Contents

Chapter 1

Introduction

> In God we trust; all others must bring data.
> —William Edwards Deming

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. For example, researchers examine racial discrimination in the labor market, evaluate the impact of new curricula on students' educational achievements, predict election outcomes, and analyze social media usage. Similar data-driven approaches have 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 past 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 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*, leading to a spike in the amount and diversity of available data. Information sharing makes 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, has meant that anyone can conduct data analysis using their personal computer and favorite data analysis software.

As a direct consequence of these technological changes, the sheer volume of data available to quantitative social scientists has grown rapidly. In the past, researchers largely relied on data published by governmental agencies (e.g., censuses, election outcomes, and economic indicators) as well as a small number of data sets collected by research groups (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. However, the wide variety of new data has significantly expanded the horizon of quantitative social science research. Researchers are 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 such as Sweden, a wide range of registers, including income, tax, education, health, and workplace, are available for academic research.

New data sets have emerged across diverse areas. Detailed data about consumer transactions are available through electronic purchasing records. International trade data are collected at the product level between many pairs of countries over several decades. Militaries have also contributed to the data revolution. During the Afghanistan war in the 2000s, 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 analyzing digitized texts as data, including legislative bills, newspaper articles, and the speeches of politicians. The availability of social media data through websites, blogs, tweets, short message service (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. They enable researchers to analyze the legislative redistricting process or civil conflict with attention paid to spatial location. Others have used satellite imagery data to measure the level of electrification in rural areas of developing countries. While still a rare approach, images, sounds, and even videos can be analyzed using quantitative methods for answering 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 policymakers, 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. 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 toward 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 have 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, this traditional approach cannot meet the current 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 the 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 writing this book. The book introduces the 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, 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, students will discover the excitement that data analysis brings. Those who want to learn Stata programming might also find this book useful, although here the emphasis is on how to use Stata to answer quantitative social science questions.

As mentioned previously, 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 societal problems 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 40 data sets tied to original publications. Readers are strongly encouraged to consult these publications to not only better understand the data with which they will be working, but to also gain exposure to new techniques and real-world applications of data analysis.

The book consists of eight chapters. The current introductory chapter explains how to best utilize the book and presents a brief introduction to Stata, a popular statistical programming environment. Stata is available in several forms and runs on Macintosh, Windows, and Linux computers. The examples in this book were created using Stata 14. With the exception of the text analysis parts of chapter 7, the code contained in this book should also run on both earlier and later versions.

This chapter ends with two exercises that are designed to help readers practice elementary Stata functionalities using data sets from published social science research. All data sets used in this book are freely available for download via links from `http://qss.princeton.press`. Links to other useful materials, such as the review exercises for each chapter, can also be found on the website. With the exception of chapter 7, the book focuses on basic syntax in Stata and does not introduce the wide range of additional packages that are available. However, upon completion of this book, readers will have acquired enough Stata programming skills to be able to utilize these packages.

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 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 résumés of fictitious job applicants to potential employers after randomly choosing applicants' names to sound either African American or Caucasian. 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.

Additionally, readers will learn about causal inference in observational studies where researchers do not have control over treatment assignment. The main application is a classic study whose goal was to figure out the impact of increasing the 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, but instead is subject to many factors, such as economic growth, that are themselves associated with employment. Since these factors influence which companies find themselves in 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 type of selection bias in observational studies. Despite the risk that we will inaccurately estimate treatment effects in observational studies, the results of such studies are often easier to generalize than those obtained from randomized controlled trials. Another example in chapter 2 includes a field experiment concerning social pressure in get-out-the-vote mobilization. Exercises then include a randomized experiment that investigates the causal effect of small class size in early education as well as a natural experiment about political leader assassination and its effects. In terms of Stata programming, chapter 2 covers logical statements and subsetting. This chapter also reviews measures of the spread of data, including quantiles and standard deviation.

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 attempted to measure the degree of support among Afghan citizens for international forces and for the Taliban insurgency during the Afghanistan war. The chapter explains the power of randomization in survey sampling. Specifically, random sampling of respondents from a population allows us to obtain a representative sample. As a result, we can infer the opinion of an entire population by analyzing one small representative group. 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 the 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. We present various ways to visualize univariate and bivariate data in Stata. We also introduce the concept of correlation and the Gini coefficient. 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 policymakers to act proactively. The main application of this chapter is the prediction of US presidential elections using preelection 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 toward 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 important programming concepts that enable easier and more efficient coding: macros and loops. The exercises at the end of the chapter include an analysis of whether betting markets can precisely forecast election outcomes.

Chapter 5 shifts the focus from data analysis to *probability*, a unified mathematical model of uncertainty. While earlier chapters examine how to estimate parameters and make predictions, they do not discuss the level of uncertainty in empirical findings, a topic introduced in chapter 6. 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 two dominant perspectives, frequentist and Bayesian. We then provide mathematical definitions of probability and conditional probability, and introduce several fundamental rules of probability. One such rule is called Bayes' rule. We show how to use Bayes' rule and 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 election predictions that we produced in chapter 4 using preelection polls. The chapter concludes by introducing 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 over repeated sampling as sample size increases. One exercise adds uncertainty to the forecasts of election outcomes based on betting market data. In a second exercise, we inspect the German cryptography machine from World War II (Enigma).

Chapter 6 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 on the groundwork laid in chapter 5, chapter 6 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 the use of hypothesis testing. In other words, the chapter concerns statistical inference. Our examples come 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 a linear regression model. To do this, 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.

Chapter 7 is about the *discovery* of patterns from less traditional types of data. When analyzing "big data," we need automated methods and visualization tools to identify consistent patterns in the data. First, we analyze network data, focusing on explaining the relationships among units. Within marriage networks in Renaissance Florence, we quantify the key role played by the Medici family. As a more contemporary example, various measures of centrality are introduced and applied to social media data generated by US senators on Twitter. We then examine 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 also 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 changes in spatial patterns over time.

Finally, in chapter 7, we analyze texts as data. Our primary application here is 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 paper of unknown authorship. The chapter brings together many of the basics reviewed in earlier chapters, while applying various data visualization techniques using specialized Stata packages.

The final chapter concludes by briefly describing the next steps readers might take on 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, which is based on the following principle:

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

This book is not just for reading. 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 via links from `http://qss.princeton.press`.

The book is cumulative. Later chapters assume that readers are already familiar with most of the materials covered in earlier parts. Hence, in general, it is not advisable to skip chapters. The exception is chapter 7, "Discovery," the contents of which include complex pieces of code and advanced techniques. 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. 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, the 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 exercises 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.[1] 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. Reading assignments introduce a particular example and a general discussion of new concepts to students. Classroom discussion then 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 have mastered the materials.

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

[1] Frederick Mosteller (1980). "Classroom and platform performance." *American Statistician*, vol. 34, no. 1 (February), pp. 11–17.

