

Introduction to Statistics and Data Science

Elizabeth Tipton Arend M Kuyper Danielle Sass
Kaitlyn G. Fitzgerald Adapted from ModernDive by
Chester Ismay and Albert Y. Kim

2022-09-19

Table of contents

Welcome	4
License	4
Preface	5
Introduction for students	5
What you will learn from this book	6
Data/science pipeline	8
Reproducible research	9
 I Getting started	 11
1 Getting Started with Data in R	12
1.1 What are R and RStudio?	12
1.1.1 Using RStudio Cloud	13
1.1.2 Installing R and RStudio on your personal computer	14
1.1.3 Using R via RStudio	14
1.2 How do I code in R?	15
1.2.1 Creating your first R Markdown document	16
1.2.2 Basic programming concepts and terminology	17
1.2.3 Errors, warnings, and messages	18
1.2.4 Tips on learning to code	19
1.3 What are R packages?	20
1.3.1 Package installation	21
1.3.2 Package loading	22
1.3.3 Package use	23
1.4 Explore your first dataset	23
1.4.1 <code>nycflights13</code> package	24
1.4.2 <code>flights</code> data frame	25
1.4.3 Exploring data frames	26
1.4.4 Help files	28
1.5 Conclusion	29
1.5.1 Additional resources	29

II	Data Exploration via the tidyverse	30
2	Data Visualization	31
	Packages Needed	31
2.1	The Grammar of Graphics	31
2.1.1	Components of the Grammar	32
2.1.2	Gapminder data	32
3	Data Wrangling	33
4	Data Importing & “Tidy Data”	34
III	Data Modeling	35
5	Basic Regression	36
6	Multiple Regression	37
IV	Statistical Theory	38
7	Randomization and Causality	39
8	Populations and Generalizability	40
9	Sampling Distributions	41
V	Statistical Inference	42
10	Confidence Intervals	43
11	P-values	44
12	Hypothesis tests	45
13	Putting it all together	46
	References	47
	Appendices	47
A	Statistical Background	48

Welcome

This is the website for **Introduction to Statistics and Data Science**. This book starts you down the path of learning how to think with data using R. You'll learn the basics of how to engage, explore, and examine many types of data arising from several contexts. Hopefully you'll have fun and see how valuable it is to be able to critically think with data.

⚠ Warning

Please note that this is a “development version” of this book for the new design of STAT 202. Meaning this is a work in progress being edited and updated as we go. We would appreciate any feedback on typos and errors.

This open textbook is produced with support from [Northwestern University Libraries](#) and [The Alumnae of Northwestern University](#).



Figure 1



Figure 2

License

This website is (and will always be) **free to use**, and is licensed under the [Creative Commons Zero v1.0 Universal](#) License. If you'd like to give back, please consider reporting a typo or leaving a pull request at github.com/NUstat/intro-stat-data-sci.

Preface

Help! I'm new to R and RStudio and I need to learn about them! However, I'm completely new to coding! What do I do?



Figure 3



Figure 4

If you're asking yourself this question, then you've come to the right place! Start with our “Introduction for Students”.

Introduction for students

This book assumes no prerequisites: no algebra, no calculus, and no prior programming/coding experience. This is intended to be a gentle introduction to the practice of analyzing data and answering questions using data the way statisticians, data scientists, data journalists, and other researchers would.

In Figure 5 we present a flowchart of what you'll cover in this book. You'll first get started with data in Chapter 1, where you'll learn about the difference between R and RStudio, start coding in R, understand what R packages are, and explore your first dataset: all domestic departure flights from a New York City airport in 2013. Then

1. **Data Exploration:** You'll assemble your data science toolbox using `tidyverse` packages. In particular:
 - Ch. 2: Visualizing data via the `ggplot2` package.
 - Ch. 3: Wrangling data via the `dplyr` package.
 - Ch. 4: Understanding the concept of “tidy” data as a standardized data input format for all packages in the `tidyverse`
2. **Data Modeling:** Using these data science tools, you'll start performing data modeling. In particular:

