

The R Series

Dynamic Documents with R and knitr

Second Edition



Yihui Xie



CRC Press

Taylor & Francis Group

A CHAPMAN & HALL BOOK

Dynamic Documents with R and knitr

Second Edition

Yihui Xie

RStudio, Inc.



CRC Press

Taylor & Francis Group

Boca Raton London New York

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

Chapman & Hall/CRC

The R Series

Series Editors

John M. Chambers
Department of Statistics
Stanford University
Stanford, California, USA

Torsten Hothorn
Division of Biostatistics
University of Zurich
Switzerland

Duncan Temple Lang
Department of Statistics
University of California, Davis
Davis, California, USA

Hadley Wickham
RStudio
Boston, Massachusetts, USA

Aims and Scope

This book series reflects the recent rapid growth in the development and application of R, the programming language and software environment for statistical computing and graphics. R is now widely used in academic research, education, and industry. It is constantly growing, with new versions of the core software released regularly and more than 6,000 packages available. It is difficult for the documentation to keep pace with the expansion of the software, and this vital book series provides a forum for the publication of books covering many aspects of the development and application of R.

The scope of the series is wide, covering three main threads:

- Applications of R to specific disciplines such as biology, epidemiology, genetics, engineering, finance, and the social sciences.
- Using R for the study of topics of statistical methodology, such as linear and mixed modeling, time series, Bayesian methods, and missing data.
- The development of R, including programming, building packages, and graphics.

The books will appeal to programmers and developers of R software, as well as applied statisticians and data analysts in many fields. The books will feature detailed worked examples and R code fully integrated into the text, ensuring their usefulness to researchers, practitioners and students.

Published Titles

Stated Preference Methods Using R, *Hideo Aizaki, Tomoaki Nakatani, and Kazuo Sato*

Using R for Numerical Analysis in Science and Engineering, *Victor A. Bloomfield*

Event History Analysis with R, *Göran Broström*

Computational Actuarial Science with R, *Arthur Charpentier*

Statistical Computing in C++ and R, *Randall L. Eubank and Ana Kupresanin*

Reproducible Research with R and RStudio, Second Edition, *Christopher Gandrud*

Introduction to Scientific Programming and Simulation Using R, Second Edition, *Owen Jones, Robert Maillardet, and Andrew Robinson*

Nonparametric Statistical Methods Using R, *John Kloke and Joseph McKean*

Displaying Time Series, Spatial, and Space-Time Data with R,
Oscar Perpiñán Lamigueiro

Programming Graphical User Interfaces with R, *Michael F. Lawrence and John Verzani*

Analyzing Sensory Data with R, *Sébastien Lê and Theirry Worch*

Parallel Computing for Data Science: With Examples in R, C++ and CUDA,
Norman Matloff

Analyzing Baseball Data with R, *Max Marchi and Jim Albert*

Growth Curve Analysis and Visualization Using R, *Daniel Mirman*

R Graphics, Second Edition, *Paul Murrell*

Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving, *Deborah Nolan and Duncan Temple Lang*

Multiple Factor Analysis by Example Using R, *Jérôme Pagès*

Customer and Business Analytics: Applied Data Mining for Business Decision Making Using R, *Daniel S. Putler and Robert E. Krider*

Implementing Reproducible Research, *Victoria Stodden, Friedrich Leisch, and Roger D. Peng*

Graphical Data Analysis with R, *Antony Unwin*

Using R for Introductory Statistics, Second Edition, *John Verzani*

Advanced R, *Hadley Wickham*

Dynamic Documents with R and knitr, Second Edition, *Yihui Xie*

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

© 2015 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20150519

International Standard Book Number-13: 978-1-4987-1697-0 (eBook - PDF)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

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

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

To my parents
Shaobai Xie *and* Guolan Xie

Contents

Preface	xiii
Author	xxi
List of Figures	xxiii
List of Tables	xxvii
1 Introduction	1
2 Reproducible Research	5
2.1 Literature	5
2.2 Good and Bad Practices	7
2.3 Barriers	9
3 A First Look	11
3.1 Setup	11
3.2 Minimal Examples	12
3.2.1 An Example in L ^A T _E X	12
3.2.2 An Example in Markdown	15
3.3 Quick Reporting	17
3.4 Extracting R Code	17
4 Editors	19
4.1 RStudio	19
4.2 LyX	23
4.3 Emacs/ESS	25
4.4 Other Editors	26
5 Document Formats	27
5.1 Input Syntax	27
5.1.1 Chunk Options	28
5.1.2 Chunk Label	29
5.1.3 Global Options	30
5.1.4 Chunk Syntax	30

5.2	Document Formats	31
5.2.1	Markdown	31
5.2.2	L ^A T _E X	35
5.2.3	HTML	35
5.2.4	reStructuredText	36
5.2.5	AsciiDoc	36
5.2.6	Textile	37
5.2.7	Customization	37
5.3	Output Renderers	39
5.4	R Scripts	43
6	Text Output	45
6.1	Inline Output	45
6.2	Chunk Output	46
6.2.1	Chunk Evaluation	46
6.2.2	Code Formatting	47
6.2.3	Code Decoration	47
6.2.4	Show/Hide Output	49
6.2.5	Collapse Output	51
6.2.6	Trim Blank Lines	52
6.3	Tables	52
6.4	Automatic Printing	55
6.5	Themes	56
7	Graphics	59
7.1	Graphical Devices	60
7.1.1	Custom Device	60
7.1.2	Choose a Device	60
7.1.3	Device Size	61
7.1.4	More Device Options	61
7.1.5	Encoding	62
7.1.6	The Dingbats Font	64
7.2	Plot Recording	64
7.3	Plot Rearrangement	69
7.3.1	Animation	70
7.3.2	Alignment	71
7.4	Plot Size in Output	72
7.5	Extra Output Options	73
7.6	The <i>tikz()</i> Device	74
7.7	Figure Environment	76
7.8	Figure Path	78