Chapter title	Chapter number	Weeks
Introduction	1	1
Causality	2	2–3
Measurement	3	4–5
Prediction	4	6–7
Probability	5	8–9
Uncertainty	6	10–12
Discovery	7	13–15

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

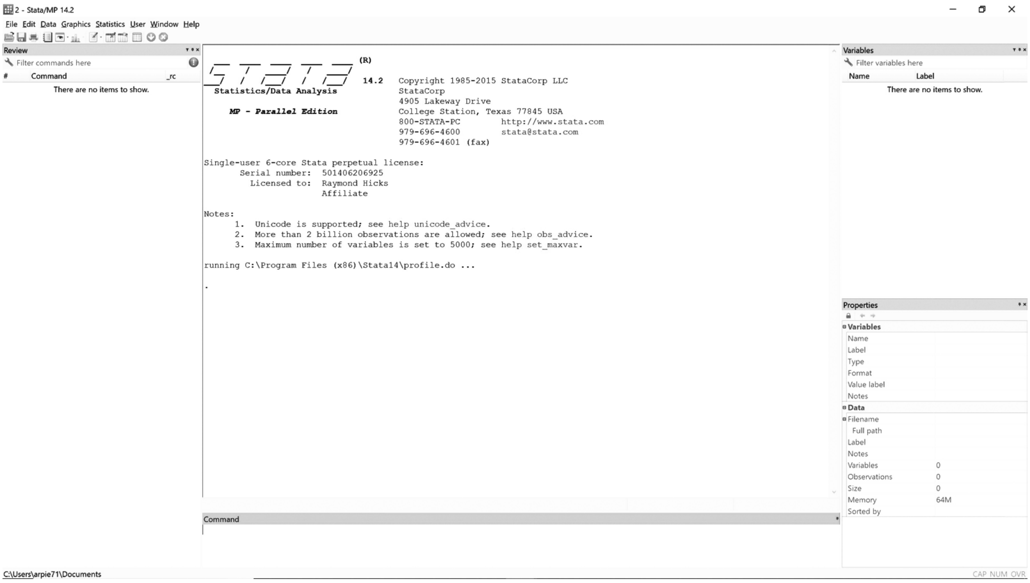
Finally, to ensure mastery of the basic methodological and programming concepts introduced in each chapter, we recommend that users first read a chapter and then practice all of the code it contains before attempting to solve the associated exercises.

1.3 Introduction to Stata

This section provides a brief, self-contained introduction to Stata that is a prerequisite for the remainder of this book. Stata is available from `http://www.stata.com` and has been around since 1985. The Stata Corporation prides itself on the software's reproducibility, so code that ran in earlier versions still runs in later versions. Stata is also constantly expanding its capabilities, adding new features and commands with each version. For some changes, files created in a newer version of Stata will not necessarily work with older versions. As previously mentioned, all the commands in this book work on Stata 14. Starting with version 13, Stata can handle very long strings. Prior versions were limited to 256 characters. Stata 15 was released in June 2017 and had some new features including the ability to create markdown files and improved graphics. Stata 16 was released while this book was in its final stages. This version allows for multiple data sets in memory, together with other new features.

Stata offers several options for purchase, including plans for business, government/nonprofits, education, and students. There are also three different flavors of Stata available, depending on the size of data one typically analyzes. Stata/IC can hold a maximum number of 2,048 variables and 2 billion observations. Stata/SE can handle up to 32,767 variables and include 10,998 variables in a regression. Finally, Stata/MP is the only version that takes advantage of multiprocessing. It can handle 120,000 variables and up to 20 billion observations. All three versions support the code used in this book.

Stata has an active community willing to answer questions about the software (`https://www.statalist.org/`). *The Stata Journal* (`https://www.stata-journal.com/`) likewise contains useful tips and advice, including articles that introduce new user-written


2 - Stata/MP 14.2
File Edit Data Graphics Statistics User Window Help
Review
Filter commands here
# Command _rc
There are no items to show.
Variables
Filter variables here
Name Label
There are no items to show.
Command
C:\Users\arpie71\Documents
CAP NUM OVR

```
 ___  ____  ____  ____  ____ (R)
/__    /   ____/   /   ____/
___/   /   /___/   /   /___/   14.2   Copyright 1985-2015 StataCorp LLC
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     MP - Parallel Edition            College Station, Texas 77845 USA
                                      800-STATA-PC        http://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

Single-user 6-core Stata perpetual license:
       Serial number:  501406206925
         Licensed to:  Raymond Hicks
                       Affiliate

Notes:
      1.  Unicode is supported; see help unicode_advice.
      2.  More than 2 billion observations are allowed; see help obs_advice.
      3.  Maximum number of variables is set to 5000; see help set_maxvar.

running C:\Program Files (x86)\Stata14\profile.do ...

.
```

Properties

Variables	
Name	
Label	
Type	
Format	
Value label	
Notes	
Data	
Filename	
Full path	
Label	
Notes	
Variables	0
Observations	0
Size	0
Memory	64M
Sorted by	

Figure 1.1. Screenshot of Stata (version 14).

commands and apply advanced analytic techniques. One of the main benefits of user-written commands from *The Stata Journal* is that they are peer reviewed.

Through numerous data analysis exercises, this book will teach you the basics of statistical programming, which will then 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. Also, we will only barely scratch the surface of what Stata is capable of, both in terms of the types of analysis possible and programming capacity. The goal is to equip readers with a foundational fluency in Stata programming that will enable them to progress to more advanced applications. Users will learn that there are often multiple ways to code the same solution to any one problem. Users interested in learning more about Stata's capabilities are encouraged to visit `https://www.stata.com/bookstore/books-on-stata/`.

In the remainder of this section, we cover four key topics: (1) arithmetic operations, (2) creating and modifying variables, (3) describing data, and (4) working with data files in Stata. Before we begin, it is necessary to first introduce Stata's user interface, captured in figure 1.1. The bottom-center Command window shows the console where we will directly type our commands. The left History window displays the commands that have been run in the current Stata session. The upper-center Results window shows the output of the executed commands. The upper-right Variables window lists all of the variables in the data set. Finally, the lower-right Properties window contains further information about the variables and dimensions of the data set, including the number of variables (columns) and number of observations (rows).

1.3.1 ARITHMETIC OPERATIONS

At its most basic level, Stata can be used as a calculator. We can enter any standard arithmetic operations in the Command window. For example, type `display 5 + 3` into the command line, then hit Enter on the keyboard. We insert the `display` command before the operation so Stata shows the result directly in the Results window. Simply entering `5 + 3` will generate an error message because a valid Stata command must always come before any additional operation or code. A list of commands used in this book is included in the appendix. Each command has an abbreviated version, which we also include in the appendix for the commands we review. But we use the full command names throughout this book for clarity.

```
. display 5 + 3
8
```

Stata ignores the spaces between arithmetic operators, so `display 5+3` will return the same result as `display 5 + 3`. However, we added a space before and after the operator because it is good practice to make your code as easy to read as possible. This not only helps others to follow your code but it can also help prevent errors. As this example illustrates, a period (.) in the Results window indicates the start of the Stata command, followed immediately by the output on the next line. Importantly, this period is not part of our syntax.