- Ch. 5: Constructing basic regression models.
 - Ch. 6: Constructing multiple regression models.
3. **Statistical Theory:** Now you'll learn about the role of randomization in making inferences and the general frameworks used to make inferences in statistics. In particular:
- Ch. 7: Randomization and causality.
 - Ch. 8: Populations and generalizability.
 - Ch. 9: Sampling distributions.
4. **Statistical Inference:** You'll learn to combine your newly acquired data analysis and modeling skills with statistical theory to make inferences. In particular:
- Ch. 10: Building confidence intervals.
 - Ch. 11: Calculating p-values.
 - Ch. 12: Conducting hypothesis tests.

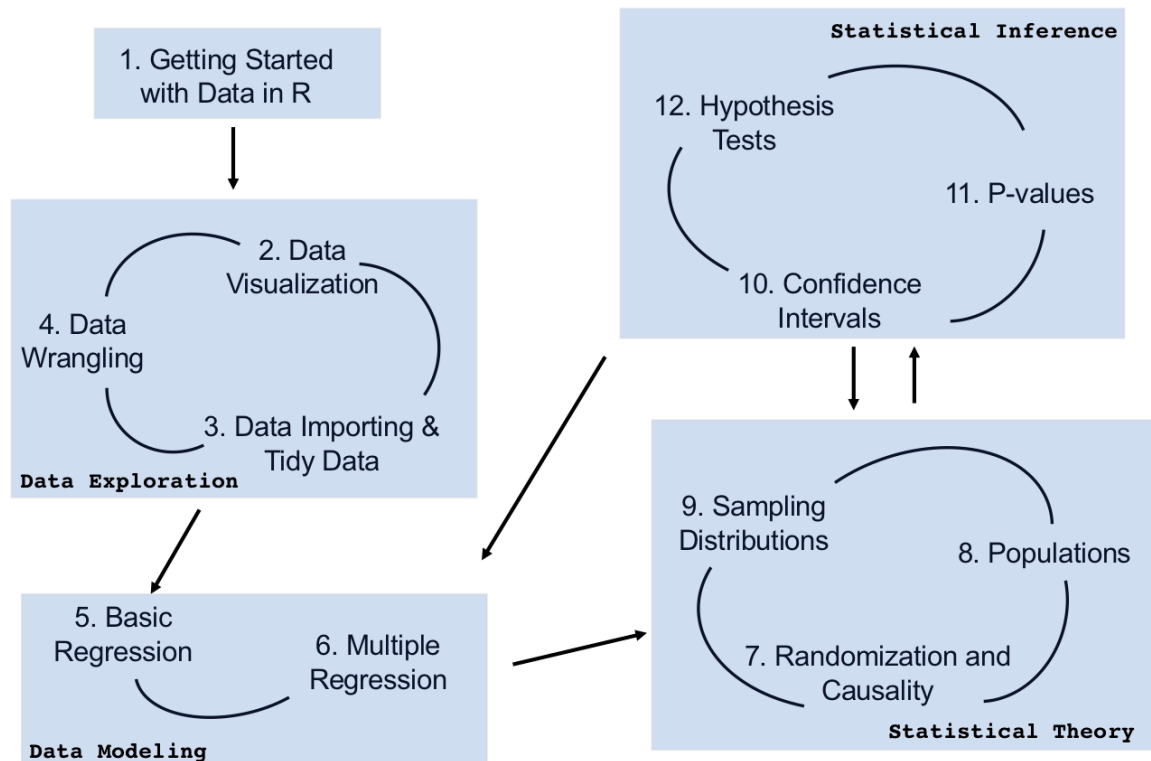


Figure 5: Course Flowchart

What you will learn from this book

We hope that by the end of this book, you'll have learned

1. How to use R to explore data.
2. How to generate research questions and hypotheses.
3. How to think like a statistician and the role of chance in your data.
4. How to answer statistical questions using tools like confidence intervals and hypothesis tests.
5. How to effectively create “data stories” using these tools.