- 8 Cache 81**
 - 8.1 Implementation 81
 - 8.2 Write Cache 82
 - 8.3 When to Update Cache 83
 - 8.4 Side Effects 84
 - 8.5 Chunk Dependencies 86
 - 8.5.1 Manual Dependency 86
 - 8.5.2 Automatic Dependency 87
 - 8.6 Load Cache Manually 88
 - 8.7 Other Options 89
- 9 Cross Reference 91**
 - 9.1 Chunk Reference 91
 - 9.1.1 Embed Code Chunks 91
 - 9.1.2 Reuse Whole Chunks 92
 - 9.2 Code Externalization 93
 - 9.2.1 Labeled Chunks 93
 - 9.2.2 Line-Based Chunks 94
 - 9.3 Child Documents 95
 - 9.3.1 Input Child Documents 95
 - 9.3.2 Child Documents as Templates 96
 - 9.3.3 Standalone Mode 96
- 10 Hooks 99**
 - 10.1 Chunk Hooks 99
 - 10.1.1 Create Chunk Hooks 99
 - 10.1.2 Trigger Chunk Hooks 100
 - 10.1.3 Hook Arguments 101
 - 10.1.4 Hooks and Chunk Options 101
 - 10.1.5 Write Output 102
 - 10.2 Examples 103
 - 10.2.1 Crop Plots 103
 - 10.2.2 rgl Plots 105
 - 10.2.3 Manually Save Plots 106
 - 10.2.4 Optimize PNG Plots 108
 - 10.2.5 Close an rgl Device 109
 - 10.2.6 WebGL 110
- 11 Language Engines 111**
 - 11.1 Design 111
 - 11.1.1 The Engine Function 112
 - 11.1.2 Engine Options 113
 - 11.2 Languages and Tools 113

11.2.1	C++	113
11.2.2	C/Fortran	115
11.2.3	Interpreted Languages	116
11.2.4	Stan	118
11.2.5	TikZ	120
11.2.6	Graphviz	121
11.2.7	Highlight	122
11.2.8	Other Engines	123
11.3	Persistent Sessions	124
12	Tricks and Solutions	127
12.1	Chunk Options	127
12.1.1	Option Aliases	127
12.1.2	Option Templates	128
12.1.3	Program Chunk Options	128
12.1.4	Code in Appendix	130
12.1.5	Local R Options	131
12.1.6	Dynamic Code	131
12.2	Package Options	131
12.3	Typesetting	132
12.3.1	Output Width	132
12.3.2	Message Colors	133
12.3.3	Box Padding	134
12.3.4	Beamer	135
12.3.5	Suppress Long Output	137
12.3.6	Escape Special Characters	138
12.3.7	The Example Environment	139
12.3.8	The Docco Style	140
12.4	Utilities	141
12.4.1	R Package Citation	143
12.4.2	Image URI	144
12.4.3	Upload Images	145
12.4.4	Compile Documents	145
12.4.5	Construct Code Chunks	146
12.4.6	Extract Source Code	147
12.4.7	Reproducible Simulation	150
12.4.8	R Documentation	151
12.4.9	Rst2pdf	151
12.4.10	Package Demos	152
12.4.11	Pretty Printing	152
12.4.12	A Macro Preprocessor	155
12.4.13	Exit Knitting Early	156
12.4.14	Literal knitr Source Code	157

12.4.15	Spell Checking	158
12.5	Debugging	159
12.6	Multilingual Support	160
13	Publishing Reports	161
13.1	RStudio	161
13.2	Pandoc	162
13.3	HTML5 Slides	163
13.4	Jekyll	165
13.5	WordPress	166
14	R Markdown	167
14.1	Overview	167
14.2	Pandoc's Markdown Extensions	169
14.2.1	Basic Syntax	169
14.2.2	YAML Metadata	172
14.3	Output Formats	172
14.3.1	HTML Document	173
14.3.2	L ^A T _E X/PDF Document	184
14.3.3	Word Document	188
14.3.4	Markdown Documents	190
14.3.5	ioslides Presentation	191
14.3.6	Slidy Presentation	193
14.3.7	Beamer Presentation	194
14.3.8	Other Formats	198
14.4	Interactive Documents with Shiny	199
14.5	Extending R Markdown v2	203
14.5.1	Templates	204
14.5.2	New Formats	205
14.5.3	HTML Widgets	208
14.6	Changes in R Markdown from v1 to v2	209
15	Applications	213
15.1	Homework	213
15.2	Serve Dynamic Documents	217
15.3	Website and Blogging	219
15.3.1	Vistat and Rcpp Gallery	219
15.3.2	UCLA R Tutorial	220
15.3.3	The cda and RHadoop Wiki	220
15.3.4	The ggbio Package	220
15.3.5	Geospatial Data in R and Beyond	221
15.4	Package Vignettes	221
15.4.1	Vignette Metadata and Engines	222

15.4.2	Vignette Examples	224
15.4.3	PDF Vignette	226
15.4.4	HTML Vignette	227
15.5	Books	227
15.5.1	This Book	227
15.5.2	The Analysis of Data	229
15.5.3	The Statistical Sleuth in R	229
15.5.4	Text Analysis with R for Students of Literature	229
15.6	Literate Programming for R Packages	230
16	Other Tools	233
16.1	Sweave	233
16.1.1	Syntax	235
16.1.2	Options	236
16.1.3	Problems	237
16.2	Other R Packages	238
16.3	Python Packages	240
16.3.1	Dexy	241
16.3.2	Python _{TeX}	241
16.3.3	IPython	242
16.4	More Tools	243
16.4.1	Org-mode	244
16.4.2	SASweave	245
16.4.3	Office	245
Appendix		247
A	Internals	247
A.1	Documentation	247
A.2	Closures	248
A.3	Implementation	250
A.3.1	Parser	250
A.3.2	Chunk Hooks	252
A.3.3	Option Aliases	253
A.3.4	Cache	254
A.3.5	Compatibility with Sweave	255
A.3.6	Concordance	255
A.4	Syntax	256
Bibliography		259
Index		265

Preface

We import a dataset into a statistical software package, run a procedure to get all results, then copy and paste selected pieces into a typesetting program, add a few descriptions, and finish a report. This is a common practice in writing statistical reports. There are obvious dangers and disadvantages in this process.

1. It is error-prone due to too much manual work.
2. It requires lots of human effort to do tedious jobs such as copying results across documents.
3. The workflow is barely recordable especially when it involves GUI (Graphical User Interface) operations, therefore it is difficult to reproduce.
4. A tiny change of the data source in the future will require the author(s) to go through the same procedure again, which can take nearly the same amount of time and effort.
5. The analysis and writing are separate, so close attention has to be paid to the synchronization of the two parts.

In fact, a report can be generated dynamically from program code. Just like a software package has its source code, a dynamic document is the source code of a report. It is a combination of computer code and the corresponding narratives. When we compile the dynamic document, the program code in it is executed and replaced with the output; we get a final report by mixing the code output with the narratives. Because we only manage the source code, we are free of all the possible problems above. For example, we can change a single parameter in the source code, and get a different report on the fly.

