A desk reference for political data preprocessing and management

Damon C. Roberts

A desk reference for political data preprocessing and management

Damon C. Roberts

Last Compiled: 2023

Table of contents

Preface 1

1 Introduction 3
1.1 Plan of the book . . . 5

2 Common programming tools, their benefits and drawbacks 7
2.1 Measuring tool performance . . . 8
2.2 Concepts that transverse (most) tools . . . 11
2.3 Programming language options, setup, and comparisons . . . 14
2.3.1 Installing an IDE, `VSCode` . . . 15
2.3.2 `R` . . . 15
2.3.3 Illustration of `OOP` in `R` . . . 17
2.3.4 `Python` . . . 20
2.3.5 `Julia` . . . 25
2.3.6 SQL via Duckdb . . . 29
2.4 Closing out the chapter . . . 36

3 A principled workflow for secure and replicable data management 37
3.1 Examples of current data management workflows . . . 40
3.2 A principled data management workflow . . . 44
3.3 Conclusions . . . 48

4 The basics of data extraction 51
4.1 Data extraction . . . 53
4.2 Loading CSV files . . . 53
4.2.1 Python . . . 54
4.2.2 R . . . 55
4.2.3 DuckDB via Python API . . . 57

References 59

List of Figures

2.1 Visualizing a function 12
2.2 Visualizing a library of functions 13

3.1 A principled workflow for data management 45

4.1 A principled workflow for data management 52

Preface

This book was originally inspired by a blog post that I had written and tried to find a more formal outlet for. When writing the blog post, a nagging thought kept hanging in the back of my mind, “What if you write this as a book?” I knew that this was a dangerous thought. At the time I was preparing for the academic job market and was in the throes of writing my dissertation (another book project). What a dumb idea. But, I was passionate about the topic and needed something to do when I needed a break from my dissertation. So, this project was born.

1 Introduction

As I often teach my students in introductory quantitative research methods classes, each and every choice you make for a study has deep implications for the way that you interpret results, what results you come up with, and the conclusions you draw from them. This book is not going to focus on the particular estimator one may choose to model a binary outcome variable nor will it focus on best practices for interpreting those estimands. Rather, this book is meant to address the process that comes before that – how to clean (which is one step of data preprocessing) and manage one's data.

This is an often neglected part of the process. When we enter graduate school, often we start by learning about the types of data we collect and link that to the implications for our options in the types of estimators we have available to us. Learning about best practices for recoding variables, for dealing with missing data, and that it is not a good idea to overwrite one's original `.csv` or `.dta` files with our cleaned data often comes with experience working on projects. While, I understand why this is the case, it is a concerning approach to quantitative research and for training those to engage in it.

As I am someone who has been tasked to bring, often reluctant, students along to learn the statistical programming language `R` as well as statistical concepts all the way through multiple regression in a 16-week timeframe, I am sympathetic to the tendency to just give students datasets that I've already cleaned and pre-processed. These challenges of introducing students to grappling with messy data are even greater when one is tasked with covering these topics as there are high expectations for each student to master the statistical concepts as well as the estimands topics by the end of the term. The steps between collecting the data and analyzing it are often not innovative nor interesting. Therefore, they are often not reported. Given these pressures, learning how to process data tends to be something that many researchers learn by doing. While there is not necessarily anything wrong with this, we often end up using some sort of ad-hoc process that we feel simply "gets the job done."

Though, I understand this position, it seems that we do not really grow out of these tendencies to eventually learn how to pre-process our data in principled, as opposed to ad-hoc, ways. There are so many examples of replication materials that do not include any documentation (e.g., code) about how the researchers took their raw data and put it in an analyzable format. Their data goes from some messy spreadsheet and viola, it is now a super neat and tidy spreadsheet with variables constructed from the raw variables without any documentation about how that was done.

Where it is becoming harder and harder to find a published paper that has results that we can replicate, the lack of documentation for how the data came to be is disheartening. While many express shock at the fact that we only just now are discovering that university presidents and high-profile scholars who have had 30-year careers have been fabricating data (in a cynical view) or at least are making serious mistakes (in a more optimistic view) when processing their data, I am not surprised at all.

Concurrently, the standards for publishing a quantitative paper are increasing. We are expected to publish more papers that are of better quality. These requirements often force us to move quicker but somehow more accurately. One side effect of demands for quality is that we are often encountering datasets with more and more rows *and* columns in them. This produces more opportunities to make mistakes (to be less accurate) as well as to take longer to pre-process and manage the data since there is so much of it (less efficient).

There are three primary goals of this book. The first goal of this book is to convince others who handle, analyze, draw conclusions from, and make recommendations about policy and political outcomes to take a less ad-hoc and a more principled approach to managing and pre-processing their data. The second goal of the book is to offer recommendations about the ways that we can implement a more principled approach to data pre-processing and management. And the final goal of the book is to act as a useful desk reference for undergraduate students, graduate students, and researchers to the various options we have out there to document our data pre-processing and management – in terms of programming languages and particular libraries within it.

Addressing these three goals should help one be more efficient and more accurate in their data pre-processing and management. If one pre-processes and manages their data with code as opposed to ad-hoc, clicking around and editing a spreadsheet, this should provide some degree of readable documentation about how the analyzed data came to be. As I will discuss in the next chapter, the efficiency comes from choices about the programming languages and the particular libraries that one uses within that language that provide the functions to make this management or pre-processing easier.

1.1 Plan of the book

The following chapter is designed to provide an introduction to the different programming languages and libraries within those languages that are available to us for data pre-processing and management. In the chapter, I will provide directions to get set up with each programming languages, will offer a brief discussion of what libraries and functions are in these programming languages, will discuss some options has to choose from for managing these libraries and functions for each programming language, will discuss some of the most popular libraries used for data management and pre-processing in each of these languages, and will discuss the concept of efficient, readable, and replicable code. Throughout the chapter, I will compare and contrast how effective each coding language and their common data management and pre-processing libraries are for efficient, readable, and replicable code.

The third chapter starts our exploration of the common ways that we can think about managing our data. That is, "how do we keep track of our raw and cleaned data?" It will discuss common bare minimum requirements for data security for human subjects set by Institutional Review Boards in the United States, how a principled approach to data management can help increase one's data security practices, and will make an argument about a workflow that one should consider implementing when working with data throughout a given research project.

As the third chapter outlines a ETL (Extraction, Transformation, Loading) process, the fourth chapter focuses on the Extraction and Loading steps to save the Transformation processes for the fifth, sixth, and seventh chapters. The fourth chapter continues the discussion about principled data management. In doing so, the chapter puts many parts of chapter three in practice. That is, we see how we can use code to achieve the lofty goals we set in chapter 3. In doing so, it provides coding examples for the data management tasks for each of the programming languages and the common libraries within each of those programming languages. The chapter will also explore the differences in performance for the extraction and loading steps for each of the coding languages and their popular libraries.

Once we are familiar with the theoretical best-practices for data management as well as how to actually implement them, the fifth chapter will act as a reference for performing common data pre-processing tasks. This chapter in particular will focus on a type of data pre-processing often referred to as data cleaning or data munging. In doing so, the chapter will discuss common tasks that we have to perform, provide coding examples of how to do so, and will compare the accuracy and efficiency of each programming language and their libraries for these tasks.

The sixth chapter will focus on a more advanced set of data munging steps – variable transformations, standardization and normalization, simple scale creation (e.g., creating an additive scale). Like the fifth chapter, the sixth chapter will provide coding examples of how to perform these tasks and will compare the accuracy and efficiency of each programming language and their common libraries for these tasks.

The seventh chapter will focus on a special but important type of data that we often need to pre-process: missing data. Each programming language has a different way of internally documenting missing values for a variable. The chapter will discuss these common pitfalls, how to detect whether there is a missing value, and will discuss the need to engage in exploratory data analysis to determine the underlying pattern that creates the missing data and will introduce some common approaches to addressing them. In doing so, the chapter will provide code examples for the common libraries in each programming language. While doing so, it will also discuss the accuracy and the efficiency of these libraries and programming languages for these tasks.

2 Common programming tools, their benefits and drawbacks

Those performing quantitative analyses have an evolving and extensive list of tools available to them. Most of the time we consistently use one tool, however. There are many reasons for this. But one that may be familiar to many reading this book is that we are either told to learn `R` for a particular class. For others in academia, we may have started to use a particular tool because it became clearer and clearer that the types of statistical procedures you want to use have better support in one language versus another.

The reasons for us using one tool consistently makes total sense. It takes a lot of time and effort to use other tools. In some cases, it is even impossible if we have an instructor tell us we cannot use `Python` because they are teaching you `R` or if you have coauthors who only use `STATA` and you want to use `R`. However, as the book argues, we need to choose the tool that is right for the job. While many of the tools that I will discuss in this chapter and throughout the rest of the book are capable of performing data management and pre-processing tasks, not all of them are great or even good tools for it as later chapters in the book will make clear. This often means that we should use multiple tools in a single project – there is no requirement that we use a single tool.

Keeping in mind the original purpose of the tool can help identify the features that the tool. As I introduce each of the tools that will be referenced throughout the rest of the book, I will make note of which tools were originally designed for which purpose. Hopefully, this can help give you a sense of which tools might be best for data management and pre-processing. Before providing introductions and providing a reference to set you up with each of the tools, I will first discuss some simple concepts underlying the way these tools work. This is not meant to provide you with a comprehensive understanding of these concepts nor for the tools. Rather, it is meant to ensure that you have enough understanding that you can follow along with the code examples provided throughout the book with at least one of the languages if you wish.

To reiterate the purpose of the book, while I do provide a number of coding examples for multiple tools that I am about to introduce, the goal is to not necessarily to teach you the tools but to advance the argument that we can use any of these tools for the management or pre-processing of our academic projects, but that there are some tools that can handle some tasks better than others and that we should be using the tools most appropriate for the job. This means that the book offers many coding examples of how these tools work for different tasks so that we may directly compare and contrast their performance. These coding examples also offer the opportunity for the book to act as a reference material for those who would like to learn more about some of these tools, implement them in their work, and to see how to do particular data management and pre-processing tasks with those tools.

I am a realist though. Many of us are balancing our limited attention, time, and energy to do quantitative analysis. We often have deadlines and have to go with what is "good-enough." The spirit of the book is not judge or chastise people for the tools that anyone chooses to use, but to encourage everyone to truly choose their tool rather than to use what they thought they had to use. The hope is that the book will give you the information needed to make the choice between whatever tradeoffs you face with using these tools an easy choice. If you choose to change things up, even if it is reluctantly, then the hope is to provide you something to reference to make that change easier!

2.1 Measuring tool performance

At the outset of the book, I kept mentioning the "efficiency, readability, and replicability" as being important features or a programming language. Here, I want to elaborate upon what I mean by these terms.

The first feature is efficiency. When we think of efficiency, we may think of how much effort is being put into a task relative to how much is accomplished by completing the task. That is what we mean here too. While there are some really technical rabbit holes one may fall into among computer scientists when defining the efficiency of one's code and various metrics that we can calculate to quantify efficiency, when I refer to efficiency, all I am referring to is, "how much work is your computer putting in relative to the task?" For example, if it takes 20 seconds for a program we wrote on our computer to evaluate the expression `2+2`, then we would say that is pretty inefficient relative to me just doing it in my head. In that case, we would say that our program is pretty inefficient.