Let's try some other examples. It is important for readers to try these examples on their own. Remember that we can learn programming only by doing!

```
. display 5 - 3
2

. display 5 / 3
1.6666667

. display 5 ^ 3
125

. display 5 * (10 - 3)
35

. display sqrt(4)
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 we will see in later chapters, Stata has numerous other functions. Users can obtain more information on Stata commands and functions, including available subcommands and options, by typing `help` followed by the command or function of interest into the command line. They can also use the `Help` drop-down menu or search bar in the top right of the main interface.

1.3.2 VARIABLES

Information is most commonly stored in *variables*. A variable is an object that takes different values across different observations. Variables will correspond to factors deemed important to our main research questions. For example, if we are interested in studying the effects of after-school programs on academic achievement, we may have variables for test scores, type of after-school program, frequency of attendance, student age, school

Table 1.1. World Population Estimates.

Year	World population (thousands)
1950	2,525,779
1960	3,026,003
1970	3,691,173
1980	4,449,049
1990	5,320,817
2000	6,127,700
2010	6,916,183

Source: United Nations, Department of Economic and Social Affairs, Population Division (2013). World Population Prospects: The 2012 Revision, DVD Edition.

demographics, and so on. To demonstrate the fundamental role variables play in any subsequent quantitative analysis, we will create a sample data set using the `input` command. Typically, we would load a data set into Stata rather than directly typing in the values of variables, but we use this simple exercise as a demonstration.

Table 1.1 contains estimates of the world population (in thousands) over the past several decades. We can enter these data directly into Stata, creating one variable for year (`year`) and another for population (`worldpop`). To ensure we are starting with an empty data set, we use the `clear` command to remove all data currently stored in memory. The `input` command must be immediately followed by a variable list when no data are in memory. This list will establish the columns of the data set. Values for variables are then entered in the same sequence in which their corresponding variable is listed. For each observation (or row), each variable value is separated by a space before entering another variable's value. Rows must be entered individually in the command line. Stata will automatically number each row, so it is only necessary to enter the values (e.g., simply type `1950 2525779` for our first row). Finally, we use `end` to tell Stata our data entry is complete; otherwise it will continue to read any input in the command line as data.

```
. clear

. input year worldpop

          year   worldpop
  1. 1950 2525779
  2. 1960 3026003
  3. 1970 3691173
  4. 1980 4449049
  5. 1990 5320817
  6. 2000 6127700
  7. 2010 6916183
  8. end
```

The `year` and `worldpop` variables are now listed in the upper-right Variables window. We can use the `list` command to show all observations with their corresponding values for year and population. The `list` command is useful here to demonstrate the structure of the data, but it becomes inefficient once we have more observations. It is important to emphasize from the start that Stata is case sensitive. Stata will consider `year` and `Year` as two separate variables.

```
. list

     +-----------------+
     | year   worldpop |
     |-----------------|
  1. | 1950    2525779 |
  2. | 1960    3026003 |
  3. | 1970    3691173 |
  4. | 1980    4449049 |
  5. | 1990    5320817 |
     |-----------------|
  6. | 2000    6127700 |
  7. | 2010    6916183 |
     +-----------------+
```

Now that we have a sample data set, we can examine the variable characteristics. Variables can be either numeric or string. Numeric variables contain numbers, which can include integers, decimal points, negative numbers, and so on. Numeric variables have five different storage types in Stata: `byte`, `int`, `long`, `float`, and `double`. Knowing the storage type can be important when it comes to merging data, compressing memory, or retaining all digits in large numbers. String variables, as the name implies, are composed of a string of characters. Sometimes, Stata will read numbers as strings, which could generate errors when calculations and commands require numbers. We can use the `destring` command with the `replace` option to convert a string of numbers into numeric format (e.g., `destring` *`stringvar`*`, replace`). This type of command is often important when we are "cleaning the data," which is the preparation of data before it can be analyzed. Storage type and display format can be identified by using the `describe` command.

```
. describe

Contains data
  obs:             7
 vars:             2
----------------------------------------------------------------------
              storage  display    value
variable name   type   format     label      variable label
----------------------------------------------------------------------
year            float  %9.0g
worldpop        float  %9.0g
```

```
----------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

We can also use the `codebook` command to obtain more detailed summary information that will prove useful in later sections, such as the range of values, number of observations that are missing data on this variable, and the frequency of example values.

```
. codebook

----------------------------------------------------------------------
year
----------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [1950,2010]                  units:  10
         unique values:  7                        missing .:  0/7

            tabulation:  Freq.  Value
                             1  1950
                             1  1960
                             1  1970
                             1  1980
                             1  1990
                             1  2000
                             1  2010

----------------------------------------------------------------------
worldpop
----------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [2525779,6916183]            units:  1
         unique values:  7                        missing .:  0/7

            tabulation:  Freq.  Value
                             1  2525779
                             1  3026003
                             1  3691173
                             1  4449049
                             1  5320817
                             1  6127700
                             1  6916183
```

The results from running `codebook` tell us that, of the seven observations in our sample data set, none are missing values for `year`. In addition, `year` is stored as a numeric (`float`) variable, its values range from 1950 to 2010 with seven total unique values, and there is one observation for 1950, 1960, 1970, and so on. The `codebook` and `describe` commands both show stored variable and value labels, when available. Labels are discussed in the next section.

There are also different kinds of variables. Continuous variables, such as gross domestic product (GDP), can accommodate many values. In contrast, categorical variables have a limited number of values, where each value is distinctly linked to a particular category. For example, gender, race, and political party affiliation are all categorical variables. A *binary variable*, also called a *dichotomous* or *dummy variable*, is a type of categorical variable assigned only the values of 0 (false) or 1 (true). Ordinal variables also have a limited number of values, but the values denote some type of order. Level of education may be an ordinal variable, where high school graduates could be assigned a value of 1, college graduates a value of 2, and postgraduates a value of 3; here, higher values reflect higher levels of education.

So far, we created new variables by directly inputting them. However, we can also create new variables from existing variables. To create variables in Stata, we use the `generate` command, followed by the new variable name and assigning it a value using the equals sign (=). Variable names can be up to 32 characters in length. And while they can contain numeric characters, they must start with a letter or underscore (_). We create a new variable called `pm`, where we transform the population estimate into millions instead of thousands by dividing the `worldpop` variable by 1000. We can then see the resulting values using `list`.

```
. generate pm = worldpop / 1000

. list pm

     +----------+
     |       pm |
     |----------|
  1. | 2525.779 |
  2. | 3026.003 |
  3. | 3691.173 |
  4. | 4449.049 |
  5. | 5320.817 |
     |----------|
  6. |   6127.7 |
  7. | 6916.183 |
     +----------+
```

Arithmetic operations can be applied only to numeric variables, but multiple variables can be included in those operations. We can add, subtract, multiply, or divide two or more variables. Here, we 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 100,000 in one year and increased to 120,000 in the following year. In this case, we say, "the population increased by 20%."

For this calculation, it is useful to define lagged and forward values. *Lagged values* are any values that occur in a previous time period, while *forward values* (or *leads*) are those that occur in the future. To access these lagged or forward values in Stata, we can use its built-in indexing variable _n, which sequentially numbers each row or observation (e.g., the fifth row is numbered 5). To access the lagged value in the previous row, which is the prior decade in this instance, we simply subtract 1 from _n. To access data in proceeding rows, we add the number of desired leads to _n (e.g., _n + 1 refers to the value of the following decade).