In this book, *dynamic documents* refer to the kind of source documents containing both program code and narratives. Sometimes we may just call them *source documents* since “dynamic” may sound confusing and ambiguous to some people (it does not mean interactivity or animations). We also use the term *report* frequently throughout the book, which really means the output document that was compiled from a dynamic document.

Who Should Read This Book

This book is written for both beginners and advanced users. The main goal is to make writing reports easier: the “report” here can range from student homework or project reports, exams, books, blogs, and Web pages to virtually any documents related to statistical graphics, computing, and data analysis.

For beginners, Chapters 1 to 8 should be enough for basic applications (which have already covered many features); for power users, Chapters 9 to 11 can be helpful for understanding the extensibility of the **knitr** package.

Familiarity with \LaTeX and HTML can be helpful, but is not required at all. Once you get the basic idea, you can write reports in simple languages such as Markdown, which should be fairly easy for beginners to learn. Unless otherwise noted, all features apply to all document formats, although we primarily use \LaTeX for examples.

We recommend that readers take a look at the website RPubS (<http://rpubs.com>), which contains a large number of user-contributed documents. Hopefully they are convincing enough to show that it is quick and easy to write dynamic documents.

Software Information and Conventions

The main tools we introduce in this book are the R language (R Core Team, 2015) and the **knitr** package (Xie, 2015b), with which this book was written, but the language in the documents is not restricted to R; for example, we can also integrate Python, awk, and shell scripts, etc., into the reports. For document formats, we mainly use \LaTeX , HTML, and Markdown.

Both R and **knitr** are available on CRAN (Comprehensive R Archive Network) as free and open-source software. You may download them from any CRAN mirrors, such as <http://cran.rstudio.com>. You can find their version information for this book in the R session information below:

```
sessionInfo()

## R version 3.2.0 (2015-04-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
```

```
## Running under: Ubuntu 14.04.2 LTS
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8
## [2] LC_NUMERIC=C
## [3] LC_TIME=en_US.UTF-8
## [4] LC_COLLATE=en_US.UTF-8
## [5] LC_MONETARY=en_US.UTF-8
## [6] LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8
## [8] LC_NAME=C
## [9] LC_ADDRESS=C
## [10] LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8
## [12] LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets
## [6] base
##
## other attached packages:
## [1] knitr_1.10
##
## loaded via a namespace (and not attached):
## [1] formatR_1.2  tools_3.2.0  highr_0.5
## [4] stringr_0.6.2 evaluate_0.7
```

The **knitr** package is thoroughly documented on the website <http://yihui.name/knitr/>, and the most important page is perhaps <http://yihui.name/knitr/options>, where you can find the complete reference for chunk options (Section 5.1.1). The development version is hosted on Github: <https://github.com/yihui/knitr>; you can always check out the latest development version, file issues/feature requests, or even participate in the development by forking the repository and making changes by yourself. There are plenty of examples in the repository <https://github.com/yihui/knitr-examples>, including both minimal and advanced examples. Karl Broman prepared a very nice minimal tutorial for **knitr** at http://kbroman.org/knitr_knutshell, which can be useful for beginners to learn **knitr** quickly. There is also a wiki page maintained by Frank Harrell et al. from the Department of Biostatistics, Vanderbilt University, which introduced several tricks and useful experience of using **knitr**: <http://biostat.mc.vanderbilt.edu>.

Unlike many other books on R, we do not add prompts to R source

code in this book, and we comment out the text output by two hashes `##` by default, as you can see from the R session information before. The reason for this convention is explained in Chapter 6. Package names are in bold text (e.g., **rpart**), function names in italic (e.g., *paste()*), inline code is formatted in a typewriter font (e.g., `mean(1:10, trim = 0.1)`), and filenames are in sans serif fonts (e.g., `figure/foo.pdf`).

Structure of the Book

Chapter 1 is an overview of dynamic documents, introducing the idea of literate programming; Chapter 2 explains why dynamic documents are important to scientific research from the viewpoint of reproducible research; Chapter 3 gives a first complete example that covers basic concepts and what we can do with **knitr**; Chapter 4 introduces a few common text editors that support **knitr**, so that it is easier to compile reports from source documents; and Chapter 5 describes the syntax for different document formats such as \LaTeX , HTML, and Markdown.

Chapters 6 to 11 explain the core functionality of the package. Chapters 6 and 7 present how to control text and graphics output from **knitr**. Chapter 8 talks about the caching mechanism that may significantly reduce the computation time. Chapter 9 shows how to reuse source code by chunk references and organize child documents. Chapter 10 consists of an advanced topic — chunk hooks, which make a literate programming document really programmable and extensible. Chapter 11 illustrates how to integrate other languages, such as Python and awk, etc., into one report in the **knitr** framework.

Chapter 12 introduces some useful tricks that make it easier to write documents with **knitr**. Chapter 13 shows how to publish reports in a variety of formats including PDF, HTML, and HTML5 slides. Chapter 14 focuses on R Markdown v2, which can be converted to a large variety of document formats, including those in Chapter 13. Chapter 15 covers a few significant applications. Chapter 16 introduces other tools for dynamic report generation, such as Sweave, other R packages, and software in other languages. Appendix A is a guide to some internal structures of **knitr**, which may be helpful to other package developers.

The topics from Chapters 6 to 11 are parallel to each other. For example, if you want to know more about graphics output, you can skip Chapter 6 and jump to Chapter 7 directly.

In all, we will show how to improve our efficiency in writing re-

ports, fine tune every aspect of a report, and go from program output to publication-quality reports.

What's New in the Second Edition

The major new content in the second edition of this book is Chapter 14, which is an introduction to R Markdown v2. Then there are a few new sections: 6.3 (how to generate tables), 6.4 (how to define custom printing methods for objects in code chunks), 11.2.2 (the C/Fortran engines), 11.2.4 (the Stan engine), 11.3 (how to run engines in a persistent session), and 15.2 (how to start a local server to serve dynamic documents). There are many minor updates here and there in the book as well.

The second edition also introduces several changes according to the changes in the **knitr** package (the first edition was based on **knitr** 1.3).

- The default value of the chunk option `tidy` was changed from `TRUE` to `FALSE`, i.e., code chunks will not be automatically reformatted by default (Section 6.2.2).
- Inline R expressions are evaluated without `try()`, i.e., if an error occurs during the inline evaluation, R will stop immediately.
- The global R option `digits` is no longer modified in **knitr**; its default value is 7, and you can set `options(digits = 4)` if you want the old behavior.
- The `plot` hook function takes the plot filename as its first argument (Section 5.3), instead of a vector of length two (basename and extension).
- The preferred way to stop **knitr** in case of errors is to set the chunk option `error = FALSE` instead of the package option `stop_on_error`, which has been deprecated (Section 6.2.4).
- Syntax highlighting is also available for other languages (Chapter 11) such as Shell scripts, `awk`, and Python, etc., if the Highlight package is installed (Section 11.2.7).
- For external code chunks (Section 9.2), the preferred chunk delimiter is `## ----` instead of `## @knitr` now.