What do we mean by data stories? We mean any analysis involving data that engages the reader in answering questions with careful visuals and thoughtful discussion, such as [How strong is the relationship between per capita income and crime in Chicago neighborhoods?](#) and [How many f**ks does Quentin Tarantino give \(as measured by the amount of swearing in his films\)?](#). Further discussions on data stories can be found in this [Think With Google article](#).

For other examples of data stories constructed by students like yourselves, look at the final projects for two courses that have previously used a version of this book:

- Middlebury College [MATH 116 Introduction to Statistical and Data Sciences](#) using student collected data.
- Pacific University [SOC 301 Social Statistics](#) using data from the [fivethirtyeight R package](#).

This book will help you develop your “data science toolbox”, including tools such as data visualization, data formatting, data wrangling, and data modeling using regression. With these tools, you’ll be able to perform the entirety of the “data/science pipeline” while building data communication skills.

In particular, this book will lean heavily on data visualization. In today’s world, we are bombarded with graphics that attempt to convey ideas. We will explore what makes a good graphic and what the standard ways are to convey relationships with data. You’ll also see the use of visualization to introduce concepts like mean, median, standard deviation, distributions, etc. In general, we’ll use visualization as a way of building almost all of the ideas in this book.

To impart the statistical lessons in this book, we have intentionally minimized the number of mathematical formulas used and instead have focused on developing a conceptual understanding via data visualization, statistical computing, and simulations. We hope this is a more intuitive experience than the way statistics has traditionally been taught in the past and how it is commonly perceived.

Finally, you’ll learn the importance of literate programming. By this we mean you’ll learn how to write code that is useful not just for a computer to execute but also for readers to understand exactly what your analysis is doing and how you did it. This is part of a greater effort to encourage reproducible research (see subsection *Reproducible research* for more details). Hal Abelson coined the phrase that we will follow throughout this book:

“Programs must be written for people to read, and only incidentally for machines to execute.”

We understand that there may be challenging moments as you learn to program. We still continue to struggle and find ourselves often using web searches to find answers and reach out to colleagues for help. In the long run though, we all can solve problems faster and more elegantly via programming. We wrote this book as our way to help you get started and you should know that there is a huge community of R users that are always happy to help everyone along as well. This community exists in particular on the internet on various forums and websites such as stackoverflow.com.

Data/science pipeline

You may think of statistics as just being a bunch of numbers. We commonly hear the phrase “statistician” when listening to broadcasts of sporting events. Statistics (in particular, data analysis), in addition to describing numbers like with baseball batting averages, plays a vital role in all of the sciences. You’ll commonly hear the phrase “statistically significant” thrown around in the media. You’ll see articles that say “Science now shows that chocolate is good for you.” Underpinning these claims is data analysis and a theoretical model relating the data collected in a sample to a larger population. By the end of this book, you’ll be able to better understand whether these claims should be trusted or whether we should be wary. Inside data analysis are many sub-fields that we will discuss throughout this book (though not necessarily in this order):

- data collection
- data wrangling
- data visualization
- data modeling
- statistical inference
- correlation and regression
- interpretation of results
- data communication/storytelling

These sub-fields are summarized in what Grolemund and Wickham term the “[Data/Science Pipeline](#)” in Figure 6.

We will begin by digging into the gray **Understand** portion of the cycle with data visualization, then with a discussion on what is meant by tidy data and data wrangling, and then conclude by talking about interpreting and discussing the results of our models via **Communication**. These steps are vital to any statistical analysis. But why should you care about statistics? “Why did they make me take this class?”

