# Contents

List of Tables  
List of Figures  
Preface  
Preface to the Original Book  

## 1 Introduction

1.1 Overview of the Book  
1.2 How to Use This Book  
1.3 Introduction to R and the tidyverse  
  1.3.1 Arithmetic Operations: R as a Calculator  
  1.3.2 R Scripts  
  1.3.3 Loading Packages  
  1.3.4 Objects  
  1.3.5 Vectors  
  1.3.6 Functions  
  1.3.7 Data Files: Loading and Subsetting  
  1.3.8 Adding Variables  
  1.3.9 Data Frames: Summarizing  
  1.3.10 Saving Objects  
  1.3.11 Loading Data in Other Formats  
  1.3.12 Programming and Learning Tips  
1.4 Summary  
1.5 Exercises  
  1.5.1 Bias in Self-Reported Turnout  
  1.5.2 Understanding World Population Dynamics  

## 2 Causality

2.1 Racial Discrimination in the Labor Market  
2.2 Subsetting Data in R  
  2.2.1 Logical Values and Operators  
  2.2.2 Relational Operators  
  2.2.3 Subsetting
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.4 Simple Conditional Statements</td>
<td>53</td>
</tr>
<tr>
<td>2.2.5 Factor Variables</td>
<td>53</td>
</tr>
<tr>
<td>2.3 Causal Effects and the Counterfactual</td>
<td>56</td>
</tr>
<tr>
<td>2.4 Randomized Controlled Trials</td>
<td>58</td>
</tr>
<tr>
<td>2.4.1 The Role of Randomization</td>
<td>59</td>
</tr>
<tr>
<td>2.4.2 Social Pressure and Voter Turnout</td>
<td>60</td>
</tr>
<tr>
<td>2.5 Observational Studies</td>
<td>65</td>
</tr>
<tr>
<td>2.5.1 Minimum Wage and Unemployment</td>
<td>65</td>
</tr>
<tr>
<td>2.5.2 Confounding Bias</td>
<td>68</td>
</tr>
<tr>
<td>2.5.3 Before-and-After and Difference-in-Differences Designs</td>
<td>71</td>
</tr>
<tr>
<td>2.6 Descriptive Statistics for a Single Variable</td>
<td>75</td>
</tr>
<tr>
<td>2.6.1 Quantiles</td>
<td>75</td>
</tr>
<tr>
<td>2.6.2 Standard Deviation</td>
<td>78</td>
</tr>
<tr>
<td>2.7 Summary</td>
<td>81</td>
</tr>
<tr>
<td>2.8 Exercises</td>
<td>82</td>
</tr>
<tr>
<td>2.8.1 Efficacy of Small Class Size in Early Education</td>
<td>82</td>
</tr>
<tr>
<td>2.8.2 Changing Minds on Gay Marriage</td>
<td>84</td>
</tr>
<tr>
<td>2.8.3 Success of Leader Assassination as a Natural Experiment</td>
<td>85</td>
</tr>
</tbody>
</table>

## 3 MEASUREMENT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Measuring Civilian Victimization during Wartime</td>
<td>88</td>
</tr>
<tr>
<td>3.2 Handling Missing Data in R</td>
<td>93</td>
</tr>
<tr>
<td>3.3 Visualizing the Univariate Distribution</td>
<td>96</td>
</tr>
<tr>
<td>3.3.1 Bar Plot</td>
<td>97</td>
</tr>
<tr>
<td>3.3.2 Histogram</td>
<td>100</td>
</tr>
<tr>
<td>3.3.3 Box Plot</td>
<td>103</td>
</tr>
<tr>
<td>3.3.4 Printing and Saving Graphs</td>
<td>105</td>
</tr>
<tr>
<td>3.4 Survey Sampling</td>
<td>106</td>
</tr>
<tr>
<td>3.4.1 The Role of Randomization</td>
<td>107</td>
</tr>
<tr>
<td>3.4.2 Nonresponse and Other Sources of Bias</td>
<td>111</td>
</tr>
<tr>
<td>3.5 Measuring Political Polarization</td>
<td>114</td>
</tr>
<tr>
<td>3.6 Summarizing Bivariate Relationships</td>
<td>116</td>
</tr>
<tr>
<td>3.6.1 Scatter Plot</td>
<td>116</td>
</tr>
<tr>
<td>3.6.2 Correlation</td>
<td>120</td>
</tr>
<tr>
<td>3.7 Quantile–Quantile Plot</td>
<td>124</td>
</tr>
<tr>
<td>3.8 Clustering</td>
<td>128</td>
</tr>
<tr>
<td>3.8.1 Matrix in R</td>
<td>128</td>
</tr>
<tr>
<td>3.8.2 List in R</td>
<td>130</td>
</tr>
<tr>
<td>3.8.3 The k-Means Algorithm</td>
<td>131</td>
</tr>
<tr>
<td>3.9 Summary</td>
<td>136</td>
</tr>
<tr>
<td>3.10 Exercises</td>
<td>137</td>
</tr>
<tr>
<td>3.10.1 Changing Minds on Gay Marriage: Revisited</td>
<td>137</td>
</tr>
<tr>
<td>3.10.2 Political Efficacy in China and Mexico</td>
<td>139</td>
</tr>
<tr>
<td>3.10.3 Voting in the United Nations General Assembly</td>
<td>141</td>
</tr>
</tbody>
</table>
## 4 PREDICTION

4.1 Predicting Election Outcomes  
4.1.1 Loops in R  
4.1.2 General Conditional Statements in R  
4.1.3 Poll Predictions  
4.2 Linear Regression  
4.2.1 Facial Appearance and Election Outcomes  
4.2.2 Correlation and Scatter Plots  
4.2.3 Least Squares  
4.2.4 Regression towards the Mean  
4.2.5 Merging Data Sets in R  
4.2.6 Model Fit  
4.3 Regression and Causation  
4.4 Randomized Experiments  
4.4.1 Regression with Multiple Predictors  
4.4.2 Heterogeneous Treatment Effects  
4.4.3 Regression Discontinuity Design  
4.5 Summary  
4.6 Exercises  
4.6.1 Prediction Based on Betting Markets  
4.6.2 Election and Conditional Cash Transfer Program in Mexico  
4.6.3 Government Transfer and Poverty Reduction in Brazil

## 5 DISCOVERY

5.1 Textual Data  
5.1.1 The Disputed Authorship of *The Federalist Papers*  
5.1.2 Document-Term Matrix  
5.1.3 Topic Discovery  
5.1.4 Authorship Prediction  
5.1.5 Cross-Validation  
5.2 Network Data  
5.2.1 Marriage Network in Renaissance Florence  
5.2.2 Undirected Graph and Centrality Measures  
5.2.3 Twitter-Following Network  
5.2.4 Directed Graph and Centrality  
5.3 Spatial Data  
5.3.1 The 1854 Cholera Outbreak in London  
5.3.2 Spatial Data in R  
5.3.3 US Presidential Elections  
5.3.4 Expansion of Walmart  
5.3.5 Animation in R  
5.4 Summary  
5.5 Exercises  
5.5.1 Analyzing the Preambles of Constitutions  
5.5.2 International Trade Network  
5.5.3 Mapping US Presidential Election Results over Time
6 PROBABILITY 279

6.1 Probability 279
   6.1.1 Frequentist versus Bayesian 279
   6.1.2 Definition and Axioms 281
   6.1.3 Permutations 284
   6.1.4 Sampling with and without Replacement 287
   6.1.5 Combinations 289

6.2 Conditional Probability 291
   6.2.1 Conditional, Marginal, and Joint Probabilities 291
   6.2.2 Independence 301
   6.2.3 Bayes’ Rule 307
   6.2.4 Predicting Race Using Surname and Residence Location 309

6.3 Random Variables and Probability Distributions 321
   6.3.1 Random Variables 321
   6.3.2 Bernoulli and Uniform Distributions 321
   6.3.3 Binomial Distribution 325
   6.3.4 Normal Distribution 328
   6.3.5 Expectation and Variance 335
   6.3.6 Predicting Election Outcomes with Uncertainty 339

6.4 Large Sample Theorems 342
   6.4.1 The Law of Large Numbers 342
   6.4.2 The Central Limit Theorem 345

6.5 Summary 350

6.6 Exercises 350
   6.6.1 The Mathematics of Enigma 350
   6.6.2 A Probability Model for Betting Market Election Prediction 352
   6.6.3 Election Fraud in Russia 354

7 UNCERTAINTY 357

7.1 Estimation 357
   7.1.1 Unbiasedness and Consistency 358
   7.1.2 Standard Error 366
   7.1.3 Confidence Interval 371
   7.1.4 Margin of Error and Sample Size Calculation in Polls 378
   7.1.5 Analysis of Randomized Controlled Trials 383
   7.1.6 Analysis Based on Student’s t-Distribution 386

7.2 Hypothesis Testing 390
   7.2.1 Tea-Tasting Experiment 390
   7.2.2 The General Framework 394
   7.2.3 One-Sample Tests 397
   7.2.4 Two-Sample Tests 404
   7.2.5 Pitfalls of Hypothesis Testing 409
   7.2.6 Power Analysis 411

7.3 Linear Regression Model with Uncertainty 418
   7.3.1 Linear Regression as a Generative Model 418
   7.3.2 Unbiasedness of Estimated Coefficients 423
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.3 Standard Errors of Estimated Coefficients</td>
<td>426</td>
</tr>
<tr>
<td>7.3.4 Inference about Coefficients</td>
<td>428</td>
</tr>
<tr>
<td>7.3.5 Inference about Predictions</td>
<td>432</td>
</tr>
<tr>
<td>7.4 Summary</td>
<td>439</td>
</tr>
<tr>
<td>7.5 Exercises</td>
<td>439</td>
</tr>
<tr>
<td>7.5.1 Sex Ratio and the Price of Agricultural Crops in China</td>
<td>439</td>
</tr>
<tr>
<td>7.5.2 Filedrawer and Publication Bias in Academic Research</td>
<td>441</td>
</tr>
<tr>
<td>7.5.3 Analysis of the 1933 German Election during the Weimar Republic</td>
<td>443</td>
</tr>
</tbody>
</table>

### 8 NEXT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Index</td>
<td>449</td>
</tr>
<tr>
<td>R Index</td>
<td>455</td>
</tr>
</tbody>
</table>
Quantitative social science is an interdisciplinary field encompassing a large number of disciplines, including economics, education, political science, public policy, psychology, and sociology. In quantitative social science research, scholars analyze data to understand and solve problems about society and human behavior. Such research projects range from the examination of racial discrimination in the labor market to the impact evaluation of new curricula on students’ educational achievements, and from the prediction of election outcomes to the analysis of social media usage. A similar data-driven approach has been taken up in other neighboring fields such as health, law, journalism, linguistics, and even literature. Because social scientists directly investigate a wide range of real-world issues, the results of their research have enormous potential to directly influence individual members of society, government policies, and business practices.

Over the last couple of decades, quantitative social science has flourished in a variety of areas at an astonishing speed. The number of academic journal articles that present empirical evidence from data analysis has soared. Outside of academia, many organizations—including corporations, political campaigns, news media, and government agencies—increasingly rely on data analysis in their decision-making processes. Two transformative technological changes have driven this rapid growth of quantitative social science. First, the internet has greatly facilitated the data revolution, a spike in the amount and diversity of available data, through information sharing, making it possible for researchers and organizations to disseminate numerous data sets in digital form. Second, the computational revolution, in terms of both software and hardware, means that essentially anyone can conduct data analysis using their personal computer and favorite data analysis software, without needing to access expensive computational facilities.