To keep track of the changes in **knitr**, you can see the release notes for each version at <https://github.com/yihui/knitr/releases>.

Acknowledgments

First, I want to thank my wireless router, which was broken when I started writing the core chapters of the first edition of this book (in the boring winter of Ames). Besides, I also thank my wife for not giving me the Ethernet cable during that period.

This book would certainly not have been possible without the powerful R language, for which I thank the R core team and its contributors. The seminal work of Sweave (by Friedrich Leisch and R-core) is the most important source of inspiration of **knitr**. Some additional features were inspired by other R packages including **cacheSweave** (Roger Peng), **pgfSweave** (Cameron Bracken and Charlie Sharpsteen), **weaver** (Seth Falcon), **SweaveListingUtils** (Peter Ruckdeschel), **highlight** (Romain Francois), and **brew** (Jeffrey Horner). The initial design was based on Hadley Wickham's **decumar** package, and the evaluator is based on his **evaluate** package. Both **LyX** and **RStudio** quickly included support to **knitr** after it came out, which made it a lot easier to write source documents, and I'd like to thank their developers (especially Jean-Marc Lasgouttes, JJ Allaire, and Joe Cheng); similarly I thank the developers of other editors such as Emacs/ESS. I do not know how to describe John MacFarlane's Pandoc. It is magic. "Yes, we do support Word! Welcome to the world of reproducible research!"

The R/**knitr** user community is truly amazing. There has been a lot of feedback since the beginning of its development in late 2011. I still remember some users shouted it from the rooftops when I released the first beta version. I appreciate this kind of excitement. Thousands of questions and comments in the mailing list (<https://groups.google.com/group/knitr>) and on the website StackOverflow (<http://stackoverflow.com/tags/knitr/>) made this package far more powerful than I imagined. The development repository is on Github, where I have received nearly 800 issues and more than 160 pull requests from many contributors, including Ramnath Vaidyanathan, Taiyun Wei, Kirill Müller, and JJ Allaire (<https://github.com/yihui/knitr/pulls>).

```
# to see a full list of contributors  
packageDescription("knitr", fields = "Authors@R")
```

I thank my PhD advisors at Iowa State University, Di Cook and Heike Hofmann, for their open-mindedness and consistent support for my research in this "non-classical" area of statistics. I also thank RStudio (<http://www.rstudio.com>) for providing me the freedom to work on the second edition of this book.

Lastly, I thank the reviewers Frank Harrell, Douglas Bates, Carl Boettiger, Joshua Wiley, Scott Kostyshak, and Jim Robison-Cox for their valuable advice on improving the quality of this book (which is the first book of my career), and I'm grateful to my editor John Kimmel, without whom I would not have been able to publish my first book quickly.

Yihui Xie
Ames, Iowa

About the Author

Yihui Xie (<http://yihui.name>) is currently a software engineer at RStudio (<http://www.rstudio.com>). He earned his PhD from the Department of Statistics, Iowa State University. He is interested in interactive statistical graphics and statistical computing. As an active R user, he has authored several R packages, such as **animation**, **knitr**, **formatR**, **fun**, **mime**, **highr**, **servr**, and **Rd2roxygen**, among which the **animation** package won the 2009 John M. Chambers Statistical Software Award (ASA), and the **knitr** package was awarded the “Honorable Mention” prize in the “Applications of R in Business Contest 2012” thanks to Revolution Analytics.

In 2006, he founded the “Capital of Statistics” (<http://cos.name>), which has grown into a large online community on statistics in China. He initiated the first Chinese R conference in 2008, and has been organizing R conferences in China since then. During his PhD training at Iowa State University, he won the Vince Sposito Statistical Computing Award (2011) and the Snedecor Award (2012) in the Department of Statistics.

List of Figures

1.1	A simulation of Brownian motion	2
3.1	The source of a minimal Rnw document	13
3.2	A minimal example in L ^A T _E X	14
3.3	The source of a minimal Rmd document	15
3.4	A minimal example in Markdown	16
4.1	Edit an Rnw document in RStudio	20
4.2	Edit an Rmd document in RStudio	22
4.3	Using knitr in L _Y X	24
5.1	The Sweave style in knitr	41
5.2	The listings style in knitr	42
7.1	A plot created in ggplot2 that does not need to be printed explicitly	59
7.2	A plot using the Bookman font family	62
7.3	A table of the Windows-1250 code page	64
7.4	Three expressions produced two plots	66
7.5	All high-level plots are captured	67
7.6	Show plots right below the code	68
7.7	Only the last plot was kept	69
7.8	A clock animation	70
7.9	A right-aligned plot adapted from ?stars	72
7.10	Rotate two plots with different angles	74
7.11	The traditional approach to writing math expressions in plots	75
7.12	Write math in native L ^A T _E X with the <i>tikz()</i> device	75
7.13	A figure environment with sub-figures	77
10.1	A plot with the default margin	100
10.2	A plot with a smaller margin	101
10.3	The original plot with a large white margin	104
10.4	The cropped plot	105
10.5	An rgl plot captured by <i>hook_rgl()</i>	106

10.6	A plot created by GGobi	107
10.7	Adding elements to an existing rgl plot	109
11.1	A diagram drawn with TikZ	121
11.2	A diagram drawn with dot in Graphviz	122
12.1	A table created by the gridExtra package	129
12.2	Break long lines with listings	134
12.3	A simple example of using knitr in beamer slides	135
12.4	A sample page of beamer slides	136
12.5	R code chunks in the R Example environments	141
12.6	The Docco style for HTML output	142
12.7	The source document of the ggplot2 geom examples	148
12.8	A sample page of the ggplot2 documentation	149
12.9	The flowchart demo in the diagram package	152
12.10	A sample page of the flowchart demo	153
12.11	A template of regression models	157
13.1	OpenDocument Text converted from Markdown	164
13.2	The source of an example of HTML5 slides	165
14.1	A preview of the HTML output document from R Mark- down v2	178
14.2	A preview of the table, footnotes, and citations	179
14.3	A preview of the “readable” theme, with a table of con- tents and numbered sections	181
14.4	A preview of the PDF output document from R Mark- down v2	185
14.5	A preview of the PDF output document, with a table of contents and numbered sections	186
14.6	A preview of the Microsoft Word document from R Mark- down v2	189
14.7	Open the styles panel in Word	190
14.8	Modify styles of elements in Word	191
14.9	The title slide of an ioslides presentation	192
14.10	One slide from a Slidy presentation	194
14.11	Two slides from the Beamer presentation created by R Markdown	196
14.12	An example page using the Tufte handout style	200
14.13	A simple interactive document using R Markdown and Shiny	201
14.14	Create a new R Markdown document from templates	206
14.15	Create an E-book from R Markdown	207