There’s a reason so many fields require a statistics course. Scientific knowledge grows through an understanding of statistical significance and data analysis. You needn’t be intimidated by

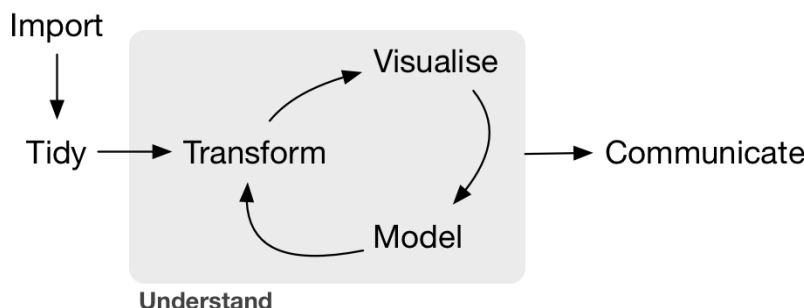


Figure 6: Data/Science Pipeline

statistics. It's not the beast that it used to be and, paired with computation, you'll see how reproducible research in the sciences particularly increases scientific knowledge.

Reproducible research

"The most important tool is the *mindset*, when starting, that the end product will be reproducible." – Keith Baggerly

Another goal of this book is to help readers understand the importance of reproducible analyses. The hope is to get readers into the habit of making their analyses reproducible from the very beginning. This means we'll be trying to help you build new habits. This will take practice and be difficult at times. You'll see just why it is so important for you to keep track of your code and well-document it to help yourself later and any potential collaborators as well.

Copying and pasting results from one program into a word processor is not the way that efficient and effective scientific research is conducted. It's much more important for time to be spent on data collection and data analysis and not on copying and pasting plots back and forth across a variety of programs.

In a traditional analysis if an error was made with the original data, we'd need to step through the entire process again: recreate the plots and copy and paste all of the new plots and our statistical analysis into your document. This is error prone and a frustrating use of time. We'll see how to use R Markdown to get away from this tedious activity so that we can spend more time doing science.

"We are talking about *computational* reproducibility." - Yihui Xie

Reproducibility means a lot of things in terms of different scientific fields. Are experiments conducted in a way that another researcher could follow the steps and get similar results? In this book, we will focus on what is known as **computational reproducibility**. This refers to being able to pass all of one's data analysis, data-sets, and conclusions to someone else and have them get exactly the same results on their machine. This allows for time to be

spent interpreting results and considering assumptions instead of the more error prone way of starting from scratch or following a list of steps that may be different from machine to machine.

Part I

Getting started

1 Getting Started with Data in R

Before we can start exploring data in R, there are some key concepts to understand first:

1. What are R and RStudio?
2. How do I code in R?
3. What are R packages?

We'll introduce these concepts in upcoming Sections [1.1](#) - [1.3](#). If you are already somewhat familiar with these concepts, feel free to skip to Section [1.4](#) where we'll introduce our first data set: all domestic flights departing a New York City airport in 2013. This is a dataset we will explore in depth in this book.

1.1 What are R and RStudio?

For much of this book, we will assume that you are using R via RStudio. First time users often confuse the two. At its simplest:

- R is like a car's engine.
- RStudio is like a car's dashboard.

R: Engine



RStudio: Dashboard



More precisely, R is a programming language that runs computations while RStudio is an *integrated development environment (IDE)* that provides an interface by adding many convenient features and tools. So just as having access to a speedometer, rearview mirrors, and a

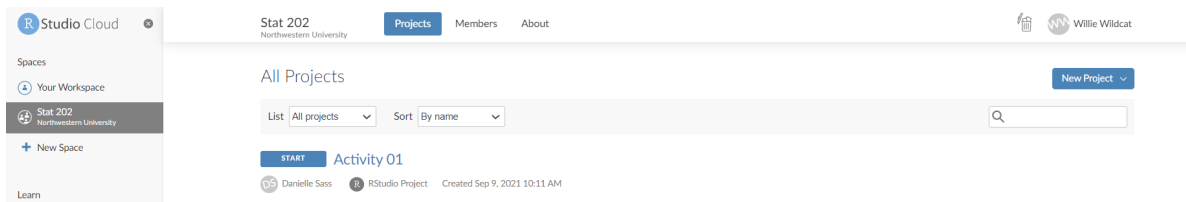
navigation system makes driving much easier, using RStudio's interface makes using R much easier as well.