As a direct consequence of these technological changes, the sheer volume of data available to quantitative social scientists has rapidly grown. In the past, most researchers relied upon data published by governmental agencies (e.g., censuses, election outcomes, and economic indicators) as well as a small number of data sets collected by groups of researchers (e.g., survey data from national election studies and hand-coded data sets about war occurrence and democratic institutions). These data sets still play an important role in empirical analysis.

In God we trust; all others must bring data.
—William Edwards Deming
However, the wide variety of new data has significantly expanded the horizon of quantitative social science research. Researchers are now designing and conducting randomized experiments and surveys on their own. Under pressure to increase transparency and accountability, government agencies are making more data publicly available online. For example, in the United States, anyone can download detailed data on campaign contributions and lobbying activities to their personal computers. In Nordic countries like Sweden, a wide range of registers, including income, tax, education, health, and workplace, are used for academic research.

New data sets have emerged across diverse areas. For example, detailed data about consumer transactions are available through electronic purchasing records. International trade data are now collected at the product level between many pairs of countries over several decades. Militaries have also contributed to the data revolution. During the recent war in Afghanistan, the United States and international forces gathered data on the geolocation, timing, and types of insurgent attacks and conducted data analysis to guide counterinsurgency strategy. Similarly, governmental agencies and nongovernmental organizations collected data on civilian casualties from the war. Political campaigns use data analysis to devise voter mobilization strategies by targeting certain types of voters with carefully selected messages.

These data sets also come in varying forms. Quantitative social scientists are now analyzing digitized texts as data, including legislative bills, newspaper articles, and speeches. The availability of social media data through websites, blogs, tweets, SMS messaging, and Facebook has enabled social scientists to explore how people interact with one another in the online sphere. Geographical information system (GIS) data sets are also widespread, enabling, for example, researchers to analyze the legislative redistricting process and its outcomes. Others have used satellite imagery data to measure the level of electrification in rural areas of developing countries. Images, sounds, and even videos can now be analyzed using quantitative methods to answer social science questions.

Together with the revolution of information technology, the availability of such abundant and diverse data means that anyone, from academics to practitioners, from business analysts to policy makers, and from students to faculty, can make data-driven discoveries. In the past, only statisticians and other specialized professionals conducted data analysis. Now, everyone can turn on their personal computer, download data from the internet, and analyze them using their favorite software. This has led to increased demands for accountability to demonstrate policy effectiveness. In order to secure funding and increase legitimacy, for example, nongovernmental organizations and governmental agencies must now demonstrate the efficacy of their policies and programs through rigorous evaluation.

This shift towards greater transparency and data-driven discovery requires that students in the social sciences learn how to analyze data, interpret the results, and effectively communicate their empirical findings. Traditionally, introductory statistics courses focused on teaching students basic statistical concepts by having them conduct straightforward calculations with paper and pencil or, at best, a scientific calculator. Although these concepts are still important and covered in this book, in the current day and age this traditional approach cannot meet the demands of society. It is simply not sufficient to achieve "statistical literacy" by learning about common statistical concepts and methods. Instead, all students in the social sciences should acquire basic data analysis skills so that they can exploit ample opportunities to learn from data and make contributions to society through data-driven discovery.

The belief that everyone should be able to analyze data is the main motivation for the writing of this book. The book introduces three elements of data analysis required for quantitative
social science research: research contexts, programming techniques, and statistical methods. Any of these elements in isolation is insufficient. Without research contexts, we cannot assess the credibility of assumptions required for data analysis and will not be able to understand what the empirical findings imply. Without programming techniques, we will not be able to analyze data and answer research questions. Without the guidance of statistical principles, we cannot distinguish systematic patterns, known as signals, from idiosyncratic ones, known as noise, possibly leading to invalid inference. Here, inference refers to drawing conclusions about unknown quantities based on observed data. This book demonstrates the power of data analysis by combining these three elements.

### 1.1 Overview of the Book

This book is written for anyone who wishes to learn data analysis and statistics for the first time. The target audience includes researchers and undergraduate and graduate students in social science and other fields, as well as practitioners and even ambitious high-school students. The book has no prerequisite other than some elementary algebra. In particular, readers do not have to possess knowledge of calculus or probability. No programming experience is necessary, though it can certainly be helpful. The book is also appropriate for those who have taken a traditional “paper-and-pencil” introductory statistics course where little data analysis is taught. Through this book, these students will discover the excitement that data analysis brings. Those who want to learn R programming might also find this book useful, although here the emphasis is on how to apply R to quantitative social science research.

As mentioned above, the unique feature of this book is the presentation of programming techniques and statistical concepts simultaneously through analysis of data sets taken directly from published quantitative social science research. The goal is to demonstrate how social scientists use data analysis to answer important questions about problems of society and human behavior. At the same time, users of the book will learn fundamental statistical concepts and basic programming skills. Most importantly, readers will gain experience with data analysis by examining approximately forty data sets.

The book consists of eight chapters. The current introductory chapter explains how to best utilize the book and presents a brief introduction to R, a popular open-source statistical programming environment. R is freely available for download and runs on Macintosh, Windows, and Linux computers. Readers are strongly encouraged to use RStudio, another freely available software package that has numerous features to make data analysis easier. This chapter ends with two exercises, which are designed to have readers practice elementary R functionalities using data sets from published social science research. All code and data sets used in this book are freely available for download at https://github.com/kosukeimai/qss. This website also provides other useful materials, such as the review exercises for each chapter.

The original version of this textbook focused on the most basic syntax of R, which is often referred to as “base R.” This book instead relies heavily on what is called the “tidyverse” (specifically version 1.3.1 of the tidyverse).¹ We will cover more on what that means as we introduce R. However, it is worth noting now that there are often many different ways to write code in R that produces the same results. You can think of the tidyverse as being a dialect or

¹For more detail, see Hadley Wickham et al. (2019) “Welcome to the tidyverse.” *Journal of Open Source Software*, vol. 4, no. 43, p. 1686, https://doi.org/10.21105/joss.01686. We are deeply indebted to an earlier contribution by Jeffrey Arnold (with additions from Nora Webb Williams and Calvin Garner), which is available at https://github.com/jrnold/qss-tidy.
specific syntax of R programming. It is another (sometimes more efficient or elegant) way of asking for the same thing from R as base R. This book is not a complete introduction to the tidyverse. We recommend R for Data Science by Hadley Wickham and Garrett Grolemond for additional reference.

Chapter 2 introduces causality, which plays an essential role in social science research whenever we wish to find out whether a particular policy or program changes an outcome of interest. Causality is notoriously difficult to study because we must infer counterfactual outcomes that are not observable. For example, in order to understand the existence of racial discrimination in the labor market, we need to know whether or not an African American candidate who did not receive a job offer would have done so if they were White. We will analyze the data from a well-known experimental study in which researchers sent the resumes of fictitious job applicants to potential employers after randomly choosing the applicants’ names to sound either African American or White. Using this study as an application, the chapter will explain how the randomization of treatment assignment enables researchers to identify the average causal effect of the treatment.

In chapter 2, readers will also learn about causal inference in observational studies where researchers do not have control over treatment assignment. The main application is a classic study of the relationship between minimum wage and employment. The goal of the study was to figure out the impact of increasing minimum wage on employment. Many economists argue that a minimum wage increase can reduce employment because employers must pay higher wages to their workers and are therefore made to hire fewer workers. Unfortunately, the decision to increase the minimum wage is not random, instead subject to many factors, like economic growth, that are themselves associated with employment. Since these factors influence which companies select themselves into the treatment group, a simple comparison between those who received treatment and those who did not can lead to biased inference.

We introduce several strategies that attempt to reduce this selection bias in observational studies. Despite the risk that we will inaccurately estimate treatment effects in observational studies, their results are often easier to generalize than those obtained from randomized controlled trials. Other examples in this chapter include field experiments concerning social pressure in get-out-the-vote mobilization. In terms of R programming, chapter 2 covers logical statements and subsetting. The exercises include a randomized experiment that investigates the impact of small class size in early education and an observational study about the assassination of political leaders.

Chapter 3 introduces the fundamental concept of measurement. Accurate measurement is important for any data-driven discovery because bias in measurement can lead to incorrect conclusions and misguided decisions. We begin by considering how to measure public opinion through sample surveys. We analyze the data from a study in which researchers attempt to measure the degree of support among Afghan citizens for international forces and the Taliban insurgency during the war in Afghanistan. We explain the power of randomization in survey sampling. Specifically, random sampling of respondents from a population allows us to obtain a representative sample of this population. As a result, we can infer the opinion of an entire population by analyzing one small representative sample. We also discuss the potential biases of survey sampling. Nonresponses can compromise the representativeness of a sample. Misreporting poses a serious threat to inference, especially when respondents are asked sensitive questions, such as whether they support the Taliban insurgency.
The second half of chapter 3 focuses on the measurement of latent or unobservable concepts that play a key role in quantitative social science. Prominent examples of such concepts include ability and ideology. In the chapter we study political ideology. We first describe a model frequently used to infer ideological positions of legislators from roll call votes and examine how the US Congress has polarized over time. We then introduce a basic clustering algorithm, \( k \)-means, that makes it possible for us to find groups of similar observations. Applying this algorithm to the data, we find that in recent years, the ideological division within Congress has been mainly characterized by the party line. In contrast, we find some divisions within each party in earlier years. This chapter also introduces various measures of the spread of data, including quantiles, standard deviation, and the Gini coefficient. In terms of \( R \) programming, the chapter introduces various ways to visualize univariate and bivariate data. The exercises include the reanalysis of a controversial same-sex marriage experiment, which raises issues of academic integrity while illustrating methods covered in the chapter.

Chapter 4 considers prediction. Predicting the occurrence of certain events is an essential component of policy- and decision-making processes. For example, the forecasting of economic performance is critical for fiscal planning, and early warnings of civil unrest allow foreign policy makers to act proactively. The main application of this chapter is the prediction of US presidential elections using pre-election polls. We show that we can make a remarkably accurate prediction by combining multiple polls in a straightforward manner. In addition, we analyze the data from a psychological experiment in which subjects are shown the facial pictures of unknown political candidates and asked to rate their competence. The analysis yields the surprising result that a quick facial impression can predict election outcomes. Through this example, we introduce linear regression models, which are useful tools to predict the values of one variable based on another variable. We describe the relationship between linear regression and correlation, and examine the phenomenon called “regression towards the mean,” which is the origin of the term “regression.”

Chapter 4 also discusses when regression models can be used to estimate causal effects rather than simply make predictions. Causal inference differs from standard prediction in requiring the prediction of counterfactual, rather than observed, outcomes using the treatment variable as the predictor. We analyze the data from a randomized natural experiment in India where randomly selected villages reserved some of the seats in their village councils for women. Exploiting this randomization, we investigate whether or not having female politicians affects policy outcomes, especially concerning the policy issues female voters care about. The chapter also introduces the regression discontinuity design for making causal inference in observational studies. We investigate how much of British politicians’ accumulated wealth is due to holding political office. We answer this question by comparing those who barely won an election with those who narrowly lost it. The chapter introduces powerful but challenging \( R \) programming concepts, loops and conditional statements. The exercises at the end of the chapter include an analysis of whether betting markets can precisely forecast election outcomes.

Chapter 5 is about the discovery of patterns from data of various types. When analyzing “big data,” we need some automated methods and visualization tools to identify consistent patterns in the data. First, we analyze texts as data. Our primary application here is the authorship prediction of The Federalist Papers, which formed the basis of the US Constitution. Some of the papers have known authors while others do not. We show that by analyzing
the frequencies of certain words in the papers with known authorship, we can predict whether Alexander Hamilton or James Madison authored each of the papers with unknown authorship. Second, we show how to analyze network data, which record information about the relationships among units. Within marriage networks in Renaissance Florence, we quantify the key role played by the Medici family. Various measures of centrality are introduced and applied to a Twitter network, an example of social media data.