When thinking about efficiency, there are more ways to measure than the time elapsed to calculate it like in my example of evaluating the expression `2+2` – though this is perhaps one of the most common ways efficiency is measured. We can also think of it as how much memory our computer must commit to using. When purchasing a computer, you may hear someone discuss the `RAM` of the computer. While `RAM` is one type of memory, it is distinct from the Hard Drive on your computer that stores your files and software on it. One way that we can think of the `RAM` is the stuff that you have asked your computer to keep track of at any given point in time. `RAM` is active memory. Let me illustrate this through an analogy. You have probably memorized your phone number. Now say that I ask you to enter your phone number while I am also asking you to keep track of my phone number as I read it out loud to you. This is probably a lot to keep track of. You are not only being asked to keep track of 18 numbers all at once, but also the order in which those numbers appear and to keep track of which one is yours and which one is mine. This task might be hard. If you don't believe me, try it with a friend. The `RAM` simply refers to the amount of stuff you are asking your computer to keep track of and readily accessible within a split second. It is the calendar events, it is the web pages you have open on your browser, it is the contents of the song that you are playing, etc. The files that are currently closed on your computer are stored on your hard drive and you are not necessarily asking your computer to have that information readily accessible in a split second, not until you open that file, at least. While this is an oversimplification, `RAM` is another common metric that people think of when thinking about the efficiency of a program. If the program has **tons** of information stored in it, it will significantly slow down your computer as it is struggling to keep up with all of the things you are asking it to do – just like it would slow you down to enter your phone number while I'm reading out my own to you at the same time.

We want the evaluation of our code for our data management and pre-processing to be both quick and not too laborious for our computer to do. If we use code that is inefficient, it will take longer for us to be able to complete these tasks. Additionally, if we have a computer with more `RAM` and it works on our computer, it may not work on someone elses that has less `RAM`, or at least it will take much longer for them than it would for us. This is an issue I will come back to in a few paragraphs.

A second important feature of a programming language and of our own code is the readability of it. In an ideal world, our code should be understandable even to those that are not familiar with the coding language we are using. One of the primary reasons in a research setting will be discussed shortly, but otherwise it makes it easier for us to detect errors in our code! If our code takes a lot of effort to understand what it is doing, then it will undoubtedly make it more difficult for us to catch a mistake that we have made. We want our code easy to understand so that we can catch our mistakes as well as others'. Now, we often do not have the luxury of jumping into any coding language and immediately understand what that code is doing, so it is often far-fetched to say that it should be readable to those who have no experience with the language, but we should still make it reasonably easy to get a sense of what is going on for everyone and super clear what is happening to those that are familiar with the language.

Reproducibility is the last key feature that we should consider about code. I have hinted to it a few times. Reproducibility is a challenging but important endeavor for every research project that we work on. Reproducibility refers to the idea that anyone can take our documentation, repeat what we did, and be able to get the same results (or at least come to similar conclusions). How can we achieve reproducibility? Well, this is a topic that is still debated. But in general, we want the steps we take when managing and pre-processing our data to be something that someone else can follow relatively easily. Meaning that the documentation we have is accurate, our code is efficient so almost anyone can do it regardless of their particular computer's hardware, and that it is readable so that as many people as possible can understand the steps we took during the project.

Later in this chapter, I will be evaluating the programming languages and common libraries used for data management and pre-processing on these three key features. It is important to note that these languages and libraries are not deterministically good or bad at achieving these goals. Often the person writing the code is at fault for writing inefficient, unreadable, and unreproducible code. However, there is no one "right" way to do anything. However, there are some programming languages and libraries that do not do much to encourage efficient, readable, and reproducible code. So, my evaluation of these languages and libraries on these features will be limited to their capacity to encourage the programmer to produce code that contains these features.

2.2 Concepts that transverse (most) tools

Before, I move on, it is important to provide a brief explanation of how many of these tools work. Most of these languages are referred to as object-oriented programming languages (`OOP`). What this means is that we can store the results of some task in something called an `object`. So, say for example that I want to evaluate the expression, `2+2`. Once I have successfully determined that the result of the expression is `4`, I may want to store that result for later. In an `OOP`, I can store the result, `4`, in an `object`.

Now, say that I won't always be adding `2` and `2`, but I may instead be adding `2` and `3` or `10` and `2`. Instead, I want to quickly be able to evaluate a summative expression and be able to just plug in those numbers when I need to rather than have to keep rewriting `x+y`. This is a perfect candidate for a `function`. A function is just a piece of code that takes a pre-specified set of parameters, such as what integer `x` and `y` represent, and perform some task using those parameters. I can store the function to evaluate the summative expression of `x+y` and just pass in the integers that represent `x` and `y` as arguments, such as `x = 2` and `y = 2`. `OOP` allows me to store this function as an object just like I would the result of `2+2`. If you are still not quite sure what this looks like in practice, don't worry, you'll see later in the chapter!

So, a function is some chunk of code that we can name and store as an object so that we do not have to reinvent the wheel over and over. While in the previous example of a function that does `x+y`, this may seem relatively not hard to do, but sometimes we have tasks that are extremely complex and take a lot of code to complete. So, functions represent stored code that take our arguments passed to the function's parameters as an input, does something to that input, and then returns some output. This is illustrated in Figure 2.1 below.

Since we can save a function as an object thanks to `OOP`, we can store this object in what is called a library. A library refers to a collection of functions which all contain code designed to complete specific tasks. It essentially is a way to "package" multiple related functions together so that they can easily be shared and used by myself and others.

For example, I can create an object storing the function depicted in Figure 2.1 to evaluate a summative expression as `addition`. I can create another object that stores the function to evaluate a differencing expression as `subtraction`. I can put these two objects in a library, as depicted in Figure 2.2, and post them to a specialized website for others to download the library so that they can easily download those

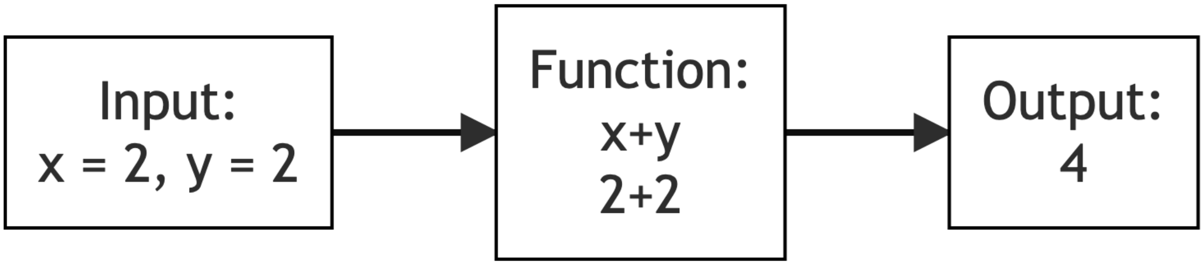

Figure 2.1: Visualizing a function

functions and easily use those functions without having to write their own functions or code.

In essence, libraries are just large files of code that someone else has written that let you use a particular function defined in that code so that you only need to provide some inputs (or arguments is what they are often called), the code does something to those inputs, and then it produces some output.

How is all of this relevant to data management and pre-processing? As I mentioned, most of the languages that we are using here are `OOP`. Most of the time, we do not have to write our own code from scratch to manage or pre-process our data. We will have to write some code, but we are often interacting with functions that someone else has written and put together in a library for us to make our code much shorter. As these functions are written by others and we often are heavily dependent on functions from a library that someone else has written, we should not and cannot blindly rely on functions or libraries. We should not always assume that they are efficient, readable, nor enable reproducible code. Just because a programming language enables these features in our code, our experience with this is heavily dependent on the particular libraries that we choose to use within a particular programming language. That is why, in the next section and the book at large, I will be putting a lot of emphasis on not just the languages but also the common packages people use within each of these programming languages to pre-process and manage their data.

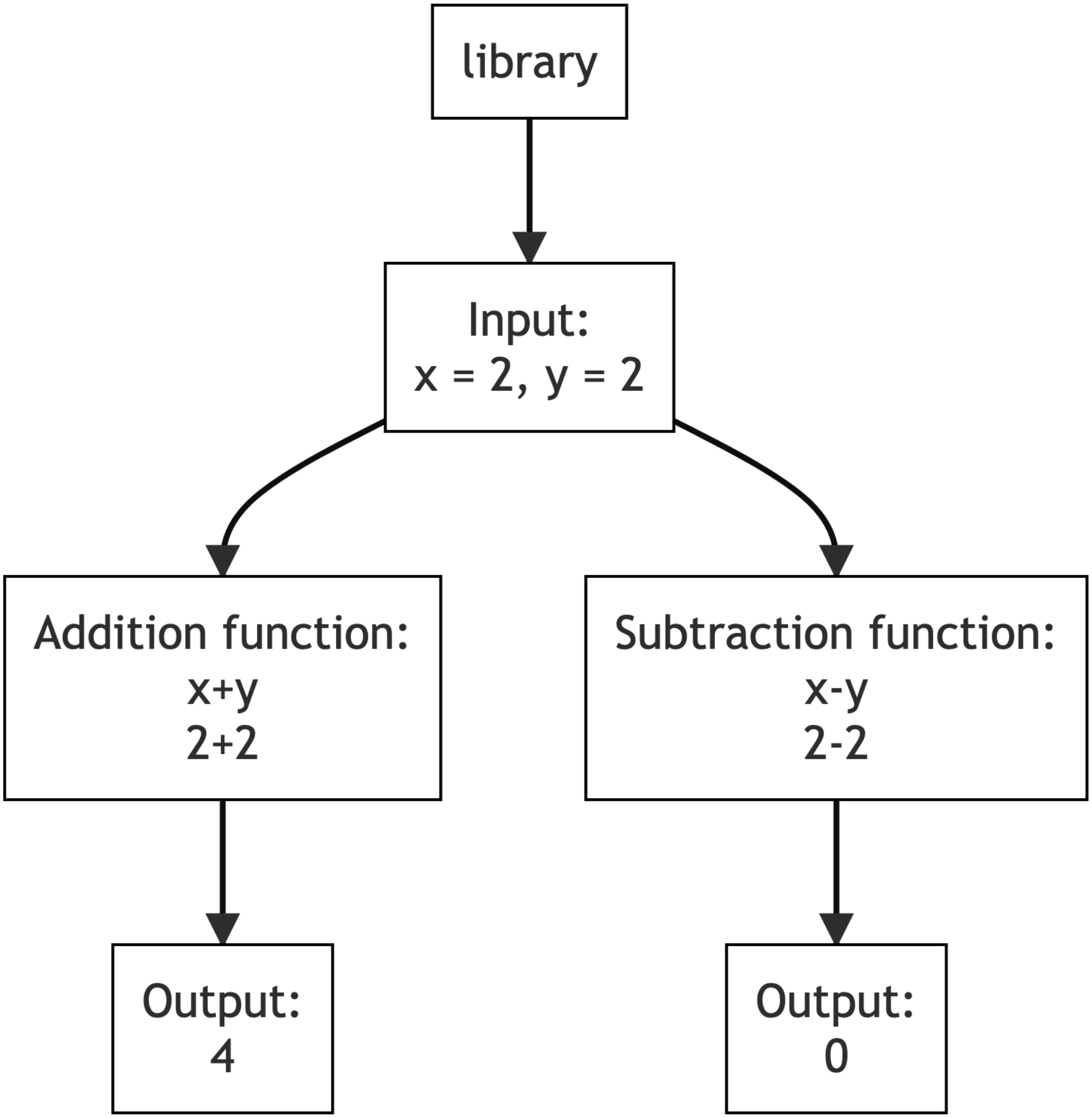

Figure 2.2: Visualizing a library of functions

2.3 Programming language options, setup, and comparisons

To re-iterate, there is no one absolute better programming language for someone to use to pre-process and manage their data. However, this book will focus on emphasizing and advocating for programming languages that are efficient, readable, and reproducible. The options that I include here are not comprehensive. One primary reason for a programming language to be excluded is due to it requiring a license to use. One popular software in the social sciences is `STATA`. While the programming language is not proprietary, you can only use `STATA` as a programming language in their proprietary software, and is therefore not reproducible. As a result, `STATA` is not discussed as an option in this book, though it is popular.

