Nina Zumel

John Mount

FOREWORD BY Jim Porzak MANNING

www.it-ebooks.info

*Practical Data Science with R* www.it-ebooks.info

www.it-ebooks.info

*Practical Data Science with R*

NINA ZUMEL

JOHN MOUNT

MANNING

SHELTER ISLAND

www.it-ebooks.info

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact

Special Sales Department

Manning Publications Co.

20 Baldwin Road

PO Box 261

Shelter Island, NY 11964

Email: orders@manning.com

©2014 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Recognizing the importance of preserving what has been written, it is Manning’s policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

Manning Publications Co. Development editor: Cynthia Kane 20 Baldwin Road Copyeditor: Benjamin Berg PO Box 261 Proofreader: Katie Tennant Shelter Island, NY 11964 Typesetter: Dottie Marsico Cover designer: Marija Tudor

ISBN 9781617291562

Printed in the United States of America

1 2 3 4 5 6 7 8 9 10 – EBM – 19 18 17 16 15 14

www.it-ebooks.info

*To our parents*

*Olive and Paul Zumel Peggy and David Mount*

www.it-ebooks.info

www.it-ebooks.info

*brief contents*

**PART 1 INTRODUCTION TO DATA SCIENCE .................................1**

1 ■ The data science process 3

2 ■ Loading data into R 18

3 ■ Exploring data 35

4 ■ Managing data 64

**PART 2 MODELING METHODS ..................................................81**

5 ■ Choosing and evaluating models 83

6 ■ Memorization methods 115

7 ■ Linear and logistic regression 140

8 ■ Unsupervised methods 175

9 ■ Exploring advanced methods 211

**PART 3 DELIVERING RESULTS ................................................253**

10 ■ Documentation and deployment 255

11 ■ Producing effective presentations 287

**vii**

www.it-ebooks.info

www.it-ebooks.info

*contents*

*foreword xv*

*preface xvii*

*acknowledgments xviii*

*about this book xix*

*about the cover illustration xxv*

**PART 1 INTRODUCTION TO DATA SCIENCE......................1** *1* ***The data science process 3***

1.1 The roles in a data science project 3

*Project roles 4*

1.2 Stages of a data science project 6

*Defining the goal 7* ■ *Data collection and management 8*

*Modeling 10* ■ *Model evaluation and critique 11*

*Presentation and documentation 13* ■ *Model deployment and*

*maintenance 14*

1.3 Setting expectations 14

*Determining lower and upper bounds on model performance 15*

1.4 Summary 17

**ix**

www.it-ebooks.info

**x** CONTENTS

*2* ***Loading data into R 18***

2.1 Working with data from files 19

*Working with well-structured data from files or URLs 19*

*Using R on less-structured data 22*

2.2 Working with relational databases 24

*A production-size example 25* ■ *Loading data from a database*

*into R 30* ■ *Working with the PUMS data 31*

2.3 Summary 34

*3* ***Exploring data 35***

3.1 Using summary statistics to spot problems 36

*Typical problems revealed by data summaries 38*

3.2 Spotting problems using graphics and visualization 41

*Visually checking distributions for a single variable 43*

*Visually checking relationships between two variables 51*

3.3 Summary 62

*4* ***Managing data 64***

4.1 Cleaning data 64

*Treating missing values (NAs) 65* ■ *Data transformations 69*

4.2 Sampling for modeling and validation 76

*Test and training splits 76* ■ *Creating a sample group*

*column 77* ■ *Record grouping 78* ■ *Data provenance 78*

4.3 Summary 79

**PART 2 MODELING METHODS ......................................81** *5* ***Choosing and evaluating models 83***

5.1 Mapping problems to machine learning tasks 84

*Solving classification problems 85* ■ *Solving scoring*

*problems 87* ■ *Working without known targets 88*

*Problem-to-method mapping 90*

5.2 Evaluating models 92

*Evaluating classification models 93* ■ *Evaluating scoring*

*models 98* ■ *Evaluating probability models 101* ■ *Evaluating*

*ranking models 105* ■ *Evaluating clustering models 105*

www.it-ebooks.info

CONTENTS **xi**

5.3 Validating models 108

*Identifying common model problems 108* ■ *Quantifying model*

*soundness 110* ■ *Ensuring model quality 111*

5.4 Summary 113

*6* ***Memorization methods 115***

6.1 KDD and KDD Cup 2009 116

*Getting started with KDD Cup 2009 data 117*

6.2 Building single-variable models 118

*Using categorical features 119* ■ *Using numeric features 121*

*Using cross-validation to estimate effects of overfitting 123*

6.3 Building models using many variables 125

*Variable selection 125* ■ *Using decision trees 127* ■ *Using*

*nearest neighbor methods 130* ■ *Using Naive Bayes 134*

6.4 Summary 138

*7* ***Linear and logistic regression 140***

7.1 Using linear regression 141

*Understanding linear regression 141* ■ *Building a linear*

*regression model 144* ■ *Making predictions 145* ■ *Finding*

*relations and extracting advice 149* ■ *Reading the model summary*

*and characterizing coefficient quality 151* ■ *Linear regression*

*takeaways 156*

7.2 Using logistic regression 157

*Understanding logistic regression 157* ■ *Building a logistic*

*regression model 159* ■ *Making predictions 160* ■ *Finding*

*relations and extracting advice from logistic models 164*

*Reading the model summary and characterizing coefficients 166*

*Logistic regression takeaways 173*

7.3 Summary 174

*8* ***Unsupervised methods 175***

8.1 Cluster analysis 176

*Distances 176* ■ *Preparing the data 178* ■ *Hierarchical*

*clustering with hclust() 180* ■ *The k-means algorithm 190*

*Assigning new points to clusters 195* ■ *Clustering*

*takeaways 198*

www.it-ebooks.info

**xii** CONTENTS

8.2 Association rules 198

*Overview of association rules 199* ■ *The example problem 200*

*Mining association rules with the arules package 201*

*Association rule takeaways 209*

8.3 Summary 209

*9* ***Exploring advanced methods 211***

9.1 Using bagging and random forests

to reduce training variance 212

*Using bagging to improve prediction 213* ■ *Using random forests*

*to further improve prediction 216* ■ *Bagging and random forest*

*takeaways 220*

9.2 Using generalized additive models (GAMs) to learn non

monotone relationships 221

*Understanding GAMs 221* ■ *A one-dimensional regression*

*example 222* ■ *Extracting the nonlinear relationships 226*

*Using GAM on actual data 228* ■ *Using GAM for logistic*

*regression 231* ■ *GAM takeaways 233*

9.3 Using kernel methods to increase data separation 233

*Understanding kernel functions 234* ■ *Using an explicit kernel on*

*a problem 238* ■ *Kernel takeaways 241*

9.4 Using SVMs to model complicated decision

boundaries 242

*Understanding support vector machines 242* ■ *Trying an SVM on*

*artificial example data 245* ■ *Using SVMs on real data 248*

*Support vector machine takeaways 251*

9.5 Summary 251

**PART 3 DELIVERING RESULTS ....................................253** *10* ***Documentation and deployment 255***

10.1 The buzz dataset 256

10.2 Using knitr to produce milestone documentation 258

*What is knitr? 258* ■ *knitr technical details 261* ■ *Using knitr*

*to document the buzz data 262*

www.it-ebooks.info

CONTENTS **xiii**

10.3 Using comments and version control for running

documentation 266

*Writing effective comments 266* ■ *Using version control to record*

*history 267* ■ *Using version control to explore your project 272*

*Using version control to share work 276*

10.4 Deploying models 280

*Deploying models as R HTTP services 280* ■ *Deploying models by*

*export 283* ■ *What to take away 284*

10.5 Summary 286

*11* ***Producing effective presentations 287***

11.1 Presenting your results to the project sponsor 288

*Summarizing the project’s goals 289* ■ *Stating the project’s*

*results 290* ■ *Filling in the details 292* ■ *Making*

*recommendations and discussing future work 294*

*Project sponsor presentation takeaways 295*

11.2 Presenting your model to end users 295

*Summarizing the project’s goals 296* ■ *Showing how the model fits*

*the users’ workflow 296* ■ *Showing how to use the model 299*

*End user presentation takeaways 300*

11.3 Presenting your work to other data scientists 301

*Introducing the problem 301* ■ *Discussing related work 302*

*Discussing your approach 302* ■ *Discussing results and future*

*work 303* ■ *Peer presentation takeaways 304*

11.4 Summary 304

*appendix A Working with R and other tools 307*

*appendix B Important statistical concepts 333*

*appendix C More tools and ideas worth exploring 369*

*bibliography 375*

*index 377*

www.it-ebooks.info

www.it-ebooks.info

*foreword*

If you’re a beginning data scientist, or want to be one, *Practical Data Science with R (PDSwR)* is the place to start. If you’re already doing data science, *PDSwR* will fill in gaps in your knowledge and even give you a fresh look at tools you use on a daily basis—it did for me.

While there are many excellent books on statistics and modeling with R, and a few good management books on applying data science in your organization, this book is unique in that it combines solid technical content with practical, down-to-earth advice on how to practice the craft. I would expect no less from Nina and John.

I first met John when he presented at an early Bay Area R Users Group about his joys and frustrations with R. Since then, Nina, John, and I have collaborated on a cou ple of projects for my former employer. And John has presented early ideas from *PDSwR*—both to the “big” group and our Berkeley R-Beginners meetup. Based on his experience as a practicing data scientist, John is outspoken and has strong views about how to do things. *PDSwR* reflects Nina and John’s definite views on how to do data sci ence—what tools to use, the process to follow, the important methods, and the impor tance of interpersonal communications. There are no ambiguities in *PDSwR*.

This, as far as I’m concerned, is perfectly fine, especially since I agree with 98% of their views. (My only quibble is around SQL—but that’s more an issue of my upbring ing than of disagreement.) What their unambiguous writing means is that you can focus on the craft and art of data science and not be distracted by choices of which tools and methods to use. This precision is what makes *PDSwR* practical. Let’s look at some specifics.

Practical tool set: R is a given. In addition, RStudio is the IDE of choice; I’ve been using RStudio since it came out. It has evolved into a remarkable tool—integrated

**xv**

www.it-ebooks.info

**xvi** FOREWORD

debugging is in the latest version. The third major tool choice in *PDSwR* is Hadley Wickham’s ggplot2. While R has traditionally included excellent graphics and visual ization tools, ggplot2 takes R visualization to the next level. (My practical hint: take a close look at any of Hadley’s R packages, or those of his students.) In addition to those main tools, *PDSwR* introduces necessary secondary tools: a proper SQL DBMS for larger datasets; Git and GitHub for source code version control; and knitr for docu mentation generation.

Practical datasets: The only way to learn data science is by doing it. There’s a big leap from the typical teaching datasets to the real world. *PDSwR* strikes a good balance between the need for a practical (simple) dataset for learning and the messiness of the real world. *PDSwR* walks you through how to explore a new dataset to find prob lems in the data, cleaning and transforming when necessary.

Practical human relations: Data science is all about solving real-world problems for your client—either as a consultant or within your organization. In either case, you’ll work with a multifaceted group of people, each with their own motivations, skills, and responsibilities. As practicing consultants, Nina and John understand this well. *PDSwR* is unique in stressing the importance of understanding these roles while working through your data science project.

Practical modeling: The bulk of *PDSwR* is about modeling, starting with an excel lent overview of the modeling process, including how to pick the modeling method to use and, when done, gauge the model’s quality. The book walks you through the most practical modeling methods you’re likely to need. The theory behind each method is intuitively explained. A specific example is worked through—the code and data are available on the authors’ GitHub site. Most importantly, tricks and traps are covered. Each section ends with practical takeaways.

In short, *Practical Data Science with R* is a unique and important addition to any data scientist’s library.

JIM PORZAK

SENIOR DATA SCIENTIST AND

COFOUNDER OF THE BAY AREA R USERS GROUP

www.it-ebooks.info

*preface*

This is the book we wish we’d had when we were teaching ourselves that collection of subjects and skills that has come to be referred to as *data science*. It’s the book that we’d like to hand out to our clients and peers. Its purpose is to explain the relevant parts of statistics, computer science, and machine learning that are crucial to data science.

Data science draws on tools from the empirical sciences, statistics, reporting, ana lytics, visualization, business intelligence, expert systems, machine learning, databases, data warehousing, data mining, and big data. It’s because we have so many tools that we need a discipline that covers them all. What distinguishes data science itself from the tools and techniques is the central goal of deploying effective decision-making models to a production environment.

Our goal is to present data science from a pragmatic, practice-oriented viewpoint. We’ve tried to achieve this by concentrating on fully worked exercises on real data— altogether, this book works through over 10 significant datasets. We feel that this approach allows us to illustrate what we really want to teach and to demonstrate all the preparatory steps necessary to any real-world project.

Throughout our text, we discuss useful statistical and machine learning concepts, include concrete code examples, and explore partnering with and presenting to non specialists. We hope if you don’t find one of these topics novel, that we’re able to shine a light on one or two other topics that you may not have thought about recently.

**xvii**

www.it-ebooks.info

*acknowledgments*

We wish to thank all the many reviewers, colleagues, and others who have read and commented on our early chapter drafts, especially Aaron Colcord, Aaron Schumacher, Ambikesh Jayal, Bryce Darling, Dwight Barry, Fred Rahmanian, Hans Donner, Jeelani Basha, Justin Fister, Dr. Kostas Passadis, Leo Polovets, Marius Butuc, Nathanael Adams, Nezih Yigitbasi, Pablo Vaselli, Peter Rabinovitch, Ravishankar Rajagopalan, Rodrigo Abreu, Romit Singhai, Sampath Chaparala, and Zekai Otles. Their comments, ques tions, and corrections have greatly improved this book. Special thanks to George Gaines for his thorough technical review of the manuscript shortly before it went into production.

We especially would like to thank our development editor, Cynthia Kane, for all her advice and patience as she shepherded us through the writing process. The same thanks go to Benjamin Berg, Katie Tennant, Kevin Sullivan, and all the other editors at Manning who worked hard to smooth out the rough patches and technical glitches in our text.

In addition, we’d like to thank our colleague David Steier, Professors Anno Saxe nian and Doug Tygar from UC Berkeley’s School of Information Science, as well as all the other faculty and instructors who have reached out to us about the possibility of using this book as a teaching text.

We’d also like to thank Jim Porzak for inviting one of us (John Mount) to speak at the Bay Area R Users Group, for being an enthusiastic advocate of our book, and for contributing the foreword. On days when we were tired and discouraged and won dered why we had set ourselves to this task, his interest helped remind us that there’s a need for what we’re offering and for the way that we’re offering it. Without his encouragement, completing this book would have been much harder.

**xviii**

www.it-ebooks.info

*about this book*

This book is about data science: a field that uses results from statistics, machine learn ing, and computer science to create predictive models. Because of the broad nature of data science, it’s important to discuss it a bit and to outline the approach we take in this book.

*What is data science?*

The statistician William S. Cleveland defined data science as an interdisciplinary field larger than statistics itself. We define data science as managing the process that can transform hypotheses and data into actionable predictions. Typical predictive analytic goals include predicting who will win an election, what products will sell well together, which loans will default, or which advertisements will be clicked on. The data scientist is responsible for acquiring the data, managing the data, choosing the modeling tech nique, writing the code, and verifying the results.

Because data science draws on so many disciplines, it’s often a “second calling.” Many of the best data scientists we meet started as programmers, statisticians, business intelligence analysts, or scientists. By adding a few more techniques to their reper toire, they became excellent data scientists. That observation drives this book: we introduce the practical skills needed by the data scientist by concretely working through all of the common project steps on real data. Some steps you’ll know better than we do, some you’ll pick up quickly, and some you may need to research further.