When working with lags or leads, it is important that the data are in the correct order. Consequently, it is prudent to always sort the data by the time variable using the sort command before processing any code involving leads or lags. Running sort *variable* places observations in ascending order according to the specified variable. To compute the percentage increase for each decade, we use generate to create a new variable called lagworldpop that stores the lagged value of worldpop (i.e., the population value in the preceding decade). We subtract worldpop by its lagged value, divide that difference by the lagged value, and multiply the result by 100 to obtain the percentage increase.

```
. sort year

. generate lagworldpop = worldpop[_n - 1]
(1 missing value generated)

. generate pctincrease = ((worldpop - lagworldpop) / lagworldpop) * 100
(1 missing value generated)

. list pctincrease

     ┌──────────┐
     │ pctinc~e │
     ├──────────┤
  1. │        . │
  2. │ 19.80474 │
  3. │  21.9818 │
  4. │ 20.53212 │
  5. │ 19.59448 │
     ├──────────┤
  6. │ 15.16465 │
  7. │ 12.86752 │
     └──────────┘
```

We can assign variables new names using the rename command. This command is useful for shortening long variable names and giving variables meaningful names. Names from downloaded data sets are not always intuitive or clear (e.g., v4 may store race). We rename our measure of population in millions to the more informative name of popmillion.

```
. rename pm popmillion
```

In some instances, we may want to change or reassign the values, not the names, of an existing variable. Using the recode command, we can directly overwrite those values, or we can store the modified values as a new variable by immediately following the command with the generate() option. We previously used generate() as a command, but using it here as an option requires us to place the name of the new variable within the parentheses. To illustrate, we create a binary variable called coldwar, where 1 is assigned to years during the Cold War and 0 to the years after the Cold War.

```
. recode year (1950 1960 1970 1980 = 1) (1990 2000 2010 = 0), generate(coldwar)
(7 differences between year and coldwar)
```

We can also use the inlist() function to recode variables. The first argument (or value passed in the command within the parentheses) specifies the variable to be recoded. The values that follow indicate which values should be coded as 1 in a separate, newly created variable named coldwar2. The years excluded from the list will be recoded as 0.

```
. generate coldwar2 = inlist(year, 1950 1960 1970 1980)
```

Stata limits the total number of arguments that can be included in the inlist() function to 10 strings or 250 real numbers. In some cases, we will want to recode a variable to something other than 0 or 1, and this will require the if qualifier, which we introduce in chapter 2, along with conditional statements and the cond() function to demonstrate alternative ways of creating and recoding variables.

1.3.3 LABELS

To make our data set easier to understand, we can give variables meaningful names, but we can also assign descriptive labels to variables and to the values of categorical or ordinal variables.[2] *Variable labels* give a brief description of the contents of a variable. Variable labels help us keep track of what each variable means and are especially useful as the number of variables in a data set increases. As an example, we label the pctincrease variable with the description "Percentage increase for each decade." Variable labels can have up to 80 characters, after which they are truncated.

```
. label variable worldpop "World population (thousands)"

. label variable popmillion "Population in millions"

. label variable pctincrease "Percentage increase for each decade"
```

Value labels give descriptions to individual values of a variable. Value labels allow us to more readily interpret results and keep track of what a variable's values represent. For example, our coldwar variable has a value of 0 or 1. However, it is more useful to know what these values represent. A value of 0 was assigned to years following the end of the Cold War, while Cold War years have a value of 1. Consequently, we can create a set of labels using label define. We give this set of labels a name, followed by the label for each value. We create

[2] There is no need to add value labels to string variables or continuous variables, which will generally have too many values to label.

a set of labels called cwlabel, where 0 has the value label "After Cold War" and 1 "During Cold War."

```
. label define cwlabel 0 "After Cold War" 1 "During Cold War"
```

Once our set of labels is defined, we attach it to a variable using label values. In the label values command, we list our variable and then the label name we created. Value labels do not replace variable values; they only attach a description to the values. The codebook command shows us that coldwar remains a numeric variable, where the numeric values are still distinct from the newly assigned labels.

```
. label values coldwar cwlabel

. codebook coldwar

--------------------------------------------------------------------------
coldwar                                                     RECODE of year
--------------------------------------------------------------------------

                  type:  numeric (float)
                 label:  cwlabel

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  0/7

            tabulation:  Freq.   Numeric  Label
                             3         0  After Cold War
                             4         1  During Cold War
```

Because value labels are tied to specific numeric values and not to specific variables, we can use the same set of value labels on different variables. If we had a survey where many of the questions had "yes" or "no" responses, represented by values of 1 and 0, respectively, we could define a set of labels called yesno to apply to all of these questions. In some cases, we may need to change the label values for an already defined set of labels, such as cwlabel. If we assigned this label to multiple variables, we would have to tediously reassign new labels to each. Alternatively, we can simply add the modify option to the label define command to automatically update the values of an existing set of labels. In our example, we modify the respective labels for 0 and 1 to "Post-Cold War" and "Cold War." Notice that we did not have to use the label values command to attach the updated value labels.

```
. label define cwlabel 0 "Post-Cold War" 1 "Cold War", modify

. list coldwar

     +-------------+
     |     coldwar |
     |-------------|
  1. |    Cold War |
  2. │       Cold War │
  3. │       Cold War │
  4. │       Cold War │
  5. │  Post-Cold War │
     ├────────────────┤
  6. │  Post-Cold War │
  7. │  Post-Cold War │
     └────────────────┘
```

Finally, on occasion, we may forget the name of a variable, especially ones created by someone else. Stata offers the `lookfor` command that allows you to enter either an entire or partial string and it will return all variables that match in name or in label. This can be particularly useful in helping you identify the appropriate variables if there are several that are similar in name.

```
. lookfor pop

              storage   display    value
variable name   type    format     label      variable label
─────────────────────────────────────────────────────────────────────
worldpop        float   %9.0g                 World population (thousands)
popmillion      float   %9.0g                 Population in millions
lagworldpop     float   %9.0g
```

1.3.4 DESCRIBING THE DATA

For empirical analysis, we often need to know some basic information about our variables. We have already reviewed the `codebook` command, which tells us the variable's type as well as its range, number of unique values, label name, and example values when there are a larger number of observations. The `summarize` command is also very useful for providing a quick overview of the variables, displaying the variable mean, minimum and maximum values, and the number of observations with nonmissing data. It also shows the standard deviation, which we discuss more fully in chapter 2.

```
. summarize year worldpop pctincrease

    Variable │       Obs        Mean    Std. Dev.       Min        Max
─────────────┼────────────────────────────────────────────────────────
        year │         7        1980    21.60247       1950       2010
    worldpop │         7     4579529     1625004    2525779    6916183
 pctincrease │         6    18.32422    3.516155   12.86752    21.9818
```

For categorical and ordinal variables, `tabulate` is another frequently used command that displays the frequency distribution of the various categories. We could use `tabulate` on string and continuous variables. However, when there are a large number of unique values, `tabulate` becomes inefficient.

```
. tabulate coldwar

   RECODE of |
        year |      Freq.     Percent        Cum.
-------------+-----------------------------------
Post-Cold War |          3       42.86       42.86
    Cold War |          4       57.14      100.00
-------------+-----------------------------------
       Total |          7      100.00
```

We can also create a *contingency table* (also called a *cross-tabulation*) between two variables, which will show us their joint distribution.

```
. tabulate year coldwar

           |     RECODE of year
      year | Post-Cold   Cold War |     Total
-----------+----------------------+----------
      1950 |         0          1 |         1
      1960 |         0          1 |         1
      1970 |         0          1 |         1
      1980 |         0          1 |         1
      1990 |         1          0 |         1
      2000 |         1          0 |         1
      2010 |         1          0 |         1
-----------+----------------------+----------
     Total |         3          4 |         7
```

Further, we can combine the `tabulate` command with the `summarize()` option to calculate summary statistics by category. This is especially appropriate if you have one categorical and one continuous variable.

```
. tabulate coldwar, summarize(popmillion)

  RECODE of |  Summary of Population in millions
       year |        Mean   Std. Dev.       Freq.
------------+------------------------------------
  Post-Cold |   6121.5667   797.70079           3
   Cold War |    3423.001   834.12731           4
------------+------------------------------------
      Total |   4579.5292    1625.004           7
```

1.3.5 DATA FILES

Before interacting with data files, we must ensure that they reside in the *working directory*. The working directory is the file path (folder) that contains the files we want to access. By default, Stata will load data from and save data to this directory. There are different ways to change the working directory. In Stata, the default working directory is shown along the bottom of the screen (see figure 1.1). Often, however, the default directory is not the directory we want to use. To change the working directory, we can use Stata's drop-down menu `File> Change Working Directory` and pick the folder from which we want to work. The `ls` command will display that directory's files and folders.