Before, jumping into the installation and beginning to interact with the particular programming languages, I want to discuss the difference between installing the software for your computer to understand the language and the installation of the Interactive Developer's Environment (`IDE`) that provides nifty features to not just write the files but to interact with the code such as running the code in parts rather than as the whole file, debugging tools, and other features.

These coding languages do not come standard on our computers. We have to install software that links the code already on our computer that it understands (often C++). This is the first thing we will need to download. Often this software will come with extremely basic `IDEs` that we can use to write code, save files that store the code, and run code, as either complete files or as smaller pieces of code such as a single line (interactively). These `IDEs` are often extremely basic and include really limited functionality. So the second think we often download is a more fully-fledged `IDE`.[1] Some programs have their own popular `IDEs` such as `R` with `RStudio`. However, in the spirit of not being beholden to a single coding language for all tasks, I'll provide instructions on how to install a really popular do-it-all `IDE` called Visual Studio Code (`VSCode`). `VSCode` has extensions that enable you to be able to fully interact with all of these coding languages discussed below.

[1] Also, for those interested, you can forego `IDEs` altogether and install a text editor such as `neovim`. While these are extremely customizable, they require quite a lot of comfort and sophistication with command line interfaces and require a lot of work to get them to a point where you can be semi-productive with them.

2.3.1 Installing an IDE, `VSCode`

Again, if you want to install RStudio or some other language specific `IDE`, that is totally up to you! Here, in this book I will cover how to install an `IDE` that works for all of the languages in this book.

First, go to https://code.visualstudio.com/download. Then you will want to choose an installation that is appropriate for your operating system. Unless you are wanting to do some custom installation, you should choose the button just under the icon for the type of operating system that you use. You can then follow the standard installation steps you take to install software on your computer. You should not need to change any of the defaults to install `VSCode` on your system.

Once successfully installed, then we can move onto installing the particular coding languages.

2.3.2 `R`

`R` is an extremely popular coding language for statistical analysis among academics, analysts, and for a decent proportion of data scientists. There are a number of reasons for its appeal but chief among them is the robust community to help with the reporting of statistical analyses. Unlike some of the other option discussed below, `R` has a large number of options for packages that help with generating tables and figures for the results of complicated statistical analyses. This is not to say that the other programming languages do not have any support for data visualization and reporting, but rather that there are more options for popular types of data visualization and reporting in the scientific community. This makes it really popular among academics.

More recent efforts have also made `R` popular for data management and pre-processing. As I will discuss later in Chapter 4, the `Tidyverse` is a suite of `R` libraries that include functions to manage data from a variety of different formats such as `STATA`, `JSON`, Excel, TSV, and CSV to name a few. It also includes an extremely popular library called `dplyr` which many use to help with data munging. These efforts are a relatively recent boost to `R`'s capability as a tool for managing and pre-processing data. This is not the only effort, however due to some limitations present in the `Tidyverse` ecosystem as well as some inherent challenges that come with `R`.

2.3.2.1 Installation

Installation of R is quite straightforward. The first thing that we will want to do is navigate to the `CRAN` website at https://cran.r-project.org. This website is the "Comprehensive R Archive Network" and is the default place that we not only install `R`, but we can often find and install many of the libraries (often referred to as packages by `R` users).

Once on the webpage, we will want to follow the link to "Download R for" the operating system that we are using. So if you are using a Mac, then you will want to follow the link to the "Download R for macOS". Once clicking on the link, it will take you to a page listing different releases of R. You will want to follow the most recent one. At the time of writing this book, that would be `R version 4.3.1`.

> ℹ For macOS users
>
> If you are installing `R` on a Mac, you cannot install any package of `R`. If you are using one of the newer Apple Silicon chips, then you should install the `R` package made for those using Apple silicon chips and not for Intel. For example, if you are trying to install `R` version 4.3.1, then you should follow the link `R-4.3.1-arm64.pkg` If you are using a Mac that has an Intel chip, then you should choose `R-4.3.1-x86_64.pkg`.

Once you have clicked on the link, you should follow the normal steps you take to install software on your computer. You should not need to make any customizations to the installation (such as installation destination).

2.3.2.2 Setting up `VSCode` with `R`

If you want to use `R` with `VSCode`, getting them to work together is pretty straight forward! You will want to install an extension that enables `VSCode` to recognize `R` code and to know what to do with it when you run your `R` code. While you can install extensions directly in `VSCode`, we will go to https://marketplace.visualstudio.com/VSCode. Once there, we can search "R". What should be the first option is an extension to work with `R` in `VSCode`. We can install that extension by clicking the "Install button".

Once installed, you may not be quite ready to run `R` in `VSCode` just yet. You should follow the steps in the "Getting Started" section of the page that you downloaded the extension from and view the extension's documentation for any extra requirements they may have for you to start running your R code in VSCode.

2.3.3 Illustration of OOP in R

Going back to the discussion above about OOP, functions, and libraries in R. Here is some simple code to help give you an understanding of how R works. To follow along, you can open VSCocde and then create a new file called my_first_r_file.R. Since you are specifying that the file is an R file with the extension .R, VSCode will now expect to only see R code in that file.

First, let me make a comment about comments in R! You should always add comments to your code. This helps with the readability and reproducibility of your code. Comments are incredibly useful in that they allow you to explain what your code is doing and can act as really useful documentation.

To make a comment in R, you can simply put the hash or pound symbol at the start of any line in your R file. Then you can write freely. There are no requirements for what words you use or how you structure your sentences for lines when you've put a comment in front of them. For example:

```r
# This is a comment in R
# Any characters that you place after a "#"
# Will not be evaluated as code.
```

Now, let's evaluate my simple expression from above with R:

```r
2+2 # add 2 and 2 together
```

```
[1] 4
```

Once we have written this code, we can highlight the code and click the play button (sideways triangle) at the top. This will run the code. Another option is to use the keyboard shortcut "CMD/CTRL + RETURN/ENTER". This will open our R console at the bottom of our screen. The console is responsible for taking the code passed from our R file and actually doing the evaluation of it.

We should see that the output is 4. Great, that is exactly what we should expect.

Now, let's say that we want to take the result from this expression and use it later such as by doing 2+2+2. Rather than typing out 2+2+2, we can take advantage of the fact that R is an OOP language and store the result of our original expression into an object that I will call "result". The <- is called an assignment operator in R and it is used specifically for assigning the result of some expression to an object. Other programming languages that we encounter will use often just use = as the assignment operator.

Once I have created the "result" object, I will then take the object that I called "result" and will add two to it, then store that result into an object called "result_two". I can then see the value stored in "result_two" by using a function called print. We should expect that the result is 6 if everything worked correctly.

```
result <- 2+2 # add 2 and 2 together, store it into an object called result
result_two <- result + 2 # add 2 to result, store it into an object called result_
print(result_two) # show me the value of result_two
```

```
[1] 6
```

I'm not limited to adding a bunch of 2's together in R. What if, I want to do what I did above but with other numbers and don't want to continuously repeat myself. Well I can write a function and store that function to an object so that I can just reuse the same code over and over again.

So, I am going to write a function that I am going to call "addition." This function will take two inputs that I am going to call x and y. These inputs can be any two integers that I want. Once I have provided the basic information about my function to R, I'll specify what I want the function to do with those inputs in the main block within the curly brackets. As you can see, I want to add x and y together and store it in a local object called temp_result. I will then return the the result.

```
# define a function called addition
  #* take the parameters x and y
  #* the arguments for x and y should be an integer
addition <- function(x = 1, y = 1) {
  # take the parameters x and y
  # add them together
  temp_result <- x + y
  # return the temp_result object
  # and forget the value for the object once returned
  return(temp_result)
}
```

So once I have defined this function, I can start using it!

```
# use the addition function
# to add 2 and 2 together
# store the result of the function in an object called result
result <- addition(x = 2, y = 2)
# print the value of result
print(result)
# use the addition function
# this time to add 10 and 10 together
# store the result of the function in an object called result
result <- addition(x = 10, y = 10)
# print the value of result
print(result)
```

There are plenty of subtleties in the code here that I won't go over as they are beyond the scope of the book. There are plenty of excellent discussions out there about `OOP` in `R` if you want to delve more into getting comfortable with writing functions in `R`. But, the main idea is that you can see that I can take the `addition` object as a function to reuse the code within the curly brackets to apply it to different inputs.

As discussed before, there are plenty of functions that we might write that could be useful to share with others. To do this, we can take these functions and put them in a `library` and upload it to `CRAN`. Details about how to do this are also outside of the scope of this book, but thankfully there are plenty of great resources on how to do this too!

`R` comes with a `base-r` library of functions that cover bare minimum needs for the users to work with the language. For example, we have already used multiple functions in the code above. For example, `print` took an input from us – our object `result`, did something with it, and then printed out the value of our object. While the way that the `print` function knew to print out the value of `4` rather than to just print out "result" in our console is a whole other discussion about a concept called pointers that are also beyond the scope of the book, the `print` function had some underlying code that knew that `result` was an object and we wanted to print out the value that `result` represented.

So, how can I access a library that someone else has written? It isn't too bad! There is a function that comes standard with `R`, in the `base-r` library, that allows us to access libraries uploaded to `CRAN`: `install.packages()`. With `install.packages()` all that we have to do is specify the name of the library we want to install, then when we can use those functions after loading them from the library that we've downloaded. [2] Do not worry, there will be plenty of examples in the following chapters of this.

> How do I know the name of a library or whether a function exists?
>
> There is no one way that this happens. I often stumble across useful libraries and functions within those libraries in different places of the internet, from talking with others, etc. As you go along, you'll begin to get more comfortable finding out these things yourself.

2.3.4 `Python`

`Python` is perhaps one of the most popular coding languages in the world. Unlike `R`, `Python` is used for more than just quantitative analysis. Beyond scientific reporting, it is used heavily in machine learning, artificial intelligence, and is also used relatively frequently in web application design and software engineering.

It is less popular than `R` among academics in the social sciences primarily due to the fewer and less robust packages for reporting scientific quantitative analyses. This is beginning to change, however. As there has been an increased growth of the computational social science community which moves past regression-based quantitative analyses to examine text and images with machine learning approaches, the rich support for these tasks in `Python` have lead to more social scientists to use `Python`. This

[2] Functions from the `base-r` library automatically loads at the start of each `R` session. Functions from the libraries that do not come standard, the ones we had to install, require that we load the library at the start of an `R` session. Requiring that we load libraries at the start of each session ensures that we aren't loading a lot of junk libraries that we aren't using for our particular project but may have used for a different project. This keeps our `RAM` clearer and keeps our code more efficient.

has also lead to more social scientists taking it upon themselves to work on libraries that are designed to help with the reporting and visualization of scientific results.

`Python` is certainly growing in popularity outside academia but also within academia. It is a language that can certainly be helpful for those who may want to pursue careers outside of academia after completing their education or who would like to use things like machine learning in their research.

Another attractive feature of `Python` is that it often has fewer concerns with efficiency, readability, and reproducibility than `R`, overall. `Python` was originally conceived as a language designed to make software engineering more accessible and easier to perform. Since the bulk of its users are software engineers, data engineers, or data scientists rather than academics, many of the libraries are designed with efficiency, readability, and reproducibility as central tenants of the libraries' designs. This is one reason why many places outside of academia often prefer a team using `Python` for their data analysis even though that `R` is specifically designed for statistical analyses.

To summarize, `Python` can be useful for increasing the efficiency, readability, and reproducibility of your code relative to `R`. It is a language with libraries that let you do more than just quantitative analysis which can be appealing to not learn a tool to complete only a super narrow set of tasks. It is one of the most preferred languages for those doing computational social science due to robust packages for machine learning and artificial intelligence. It is also a language that is in high-demand for many sectors and job titles outside of academia. However, it is less than ideal for scientific statistical computing and has fewer libraries than `R` for such tasks.