Much of the theoretical basis of data science comes from statistics. But data science as we know it is strongly influenced by technology and software engineering method ologies, and has largely evolved in groups that are driven by computer science and

**xix**

www.it-ebooks.info

**xx** ABOUT THIS BOOK

information technology. We can call out some of the engineering flavor of data sci ence by listing some famous examples:

◼ Amazon’s product recommendation systems

◼ Google’s advertisement valuation systems

◼ LinkedIn’s contact recommendation system

◼ Twitter’s trending topics

◼ Walmart’s consumer demand projection systems

These systems share a lot of features:

◼ All of these systems are built off large datasets. That’s not to say they’re all in the realm of big data. But none of them could’ve been successful if they’d only used small datasets. To manage the data, these systems require concepts from com puter science: database theory, parallel programming theory, streaming data techniques, and data warehousing.

◼ Most of these systems are online or live. Rather than producing a single report or analysis, the data science team deploys a decision procedure or scoring pro cedure to either directly make decisions or directly show results to a large num ber of end users. The production deployment is the last chance to get things right, as the data scientist can’t always be around to explain defects.

◼ All of these systems are allowed to make mistakes at some non-negotiable rate. ◼ None of these systems are concerned with cause. They’re successful when they find useful correlations and are not held to correctly sorting cause from effect.

This book teaches the principles and tools needed to build systems like these. We teach the common tasks, steps, and tools used to successfully deliver such projects. Our emphasis is on the whole process—project management, working with others, and presenting results to nonspecialists.

*Roadmap*

This book covers the following:

◼ Managing the data science process itself. The data scientist must have the ability to measure and track their own project.

◼ Applying many of the most powerful statistical and machine learning tech niques used in data science projects. Think of this book as a series of explicitly worked exercises in using the programming language R to perform actual data science work.

◼ Preparing presentations for the various stakeholders: management, users, deployment team, and so on. You must be able to explain your work in concrete terms to mixed audiences with words in their common usage, not in whatever technical definition is insisted on in a given field. You can’t get away with just throwing data science project results over the fence.

www.it-ebooks.info

ABOUT THIS BOOK **xxi**

We’ve arranged the book topics in an order that we feel increases understanding. The material is organized as follows.

Part 1 describes the basic goals and techniques of the data science process, empha sizing collaboration and data.

Chapter 1 discusses how to work as a data scientist, and chapter 2 works through loading data into R and shows how to start working with R.

Chapter 3 teaches what to first look for in data and the important steps in charac terizing and understanding data. Data must be prepared for analysis, and data issues will need to be corrected, so chapter 4 demonstrates how to handle those things.

Part 2 moves from characterizing data to building effective predictive models. Chapter 5 supplies a starting dictionary mapping business needs to technical evalua tion and modeling techniques.

Chapter 6 teaches how to build models that rely on memorizing training data. Memorization models are conceptually simple and can be very effective. Chapter 7 moves on to models that have an explicit additive structure. Such functional structure adds the ability to usefully interpolate and extrapolate situations and to identify important variables and effects.

Chapter 8 shows what to do in projects where there is no labeled training data available. Advanced modeling methods that increase prediction performance and fix specific modeling issues are introduced in chapter 9.

Part 3 moves away from modeling and back to process. We show how to deliver results. Chapter 10 demonstrates how to manage, document, and deploy your models. You’ll learn how to create effective presentations for different audiences in chapter 11.

The appendixes include additional technical details about R, statistics, and more tools that are available. Appendix A shows how to install R, get started working, and work with other tools (such as SQL). Appendix B is a refresher on a few key statistical ideas. Appendix C discusses additional tools and research ideas. The bibliography supplies references and opportunities for further study.

The material is organized in terms of goals and tasks, bringing in tools as they’re needed. The topics in each chapter are discussed in the context of a representative project with an associated dataset. You’ll work through 10 substantial projects over the course of this book. All the datasets referred to in this book are at the book’s GitHub repository, https://github.com/WinVector/zmPDSwR. You can download the entire repository as a single zip file (one of GitHub’s services), clone the repository to your machine, or copy individual files as needed.

*Audience*

To work the examples in this book, you’ll need some familiarity with R, statistics, and (for some examples) SQL databases. We recommend you have some good introduc tory texts on hand. You don’t need to be an expert in R, statistics, and SQL before starting the book, but you should be comfortable tutoring yourself on topics that we mention but can’t cover completely in our book.

www.it-ebooks.info

**xxii** ABOUT THIS BOOK

For R, we recommend *R in Action, Second Edition,* by Robert Kabacoff (www. man ning.com/kabacoff2/), along with the text’s associated website, Quick-R (www.stat methods.net). For statistics, we recommend *Statistics, Fourth Edition* by David Freedman, Robert Pisani, and Roger Purves. For SQL, we recommend *SQL for Smarties, Fourth Edition* by Joe Celko.

In general, here’s what we expect from our ideal reader:

◼ *An interest in working examples.* By working through the examples, you’ll learn at least one way to perform all steps of a project. You must be willing to attempt simple scripting and programming to get the full value of this book. For each example we work, you should try variations and expect both some failures (where your variations don’t work) and some successes (where your variations outperform our example analyses).

◼ *Some familiarity with the R statistical system and the will to write short scripts and pro grams in R.* In addition to Kabacoff, we recommend a few good books in the bib liography. We work specific problems in R; to understand what’s going on, you’ll need to run the examples and read additional documentation to under stand variations of the commands we didn’t demonstrate.

◼ *Some experience with basic statistical concepts such as probabilities, means, standard devi ations, and significance.* We introduce these concepts as needed, but you may need to read additional references as we work through examples. We define some terms and refer to some topic references and blogs where appropriate. But we expect you will have to perform some of your own internet searches on certain topics.

◼ *A computer (OS X, Linux, or Windows) to install R and other tools on, as well as internet access to download tools and datasets.* We strongly suggest working through the examples, examining R help() on various methods, and following up some of the additional references.

*What is not in this book?*

This book is not an R manual. We use R to concretely demonstrate the important steps of data science projects. We teach enough R for you to work through the exam ples, but a reader unfamiliar with R will want to refer to appendix A as well as to the many excellent R books and tutorials already available.

This book is not a set of case studies. We emphasize methodology and technique. Example data and code is given only to make sure we’re giving concrete usable advice. This book is not a big data book. We feel most significant data science occurs at a database or file manageable scale (often larger than memory, but still small enough to be easy to manage). Valuable data that maps measured conditions to dependent out comes tends to be expensive to produce, and that tends to bound its size. For some report generation, data mining, and natural language processing, you’ll have to move into the area of big data.

www.it-ebooks.info

ABOUT THIS BOOK **xxiii**

This is not a theoretical book. We don’t emphasize the absolute rigorous theory of any one technique. The goal of data science is to be flexible, have a number of good techniques available, and be willing to research a technique more deeply if it appears to apply to the problem at hand. We prefer R code notation over beautifully typeset equations even in our text, as the R code can be directly used.

This is not a machine learning tinkerer’s book. We emphasize methods that are already implemented in R. For each method, we work through the theory of opera tion and show where the method excels. We usually don’t discuss how to implement them (even when implementation is easy), as that information is readily available.

*Code conventions and downloads*

This book is example driven. We supply prepared example data at the GitHub reposi tory (https://github.com/WinVector/zmPDSwR), with R code and links back to orig inal sources. You can explore this repository online or clone it onto your own machine. We also supply the code to produce all results and almost all graphs found in the book as a zip file (https://github.com/WinVector/zmPDSwR/raw/master/ CodeExamples.zip), since copying code from the zip file can be easier than copying and pasting from the book. You can also download the code from the publisher’s web site at www.manning.com/PracticalDataSciencewithR.

We encourage you to try the example R code as you read the text; even when we discuss fairly abstract aspects of data science, we illustrate examples with concrete data and code. Every chapter includes links to the specific dataset(s) that it references.

In this book, code is set with a fixed-width font like this to distinguish it from regular text. Concrete variables and values are formatted similarly, whereas abstract math will be in *italic font like this*. R is a mathematical language, so many phrases read correctly in either font. In our examples, any prompts such as > and $ are to be ignored. Inline results may be prefixed by R’s comment character #.

*Software and hardware requirements*

To work through our examples, you’ll need some sort of computer (Linux, OS X, or Windows) with software installed (installation described in appendix A). All of the software we recommend is fully cross-platform (Linux, OS X, or Windows), freely avail able, and usually open source.

We suggest installing at least the following:

◼ R itself: http://cran.r-project.org.

◼ Various packages from CRAN (installed by R itself using the install.packages() command and activated using the library() command).

◼ Git for version control: http://git-scm.com.

◼ RStudio for an integrated editor, execution and graphing environment—http:// www.rstudio.com.

◼ A bash shell for system commands. This is built-in for Linux and OS X, and can be added to Windows by installing Cygwin (http://www.cygwin.com). We don’t

www.it-ebooks.info

**xxiv** ABOUT THIS BOOK

write any scripts, so an experienced Windows shell user can skip installing Cyg win if they’re able to translate our bash commands into the appropriate Win dows commands.

*Author Online*

The purchase of *Practical Data Science with R* includes free access to a private web forum run by Manning Publications, where you can make comments about the book, ask technical questions, and receive help from the authors and from other users. To access the forum and subscribe to it, point your web browser to www.manning.com/ PracticalDataSciencewithR. This page provides information on how to get on the forum once you are registered, what kind of help is available, and the rules of conduct on the forum.

Manning’s commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the authors can take place. It is not a commitment to any specific amount of participation on the part of the authors, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking the authors some challenging questions lest their interest stray!

The Author Online forum and the archives of previous discussions will be accessi ble from the publisher’s website as long as the book is in print.

*About the authors*

NINA ZUMEL has worked as a scientist at SRI International, an inde 

pendent, nonprofit research institute. She has worked as chief sci

entist of a price optimization company and founded a contract

research company. Nina is now a principal consultant at Win-Vector

LLC. She can be reached at nzumel@win-vector.com.

JOHN MOUNT has worked as a computational scientist in biotech 

nology and as a stock trading algorithm designer, and has managed

a research team for Shopping.com. He is now a principal con

sultant at Win-Vector LLC. John can be reached at jmount@win

vector.com.

www.it-ebooks.info

*about the cover illustration*

The figure on the cover of *Practical Data Science with R* is captioned “Habit of a Lady of China in 1703.” The illustration is taken from Thomas Jefferys’ *A Collection of the Dresses of Different Nations, Ancient and Modern* (four volumes), London, published between 1757 and 1772. The title page states that these are hand-colored copperplate engravings, heightened with gum arabic. Thomas Jefferys (1719–1771) was called “Geographer to King George III.” He was an English cartographer who was the lead ing map supplier of his day. He engraved and printed maps for government and other official bodies and produced a wide range of commercial maps and atlases, especially of North America. His work as a mapmaker sparked an interest in local dress customs of the lands he surveyed and mapped; they are brilliantly displayed in this four-volume collection.

Fascination with faraway lands and travel for pleasure were relatively new pheno mena in the eighteenth century, and collections such as this one were popular, intro ducing both the tourist as well as the armchair traveler to the inhabitants of other countries. The diversity of the drawings in Jeffreys’ volumes speaks vividly of the uniqueness and individuality of the world’s nations centuries ago. Dress codes have changed, and the diversity by region and country, so rich at that time, has faded away. It is now often hard to tell the inhabitant of one continent from another. Perhaps, try ing to view it optimistically, we have traded a cultural and visual diversity for a more varied personal life—or a more varied and interesting intellectual and technical life.

At a time when it is hard to tell one computer book from another, Manning cele brates the inventiveness and initiative of the computer business with book covers based on the rich diversity of national costumes three centuries ago, brought back to life by Jeffreys’ pictures.

**xxv**

www.it-ebooks.info

www.it-ebooks.info

*Part 1*

*Introduction to data science*

In part 1, we concentrate on the most essential tasks in data science: working with your partners, defining your problem, and examining your data. Chapter 1 covers the lifecycle of a typical data science project. We look at the different roles and responsibilities of project team members, the different stages of a typical project, and how to define goals and set project expectations. This chapter serves as an overview of the material that we cover in the rest of the book and is organized in the same order as the topics that we present. Chapter 2 dives into the details of loading data into R from various external formats and transforming the data into a format suitable for analysis. It also dis cusses the most important R data structure for a data scientist: the data frame. More details about the R programming language are covered in appendix A. Chapters 3 and 4 cover the data exploration and treatment that you should do before proceeding to the modeling stage. In chapter 3, we discuss some of the typical problems and issues that you’ll encounter with your data and how to use summary statistics and visualization to detect those issues. In chapter 4, we discuss data treatments that will help you deal with the problems and issues in your data. We also recommend some habits and procedures that will help you better manage the data throughout the different stages of the project. On completing part 1, you’ll understand how to define a data science proj ect, and you’ll know how to load data into R and prepare it for modeling and analysis.

www.it-ebooks.info

www.it-ebooks.info

*The data science process*

*This chapter covers*

◼ Defining data science project roles

◼ Understanding the stages of a data

science project

◼ Setting expectations for a new data

science project

The data scientist is responsible for guiding a data science project from start to fin ish. Success in a data science project comes not from access to any one exotic tool, but from having quantifiable goals, good methodology, cross-discipline interac tions, and a repeatable workflow.

This chapter walks you through what a typical data science project looks like: the kinds of problems you encounter, the types of goals you should have, the tasks that you’re likely to handle, and what sort of results are expected.

*1.1 The roles in a data science project*

Data science is not performed in a vacuum. It’s a collaborative effort that draws on a number of roles, skills, and tools. Before we talk about the process itself, let’s look at the roles that must be filled in a successful project. Project management has

**3**

www.it-ebooks.info

**4** CHAPTER 1 ***The data science process***

been a central concern of software engineering for a long time, so we can look there for guidance. In defining the roles here, we’ve borrowed some ideas from Fredrick Brooks’s *The Mythical Man-Month: Essays on Software Engineering* (Addison-Wesley, 1995) “surgical team” perspective on software development and also from the agile software development paradigm.

*1.1.1 Project roles*

Let’s look at a few recurring roles in a data science project in table 1.1.

Table 1.1 Data science project roles and responsibilities

Role Responsibilities

Project sponsor Represents the business interests; champions the project

Client Represents end users’ interests; domain expert

Data scientist Sets and executes analytic strategy; communicates with sponsor and client Data architect Manages data and data storage; sometimes manages data collection Operations Manages infrastructure; deploys final project results

Sometimes these roles may overlap. Some roles—in particular client, data architect, and operations—are often filled by people who aren’t on the data science project team, but are key collaborators.

PROJECT SPONSOR

*The most important role in a data science project is the project sponsor.* The sponsor is the per son who wants the data science result; generally they represent the business interests. The sponsor is responsible for deciding whether the project is a success or failure. The data scientist may fill the sponsor role for their own project if they feel they know and can represent the business needs, but that’s not the optimal arrangement. The ideal sponsor meets the following condition: if they’re satisfied with the project out come, then the project is by definition a success. *Getting sponsor sign-off becomes the cen tral organizing goal of a data science project.*

KEEP THE SPONSOR INFORMED AND INVOLVED It’s critical to keep the sponsor informed and involved. Show them plans, progress, and intermediate suc cesses or failures in terms they can understand. A good way to guarantee proj ect failure is to keep the sponsor in the dark.

