. Most are contained in the Options dialog accessed by clicking `Tools >> Global Options`, as seen in Figure 2.11.

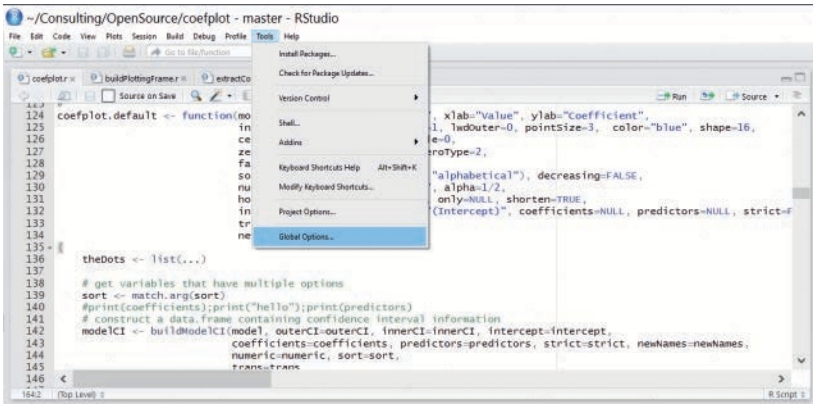


Figure 2.11 Clicking `Tools >> Options` brings up RStudio options.

First are the General options, shown in Figure 2.12. On Windows there is a control for selecting which version of R to use. This is a powerful tool when a computer has a

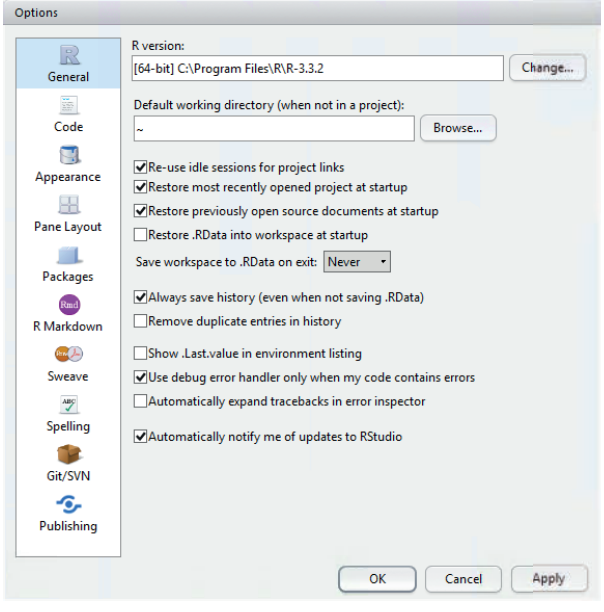


Figure 2.12 General options in RStudio.

number of versions of R. However, RStudio must be restarted after changing the R version. In the future, RStudio is slated to offer the capability to set different versions of R for each project. It is also a good idea to not restore or save `.RData` files on startup and exiting.[4] This way each time R is started it is a fresh session without potential variable corruptions or unnecessary data occupying memory.

Code editing options, shown in Figure 2.13, control the way code is entered and displayed in the text editor. It is generally considered good practice to replace tabs with spaces, either two or four,[5] as tabs can sometimes be interpreted differently by different text editors. Some hard-core programmers will appreciate vim and Emacs modes.

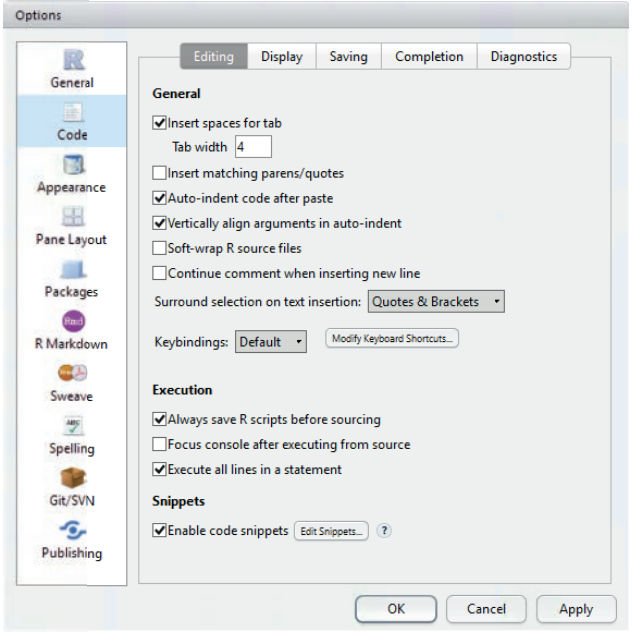


Figure 2.13 Options for customizing code editing.