1.1.1 Using RStudio Cloud

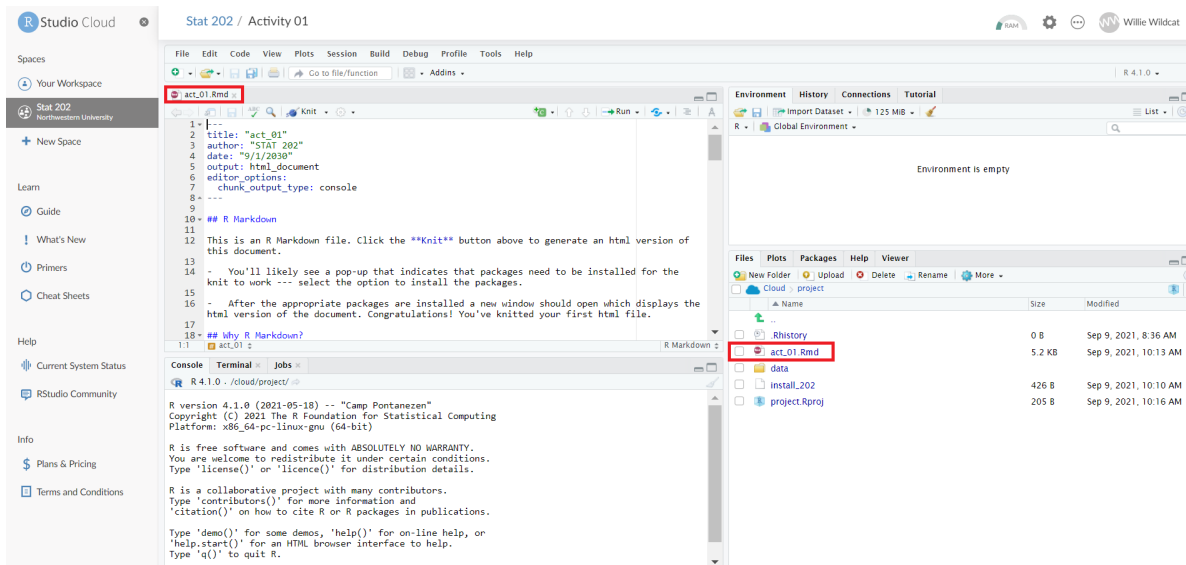
RStudio Cloud (<https://rstudio.cloud>) is a hosted version of RStudio that allows you to begin coding directly from your browser - there is no software to install and nothing to configure on your computer.

To begin using RStudio Cloud use the link provided by your instructor to gain access to the classroom workspace. You will be prompted to create a free account or log in if you have an existing account.

After you open RStudio Cloud, you should now have access to the classroom under 'Spaces' on the left hand side (in this case 'Stat 202').



Throughout class you will be working on various activities. Once the instructor has made an activity available you will click on the classroom Workspace (Stat 202) to access the available projects. To begin working on an activity click 'Start'. Once that activity project is open navigate to the 'File' pane and open the R Markdown '.Rmd' file.



You can use RStudio Cloud for personal use as well by creating projects in ‘Your Workspace’. However, RStudio Cloud limits the number of projects and amount of accessible time so it is recommended that you later install the software on your own computer.

1.1.2 Installing R and RStudio on your personal computer

Note about RStudio Server or RStudio Cloud: If your instructor has provided you with a link and access to RStudio Server or RStudio Cloud, then you can skip this section. We do recommend after a few months of working on RStudio Server/Cloud that you return to these instructions to install this software on your own computer though. You will first need to download and install both R and RStudio (Desktop version) on your computer. It is important that you install R first and then install RStudio second.

1. **You must do this first:** [Download and install R](#).