To ensure sponsor sign-off, you must get clear goals from them through directed interviews. You attempt to capture the sponsor’s expressed goals as quantitative state ments. An example goal might be “Identify 90% of accounts that will go into default at least two months before the first missed payment with a false positive rate of no more than 25%.” This is a precise goal that allows you to check in parallel if meeting the

www.it-ebooks.info

***The roles in a data science project* 5**

goal is actually going to make business sense and whether you have data and tools of sufficient quality to achieve the goal.

CLIENT

While the sponsor is the role that represents the business interest, the client is the role that represents the model’s end users’ interests. Sometimes the sponsor and client roles may be filled by the same person. Again, the data scientist may fill the client role if they can weight business trade-offs, but this isn’t ideal.

The client is more hands-on than the sponsor; they’re the interface between the technical details of building a good model and the day-to-day work process into which the model will be deployed. They aren’t necessarily mathematically or statistically sophisticated, but are familiar with the relevant business processes and serve as the domain expert on the team. In the loan application example that we discuss later in this chapter, the client may be a loan officer or someone who represents the interests of loan officers.

As with the sponsor, you should keep the client informed and involved. Ideally you’d like to have regular meetings with them to keep your efforts aligned with the needs of the end users. Generally the client belongs to a different group in the organi zation and has other responsibilities beyond your project. Keep meetings focused, present results and progress in terms they can understand, and take their critiques to heart. If the end users can’t or won’t use your model, then the project isn’t a success, in the long run.

DATA SCIENTIST

The next role in a data science project is the data scientist, who’s responsible for tak ing all necessary steps to make the project succeed, including setting the project strat egy and keeping the client informed. They design the project steps, pick the data sources, and pick the tools to be used. Since they pick the techniques that will be tried, they have to be well informed about statistics and machine learning. They’re also responsible for project planning and tracking, though they may do this with a project management partner.

At a more technical level, the data scientist also looks at the data, performs statisti cal tests and procedures, applies machine learning models, and evaluates results—the science portion of data science.

DATA ARCHITECT

The data architect is responsible for all of the data and its storage. Often this role is filled by someone outside of the data science group, such as a database administrator or architect. Data architects often manage data warehouses for many different proj ects, and they may only be available for quick consultation.

OPERATIONS

The operations role is critical both in acquiring data and delivering the final results. The person filling this role usually has operational responsibilities outside of the data science group. For example, if you’re deploying a data science result that affects how

www.it-ebooks.info

**6** CHAPTER 1 ***The data science process***

products are sorted on an online shopping site, then the person responsible for run ning the site will have a lot to say about how such a thing can be deployed. This person will likely have constraints on response time, programming language, or data size that you need to respect in deployment. The person in the operations role may already be supporting your sponsor or your client, so they’re often easy to find (though their time may be already very much in demand).

*1.2 Stages of a data science project*

The ideal data science environment is one that encourages feedback and iteration between the data scientist and all other stakeholders. This is reflected in the lifecycle of a data science project. Even though this book, like any other discussions of the data science process, breaks up the cycle into distinct stages, in reality the boundaries between the stages are fluid, and the activities of one stage will often overlap those of other stages. Often, you’ll loop back and forth between two or more stages before moving forward in the overall process. This is shown in figure 1.1.

Even after you complete a project and deploy a model, new issues and questions can arise from seeing that model in action. The end of one project may lead into a follow-up project.

What problem am

I solving?

Deploy the model to solve the problem in the real world.

Deploy

model

Present

results and

document

Establish that I can solve the problem, and how.

Define the

goal

Evaluate

and critique

model

Does the model solve my problem?

What information

do I need?

Collect

and manage

data

Build the

model

Find patterns in the

data that lead to

solutions.

Figure 1.1 The lifecycle of a

data science project: loops

within loops

www.it-ebooks.info

***Stages of a data science project* 7**

Let’s look at the different stages shown in figure 1.1. As a real-world example, sup pose you’re working for a German bank.1 The bank feels that it’s losing too much money to bad loans and wants to reduce its losses. This is where your data science team comes in.

*1.2.1 Defining the goal*

The first task in a data science project is to define a measurable and quantifiable goal. At this stage, learn all that you can about the context of your project:

◼ Why do the sponsors want the project in the first place? What do they lack, and what do they need?

◼ What are they doing to solve the problem now, and why isn’t that good enough? ◼ What resources will you need: what kind of data and how much staff? Will you have domain experts to collaborate with, and what are the computational resources?

◼ How do the project sponsors plan to deploy your results? What are the con straints that have to be met for successful deployment?

Let’s come back to our loan application example. The ultimate business goal is to reduce the bank’s losses due to bad loans. Your project sponsor envisions a tool to help loan officers more accurately score loan applicants, and so reduce the number of bad loans made. At the same time, it’s important that the loan officers feel that they have final discretion on loan approvals.

Once you and the project sponsor and other stakeholders have established prelim inary answers to these questions, you and they can start defining the precise goal of the project. The goal should be specific and measurable, not “We want to get better at finding bad loans,” but instead, “We want to reduce our rate of loan charge-offs by at least 10%, using a model that predicts which loan applicants are likely to default.”

A concrete goal begets concrete stopping conditions and concrete acceptance cri teria. The less specific the goal, the likelier that the project will go unbounded, because no result will be “good enough.” If you don’t know what you want to achieve, you don’t know when to stop trying—or even what to try. When the project eventually terminates—because either time or resources run out—no one will be happy with the outcome.

This doesn’t mean that more exploratory projects aren’t needed at times: “Is there something in the data that correlates to higher defaults?” or “Should we think about reducing the kinds of loans we give out? Which types might we eliminate?” In this situ ation, you can still scope the project with concrete stopping conditions, such as a time

1 For this chapter, we use a credit dataset donated by Professor Dr. Hans Hofmann to the UCI Machine Learn ing Repository in 1994. We’ve simplified some of the column names for clarity. The dataset can be found at http://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data). We show how to load this data and prepare it for analysis in chapter 2. Note that the German currency at the time of data collection was the deutsch mark (DM).

www.it-ebooks.info

**8** CHAPTER 1 ***The data science process***

limit. The goal is then to come up with candidate hypotheses. These hypotheses can then be turned into concrete questions or goals for a full-scale modeling project. Once you have a good idea of the project’s goals, you can focus on collecting data to meet those goals.

*1.2.2 Data collection and management*

This step encompasses identifying the data you need, exploring it, and conditioning it to be suitable for analysis. This stage is often the most time-consuming step in the pro cess. It’s also one of the most important:

◼ What data is available to me?

◼ Will it help me solve the problem?

◼ Is it enough?

◼ Is the data quality good enough?

Imagine that for your loan application problem, you’ve collected a sample of repre sentative loans from the last decade (excluding home loans). Some of the loans have defaulted; most of them (about 70%) have not. You’ve collected a variety of attributes about each loan application, as listed in table 1.2.

Table 1.2 Loan data attributes

Status.of.existing.checking.account *(at time of application)*

Duration.in.month *(loan length)*

Credit.history

Purpose *(car loan, student loan, etc.)*

Credit.amount *(loan amount)*

Savings.Account.or.bonds *(balance/amount)*

Present.employment.since

Installment.rate.in.percentage.of.disposable.income

Personal.status.and.sex

Cosigners

Present.residence.since

Collateral *(car, property, etc.)*

Age.in.years

Other.installment.plans *(other loans/lines of credit—the type)*

Housing *(own, rent, etc.)*

Number.of.existing.credits.at.this.bank

Job *(employment type)*

Number.of.dependents

Telephone *(do they have one)*

Good.Loan *(dependent variable)*

www.it-ebooks.info

***Stages of a data science project* 9**

In your data, Good.Loan takes on two possible values: GoodLoan and BadLoan. For the purposes of this discussion, assume that a GoodLoan was paid off, and a BadLoan defaulted.

As much as possible, try to use information that can be directly measured, rather than information that is inferred from another measurement. For example, you might be tempted to use income as a variable, reasoning that a lower income implies more difficulty paying off a loan. The ability to pay off a loan is more directly measured by considering the size of the loan payments relative to the borrower’s disposable income. This information is more useful than income alone; you have it in your data as the variable Installment.rate.in.percentage.of.disposable.income.

This is the stage where you conduct initial exploration and visualization of the data. You’ll also clean the data: repair data errors and transform variables, as needed. In the process of exploring and cleaning the data, you may discover that it isn’t suit able for your problem, or that you need other types of information as well. You may discover things in the data that raise issues more important than the one you origi nally planned to address. For example, the data in figure 1.2 seems counterintuitive.

Why would some of the seemingly safe applicants (those who repaid all credits to the bank) default at a higher rate than seemingly riskier ones (those who had been delinquent in the past)? After looking more carefully at the data and sharing puzzling findings with other stakeholders and domain experts, you realize that this sample is inherently biased: *you only have loans that were actually made (and therefore already*

No credits/all paid back

All credits at this bank paid back

y**Good.Loan** r

o

t

s

i

h

ti

d

e

r

C

No current delinquencies Delinquencies in past

Other credits (not at this bank)

0.00 0.25 0.50 0.75 1.00 fraction of defaulted loans

BadLoan GoodLoan

Figure 1.2 The fraction of defaulting loans by credit history category. The dark region of each bar represents the fraction of loans in that category that defaulted.

www.it-ebooks.info

**10** CHAPTER 1 ***The data science process***

*accepted)*. Overall, there are fewer risky-looking loans than safe-looking ones in the data. The probable story is that risky-looking loans were approved after a much stricter vetting process, a process that perhaps the safe-looking loan applications could bypass. This suggests that if your model is to be used downstream of the current application approval process, credit history is no longer a useful variable. It also sug gests that even seemingly safe loan applications should be more carefully scrutinized.

Discoveries like this may lead you and other stakeholders to change or refine the project goals. In this case, you may decide to concentrate on the seemingly safe loan applications. It’s common to cycle back and forth between this stage and the previous one, as well as between this stage and the modeling stage, as you discover things in the data. We’ll cover data exploration and management in depth in chapters 3 and 4.

*1.2.3 Modeling*

You finally get to statistics and machine learning during the modeling, or analysis, stage. Here is where you try to extract useful insights from the data in order to achieve your goals. Since many modeling procedures make specific assumptions about data distribution and relationships, there will be overlap and back-and-forth between the modeling stage and the data cleaning stage as you try to find the best way to represent the data and the best form in which to model it.

The most common data science modeling tasks are these:

◼ *Classification*—*Deciding* if something belongs to one category or another ◼ *Scoring*—*Predicting* or *estimating* a numeric value, such as a price or probability ◼ *Ranking*—Learning to *order items* by preferences

◼ *Clustering*—*Grouping items* into most-similar groups

◼ *Finding relations*—*Finding correlations* or potential causes of effects seen in the data ◼ *Characterization*—Very general *plotting* and *report generation* from data

For each of these tasks, there are several different possible approaches. We’ll cover some of the most common approaches to the different tasks in this book. The loan application problem is a classification problem: you want to identify loan applicants who are likely to default. Three common approaches in such cases are logistic regression, Naive Bayes classifiers, and decision trees (we’ll cover these meth ods in-depth in future chapters). You’ve been in conversation with loan officers and others who would be using your model in the field, so you know that they want to be able to understand the chain of reasoning behind the model’s classification, and they want an indication of how confident the model is in its decision: is this applicant highly likely to default, or only somewhat likely? Given the preceding desiderata, you decide that a decision tree is most suitable. We’ll cover decision trees more extensively in a future chapter, but for now the call in R is as shown in the following listing (you can download data from https://github.com/WinVector/zmPDSwR/tree/master/ Statlog).2

2 In this chapter, for clarity of illustration we deliberately fit a small and shallow tree. www.it-ebooks.info

***Stages of a data science project* 11**

Listing 1.1 Building a decision tree

library('rpart')

load('GCDData.RData')

model <- rpart(Good.Loan ~

Duration.in.month +

Installment.rate.in.percentage.of.disposable.income +

Credit.amount +

Other.installment.plans,

data=d,

control=rpart.control(maxdepth=4),

method="class")

Let’s suppose that you discover the model shown in figure 1.3.

Duration ≥

34 months

Yes No

Credit amount

≥ 11,000

BadLoan

Credit amount

< 2249

Duration ≥

BadLoan

GoodLoan

(0.88)

Credit amount

< 7413

44 months

(1.0)

GoodLoan

(0.61)

(0.75)

Confidence scores are for

BadLoan (0.68)

GoodLoan (0.56)

the declared class:

• BadLoan (1.0) means all the loans that land at the node are bad.

• GoodLoan (0.75) means 75% of the loans that land at the node are good.

Figure 1.3 A decision tree model for finding bad loan applications, with confidence scores

We’ll discuss general modeling strategies in chapter 5 and go into details of specific modeling algorithms in part 2.

*1.2.4 Model evaluation and critique*

Once you have a model, you need to determine if it meets your goals:

◼ Is it accurate enough for your needs? Does it generalize well?

◼ Does it perform better than “the obvious guess”? Better than whatever estimate you currently use?

◼ Do the results of the model (coefficients, clusters, rules) make sense in the con text of the problem domain?

www.it-ebooks.info

**12** CHAPTER 1 ***The data science process***

If you’ve answered “no” to any of these questions, it’s time to loop back to the model ing step—or decide that the data doesn’t support the goal you’re trying to achieve. No one likes negative results, but understanding when you can’t meet your success crite ria with current resources will save you fruitless effort. Your energy will be better spent on crafting success. This might mean defining more realistic goals or gathering the additional data or other resources that you need to achieve your original goals.

Returning to the loan application example, the first thing to check is that the rules that the model discovered make sense. Looking at figure 1.3, you don’t notice any obviously strange rules, so you can go ahead and evaluate the model’s accuracy. A good summary of classifier accuracy is the *confusion matrix*, which tabulates actual clas sifications against predicted ones.3

Listing 1.2 Plotting the confusion matrix

> resultframe <- data.frame(Good.Loan=creditdata$Good.Loan,

pred=predict(model, type="class"))

> rtab <- table(resultframe)

**Overall model accuracy:**

**73% of the predictions**

> rtab

pred

Good.Loan BadLoan GoodLoan BadLoan 41 259 GoodLoan 13 687

**Create the confusion matrix. Rows represent actual loan status; columns represent predicted loan status. The diagonal entries represent correct predictions.**

**were correct. Model precision: 76% of** > sum(diag(rtab))/sum(rtab)

[1] 0.728

> sum(rtab[1,1])/sum(rtab[,1]) [1] 0.7592593

> sum(rtab[1,1])/sum(rtab[1,]) [1] 0.1366667

> sum(rtab[2,1])/sum(rtab[2,]) [1] 0.01857143

**the applicants predicted**

**as bad really did default.**

**Model recall: the model found**

**14% of the defaulting loans.**

**False positive rate: 2% of the good applicants were mistakenly identified as bad.**

The model predicted loan status correctly 73% of the time—better than chance (50%). In the original dataset, 30% of the loans were bad, so guessing GoodLoan all the time would be 70% accurate (though not very useful). So you know that the model does better than random and somewhat better than obvious guessing.

Overall accuracy is not enough. You want to know what kinds of mistakes are being made. Is the model missing too many bad loans, or is it marking too many good loans as bad? *Recall* measures how many of the bad loans the model can actually find. *Preci sion* measures how many of the loans identified as bad really are bad. *False positive rate* measures how many of the good loans are mistakenly identified as bad. Ideally, you want the recall and the precision to be high, and the false positive rate to be low. What constitutes “high enough” and “low enough” is a decision that you make together with

3 Normally, we’d evaluate the model against a test set (data that wasn’t used to build the model). In this exam ple, for simplicity, we evaluate the model against the training data (data that was used to build the model).

www.it-ebooks.info

***Stages of a data science project* 13**

the other stakeholders. Often, the right balance requires some trade-off between recall and precision.