Code display options, shown in Figure 2.14, control visual cues in the text editor and console. Highlighting selected words makes it easy to spot multiple occurrences. Showing line numbers are a must to ease code navigation. Showing a margin column gives a good indication of when a line of code is too long to be easily read.

4. `RData` files are a convenient way of saving and sharing R objects and are discussed in Section 6.5.
5. Four is better for working with Markdown documents.

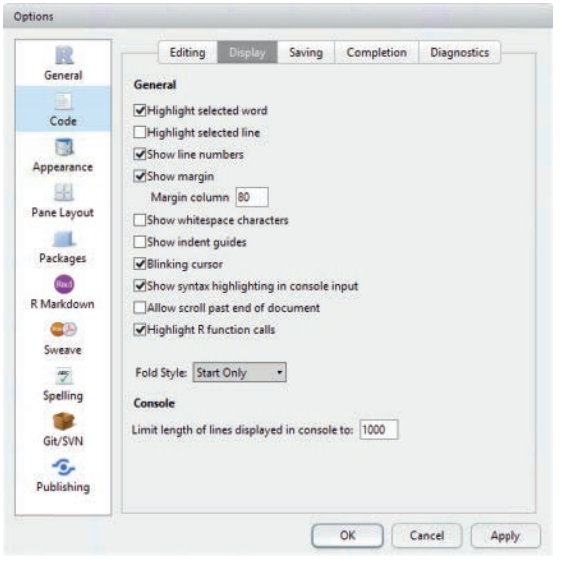


Figure 2.14 Options for customizing code display.

Code saving options, shown in Figure 2.15, control how the text files containing code are saved. For the most part it is good to use the defaults, especially selecting "Platform Native" for the line ending conversion under "Serialization."

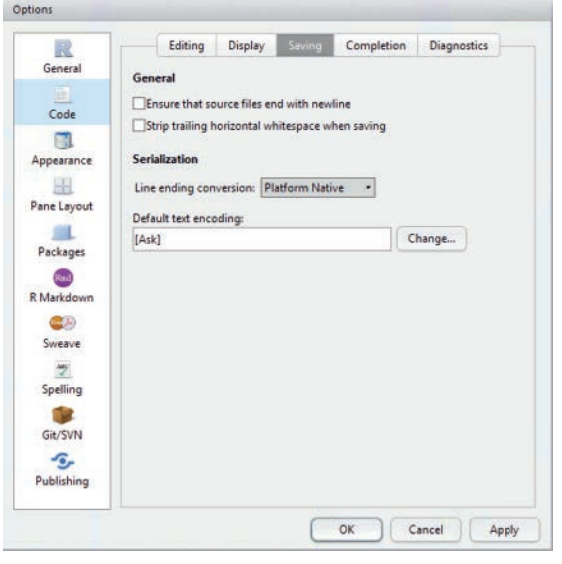


Figure 2.15 Options for customizing code saving.

Code Completion options, shown in Figure 2.16, control how code is completed while programming. Some people like having parentheses added automatically after typing a function, and others do not. One particularly divisive setting is whether to put spaces around the equals sign for named function arguments.

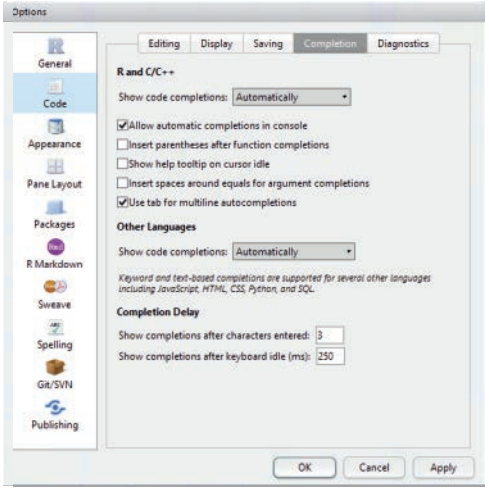


Figure 2.16 Options for customizing code completion.

Code diagnostics options, shown in Figure 2.17, enable code checking. These can be very helpful in identifying mistyped object names, poor style and general mistakes.