2.3.4.1 Installation

The installation of `Python` is relatively straightforward. You will want to follow this link https://www.python.org/downloads/.

Once on the webpage to download the latest version of `Python` (at the time of writing this was `Python 3.11.4`). You can just click the yellow button to download it. Python should automatically detect the operating system that you are using and will download the proper installation file. If not, there is a link just below the yellow button that lets you choose which operating system you are using.

Once you have downloaded the installation file, you should do a default install and follow the normal installation steps you take for an application. You should not need to make any customizations to the installation (such as installation destination).

2.3.4.2 Setting up VSCode with Python

If you want to use Python with VSCode, getting them to work together is probably even easier than it is with R. You can go to https://marketplace.visualstudio.com/VSCode. Once there, you can search “Python”. The first option should be an extension called “Python” written by Microsoft. We can install that extension by clicking the “Install” button.

Once installed, you may not be quite ready to run Python in VSCode just yet. You should follow the steps in the “Getting Started” section of the page that you downloaded the extension from and view the extension’s documentation for any extra requirements they may have for you to start running your Python code in VSCode.

2.3.4.3 Illustration of OOP in Python

Like R, Python is also a OOP language. Here is some simple code to give you an understanding of how Python works. To follow along, you can open VSCode and then create a new file called my_first_python_file.py. Since you are specifying that the file is a Python file with the extension .py, VSCode will now expect to only see Python code in that file.

Comments are the same in Python as they are in R. In any language, you should always add comments to your code to increase the readability and reproducibility of it.

The symbol for a comment in Python is the same as in R. You can simply put the hash or pound symbol at the start of any line in your Python file. Then you can write freely. For example:

```
# This is a comment in Python
# Any characters that you place after a "#"
# Will not be evaluated as code
```

Now lets go back to our evaluation of the expression of 2+2 as we did in R:

```
2+2 # add 2 and 2 together
```

Here you'll notice the code looks exactly the same as it did in R. Once we are doing more complicated tasks, the languages will certainly begin to look a lot different.

Like in R when we want to evaluate this Python code, we can highlight the code and click the play button (sideways triangle) at the top of our VSCode window. This will run the highlighted code. Like it did with our R code, we should see a console open at the bottom of our VSCode window. Instead of it being an R console, it will now be a Python window even though the code looked exactly the same between the two. VSCode knew that you were running Python code because the code you were running should have been in a .py file instead of a .R file.

Like we did before, say we want to store the result of our expression 2+2 into an object that we can use later. Since Python is also a OOP language, we can do this. Unlike R, the assignment operator in Python is = instead of <-. So we will take our expression and will assign the result to an object called "result".

Once I have created the "result" object, I can use that object to add 2 to it and store that result in a object called "result_two" like I did in R. To see the value of the "result_two" object I can use a print() function.

```python
# add 2 and 2 together, store it in an object called result
result = 2 + 2
# add 2 to result, store it in an object called result_two
result_two = result + 2
# show me the value of result_two
print(result_two)
```

```
6
```

Like I did with R, let's say that I want to do addition a few times but I do not want to have to keep writing a + b over and over again. So I decide to write a function so I can reuse that code over and over.

I'll write a function that I am going to call "addition." This function will take two inputs that I will call a and b. Again, these inputs can take any integers that I want. Once I have provided the basic information about my function to Python, I'll specify what I want Python to do with those inputs in the main block of the function after the colon and when indented by 4 spaces.

```python
# define a function called addition
    #* take the parameters x and y
    #* take the arguments for x and y, should be an integer
def addition(x = 1, y = 1):
    # take the parameters x and y
    # add them together
    temp_result = x + y
    # return the temp_result object
    # and forget the value for the object once returned
    return temp_result
```

Notice the difference

Unlike `R`, the main block of code for my function in `Python` is not within curly brackets. Instead, it is indented code. `Python` is extremely sensitive to the indentation in a way that `R` is much more flexible. In `Python` you should use 4 spaces for an indentation – do not use the `tab` key as this can sometimes be an inconsistent number of spaces and may lead `Python` to throw you an error about improper indentation.
How `Python` knows that the code for the function for the function is completed is once it sees a line that does not indent by four spaces. Once it notices that, it will go to the last consecutive line with four spaces and will consider the function's main block of code to end on that line.

Now that I have defined the function, I can use it!

```python
# use the addition function
# to add 2 and 2 together
# store the result of the function in an object called result
result = addition(x = 2, y = 2)
# print the value of the result
print(result)
# use the addition function
# this time to add 10 and 10 together
# store the result of the function in an object called result
result = addition(x = 10, y = 10)
# print the value of result
print(result)
```

As Python is also an OOP language, I am capable of taking the code within the addition function that is stored as an object called "addition" and am able to reapply that code within the function to different inputs.

Like in R, I am able to use various functions that come with the standard installation of Python. I will want to install libraries containing nifty functions as my code becomes more complex, though. In the next chapter, I will provide concrete examples of how to install and work with some common libraries for data management.

2.3.5 Julia

Like R, Julia is specifically designed for scientific statistical computing. It is a much newer language than R, however. Julia has started to gain some traction in some areas of machine learning engineering, some circles of data science and in academia. It still remains relatively uncommon, however. The main reason that it remains relatively uncommon is how new it is. It is also much more dedicated to statistical computing than the other languages which makes it relatively narrow in the tasks that it can complete compared to Python and even R.

The appeal of Julia is that it is an extremely efficient and reproducible language. The language is not as bogged down as languages like R and Python which reduces a lot of problems. It also has built in support for virtual environments so that one does not have to jump through as many hoops to install, manage, and document the libraries that someone uses for their project.

Julia also has some really interesting features such as allowing you to use scientific and greek symbols as object names. Python and R does not allow you to do this. While this might be a small feature for some, others it may be super useful. So there is that.

In a time where people are wanting to publish papers faster, are using more data, and are dealing with concerns about how reproducible their results are, Julia certainly seems like a language that could eventually become extremely popular in academia. It is already gaining some success in many sectors of industry as a language that reduces the amount of time that analysts are spending to produce reports.

2.3.5.1 Installation

Thankfully, the installation of `Julia` is also quite straightforward. You will want to follow this link: https://julialang.org/downloads/#download_julia.

Once on the webpage to download the latest version of `Julia` (at the time of writing this was ‘Julia 1.9.2). If you are on Windows, you should follow the link to install the Windows 64-bit installer. If you are on a Mac with a non-intel processor, you should follow the link to install the macOS (Apple Silicon) 64-bit (.dmg). If you are on a Mac with a intel processor, then you should follow the link to install the macOS x86 (Intel or Rosetta) 64-bit (.dmg). If you are on linux, you should follow the most appropriate option for your setup.

Once you have downloaded the installation file, you should do a default install and follow the normal installation steps you take for an application. You should not need to make any customizations to the installation (such as installation destination).

2.3.5.2 Setting up VSCode with `Julia`

Like the other languages, working with `Julia` in `VSCode` is not too hard to setup. You can go to https://marketplace.visualstudio.com/VSCode. Once there, you can search “Julia”. The first option should be an extension called “Julia”. You can install the extension by clicking the “Install button”.

Once installed, you may not be quite ready to run `Julia` in `VSCode` just yet. You should follow the steps in the “Getting Started” section of the page that you downloaded the extension form and view the extension’s documentation for any extra requirements they may have for you to start running your `Julia` code in `VSCode`.

2.3.5.3 Illustration of `OOP` in `Julia`

Like the other languages so far, `Julia` is also an `OOP` language. To follow along with my examples below, you can open `VSCode` and then create a new file called `my_first_julia_file.jl`. You are specifying the file as being a `Julia` file with the extension `.jl`.

Like the other languages, you can add a comment to your file using the hash or pound symbol. Any characters that follow will not be evaluated by `Julia`. For example

```
# This is a comment in Julia
# Any characters that you place after a "#"
# Will not be evaluated as code
```

Now let's evaluate the expression `2+2` in `Julia`:

```
2+2 # add 2 and 2 together
```

Like in the other languages, the code looks exactly the same. But once we start to do more complicated tasks, we will start to notice the differences in the syntax of the languages.

Like in the other languages, if we want to evaluate this `Julia` code, we can highlight the code and click the play button (sideways triangle) at the top of our `VSCode` window. This will run the highlighted code.

Unlike the other coding languages, instead of opening a console specific to that language, `Julia` will open an instance of the `Julia REPL`. It is not necessary to understand the differences between the two, but how you interact with them will look slightly differently. The reason for these differences is that the goal of `Julia` is for the code to be interactively run as opposed to run as a whole file like is the common expectation with `Python` and to a lesser extent `R`. If you get an error from `VSCode` telling you that the `REPL` is in a different directory than the file, then you may want to go to the top of your `VSCode` window, click on "File", then click on "Open Folder", then choose the folder that you stored your `my_first_julia_file.jl` in. This then should solve the issue and the code should successfully run.

Like we did with the other languages, say that we want to store the result of this into an object. We can do that. Like `Python`, the assignment operator in `Julia` is `=`. I will evaluate the expression `2+2` but store it in an object called "result."

Once I have created the result object, I can use that object to add 2 to it and store that result in an object called "result_two" like I did in the other examples. Unlike the other coding languages, `Julia` should automatically print the value stored in my "result_two" object. If I wanted, I could also use a native `print()` function to print the value for "result_two".

```
# add 2 and 2 together, store it in an object called result
result = 2 + 2
# add 2 to result, store it in an object called result_two
result_two = result + 2
# show me the value of result_two
print(result_two)
```

```
4
```

```
6
```

Like I did with the other languages, let's say that I want to change this to a function so that I do not have to keep repeating myself.

I'll write a function that I will call "addition". The function will take the inputs `x` and `y`, which will be two integers I want to add together.[3] Once I have provided the basic information about the function, I'll specify what I want `Julia` to do with those inputs. Like in `Python`, I will use indentation to indicate what code should be included with the function. This time I do not put a colon at the end of the basic information about the function. Unlike `Python`, I will explicitly define the end of the function. Whereas `Python` waits for the first line of code that is not indented to close the function, `Julia` has me type "end" where I want the function to end. Notice that the "end" is not indented.

```
# define a function called addition
    #* take the parameters x and y
    #* take the arguments for x and y, should be an integer
function addition(x,y)
    # take the parameters x and y
    # add them together
    temp_result = x + y
    # return the temp_result object
    # and forget the value for the object once returned
    return temp_result
end
```

[3] Previously, I was able to name the parameters whatever I wanted, but `Julia` is specifically designed for scientific computing and so it is not as flexible.

```
# use the addition function
# to add 2 and 2 together
# store the result of the function in an object called result
result = addition(2,2)
# print the value of the result
print(result)
# use the addition function
# this time to add 10 and 10 together
# store the result of the function in an object called result
result = addition(10,10)
# print the value of result
print(result)
```

`Julia` is a little bit different than the other `OOP` languages we have used so far. It is `OOP`-enough, but not quite `OOP` like the other languages so far. As we saw here, we can have objects own our results, but we can't really make functions the same way we do with the other languages. There is a discussion that we have elaborating on all of these points that are rather niche, just know that we can do *some* `OOP` stuff in `Julia` but not in the same way as languages like `R` and `Python`.

2.3.6 SQL via Duckdb

`SQL` is a language designed exclusively for data management and pre-processing. Rather than loading files to then work within active memory on your `RAM`, it is often used to interact with databases. `SQL` is an extremely common language and is widely used in many sectors. There are two key flavors of SQL: the open-source `PostgreSQL` and the proprietary Microsoft `MySQL`. These two flavors of `SQL` are not different versions of the `SQL` language but are more so dialects.