- If you are a Windows user: Click on “Download R for Windows”, then click on “base”, then click on the Download link.
- If you are macOS user: Click on “Download R for (Mac) OS X”, then under “Latest release:” click on R-X.X.X.pkg, where R-X.X.X is the version number. For example, the latest version of R as of August 10, 2019 was R-3.6.1.

2. **You must do this second:** [Download and install RStudio](#).

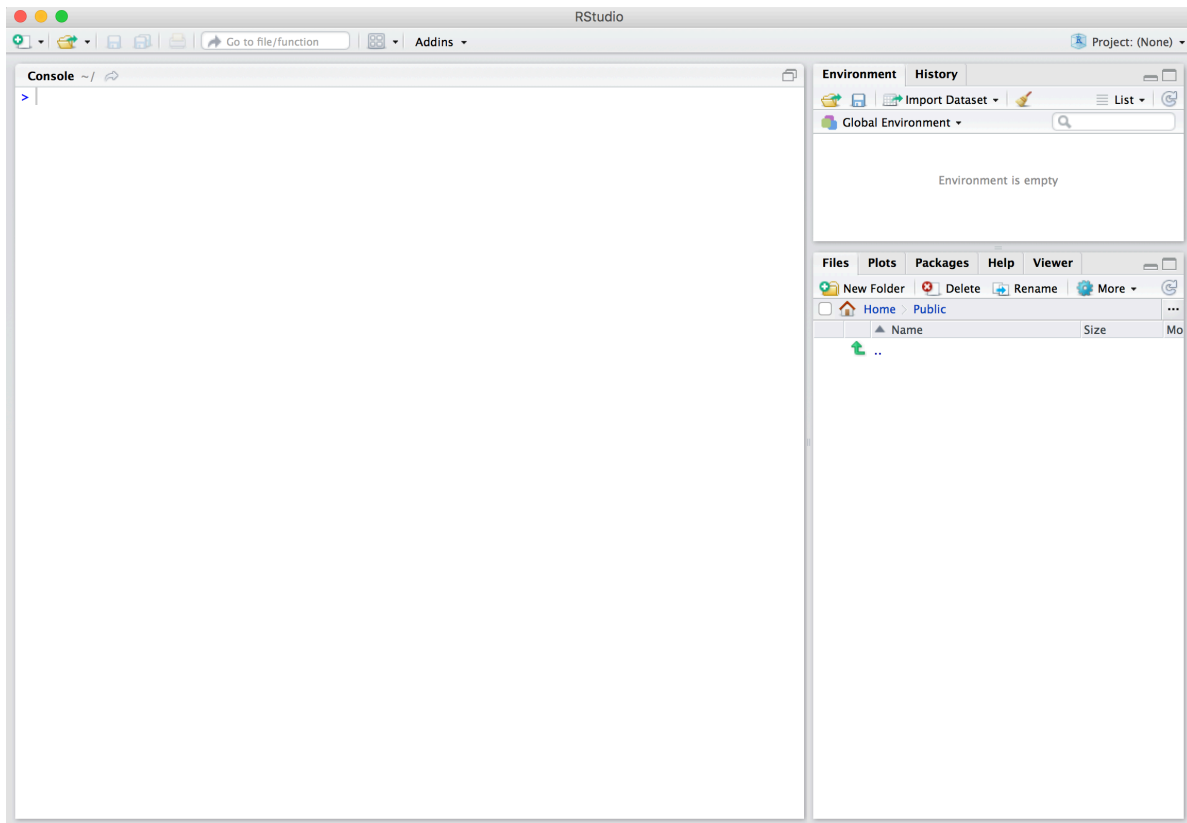
- Scroll down to “Installers for Supported Platforms” near the bottom of the page.
- Click on the download link corresponding to your computer’s operating system.

1.1.3 Using R via RStudio

Recall our car analogy from above. Much as we don’t drive a car by interacting directly with the engine but rather by interacting with elements on the car’s dashboard, we won’t be using R directly but rather we will use RStudio’s interface. After you install R and RStudio on your computer, you’ll have two new programs AKA applications you can open. We will always work in RStudio and not R. In other words:

R: Do not open this	RStudio: Open this
	

After you open RStudio, you should see the following:



Note the three panes, which are three panels dividing the screen: The *Console pane*, the *Files pane*, and the *Environment pane*. Over the course of this chapter, you’ll come to learn what purpose each of these panes serve.