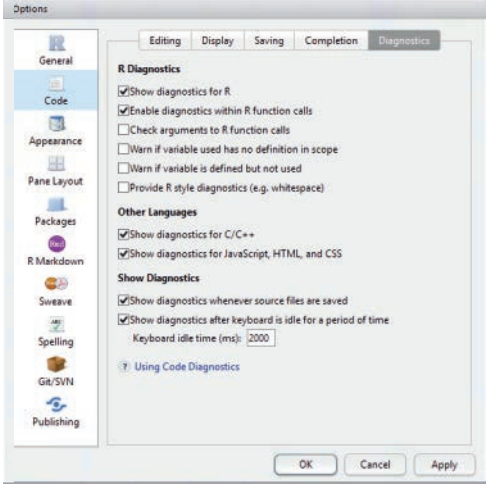


Figure 2.17 Options for customizing code diagnostics.

Appearance options, shown in Figure 2.18, change the way code looks, aesthetically. The font, size and color of the background and text can all be customized here.

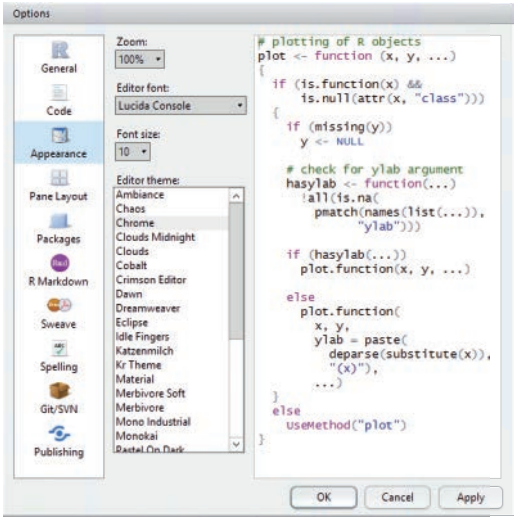


Figure 2.18 Options for code appearance.

The Pane Layout options, shown in Figure 2.19, simply rearrange the panes that make up RStudio.

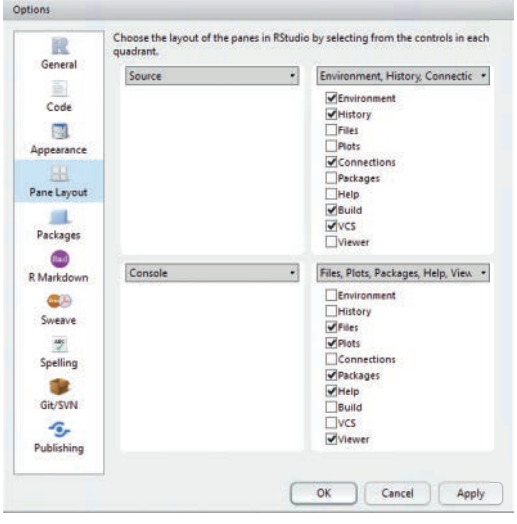


Figure 2.19 These options control the placement of the various panes in RStudio.

The Packages options, shown in Figure 2.20, set options regarding packages, although the most important is the CRAN mirror. While this is changeable from the console, this is the default setting. It is best to pick the mirror that is geographically the closest.

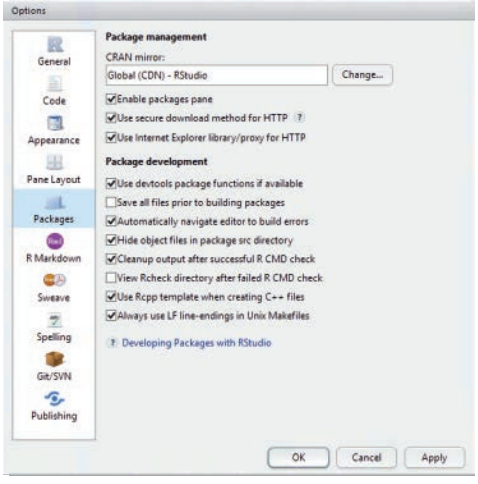


Figure 2.20 Options related to packages. The most important is the CRAN mirror selection.

The RMarkdown options, seen in Figure 2.21, control settings for working with RMarkdown documents. This allows rendered documents to be previewed in an external window or in the Viewer pane. It also lets RMarkdown files be treated like notebooks, rendering results, images and equations inline.

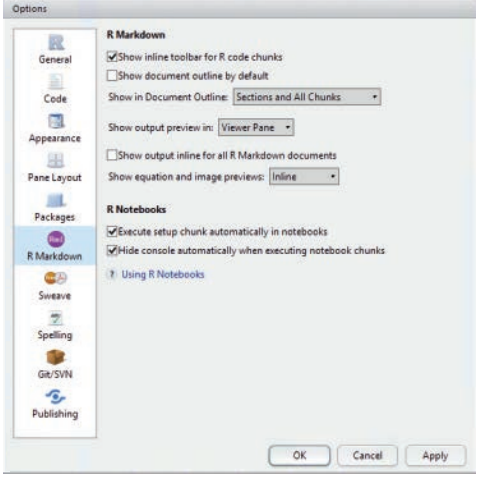


Figure 2.21 Options for RMarkdown, including whether to treat them like notebooks.