Finally, chapter 5 introduces geospatial data. We begin by discussing the classic spatial data analysis conducted by John Snow to examine the cause of the 1854 cholera outbreak in London. We then demonstrate how to visualize spatial data through the creation of maps, using US election data as an example. For spatial–temporal data, we create a series of maps as an animation in order to visually characterize change in spatial patterns over time. Thus, the chapter applies various data visualization techniques using several specialized R packages.

Chapter 6 shifts the focus from data analysis to probability, a unified mathematical model of uncertainty. While earlier chapters focus on how to estimate parameters and make predictions, they do not discuss the level of uncertainty in empirical findings, a topic that chapter 7 introduces. Probability is important because it lays a foundation for statistical inference, the goal of which is to quantify inferential uncertainty. We begin by discussing the question of how to interpret probability from frequentist and Bayesian perspectives. We then provide mathematical definitions of probability and conditional probability, and we introduce several fundamental rules of probability. One such rule is Bayes’ rule. We show how we can use Bayes’ rule to accurately predict individual ethnicity using surname and residence location when no survey data are available.

This chapter also introduces the important concepts of random variables and probability distributions. We use these tools to add a measure of uncertainty to the election predictions we produced using preelection polls in chapter 4. Another exercise adds uncertainty to the forecasts of election outcomes based on betting market data. Finally, chapter 6 introduces two fundamental theorems of probability, the law of large numbers and the central limit theorem. These two theorems are widely applicable and help characterize how our estimates behave as sample size increases and over repeated sampling. The last set of exercises in this chapter are about the German cryptography machine from World War II, Enigma, and the detection of election fraud in Russia.

Chapter 7 discusses how to quantify the uncertainty of our estimates and predictions. In earlier chapters, we introduced various data analysis methods to find patterns in data. Building off the groundwork laid in chapter 6, chapter 7 thoroughly explains how certain we should be about such patterns. This chapter shows how to distinguish signals from noise through the computation of standard errors and confidence intervals, as well as hypothesis testing. In other words, the chapter is concerned with statistical inference. Our examples are drawn from earlier chapters and we focus on measuring the uncertainty of these previously computed estimates. They include the analysis of preelection polls, randomized experiments concerning the effects of class size in early education on students’ performance, and an observational study assessing the effects of a minimum wage increase on employment. When discussing statistical hypothesis tests, we also draw attention to the dangers of multiple testing and publication bias. Finally, we discuss how to quantify the level of uncertainty about the estimates derived from the linear regression model. We revisit the randomized natural experiment of female politicians in India and the regression discontinuity design for estimating the amount of wealth British politicians are able to accumulate by holding political office.
The final chapter concludes by briefly describing the next steps readers might take upon completion of this book. The chapter also discusses the role of data analysis in quantitative social science research.

### 1.2 How to Use This Book

In this section we explain how to use this book. The book is based on the following principle:

One can learn data analysis only by doing, not by reading.

The book is not just for reading. Therefore, the emphasis must be placed on gaining experience in analyzing data. This is best accomplished by trying out the code in the book on one's own, playing with it, and working on various exercises that appear at the end of each chapter. All code and data sets used in the book are freely available for download at https://github.com/kosukeimai/qss/.

The book is cumulative. Materials that appear later in the book assume that readers are already familiar with most of the materials covered in earlier parts of the book. Hence, in general, it is not a good idea to skip chapters. The exception is chapter 5, “Discovery,” the contents of which are not used in subsequent chapters. Nevertheless, this chapter contains some of the most interesting data analysis examples of the book and readers are encouraged to study it.

The book can be used for course instruction in a variety of ways. In a traditional introductory statistics course one can assign the book, or parts of it, as supplementary reading that provides data analysis exercises. In the authors' experience, however, the book is best utilized in a data analysis course where an instructor spends less time on lecturing to students and instead works interactively with students on data analysis exercises in the classroom. In such a course, a relevant portion of the book is assigned prior to each class. In the classroom, the instructor reviews new methodological and programming concepts and then applies them to one of the exercises from the book or any other similar application of their choice. Throughout this process, the instructor can discuss the exercise interactively with students, perhaps using the Socratic method, until the class collectively arrives at a solution. After such a classroom discussion, it would be ideal to follow up with a computer lab session in which a small number of students, together with an instructor, work on another exercise.

This teaching format is consistent with the “particular general particular” principle. This principle states that an instructor should first introduce a particular example to illustrate a new concept, then provide a general treatment of it, and finally apply it to another particular example. The reading assignment introduces a particular example and a general discussion of new concepts to students. The classroom discussion allows the instructor to provide another general treatment of these concepts and then, together with students, apply them to another example. This is an effective teaching strategy that engages students with active learning and builds their ability to conduct data analysis in social science research. Finally, the instructor can assign another application as a problem set to assess whether students mastered the

---

Chapter 1: Introduction

materials. To facilitate this, for each chapter, instructors can obtain, upon request, access to additional exercises and their solutions at a private repository.

In terms of the materials to cover, an example of the course outline for a 15-week semester is given below. We assume that there are approximately two hours of lectures and one hour of computer lab each week. Having hands-on computer lab sessions with a small number of students, in which they learn how to analyze data, is essential:

- Week 1: Introduction (chapter 1)
- Weeks 2–3: Causality (chapter 2)
- Weeks 4–5: Measurement (chapter 3)
- Weeks 6–7: Prediction (chapter 4)
- Weeks 8–9: Discovery (chapter 5)
- Weeks 10–12: Probability (chapter 6)
- Weeks 13–15: Uncertainty (chapter 7)

For a shorter course, there are at least two ways to reduce the materials. One option is to focus on the aspects of “data analysis” and omit statistical inference. Specifically, from the above outline, we can remove chapter 6, “Probability,” and chapter 7, “Uncertainty.” An alternative approach is to skip chapter 5, “Discovery,” which covers the analysis of textual, network, and spatial data, and include the chapters on probability and uncertainty.

1.3 Introduction to R and the tidyverse

This section provides a brief, self-contained introduction to R that is a prerequisite for the remainder of this book. R is an open-source statistical programming environment, which means that anyone can download it for free, examine source code, and make their own contributions. R is powerful and flexible, enabling us to handle a variety of data sets and create appealing graphics. For this reason, it is widely used in academia and industry. The New York Times described R as a popular programming language used by a growing number of data analysts inside corporations and academia. It is becoming their lingua franca—whether being used to set ad prices, find new drugs more quickly or fine-tune financial models. Companies as diverse as Google, Pfizer, Merck, Bank of America, the InterContinental Hotels Group and Shell use it…. “The great beauty of R is that you can modify it to do all sorts of things,” said Hal Varian, chief economist at Google. “And you have a lot of prepackaged stuff that’s already available, so you’re standing on the shoulders of giants.”

To obtain R, go to https://cran.r-project.org/ (The Comprehensive R Archive Network or CRAN), select the link that matches your operating system, and then follow the installation instructions. We used version 4.0.2 of R when writing this textbook.

While a powerful tool for data analysis, R’s main cost from a practical viewpoint is that it must be learned as a programming language. This means that we must master various syntax and rules of computer programming. Learning computer programming is like becoming

1.3 Introduction to R and the tidyverse

proficient in a foreign language. It requires a lot of practice and patience, and the learning process may be frustrating. Through numerous data analysis exercises, this book will teach you the basics of statistical programming, which then will allow you to conduct data analysis on your own. The core principle of the book is that we can learn data analysis only by analyzing data.

Unless you have prior programming experience (or have a preference for another text editor such as Emacs), we recommend that you use RStudio. RStudio is an open-source and free program that greatly facilitates the use of R. For example, we wrote this entire book in RStudio using bookdown. RStudio gives users a text editor where we write programs, a graph viewer which displays graphics we make, the R console where we execute our programs, a help section, and many other features. It may look complicated at first, but RStudio should make learning how to use R much easier. To obtain RStudio, go to http://www.rstudio.com/ and follow the download and installation instructions. Figure 1.1 shows a screenshot of RStudio.

In the remainder of this section we cover five main topics: (1) using R as a calculator, (2) writing R scripts, (3) loading packages, (4) creating and manipulating various objects in R, and (5) loading, subsetting, adding variables to, and summarizing data sets in R.

1.3.1 ARITHMETIC OPERATIONS: R AS A CALCULATOR

We begin by using R as a calculator with standard arithmetic operators. In the R console (the left panel in RStudio, with a > and a cursor; see figure 1.1), we can type in, for example, 5 + 3, then hit Enter on our keyboard.

Figure 1.1. Screenshot of RStudio. The left window shows the R console where R commands can be entered. The upper-right window lists R objects once they are created and a history of executed R commands. Finally, the lower-right window enables us to view plots, data sets, files and subdirectories in the working directory, R packages, and help pages.

**R** ignores spaces, and so typing 5+3 will return the identical result to 5 + 3. However, we added a space before and after the operator + to make it easier to read. As this example illustrates, in this book, **R** commands will be displayed, followed by the outputs they would produce if entered in the **R** console. These outputs are marked by `##` to distinguish them from the **R** commands that produced them. You will not see the `##` in your **R** console, you will just see the output (8, in this case). Finally, in this example, [1] indicates that the output is the first element of a vector of length 1 (we will discuss vectors in section 1.3.5). It is important for readers to try these examples on their own. Remember that we can only learn programming by doing! Let’s try some additional examples.

```
5 + 3
## [1] 8

5 - 3
## [1] 2
5 / 3
## [1] 1.666667
5 ^ 3
## [1] 125
5 * (10 - 3)
## [1] 35
sqrt(4)
## [1] 2
```

The final expression is an example of a so-called function, which takes an input (or multiple inputs) and produces an output. Here, the function `sqrt()` takes a nonnegative number and returns its square root. As discussed in section 1.3.6, **R** has numerous other functions, and users can even make their own functions.

### 1.3.2 R SCRIPTS

You have now run your first commands in **R**! If you scroll up in the console, you will see a record of the commands you entered and the output. However, if you close **RStudio** and restart it, the console will start again clean. To save a record of our commands, we can use an **R** script. An **R** script is a text file with the file extension (ending) of `.R`. In **RStudio**, you can start a new script using the dropdown menu `File > New File > R Script`. There are at least two other ways to start a new **R** script in **RStudio**—see if you can find them.

Try out your new **R** script by typing in the commands from the section above on arithmetic operations into the script. Notice that if you hit **Enter**, you get a line break in the script but no output. In order to have **R** actually evaluate the command you have written,
highlight it and press the Run button in RStudio. You should see the command and the output appear in the console window. In RStudio, there are many ways to ask R to run R code from a script. For example, you can use keyboard shortcuts, which is much faster than clicking the Run button. The shortcut Ctrl+Enter (on a Windows machine) or Cmd+Return (on a Mac) will run the line of code your cursor is on. You can see all of the many RStudio keyboard shortcuts by searching online or clicking on Tools > Keyboard Shortcuts Help.