It is also possible to change the working directory using the `cd` command by specifying the full path to the folder of our choosing as a character string. Entering the `cd` command on its own, without an input, will display the current working directory. The syntax `cd "/Desktop/qss/introduction"` would set the working directory to the `introduction` subfolder. If we previously set our working directory to `cd "/Desktop/qss"`, we could simply use `cd introduction` to navigate to the `introduction` subfolder. So far, the only data we have used has been manually entered into Stata. But most of the time, we will load data from an external file. `.dta` files are data sets in Stata format. These can contain both variable labels that describe the variable and value labels that tell us how values of a variable are coded. `.dta` files might also have notes attached to them with information about the data set or the variables, such as data sources.

Once we have set our working directory, it is much easier to call upon files and avoid unnecessary retyping. Suppose that the United Nations population data in table 1.1 are saved as Stata file `UNpop.dta`, which resembles the information below.

```
year worldpop
1950 2525779
1960 3026003
1970 3691173
1980 4449049
1990 5320817
2000 6127700
2010 6916183
```

We open the file with the `use` command, which will load the data into our Stata session. Because we have already specified the working directory and the file is already in Stata format, we can simply call on the file by name. It is not necessary to type the entire path. When opening a data file with `use`, it is advisable to add the `clear` option. If there have been changes to the data set in memory, Stata will not load a new file unless those data are saved or the `clear` option is specified. We can also use the drop-down menu `File > Open > *.dta`, which will present us with a prompt to save if there are currently unsaved data in memory.

```
. use UNpop, clear
```

A data set is a collection of variables, but we can think of it like a spreadsheet. It is often useful to visually inspect the data and its structure. We can view a spreadsheet-like representation of the data set in Stata by clicking on the `Data Editor (Edit)` or `Data Editor (Browse)` button underneath the drop-down menus (see figure 1.1). This will open a new window displaying the data. Alternatively, we can use the `browse` or `edit` commands.

Another common file type that we may encounter are CSV (comma-separated values) files. These represent tabular data and are conceptually similar to a spreadsheet of data values like those generated by 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. We can read a CSV file into Stata by going to the main drop-down menu (see figure 1.1) and clicking `File > Import > Text data (delimited, *.csv, ...)`.

We could also use the `import delimited` command. The syntax that follows loads the data into Stata's memory. Again, if we do not set the working directory, we have to specify the entire file path in quotations. Options for tailoring imported data can be found by consulting `help import delimited`. Note that when the file is not in Stata format, it is necessary to include the extension (e.g., `.csv`).

```
. import delimited using UNpop.csv, clear
(2 vars, 7 obs)
```

Additional software options, such as `StatTransfer`, and packages within Stata itself, including `usespss`, enable users to convert a variety of file formats into Stata-readable `.dta` files. Users may also find it easier to save data from other statistical software programs as CSV or ASCII files that they can then import into Stata.

Saving data in Stata is straightforward. We just enter the command `save` *`filename`*. Should the file name already exist, we have to include the `replace` option to save over the existing file (`save` *`filename`*`, replace`). We can also use the drop-down menu `File` and select either Save or Save As. To delete a data set, we use `erase` *`filename`*. Deleting or saving over original data sets is not advised. Instead, when we make any changes to a data set, it should be saved as a new file, leaving our original (raw) data file intact.

1.3.6 MERGING DATA SETS IN STATA

Unlike in some programming packages, it is not possible to simultaneously load multiple data sets in versions of Stata that predate its most recent release (Stata 16). And sometimes, the data you need will come from various files. Consequently, two key commands you will use to combine data sets are `merge` and `append`. When we want to add additional columns (or variables) to our data set, we use `merge`. For example, we may have data on the same people or the same states in two different data sets, but different variables are contained in these files. There must be an overlap on a shared key variable or identifier, such as id number or name, to make a meaningful merger. These shared variables can be numeric or string, but values for strings must share the same exact characters and case to be matched. Although not reviewed in this book, Stata can also conduct more complex mergers based on fuzzy matches of string values.

The command structure for merge is `merge n:n` *`varlist`* `using` *`filename`*, where *varlist* includes the shared variables or identifier on which the merger will be based. The n:n argument takes on the value of either `1` or `m` (many) depending on whether the listed variables uniquely identify observations in the master data (represented by the character before the colon) or the using data (represented by the character after the colon). The `1:1` specification clarifies that we expect each observation in the master data set (the file that is open when we execute the command) to have exactly one match in the using file (specified by *filename* in the general structure). Stata accommodates matches that are not one-to-one by specifying `m`. An example is if we have a data set that contains both states and congressional districts. Our second data set may have state-level factors, including economic and demographic measures. Because there are multiple observations of each state (one observation for each congressional district) in the master data set, we would conduct a many-to-one match (`m:1`), which would merge in all of the state-level variables (creating new columns) for each row of that state.[3]

Let's take the `UNpop.dta` data set, which should currently be in memory, and merge in subpopulation figures from the data set `UNsubpop.dta`. This data set breaks down the world population by region. The shared identifier variable is year. By default, the type of match is stored in the variable `_merge`. With a normal merge, the `_merge` variable will have three values:

- 1: The observation was in the master data but not the using data.
- 2: The observation was in the using data but not the master data.
- 3: The observation was in both data sets.

We can use this information to evaluate the success of the merge. If we expect all observations to merge but there are some that have a value of 1 or 2 on the `_merge` variable, we can investigate those observations to figure out why they did not merge.

```
. merge 1:1 year using UNsubpop

    Result                           # of obs.
    ─────────────────────────────────────────

    not matched                              3
        from master                          0  (_merge==1)
        from using                           3  (_merge==2)

    matched                                  7  (_merge==3)
    ─────────────────────────────────────────
```

The output tells us that each year in the master data set was successfully matched to an observation in the using (subpopulation) data set. We also see that there were three unmatched observations in the using data set. This is because there are three years (1985, 1995, and 2005) that exist in the subpopulation data set with no match in the master data set. Stata will nonetheless merge in these observations, creating additional rows. However, these observations will have missing values for the variable `worldpop`.

[3] While a `m:m` is possible, even the Stata documentation does not recommend it, so in most cases the merge will be `1:1`, `1:m`, or `m:1`.

It is important to note that Stata will not change the master data unless otherwise instructed and will never replace existing values in the master data with missing data from the using data. This means that if we have the same variable in both the master and using data, Stata will ignore values of this variable in the using data, even if there are missing values in the master data. We can override this behavior with the `update` option, which replaces missing values in the master data with nonmissing values in the using data. If we additionally specify the `replace` option, Stata will replace values in the master data with nonmissing values in the using data. It will not, however, replace nonmissing values in the master data with missing values from the using data. If either `update` or `replace` are specified, the `_merge` variable can take on two additional values:

- 4: The observation was in both data sets and missing values in the master data were updated with the using data.
- 5: The observation was in both data sets and nonmissing values in the master data were replaced by nonmissing values in the using data.