Traditionally databases are stored as servers that many computers and client-side servers. These devices can access the database server at the same time to both store new data and to be accessed for analysis from data engineers and analysts. There are local options, however.

Academics have not implemented the use of `SQL` databases as a way to manage their data. As I will argue in the next chapter, I think that this is a missed opportunity for a number of reasons. One of the key reasons for academics not adopting SQL databases is that they are often not suited to the types of data and the traffic to add new and access that data that most database servers are designed to handle. This coupled with the fact that most database servers often require quite a lot of work to design, maintain, and to develop a sustainable architecture for – this is often a data engineer's job.

There are a number of local, as opposed to server, implementations of `SQL` databases that we can take advantage of for our research projects, however. To give a preview of the next chapter, while it requires some familiarity with the SQL language, these databases are much more secure due to the added levels of technical competence to access the data. That is, you cannot just click to open the file to view the data – instead, you have to connect to the database (either as the server or as a local file) and then execute a query in SQL in order to view its contents. These databases are also designed to store and allow for users to quickly access massive amounts of data. As we are using larger and larger datasets for our projects, in terms of both variables and observations, this is a really helpful way to future-proof your skills so that you are not limited to software limitations imposed by Excel, CSVs, libraries from the languages we discussed earlier, or `RAM`.

To this last point and to balance the consideration that most researchers do not want to learn a lot about data engineering, one tremendous option that is easy to integrate with the current workflow for many academic researchers while reaping many of the benefits of using databases rather than files like CSVs is `DuckDB`.

`DuckDB` is a tool that allows us to use `PostgreSQL` to store and access data stored in a local database file. In other words, it allows for us to access and store a database on our computer just as we would with a CSV file, but it does not require that we manage a server. `DuckDB` also allows for `lazy evaluation` which refers to the idea that you access your database only when you absolutely need to and that it only accesses the data in the database that you need rather than all of it. The use of a database increases data security and the last feature also allows for us to *very* efficiently handle large amounts of data regardless of the hardware specifications of our computer.

`SQL` is a relatively easy language to learn. Much easier than the other languages used so far. This is because it is highly specialized for accessing data. This means that it does not require one to learn a whole lot of different things. Because of this, it also does not require the use of libraries or functions. It is also extremely readable as a language. The language has preset commands that you can pass to access the data, clean it, and to store it. Often times, it feels like how you would verbally say how you accessed your data, just without filler words.

2.3.6.1 Installation

`DuckDB` has libraries for `R`, `Python`, and `Julia`. These libraries allow for you to use functions that reduce the amount of SQL code you have to write. While this is useful, there are some challenges that I have run into with these libraries that have made it less than ideal. `DuckDB` is a new tool and not only are they trying to improve the tool, but they are also having to manage libraries for a number of other languages. As a result, it can create some pain points.

If you would like to interact with `DuckDB` through the libraries in the other languages I've discussed, go for it! However, I will stick with `DuckDB` as implemented as a standalone tool rather than as implemented through their libraries in the languages we have discussed so far. This means that I do not need to worry about how up-to-date that particular library is relative to the standalone tool.

To install `DuckDB` as this standalone tool, you will want to install it via the command line. To do this, navigate to the web address https://duckdb.org/docs/installation/index. From there, you can choose whether how you want to interact with `DuckDB`. For those that want to go with my route by using the standalone tool, you would choose the latest version tab (at the time of writing the latest release was 0.8.1). Then for the "Environment", you'd choose the "Command Line" tab. For package, be sure to choose the "Binary" tab. Then you would choose whatever operating system that you are interacting with such as macOS if you are on a Mac.

Once you have gotten here, it will give you instructions for installation. For example if you choose macOS, the installation instructions would look like this:

```
Homebrew: brew install duckdb

Github binary: https://github.com/duckdb/duckdb/releases/download/v0.8.1/duckdb_cli-osx-
```

If you see this, then you have two options. If you know what homebrew is, then you should be able to open your `Terminal` app and run `brew install duckdb`. Otherwise, then you can type this address into your web browser. This will download the Zip file. Once it is downloaded, you can open the file. It will automatically create a duckdb executable on your desktop. You can then right click on it, and open it with your Terminal app. If you are on Windows, your only option will be to download the zip file, open that zip file, then right click and open it with your Terminal app.

You should see something like this in your Terminal app.

```
v0.8.1 6536a77232
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
D
```

This means that DuckDB has been successfully installed and is ready to go. You should not need to do this each time you want to work with DuckDB. We will be accessing DuckDb through the VSCode terminal (where we saw the R and Python consoles and Julia's REPL.)

2.3.6.2 An IDE for DuckDB

While there is a VSCode extension for DuckDB, it is not an official extension created by the team at DuckDB and has pretty limited capabilities unless you are willing to pay $10 a month. I am not. So, instead, there is an IDE that you can run within your VSCode terminal at the bottom of the screen (where your R and Python) consoles appear that is officially supported by the DuckDB team.

The IDE is called harlequin. It is super nice because it allows you to copy and paste your SQL code into a Query editor, it shows you your table (dataset in database speak) and any results to you queries that you run.

To install it, be sure to first install Python using the instructions provided earlier in the chapter. Then once you have done that, go to your Terminal application, and type pipx install harlequin. You now have your IDE to interact with a DuckDB database!

2.3.6.3 Your first time with a DuckDb database

First thing that we will do is go to the website for the American National Election Study and will download a CSV file of the timeseries data, import the CSV and make it a table in a .duckdb database file. We will then look at our data after storing it in the database. As I will discuss later in the book, `SQL` through `DuckDB` is not only efficient, readable, but it is reproducible in that I can download my CSV file and tansfer all of the data into my database without even needing to open the CSV file. It significantly reduces the chance of unknowingly hitting a key and changing some data or making some other mistake. The ANES Timeseries data includes all responses since they first started the survey until present day. Each study usually includes hundreds of variables and hundreds of survey responses. Therefore, the Timeseries file is massive with hundreds of columns and tens of thousands of rows. One problem with accessing and pre-processing the data with the libraries in the other languages is that it often requires the file to be opened and then loaded into the active `R`, `Python`, etc. session. This puts intense demands on your `RAM`. It also does not require me to use up my `RAM` by opening a program like Microsoft Excel (which does have limits on how large of a file it will open because of memory limitations). This is a huge advantage. These two features mean that I am less likely to make an undocumented change to my data as well as coming with significant efficiency boosts. Let me show you.

First thing to do is to download the data. So, first go to https://electionstudies.org/data-center/. Click on the “Time Series Cumulative Data File (1948-2020)” (may include years past 2020 if you are reading this a few years from the time of writing this). You will then go to “Download Data” on the left sidebar and click the link. You will be asked to sign-in in order to access the data. If you have an account with the ANES, sign in as usual. If you do not register for a free account. Once you have done that, it will bring you back to the page describing the cumulative data file. You should now see options to download a CSV, SPSS, STATA, or SYNTAX version of the data under the “Download Data” section. Download the CSV file. This will download a `.zip` file containing the codebooks and the data file as a `.csv` file. You will want to unzip the file.

Once you have downloaded the CSV file, we should be ready to go. Open `VSCode`. Open a fresh Powershell, zsh, or bash terminal by using the command “CRTL(Mac)/Shift + ‘”.

Once you have started a terminal session, type `duckdb`. This should bring up something that looks like this:

```
v0.8.1 6536a77232
Enter ".help" for usage hints.
Connected to a transient in-memory database.
```

```
Use ".open FILENAME" to reopen on a persistent database.
D
```

Once this has happened, you can type `open my_first_database.duckdb`. This will create a `DuckDB` database file as well as connecting you to the database. Once you have done that, you can close that terminal session and open a new one. Once you have a new terminal session, type in `harlequin my_first_database.duckdb`. This will open the `harlequin IDE` as well as create an empty `DuckDB` database file called `my_first_database`.

Next, we should make a `.sql` file that we write our `SQL` code into so we have all of that information stored for documentation purposes. So in `VSCode`, you can create a new file by going to the top of your `VSCode` window and click “File”, then click “New File”. Type in `csv_to_table.sql`. This will open an empty `.sql` file.

Now, we can write our `SQL` code to take the data in the ANES CSV file that we have downloaded and put it into our `DuckDB` database file that we just created as a table. To do this, we can write some SQL code:

```
/* Take the CSV file I downloaded,
  ... and make a table called anes_raw
  ... with it.
  Then store it in my database file.
*/
CREATE TABLE anes_raw AS /* Create a table called ANES raw with the contents...*/
SELECT * /* select all of the columns */
/* from my ANES's csv file */
FROM read_csv_auto('~/Desktop/anes_timeseries_cdf_csv_20220916/anes_timeseries_cdf
/* now commit this table to the database file */
/* tell it that I am done with my query with the semi-colon */
COMMIT;
```

There are a few things to explain here. First, in `SQL` comments are started with `/*` and ended with `*/`. Since comments are explicitly started and ended, comments can span multiple lines without having to start each new line with `/*`. Next, I am going to use the commands `CREATE` and `TABLE` to say that I want to create a new table and I am going to call that new table `anes_raw`. Now that I have told `SQL` that I want to make a new table, I need to specify what the contents of that table is going to be.

I indicate that by using the command `AS`. I then specify on a new line that I want to `SELECT` all columns (what the `*` represents) `FROM` the `.csv` file that I downloaded. Here I am not just specifying the name of that CSV file, but I also need to tell it where on my computer that CSV file is. In this case, I downloaded that CSV file and put it on my Desktop. The CSV file is still in the unzipped folder from earlier so I need to specify that too. Once I've passed the information about what CSV file I am talking about, I am going to pass an option to the `read_csv_auto` function made by `DuckDB` to make the loading of CSV files easy. This option `all_varchar = true` is me telling it to just load in all of the data as characters rather than trying to interpret them as numbers. Of course most of my variables are numbers and not characters so this will be a task for the pre-processing chapters. Finally, I will want to `COMMIT` this table to the database that I am currently connected to. To tell `SQL` that I am now done with my query, I will add a semicolon (;) at the very end of my query.

Now, to run this code, I copy and paste the code into my `harlequin` Query Editor near the bottom of my `VSCode` window. I will then click "Run Query". You should now be able to go to the "Data Catalog" and see that in the my_first_duckdb_file connection under main, there is a table called anes_raw with tons of variables. Let's say that I want to get a preview of my data just to be extra sure everything worked.

```sql
/* check to see if transfer to database worked */

SELECT * /* select all (*) columns */
FROM anes_raw; /* from the anes_raw table */
```

Here, I am going to `SELECT` all of the columns `FROM` my anes_raw table. And since my `harlequin` session is still connected to the database from my `my_first_duckdb_file.duckdb` file, `SQL` will first check for a table called anes_raw and then will grab and display all of the data in that table. I can take that code, paste it into my "Query Editor" in `harlequin` and then click "Run Query". You will see pretty quickly that the "Query Viewer" will show that it is loading something like "50,000 of 68,224 records". This is a lot of data so it will take a while to display the data. But this is your first sign that everything worked. After a few seconds, you will see a table of your data from the ANES!

2.4 Closing out the chapter

This was a lot of information! The goal was to offer a reference and to get you set up with at least one language so that you can follow along through the rest of the book. The rest of the book is designed to advance the argument that we have autonomy over the tools that we use and that to use a tool is a choice to use *that* tool for that particular task. We should be using the tool that is most effective at enabling us to write efficient, readable, and reproducible code. Sometimes this means we have to learn a new tool.