There are other measures of accuracy and other measures of the quality of a model, as well. We’ll talk about model evaluation in chapter 5.

*1.2.5 Presentation and documentation*

Once you have a model that meets your success criteria, you’ll present your results to your project sponsor and other stakeholders. You must also document the model for those in the organization who are responsible for using, running, and maintaining the model once it has been deployed.

Different audiences require different kinds of information. Business-oriented audiences want to understand the impact of your findings in terms of business met rics. In our loan example, the most important thing to present to business audiences is how your loan application model will reduce charge-offs (the money that the bank loses to bad loans). Suppose your model identified a set of bad loans that amounted to 22% of the total money lost to defaults. Then your presentation or executive sum mary should emphasize that the model can potentially reduce the bank’s losses by that amount, as shown in figure 1.4.

Result: Charge-offs reduced 22%

Charge−off amounts by loan category

Dark blue represents loans rejected by model

e

s

o

pr

u

P

car (new)

furniture/equipment business

radio/television

car (used)

education

others

repairs

domestic appliances retraining

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

0 100,000 200,000 300,000 Charge−offs (DM) 

**detected** 

detected

not detected

Figure 1.4 Notional slide from an executive presentation www.it-ebooks.info

**14** CHAPTER 1 ***The data science process***

You also want to give this audience your most interesting findings or recommenda tions, such as that new car loans are much riskier than used car loans, or that most losses are tied to bad car loans and bad equipment loans (assuming that the audience didn’t already know these facts). Technical details of the model won’t be as interesting to this audience, and you should skip them or only present them at a high level.

A presentation for the model’s end users (the loan officers) would instead empha size how the model will help them do their job better:

◼ How should they interpret the model?

◼ What does the model output look like?

◼ If the model provides a trace of which rules in the decision tree executed, how do they read that?

◼ If the model provides a confidence score in addition to a classification, how should they use the confidence score?

◼ When might they potentially overrule the model?

Presentations or documentation for operations staff should emphasize the impact of your model on the resources that they’re responsible for.

We’ll talk about the structure of presentations and documentation for various audiences in part 3.

*1.2.6 Model deployment and maintenance*

Finally, the model is put into operation. In many organizations this means the data sci entist no longer has primary responsibility for the day-to-day operation of the model. But you still should ensure that the model will run smoothly and won’t make disas trous unsupervised decisions. You also want to make sure that the model can be updated as its environment changes. And in many situations, the model will initially be deployed in a small pilot program. The test might bring out issues that you didn’t anticipate, and you may have to adjust the model accordingly. We’ll discuss model deployment considerations in chapter 10.

For example, you may find that loan officers frequently override the model in cer tain situations because it contradicts their intuition. Is their intuition wrong? Or is your model incomplete? Or, in a more positive scenario, your model may perform so successfully that the bank wants you to extend it to home loans as well.

Before we dive deeper into the stages of the data science lifecycle in the following chapters, let’s look at an important aspect of the initial project design stage: setting expectations.

*1.3 Setting expectations*

Setting expectations is a crucial part of defining the project goals and success criteria. The business-facing members of your team (in particular, the project sponsor) proba bly already have an idea of the performance required to meet business goals: for example, the bank wants to reduce their losses from bad loans by at least 10%. Before

www.it-ebooks.info

***Setting expectations* 15**

you get too deep into a project, you should make sure that the resources you have are enough for you to meet the business goals.

In this section, we discuss ways to estimate whether the data you have available is good enough to potentially meet desired accuracy goals. This is an example of the flu idity of the project lifecycle stages. You get to know the data better during the explora tion and cleaning phase; after you have a sense of the data, you can get a sense of whether the data is good enough to meet desired performance thresholds. If it’s not, then you’ll have to revisit the project design and goal-setting stage.

*1.3.1 Determining lower and upper bounds on model performance*

Understanding how well a model *should* do for acceptable performance and how well it *can* do given the available data are both important when defining acceptance criteria.

THE NULL MODEL: A LOWER BOUND ON PERFORMANCE

You can think of the *null model* as being “the obvious guess” that your model must do better than. In situations where there’s a working model or solution already in place that you’re trying to improve, the null model is the existing solution. In situations where there’s no existing model or solution, the null model is the simplest possible model (for example, always guessing GoodLoan, or always predicting the mean value of the output, when you’re trying to predict a numerical value). The null model repre sents the lower bound on model performance that you should strive for.

In our loan application example, 70% of the loan applications in the dataset turned out to be good loans. A model that labels all loans as GoodLoan (in effect, using only the existing process to classify loans) would be correct 70% of the time. So you know that any actual model that you fit to the data should be better than 70% accu rate to be useful. Since this is the simplest possible model, its error rate is called the *base error rate*.

How much better than 70% should you be? In statistics there’s a procedure called *hypothesis testing*, or *significance testing*, that tests whether your model is equivalent to a null model (in this case, whether a new model is basically only as accurate as guessing GoodLoan all the time). You want your model’s accuracy to be “significantly better”—in statistical terms—than 70%. We’ll cover the details of significance testing in chapter 5.

Accuracy is not the only (or even the best) performance metric. In our example, the null model would have zero recall in identifying bad loans, which obviously is not what you want. Generally if there is an existing model or process in place, you’d like to have an idea of its precision, recall, and false positive rates; if the purpose of your proj ect is to improve the existing process, then the current model must be unsatisfactory for at least one of these metrics. This also helps you determine lower bounds on desired performance.

THE BAYES RATE: AN UPPER BOUND ON MODEL PERFORMANCE

The business-dictated performance goals will of course be higher than the lower bounds discussed here. You should try to make sure as early as possible that you have the data to meet your goals.

www.it-ebooks.info

**16** CHAPTER 1 ***The data science process***

One thing to look at is what statisticians call the *unexplainable variance*: how much of the variation in your output can’t be explained by your input variables. Let’s take a very simple example: suppose you want to use the rule of thumb that loans that equal more than 15% of the borrower’s disposable income will default; otherwise, loans are good. You want to know if this rule alone will meet your goal of predicting bad loans with at least 85% accuracy. Let’s consider the two populations next.

Listing 1.3 Plotting the relation between disposable income and loan outcome

**The count of correct predictions is on the diagonal of tab1. In this first population, all the loans that were less than 15% of disposable income were good loans, and all but six of the loans that were greater than 15% of disposable income defaulted. So you know that**

**loan.as.pct.disposable.income models loan quality well in this population. Or as statisticians**

> tab1

**might say, loan.as.pct.disposable.income “explains” the output (loan quality).** loan.quality.pop1

**In fact,**

**it’s 94% accurate.**

loan.as.pct.disposable.income goodloan badloan LT.15pct 50 0

GT.15pct 6 44

> sum(diag(tab1))/sum(tab1)

[1] 0.94

>

> tab2

loan.quality.pop2

loan.as.pct.disposable.income goodloan badloan LT.15pct 34 16

GT.15pct 18 32

> sum(diag(tab2))/sum(tab2)

**In the second population, about a third of the loans that were less than 15% of disposable income defaulted, and over half of the loans that were greater than 15% of disposable income were good. So you know that**

**loan.as.pct.disposable.income doesn’t model loan quality well**

[1] 0.66

**in this population. The rule of thumb is only 66% accurate.**

For the second population, you know that you can’t meet your goals using only loan.as.pct.disposable.income. To build a more accurate model, you’ll need addi tional input variables.

The limit on prediction accuracy due to unexplainable variance is known as the *Bayes rate*. You can think of the Bayes rate as describing the best accuracy you can achieve given your data. If the Bayes rate doesn’t meet your business-dictated perfor mance goals, then you shouldn’t start the project without revisiting your goals or find ing additional data to improve your model.4

Exactly finding the Bayes rate is not always possible—if you could always find the best possible model, then your job would already be done. If all your variables are discrete (and you have a lot of data), you can find the Bayes rate by building a lookup table for all possible variable combinations. In other situations, a nearest-neighbor classifier (we’ll discuss them in chapter 8) can give you a good estimate of the Bayes rate, even though a nearest-neighbor classifier may not be practical to deploy as an actual production model. In any case, you should try to get some idea of the

4 The Bayes rate gives the best possible accuracy, but the most accurate model doesn’t always have the best pos sible precision or recall (though it may represent the best trade-off of the two).

www.it-ebooks.info

***Summary* 17**

limitations of your data early in the process, so you know whether it’s adequate to meet your goals.

*1.4 Summary*

The data science process involves a lot of back-and-forth—between the data scientist and other project stakeholders, and between the different stages of the process. Along the way, you’ll encounter surprises and stumbling blocks; this book will teach you pro cedures for overcoming some of these hurdles. It’s important to keep all the stake holders informed and involved; when the project ends, no one connected with it should be surprised by the final results.

In the next chapters, we’ll look at the stages that follow project design: loading, exploring, and managing the data. Chapter 2 covers a few basic ways to load the data into R, in a format that’s convenient for analysis.

| Key takeaways  ◼ A successful data science project involves more than just statistics. It also requires a variety of roles to represent business and client interests, as well as operational concerns.  ◼ Make sure you have a clear, verifiable, quantifiable goal.  ◼ Make sure you’ve set realistic expectations for all stakeholders. |
| --- |

www.it-ebooks.info

*Loading data into R*

*This chapter covers*

◼ Understanding R’s data frame structure

◼ Loading data into R from files and from

relational databases

◼ Transforming data for analysis

If your experience has been like ours, many of your data science projects start when someone points you toward a bunch of data and you’re left to make sense of it. Your first thought may be to use shell tools or spreadsheets to sort through it, but you quickly realize that you’re taking more time tinkering with the tools than actu ally analyzing the data. Luckily, there’s a better way. In this chapter, we’ll demon strate how to quickly load and start working with data using R. Using R to transform data is easy because R’s main data type (the data frame) is ideal for working with structured data, and R has adapters that read data from many common data for mats. In this chapter, we’ll start with small example datasets found in files and then move to datasets from relational databases. By the end of the chapter, you’ll be able to confidently use R to extract, transform, and load data for analysis.1

For our first example, let’s start with some example datasets from files.

1 We’ll demonstrate and comment on the R commands necessary to prepare the data, but if you’re unfamil iar with programming in R, we recommend at least skimming appendix A or consulting a good book on R such as *R in Action, Second Edition* (Robert Kabacoff, Manning Publications (2014), http://mng.bz/ ybS4). All the tools you need are freely available and we provide instructions how to download and start working with them in appendix A.

**18**

www.it-ebooks.info

***Working with data from files* 19**

*2.1 Working with data from files*

The most common ready-to-go data format is a family of tabular formats called *struc tured values*. Most of the data you find will be in (or nearly in) one of these formats. When you can read such files into R, you can analyze data from an incredible range of public and private data sources. In this section, we’ll work on two examples of loading data from structured files, and one example of loading data directly from a relational database. The point is to get data quickly into R so we can then use R to perform inter esting analyses.

*2.1.1 Working with well-structured data from files or URLs*

The easiest data format to read is table-structured data with headers. As shown in fig ure 2.1, this data is arranged in rows and columns where the first row gives the column names. Each column represents a different fact or measurement; each row represents an instance or datum about which we know the set of facts. A lot of public data is in this format, so being able to read it opens up a lot of opportunities.

Before we load the German credit data that we used in the previous chapter, let’s demonstrate the basic loading commands with a simple data file from the University of California Irvine Machine Learning Repository (http://archive.ics.uci.edu/ml/). The UCI data files tend to come without headers, so to save steps (and to keep it very basic, at this point) we’ve pre-prepared our first data example from the UCI car dataset: http://archive.ics.uci.edu/ml/machine-learning-databases/car/. Our pre-prepared file is at http://win-vector.com/dfiles/car.data.csv and looks like the following (details found at https://github.com/WinVector/zmPDSwR/tree/master/UCICar):

buying,maint,doors,persons,lug\_boot,safety,rating vhigh,vhigh,2,2,small,low,unacc

vhigh,vhigh,2,2,small,med,unacc

vhigh,vhigh,2,2,small,high,unacc

vhigh,vhigh,2,2,med,low,unacc

...

**The data rows are in the same format as the header row, but each row contains actual data values. In this case, the first row represents the set of name/value pairs: buying=vhigh,**

**maintenance=vhigh, doors=2, persons=2, and so on.**

**The header row contains the names of the data columns, in this case separated by commas. When the separators are commas, the format is called comma-separated values, or .csv.**

****Figure 2.1 Car data viewed as a table

www.it-ebooks.info

**20** CHAPTER 2 ***Loading data into R***

AVOID “BY HAND” STEPS We strongly encourage you to avoid performing any steps “by hand” when importing data. It’s tempting to use an editor to add a header line to a file, as we did in our example. A better strategy is to write a script either outside R (using shell tools) or inside R to perform any necessary reformatting. Automating these steps greatly reduces the amount of trauma and work during the inevitable data refresh.

Notice that this file is already structured like a spreadsheet with easy-to-identify rows and columns. The data shown here is claimed to be the details about recommenda tions on cars, but is in fact made-up examples used to test some machine-learning the ories. Each (nonheader) row represents a review of a different model of car. The columns represent facts about each car model. Most of the columns are objective mea surements (purchase cost, maintenance cost, number of doors, and so on) and the final column “rating” is marked with the overall rating (vgood, good, acc, and unacc). These sorts of explanations can’t be found in the data but must be extracted from the documentation found with the original data.

LOADING WELL-STRUCTURED DATA FROM FILES OR URLS

Loading data of this type into R is a one-liner: we use the R command read.table() and we’re done. If data were always in this format, we’d meet all of the goals of this section and be ready to move on to modeling with just the following code.

Listing 2.1 Reading the UCI car data

**Filename or URL to get the data from.**

**Specify the**

uciCar <- read.table(

'http://www.win-vector.com/dfiles/car.data.csv', sep=',',

**Command to read from a file or URL and store the result in a new data frame object called uciCar.**

**column or field**

**separator**

header=T )

**Tell R to expect a header line that defines the data column names.**

**as a comma.**

This loads the data and stores it in a new R data frame object called uciCar. Data frames are R’s primary way of representing data and are well worth learning to work with (as we discuss in our appendixes). The read.table() command is powerful and flexible; it can accept many different types of data separators (commas, tabs, spaces, pipes, and others) and it has many options for controlling quoting and escaping data. read.table() can read from local files or remote URLs. If a resource name ends with the *.gz* suffix, read.table() assumes the file has been compressed in gzip style and will automatically decompress it while reading.

EXAMINING OUR DATA

Once we’ve loaded the data into R, we’ll want to examine it. The commands to always try first are these:

◼ class()—Tells you what type of R object you have. In our case, class(uciCar) tells us the object uciCar is of class data.frame.

◼ help()—Gives you the documentation for a class. In particular try help (class(uciCar)) or help("data.frame").

www.it-ebooks.info

***Working with data from files* 21**

◼ summary()—Gives you a summary of almost any R object. summary(uciCar) shows us a lot about the distribution of the UCI car data.

For data frames, the command dim() is also important, as it shows you how many rows and columns are in the data. We show the results of a few of these steps next (steps are prefixed by > and R results are shown after each step).