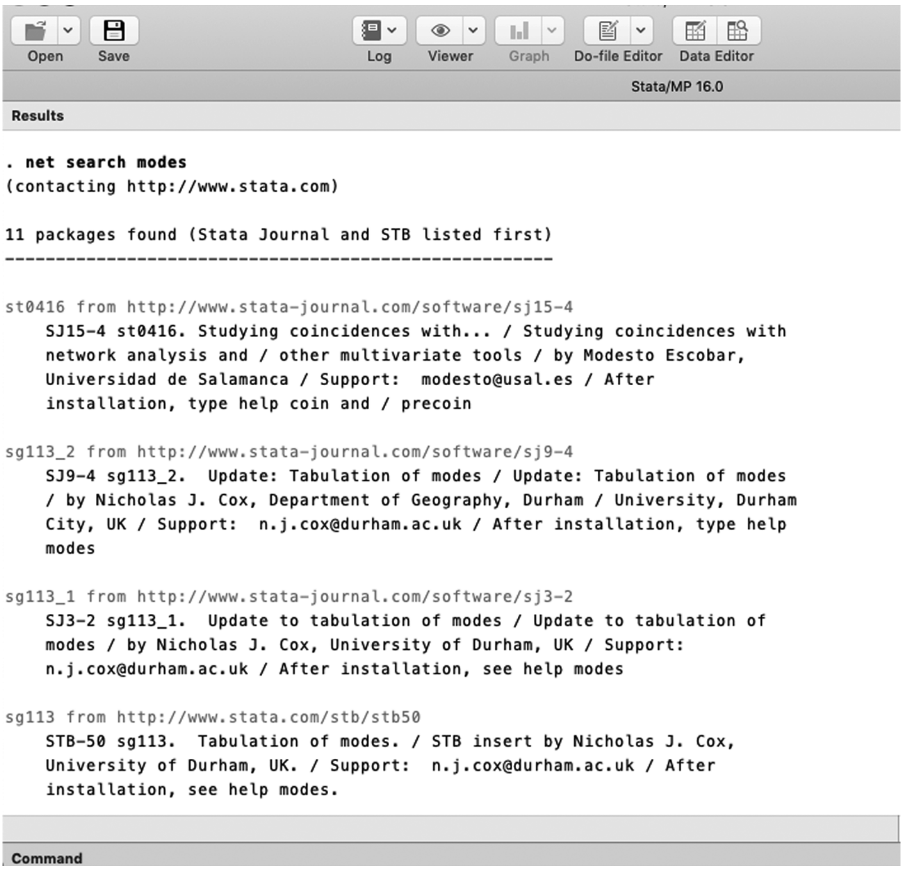
When we want to add rows to a data set where there is no overlap in values on the key identifier variable, we use the `append` command. For example, we may have yearly economic data stored in separate files by country. Because countries, the key identifier, will vary across files, we must append because merge is not possible. When one file is open, others can be appended with the command structure `append` *`[varlist]`* `using` *`filename`*. The command will add observations from a using data set to the end of the master data set. Including a variable list will only add the specified variables, but users can append an entire using file by omitting a variable list from the command. After running the `append` command, the number of observations will be equal to the number from the master data plus the number from the using data.[4]

Two things are important to remember about both the `merge` and `append` commands. First, prior to using the commands, one data set must already be open in Stata and it will be considered the master data set. Second, the key identifier variable must be sorted. This means that values of the variables on which the combining of files is based must be in sequential (numeric variables) or alphabetical (string variables) order. As previously introduced, we use the `sort` command to put our observations in ascending order. We can also use the `gsort` command if we need to place our observations in descending order, placing a minus sign before the sort variable (i.e., `gsort` *`-varlist`*). Observations in the current data set are ordered by year. Newer versions of Stata will automatically sort the master and using data.

1.3.7 PACKAGES

One of Stata's strengths is the existence of a large community of users who contribute various functionalities as Stata *packages*. Packages provide additional commands that are not included in the base version of Stata. Some of these packages first appeared in *The Stata Journal* (`http:/www.stata-journal.com`) or are available from the Statistical Software Components (SSC) archive at Boston College. Throughout this book, we will employ various

[4] Though not used in this book, Stata offers a third way to combine two data sets using the `joinby` command, which forms all pairwise combinations within groups defined by the variables specified in the command.



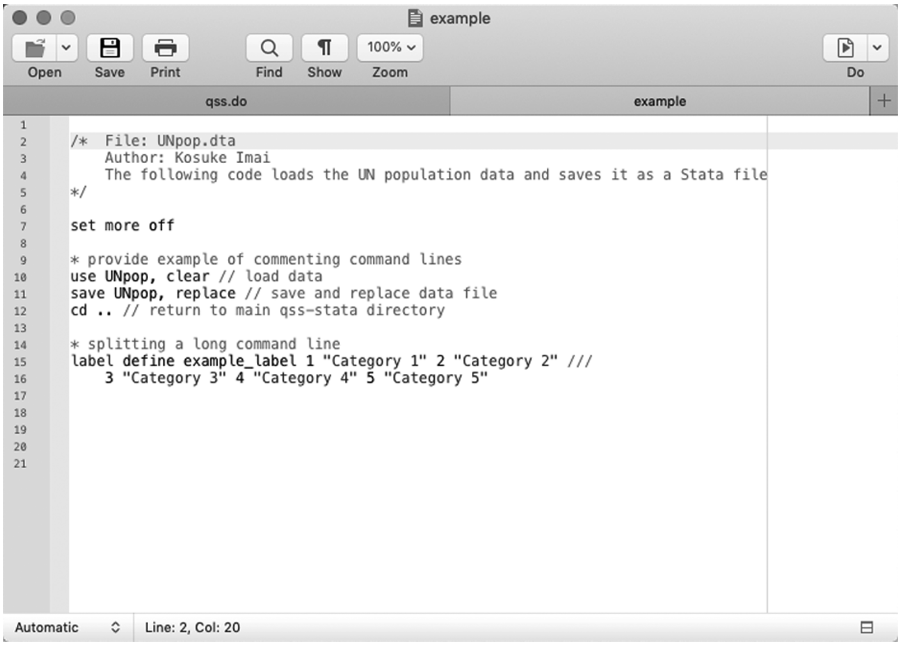
```
. net search modes
(contacting http://www.stata.com)

11 packages found (Stata Journal and STB listed first)
------------------------------------------------------

st0416 from http://www.stata-journal.com/software/sj15-4
    SJ15-4 st0416. Studying coincidences with... / Studying coincidences with
    network analysis and / other multivariate tools / by Modesto Escobar,
    Universidad de Salamanca / Support:  modesto@usal.es / After
    installation, type help coin and / precoin

sg113_2 from http://www.stata-journal.com/software/sj9-4
    SJ9-4 sg113_2.  Update: Tabulation of modes / Update: Tabulation of modes
    / by Nicholas J. Cox, Department of Geography, Durham / University, Durham
    City, UK / Support:  n.j.cox@durham.ac.uk / After installation, type help
    modes

sg113_1 from http://www.stata-journal.com/software/sj3-2
    SJ3-2 sg113_1.  Update to tabulation of modes / Update to tabulation of
    modes / by Nicholas J. Cox, University of Durham, UK / Support:
    n.j.cox@durham.ac.uk / After installation, see help modes

sg113 from http://www.stata.com/stb/stb50
    STB-50 sg113.  Tabulation of modes. / STB insert by Nicholas J. Cox,
    University of Durham, UK. / Support:  n.j.cox@durham.ac.uk / After
    installation, see help modes.
```

Figure 1.2. Screenshot of the Net Search Package Results. The results show the original package as well as updates.

packages. Packages from the SSC archive can be installed directly into Stata by typing `ssc install` *`packagename`*. For packages not available in the SSC archive, we use the `net search` command.