The coding aspects that we will cover in the remainder of the book will be focused on providing examples as to how to use some of these languages and their popular libraries to manage and pre-process our data. This chapter set you up and gave you very basic introductions to how the languages work so that you can follow along with the coding examples in at least one language. However, the expectation is not that you can get all of the code to work and fully understand the intricacies of all of these languages, or even one. I would recommend that you read this book, follow along as best as you can, and then choose which language or language(s) you want to get a deeper understanding of. Then you can return to this book as a reference for when you are managing or pre-processing your data with one of these languages.

3 A principled workflow for secure and replicable data management

There are three key issues that we have to grapple with when it comes to data management. The first is that we need to ensure that our data are secure for the protection of our subjects. The second is that we need to ensure the replicability of our project by keeping original data intact and any changes need to be documented. The final concern we should consider is that we should have quick access to our data, even when contained within relatively large datasets – I will refer to this as efficiency. This chapter presents a principled workflow that should aid researchers achieve these three goals.

In the United States, Institutional Review Boards (IRBs) are tasked with the responsibility of enforcing federal regulations that require that researchers keep their data secure and that they protect the confidentiality of research subjects. This means that we have an obligation to ensure that no one can uncover the identity of a participant either through obtaining information that directly identifies a participant or by using a combination of data that we collect to infer a participant's identity.

While enforcement of these regulations can vary to some extent, this often means that we need to ensure that our data stays stored on a secure computer or locked away if they are physical files and that we "de-identify" our data. Thankfully many computers come with increasingly sophisticated security to the underlying files stored on our hard drive. This often means that if we lose a computer, we are generally safe so long as we have used a password to restrict access to your files or by using more sophisticated credentials such as a security key, biometrics, or by using 2 Factor Authentication. Many of us often use some sort of cloud-based file storage such as Dropbox, Google Drive, or iCloud – which are all password protected and often mean that many of our files are not physically stored on our computer's hard drive (in the traditional sense). While securing our computer and our files on them is relatively easy to do and protects the confidentiality of our research subjects, maintaining the confidentiality of our subjects has become a bit more difficult due to increasing calls that we make our research replicable at every step of the process which often means that we need to share data with journals upon publication of our papers.

Resulting from a significant number of high-profile cases of famous academic papers and researchers who have been accused of mis-handling their data, there are increasingly higher expectations for researchers to provide documentation and materials that allow researchers to independently re-do your study. This occurs as the social sciences currently grapple with a "Replication Crisis" where there are many papers being retracted for a number of issues with the empirical evidence presented in them.

The common approach that researchers employ is to share the cleaned version of the dataset, that is often de-identified, with journals so that others may independently replicate the study. Some criticize this practice by arguing that this is still not entirely replicable. Many of these famous instances of data mis-handling shared "cleaned" versions of their datasets. This has lead many to argue that our analyses are not fully replicable until we can replicate literally every step of the process – from taking the raw data, cleaning it, and then analyzing it. This puts researchers who want their work to be replicable in a bind.

We cannot share our data in its rawest form due to our need to protect the identities of our participants. There will necessarily be some changes to our raw data that others cannot see and replicate. We can share raw but de-identified data, however. If we follow a consistent and principled set of practices to managing our data, then we should be able to provide data files that allow others to replicate essentially every step of our research, at least the parts that will impact our analyses.

While we need to be protective of our data and at the same time willing to share it so that others may access it, we have a third challenge that is increasingly salient for researchers: we need to be able to quickly access our data. There are two things that limit efficiency when it comes to data management: 1) protecting the identities of our participants through keeping our files secure and 2) the increasing size of our datasets in the social sciences.

A common paradox in security is that the more secure something is, the more inconvenient it is to use. A common example is internet accounts. 2 Factor Authentication has been a major advancement over passwords to secure our online accounts. However, many bemoan the added work one has to perform to access an account because of this extra step to login to an account. This same paradox can apply to protecting the identities of our research participants.

We can take extra steps besides storing the files behind a required login (either as a login to an account on a computer and/or a login to a cloud service). These steps could be things like encrypting the files and password protecting the files themselves, storing these files on a secure server hosted by our organization or an approved vendor, or not collecting *any* identifiable information whatsoever. However, these extra steps are not taken because they are a lot of extra work and slow down our ability to access the data for analysis as well as complicate the process of sharing our data for replication purposes with other researchers.

The second source of inefficiency for data management is that many of our datasets are becoming much larger (both in terms of number of observations, but almost universally in terms of the number of variables we collect data on). These datasets are becoming larger as computational social science and its techniques are increasing in popularity as well as higher competition for journal space for tests of causal hypotheses that often mean that reviewers and editors are increasingly critical of which confounds or moderators are excluded from some statistical test.

This increase in size of our datasets means that there are increasingly higher requirements for computational resources, resources that we may not always have access to or resources that others replicating our work may not have either. As a result, we need to be particularly concerned about the ease of accessing our data. Thankfully computers are increasingly powerful and come standard with more computational resources. However, these increases in resources are not always show up as large performance gains as many basic applications we rely upon use up more resources due to increased functionality. Additionally, a reliance on the increased availability to more computational resources assumes that we have the latest models of computers, which is not always the case. Relying on advancements in standard offerings of hardware is a relatively fragile position to be in. As a result, we can look for software solutions that have low resource overhead to maximize how much of our available resources may be dedicated to our research.

These three goals are all inter-related and as I have pointed out produce a number of tensions that creates a very tricky situation for researchers. We need to take every effort available to us to protect the identities of our participants while simultaneously attempting to avoid making our files so secure that it takes a significant amount of effort to access them as well as making efforts to assuage concerns of others that we are not engaging in academic malpractice by sharing as much documentation and raw data as possible with others. One reason the current state of data management for academic researchers is so tricky is because there is no concrete set of standards or best practices that researchers follow when it comes to data management.

Looking at posted syllabi and materials for the “Math Camp” that incoming political science PhD students take as well as introductory quantitative research methods classes for the “Top-10” departments (As defined by U.S. News and World Report), I see some departments such as Harvard discuss version control for files as well as how to document your research and code. However, there are no primers or sessions dedicated to managing one’s data or how to pre-process it. This reflects the experiences of many. We learn how to manage and pre-process our data as we go through these courses and as we work on projects with faculty. This ad-hoc approach to learning these skills also show up in the inconsistency by which researchers manage and pre-process their data. It is not a stretch that many of us have quite different experiences working with different collaborators on how the team manages and pre-processes the data for a project.

While we receive extensive training and spill a significant amount of ink about where to collect data, what data to collect and how to analyze them, my examination of syllabi in top departments for their introductory quantitative research methods courses and bootcamps confirms my suspicions that we receive little-to-no guidance in what to do with the data between the steps of collecting and analyzing those data. Beyond this training, we follow IRB regulations by storing our data on a password protected computer and then make judgment calls about what subset of our data to share publicly with the journal and readers of our papers. Because of the inconsistency in training and enforcement mechanisms, we have non-standardized processes for managing our data. This naturally leads to very idiosyncratic workflows for data management and these patterns reflects the extent to which these steps are an afterthought in the research process. The goal of this chapter is to convince you that the current state of data management in the social sciences indeed poses a problem and that we can be more secure, replicable, and efficient in terms of data management.

3.1 Examples of current data management workflows

Let me provide a couple of case studies of the current way that people manage their data.

Case Study 1: A researcher downloads a `.csv` (or `.xlsx`, `.dta`, or even a `.sav`) file from the site that you hosted your study and collected your data on. The researcher then opens the file and starts relabeling column names and values. For example, the researcher may have a column that originally comes in something like "VAR_001" and they change the column name to "PARTICIPANT_ID". The researcher also notices that some of the variables have rows with the label of the response rather than an integer. So the researcher starts to change any cell that says "Strongly Disagree" to "1". Once the researcher have gone through all of the rows, they save a new copy of the file and call it "cleaned-data.csv". Then they load the file and start performing descriptive statistics and fit simple regressions to get a preliminary sense of whether the study paid off. If the researcher find any mistakes, you open the "cleaned-data.csv" file again to fix them or go back to the original file and restart the cleaning process. Or the researcher may even write code to correct the mistakes in the "cleaned-data.csv" file rather than having to deal with fixing those mistakes by restarting or trying to find them in a few-hundred row spreadsheet.

There are a number of things that are dangerous to this process. The first is that the data management and data pre-processing is not at all replicable. The goal of replicability is that someone else can take a researcher's original data, and they can follow step-by-step what they did to come to the exact same conclusions.

The first thing that is dangerous is that the file is not secure. If the researcher leaves their computer open or logged-in and someone gets access to it, the unauthorized user can quickly and easily open that `.csv` file. Confidentiality for the study's participants are extremely dependent on a researcher's choices to ensure that their computer is not easily accessed by others. While often password protecting our computers is often sufficient to meet the IRB guidelines that social scientists often need to meet, we often store these files on a cloud-based hosting service such as Dropbox, Google Drive, or iCloud. Both ways of storing these files means that they are only as secure as our accounts are. As we often see in the news, only using a password protected account often is not sufficient to keep prying eyes from accessing our accounts. In other words, there is nothing stopping us from taking a few extra steps to make it harder for those without authorized access to view data that needs to be kept confidential. Unlike common steps like using 2 Factor Authentication that require a lot of extra work only for the purposes of security, the workflow that I advocate for later in this chapter actually makes access to our data more efficient, replicable, and secure – it finds a balance to achieve all three goals without having to sacrifice one of these goals in the name of security as the common paradox for security often posits.

The second issue relates to the reproducibility of the researcher's data management. Manually editing the original file does not populate any documentation about what things the researcher is clicking or typing. There is so little documentation, that the researcher might not even be able to replicate the their own steps to manage and pre-process their data; let alone someone else. Even scarier, the researcher's finger may slip and they type the wrong number or press the delete key while going over the wrong cell. Sometimes the researcher catches this, but sometimes they may not. Without this documentation, if another researcher finds discrepancies there can only be speculation as to whether those discrepancies arose from research design choices or data management issues. Not only can this carry professional costs, but it also limits our ability to be confident that we understand the true answer to the researcher's question.

The final issue is that these steps are highly inefficient. Manually going through cells on a spreadsheet may take a few minutes to a few hours to clean if a researcher has a few hundred rows with a dozen or so columns. However, many of the data sets that we use in the social sciences are often approaching thousands (if not millions) of rows and hundreds of columns; the dimensionality of our studies are only increasing. To do this for such a large dataset would take a significant amount of time and any mistakes would be extremely hard to catch in such a vast amount of data. This also all assumes that the dataset is not so large that common software – such as Microsoft Excel – actually opens the file and that we are not filling up precious space on our Hard Drive to store such a large file.

The most likely thing that will happen when the researcher is asked to share data with the journal upon publication of their research is that a researcher will just share the cleaned excel file. Why? Well, because researcher's are not going to want to reinvent the wheel. At this step, it can also be quite dangerous. Not only does this cause the lack of replicability of a researcher's analysis, but the researcher may forget to remove identifiable information from the file – there is no real clear step that the researcher had to take to remove this information, so in the fog of trying to get a paper prepared for publication, it is relatively easy to forget to do this if it is not baked into one's process. If this happens, the researcher has failed to meet federal regulations requiring that they take every effort to ensure your participants are not easily identifiable.

Case Study 2: Similar to Case Study 1, the researcher downloads their file. Instead, the researcher opens up an R, STATA, or Python session. The researcher starts writing code in a cleaning script that renames the columns, and recodes columns to have the correct integer values. The researcher then writes code to perform descriptive statistics and to fit simple regressions for preliminary analyses. While doing this, the researcher may notice some mistakes and so they go back to their code and make some adjustments to erase those mistakes.

This process is a little bit more replicable and efficient than the first one. Writing code allows the researcher to systematically make changes to any cell where some set of conditions are met. The code also does this relatively quickly (in many cases). This also is safer (in terms of replicability) as the code acts as the researcher's documentation as to what they changed from the original dataset for the study.