R scripts are a useful way to save R code. Go ahead and save the new script you have just created. The next time you open RStudio, you can now open that script and rerun the same R commands. R scripts can get very messy very quickly and there are many guides and suggestions for best practice. We offer two suggestions for now. First, use your R script to save finalized commands. Type in the console to practice and debug, then use the script to save what worked. Second, R scripts allow you to save comments, or annotations, to your code. Anything following the # symbol in a script will not be treated like an R command—essentially, R will skip over it. You can use comments to explain what your commands do. This is helpful for transparency when sharing code with others and for reminding yourself what you were doing when you next return to R. It is customary to use a double comment character ## if a comment occupies an entire line and use a single comment character # if a comment is made within a line after an R command. The code below provides an example of commenting.

```r
## this is the start of an R script
## the heading provides some information about the file

## File name: testing_arithm.R
## Author: Kosuke Imai and Nora Webb Williams
## Purpose: Practicing basic math commands and commenting in R
##
5 - 3 # what is 5 minus 3?
5 / 3
5 ^ 3
5 * (10 - 3) # a bit more complex
sqrt(4) # this function will take the square root of a number
```

1.3.3 LOADING PACKAGES
One of the benefits of using R is that many researchers have developed open-source packages that are easily installed and used for your own projects. Packages might contain data, functions, or other tools. In this book, we rely heavily on a family of packages collectively referred to as the tidyverse. The tidyverse packages include ggplot2, dplyr, tidyr, readr, purrr, tibble, and a few others. As mentioned previously, you can think of the tidyverse as a dialect of R programming. The tidyverse is in continuing development, meaning that some of the functions are subject to change (though many of the core functions are fairly stable). If you encounter difficulties getting our example code to run, it may be because the tidyverse
has changed. If this is the case, you can look up documentation about the package to learn how to update your code.

When you open an \texttt{R} script, you can usually tell fairly quickly whether the author is using base \texttt{R} or the \texttt{tidyverse} (or both—we can mix and match to tell \texttt{R} what we want it to do). Scripts in base \texttt{R} tend to have lots of dollar signs ($$\,$$) and square brackets ([ ]) while scripts using the \texttt{tidyverse} tend to have lots of what we call the pipe operator ($$\%\%$$).

The \texttt{install.packages()} command will download the required package materials to your computer. You only need to install a package once per computer, though we can update packages later upon the release of a new version (by clicking \texttt{Update} or reinstalling it via the \texttt{install.packages()} function). In RStudio, you can also install packages by using the \texttt{Tools > Install Packages...} dropdown menu. After you have successfully installed a package, you load it for your current \texttt{R} session with the \texttt{library()} command. Although you do not need to install packages at the start of every \texttt{R} session (or script), you do need to load your necessary packages every time. If you receive an error that a certain function is not found, it may mean that you forgot to run a \texttt{library()} command for the package that contains that function.

You can run the following commands to install the package \texttt{devtools}, which contains a function that will then allow you to install the \texttt{qss} package. This package has all the data you will need for the examples and exercises in this book. The double colons (::) in \texttt{devtools::install_github()} tell \texttt{R} to look in the \texttt{devtools} package for the \texttt{install_github()} function. This function is used to install a package from GitHub, which is a website to build and share software. Note that \texttt{R} is particular about quotation marks when installing and loading packages. Make sure to use quotes around the package name with the \texttt{install()} commands. You do not have to use quotes around the package name with \texttt{library()} (but you can).

```r
install.packages("devtools") # install the package
library(devtools) # load the package
## install a package from GitHub
devtools::install_github("kosukeimai/qss-package",
                         build_vignettes = TRUE)
## you may need to allow R to update/install additional packages
## load the qss package
library("qss")
```

It is best practice to load the packages you will need at the start of your \texttt{R} session (or at the start of your \texttt{R} script). That way you can troubleshoot package installation at the start. Sometimes, installing a package will require you to install additional packages, often called dependencies (the other packages that the one you are installing \texttt{depends} on to run properly). You may also encounter difficulties depending on which version of \texttt{R} you have and which version of a package you are installing. The internet is a great resource for troubleshooting these issues. Your instructor or computing/statistics helpdesks may also be able to provide assistance. For now, practice by installing and loading the \texttt{tidyverse} family of packages.
library(tidyverse)
## if this command does not work, remember to install the package

### 1.3.4 OBJECTS

R can store information as an object with a name of our choice. Once we have created an object, we just refer to it by name. That is, we are using objects as “shortcuts” to some piece of information or data. For this reason, it is important to use an intuitive and informative name. The name of our object must follow certain restrictions. For example, it cannot begin with a number (but it can contain numbers). Object names also should not contain spaces. We must avoid special characters such as % and $, which have specific meanings in R. In RStudio, we will see the objects we created in the upper-right window called Environment (see figure 1.1). We use the assignment operator <- to assign some value to an object.

For example, we can store the result of the above calculation as an object named `result`, and once this is done we can access the value by referring to the object's name. By default, R will print the value of the object to the console if we just enter the object name and hit Enter. Alternatively, we can explicitly print it by using the `print()` function.

```r
result <- 5 + 3
result
## [1] 8
print(result)
## [1] 8
```

Note that if we assign a different value to the same object name, then the value of the object will be changed. And so, we must be careful not to overwrite previously assigned information that we plan to use later.

```r
result <- 5 - 3
result
## [1] 2
```

Another thing to be careful about is that object names are case sensitive. For example, `Hello` is not the same as either `hello` or `HELLO`. As a consequence, we receive an error in the R console when we type `Result` rather than `result`, which is defined above.

```r
Result
## Error in eval(expr, envir, enclos): object ’Result’ not found
```
Encountering programming errors or bugs is part of the learning process. The tricky part is figuring out how to fix them. Here, the error message tells us that the `Result` object does not exist. We can see the list of existing objects in the `Environment` tab in the upper-right window (see figure 1.1), where we will find that the correct object is `result`. It is also possible to obtain the same list by using the `ls()` function, which will list all the current objects that exist in R.

So far, we have only assigned numbers to an object. But R can represent various other types of values as objects. For example, we can store a string of characters by using quotation marks.

```r
kosuke <- "instructor"
kosuke
## [1] "instructor"
```

In character strings, spacing is allowed.

```r
kosuke <- "instructor and author"
kosuke
## [1] "instructor and author"
```

Notice that R treats numbers like characters when we tell it to do so by surrounding a number with quotation marks.

```r
Result <- "5"
Result
## [1] "5"
```

However, arithmetic operations like addition and subtraction cannot be used for character strings. For example, attempting to divide or take a square root of a character string will result in an error.

```r
Result / 3
## Error in Result/3: non-numeric argument to binary operator
sqrt(Result)
## Error in sqrt(Result): non-numeric argument to mathematical function
```

R recognizes different types of objects by assigning each object to a class. Separating objects into classes allows R to perform appropriate operations on an object depending on its class. For example, a number is stored as a `numeric` object whereas a character string is recognized...
as a character object (sometimes also referred to as a string). In RStudio, the Environment window will show the class of an object as well as its name. The function (which by the way is another class) class() tells us to which class an object belongs.

```r
result
## [1] 2
class(result)
## [1] "numeric"

Result
## [1] "5"
class(Result)
## [1] "character"
class(sqrt)
## [1] "function"
```

There are many other classes of objects in R, some of which will be introduced throughout this book. In fact, it is even possible to create our own object classes.

### 1.3.5 VECTORS

We present the simplest (but most inefficient) way of entering data into R. Table 1.1 contains estimates of the world population (in thousands) over the past several decades.

We can enter these data into R as a numeric vector object. A vector or a one-dimensional array simply represents a collection of information stored in a specific order. We use the function `c()`, which stands for “concatenate,” to enter a data vector with multiple values.

<table>
<thead>
<tr>
<th>Year</th>
<th>World population (thousands)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>2,525,779</td>
</tr>
<tr>
<td>1960</td>
<td>3,026,003</td>
</tr>
<tr>
<td>1970</td>
<td>3,691,173</td>
</tr>
<tr>
<td>1980</td>
<td>4,449,049</td>
</tr>
<tr>
<td>1990</td>
<td>5,320,817</td>
</tr>
<tr>
<td>2000</td>
<td>6,127,700</td>
</tr>
<tr>
<td>2010</td>
<td>6,916,183</td>
</tr>
</tbody>
</table>

with commas separating different elements of the vector we are creating. For example, we can enter the world population estimates as elements of a single vector.

```r
world.pop <- c(2525779, 3026003, 3691173, 4449049, 5320817, 6127700, 6916183)
world.pop
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

The `c()` function can be used to combine multiple vectors into one.

```r
pop.first <- c(2525779, 3026003, 3691173)
pop.second <- c(4449049, 5320817, 6127700, 6916183)
pop.all <- c(pop.first, pop.second)
pop.all
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

To access specific elements of a vector, we use square brackets `[ ]`. This is called *indexing*. Multiple elements can be extracted via a vector of indices within square brackets, while the minus sign, `-`, removes the corresponding element from a vector. Note that none of these operations changes the original vector.

```r
world.pop[2]
## [1] 3026003
world.pop[c(2, 4)]
## [1] 3026003 4449049
world.pop[c(4, 2)]
## [1] 4449049 3026003
world.pop[-3]
## [1] 2525779 3026003 4449049 5320817 6127700 6916183
```

Since each element of this vector is a numeric value, we can apply arithmetic operations to it. The operations will be repeated for each element of the vector. Let's give the population estimates in millions instead of thousands by dividing each element of the vector by 1000.

```r
pop.million <- world.pop / 1000
pop.million
```

```
## [1] 2525.779 3026.003 3691.173 4449.049 5320.817 6127.700 6916.183
```
We can also express each population estimate as a proportion of the 1950 population estimate. Recall that the 1950 estimate is the first element of the vector `world.pop`.

```
pop.rate <- world.pop / world.pop[1]
pop.rate
## [1] 1.000000 1.198047 1.461400 1.761456 2.106604 2.426063 2.738238
```

In addition, arithmetic operations can be done using multiple vectors. For example, we can calculate the percentage increase in population for each decade, defined as the increase over the decade divided by its beginning population. Suppose that the population was 100 thousand in one year and increased to 120 thousand in the following year. In this case, we say that the population increased by 20%. To compute the percentage increase for each decade, we first create two vectors, one without the first decade and the other without the last decade. We then subtract the second vector from the first vector. Each element of the resulting vector equals the population increase from the decade before. For example, the first element is the difference between the 1960 population estimate and the 1950 estimate. We then divide each element in the vector of population increases by the population at the end of the decade to get the percentage increase.

```
pop.increase <- world.pop[-1] - world.pop[-7]
percent.increase <- (pop.increase / world.pop[-7]) * 100
percent.increase
```

Finally, we can also replace the values associated with particular indices by using the usual assignment operator (`<-`).

```
percent.increase[c(1, 2)] <- c(20, 22)
percent.increase
```

### 1.3.6 FUNCTIONS

Functions are important objects in R and perform a wide range of tasks. A function often takes multiple input objects and returns an output object. We have already seen several functions: `sqrt()`, `print()`, `class()`, and `c()`. In R, a function generally runs as `funcname(input)` where `funcname` is the function name and `input` is the input object.
In programming (and in math), we call these inputs arguments. For example, in the syntax `sqrt(4), `sqrt` is the function name and 4 is the argument or the input object.

Some basic functions useful for summarizing data include `length()` for length of a vector or equivalently the number of elements in the vector, `min()` for minimum value, `max()` for maximum value, `range()` for the range of the data (the highest and lowest values), `mean()` for the mean or average, and `sum()` for the sum of the data (adding all the values). Right now we are only inputting one object into these functions so we will not use argument names.

```r
length(world.pop)
## [1] 7
min(world.pop)
## [1] 2525779
max(world.pop)
## [1] 6916183
range(world.pop)
## [1] 2525779 6916183
mean(world.pop)
## [1] 4579529
sum(world.pop) / length(world.pop)
## [1] 4579529
```

The last expression above gives another way of calculating the mean: as the sum of all elements in a vector divided by the number of elements in that vector.

When multiple arguments are given, the syntax looks like `funcname(input1, input2). The order of inputs matters. That is, `funcname(input1, input2)` is different from `funcname(input2, input1). To avoid confusion and problems stemming from the order in which we list arguments, it is also a good idea to specify the name of the argument that each input corresponds to. This looks like `funcname(arg1 = input1, arg2 = input2).