14.16	A table created by the DataTables library in R Markdown	210
15.1	Trace of Gibbs sampling for a bivariate Normal distribution	215
15.2	5000 points from Gibbs sampling	215
15.3	The layout of an R Markdown document and its output in the RStudio Viewer	218
15.4	A Makefile example for the function <i>make()</i> in servr . . .	219
15.5	The metadata of a knitr vignette	223
15.6	A sample page of the ggplot2 transition guide	225
15.7	The Makefile to compile PDF vignettes using knitr . . .	226
15.8	The Makefile to compile HTML vignettes	227
16.1	A screenshot of IPython	243

List of Tables

1.1	A subset of the <code>mtcars</code> dataset	4
5.1	A syntax summary of all document formats	32
5.2	Output hook functions and the object classes of results from the evaluate package.	40
11.1	Interpreted languages supported by knitr	117

1

Introduction

The basic idea behind dynamic documents stems from *literate programming*, a programming paradigm conceived by Donald Knuth (Knuth, 1984). The original idea was mainly for writing software: mix the source code and documentation together; we can either extract the source code out (called *tangle*) or execute the code to get the compiled results (called *weave*). A dynamic document is not entirely different from a computer program: for a dynamic document, we need to run software packages to compile our ideas (often implemented as source code) into numeric or graphical output, and insert the output into our literal writings (like documentation).

We explain the idea with a trivial example: suppose we need to write the value of 2π into a report; of course, we can directly write the number 6.2832. Now, if I change my mind and I want 6π instead, I may have to find a calculator, erase the previous value, and write the new answer. Since it is extremely easy for the computer to calculate 6π , why not leave this job to the computer completely and free oneself from this kind of manual work? What we need to do is to leave the source code in the document instead of a hard-coded value, and tell the computer how to find and execute the source code. Usually we use special markers for computer code in the source report; e.g., we can write

The correct answer is `{{ 6 * pi }}`.

in which `{{` and `}}` is a pair of markers that tell the computer `6 * pi` is the source code and should be executed. Note here `pi` (π) is a constant in R.

If you know a Web scripting language such as PHP (which can embed program code into HTML documents), this idea should look familiar. The above example shows the *inline* code output, which means source code is mixed inline with a sentence. The other type of output is the *chunk* output, which gives the results from a whole block of code. The chunk output has much more flexibility; for example, we can produce graphics and tables from a code chunk.

Figure 1.1 was dynamically created with a chunk of R code, which is printed below:

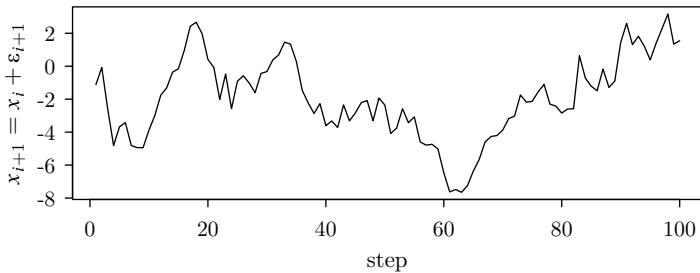


FIGURE 1.1: A simulation of Brownian motion for 100 steps: $x_1 = \epsilon_1$, $x_{i+1} = x_i + \epsilon_{i+1}$, $\epsilon_i \stackrel{iid}{\sim} N(0,1)$, $i = 1, 2, \dots, 100$

```
set.seed(1213) # for reproducibility of random numbers
x <- cumsum(rnorm(100))
plot(x, type = "l", ylab = "$x_{i+1}=x_i+\\epsilon_{i+1}$",
     xlab = "step")
```

If we were to do this by hand, we would have to open R, paste the code into the R console to draw the plot, save it as a PDF file, and insert it into a \LaTeX document with `\includegraphics{}`. This is both tedious for the author and difficult to maintain — supposing we want to change the random seed in `set.seed()`, increase the number of steps, or use a scatterplot instead of a line graph, we will have to update both the source code and the output. In practice, the computing and analysis can be far more complicated than the toy example in Figure 1.1, and more manual work will be required accordingly.

The spirit of dynamic documents may best be described by the philosophy of the ESS project (Rossini et al., 2004) for the S language:

The source code is real.

Philosophy for using ESS[S]

Since the output can be produced by the source code, we can maintain the source code only. However, in most cases, the direct output from the source code alone does not constitute a report that is readable

for a human. That is why we need the literate programming paradigm. In this paradigm, an author has two tasks:

1. write program code to do computing, and
2. write narratives to explain what is being done by the program code

The traditional approach to doing the second task is to write comments for the code, but comments are often limited in terms of expressing the full thoughts of the authors. Normally we write our ideas in a paper or a report instead of hundreds of lines of code comments.

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.

Donald E. Knuth
Literate Programming, 1984

Technically, literate programming involves three steps:

1. parse the source document and separate code from narratives
2. execute source code and return results
3. mix results from the source code with the original narratives

These steps can be implemented in software packages, so the authors do not need to take care of these technical details. Instead, we only control what the output should look like. There are many details that we can tune for a report (especially for reports related to data analysis), although the idea of literate programming seems to be simple. For example, data reports often include tables, and Table 1.1 is a table generated from the R code below using the *kable()* function in **knitr**:

```
library(knitr)
kable(head(mtcars[, 1:6]))
```

Think how easy it is to maintain two lines of R code compared to maintaining many lines of messy \LaTeX code!

Generating reports dynamically by integrating computer code with

TABLE 1.1: A subset of the `mtcars` dataset: the first 6 rows and 6 columns.

	mpg	cyl	disp	hp	drat	wt
Mazda RX4	21.0	6	160	110	3.90	2.620
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875
Datsun 710	22.8	4	108	93	3.85	2.320
Hornet 4 Drive	21.4	6	258	110	3.08	3.215
Hornet Sportabout	18.7	8	360	175	3.15	3.440
Valiant	18.1	6	225	105	2.76	3.460

narratives is not only easier, but also closely related to reproducible research, which we will discuss in the next chapter.