The **modes** package provides a simple illustration. Typing `net search modes` in the command line will bring up a number of options in the Results window, shown in figure 1.2. We can identify the original version of the **modes** package as *sg113* and subsequent updates are numbered with an underscore (e.g., *sg113_2*). Clicking on the link will give us the option to install the package. We can similarly download packages using the `findit` command, which will open and display the results in the Viewer window. However, `findit` is far more thorough, as it conducts both a net search and a search through Stata's documentation.

Once the package is installed, its capabilities will be available immediately and upon opening each new instance of Stata. There is no need to call each user-written package as a library. We can obtain details on installed packages and their customized commands using `help` *`packagename`*. Typing `ado dir` will list all of the packages currently installed. And we can update all of our user-written packages by typing `adoupdate`.



```

/*  File: UNpop.dta
    Author: Kosuke Imai
    The following code loads the UN population data and saves it as a Stata file
*/

set more off

* provide example of commenting command lines
use UNpop, clear // load data
save UNpop, replace // save and replace data file
cd .. // return to main qss-stata directory

* splitting a long command line
label define example_label 1 "Category 1" 2 "Category 2" ///
    3 "Category 3" 4 "Category 4" 5 "Category 5"
```

Figure 1.3. Screenshot of the Stata Do-file Editor. Once we open a Stata do-file, the Do-file Editor will appear in a separate window. It can then be used to write our code.

1.3.8 PROGRAMMING AND LEARNING TIPS

We conclude this brief introduction to Stata by providing several practical tips for learning how to program in Stata's language. First, we should use a text editor like the one that comes with Stata to write our code rather than directly typing it into the command line. If we just want to see what a command does, or quickly calculate some quantity, we can go ahead and enter it directly into the command line. However, for more involved programming, it is always better to use the text editor and save our code as a text file with the `.do` file extension. This way, we can keep a record of our code and run it again whenever necessary.

To create a do-file, use the drop-down menu `Window > Do-file Editor > New Do-file Editor` or click the `New Do-file Editor` icon (a white square with lines and a pencil). Either approach will open a blank document for text editing in a new window where we can start writing our code (see figure 1.3). To run our code from the Do-file Editor, press the `Execute (do)` icon in the top-right corner. To run only parts of the code, highlight those lines and then press the `Execute (do)` icon. In Windows, Ctrl+d is an alternative shortcut for executing the code. The equivalent shortcut for Mac OS is Command+Shift+d. Finally, we can also run the entire code in the background (i.e., the code will not appear in the Results console) by using the drop-down menu `Tools > Execute quietly (Run)`, or by typing Ctrl+r in Windows or Command+Shift+r for Mac. Results for individual lines of code can likewise be suppressed by adding `quietly` to the beginning of the command. To execute a do-file using the command line, we use the `do` command followed by the file name.

```
. * run example do-file
. quietly do example.do
```

As the length of our code increases, it is advised to place the command `set more off` at the top of your do-file. This will allow the code to run without interruption, assuming there are no other errors. If this command is omitted, Stata will only run some of the code and will prompt you to click “More” in the Results window before it will continue onto the next chunk of code. Depending on the size of the data set and version of Stata in use, it may also be necessary to increase the memory size (`set memory`) or the maximum number of variables (`set maxvar`).

It is sometimes useful to record a log of all activity. Typing `log using UNpop` will save all commands and output that appear in the Results window as an SMCL Stata text file. You can specify the path if you would like to save the log somewhere other than in your working directory. To close the log, simply type `log close`. Alternatively, you can open and close a log by using the `File > Log` drop-down menu, which also offers the option to view logs.

To make our code more accessible to others and readable to ourselves, we can annotate it. Adding clear comments within do-files is a good habit to establish early on, especially as our code gets more complicated. This becomes invaluable when you may later revisit the code or share your do-file with others. Stata offers several ways to write comments. First, we can place the comment within `/*` and `*/` characters. This combination is typically used to block out multiple lines of text or to deactivate larger chunks of code. Stata will consider all text following the initial `/*` as a comment, so it is necessary to add the ending `*/` when the comment ends to indicate that any further code should continue to run. We can use a simple asterisk * to leave a short comment (perhaps a short description of the code that will follow) or to deactivate a line of code. Similarly, comments can also follow double slashes `//` and these are particularly useful when added to the end of single command lines. Triple slashes `///` can be used to split a command line over multiple lines. Note that it is necessary to add a space between the command line and the comment indicators `//` and `///`. It is good practice to indent any lines that continue from an original command line, as illustrated in figure 1.3.

For further clarity, it is important to follow a certain set of coding rules. For example, we should use informative names for files, variables, and functions. Systematic spacing and indentation are also essential. In the preceding examples, we place spaces around all binary operators such as =, +, and −, and always add a space after a comma.

Finally, starting with version 15, Stata allows Markdown, which enables us to easily embed Stata code and its output within a document using straightforward syntax in a plain-text format. The resulting documents can be produced in HTML, PDF, or even Microsoft Word format. Because Stata Markdown embeds Stata code as well as its output, the results of data analysis presented in documents are reproducible. For more information, see `https://www.stata.com/new-in-stata/markdown/`.

1.4 Summary

This chapter began with a discussion of the important role that quantitative social science research can play in today’s data-rich society. To make contributions to this society through data-driven discovery, we must learn how to analyze data, interpret the results, and communicate our findings to others. To start our journey, we presented a brief introduction to Stata, which is a powerful programming tool for data analysis. The remaining pages of this chapter are dedicated to exercises, designed to ensure that you have mastered the chapter's contents.

1.5 Exercises

1.5.1 BIAS IN SELF-REPORTED TURNOUT

Surveys are frequently used to measure political behavior such as voter turnout, but some researchers are concerned about the accuracy of self-reports. In particular, they worry about possible *social desirability bias* where, in postelection surveys, respondents who did not vote in an election lie about not having voted because they may feel that they should have voted. Is such a bias present in the American National Election Studies (ANES)? ANES is a nationwide survey that has been conducted for every election since 1948. ANES is based on face-to-face interviews with a nationally representative sample of adults. Table 1.2 displays the names and descriptions of variables in the `turnout.dta` data file.

1. Load the data into Stata and check the dimensions of the data. How many observations are there? How many variables? Also, obtain a summary of the data. What is the range of years covered in this data set?

2. Calculate the turnout rate for each year based on the voting age population or VAP. Note that for this data set, we must add the total number of eligible overseas voters, since the `vap` variable does not include these individuals in the count. Next, calculate the turnout rate using the voting eligible population or VEP. What difference do you observe?

3. Compute the differences between the VAP and ANES estimates of turnout rate. How big is the difference on average? What is the range of the differences? Conduct the same comparison for the VEP and ANES estimates of voter turnout. Briefly comment on the results.