1.2 How do I code in R?

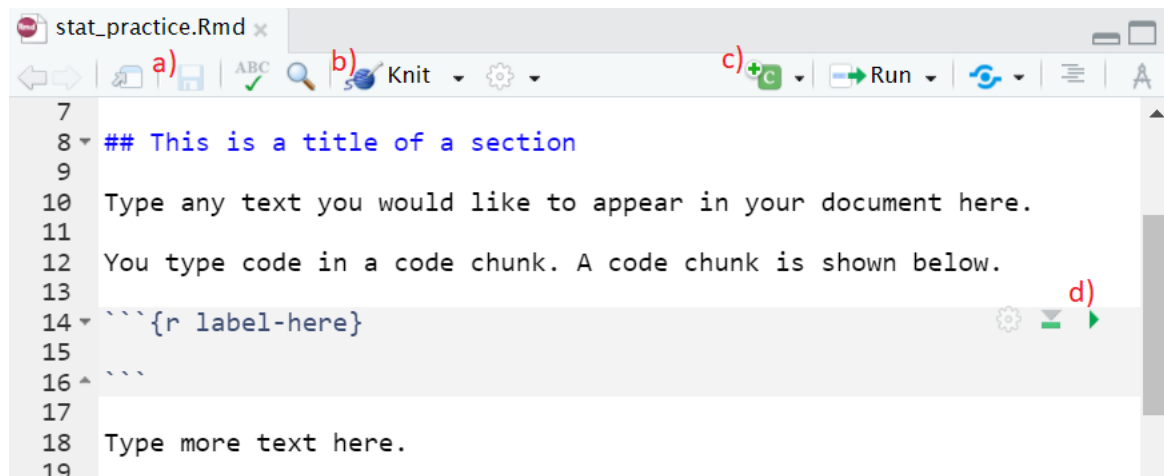
Now that you’re set up with R and RStudio, you are probably asking yourself “OK. Now how do I use R?” The first thing to note is that unlike other statistical software programs like Excel, STATA, or SAS that provide [point and click](#) interfaces, R is an [interpreted language](#), meaning you have to enter in R commands written in R code. In other words, you have to code/program in R. Note that we’ll use the terms “coding” and “programming” interchangeably in this book.

While it is not required to be a seasoned coder/computer programmer to use R, there is still a set of basic programming concepts that R users need to understand. Consequently, while this book is not a book on programming, you will still learn just enough of these basic programming concepts needed to explore and analyze data effectively.

1.2.1 Creating your first R Markdown document

R Markdown allows you to easily create a document which combines your code, the results from your code, as well as any text that accompanies the analysis. To create a new R Markdown file, in R Studio select **File>New File>R Markdown**. Then, you will see a window pop-up titled *New R Markdown*. Here, you specify the type of file you wish to create. HTML is generally the recommended document type since it does not have traditional *page* separators like PDF and Word do. You can also choose a title and author for your document using their respective fields. Finally, select *Ok* to create your new R Markdown file. You will see it appear as a tab in your R Studio session. Click the *save icon* to save your new document.

The following is an example of an R Markdown document:



- a) Save your document.
- b) Click *knit* to compile your R Markdown into the document file type that you specified. The file will be saved in your *Files pane*. This will also save your document.
- c) Insert a new code chunk in your document where the cursor is located. You will often have many code chunks in your document.
- d) Run the current code chunk.

When you create your Markdown file and *knit* it into a document, the chunks are run in order and any output from them is shown in the document, in the order and location that their respective chunk appears. Sometimes you may wish to type code or analyze data without it printing in the document. If that is the case, you type the code in the *Console* rather than in the *.Rmd* file.