For example, the `seq()` function generates a vector composed of an increasing or decreasing sequence of numbers. The first argument `from` specifies the number to start from; the second argument `to` specifies the number at which to end the sequence; the last argument `by` indicates the interval to increase or decrease by. We can create an object for the `year` variable from table 1.1 using this function.

```r
year <- seq(from = 1950, to = 2010, by = 10)
year
```
Notice how we can switch the order of the arguments without changing the output because we have named the input objects.

```r
seq(to = 2010, by = 10, from = 1950)
```

Although not relevant in this particular example, we can also create a decreasing sequence using `seq()`. In addition, the colon operator `:` creates a simple sequence, beginning with the first number specified and increasing or decreasing by 1 to the last number specified.

```r
seq(from = 2010, to = 1950, by = -10)
2008:2012
2012:2008
```

The `names()` function can access and assign names to elements of a vector. Element names are not part of the data themselves, but are helpful attributes of the R object. Below, we see that the object `world.pop` does not yet have the names attribute, with `names(world.pop)` returning the NULL value. However, once we assign the `year` vector as the labels for the object, each element of `world.pop` is printed with an informative label.

```r
names(world.pop)
## NULL
names(world.pop) <- year
names(world.pop)

world.pop
## 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

In many situations, we want to create our own functions and use them repeatedly. This allows us to avoid duplicating identical (or nearly identical) sets of code chunks, making our code more efficient and easily interpretable. The `function()` function can create a new function. The syntax takes the following form:
In this example code, myfunction is the function name, input1, input2,..., inputN are the input arguments, and the commands within the brackets { } define the actual function. Finally, the return() function returns the output of the function. We begin with a simple example, creating a function to compute a summary of a numeric vector. It takes a vector as an input and returns the sum of all the numbers in the vector, the number of objects in the vector (the length), and the mean of the vector.

```r
my.summary <- function(x) {
  s.out <- sum(x)
  l.out <- length(x)
  m.out <- s.out / l.out
  out <- c(s.out, l.out, m.out)
  names(out) <- c("sum", "length", "mean")
  return(out)
}
```

z <- 1:10  # z is a vector from 1 to 10
my.summary(z)  # run my.summary function on z
## sum length mean
## 55.0 10.0 5.5

my.summary(world.pop)  # run my.summary function on world.pop
## sum length mean
## 32056704 7 4579529

Note that objects, e.g., x, s.out, l.out, m.out, and out in the above example, can be defined within a function independently of the environment in which the function is being created. This means that we need not worry about using identical names for objects inside a function and those outside it.

### 1.3.7 DATA FILES: LOADING AND SUBSETTING

So far, we have only used data that we manually entered into R. But, most of the time, we will load data from an external file. In this book, we will use the following two data file types:

- CSV or comma-separated values files represent tabular data. This is conceptually similar to a spreadsheet of data values like Microsoft Excel or Google Sheets. Each observation is separated by line breaks and each field within the observation is separated by a comma, a tab, or some other character or string.
• *RData* files represent a collection of *R* objects including data sets. These can contain multiple *R* objects of different kinds. They are useful for saving intermediate results from our *R* code as well as data files.

Before interacting with data files, we must ensure they reside in the *working directory*, which is the location on your computer where *R* will by default load data from and save data to. There are different ways to change the working directory. In *RStudio*, the default working directory is shown in the bottom-right window under the *Files* tab (see figure 1.1). Oftentimes, however, the default directory is not the directory we want to use. To change the working directory, click on *More > Set As Working Directory* after picking the folder we want to work from. Alternatively, we can use the *RStudio* pull-down menu *Session > Set Working Directory > Choose Directory...* and then pick the folder we want to work from. Then we will see our files and folders in the bottom-right window.

It is also possible to change the working directory using the *setwd()* function by specifying the full path to the folder of our choice as a character string. To display the current working directory, use the function *getwd()* without providing an input. For example, the following syntax will set the working directory to *qss/INTRO* and confirms the result (we suppress the output here).

```r
getwd() # check what your current working directory is
setwd("qss/INTRO") # set your working directory with a path
getwd() # check that you changed your working directory
```

Suppose that the United Nations population data in table 1.1 are saved as a CSV file called *UNpop.csv* on our computer. The data resemble the example below. In tabular data like this, we often think of (and refer to) the columns as *variables* and the rows as *observations*. You will notice that we use the terms *column* and *variable* interchangeably, just as we use *row* and *observation* interchangeably.

```r
year, world.pop
1950, 2525779
1960, 3026003
1970, 3691173
1980, 4449049
1990, 5320817
2000, 6127700
2010, 6916183
```

In *RStudio*, we can read in or load CSV files by going to the dropdown menu in the upper-right *Environment* window (see figure 1.1) and clicking *Import Dataset > From Text File...*. Alternatively, we can use the *read_csv()* function from the *readr* package. The following syntax loads the data as a *tibble* object (more on this object below).
Chapter 1: Introduction

On the other hand, if the same data set is saved on our computer as an object in an RData file named `UNpop.RData`, then we can use the `load()` function, which will load all the R objects saved in `UNpop.RData` into our R session. We do not need to use the assignment operator (`<-`) with the `load()` function with an RData file because the R objects stored in the file already have object names. Note that this code will overwrite the data set we loaded with `read_csv()` above because the objects have the same name. Although that is what we want in this case, there is a general lesson here that it is important to keep track of object names so that we do not accidentally overwrite objects.

```
load("UNpop.RData")
```

Note that R can access any file on our computer if the full location is specified. The location of a file on your computer is called the path. For example, we can use syntax such as `read_csv("Documents/qss/INTRO/UNpop.csv")` if the data file `UNpop.csv` is stored in the directory `Documents/qss/INTRO/`. However, setting the working directory as shown above allows us to avoid a lot of tedious typing (and typos).

Increasingly, researchers use R projects instead of changing the working directory at the start of each script. An R project tells RStudio which directory is home base. Then, instead of pointing R to look for files on your computer using absolute paths, such as `read_csv("Documents/qss/INTRO/UNpop.csv")`, you can instead use relative paths. For example, say your R project is in a directory called QSS on your computer, and you have a subdirectory in that folder called INTRO, where you are saving all the .csv files for this chapter. You could then use the following relative path in the `read_csv()` function to find `UNpop.csv`. To create an R project, you can use the File > New Project dropdown menu in RStudio or click on the project icon in the upper right.

```
## specifying a relative path to find and read in UNpop.csv
## will overwrite previously loaded UNpop object
UNpop <- read_csv("INTRO/UNpop.csv")
class(UNpop) # what type of object is UNpop?
## [1] "spec_tbl_df" "tbl_df" "tbl" "data.frame"
```

Having discussed multiple ways to read in data saved on your computer as a .csv or .Rdata, we can now make things even easier for the data used in this textbook. If you have successfully installed the qss package (see section 1.3.3), you can load in any of the referenced data with the following syntax and the data() command.
Now that we've loaded `UNpop`, let's see what we have. We can think of a `data.frame` or `tibble` object as a spreadsheet. We can view a table-like representation of `data.frame` or `tibble` objects in RStudio by clicking on the object name in the Environment tab in the upper-right window (see figure 1.1). Alternatively, we can use the `View()` function with the object name as the input argument. This will open a new tab displaying the data. Useful functions for this object include the `names()` function to return a vector of variable names, the `nrow()` function to return the number of rows, the `ncol()` function to return the number of columns, and the `dim()` function to combine the outputs of `ncol()` and `nrow()` into a vector (also known as the dimensions of the data).

```r
names(UNpop)
## [1] "year"       "world.pop"
nrow(UNpop)
## [1] 7
ncol(UNpop)
## [1] 2
dim(UNpop)
## [1] 7 2
```

There are several ways to access an individual variable. We begin with the base R syntax. First, one could use the `$` operator to extract a variable from a `data.frame` object, which returns a vector containing the specified variable.

```r
UNpop$world.pop
## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183
```

Second, we could also use indexing `[ ]`, as done for a vector. Since a `data.frame` object is a two-dimensional array, we need two indexes, one for rows and the other for columns. Using brackets with a comma `[rows, columns]` allows users to call specific rows and columns by either row/column numbers or row/column names. If we use row/column numbers, sequencing functions covered above, i.e., `:` and `c()`, will be useful. If we do not specify a row (column) index, then the syntax will return all rows (columns). Here are some syntax examples, which show how this indexing works.
## subset all rows for column called "world.pop" from UNpop data

```r
UNpop[, "world.pop"]
```

## [1] 2525779 3026003 3691173 4449049 5320817 6127700 6916183

## subset the first three rows (and all columns)

```r
UNpop[c(1, 2, 3),]
```

## year world.pop

```r
## 1 1950 2525779
## 2 1960 3026003
## 3 1970 3691173
```

## subset the first three rows of the "year" column

```r
UNpop[1:3, "year"]
```


In the **tidyverse** syntax, extracting subsets of data looks a bit different. Instead of using lots of brackets, we will begin by using two functions: `slice()` and `select()`. The `slice()` function returns rows by number (or other criteria), while `select()` returns columns by name, number, or other criteria (more on this in a bit). The first input argument in `slice()` or `select()` is the data. The second input provides information about how to subset the data. Notice that the base **R** syntax (see code above) returns a vector, while the **tidyverse** syntax returns a *tibble*.

```r
## subset the first three rows of UNpop with tidyverse

```

```r
slice(UNpop, n = 1:3)
```

## year world.pop

```r
## 1 1950 2525779
## 2 1960 3026003
## 3 1970 3691173
```

## extract/subset the world.pop variable (column)

```r
select(UNpop, world.pop)
```

## world.pop

```r
## 1 2525779
## 2 3026003
## 3 3691173
## 4 4449049
## 5 5320817
## 6 6127700
## 7 6916183
```

Let's say we wanted to subset the first three rows just for the variable `year`. We could do that in any of the following ways—notice that we can nest a `slice()` command

```r
## subset all rows for column called "world.pop" from UNpop data

```
inside a `select()` command. R will select the `year` column and then slice the first three rows.

```r
## base R subset the first three rows of the year variable
UNpop[1:3, "year"]
## or in tidyverse, combining `slice()` and `select()`
select(slice(UNpop, 1:3), year)
## year
## 1 1950
## 2 1960
## 3 1970
```

Instead of nesting the `slice()` into `select()`, we could use the pipe operator `%>%` to link commands together. This tells R to do something and then do something else to the output of the first something. Chaining functions together like this will become very useful as our tasks become more complicated.

```r
UNpop %>% # take the UNpop data we have loaded, and then...
  slice(1:3) %>% # subset the first three rows, and then...
  select(year) # subset the year column
## year
## 1 1950
## 2 1960
## 3 1970
```

As another subsetting example, imagine that we want to extract every other row of the `world.pop` column from `UNpop` (i.e., we want rows 1, 3, 5, etc. for the `world.pop` variable). We could use an additional helper function, `n()`, which returns the number of rows in the `data.frame` or `tibble`.

```r
UNpop %>%
  slice(seq(1, n(), by = 2)) %>% # using a sequence from 1 to n()
  select(world.pop)
## world.pop
## 1 2525779
## 2 3691173
## 3 5320817
## 4 6916183
```
A final example of how to subset these specific rows and column uses the `filter()` function. `filter()` is to rows what `select()` is to columns—it subsets rows by name, order, or other criteria. These are two very commonly used functions in the tidyverse. To keep them straight, note that `select()` has a `c` in it (and therefore is for columns), while `filter()` has an `r` in it (and therefore is used for rows). In the example below, `filter()` says to subset rows if their row number divided by 2 gives a remainder of 1. The `%%` operator returns the modulus, i.e., division remainder. The function `row_number()` returns the row number of an observation.