2

Reproducible Research

Results from scientific research have to be reproducible to be trustworthy. We do not want a finding to be merely due to an isolated occurrence, e.g., only one specific laboratory researcher can produce the results on one specific day, and nobody else can produce the same results under the same conditions.

Reproducible research (RR) is one possible by-product of dynamic documents, but dynamic documents do not absolutely guarantee RR. Because there is usually no human intervention when we generate a report dynamically, it is likely to be reproducible since it is relatively easy to prepare the same software and hardware environment, which is everything we need to reproduce the results. However, the meaning of reproducibility can be beyond reproducing one specific result or one particular report. As a trivial example, one might have done a Monte Carlo simulation with a certain random seed and got a good estimate of a parameter, but the result was actually due to a “lucky” random seed. Although we can strictly reproduce the estimate, it is not actually reproducible in the general sense. Similar problems exist in optimization algorithms, e.g., different starting values can lead to different roots of the same equation.

Anyway, dynamic report generation is still an important step toward RR. In this chapter, we discuss a selection of the RR literature and practices of RR.

2.1 Literature

The term reproducible research was first proposed by Jon Claerbout at Stanford University (Fomel and Claerbout, 2009). The idea is that the final product of research is not only the paper itself, but also the full computational environment used to produce the results in the paper such as the code and data necessary for reproduction of the results and building upon the research.

Similarly, Buckheit and Donoho (1995) pointed out the essence of the scholarship of an article as follows:

An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.

D. Donoho
WaveLab and Reproducible Research

That was well said! Fortunately, journals have been moving in that direction as well. For example, Peng (2009) provided detailed instructions to authors on the criteria of reproducibility and how to submit materials for reproducing the paper in the *Biostatistics* journal.

At the technical level, RR is often related to literate programming (Knuth, 1984), a paradigm conceived by Donald Knuth to integrate computer code with software documentation in one document. However, early implementations like WEB (Knuth, 1983) and Noweb (Ramsey, 1994) were not directly suitable for data analysis and report generation. There are other tools on this path of documentation generation, such as **roxygen2** (Wickham et al., 2015), which is an R implementation of Doxygen (van Heesch, 2008). Sweave (Leisch, 2002) was among the first implementations for dealing with dynamic documents in R (Ihaka and Gentleman, 1996; R Core Team, 2015). There are still a number of challenges that were not solved by the existing tools; for example, Sweave is closely tied to \LaTeX and hard to extend. The **knitr** package (Xie, 2015b) was built upon the ideas of previous tools with a framework redesign, enabling easy and fine control of many aspects of a report. We will introduce other tools in Chapter 16.

An overview of literate programming applied to statistical analysis can be found in Rossini (2002). Gentleman and Temple Lang (2004) introduced general concepts of literate programming documents for statistical analysis, with a discussion of the software architecture. Gentleman (2005) is a practical example based on Gentleman and Temple Lang (2004), using an R package **GolubRR** to distribute reproducible analysis. Baggerly et al. (2004) revealed several problems that may arise with the standard practice of publishing data analysis results, which can lead to false discoveries due to lack of details for reproducibility

(even with datasets supplied). Instead of separating results from computing, we can put everything in one document (called a *compendium* in Gentleman and Temple Lang (2004)), including the computer code and narratives. When we compile this document, the computer code will be executed, giving us the results directly.

2.2 Good and Bad Practices

The key to keep in mind for RR is that other people should be able to reproduce our results, therefore we should try our best to make our computation *portable*. We discuss some good practices for RR below and explain why it can be bad not to follow them.

- Manage all source files under the same directory and use relative paths whenever possible: absolute paths can break reproducibility, e.g., a data file like `C:/Users/john/foo.csv` or `/home/joe/foo.csv` may only exist in one computer, and other people may not be able to read it since the absolute path is likely to be different in their hard disk. If we keep everything under the same directory, we can read a data file with `read.csv('foo.csv')` (if it is under the current working directory) or `read.csv('../data/foo.csv')` (go one level up and find the file under the `data/` directory); when we disseminate the results, we can make an archive of the whole directory (e.g., as a zip package).
- Do not change the working directory after the computing has started: `setwd()` is the function in R to set the working directory, and it is not uncommon to see `setwd('C:/path/to/some/dir')` in user's code, which is bad because it is not only an absolute path, but also has a global effect on the rest of the source document. In that case, we have to keep in mind that all relative paths may need adjustments since the root directory has changed, and the software may write the output in an unexpected place (e.g., the figures are expected to be generated in the `./figures/` directory, but are actually written to `./data/figures/` instead if we `setwd('../data/')`). If we have to set the working directory at all, do it in the very beginning of an R session; most of the editors to be introduced in Chapter 4 follow this rule, and the working directory is set to the directory of the source document before **knitr** is called to compile documents. If it is unavoidable or makes it much more convenient for you to write code after setting a different working directory, you should restore the directory later; e.g.,

```
f <- function(...) {  
  # stores current dir to a variable owd  
  owd <- setwd("a/different/dir/")  
  # restore working dir when the function exits  
  on.exit(setwd(owd), add = TRUE)  
  # now you can work under a/different/dir  
  ...  
}
```

- Compile the documents in a clean R session: existing R objects in the current R session may “contaminate” the results in the output. It is fine if we write a report by accumulating code chunks one by one and running them interactively to check the results, but in the end we should compile a report in the batch mode with a new R session so all the results are freshly generated from the code.
- Avoid the commands that require human interaction: human input can be highly unpredictable; e.g., we do not know for sure which file the user will choose if we pop up a dialog box asking the user to choose a data file. Instead of using functions like *file.choose()* to input a file to *read.table()*, we should write the filename explicitly; e.g., *read.table('a-specific-file.txt')*.
- Avoid environment variables for data analysis: while environment variables are often heavily used in programming for configuration purposes, it is ill-advised to use them in data analysis because they require additional instructions for users to set up, and humans can simply forget to do this. If there are any options to set up, do it inside the source document.
- Attach *sessionInfo()* (or **devtools::session_info()**) and instructions on how to compile this document: the session information makes a reader aware of the software environment, such as the version of R, the operating system, and add-on packages used. Sometimes it is not as simple as calling one single function to compile a document, and we have to make it clear how to compile it if additional steps are required; but it is better to provide the instructions in the form of a computer script; e.g., a shell script, a Makefile, or a batch file.

These practices are not necessarily restricted to the R language, although we used R for examples. The same rules also apply to other computing environments.