4. Compare the VEP turnout rate with the ANES turnout rate separately for presidential elections and midterm elections. Note that the data set excludes the year 2006. Does the bias of the ANES estimates vary across election types?

5. Divide the data into half by election years such that you subset the data into two periods and label the value of the two periods as Early and Later years. Summarize the difference between the VEP turnout rate and the ANES turnout rate for each period. Has the bias of ANES increased over time?

6. ANES does not interview prisoners and overseas voters. Calculate an adjustment to the 2008 VAP turnout rate. Begin by subtracting the total number of ineligible felons and noncitizens from the VAP to calculate an adjusted VAP. Next, calculate an adjusted VAP turnout rate, taking care to subtract the number of overseas ballots

Table 1.2. US Election Turnout Data.

Variable	Description
`year`	election year
`anes`	ANES estimated turnout rate
`vep`	voting eligible population (in thousands)
`vap`	voting age population (in thousands)
`total`	total ballots cast for highest office (in thousands)
`felons`	total ineligible felons (in thousands)
`noncitizens`	total noncitizens (in thousands)
`overseas`	total eligible overseas voters (in thousands)
`osvoters`	total ballots counted by overseas voters (in thousands)

counted from the total ballots in 2008. Compare the adjusted VAP turnout with the unadjusted VAP, VEP, and the ANES turnout rate. Briefly discuss the results.

1.5.2 UNDERSTANDING WORLD POPULATION DYNAMICS

Understanding population dynamics is important for many areas of social science. We will calculate some basic demographic quantities of births and deaths for the world's population from two time periods: 1950 to 1955 and 2005 to 2010. We will analyze the following Stata data files: `kenya.dta`, `sweden.dta`, and `world.dta`. The files contain population data for Kenya, Sweden, and the world, respectively. Table 1.3 presents the names and descriptions of the variables in each data set. The data are collected for a period of five years where *person-year* is a measure of the time contribution of each person during the period. For example, a person who lives through the entire five-year period contributes five person-years, whereas someone who lives only through the first half of the period contributes 2.5 person-years.

1. With the exception of version 16, Stata stores only one data set in memory at a time. To avoid having to open each country file individually for each calculation, append the three data files. The variable `country` distinguishes the observations from each data set. Inspect the appended data set by using the `browse` command or by clicking on the `Data Editor (Browse)` button. This allows you to view the data in a spreadsheet-like form.

2. Now compute the *crude birth rate* (CBR) for a given period. The CBR is defined as

$$\text{CBR} = \frac{\text{number of births}}{\text{number of person-years lived}}.$$

Calculate the CBR for each period, separately for Kenya, Sweden, and the world. Here, you can use the `bysort` command (see also section 3.6.1) to make separate

Table 1.3. Fertility and Mortality Estimate Data.

Variable	Description
`country`	abbreviated country name
`period`	period during which data are collected
`age`	age group
`births`	number of births (in thousands), i.e., the number of children born to women of the age group
`deaths`	number of deaths (in thousands)
`pymen`	person-years for men (in thousands)
`pywomen`	person-years for women (in thousands)

Source: United Nations, Department of Economic and Social Affairs, Population Division (2013). World Population Prospects: The 2012 Revision, DVD Edition.

calculations for regions and periods. For example, any command preceded by `bysort country`: will perform the calculation separately for each country. Using `bysort country period`: will provide the calculation separately for each period in each country.

Start by adding the person-years for men and women to create a variable for total person-years. Then, sum the total number of person-years and births across all age groups within each country and period using `bysort country period`: and creating two additional variables (`py_cnty_yr` and `birth_cnty_yr`, respectively). Calculate the CBR and assign appropriate labels to all variables. Finally, compute summary statistics within each country and period using cross-tabulation and briefly describe the patterns you observe.

3. The CBR is easy to understand but contains both men and women of all ages in the denominator. We next calculate the *total fertility rate* (TFR). Unlike the CBR, the TFR adjusts for age compositions in the female population. To do this, we need to first calculate the *age-specific fertility rate* (ASFR), which represents the fertility rate for women of the reproductive age range $[15, 50)$. The ASFR for the age range $[x, x+\delta)$, where x is the starting age and δ is the width of the age range (measured in years), is defined as

$$\text{ASFR}_{[x,\ x+\delta)} = \frac{\text{number of births to women of age } [x,\ x+\delta)}{\text{number of person-years lived by women of age } [x,\ x+\delta)}.$$

Note that square brackets, [and], include the limit whereas parentheses, (and), exclude it. For example, $[20, 25)$ represents the age range that is greater than or equal to 20 years old and less than 25 years old. In typical demographic data, the age range δ is set to 5 years.

Compute the ASFR for Sweden, Kenya, and the entire world for each of the two periods and save as variable `asfr`. Assign `asfr` a meaningful variable label. Then, using the `bysort country:` command, summarize the ASFR over age groups and period. What does the pattern of the ASFRs say about reproduction among women in Sweden and Kenya?

4. Using the ASFR, we can define the TFR as the average number of children that women give birth to if they live through their entire reproductive age:

$$\text{TFR} = \text{ASFR}_{[15,\,20)} \times 5 + \text{ASFR}_{[20,\,25)} \times 5 + \cdots + \text{ASFR}_{[45,\,50)} \times 5.$$

To begin, multiply the ASFR by 5 because the age range is 5 years and save as a new variable called `asfr5`. Compute the TFR for Sweden and Kenya as well as the entire world for each of the two periods using `bysort country period:` with the `egen` command and `sum()` function. We review the `egen` command in section 2.2.3. It provides an extension to the `generate` command, offering functions such as `sum()`, which will calculate the total sum of any variable specified within the parentheses. Here, we want to place our new variable `asfr5` within the parentheses.

We also want to limit the calculation to only reproductive women. This requires recoding `age` into a binary variable called `reproductive`, which indicates whether the age category is of reproductive age. As in the previous question, continue to assume that the reproductive age range of women is $[15, 50)$. We confine our calculation of TFR to only women of reproductive age by including a conditional statement using the `if` qualifier at the end of our `egen` command (i.e., `egen tfr = sum(asfr5) if reproductive == 1`). We discuss conditional statements and the `if` qualifier in the next chapter.

Label the new binary and TFR variables. In general, how has the number of women changed in the world from 1950 to 2000? What about the total number of births in the world?

5. Next, we will examine another important demographic process: death. Compute the *crude death rate* (CDR), which is a concept analogous to the CBR, separately for each region in each period. The CDR is defined as

$$\text{CDR} = \frac{\text{number of deaths}}{\text{number of person-years lived}}.$$

Label any newly created variables. Briefly describe the patterns you observe in the resulting CDRs.

6. One puzzling finding from the previous question is that the CDR for Kenya during the period 2005–2010 is about the same level as that for Sweden. We would expect people in developed countries like Sweden to have a lower death rate than those in developing countries like Kenya. While it is simple and easy to understand, the CDR does not take into account the age composition of a population. We therefore

compute the *age-specific death rate* (ASDR). The ASDR for age range $[x, x+\delta)$ is defined as

$$\text{ASDR}_{[x,\ x+\delta)} = \frac{\text{number of deaths for people of age } [x,\ x+\delta)}{\text{number of person-years of people of age } [x,\ x+\delta)}.$$

Calculate the ASDR and multiply it by 1000 to display results in the thousands. Create a command to summarize the ASDR for each age group in each period, separately for each country. Focusing on just the 2005–2010 period, briefly describe the pattern you observe.

General Index