While this is a vast improvement over the first case, there are still some aspects of this that are relatively dangerous. The first is that the replicability and accuracy of the researcher's documentation is still heavily dependent on whether they write their code as a script or run the code line-by-line (interactively). Executing code as a script means that a researcher runs the whole file at once rather than executing the code line-by-line (interactively). Executing this code as a script is much more replicable because it does not allow researchers to cherry-pick which lines of code they execute, but it requires one's code to be clean and to *only* execute what one intends to execute. The second is that there is no clear step to ensure that the researcher's data is de-identified to ensure that they do not accidentally share data that would betray the identity of their subjects. Even still, if the researcher take steps to de-identify the data, they cannot include that code in their script or else it will throw errors back at an independent researcher that only has the de-identified version of the data. So this still means that there are a high number of researcher degrees of freedom by which some researchers can be more or less compliant with protecting the anonymity of their subjects as well as providing documentation for replicating one's analyses.

In the rest of this chapter I will introduce and advocate for a workflow that should enable replicable, efficient, and secure files for one's research. As I will argue, this process removes much of the ad-hoc and idiosyncratic processes by replacing particular choices with a formulaic and principled set of steps for one to follow when processing data from a study. What this process also encourages is that researchers use specialized tools designed to be used for the distinct steps of the process. As the next chapter will grapple with more, moving away from using a single tool for all steps of the research process encourages one to follow the best practices for each step as these tools' capabilities are a reflection of the widely-accepted best practices for that particular task.

3.2 A principled data management workflow

Figure 4.1 provides an overview to my proposed workflow to data management. The remainder of this chapter will cover each of these steps and explain what the value added is in terms of security, replicability, and efficiency by using this workflow.

In **Step 1**, you should make sure to download a `.csv` file if possible. Why a `.csv` file specifically? Other file types such as STATA's `.dta`, Microsoft Excel's `.xlsx` and other common ones often have some degree of proprietary protection of them and one's ability to access that raw data is dependent on these companies' continued willingness to allow for users to develop packages that allow for one to load that file.

Further, as I will advocate for in the next chapter, one should use the `SQL` language for their data management as it is designed for such a task and is often the most replicable, efficient, and secure option. `DuckDB`'s implementation of `PostgreSQL` enables one to create tables from a raw data `.csv` file and to initiate a new `.duckdb` file without having to open the original data file or loading it with some other language. This reduces the chances that one makes changes to the data that are not documented through the automatic formatting that occurs by Microsoft Excel, Apple Numbers, or some other software when parsing a `.csv` file to make it more readable in a tabular format. It also removes the temptation to perform ad-hoc data management by loading the file into R, Python, Julia or some other language for data analysis.

Once you have downloaded the raw data as a `.csv` file, you can immediately load and interface with it using `DuckDB`. Once you have done this, in **Step 2** you should create a case (participant) identifier column. This can be as simple as creating a column that records the current row number for the participant. As you will see based on my recommendations in the next few steps, the goal of this is that you should decentralize your data so that you ensure that you do not accidentally share information that makes it easy for people to identify your participants, while also not deleting data so that, internally, you may have access the original data in its complete form.

In **Step 3** you can start creating and storing tables in your `.duckdb` file. You should pull any identifiable data from the loaded data and store it in a separate table. This sort of information would certainly be columns containing the names and addresses (mailing or IP), Session ID's, Time and/or location they took the study from, as well as any Participant ID provided by a vendor. With this subset of your original data, you can store a table with a copy your Participant ID column (from **Step 2**).

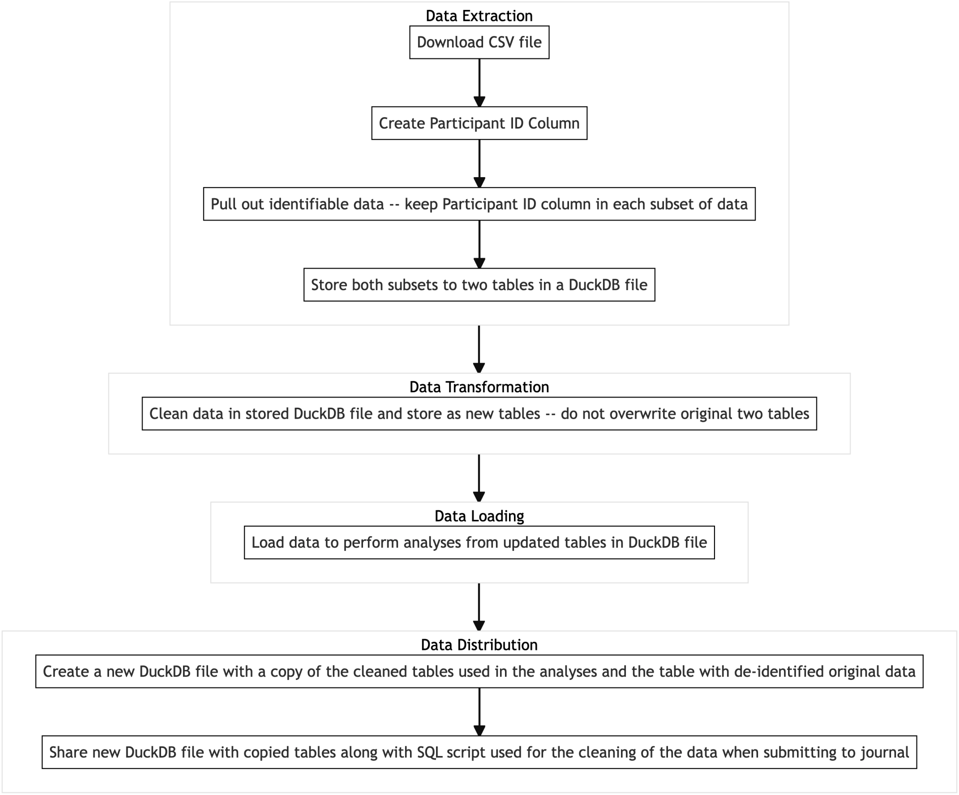
Data Extraction
Download CSV file
Create Participant ID Column
Pull out identifiable data -- keep Participant ID column in each subset of data
Store both subsets to two tables in a DuckDB file
Data Transformation
Clean data in stored DuckDB file and store as new tables -- do not overwrite original two tables
Data Loading
Load data to perform analyses from updated tables in DuckDB file
Data Distribution
Create a new DuckDB file with a copy of the cleaned tables used in the analyses and the table with de-identified original data
Share new DuckDB file with copied tables along with SQL script used for the cleaning of the data when submitting to journal

Figure 3.1: A principled workflow for data management

Keeping these data allow you to immediately de-identify your data while also not deleting it so that you are able to continue to use any of that data in case you need to apply exclusion restrictions, use those data for payment to participants, confirm participation in a study, etcetera. The generated Participant ID column allows you to merge data from that table with identifiable information if need be at a later time, but ensures that you do not have *any* data that may make it easy to identify your participants in an analysis or any of your files that you make publicly available.

Taking this step early on not only helps with ensuring the confidentiality of your participants, but doing it this way helps with replicability in that you will have to write `SQL` code that documents every step you took from downloading the file on your computer through your analysis. It also aids in efficiency in that you will still have those data that you can easily merge with the main subset of your data stored in the second table in case you need to when figuring out which participants to exclude from the study, confirm whether participants completed the study and are eligible for compensation, etcetera. Also, if internally, you need to demonstrate that your data is intact, then you can easily merge on your generated Participant ID column.

Once you have pulled out the identifiable information about your subjects that you want to keep separate from the subset you will use for your analyses, you will want to store both as separate tables in your `.duckdb` file. So, by the end of **Step 4**, you should have a `.duckdb` file saved on your computer that contains two tables: a table with your Participant ID column (**Step 2**) that contains identifiable information about your participants, and a second table that contains the same Participant ID column (**Step 2**) with the data that you will use for your analyses.

The benefit of having a `.duckdb` file rather than separate files or a single excel file with multiple sheets containing the same amount of information is that if there is unauthorized access to your computer a `.duckdb` file requires that someone write `SQL` code in order to view any of the contents of the file. This increases security in the event that someone gains access to the file. Obviously if the person with unauthorized access to the file understands `SQL` they can access the data. However, the architecture of the file (having multiple tables and knowing which column contains what information and what that information represents) also requires knowledge about the file that should be ideally stored as some sort of codebook or separate documentation. The increased requirements for technical know-how and of internal documentation of the "schema" (the structure of the tables within the file) significantly increases the complication of viewing the data for those that are not part of the research team, thus increasing security and your ability to retain the confidentiality of your subjects' identities.

Once we have completed **Step 4**, we can begin to write `SQL` code to clean our data in **Step 5**. While this asks researchers to eschew packages and languages that many academics have become accustomed to using, as I argue in the next chapter, it ensures that we are using the right tool for the task. `SQL` is a language designed with the express purpose of data management. Because of its specialization, this means that the language's capabilities, design, and workflow are optimized for this particular task. There is also something to be said about the psychological benefits of using different tools for different steps of the research process – it encourages you to consider each step as complete and separate, therefore encouraging better documentation.

As I will discuss more in the following chapter as well, many of the languages that researchers use for data analysis (and as a consequence default to using the same language for data management), have very large and very popular packages for data management. While these packages are large, open source, and have teams of professional developers who consistently seek to increase the functionality and efficiency of the functions in those packages, they also change rapidly by changing names of functions, implementing new functions, depreciating functions, and make adjustments to how those functions work. These characteristics are the main sources of criticism towards using them as they create many headaches for researchers as new versions of these packages often create new errors or depreciation warnings that make it harder to be confident in the continued replicability of one's code. There are also other criticisms to some of these packages as they are very reliant on other functions (have a relatively higher number of dependencies) and if those packages that they depend upon change or are no longer supported, this creates problems as well as it can limit how fast one's code runs as well as the replicability of one's code.

These features are reflective of what these languages are often built for: data analysis. As we innovate in the models and techniques available to us to detect patterns in our data, the need for constant integration of these new tools encourages developers and applied users to adjust our code to reflect these changes in standards and techniques.

`SQL`, however, is a quite old language (in terms of coding languages existing today). It also has not changed much. There are two primary flavors of `SQL` such as the open-source `PostgreSQL` that `duckdb` relies upon. Given that most needed innovations in data management are focused on a user's ability to access an increasingly larger volume of data in a shorter amount of time, these demands do not require or incentivize changes to the language's functions that a researcher would use. As a result, we do not need to worry about our `SQL` code needing to be updated in response to changes to function names or any depreciations. The only thing that really changes are the versions of the packaged software that we interface with, but the underlying code does not because the developers are responding to demands to make the database file sizes smaller, for them to load faster, and to do so with ballooning datasets; but the common need to select certain columns, rename them, use row-wise and column-wise aggregations, etc. do not change.

If I have successfully convinced you to do the data cleaning with `SQL` rather than using a package in a language for data analysis, we will want to save a *third* table that contains the cleaned data you will use in your analyses. From there, in **Step 6** we can begin to perform our analyses by loading this third, cleaned, table of the data for our analyses.

After we have completed our analyses and we are preparing our paper for submission to a journal or to share it publicly, we will want to perform **Step 7** which means that we will create and save a new `.duckdb` file. This second `.duckdb` file will store a copy of the cleaned and de-identified data table(s) we use in our analyses as well as the table containing the original and de-identified data.

3.3 Conclusions

I hope with this chapter I have given you a sense of my proposed workflow. You still may not be convinced that this is workflow is better or that it is more worth doing that what you currently do. After all, I am encouraging you to eschew packages and languages that you are comfortable with and probably have used for quite a while in favor of one that may seem unnecessary. That is fine. As our data management workflows are so idiosyncratic, it is not easy nor prudent to compare every single workflow academics have with data management to the one that I propose here. Therefore, it is absurd to claim that this proposed workflow is better than all possible workflows.