The Sweave options, seen in Figure 2.22, may be a bit misnamed, as this is where to choose between using Sweave or **knitr**. Both are used for the generation of PDF documents with **knitr** also enabling the creation of `HTML` documents. **knitr**, detailed in Chapter 27, is by far the better option, although it must be installed first, which is explained in Section 3.1. This is also where the PDF viewer is selected.

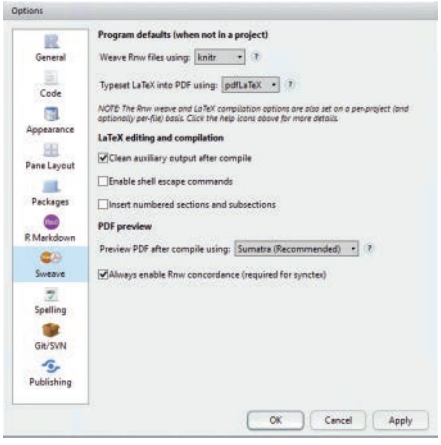


Figure 2.22 This is where to choose whether to use Sweave or knitr and select the PDF viewer.

RStudio contains a spelling checker for writing LaTeX and Markdown documents (using **knitr**, preferably), which is controlled from the Spelling options, shown in Figure 2.23. Not much needs to be set here.

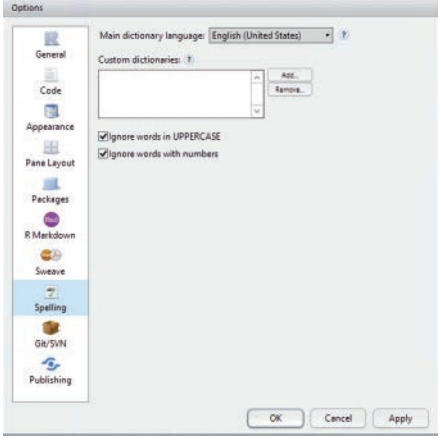


Figure 2.23 These are the options for the spelling check dictionary, which allows language selection and the custom dictionaries.

The Git/SVN options, shown in Figure 2.24, indicates where the executables for Git and SVN exist. This needs to be set only once but is necessary for version control.

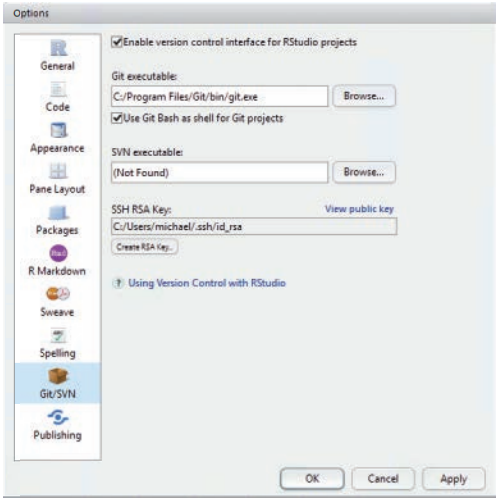


Figure 2.24 This is where to set the location of Git and SVN executables so they can be used by RStudio.

The last option, Publishing, Figure 2.25, sets connections for publishing documents to ShinyApps.io or RStudio Connect.

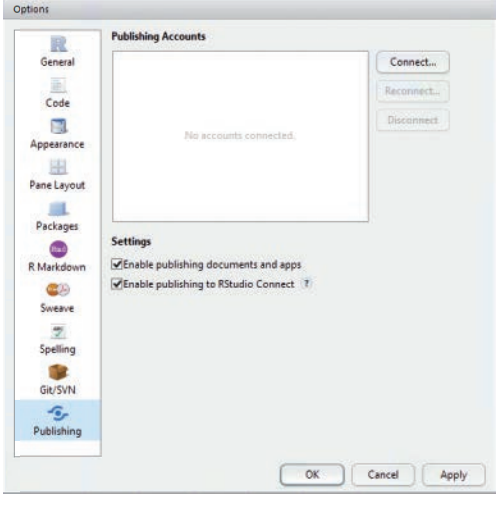


Figure 2.25 This is where to set connections to ShinyApps.io or RStudio Connect.