Note that literate programming tools often require users to compile the documents in batch mode, and it is good for reproducible research,

but the batch mode can be cumbersome for exploratory data analysis. When we have not decided what to put in the final document, we may need to interact with the data and code frequently, and it is not worth compiling the whole document each time we update the code. This problem can be solved by a capable editor such as RStudio and Emacs/ESS, which are introduced in Chapter 4. In these editors, we can interact with the code and explore the data freely (e.g., send or write R code in an associated R session), and once we finish the coding work, we can compile the whole document in the batch mode to make sure all the code works in a clean R session.

2.3 Barriers

Despite all the advantages of RR, there are some practical barriers, and here is a non-exhaustive list:

- the data can be huge: for example, it is common that high energy physics and next-generation sequencing data in biology can produce tens of terabytes of data, and it is not trivial to archive the data with the reports and distribute them
- confidentiality of data: it may be prohibited to release the raw data with the report, especially when it is involved with human subjects due to the confidentiality issues
- software version and configuration: a report may be generated with an old version of a software package that is no longer available, or with a software package that compiles differently on different operating systems
- competition: one may choose not to release the code or data with the report due to the fact that potential competitors can easily get everything for free, whereas the original authors have invested a large amount of money and effort

We certainly should not expect all reports in the world to be publicly available and strictly reproducible, but it is better to share even mediocre or flawed code or problematic datasets than not to share anything at all. Instead of persuading people into RR by policies, we may try to create tools that make RR easier than cut-and-paste, and **knitr** is such an attempt. The success of RPubs (<http://rpubs.com>) is evidence that an

easy tool can quickly promote RR, because users enjoy using it. Readers can find hundreds of reports contributed by users in the RPubS website. It is fairly common to see student homework and exercises there, and once the students are trained in this manner, we may expect more reproducible scientific research in the future.

3

A First Look

The **knitr** package is a general-purpose literate programming engine — it supports document formats including L^AT_EX, HTML, and Markdown (see Chapter 5), and programming languages such as R, Python, awk, C++, and shell scripts (Chapter 11). Before we get started, we need to install **knitr** in R. Then we will introduce the basic concepts with minimal examples. Finally, we will show how to generate reports quickly from pure R scripts, which can be useful for beginners who do not know anything about dynamic documents.

3.1 Setup

Since **knitr** is an R package, it can be installed from CRAN in the usual way in R:

```
install.packages("knitr", dependencies = TRUE)
```

Note here that `dependencies = TRUE` is optional, and will install all packages that are not absolutely necessary but can enhance this package with some useful features. The development version is hosted on Github: <https://github.com/yihui/knitr>, and you can always check out the latest development version, which may not be stable but contains the latest bug fixes and new features. If you have any problems with **knitr**, the first thing to check is its version:

```
packageVersion("knitr")  
# if not the latest version, run  
update.packages()
```

If you choose L^AT_EX as the typesetting tool, you may need to install MiK_TE_X (Windows, <http://miktex.org/>), Mac_TE_X (Mac OS, <http://tug.org/mactex/>), or T_EXLive (Linux, <http://tug.org/texlive/>). If

you are going to work with HTML or Markdown, nothing else needs to be installed, since the output will be Web pages, which you can view with a Web browser.

Once we have **knitr** installed, we can compile source documents using the function `knit()`, e.g.,

```
library(knitr)
knit("your-file.Rnw")
```

A *.Rnw file is usually a \LaTeX document with R code embedded in it, as we will see in the following section and Chapter 5, in which more types of documents will be introduced.

3.2 Minimal Examples

We use two minimal examples written in \LaTeX and Markdown, respectively, to illustrate the structure of dynamic documents. We do not discuss the syntax of \LaTeX or Markdown for the time being (see Chapter 5 instead). For the sake of simplicity, the `cars` dataset in base R is used to build a simple linear regression model. Type `?cars` in R to see detailed documentation. Basically it has two variables, speed and distance:

```
str(cars)

## 'data.frame':    50 obs. of  2 variables:
## $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
## $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

3.2.1 An Example in \LaTeX

Figure 3.1 is a full example of R code embedded in \LaTeX ; we call this kind of documents *Rnw documents* hereafter because their filename extension is Rnw by convention. If we save it as a file `minimal.Rnw` and run `knit('minimal.Rnw')` in R as described before, **knitr** will generate a \LaTeX output document named `minimal.tex`. For those who are familiar with \LaTeX , you can compile this document to PDF via `pdflatex`. Figure 3.2 is the PDF document compiled from the Rnw document.

What is essential here is how we embedded R code in \LaTeX . In an Rnw document, `<<>>=` marks the beginning of code chunks, and `@` terminates a code chunk (this description is not rigorous but is often easier

```
\documentclass{article}
\begin{document}
\title{A Minimal Example}
\author{Yihui Xie}
\maketitle

We examine the relationship between speed and stopping
distance using a linear regression model:

$$Y = \beta_0 + \beta_1 x + \epsilon$$


<<model, fig.width=4, fig.height=3, fig.align='center'>>=
par(mar = c(4, 4, 1, 1), mgp = c(2, 1, 0), cex = 0.8)
plot(cars, pch = 20, col = 'darkgray')
fit <- lm(dist ~ speed, data = cars)
abline(fit, lwd = 2)
@

The slope of a simple linear regression is
\Sexpr{coef(fit)[2]}.
\end{document}
```

FIGURE 3.1: The source of a minimal Rnw document: see output in Figure 3.2.

to understand); we have four lines of R code between the two markers in this example to draw a scatterplot, fit a linear model, and add a regression line to the scatterplot. The command `\Sexpr{}` is used to embed inline R code, e.g., `coef(fit)[2]` in this example. We can write chunk options for a code chunk between `<<` and `>>=`; the chunk options in this example specified the plot size to be 4 by 3 inches (`fig.width` and `fig.height`), and plots should be aligned in the center (`fig.align`).

In this minimal example, we have most basic elements of a report:

1. title, author, and date
2. model description
3. data and computation
4. graphics
5. numeric results

All the output is generated dynamically from R. Even if the data has

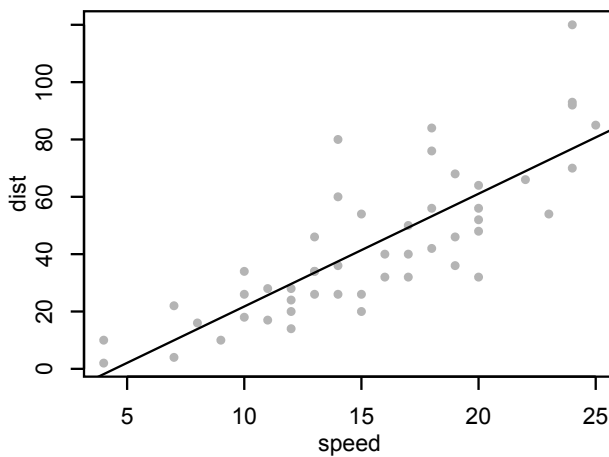
A Minimal Example

Yihui Xie

April 11, 2015

We examine the relationship between speed and stopping distance using a linear regression model: $Y = \beta_0 + \beta_1 x + \epsilon$.