Nevertheless, my primary goal of this chapter is to introduce this workflow and to plant the seeds about how having *any* principled and formulaic workflow for managing data is a vast improvement to any *ad hoc* approach in terms of security, reproducibility, and efficiency. I also hope that the present chapter makes it clear that researchers are stuck in this difficult position of having to balance concerns about security, replicability, and efficiency that seem at odds with one another. Instead of having to choose which of these goals to pursue with our projects, I hope that this chapter demonstrated how my proposed workflow threads the needle by reaching all three of these goals simultaneously.

The primary goal of the following chapter is to encourage the uptake of this particular workflow and its tools – or at least to be used as a skeleton workflow that you make amendments to for your particular needs. I hope to show what this workflow looks like in practice and to demonstrate how painless it is to implement. I also hope that the following chapter demonstrates how the particular tools I advocated to use in this workflow are worth learning and using. Therefore, the following chapter will contain much more code.

4 The basics of data extraction

```r
#| label: r-setup-block
#| include: false
# Load libraries

library(tictoc)
library(readr)

# Specify Path of CSV

csv_path <- '../data/WaffleDivorce.csv'
```

The last chapter presented an overview of a principled workflow for data processing in a research project. I include a diagram of this workflow in Figure 4.1 for your reference. This chapter will focus on the first part of the workflow: data extraction. When discussing data extraction, I am referring to the idea that you are taking raw data outputs from some source and are converting it into a usable tabular format. In this chapter, we are going to cover common data formats such as CSV and TSV files, Parquet files, and proprietary file types such as Microsoft's XLSX, STATA's DTA, and SPSS' SAV files.

Though I recommend that you perform many of these tasks with `DuckDB`'s implementation of `SQL` goal of this chapter is to give you some examples of performing these steps with code in `Python`, and `R` as well. The objective is to encourage researchers to use this principled data processing strategy, even if I am unable to convince everyone to use `SQL`. I also will provide coding benchmarks to demonstrate the efficiency of `DuckDB`'s implementation of `SQL` relative to the common libraries in `Python`, and `R`.

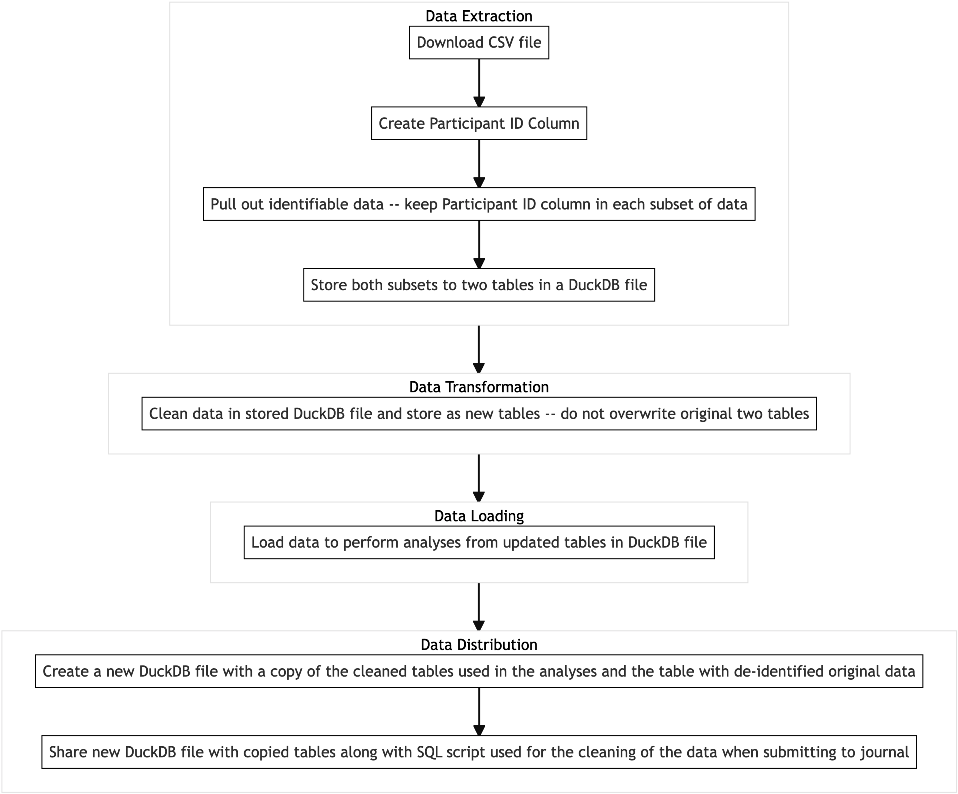
Data Extraction
Download CSV file
Create Participant ID Column
Pull out identifiable data -- keep Participant ID column in each subset of data
Store both subsets to two tables in a DuckDB file
Data Transformation
Clean data in stored DuckDB file and store as new tables -- do not overwrite original two tables
Data Loading
Load data to perform analyses from updated tables in DuckDB file
Data Distribution
Create a new DuckDB file with a copy of the cleaned tables used in the analyses and the table with de-identified original data
Share new DuckDB file with copied tables along with SQL script used for the cleaning of the data when submitting to journal

Figure 4.1: A principled workflow for data management

4.1 Data extraction

What is data extraction? Data extraction is a task that every researcher engages in. While many may think that it is a easy process, the reality is that in *some* cases data extraction is an easy process. When working with pre-compiled datasets, data extraction can certainly be an easy process. You can download some file and then load it into your statistical software of choice. However, even when you have a pre-compiled dataset, this process can still be quite challenging.

As the scale and the number of types of data that social scientists consider for their projects increases, data extraction steps are less standard and are orders of magnitude more complicated. In this chapter, we are focusing on data sets that have already been compiled, are in a tabular format, and are relatively small (thousands of rows and a couple of hundred columns). However, when we are working with raw data that are not in tabular format already – such as reading text from PDF files – or contains millions of observations for thousands of variables, data extraction is an extremely difficult process and choosing a tool that is responsive to scalability and facilitate documentation are imperative. The next chapter will provide examples of some of these more complicated features. However, the remainder of this chapter will be focused on providing simple and common examples of the data that social scientists will want to extract.

4.2 Loading CSV files

Loading CSV files is a common data extraction task that many experienced researchers are familiar and comfortable with. Often times, to execute the code for data extraction takes milliseconds once you have the dataset's CSV file downloaded on your computer for many common datasets we encounter in the social sciences.

Since CSV files are such a common file to extract our data from, let's start there. The CSV file I am working with in these examples is a dataset that has recorded the amount of Waffle Houses in a particular state and information about the rate of marriages and divorces in that same state. This is a relatively small dataset as it is 50×13 (50 rows and 13 columns). I've included this CSV file in the supplementary materials for the book online. For those interested in going straight to the source for the data, you should reference McElreath (2020).

4.2.1 Python

In Python there are two dataframe management libraries that I will consider. Of course, there are many more than this, but here I want to focus on Pandas which is by far one of the most popular libraries in Python. I will also discuss Polars which is newer and not as adopted as Pandas. The benefit of looking into Polars is that it is extremely performant. The Polars library is a API for Polars which is written natively in Rust (an increasingly popular alternative to compiled languages such as C++).

Once I downloaded the dataset, I put it in my project folder called data/. I first want to specify the location of that particular file. It is not sufficient for me to just tell my computer to look for a file called WaffleDivorce.csv because it would take an extremely long time for my computer to look through millions of files for a matched name. The other problem of not being a bit more specific about the location of the file is that I may have multiple files on my computer called WaffleDivorce.csv but for other projects. If you have ever looked through your computer and have found multiple files called Untitled1.docx, this is a pretty common experience that many of us have. Just as you'd be confused if you need to open Untitled1.docx on your Desktop or in a specific folder somewhere else on your computer, so is your computer if you aren't a bit more specific about *where* on your computer to find the file. So, in this case, I am going to tell it to find the file in my current project (./) that is within my data/ folder. So, the specific path to the CSV is ./data/WaffleDivorce.csv.

Once, I have specified the path to my file by placing it in an object called csv_path, I can then load the Pandas library by writing import pandas. I have an extra step on that same line where I add as pd which allows me to load Pandas but to give it a nickname so I do not have to write out pandas to then access a function.

So as you see in the following line where I load the csv file, I write pd.read_csv. Which means that I am using the function read_csv from the pandas library.

```
# Define location of csv file
csv_path='./data/WaffleDivorce.csv'

# Pandas
    #* Import Pandas
import pandas as pd
    #* Load CSV
```

```
waffle_df = pd.read_csv(csv_path, delimiter=';')

# Polars
    #* Import Polars
import polars as pl
    #* Load CSV
waffle_df = pl.read_csv(csv_path, separator=';')

# Preview of dataframe
waffle_df.head()
```

I use the `timeit` package to calculate the average amount of time it took `Pandas` and `Polars` to load the WaffleDivorce.csv file 100 times. It took `Pandas` on average 3.17e-04 (Std. Deviation = 9.537854e-05) seconds while it took `Polars` an average of 1.06e-04 (Std. Deviation = 8.34e-05) seconds.

4.2.2 R

```
#| label: r-waffle-house-load
#| eval: false
# Base R
    #* Load CSV file
waffle_df <- read.csv(csv_path, sep=';')

# Tidyverse
    #* Load readr
library(readr)
    #* Load CSV file
waffle_df <- read_delim(csv_path, delim=';')

# Preview dataframe
head(waffle_df)
```

```r
#| label: profile-r-waffle-house-load
#| include: false
# Base R
    #* Clear tictoc log
tic.clearlog()
    #* Profiling each step of this process
for(i in 1:100) { # repeat the following 100 times
    tic(i) # start timer
    waffle_df <- read.csv(csv_path, sep=';') # load the csv
    waffle_df # return the result of it
    toc(i, quiet=TRUE) # stop timer
}
base_r_benchmark_log <- tic.log(format=FALSE) # log how long this loop took
base_r_benchmark <- unlist(lapply(base_r_benchmark_log, function(x) x$toc - x$tic)

# Tidyverse
    #* Clear tic toc log
tic.clearlog()
    #* Profiling each step of this process
for(i in 1:100) { # repeat the following 100 times
    tic(i) # start timer
    waffle_df <- read_delim(csv_path, delim=';', show_col_types=FALSE) # load the
    waffle_df # return the result of it
    toc(i, quiet=TRUE) # stop timer
}
tidyverse_benchmark_log <- tic.log(format=FALSE) # log how long this loop took
tidyverse_benchmark <- unlist(lapply(tidyverse_benchmark_log, function(x) x$toc -
```

```r
#| output: asis
#| echo: false
cat(
    "I use the `tictoc` package to calculate the average amount of time it took `B
    , sep=""
)
```

4.2.3 DuckDB via Python API

```python
# Import the DuckDB library
import duckdb as db

# Use the Python API for DuckDB to connect to the database
con = db.connect('./data/WaffleDivorce.db')

# Use the connection to load the dataset as a Polars DataFrame
waffle_df = con.execute(# Execute the following SQL query ...
    '''
        SELECT * -- select all of the columns
        FROM main -- from the table called 'main'
    '''
).pl() # ... then store the result as a polars dataframe

# Preview the dataframe
waffle_df.head()
```

I use the `timeit` package to calculate the average amount of time it took the `DuckDB` API in `Python` to load the main table from the WaffleDivorce.DB 100 times. It took `DuckDB` on average 9.51e-05 (Std. Deviation = 2.07e-05) seconds.

References

McElreath, Richard. 2020. *Statistical Rethinking: A Bayesian Course with Examples in r and Stan*. Second edition. Boca Raton, FL: Chapman & Hall/CRC.