```r
UNpop %>%
  filter(row_number() %% 2 == 1) %>%
  select(world.pop)
```

The `filter(row_number() %% 2 == 1)` in the above code makes use of what is called a conditional or logical statement. We will discuss these more in depth in chapter 2. For now, think of these as “if” conditions, telling R to do something if a condition is met. The condition might be that something is equal to something else, such as the modulus being equal to 1 as in the example. The “equal to” condition is indicated in R with `==` (note that this is not the same as a single `=`). In the example, we are telling R to return the subset of rows where the modulus of dividing the row number by 2 is equal to 1 (in other words, returning the odd number observations 1, 3, 5, etc.).

For conditional statements, we can also use the “less than” `<`, “less than or equal to” `<=`, “greater than” `>`, and “greater than or equal to” `>=` syntax. An exclamation point in a conditional indicates negation, so `!=` means “is not equal to.”

The following code uses `filter()`, `select()`, and a conditional statement with the function `pull()` to extract a specific value from our data as a vector instead of a tibble. Let’s say, for example, that we wanted to know what the world population was in 1970. We could use the following commands.

```r
pop.1970 <- UNpop %>%
  filter(year == 1970) %>%
  select(world.pop) %>%
  pull()

print(pop.1970)
```

## [1] 3691173
1.3.8 ADDING VARIABLES

Suppose we wanted to take the population data and add an additional column based on a current column. For example, perhaps we want to have the world population in millions, instead of the raw figure in the original data. We can use the `mutate()` function to create that variable, which we call `world.pop.mill`, and add it to the tibble. We can then drop the original `world.pop` variable using the `select()` function with a `-` and the column name. In the example below, we also use `<-` to save a new version of the data that contains the new column. Note that if we put the same object name on both sides of the `<-`, that would overwrite the existing data. If you run the code below, you should have both `UNpop` and `UNpop.mill` in your Environment. You may want to look at the new data to confirm that the new variable is as you expect.

```r
UNpop.mill <- UNpop %>%
  # create a new tibble from UNpop
  ## create new variable world.pop.mill
  mutate(world.pop.mill = world.pop / 1000) %>%
  select(-world.pop) # drop the original world.pop column
```

The `mutate()` function is a very useful command. We used it above to do an arithmetic operation on a column. We can also use it to combine columns based on our specifications, as in the example below. Let's say we wanted a variable that took the world population and divided it by the year (why we would want to do this is unclear, but let's go with it for now). The following code shows how we could do that by using the column names.

```r
## adding a nonsense variable to the UNpop.mill data
UNpop.mill <- UNpop.mill %>%
  mutate(nonsense.var = world.pop.mill / year)
```

We can combine the `mutate()` function with conditional statements in useful ways by using the function `if_else()`. The function tells R to do something if a conditional statement is met and to do something else if the statement is not met. Say we wanted a new variable that indicates whether or not a row contains data from after 1980. We'll call this new variable `after.1980`. We want this variable to have two possible values: 1 if the row is from after 1980 and 0 if it's not. In the example below, we use `mutate()` to tell R that we want a new variable called `after.1980`. The value that each row will have for this new column is set by the `if_else()` function. The conditional statement is the first argument. It tells R to check the `year` column for a given row. If `year` is greater than or equal to 1980, then the value for the variable `after.1980` should be 1. If `year` is not greater than or equal to 1980, then the value for the variable `after.1980` should be 0. It is usually a good idea to check that your new variable looks the way you expect.

```r
## adding a variable with if_else
UNpop.mill <- UNpop.mill %>%
  mutate(after.1980 = if_else(year >= 1980, 1, 0))
```
A final example with `mutate()` and `if_else()` uses a very helpful conditional symbol: `%in%`. We can follow `%in%` with a vector of values. R will then check whether a specific value matches something in that vector. For example, let's say that we also wanted to add a variable noting whether a row was from the following specific set of years (imagine that these years are of particular interest to us): 1950, 1980, and 2000. In the following code, we first create a vector of those years, then we reference it within `if_else()` to create a new variable, `years.of.interest`.

```r
## creating a vector of the years of interest
specific.years <- c(1950, 1980, 2000)

## adding a variable with if_else and %in%
UNpop.mill <- UNpop.mill %>%
  mutate(year.of.interest = if_else(year %in% specific.years, 1, 0))
```

### 1.3.9 DATA FRAMES: SUMMARIZING

Having loaded our data and created some new variables, we now turn to some ways to summarize the data. The `summary()` function is useful for this. The `summary()` function yields, for each variable in the input `data.frame` or `tibble` object, the minimum value, the first quartile (or 25th percentile), the median (or 50th percentile), the third quartile (or 75th percentile), and the maximum value. We can also use functions like `mean()` to compute summary values for specific variables in the data. See section 3.3 for more discussion.

```r
summary(UNpop.mill)

# year  world.pop.mill  nonsense.var
# Min. :1950 Min. :2526 Min. :1.295
# 1st Qu.:1965 1st Qu.:3359 1st Qu.:1.709
# Median :1980 Median :4449 Median :2.247
# Mean :1980 Mean :4580 Mean :2.305
# 3rd Qu.:1995 3rd Qu.:5724 3rd Qu.:2.869
# Max. :2010 Max. :6916 Max. :3.441
# after.1980 year.of.interest
# Min. :0.0000 Min. :0.0000
# 1st Qu.:0.0000 1st Qu.:0.0000
# Median :1.0000 Median :0.0000
# Mean :0.5714 Mean :0.4286
# 3rd Qu.:1.0000 3rd Qu.:1.0000
# Max. :1.0000 Max. :1.0000
mean(UNpop.mill$world.pop.mill)
# [1] 4579.529
```

In R, missing values are represented by `NA`. When applied to an object with missing values, functions may or may not automatically remove those values before performing operations.
We will discuss the details of handling missing values in section 3.2. Here we note that for many functions, like `mean()`, the argument `na.rm = TRUE` will remove missing data before operations occur. Below we use the `add_row()` function to add a row with NA values to demonstrate the issue that arises with `mean()` and how to fix it.

```r
## add a row where value for each column is NA
UNpop.mill.wNAs <- UNpop.mill %>%
    add_row(year = NA, world.pop.mill = NA,
             nonsense.var = NA, after.1980 = NA, year.of.interest = NA)
## take the mean of world.pop.mill (returns NA)
mean(UNpop.mill.wNAs$world.pop.mill)
## [1] NA
## take the mean of world.pop.mill (ignores the NA)
mean(UNpop.mill.wNAs$world.pop.mill, na.rm = TRUE)
## [1] 4579.529
```

The `tidyverse` offers a useful way to generate summaries with the `summarize()` function (or `summarise()`, both spellings are accepted). With this function, you can specify multiple functions to apply to variables within a data set, returning the results as new columns in a tibble. The example below returns both the median and the mean of `world.pop.mill`.

```r
UNpop.mill %>%
    summarize(mean.pop = mean(world.pop.mill),
              median.pop = median(world.pop.mill))
## mean.pop median.pop
## 1 4579.529  4449.049
```

What if we wanted to know the average (mean) world population but not for the full time period? For example, we might want to know what the average population was before 1980 and after 1980. To do this, we can combine the `summarize()` function with the `group_by()` function, which tells R to treat subsets of the data separately. We need to tell R which variable to use to group the rows. We can use the variable we created earlier, `after.1980`, for this purpose.

```r
UNpop.mill %>%
    group_by(after.1980) %>%
    summarize(mean.pop = mean(world.pop.mill))
```
## 1.3.10 SAVING OBJECTS

The objects we create in an R session will be temporarily saved in the workspace, which is the current working environment. As mentioned earlier, the `ls()` function displays the names of all objects currently stored in the workspace. In RStudio, all objects in the workspace appear in the Environment tab in the upper-right corner. However, these objects will be lost once we terminate the current session. This can be avoided if we save the workspace at the end of each session as an RData file.

When we quit R, we will be asked whether we would like to save the workspace. We should answer no to this so that we get in the habit of explicitly saving only what we need. If we answer yes, then R will save the entire workspace as .RData in the working directory without an explicit file name and automatically load it next time we launch R. This is not recommended practice because the .RData file is invisible to users of many operating systems and R will not tell us what objects are loaded unless we explicitly issue the `ls()` function.

In RStudio, we can explicitly save the workspace by clicking the save icon in the upper-right Environment window (see figure 1.1). Alternatively, from the navigation bar, click on Session > Save Workspace As..., and then pick a location to save the file. Be sure to use the file extension .RData. To load the same workspace the next time we start RStudio, click the open file icon in the upper-right Environment window or select Session > Load Workspace....

It is also possible to save the workspace using the `save.image()` function. The file extension .RData should always be used at the end of the file name to save the workspace. Unless the full path is specified, objects will be saved to the working directory. For example, the following syntax saves the workspace as Chapter1.RData in the qss/INTRO directory provided that this directory already exists.