Listing 2.2 Exploring the car data

> class(uciCar)

[1] "data.frame"

> summary(uciCar)

buying maint doors high :432 high :432 2 :432 low :432 low :432 3 :432 med :432 med :432 4 :432 vhigh:432 vhigh:432 5more:432

persons lug\_boot safety 2 :576 big :576 high:576 4 :576 med :576 low :576 more:576 small:576 med :576

rating

acc : 384

good : 69

unacc:1210

vgood: 65

> dim(uciCar)

[1] 1728 7

**The loaded object**

**uciCar is of type**

**data.frame.**

**The [1] is just an output sequence marker. The actual information is this: uciCar has 1728 rows and 7 columns. Always try to confirm you got a good parse by at least checking that the number of rows is exactly one fewer than the number of lines of text in the original file. The difference of one is because the column header counts as a line, but not as a data row.**

The summary() command shows us the distribution of each variable in the dataset. For example, we know each car in the dataset was declared to seat 2, 4 or more per sons, and we know there were 576 two-seater cars in the dataset. Already we’ve learned a lot about our data, without having to spend a lot of time setting pivot tables as we would have to in a spreadsheet.

WORKING WITH OTHER DATA FORMATS

.csv is not the only common data file format you’ll encounter. Other formats include .tsv (tab-separated values), pipe-separated files, Microsoft Excel workbooks, JSON data, and XML. R’s built-in read.table() command can be made to read most separated value formats. Many of the deeper data formats have corresponding R packages:

◼ *XLS/XLSX*—http://cran.r-project.org/doc/manuals/

R-data.html#Reading-Excel-spreadsheets

◼ *JSON*—http://cran.r-project.org/web/packages/rjson/index.html ◼ *XML*—http://cran.r-project.org/web/packages/XML/index.html ◼ *MongoDB*—http://cran.r-project.org/web/packages/rmongodb/index.html ◼ *SQL*—http://cran.r-project.org/web/packages/DBI/index.html

www.it-ebooks.info

**22** CHAPTER 2 ***Loading data into R***

*2.1.2 Using R on less-structured data*

Data isn’t always available in a ready-to-go format. Data curators often stop just short of producing a ready-to-go machine-readable format. The German bank credit dataset discussed in chapter 1 is an example of this. This data is stored as tabular data without headers; it uses a cryptic encoding of values that requires the dataset’s accompanying documentation to untangle. This isn’t uncommon and is often due to habits or limita tions of other tools that commonly work with the data. Instead of reformatting the data before we bring it into R, as we did in the last example, we’ll now show how to reformat the data using R. This is a much better practice, as we can save and reuse the R commands needed to prepare the data.

Details of the German bank credit dataset can be found at http://mng.bz/mZbu. We’ll show how to transform this data into something meaningful using R. After these steps, you can perform the analysis already demonstrated in chapter 1. As we can see in our file excerpt, the data is an incomprehensible block of codes with no meaning ful explanations:

A11 6 A34 A43 1169 A65 A75 4 A93 A101 4 ...

A12 48 A32 A43 5951 A61 A73 2 A92 A101 2 ...

A14 12 A34 A46 2096 A61 A74 2 A93 A101 3 ...

...

TRANSFORMING DATA IN R

Data often needs a bit of transformation before it makes any sense. In order to decrypt troublesome data, you need what’s called the *schema documentation* or a *data dictionary*. In this case, the included dataset description says the data is 20 input col umns followed by one result column. In this example, there’s no header in the data file. The column definitions and the meaning of the cryptic A-\* codes are all in the accompanying data documentation. Let’s start by loading the raw data into R. We can either save the data to a file or let R load the data directly from the URL. Start a copy of R or RStudio (see appendix A) and type in the commands in the following listing.

Listing 2.3 Loading the credit dataset

d <- read.table(paste('http://archive.ics.uci.edu/ml/',

'machine-learning-databases/statlog/german/german.data',sep=''),

stringsAsFactors=F,header=F)

print(d[1:3,])

Notice that this prints out the exact same three rows we saw in the raw file with the addition of column names V1 through V21. We can change the column names to something meaningful with the command in the following listing.

Listing 2.4 Setting column names

colnames(d) <- c('Status.of.existing.checking.account',

'Duration.in.month', 'Credit.history', 'Purpose',

'Credit.amount', 'Savings account/bonds',

'Present.employment.since',

www.it-ebooks.info

***Working with data from files* 23**

'Installment.rate.in.percentage.of.disposable.income',

'Personal.status.and.sex', 'Other.debtors/guarantors',

'Present.residence.since', 'Property', 'Age.in.years',

'Other.installment.plans', 'Housing',

'Number.of.existing.credits.at.this.bank', 'Job',

'Number.of.people.being.liable.to.provide.maintenance.for',

'Telephone', 'foreign.worker', 'Good.Loan')

d$Good.Loan <- as.factor(ifelse(d$Good.Loan==1,'GoodLoan','BadLoan')) print(d[1:3,])

The c() command is R’s method to construct a vector. We copied the names directly from the dataset documentation. By assigning our vector of names into the data frame’s colnames() slot, we’ve reset the data frame’s column names to something sen sible. We can find what slots and commands our data frame d has available by typing help(class(d)).

The data documentation further tells us the column names, and also has a diction ary of the meanings of all of the cryptic A-\* codes. For example, it says in column 4 (now called *Purpose*, meaning the purpose of the loan) that the code A40 is a new car loan, A41 is a used car loan, and so on. We copied 56 such codes into an R list that looks like the next listing.

Listing 2.5 Building a map to interpret loan use codes

mapping <- list(

'A40'='car (new)',

'A41'='car (used)',

'A42'='furniture/equipment',

'A43'='radio/television',

'A44'='domestic appliances',

...

)

LISTS ARE R’S MAP STRUCTURES Lists are R’s map structures. They can map strings to arbitrary objects. The important list operations [] and %in% are *vec torized*. This means that, when applied to a vector of values, they return a vec tor of results by performing one lookup per entry.

With the mapping list defined, we can then use the following for loop to convert val ues in each column that was of type character from the original cryptic A-\* codes into short level descriptions taken directly from the data documentation. We, of course, skip any such transform for columns that contain numeric data.

Listing 2.6 Transforming the car data

for(i in 1:(dim(d))[2]) {

if(class(d[,i])=='character') {

d[,i] <- as.factor(as.character(mapping[d[,i]]))

**(dim(d))[2] is the number of columns in the data frame d.**

}

}

**Note that the indexing operator [] is vectorized. Each step in the for loop remaps an entire column of data through our list.**

www.it-ebooks.info

**24** CHAPTER 2 ***Loading data into R***

We share the complete set of column preparations for this dataset here: https:// github.com/WinVector/zmPDSwR/tree/master/Statlog/. We encourage readers to download the data and try these steps themselves.

EXAMINING OUR NEW DATA

We can now easily examine the purpose of the first three loans with the command print(d[1:3,'Purpose']). We can look at the distribution of loan purpose with summary(d$Purpose) and even start to investigate the relation of loan type to loan outcome, as shown in the next listing.

Listing 2.7 Summary of **Good.Loan** and **Purpose**

> table(d$Purpose,d$Good.Loan)

BadLoan GoodLoan

business 34 63

car (new) 89 145

car (used) 17 86

domestic appliances 4 8

education 22 28

furniture/equipment 58 123

others 5 7

radio/television 62 218

repairs 8 14

retraining 1 8

You should now be able to load data from files. But a lot of data you want to work with isn’t in files; it’s in databases. So it’s important that we work through how to load data from databases directly into R.

*2.2 Working with relational databases*

In many production environments, the data you want lives in a relational or SQL data base, not in files. Public data is often in files (as they are easier to share), but your most important client data is often in databases. Relational databases scale easily to the millions of records and supply important production features such as parallelism, consistency, transactions, logging, and audits. When you’re working with transaction data, you’re likely to find it already stored in a relational database, as relational data bases excel at online transaction processing (OLTP).

Often you can export the data into a structured file and use the methods of our previous sections to then transfer the data into R. But this is generally not the right way to do things. Exporting from databases to files is often unreliable and idiosyn cratic due to variations in database tools and the typically poor job these tools do when quoting and escaping characters that are confused with field separators. Data in a database is often stored in what is called a *normalized form*, which requires relational preparations called *joins* before the data is ready for analysis. Also, you often don’t want a dump of the entire database, but instead wish to freely specify which columns and aggregations you need during analysis.

www.it-ebooks.info

***Working with relational databases* 25**

The right way to work with data found in databases is to connect R directly to the database, which is what we’ll demonstrate in this section.

As a step of the demonstration, we’ll show how to load data into a database. Know ing how to load data into a database is useful for problems that need more sophisti cated preparation than we’ve so far discussed. Relational databases are the right place for transformations such as joins or sampling. Let’s start working with data in a data base for our next example.

*2.2.1 A production-size example*

For our production-size example we’ll use the United States Census 2011 national PUMS American Community Survey data found at www.census.gov/acs/www/ data\_documentation/pums\_data/. This is a remarkable set of data involving around 3 million individuals and 1.5 million households. Each row contains over 200 facts about each individual or household (income, employment, education, number of rooms, and so on). The data has household cross-reference IDs so individuals can be joined to the household they’re in. The size of the dataset is interesting: a few giga bytes when zipped up. So it’s small enough to store on a good network or thumb drive, but larger than is convenient to work with on a laptop with R alone (which is more comfortable when working in the range of hundreds of thousands of rows).

This size—millions of rows—is the sweet spot for relational database or SQL assisted analysis on a single machine. We’re not yet forced to move into a MapReduce or database cluster to do our work, but we do want to use a database for some of the initial data handling. We’ll work through all of the steps for acquiring this data and preparing it for analysis in R.

CURATING THE DATA

A hard rule of data science is that you must be able to reproduce your results. At the very least, be able to repeat your own successful work through your recorded steps and without depending on a stash of intermediate results. Everything must either have directions on how to produce it or clear documentation on where it came from. We call this the “no alien artifacts” discipline. For example, when we said we’re using PUMS American Community Survey data, this statement isn’t precise enough for any body to know what data we specifically mean. Our actual notebook entry (which we keep online, so we can search it) on the PUMS data is as shown in the next listing.

Listing 2.8 PUMS data provenance documentation

**Where we found the data documentation. This is important to**

3-12-2013

PUMS Data set from:

**record as many data files don’t contain links back to the documentation. Census PUMS does in fact contain embedded documentation, but not every source is so careful.**

http://www.census.gov/acs/www/data\_documentation/pums\_data/

select "2011 ACS 1-year PUMS"

**How we navigated from the documentation site to the actual data files. It may be necessary to record this if the data supplier requires any sort of click through license to get to the actual data.**

www.it-ebooks.info

**26** CHAPTER 2 ***Loading data into R***

select "2011 ACS 1-year Public Use Microdata Samples\ (PUMS) - CSV format"

download "United States Population Records" and "United States Housing Unit Records"

**The actual files we**

**downloaded.**

http://www2.census.gov/acs2011\_1yr/pums/csv\_pus.zip http://www2.census.gov/acs2011\_1yr/pums/csv\_hus.zip downloaded file details:

**The sizes of the files**

**after we**

**downloaded them.**

$ ls -lh \*.zip

239M Oct 15 13:17 csv\_hus.zip 580M Mar 4 06:31 csv\_pus.zip $ shasum \*.zip

**Cryptographic hashes of the file contents we down loaded. These are very short summaries (called hashes) that are very unlikely to have the same value for different files. These summaries can later help us determine if another researcher in our organization is using the same data distribution or not.**

cdfdfb326956e202fdb560ee34471339ac8abd6c csv\_hus.zip

aa0f4add21e327b96d9898b850e618aeca10f6d0 csv\_pus.zip

KEEP NOTES A big part of being a data scientist is being able to defend your results and repeat your work. We strongly advise keeping a notebook. We also strongly advise keeping all of your scripts and code under version con trol, as we discuss in appendix A. You absolutely need to be able to answer exactly what code and which data were used to build the results you pre sented last week.

STAGING THE DATA INTO A DATABASE

Structured data at a scale of millions of rows is best handled in a database. You can try to work with text-processing tools, but a database is much better at representing the fact that your data is arranged in both rows and columns (not just lines of text).

We’ll use three database tools in this example: the serverless database engine H2, the database loading tool SQL Screwdriver, and the database browser SQuirreL SQL. All of these are Java-based, run on many platforms, and are open source. We describe how to download and start working with all of them in appendix A.2

If you have a database such as MySQL or PostgreSQL already available, we recom mend using one of them instead of using H2.3 To use your own database, you’ll need to know enough of your database driver and connection information to build a JDBC connection. If using H2, you’ll only need to download the H2 driver as described in appendix A, pick a file path to store your results, and pick a username and password (both are set on first use, so there are no administrative steps). H2 is a serverless zero install relational database that supports queries in SQL. It’s powerful enough to work on PUMS data and easy to use. We show how to get H2 running in appendix A.

2 Other easy ways to use SQL in R include the sqldf and RSQLite packages.

3 If you have access to a parallelized SQL database such as Greenplum, we strongly suggest using it to perform aggregation and preparation steps on your big data. Being able to write standard SQL queries and have them finish quickly at big data scale can be game-changing.

www.it-ebooks.info

***Working with relational databases* 27**

We’ll use the Java-based tool SQL Screwdriver to load the PUMS data into our data base. We first copy our database credentials into a Java properties XML file.

Listing 2.9 SQL Screwdriver XML configuration file

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

**Password to use for**

**database**

**connection.**

<properties>

<comment>testdb</comment>

<entry key="user">u</entry> <entry key="password">u</entry>

**Username to use for database connection.**

**Java classname of the database driver. SQL Screwdriver used JDBC, which is a broad database application programming interface layer. You could use another database such as PostgreSQL by specifying a different driver name, such as**

<entry key="driver">org.h2.Driver</entry> <entry key="url">jdbc:h2:H2DB \

**org.postgresql.Driver.**

;LOG=0;CACHE\_SIZE=65536;LOCK\_MODE=0;UNDO\_LOG=0</entry>

</properties>

**URL specifying database. For H2, it’s just jdbc:h2: followed by the file prefix you wish to use to store data. The items after the semicolon are performance options. For PostgreSQL, it would be something more like jdbc:postgresql://host:5432/db. The descriptions of the URL format and drivers should be part of your database documentation, and you can use SQuirreL SQL to confirm you have them right.**

We’ll then use Java at the command line to load the data. To load the four files con taining the two tables, run the commands in the following listing.

Listing 2.10 Loading data with SQL Screwdriver

**URL**

java -classpath SQLScrewdriver.jar:h2-1.3.170.jar \

**Java command and required**

**pointing to database credentials.**

com.winvector.db.LoadFiles \ file:dbDef.xml \

**Class to run: LoadFiles, the meat of SQL Screwdriver.**

**JARs. The JARs in this case are SQL Screwdriver and the required database driver.**

**Separator to expect in**

**input file (use t for tab).**

**Name of**

**table to**

, \

hus \

**List of comma separated files to**

**create.**

file:csv\_hus/ss11husa.csv file:csv\_hus/ss11husb.csv java -classpath SQLScrewdriver.jar:h2-1.3.170.jar \ com.winvector.db.LoadFiles \

file:dbDef.xml , pus \

file:csv\_pus/ss11pusa.csv file:csv\_pus/ss11pusb.csv www.it-ebooks.info

**load into table.**

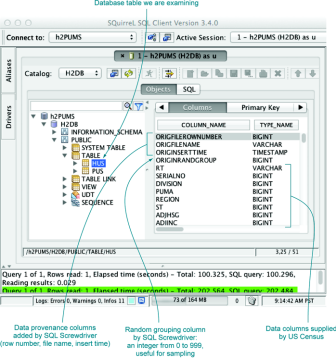
**Same load pattern for personal**

**information table.**

**28** CHAPTER 2 ***Loading data into R***

SQL Screwdriver infers data types by scanning the file and creates new tables in your database. It then populates these tables with the data. SQL Screwdriver also adds four additional “provenance” columns when loading your data. These columns are ORIGINSERTTIME, ORIGFILENAME, ORIGFILEROWNUMBER, and ORIGRANDGROUP. The first three fields record when you ran the data load, what filename the row came from, and what line the row came from. The ORIGRANDGROUP is a pseudo-random integer distributed uniformly from 0 through 999, designed to make repeatable sam pling plans easy to implement. You should get in the habit of having annotations and keeping notes at each step of the process.