```
par(mar = c(4, 4, 1, 1), mgp = c(2, 1, 0), cex = 0.8)
plot(cars, pch = 20, col = "darkgray")
fit <- lm(dist ~ speed, data = cars)
abline(fit, lwd = 2)
```



The slope of a simple linear regression is 3.9324088.

FIGURE 3.2: A minimal example in \LaTeX with an R code chunk, a plot, and numeric output (regression coefficient).

```

---
title: A Minimal Example
---

We examine the relationship between speed and stopping
distance using a linear regression model:

$$Y = \beta_0 + \beta_1 x + \epsilon$$


```{r fig.width=4, fig.height=3, fig.align='center'}
par(mar = c(4, 4, 1, 1), mgp = c(2, 1, 0), cex = 0.8)
plot(cars, pch = 20, col = 'darkgray')
fit <- lm(dist ~ speed, data = cars)
abline(fit, lwd = 2)
```

The slope of a simple linear regression is
`r coef(fit)[2]`.

```

FIGURE 3.3: The source of a minimal Rmd document: see output in Figure 3.4.

changed, we do not need to redo the report from the ground up, and the output will be updated accordingly if we update the data and recompile the report.

3.2.2 An Example in Markdown

L^AT_EX may look overwhelming to beginners due to the large number of commands. By comparison, Markdown (Gruber, 2004) is a much simpler format. Figure 3.3 is a Markdown example doing the same analysis with the previous example:

The ideal output from Markdown is an HTML Web page, as shown in Figure 3.4 (in Mozilla Firefox). Similarly, we can see the syntax for R code in a Markdown document: ````{r}` opens a code chunk, ````` terminates a chunk, and inline R code can be put inside ``r ``, where ``` is a backtick.

A slightly longer example in **knitr** is a demo named *notebook*, which is based on Markdown. It shows not only the potential power of Markdown, but also the possibility of building Web applications with **knitr**. To watch the demo, run the code below:

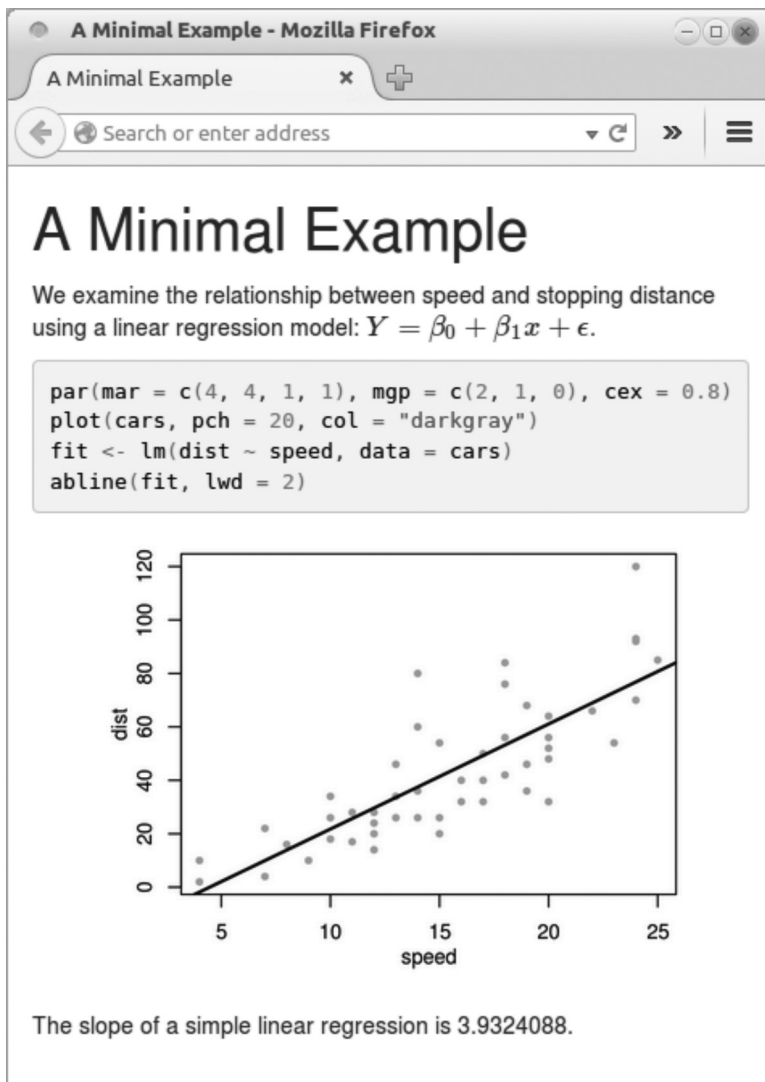


FIGURE 3.4: A minimal example in Markdown with the same analysis as in Figure 3.2, but the output is HTML instead of PDF now.

```
if (!require("shiny")) install.packages("shiny")
demo("notebook", package = "knitr")
```

Your default Web browser will be launched to show a Web notebook. The source code is in the left panel, and the live results are in the right panel. You are free to experiment with the source code and recompile the notebook.

3.3 Quick Reporting

If a user only has basic knowledge of R but knows nothing about **knitr**, or one does not want to write anything other than an R script, it is also possible to generate a quick report from this R script using the *stitch()* function.

The basic idea of *stitch()* is that **knitr** provides a template of the source document with some default settings, so that the user only needs to feed this template with an R script (as one code chunk); then **knitr** will compile the template to a report. Currently it has built-in templates for L^AT_EX, HTML, and Markdown. The usage is like this:

```
library(knitr)
stitch("your-script.R")
```

3.4 Extracting R Code

For a literate programming document, we can either compile it to a report (run the code), or extract the program code in it. They are called “weaving” and “tangling,” respectively. Apparently the function *knit()* is for weaving, and the corresponding tangling function is *purl()* in **knitr**. For example,

```
library(knitr)
purl("your-file.Rnw")
purl("your-file.Rmd")
```

The result of tangling is an R script; in the above examples, the default output will be `your-file.R`, which consists of all code chunks in the source document.

So far we have been introducing the command line usage of **knitr**, and it is often tedious to type the commands repeatedly. In the next chapter, we show how a decent editor can help edit and compile the source document with one single mouse click or a keyboard shortcut.