```
save.image("qss/INTRO/Chapter1.RData")
```

Generally speaking, however, it is not recommended to save the workspace. Best practice is to turn off the RStudio prompt to save your workspace by going to Tools > Global Options > General and setting “Save workspace to .RData on exit” to “Never” (while you have the Global Options open, note that this is also where you can change the appearance of RStudio—changing to a different color scheme with a nonwhite background can be easier on the eyes.)

It is better to save your R source code (e.g., scripts), which you can rerun to reproduce your results. Sometimes we wish to save a specific object (e.g., a data.frame or tibble object) rather than the entire workspace. This can be done with the `save()` function as in `save(xxx, file = "yyy.RData")` where xxx is the object name and yyy.RData

(continued...)
### General Index

**A**  
absolute value, 79  
addition rule, 282  
adjacency matrix, 239  
adjusted $R^2$, 195, 432  
aesthetics, 97  
alternative hypothesis, 395  
AND, 46  
animation, 271  
arguments, 18  
assignment operator, 13  
association, 59  
asymptotic theorems, 342  
average, 18  
average treatment effect, 59  
axioms, 282

**B**  
background, 33  
backtick, 63  
bag of words, 223  
bar plot, 97  
Bayes' rule, 307  
Bayesian, 280  
before-and-after design, 72  
Bernoulli random variable, 321  
betweenness, 243, 248  
bias, 154, 359  
bin, 100  
binary random variable, 321  
binary variable, 44, 45  
binomial distribution, 325  
binomial theorem, 327  
birthday problem, 285  
bivariate relationships, 116  
box plot, 103  
butternfly ballot, 185

**C**  
categorical variable, 53  
causal effects, 56  
causal inference, 56  
CDF, see cumulative distribution function  
ceiling effects, 114  
census, 106  
centering, 131  
central limit theorem, 346, 371, 388, 412, 428, 433  
centrality, 241  
centroid, 131  
ceteris paribus, 191  
character, 15, 55  
character variable, 97  
class, 14  
classification, 158  
classification error, 235  
closeness, 243, 248  
clustering algorithms, 128  
clusters, 128  
coefficient of determination, 181, 195, 432  
coefficients, 166  
column, 21  
combinations, 289  
comma-separated values, 20  
complement, 283  
complete randomization, 360, 390  
computational revolution, 1  
conditional, 26  
conditional expectation, 418  
conditional expectation function, 419  
conditional independence, 303, 315  
conditional probability, 294, 297  
conditional statements, 53  
confidence bands, 372  
confidence interval, 372  
confidence level, 372  
confounders, 69, 258  
confounding bias, 69  
confusion matrix, 158  
consistent, 359  
contingency table, 42  
continuous random variable, 321, 322  
control group, 59, 65  
corpus, 218  
correlation, 122, 165  
correlation coefficient, 122  
counterfactual, 38, 56
covariance, 432
coverage probability, 374
critical value, 372, 388
cross tabulation, 42
cross-section comparison design, 65
cross-section data, 72
CSV, see comma-separated values
cumulative distribution function, 322, 323
cumulative sum, 344

data revolution, 1
data-generating process, 167, 288, 362, 418
dates, 153
deciles, 77
degree, 241, 247
degrees of freedom, 195, 369, 387
density, 100, 323
dependencies, 12
descriptive statistics, 75
difference-in-differences, 73
difference-in-means estimator, 59, 190, 360
dimensions, 23
directed, 245
directed network, 239
discrete random variable, 321
disturbance, 166
document frequency, 225
document-term matrix, 222
dummy variable, 45
DW-NOMINATE scores, 116

edgelist, 246
edges, 240
Electoral College, 144
error bands, 372
error term, 166
estimation error, 358
estimator, 357
event, 281
exogeneity, 418
expectation, 335, 339
experiment, 281
experimental data, 40
exploratory data analysis, 216
external validity, 60, 65, 208

factor variable, 53, 97
factorial, 285
factorial variable, 53
false discovery, 409
false discovery rate, 313
false negative, 158
false positive, 158, 313
false positive rate, 308
farness, 242
figures, 96
first moment, 337
first quartile, 76
Fisher's exact test, 395
fitted value, 167
floor effects, 114
frequency, 100
frequentist, 279
function, 10
fundamental problem of causal inference, 57, 390

Gaussian distribution, 328
google, 250

Gini coefficient, 120
Gini index, 120

heterogeneous treatment effects, 197
heteroskedasticity, 428
heteroskedasticity-robust standard errors, 428
hexadecimal, 262
hexadecimal color code, 262
histogram, 100, 155
homoskedasticity, 426
hypothesis testing, 390

i.i.d., see independently and identically distributed
ideology, 114
idf, see inverse document frequency
immutable characteristics, 58
in-sample prediction, 187, 236
indegree, 247
independence, 301
independently and identically distributed, 325
indexing, 16
indicator, 192
indicator function, 342
Institutional Review Board, 113
integration, 335
interaction effect, 198
intercept, 166
internal validity, 60, 65, 207, 208
interquartile range, 76
inverse document frequency, 225
inverse function, 207
IQR, see interquartile range
item count technique, 113
item nonresponse, 111
item response theory, 115
iterations, 146
iterative algorithm, 131
General Index

**J**
- joint independence, 303
- joint probability, 295

**K**
- \( k \)-means, 128
- Kish, 109

**L**
- large sample theorems, 342
- latent, 116
- law of iterated expectation, 425
- law of large numbers, 342, 359, 362
- law of total probability, 284, 292, 296, 303
- law of total variance, 427
- least squares, 171
- leave-one-out cross-validation, 236
- lemmatization, 219
- level of test, 395
- levels, 55
- limit, 279
- linear algebra, 128
- linear model, 166
- linear regression, 162
- linear relationship, 165
- list, 128, 132
- list experiment, 113
- listwise deletion, 96
- logarithmic transformation, 109, 286
- logical, 26
- logical conjunction, 46
- logical disjunction, 46
- logical operators, 46
- logical values, 46
- longitudinal data, 71
- loop, 145, 236
- loop counter, 145
- Lorenz curve, 120
- lower quartile, 76

**M**
- main effect, 199
- maps, 255
- margin of error, 378
- marginal probability, 294
- matrix, 128, 131
- maximum, 18
- mean, 18
- mean-squared error, 367
- measurement models, 114
- median, 28, 75, 102
- merge, 174
- metadata, 219
- minimum, 18
- misclassification, 158
- misreporting, 112
- Monte Carlo error, 289, 364
- Monte Carlo simulation, 306, 324, 339, 343, 362
- Monte Carlo simulation method, 287
- Monty Hall problem, 305
- multiple testing, 409
- multistage cluster sampling, 109

**N**
- natural experiment, 86, 257, 258
- natural language processing, 218
- natural logarithm, 109
- network data, 238
- no omitted variables, 420
- nodes, 240
- nonlinear relationship, 165
- nonresponse, 359
- normal distribution, 328, 346
- null hypothesis, 394
- numeric, 14
- numeric variable, 100

**O**
- object, 13
- observational studies, 65, 420
- observations, 21
- one-sample tests, 397
- one-sample \( t \)-test, 401
- one-sample \( z \)-test, 400
- one-sided \( p \)-values, 395
- one-tailed \( p \)-values, 395
- OR, 46
- out-of-sample prediction, 187, 236
- outcome variable, 40
- outdegree, 247
- outliers, 75, 185
- overfit, 236
- overfitting, 187

**P**
- \( p \)-value, 394
- packages, 11
- PageRank, 250
- panel data, 72
- parameter, 357
- participation rate, 89
- Pascal’s triangle, 328
- path, 22
- PDF, see probability density function
- percentiles, 28, 77
- permutations, 284
- pipe, 12
- placebo test, 207
- plots, 96
- PMF, see probability mass function
- political polarization, 119
- population average treatment effect, 362
- population mean, 335
- positive predictive value, 308
- posterior probability, 307
- potential outcomes, 57
power, 411
power analysis, 413
predicted value, 167
prediction error, 154, 167
pretreatment variables, 64, 69
prior probability, 307
probability, 279
probability density function, 322
probability distributions, 321
probability mass function, 321
probability model, 321
probability sampling, 107
projection, 259
proof by contradiction, 394
proportion, 44
publication bias, 409

Q
Q–Q plot, 125, see quantile–quantile plot
quadratic function, 201
quantile–quantile plot, 125, 332, 388
quantiles, 75, 77, 125
quartiles, 28, 76
quincunx, 346
quintiles, 77
quota sampling, 107

R
R projects, 22
R^2, 181, 195, 432
random digit dialing, 108
random variables, 321
randomization inference, 394
randomized controlled trials, 58, 360, 419
randomized experiments, 58
randomized response technique, 114
range, 18
rational number, 394
RData, 21
receiver, 239
reference distribution, 394
regex, 220
regression discontinuity design, 204
regression line, 167
regression towards the mean, 173, 332
regular expression, 220
representative, 107
residual, 167, 192
residual plot, 184
residual standard error, 432
residuals, 332
RGB, 262
RMS, see root-mean-square
root-mean-square, 79, 155, 171
root-mean-squared error, 155, 171, 367
row, 21
rule of thumb, 378

S
sample average, 45
sample average treatment effect, 59, 360
sample average treatment effect for the treated, 74
sample correlation, 433
sample mean, 45, 68, 335
sample selection bias, 60, 107
sample size calculation, 379
sample space, 281
sample without replacement, 393
sampling distribution, 359, 368, 394
sampling frame, 108, 111
sampling variability, 339
sampling with replacement, 288
sampling without replacement, 288
SATE, see sample average treatment effect
scaling, 131
scatter plot, 116, 163
scientific significance, 397, 402
scraping, 218
script, 10
second moment, 337
second quartile, 76
selection bias, 69
selection on observables, 420
sender, 239
set, 281
sharp null hypothesis, 394
simple random sampling, 107, 288, 358
simple randomization, 360, 390
simulation, 287
slope, 166
social desirability bias, 112
sparse, 222
sparsity, 222
spatial data, 255
spatial point data, 255
spatial polygon data, 255, 258
spatial voting, 115
spatial–temporal data, 256
SPSS, 31
standard deviation, 78, 80, 336
standard error, 368
standard normal distribution, 329, 332, 362
standardize, 131
standardized residuals, 332
STATA, 31
statistical control, 69
statistical significance, 397, 402
stemming, 219
step function, 326
stop words, 221
string, 15
Student's t-distribution, 387
Student's t-test, 416
subclassication, 69
sum, 18
sum of squared residuals, 171, 192
supervised learning, 136, 224
support, 336
survey, 89
survey sampling, 106
symmetric, 239

T
t-distribution, 387
t-statistic, 387, 429