We can now use a database browser like SQuirreL SQL to examine this data. We start up SQuirreL SQL and copy the connection details from our XML file into a data base alias, as shown in appendix A. We’re then ready to type SQL commands into the

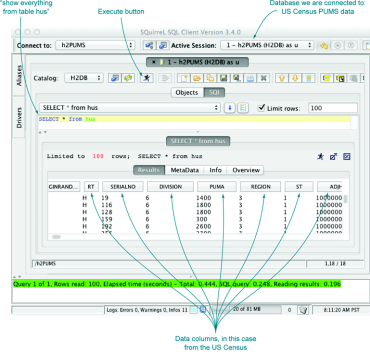
Figure 2.2 SQuirreL SQL table explorer

www.it-ebooks.info

***Working with relational databases* 29**

execution window. A couple of commands you can try are SELECT COUNT(1) FROM hus and SELECT COUNT(1) FROM pus, which will tell you that the hus table has 1,485,292 rows and the pus table has 3,112,017 rows. Each of the tables has over 200 columns, and there are over a billion cells of data in these two tables. We can actually do a lot more. In addition to the SQL execution panel, SQuirreL SQL has an Objects panel that allows graphical exploration of database table definitions. Figure 2.2 shows some of the columns in the hus table.

Now we can view our data as a table (as we would in a spreadsheet). We can now exam ine, aggregate, and summarize our data using the SQuirreL SQL database browser. Fig ure 2.3 shows a few example rows and columns from the household data table.

Figure 2.3 Browsing PUMS data using SQuirreL SQL

www.it-ebooks.info

**30** CHAPTER 2 ***Loading data into R***

*2.2.2 Loading data from a database into R*

To load data from a database, we use a database connector. Then we can directly issue SQL queries from R. SQL is the most common database query language and allows us to specify arbitrary joins and aggregations. SQL is called a *declarative* language (as opposed to a *procedural* language) because in SQL we specify what relations we would like our data sample to have, not how to compute them. For our example, we load a sample of the household data from the hus table and the rows from the person table (pus) that are associated with those households.4

Listing 2.11 Loading data into R from a relational database

options( java.parameters = "-Xmx2g" ) library(RJDBC)

**Set Java option for extra memory before DB drivers are loaded.**

**Specify where to find the**

**implementation of the database**

drv <- JDBC("org.h2.Driver",

**Specify the name of the database driver, same as in our XML database configuration.**

**driver. SQL column names with mixed-case capitalization,**

"h2-1.3.170.jar", identifier.quote="'")

**special characters, or that collide with reserved words must be quoted. We specify single-quote as the quote we’ll use when quoting column names, which may be different than the quote we use for SQL literals.**

**Create a data frame called dpus from the database**

**table pus,**

**taking only records that have a**

**household ID in the set of household**

**IDs we**

**selected from households**

options<-";LOG=0;CACHE\_SIZE=65536;LOCK\_MODE=0;UNDO\_LOG=0" conn <- dbConnect(drv,paste("jdbc:h2:H2DB",options,sep=''),"u","u") dhus <- dbGetQuery(conn,"SELECT \* FROM hus WHERE ORIGRANDGROUP<=1") dpus <- dbGetQuery(conn,"SELECT pus.\* FROM pus WHERE pus.SERIALNO IN \ (SELECT DISTINCT hus.SERIALNO FROM hus \

**table hus.Create a data frame called dhus from \* (everything)**

**Disconnect for the**

**database.**

WHERE hus.ORIGRANDGROUP<=1)") dbDisconnect(conn)

**from the database table hus, taking only rows where ORGINRANGGROUP <= 1. The ORGINRANDGROUP column is a random integer from 0 through 999 that SQL Screwdriver adds to the rows during data load to facilitate sampling. In this case, we’re taking 2/1000 of the data rows to get a small sample.**

save(dhus,dpus,file='phsample.RData')

**Save the two data frames into a file named phsample.RData, which can**

**be read in with load(). Try help("save") or help("load") for more details.**

4 Producing composite records that represent matches between one or more tables (in our case hus and pus) is usually done with what is called a *join*. For this example, we use an even more efficient pattern called a sub select that uses the keyword in.

www.it-ebooks.info

***Working with relational databases* 31**

And we’re in business; the data has been unpacked from the Census-supplied .csv files into our database and a useful sample has been loaded into R for analysis. We have actually accomplished a lot. Generating, as we have, a uniform sample of households and matching people would be tedious using shell tools. It’s exactly what SQL data bases are designed to do well.

DON’T BE TOO PROUD TO SAMPLE Many data scientists spend too much time adapting algorithms to work directly with big data. Often this is wasted effort, as for many model types you would get almost exactly the same results on a reasonably sized data sample. You only need to work with “all of your data” when what you’re modeling isn’t well served by sampling, such as when char acterizing rare events or performing bulk calculations over social networks.

Note that this data is still in some sense large (out of the range where using spread sheets is actually reasonable). Using dim(dhus) and dim(dpus), we see that our house hold sample has 2,982 rows and 210 columns, and the people sample has 6,279 rows and 288 columns. All of these columns are defined in the Census documentation.

*2.2.3 Working with the PUMS data*

Remember that the whole point of loading data (even from a database) into R is to facilitate modeling and analysis. Data analysts should always have their “hands in the data” and always take a quick look at their data after loading it. If you’re not willing to work with the data, you shouldn’t bother loading it into R. To emphasize analysis, we’ll demonstrate how to perform a quick examination of the PUMS data.

LOADING AND CONDITIONING THE PUMS DATA

Each row of PUMS data represents a single anonymized person or household. Per sonal data recorded includes occupation, level of education, personal income, and many other demographics variables. To load our prepared data frame, download phsample.Rdata from https://github.com/WinVector/zmPDSwR/tree/master/ PUMS and run the following command in R: load('phsample.RData').

Our example problem will be to predict income (represented in US dollars in the field PINCP) using the following variables:

◼ *Age*—An integer found in column AGEP.

◼ *Employment class*—Examples: for-profit company, nonprofit company, ... found in column COW.

◼ *Education level*—Examples: no high school diploma, high school, college, and so on, found in column SCHL.

◼ *Sex of worker*—Found in column SEX.

We don’t want to concentrate too much on this data; our goal is only to illustrate the modeling procedure. Conclusions are very dependent on choices of data condition ing (what subset of the data you use) and data coding (how you map records to infor mative symbols). This is why empirical scientific papers have a mandatory “materials

www.it-ebooks.info

**32** CHAPTER 2 ***Loading data into R***

and methods” section describing how data was chosen and prepared. Our data treat ment is to select a subset of “typical full-time workers” by restricting the subset to data that meets all of the following conditions:

◼ Workers self-described as full-time employees

◼ Workers reporting at least 40 hours a week of activity

◼ Workers 20–50 years of age

◼ Workers with an annual income between $1,000 and $250,000 dollars The following listing shows the code to limit to our desired subset of the data.

Listing 2.12 Selecting a subset of the Census data

psub = subset(dpus,with(dpus,(PINCP>1000)&(ESR==1)&

(PINCP<=250000)&(PERNP>1000)&(PERNP<=250000)& (WKHP>=40)&(AGEP>=20)&(AGEP<=50)&

(PWGTP1>0)&(COW %in% (1:7))&(SCHL %in% (1:24)))) RECODING THE DATA

**Subset of data rows matching detailed employment conditions**

Before we work with the data, we’ll recode some of the variables for readability. In par ticular, we want to recode variables that are enumerated integers into meaningful factor-level names, but for readability and to prevent accidentally treating such vari ables as mere numeric values. Listing 2.13 shows the typical steps needed to perform a useful recoding.

Listing 2.13 Recoding variables

psub$SEX = as.factor(ifelse(psub$SEX==1,'M','F')) psub$SEX = relevel(psub$SEX,'M')

cowmap <- c("Employee of a private for-profit", "Private not-for-profit employee",

"Local government employee",

"State government employee",

"Federal government employee",

"Self-employed not incorporated",

"Self-employed incorporated")

psub$COW = as.factor(cowmap[psub$COW]) psub$COW = relevel(psub$COW,cowmap[1]) schlmap = c(

rep("no high school diploma",15),

"Regular high school diploma",

"GED or alternative credential",

"some college credit, no degree",

"some college credit, no degree",

"Associate's degree",

"Bachelor's degree",

www.it-ebooks.info

**Reencode sex from 1/2 to M/F.**

**Make the reference**

**sex M, so F encodes**

**a difference from M**

**in models.**

**Reencode class of**

**worker info into a**

**more readable form.**

**Reencode education info into a more readable form and fewer levels (merge all levels below high school into same encoding).**

***Working with relational databases* 33**

"Master's degree",

"Professional degree",

"Doctorate degree")

psub$SCHL = as.factor(schlmap[psub$SCHL]) psub$SCHL = relevel(psub$SCHL,schlmap[1]) dtrain = subset(psub,ORIGRANDGROUP >= 500) dtest = subset(psub,ORIGRANDGROUP < 500)

**Subset of data**

**rows used for**

**model training.**

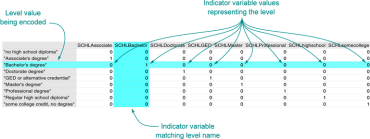
**Subset of data**

**rows used for**

**model testing.**

The data preparation is making use of R’s vectorized lookup operator []. For details on this or any other R commands, we suggest using the R help() command and appendix A (for help with [], type help('[')).

The standard trick to work with variables that take on a small number of string values is to reencode them into what’s called a *factor* as we’ve done with the as.factor() com mand. A factor is a list of all possible values of the variable (possible values are called *lev els*), and each level works (under the covers) as an *indicator variable*. An indicator is a variable with a value of 1 (one) when a condition we’re interested in is true, and 0 (zero) otherwise. Indicators are a useful encoding trick. For example, SCHL is reen coded as 8 indicators with the names shown in figure 7.6 in chapter 7, plus the undis played level “no high school diploma.” Each indicator takes a value of 0, except when the SCHL variable has a value equal to the indicator’s name. When the SCHL variable matches the indicator name, the indicator is set to 1 to indicate the match. Figure 2.4 illustrates the process. SEX and COW underwent similar transformations.

Figure 2.4 Strings encoded as indicators

www.it-ebooks.info

**34** CHAPTER 2 ***Loading data into R***

EXAMINING THE PUMS DATA

At this point, we’re ready to do some science, or at least start looking at the data. For example, we can quickly tabulate the distribution of category of work.

Listing 2.14 Summarizing the classifications of work

> summary(dtrain$COW)

Employee of a private for-profit Federal government employee 423 21

Local government employee Private not-for-profit employee

39 55

Self-employed incorporated Self-employed not incorporated

17 16

State government employee

24

WATCH OUT FOR NAS R’s representation for blank or missing data is NA. Unfortunately a lot of R commands quietly skip NAs without warning. The command table(dpus$COW,useNA='always') will show NAs much like summary(dpus$COW) does.

We’ll return to the Census example and demonstrate more sophisticated modeling techniques in chapter 7.

*2.3 Summary*

In this chapter, we’ve shown how to extract, transform, and load data for analysis. For smaller datasets we perform the transformations in R, and for larger datasets we advise using a SQL database. In either case we save *all* of the transformation steps as code (either in SQL or in R) that can be reused in the event of a data refresh. The whole purpose of this chapter is to prepare for the actual interesting work in our next chap ters: exploring, managing, and correcting data.

The whole point of loading data into R is so we can start to work with it: explore, examine, summarize, and plot it. In chapter 3, we’ll demonstrate how to characterize your data through summaries, exploration, and graphing. These are key steps early in any modeling effort because it is through these steps that you learn the actual details and nature of the problem you’re hoping to model.

| Key takeaways  ◼ Data frames are your friend.  ◼ Use read\_table() to load small, structured datasets into R.  ◼ You can use a package like RJDBC to load data into R from relational data bases, and to transform or aggregate the data before loading using SQL. ◼ Always document data provenance. |
| --- |

www.it-ebooks.info

*This chapter covers*

◼ Using summary statistics to explore data ◼ Exploring data using visualization ◼ Finding problems and issues during data exploration

*Exploring data*

In the last two chapters, you learned how to set the scope and goal of a data science project, and how to load your data into R. In this chapter, we’ll start to get our hands into the data.

Suppose your goal is to build a model to predict which of your customers don’t have health insurance; perhaps you want to market inexpensive health insurance packages to them. You’ve collected a dataset of customers whose health insurance status you know. You’ve also identified some customer properties that you believe help predict the probability of insurance coverage: age, employment status, income, information about residence and vehicles, and so on. You’ve put all your data into a single data frame called *custdata* that you’ve input into R.1 Now you’re ready to start building the model to identify the customers you’re interested in.

1 We have a copy of this synthetic dataset available for download from https://github.com/WinVector/ zmPDSwR/tree/master/Custdata, and once saved, you can load it into R with the command custdata <- read.table('custdata.tsv',header=T,sep='\t').

**35**

www.it-ebooks.info

**36** CHAPTER 3 ***Exploring data***

It’s tempting to dive right into the modeling step without looking very hard at the dataset first, especially when you have a lot of data. Resist the temptation. No dataset is perfect: you’ll be missing information about some of your customers, and you’ll have incorrect data about others. Some data fields will be dirty and inconsistent. If you don’t take the time to examine the data before you start to model, you may find your self redoing your work repeatedly as you discover bad data fields or variables that need to be transformed before modeling. In the worst case, you’ll build a model that returns incorrect predictions—and you won’t be sure why. By addressing data issues early, you can save yourself some unnecessary work, and a lot of headaches!

You’d also like to get a sense of who your customers are: Are they young, middle aged, or seniors? How affluent are they? Where do they live? Knowing the answers to these questions can help you build a better model, because you’ll have a more spe cific idea of what information predicts the probability of insurance coverage more accurately.

In this chapter, we’ll demonstrate some ways to get to know your data, and discuss some of the potential issues that you’re looking for as you explore. Data exploration uses a combination of *summary statistics*—means and medians, variances, and counts— and *visualization*, or graphs of the data. You can spot some problems just by using sum mary statistics; other problems are easier to find visually.

| Organizing data for analysis  For most of this book, we’ll assume that the data you’re analyzing is in a single data frame. This is not how that data is usually stored. In a database, for example, data is usually stored in *normalized form* to reduce redundancy: information about a single customer is spread across many small tables. In log data, data about a single cus tomer can be spread across many log entries, or sessions. These formats make it easy to add (or in the case of a database, modify) data, but are not optimal for anal ysis. You can often join all the data you need into a single table in the database using SQL, but in appendix A we’ll discuss commands like join that you can use within R to further consolidate data. |
| --- |

*3.1 Using summary statistics to spot problems*

In R, you’ll typically use the summary command to take your first look at the data. Listing 3.1 The **summary()** command

> summary(custdata)

custid sex

Min. : 2068 F:440

1st Qu.: 345667 M:560

Median : 693403

Mean : 698500

3rd Qu.:1044606

Max. :1414286

www.it-ebooks.info

***Using summary statistics to spot problems* 37**

is.employed income Mode :logical Min. : -8700 FALSE:73 1st Qu.: 14600 TRUE :599 Median : 35000 NA's :328 Mean : 53505 3rd Qu.: 67000

Max. :615000

marital.stat

Divorced/Separated:155

Married :516

Never Married :233

Widowed : 96

health.ins

Mode :logical

FALSE:159

TRUE :841

NA's :0

housing.type

Homeowner free and clear :157 Homeowner with mortgage/loan:412 Occupied with no rent : 11 Rented :364 NA's : 56

recent.move num.vehicles Mode :logical Min. :0.000 FALSE:820 1st Qu.:1.000 TRUE :124 Median :2.000 NA's :56 Mean :1.916 3rd Qu.:2.000

Max. :6.000

NA's :56

age state.of.res Min. : 0.0 California :100 1st Qu.: 38.0 New York : 71 Median : 50.0 Pennsylvania: 70 Mean : 51.7 Texas : 56 3rd Qu.: 64.0 Michigan : 52 Max. :146.7 Ohio : 51 (Other) :600

**The variable is.employed**

**is missing for about a**

**third of the data. The**

**variable income has**

**negative values, which**

**are potentially invalid.**

**About 84% of the**

**customers have health**

**insurance.**

**The variables housing.type,**

**recent.move, and num.vehicles**

**are each missing 56 values.**

**The average value of the variable**

**age seems plausible, but the**

**minimum and maximum values**

**seem unlikely. The variable**

**state.of.res is a categorical**

**variable; summary() reports how**

**many customers are in each state**

**(for the first few states).**

The summary command on a data frame reports a variety of summary statistics on the numerical columns of the data frame, and count statistics on any categorical columns (if the categorical columns have already been read in as factors2). You can also ask for summary statistics on specific numerical columns by using the commands mean, variance, median, min, max, and quantile (which will return the quartiles of the data by default).

2 Categorical variables are of class factor in R. They can be represented as strings (class character), and some analytical functions will automatically convert string variables to factor variables. To get a summary of a variable, it needs to be a factor.

www.it-ebooks.info

**38** CHAPTER 3 ***Exploring data***

As you see from listing 3.1, the summary of the data helps you quickly spot poten tial problems, like missing data or unlikely values. You also get a rough idea of how categorical data is distributed. Let’s go into more detail about the typical problems that you can spot using the summary.

*3.1.1 Typical problems revealed by data summaries*

At this stage, you’re looking for several common issues: missing values, invalid values and outliers, and data ranges that are too wide or too narrow. Let’s address each of these issues in detail.

MISSING VALUES

A few missing values may not really be a problem, but if a particular data field is largely unpopulated, it shouldn’t be used as an input without some repair (as we’ll dis cuss in chapter 4, section 4.1.1). In R, for example, many modeling algorithms will, by default, quietly drop rows with missing values. As you see in listing 3.2, all the missing values in the is.employed variable could cause R to quietly ignore nearly a third of the data.

Listing 3.2 Will the variable **is.employed** be useful for modeling?

is.employed

Mode :logical

FALSE:73

TRUE :599

NA's :328

housing.type

Homeowner free and clear :157 Homeowner with mortgage/loan:412 Occupied with no rent : 11 Rented :364 NA's : 56

recent.move num.vehicles Mode :logical Min. :0.000 FALSE:820 1st Qu.:1.000 TRUE :124 Median :2.000 NA's :56 Mean :1.916 3rd Qu.:2.000

Max. :6.000

NA's :56

**The variable is.employed is missing for about a third of the data. Why? Is employment status unknown? Did the company start collecting employment data only recently? Does NA mean “not in the active workforce” (for example, students or stay-at-home parents)?**

**The variables housing.type,**

**recent.move, and num.vehicles**

**are only missing a few values. It’s**

**probably safe to just drop the**

**rows that are missing values—**

**especially if the missing values**

**are all the same 56 rows.**

If a particular data field is largely unpopulated, it’s worth trying to determine why; sometimes the fact that a value is missing is informative in and of itself. For example, why is the is.employed variable missing so many values? There are many possible rea sons, as we noted in listing 3.2.

Whatever the reason for missing data, you must decide on the most appropriate action. Do you include a variable with missing values in your model, or not? If you

www.it-ebooks.info

***Using summary statistics to spot problems* 39**

decide to include it, do you drop all the rows where this field is missing, or do you con vert the missing values to 0 or to an additional category? We’ll discuss ways to treat missing data in chapter 4. In this example, you might decide to drop the data rows where you’re missing data about housing or vehicles, since there aren’t many of them. You probably don’t want to throw out the data where you’re missing employment information, but instead treat the NAs as a third employment category. You will likely encounter missing values when model scoring, so you should deal with them during model training.

INVALID VALUES AND OUTLIERS

Even when a column or variable isn’t missing any values, you still want to check that the values that you do have make sense. Do you have any invalid values or outliers? Examples of invalid values include negative values in what should be a non-negative numeric data field (like age or income), or text where you expect numbers. Outliers are data points that fall well out of the range of where you expect the data to be. Can you spot the outliers and invalid values in listing 3.3?

Listing 3.3 Examples of invalid values and outliers

> summary(custdata$income)

Min. 1st Qu. Median Mean 3rd Qu. -8700 14600 35000 53500 67000 Max.

615000

> summary(custdata$age)

Min. 1st Qu. Median Mean 3rd Qu. 0.0 38.0 50.0 51.7 64.0 Max.

146.7

**Negative values for income could indicate bad data. They might also have a special meaning, like “amount of debt.”**

**Either way, you should check how prevalent the issue is, and decide what to do: Do you drop the data with negative income? Do you convert negative values to zero?**

**Customers of age zero, or customers of an age greater than about 110 are outliers. They fall out of the range of expected customer values. Outliers could be data input errors. They could be special sentinel values: zero might mean “age unknown” or “refuse to state.” And some of your customers might be especially long-lived.**

Often, invalid values are simply bad data input. Negative numbers in a field like age, however, could be a *sentinel value* to designate “unknown.” Outliers might also be data errors or sentinel values. Or they might be valid but unusual data points—people do occasionally live past 100.

As with missing values, you must decide the most appropriate action: drop the data field, drop the data points where this field is bad, or convert the bad data to a useful value. Even if you feel certain outliers are valid data, you might still want to omit them from model construction (and also collar allowed prediction range), since the usual achievable goal of modeling is to predict the typical case correctly.

DATA RANGE

You also want to pay attention to how much the values in the data vary. If you believe that age or income helps to predict the probability of health insurance coverage, then

www.it-ebooks.info

**40** CHAPTER 3 ***Exploring data***

you should make sure there is enough variation in the age and income of your cus tomers for you to see the relationships. Let’s look at income again, in listing 3.4. Is the data range wide? Is it narrow?

Listing 3.4 Looking at the data range of a variable

> summary(custdata$income)

Min. 1st Qu. Median Mean 3rd Qu. -8700 14600 35000 53500 67000 Max.

615000

**Income ranges from zero to over half a million dollars; a very wide range.**

Even ignoring negative income, the income variable in listing 3.4 ranges from zero to over half a million dollars. That’s pretty wide (though typical for income). Data that ranges over several orders of magnitude like this can be a problem for some modeling methods. We’ll talk about mitigating data range issues when we talk about logarithmic transformations in chapter 4.

Data can be too narrow, too. Suppose all your customers are between the ages of 50 and 55. It’s a good bet that age range wouldn’t be a very good predictor of the probability of health insurance coverage for that population, since it doesn’t vary much at all.

| How narrow is “too narrow” a data range?  Of course, the term *narrow* is relative. If we were predicting the ability to read for chil dren between the ages of 5 and 10, then age probably is a useful variable as-is. For data including adult ages, you may want to transform or bin ages in some way, as you don’t expect a significant change in reading ability between ages 40 and 50. You should rely on information about the problem domain to judge if the data range is nar row, but a rough rule of thumb is the ratio of the standard deviation to the mean. If that ratio is very small, then the data isn’t varying much. |
| --- |

We’ll revisit data range in section 3.2, when we talk about examining data graphically. One factor that determines apparent data range is the unit of measurement. To take a nontechnical example, we measure the ages of babies and toddlers in weeks or in months, because developmental changes happen at that time scale for very young children. Suppose we measured babies’ ages in years. It might appear numerically that there isn’t much difference between a one-year-old and a two-year-old. In reality, there’s a dramatic difference, as any parent can tell you! Units can present potential issues in a dataset for another reason, as well.

UNITS

Does the income data in listing 3.5 represent hourly wages, or yearly wages in units of $1000? As a matter of fact, it’s the latter, but what if you thought it was the former? You might not notice the error during the modeling stage, but down the line someone will start inputting hourly wage data into the model and get back bad predictions in return.

www.it-ebooks.info

***Spotting problems using graphics and visualization* 41**

Listing 3.5 Checking units can prevent inaccurate results later

**The variable Income is defined**

**as Income = custdata$income/**

> summary(Income)

Min. 1st Qu. Median Mean 3rd Qu. Max. -8.7 14.6 35.0 53.5 67.0 615.0

**1000. But suppose you didn’t know that. Looking only at the summary, the values could plausibly be interpreted to mean either “hourly wage” or “yearly income in units of $1000.”**

Are time intervals measured in days, hours, minutes, or milliseconds? Are speeds in kilometers per second, miles per hour, or knots? Are monetary amounts in dollars, thousands of dollars, or 1/100 of a penny (a customary practice in finance, where cal culations are often done in fixed-point arithmetic)? This is actually something that you’ll catch by checking data definitions in data dictionaries or documentation, rather than in the summary statistics; the difference between hourly wage data and annual salary in units of $1000 may not look that obvious at a casual glance. But it’s still something to keep in mind while looking over the value ranges of your variables, because often you can spot when measurements are in unexpected units. Automobile speeds in knots look a lot different than they do in miles per hour.

*3.2 Spotting problems using graphics and visualization* As you’ve seen, you can spot plenty of problems just by looking over the data summa ries. For other properties of the data, pictures are better than text.

*We cannot expect a small number of numerical values [summary statistics] to consistently convey the wealth of information that exists in data. Numerical reduction methods do not retain the information in the data.*

—William Cleveland

*The Elements of Graphing Data*

Figure 3.1 shows a plot of how customer ages are distributed. We’ll talk about what the y-axis of the graph means later; for right now, just know that the height of the graph corresponds to how many customers in the population are of that age. As you can see, information like the peak age of the distribution, the existence of subpopulations, and the presence of outliers is easier to absorb visually than it is to determine textually.

The use of graphics to examine data is called *visualization*. We try to follow William Cleveland’s principles for scientific visualization. Details of specific plots aside, the key points of Cleveland’s philosophy are these:

◼ A graphic should display as much information as it can, with the lowest possible cognitive strain to the viewer.

◼ Strive for clarity. Make the data stand out. Specific tips for increasing clarity include

– Avoid too many superimposed elements, such as too many curves in the same graphing space.

www.it-ebooks.info

**42** CHAPTER 3 ***Exploring data***

0.020

The peak of the customer population is just under 50. That’s not obvious from the summary.

> summary(custdata$age)

Min. 1st Qu. Median Mean 3rd Qu. Max. 0.0 38.0 50.0 51.7 64.0 146.7

It’s easier to read the mean, median

y

t

is

n

e

d

0.015

0.010

0.005

Invalid

|  | and central 50% of the customerpopulation off the summary.  It’s easier to get a sense of thecustomer age range from the graph. |
| --- | --- |
|  | Customer  “subpopulation”: more |
|  | customers over 75 than  you would expect. |

0.000

values?

Outliers

0 50 100 150 age

Figure 3.1 Some information is easier to read from a graph, and some from a summary.

– Find the right aspect ratio and scaling to properly bring out the details of the data.

– Avoid having the data all skewed to one side or the other of your graph.

◼ Visualization is an iterative process. Its purpose is to answer questions about the data.

During the visualization stage, you graph the data, learn what you can, and then regraph the data to answer the questions that arise from your previous graphic. Differ ent graphics are best suited for answering different questions. We’ll look at some of them in this section.

In this book, we use ggplot2 to demonstrate the visualizations and graphics; of course, other R visualization packages can produce similar graphics.

| A note on ggplot2  The theme of this section is how to use visualization to explore your data, not how to use ggplot2. We chose ggplot2 because it excels at combining multiple graphical elements together, but its syntax can take some getting used to. The key points to understand when looking at our code snippets are these:  ◼ Graphs in ggplot2 can only be defined on data frames. The variables in a graph—the *x* variable, the *y* variable, the variables that define the color or the |
| --- |

www.it-ebooks.info

***Spotting problems using graphics and visualization* 43**

| size of the points—are called *aesthetics*, and are declared by using the aes function.  ◼ The ggplot() function declares the graph object. The arguments to ggplot() can include the data frame of interest and the aesthetics. The ggplot() function doesn’t of itself produce a visualization; visualizations are produced by *layers*.  ◼ Layers produce the plots and plot transformations and are added to a given graph object using the + operator. Each layer can also take a data frame and aesthetics as arguments, in addition to plot-specific parameters. Examples of layers are geom\_point (for a scatter plot) or geom\_line (for a line plot).  This syntax will become clearer in the examples that follow. For more information, we recommend Hadley Wickham’s reference site http://ggplot2.org, which has pointers to online documentation, as well as to Dr. Wickham’s *ggplot2: Elegant Graphics for Data Analysis (Use R!)* (Springer, 2009). |
| --- |

In the next two sections, we’ll show how to use pictures and graphs to identify data characteristics and issues. In section 3.2.2, we’ll look at visualizations for two variables. But let’s start by looking at visualizations for single variables.

*3.2.1 Visually checking distributions for a single variable*

The visualizations in this section help you answer questions like these:

◼ What is the peak value of the distribution?

◼ How many peaks are there in the distribution (unimodality versus bimodality)? ◼ How normal (or lognormal) is the data? We’ll discuss normal and lognormal distributions in appendix B.

◼ How much does the data vary? Is it concentrated in a certain interval or in a cer tain category?

One of the things that’s easier to grasp visually is the shape of the data distribution. Except for the blip to the right, the graph in figure 3.1 (which we’ve reproduced as the gray curve in figure 3.2) is almost shaped like the normal distribution (see appen dix B). As that appendix explains, many summary statistics assume that the data is approximately normal in distribution (at least for continuous variables), so you want to verify whether this is the case.

You can also see that the gray curve in figure 3.2 has only one peak, or that it’s *uni modal*. This is another property that you want to check in your data.

Why? Because (roughly speaking), a unimodal distribution corresponds to one population of subjects. For the gray curve in figure 3.2, the mean customer age is about 52, and 50% of the customers are between 38 and 64 (the first and third quartiles). So you can say that a “typical” customer is middle-aged and probably pos sesses many of the demographic qualities of a middle-aged person—though of course you have to verify that with your actual customer information.

www.it-ebooks.info

**44** CHAPTER 3 ***Exploring data*** > summary(Age)

y

t

is

n

e

d

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.03

–3.983 25.270 61.400 50.690 75.930 82.230

> summary(custdata$age)

| Min. 1st Qu. Median 38.0 50.0 | Mean 3rd Qu. Max. 51.7 64.0 146.7 |  |
| --- | --- | --- |
|  | “Average”  customer–but  not “typical”  customer! |  |

0.0

0.02

0.01

0.00

0 25 50 75 100 age

Figure 3.2 A unimodal distribution (gray) can usually be modeled as coming from a single population of users. With a bimodal distribution (black), your data often comes from two populations of users.