t-test, 405
terciles, 77
term frequency, 221, 223, 225
term frequency-inverse document frequency, 225
term-document matrix, 222
test statistic, 394
tf-idf, see term frequency-inverse document frequency
The Federalist Papers, 217
third quartile, 76
tilde, 54
time trend, 72
time-series plot, 119, 161
tokenizing, 219
topics, 223
total sum of squares, 181
treatment, 57
treatment group, 59, 65
treatment variable, 40, 57
true positive rate, 308, 313
true positives, 313
two-sample tests, 397
two-sample t-test, 404, 405
two-sample z-test, 404
two-sided p-value, 395, 398
two-tailed p-value, 395
type I error, 395
type II error, 395, 411

U
unbiased, 155, 359
unconfoundedness, 420
uncorrelated, 419
undirected, 245
undirected network, 239
uniform random variable, 322
unit nonresponse, 111, 383
unit response rate, 89
unobserved confounders, 419
unsupervised learning, 136, 223
upper quartile, 76

V
variable, 21
variance, 80, 336
vector, 10, 15
Venn diagram, 282, 283
vertices, 240

W
weighted average, 316
weights, 252
with replacement, 107
without replacement, 107
word cloud, 223
working directory, 21
workspace, 30

Z
z-score, 122, 131, 178, 330, 332, 347
R Index

SYMBOLS
+-, 199
+, 10, 33, 97
-, 16, 33
:, 19, 23, 198
::, 12
<, 26, 48
<=, 13, 17, 22, 33
<-, 26, 48
~, 26, 33, 48
=, 26, 48
>, 26, 48
>=, 26, 48
[, ], 23, 223
[[ ]], 130
[ ], 12, 16, 23
#, 11
##, 10, 11
$, 12, 13, 23, 49, 128, 130, 132, 252
%, 13
%>%, 12, 25, 52, 124
%<%, 26, 151
%in%, 28, 71
&., 46, 48
&-, 79
&^2, 79
&^, 63
&^, 54, 124, 167
., 46–48, 95
{ }, 146
{ }, 20
{ }, 146
} ], 221

A
abs(), 79
across(), 297
add_predictions(), 185, 193, 201, 205, 203, 205
add_residuals(), 185
add_row(), 29
aes, 97
aes(), 97, 98, 100, 104, 116, 164
alpha, 263
anim_save(), 271
annotate(), 102, 161
anti_join(), 221
arrange(), 94, 248
arrange(desc()), 94
as.factor(), 55, 97
as.integer(), 46
as.matrix(), 129, 223, 241
as.numeric(), 153, 179
as_tibble(), 155
augment(), 169, 184, 434, 437

B
base, 110
betweenness(), 244, 248
bind_cols(), 177
bind_rows(), 161, 177, 236
bind_tfidf(), 225
bookdown package, 9
borders(), 259–261
broom package, 133, 168, 183, 184, 429

c(), 15–17, 23
c_across(), 240
case_when(), 54, 148, 233, 298
cast_dtm(), 222, 227
cast_tfidf(), 222
cbind(), 156, 177
centers, 132
class, 40
choose(), 291, 392
class, 39
class(), 15, 17
closeness(), 243, 248
cluster, 132
coeff(), 168, 169
col_types, 40
colMeans(), 129
colnames(), 128, 228
color, 116
colors(), 262
cols(), 129
column_to_rownames(), 241
content(), 221
coord_fixed(), 117
coord_quickmap(), 259
cor(), 124, 165
count(), 42, 90, 91, 133, 222, 294
cowplot package, 106
crossing(), 201, 301
cumsum(), 343, 344
cut(), 298
data(), 22, 258
data.frame, 23, 40, 128
data.frame(), 147
data_grid(), 193, 202, 205, 435
Date, 153
dbinom(), 326, 341
desc(), 248
decor() package, 106
diff_time(), 153
dim(), 23, 40, 65
DirSource(), 218
dnorm(), 332
double, 40
dplyr package, 11
drop_na(), 96
dunif(), 324
E
E(), 252
else if(){}[], 150
enframe(), 228
exp(), 110, 207
F
facet_grid(), 124, 384
facet_wrap(), 124
factor, 55
factorial(), 286
FALSE, 46
fill, 98
filter(), 26, 49, 54, 71, 92, 94, 95
fisher.test(), 395, 396
fitted(), 168, 183, 235
floor_date(), 271
for, 146
foreign package, 31
formula, 167
full_join(), 174–177, 221, 266
function(), 19
G
geom_abline(), 126, 157, 169
geom_bar(), 97, 98
gganimate package, 271
ggplot(), 96–98, 100, 103, 106, 116, 124, 126, 135, 155, 156, 249, 254, 259, 263, 268, 332
ggplot2 package, 11, 96, 97, 133
ggsave(), 105
ggtitle(), 97
glance, 168, 183
glimpse(), 40, 65
graph.adjacency(), 241
graph_from_data_frame(), 251
graph_from_edgelist(), 246
ggarrange(), 106
ggridExtra package, 106
group(), 294
group_by(), 29, 42, 45, 51, 55, 63, 70, 91, 133, 232, 297
H
haven package, 31, 32
head(), 40, 65
hist(), 332, 355
I
i, 146
I(), 202
if(){}[], 149
if{}else(), 148, 149
if(){}else(), 148, 149
if_else(), 27, 53, 148
igraph, 251, 254
igraph package, 241
import(), 31
ineq package, 278
ineq(), 278
inner_join(), 311
inspect(), 223
install(), 12
install.packages(), 12
install_github(), 12
integer, 46
IQR(), 76
is.na(), 93–95

For general queries, contact webmaster@press.princeton.edu
<table>
<thead>
<tr>
<th>Function</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter, iter.max</td>
<td>132</td>
</tr>
<tr>
<td>kmeans()</td>
<td>131–133, 227</td>
</tr>
<tr>
<td>labs</td>
<td>100</td>
</tr>
<tr>
<td>lchoose()</td>
<td>291</td>
</tr>
<tr>
<td>left_join()</td>
<td>177, 317</td>
</tr>
<tr>
<td>length()</td>
<td>18, 130</td>
</tr>
<tr>
<td>levels()</td>
<td>55</td>
</tr>
<tr>
<td>lfactorial()</td>
<td>286</td>
</tr>
<tr>
<td>library()</td>
<td>12</td>
</tr>
<tr>
<td>list, list()</td>
<td>130, 183, 80, 130</td>
</tr>
<tr>
<td>lm</td>
<td>167, 183, 192, 193, 196, 420–422, 429, 431</td>
</tr>
<tr>
<td>load()</td>
<td>22, 31, 354</td>
</tr>
<tr>
<td>log()</td>
<td>110</td>
</tr>
<tr>
<td>logical</td>
<td>46, 49</td>
</tr>
<tr>
<td>ls()</td>
<td>14, 30</td>
</tr>
<tr>
<td>lubridate package</td>
<td>153</td>
</tr>
<tr>
<td>map_data()</td>
<td>259, 277</td>
</tr>
<tr>
<td>map_df()</td>
<td>340, 364, 377</td>
</tr>
<tr>
<td>map_lgl()</td>
<td>377</td>
</tr>
<tr>
<td>maps package</td>
<td>258, 259</td>
</tr>
<tr>
<td>matrix, matrix()</td>
<td>128, 128</td>
</tr>
<tr>
<td>max()</td>
<td>18, 94</td>
</tr>
<tr>
<td>mean()</td>
<td>18, 28, 29, 45, 46, 49, 94, 105</td>
</tr>
<tr>
<td>median()</td>
<td>76, 94, 148</td>
</tr>
<tr>
<td>min()</td>
<td>18, 94</td>
</tr>
<tr>
<td>modelr package</td>
<td>185, 193, 207, 422</td>
</tr>
<tr>
<td>mutate, mutate()</td>
<td>317</td>
</tr>
<tr>
<td>mutate_at()</td>
<td>27, 44, 51, 63, 133, 135, 189, 297, 311</td>
</tr>
<tr>
<td>mutate_at()</td>
<td>232</td>
</tr>
<tr>
<td>n()</td>
<td>25, 44</td>
</tr>
<tr>
<td>n_distinct()</td>
<td>86</td>
</tr>
<tr>
<td>NA, 28, 93, 94, 146</td>
<td></td>
</tr>
<tr>
<td>na.omit()</td>
<td>96, 293</td>
</tr>
<tr>
<td>names()</td>
<td>19, 23, 130</td>
</tr>
<tr>
<td>ncol()</td>
<td>23</td>
</tr>
<tr>
<td>nrow()</td>
<td>23</td>
</tr>
<tr>
<td>NULL</td>
<td>19</td>
</tr>
<tr>
<td>page.rank()</td>
<td>251</td>
</tr>
<tr>
<td>pairwise.similarity()</td>
<td>275</td>
</tr>
<tr>
<td>par()</td>
<td>224</td>
</tr>
<tr>
<td>pbinom()</td>
<td>326, 341</td>
</tr>
<tr>
<td>pivot_longer()</td>
<td>98, 189, 311, 316, 317</td>
</tr>
<tr>
<td>pivot_wider()</td>
<td>42, 43, 51, 63, 98, 133, 189, 296</td>
</tr>
<tr>
<td>plot()</td>
<td>241, 244, 254</td>
</tr>
<tr>
<td>pnorm()</td>
<td>331, 334, 353, 398</td>
</tr>
<tr>
<td>position</td>
<td>260</td>
</tr>
<tr>
<td>power.prop.test()</td>
<td>415</td>
</tr>
<tr>
<td>power.t.test()</td>
<td>416, 417</td>
</tr>
<tr>
<td>print()</td>
<td>13, 17, 147</td>
</tr>
<tr>
<td>probs</td>
<td>77</td>
</tr>
<tr>
<td>prop.test()</td>
<td>402, 403, 406–408</td>
</tr>
<tr>
<td>pull()</td>
<td>26</td>
</tr>
<tr>
<td>punif()</td>
<td>324</td>
</tr>
<tr>
<td>purrr package</td>
<td>11, 340</td>
</tr>
<tr>
<td>qnorm()</td>
<td>353, 373, 388</td>
</tr>
<tr>
<td>qqnorm()</td>
<td>332</td>
</tr>
<tr>
<td>qqplot()</td>
<td>126</td>
</tr>
<tr>
<td>qas package</td>
<td>12, 39, 218</td>
</tr>
<tr>
<td>qt()</td>
<td>388</td>
</tr>
<tr>
<td>quantile()</td>
<td>77, 142, 181</td>
</tr>
<tr>
<td>range()</td>
<td>18</td>
</tr>
<tr>
<td>rbins()</td>
<td>339</td>
</tr>
<tr>
<td>RColorBrewer package</td>
<td>264</td>
</tr>
<tr>
<td>read.dta()</td>
<td>31</td>
</tr>
<tr>
<td>read.spss()</td>
<td>31</td>
</tr>
<tr>
<td>read.csv()</td>
<td>21, 22, 39, 40, 62, 65, 89</td>
</tr>
<tr>
<td>read.dta()</td>
<td>31</td>
</tr>
<tr>
<td>readr package</td>
<td>11</td>
</tr>
<tr>
<td>recode()</td>
<td>312</td>
</tr>
<tr>
<td>rename()</td>
<td>43</td>
</tr>
<tr>
<td>rep()</td>
<td>146</td>
</tr>
<tr>
<td>replace_na()</td>
<td>232</td>
</tr>
<tr>
<td>resid()</td>
<td>171, 182</td>
</tr>
<tr>
<td>return()</td>
<td>20</td>
</tr>
<tr>
<td>rgb()</td>
<td>262, 263</td>
</tr>
<tr>
<td>rio package</td>
<td>31</td>
</tr>
<tr>
<td>markdown package</td>
<td>33</td>
</tr>
<tr>
<td>rnorm()</td>
<td>363</td>
</tr>
<tr>
<td>row_number()</td>
<td>26</td>
</tr>
<tr>
<td>rowMeans()</td>
<td>129</td>
</tr>
<tr>
<td>rownames()</td>
<td>128</td>
</tr>
<tr>
<td>rowSums()</td>
<td>129, 297</td>
</tr>
<tr>
<td>rowwise()</td>
<td>239</td>
</tr>
<tr>
<td>runif()</td>
<td>324, 344</td>
</tr>
<tr>
<td>sample()</td>
<td>288, 306, 363, 393</td>
</tr>
<tr>
<td>save()</td>
<td>30</td>
</tr>
<tr>
<td>save.image()</td>
<td>30</td>
</tr>
<tr>
<td>scale()</td>
<td>131, 178, 179, 332</td>
</tr>
<tr>
<td>scale_alpha_identity()</td>
<td>263</td>
</tr>
<tr>
<td>scale_color_identity()</td>
<td>263</td>
</tr>
<tr>
<td>scale_color_manual()</td>
<td>117, 164, 249</td>
</tr>
<tr>
<td>scale_fill_discrete()</td>
<td>98</td>
</tr>
<tr>
<td>scale_fill_identity()</td>
<td>267</td>
</tr>
</tbody>
</table>
scale_shape_manual(), 117, 249
scale_x_continuous, 100
scale_x_continuous(), 117, 169
scale_x_discrete(), 97
scale_y_continuous(), 117, 169
sd(), 80, 235
select(), 24–27
seq(), 18, 19, 77, 160
seq_along(), 147
seq_len(), 340
set.seed(), 227
setwd(), 21
shadow_mark(), 271
shape, 116
sign(), 158
slice, 248
slice(), 24, 25
slice_max(), 226, 248
SnowballC package, 218
source(), 33
spread_predictions(), 207
sqrt(), 10, 17
starts_with(), 317
stat_function(), 348
state_length, 271
stemCompletion(), 224
str_c(), 147
str_replace(), 311, 317
str_replace_all(), 220
str_sub(), 220
str_to_lower(), 265
stringr package, 147, 218, 352
suffix, 176
sum(), 18, 44, 46, 49, 91
summarise(), 29
summarize, 316
summarize(), 29, 49, 51, 52, 63, 77, 80, 92, 105, 240
summarize_at(), 80
summary(), 28, 41, 65, 76, 89, 183, 196, 429, 431
swirl package, xx

t t.t.test(), 389, 403, 405, 406
table(), 228
tail(), 40
theme(), 164
theme_classic(), 97, 100, 155
theme_void(), 259
tibble, 21, 23, 24, 39, 40
tibble package, 11
tidy(), 132, 133, 169, 208, 219, 429–431
tidymodels package, 132, 133, 168
tidyrr package, 11, 201
tidytext package, 218, 221, 222
tm package, 218
tokenizers package, 220
transition_length, 271
transition_states(), 271
TRUE, 46

U
ungroup(), 91, 232, 294
unique(), 86, 90, 153, 154
unnest_tokens(), 220

V
V(), 252
var(), 80
vars(), 232
VCorpus(), 218
vertex.color, 254
View(), 23

W
weighted.mean(), 316
weightTfIdf(), 226, 228
where(), 297
while, 254
while(), 254, 255
widyr package, 275
wordcloud package, 223
wordcloud(), 223, 224
wordStem(), 220
write.dta(), 32
write_csv(), 31
write_dta(), 32

X
xlab(), 97
xlim(), 126

Y
ylab(), 97
ylim(), 126
ymd(), 153