The black curve in figure 3.2 shows what can happen when you have two peaks, or a *bimodal distribution*. (A distribution with more than two peaks is *multimodal*.) This set of customers has about the same mean age as the customers represented by the gray curve—but a 50-year-old is hardly a “typical” customer! This (admittedly exaggerated) example corresponds to two populations of customers: a fairly young population mostly in their 20s and 30s, and an older population mostly in their 70s. These two populations probably have very different behavior patterns, and if you want to model whether a customer probably has health insurance or not, it wouldn’t be a bad idea to model the two populations separately—especially if you’re using linear or logistic regression.

The histogram and the density plot are two visualizations that help you quickly examine the distribution of a numerical variable. Figures 3.1 and 3.2 are density plots. Whether you use histograms or density plots is largely a matter of taste. We tend to prefer density plots, but histograms are easier to explain to less quantitatively-minded audiences.

HISTOGRAMS

A basic histogram bins a variable into fixed-width buckets and returns the number of data points that falls into each bucket. For example, you could group your customers by age range, in intervals of five years: 20–25, 25–30, 30–35, and so on. Customers at a

www.it-ebooks.info

t

nu

oc

100

80

60

40

20

Invalid

***Spotting problems using graphics and visualization* 45**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

~~values Outliers~~ 0

0 50 100 150 age

Figure 3.3 A histogram tells you where your data is concentrated. It also visually highlights outliers and anomalies.

boundary age would go into the higher bucket: 25-year-olds go into the 25–30 bucket. For each bucket, you then count how many customers are in that bucket. The result ing histogram is shown in figure 3.3.

You create the histogram in figure 3.3 in ggplot2 with the geom\_histogram layer.

Listing 3.6 Plotting a histogram

**The binwidth parameter tells**

**the geom\_histogram call how to**

library(ggplot2)

ggplot(custdata) +

geom\_histogram(aes(x=age), binwidth=5, fill="gray")

**Load the ggplot2 library, if you haven’t already done so.**

**make bins of five-year intervals (default is datarange/30). The fill parameter specifies the color of**

**the histogram bars (default: black).**

The primary disadvantage of histograms is that you must decide ahead of time how wide the buckets are. If the buckets are too wide, you can lose information about the shape of the distribution. If the buckets are too narrow, the histogram can look too noisy to read easily. An alternative visualization is the density plot.

DENSITY PLOTS

You can think of a *density plot* as a “continuous histogram” of a variable, except the area under the density plot is equal to 1. A point on a density plot corresponds to the

www.it-ebooks.info

**46** CHAPTER 3 ***Exploring data***

y

t

is

n

e

d

1e~~0~~5 5e~~0~~6 0e+00

| Most of the distribution is concentrated atthe low end: less than $100,000 a year.It’s hard to get good resolution here. |  |
| --- | --- |
|  | Subpopulation  of wealthy |

customers in

the $400,000

range.

Wide data range: several orders of magnitude.

$0 $200,000 $400,000 $600,000 income

Figure 3.4 Density plots show where data is concentrated. This plot also highlights a population of higher-income customers.

fraction of data (or the percentage of data, divided by 100) that takes on a particular value. This fraction is usually very small. When you look at a density plot, you’re more interested in the overall shape of the curve than in the actual values on the y-axis. You’ve seen the density plot of age; figure 3.4 shows the density plot of income. You produce figure 3.4 with the geom\_density layer, as shown in the following listing.

Listing 3.7 Producing a density plot

library(scales)

ggplot(custdata) + geom\_density(aes(x=income)) + scale\_x\_continuous(labels=dollar)

**The scales package brings in the dollar scale notation.**

**Set the x-axis labels to dollars.**

When the data range is very wide and the mass of the distribution is heavily concen trated to one side, like the distribution in figure 3.4, it’s difficult to see the details of its shape. For instance, it’s hard to tell the exact value where the income distribution has its peak. If the data is non-negative, then one way to bring out more detail is to plot the distribution on a logarithmic scale, as shown in figure 3.5. This is equivalent to plotting the density plot of log10(income).

www.it-ebooks.info

y

t

is

n

e

d

***Spotting problems using graphics and visualization* 47**

1.00

Peak of income

distribution at ~$40,000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Most customers have income in the$20,000–$100,000 range.  More customers have income in the$10,000 range than you would expect. |  |  | Customers with incomeover $200,000 are rare, |
|  |  |  |  | but they no longer looklike “outliers” in logspace. |

0.75

0.50

0.25

Very-low-income outliers

0.00

$100 $1,000 $10,000 $100,000

income

Figure 3.5 The density plot of income on a log10 scale highlights details of the income distribution that are harder to see in a regular density plot.

In ggplot2, you can plot figure 3.5 with the geom\_density and scale\_x\_log10 layers, such as in the next listing.

Listing 3.8 Creating a log-scaled density plot

**Set the x-axis to be in log10 scale, with manually**

**set tick points and labels as dollars.**

ggplot(custdata) + geom\_density(aes(x=income)) +

scale\_x\_log10(breaks=c(100,1000,10000,100000), labels=dollar) + annotation\_logticks(sides="bt")

**Add log-scaled tick marks to the**

**top and bottom of the graph.**

When you issued the preceding command, you also got back a warning message:

Warning messages:

1: In scale$trans$trans(x) : NaNs produced

2: Removed 79 rows containing non-finite values (stat\_density).

This tells you that ggplot2 ignored the zero- and negative-valued rows (since log(0) = Infinity), and that there were 79 such rows. Keep that in mind when evaluating the graph.

In log space, income is distributed as something that looks like a “normalish” distri bution, as will be discussed in appendix B. It’s not exactly a normal distribution (in fact, it appears to be at least two normal distributions mixed together).

www.it-ebooks.info

**48** CHAPTER 3 ***Exploring data***

| When should you use a logarithmic scale?  You should use a logarithmic scale when percent change, or change in orders of mag nitude, is more important than changes in absolute units. You should also use a log scale to better visualize data that is heavily skewed.  For example, in income data, a difference in income of five thousand dollars means something very different in a population where the incomes tend to fall in the tens of thousands of dollars than it does in populations where income falls in the hundreds of thousands or millions of dollars. In other words, what constitutes a “significant dif ference” depends on the order of magnitude of the incomes you’re looking at. Simi larly, in a population like that in figure 3.5, a few people with very high income will cause the majority of the data to be compressed into a relatively small area of the graph. For both those reasons, plotting the income distribution on a logarithmic scale is a good idea. |
| --- |

BAR CHARTS

A *bar chart* is a histogram for discrete data: it records the frequency of every value of a categorical variable. Figure 3.6 shows the distribution of marital status in your cus tomer dataset. If you believe that marital status helps predict the probability of health insurance coverage, then you want to check that you have enough customers with dif ferent marital statuses to help you discover the relationship between being married (or not) and having health insurance.

500

400

300

t

n

u

o

c

200

100

0

Divorced/Separated Married Never Married Widowed

marital.stat

Figure 3.6 Bar charts show the distribution of categorical variables.

www.it-ebooks.info

**Flip the**

***Spotting problems using graphics and visualization* 49**

The ggplot2 command to produce figure 3.6 uses geom\_bar:

ggplot(custdata) + geom\_bar(aes(x=marital.stat), fill="gray")

This graph doesn’t really show any more information than summary(custdata$marital .stat) would show, although some people find the graph easier to absorb than the text. Bar charts are most useful when the number of possible values is fairly large, like state of residence. In this situation, we often find that a horizontal graph is more legible than a vertical graph.

The ggplot2 command to produce figure 3.7 is shown in the next listing. Listing 3.9 Producing a horizontal bar chart

**Plot bar chart as before:**

**x and y**

**axes:**

**state.of.res is now on**

ggplot(custdata) +

geom\_bar(aes(x=state.of.res), fill="gray") + coord\_flip() +

theme(axis.text.y=element\_text(size=rel(0.8)))

**state.of.res is on x axis, count is on y-axis.**

**the y-axis. Reduce the size of the y-axis**

**tick labels to 80% of default**

**size for legibility.**

Wyoming

Wisconsin

West Virginia

Virginia Washington

Vermont

Utah

Texas

Tennessee

South Dakota

South Carolina

Rhode Island

Oklahoma Oregon Pennsylvania

Ohio

North Dakota

North Carolina

New York

New Mexico

s

e

r

.

f

o

.

e

t

a

ts

Nevada New Hampshire New Jersey

Nebraska

Montana

Missouri

Mississippi

Minnesota

Massachusetts Michigan

Maine Maryland

Louisiana

Kentucky

Kansas

Iowa

Indiana

Illinois

Idaho

Hawaii

Georgia

Florida

Delaware

Connecticut

Colorado

California

Arkansas

Arizona

Alaska

Alabama

0 25 50 75 100 count

Figure 3.7 A horizontal bar chart can be easier to read when there are several categories with long names. www.it-ebooks.info

**50** CHAPTER 3 ***Exploring data***

California

New York

Texas Pennsylvania

Ohio Michigan

Illinois

Florida

Virginia New Jersey

Indiana

Wisconsin

Massachusetts Georgia

Tennessee

Missouri

Minnesota

North Carolina Maryland Washington

s

e

r

.

f

o

.

e

t

a

ts

South Carolina Kentucky

Louisiana

Connecticut

Oklahoma West Virginia

Colorado

Alabama

Iowa

Arizona

Nebraska

Mississippi Oregon

Arkansas

New Mexico

Maine

Utah

South Dakota

Hawaii New Hampshire

Nevada

Kansas

Idaho

Vermont

Montana

Alaska

Rhode Island

North Dakota Wyoming

Delaware

0 25 50 75 100 count

Figure 3.8 Sorting the bar chart by count makes it even easier to read.

Cleveland3 recommends that the data in a bar chart (or in a *dot plot*, Cleveland’s pre ferred visualization in this instance) be sorted, to more efficiently extract insight from the data. This is shown in figure 3.8.

This visualization requires a bit more manipulation, at least in ggplot2, because by default, ggplot2 will plot the categories of a factor variable in alphabetical order. To change this, we have to manually specify the order of the categories—in the factor variable, not in ggplot2.

Listing 3.10 Producing a bar chart with sorted categories

**Rename the columns for readability.**

> statesums <- table(custdata$state.of.res) > statef <- as.data.frame(statesums) > colnames(statef)<-c("state.of.res", "count")

> summary(statef)

**Notice that the default ordering for the**

**state.of.res variable is alphabetical.**

**Convert the table object to a data frame using**

**as.data.frame(). The default**

**column names are Var1 and Freq.**

**The table()**

**command**

**aggregates the data by state of residence—**

**exactly the**

**information the bar chart plots.**

3 See William S. Cleveland, *The Elements of Graphing Data*, Hobart Press, 1994. www.it-ebooks.info

***Spotting problems using graphics and visualization* 51**

state.of.res count

Alabama : 1 Min. : 1.00

Alaska : 1 1st Qu.: 5.00

Arizona : 1 Median : 12.00

Arkansas : 1 Mean : 20.00

California: 1 3rd Qu.: 26.25

Colorado : 1 Max. :100.00

(Other) :44

> statef <- transform(statef,

state.of.res=reorder(state.of.res, count)) > summary(statef)

state.of.res count

Delaware : 1 Min. : 1.00 North Dakota: 1 1st Qu.: 5.00 Wyoming : 1 Median : 12.00 Rhode Island: 1 Mean : 20.00 Alaska : 1 3rd Qu.: 26.25 Montana : 1 Max. :100.00 (Other) :44

**Use the reorder() function to set the state.of.res**

**variable to be count**

**ordered. Use the**

**transform() function to**

**apply the transformation to the state.of.res data frame.**

**The state.of.res**

**variable is now**

**count ordered.**

**Since the data is being**

**passed to geom\_bar pre**

**aggregated, specify both**

**the x and y variables,**

> ggplot(statef)+ geom\_bar(aes(x=state.of.res,y=count), stat="identity",

**and use stat="identity" to plot the data exactly as given.**

fill="gray") +

coord\_flip() +

theme(axis.text.y=element\_text(size=rel(0.8)))

**Flip the axes and reduce the size of the label text as before.**

Before we move on to visualizations for two variables, in table 3.1 we’ll summarize the visualizations that we’ve discussed in this section.

Table 3.1 Visualizations for one variable

Graph type Uses

Histogram or density plot

Examines data range

Checks number of modes

Checks if distribution is normal/lognormal Checks for anomalies and outliers

Bar chart Compares relative or absolute frequencies of the values of a categorical variable

*3.2.2 Visually checking relationships between two variables*

In addition to examining variables in isolation, you’ll often want to look at the relation ship between two variables. For example, you might want to answer questions like these:

www.it-ebooks.info

**52** CHAPTER 3 ***Exploring data***

◼ Is there a relationship between the two inputs *age* and *income* in my data? ◼ What kind of relationship, and how strong?

◼ Is there a relationship between the input *marital status* and the output *health insurance*? How strong?

You’ll precisely quantify these relationships during the modeling phase, but exploring them now gives you a feel for the data and helps you determine which variables are the best candidates to include in a model.

First, let’s consider the relationship between two continuous variables. The most obvious way (though not always the best) is the line plot.

LINE PLOTS

*Line plots* work best when the relationship between two variables is relatively clean: each *x* value has a unique (or nearly unique) *y* value, as in figure 3.9. You plot figure 3.9 with geom\_line.

Listing 3.11 Producing a line plot

**First, generate the data for this example. The x variable**

**Plot the line**

x <- runif(100) y <- x^2 + 0.2\*x

**is uniformly randomly distributed between 0 and 1. The y variable is a**

**plot.**

ggplot(data.frame(x=x,y=y), aes(x=x,y=y)) + geom\_line() 1.25

**quadratic function of x.**

y

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

1.00

0.75

0.50

0.25

0.00

0.00 0.25 0.50 0.75 1.00 x

Figure 3.9 Example of a line plot

www.it-ebooks.info

***Spotting problems using graphics and visualization* 53**

When the data is not so cleanly related, line plots aren’t as useful; you’ll want to use the scatter plot instead, as you’ll see in the next section.

SCATTER PLOTS AND SMOOTHING CURVES

You’d expect there to be a relationship between age and health insurance, and also a relationship between income and health insurance. But what is the relationship between age and income? If they track each other perfectly, then you might not want to use both variables in a model for health insurance. The appropriate summary statis tic is the correlation, which we compute on a safe subset of our data.

Listing 3.12 Examining the correlation between age and income

custdata2 <- subset(custdata,

(custdata$age > 0 & custdata$age < 100 & custdata$income > 0))

cor(custdata2$age, custdata2$income)

**Only consider a subset of**

**data with reasonable age**

**and income values.**

**Get correlation of age and income.**

[1] -0.02240845

**Resulting correlation.**

The negative correlation is surprising, since you’d expect that income should increase as people get older. A visualization gives you more insight into what’s going on than a single number can. Let’s try a scatter plot first; you plot figure 3.10 with geom\_point:

ggplot(custdata2, aes(x=age, y=income)) +

geom\_point() + ylim(0, 200000)

e

m

o

c

n

i

200000 150000 100000

50000 0

Income tends to increase in this range.

|  |  | And it tends to decrease in this range. |
| --- | --- | --- |
|  |  |
|  | But the relationship is hard to see. |  |
|  |  |  |
|  |  |  |

20 40 60 80 age

Figure 3.10 A scatter plot of income versus age www.it-ebooks.info