While you read through this book, it will be helpful to have an RMarkdown document open so you can copy code provided and paste it into a code chunk to run.

1.2.2 Basic programming concepts and terminology

We now introduce some basic programming concepts and terminology. Instead of asking you to learn all these concepts and terminology right now, we'll guide you so that you'll "learn by doing." Note that in this book we will always use a different font to distinguish regular text from `computer_code`. The best way to master these topics is, in our opinions, "learning by doing" and lots of repetition.

- **Basics:**
 - *Console:* Where you enter in commands.
 - *Running code:* The act of telling R to perform an action by giving it commands in the console.
 - *Objects:* Where values are saved in R. In order to do useful and interesting things in R, we will want to *assign* a name to an object. For example we could do the following assignments: `x <- 44 - 20` and `three <- 3`. This would allow us to run `x + three` which would return 27.
 - *Data types:* Integers, doubles/numerics, logicals, and characters.

In R Studio try typing the following code into the console or code chunk.

```
x <- 44-20
three <- 3
x+three
```

[1] 27

You should see `x` and `three` appear as stored objects in the *Environment* pane. Anything you store in the *Environment* pane can be referenced and used later. R can also be used as a calculator, notice how it evaluates `x+three`.

- *Vectors:* A series of values. These are created using the `c()` function, where `c()` stands for "combine" or "concatenate". For example: `c(6, 11, 13, 31, 90, 92)`.
- *Factors:* *Categorical data* are represented in R as factors.
- *Data frames:* Data frames are like rectangular spreadsheets: they are representations of datasets in R where the rows correspond to *observations* and the columns correspond to *variables* that describe the observations. We'll cover data frames later in Section 1.4.
- *Conditionals:*

- Testing for equality in R using `==` (and not `=` which is typically used for assignment). Ex: `2 + 1 == 3` compares `2 + 1` to `3` and is correct R code, while `2 + 1 = 3` will return an error.
- Boolean algebra: `TRUE/FALSE` statements and mathematical operators such as `<` (less than), `<=` (less than or equal), and `!=` (not equal to).
- Logical operators: `&` representing “and” as well as `|` representing “or.” Ex: `(2 + 1 == 3) & (2 + 1 == 4)` returns `FALSE` since both clauses are not `TRUE` (only the first clause is `TRUE`). On the other hand, `(2 + 1 == 3) | (2 + 1 == 4)` returns `TRUE` since at least one of the two clauses is `TRUE`.
- *Functions*, also called *commands*: Functions perform tasks in R. They take in inputs called *arguments* and return outputs. You can either manually specify a function’s arguments or use the function’s *default values*.

This list is by no means an exhaustive list of all the programming concepts and terminology needed to become a savvy R user; such a list would be so large it wouldn’t be very useful, especially for novices. Rather, we feel this is a minimally viable list of programming concepts and terminology you need to know before getting started. We feel that you can learn the rest as you go. Remember that your mastery of all of these concepts and terminology will build as you practice more and more.

1.2.3 Errors, warnings, and messages

One thing that intimidates new R and RStudio users is how it reports *errors*, *warnings*, and *messages*. R reports errors, warnings, and messages in a glaring red font, which makes it seem like it is scolding you. However, seeing red text in the console is not always bad.

R will show red text in the console pane in three different situations:

- **Errors:** When the red text is a legitimate error, it will be prefaced with “Error in...” and try to explain what went wrong. Generally when there’s an error, the code will not run. For example, we’ll see in Subsection 1.3.3 if you see `Error in ggplot(...): could not find function "ggplot"`, it means that the `ggplot()` function is not accessible because the package that contains the function (`ggplot2`) was not loaded with `library(ggplot2)`. Thus you cannot use the `ggplot()` function without the `ggplot2` package being loaded first.
- **Warnings:** When the red text is a warning, it will be prefaced with “Warning:” and R will try to explain why there’s a warning. Generally your code will still work, but with some caveats. For example, you will see in Chapter 2 if you create a scatterplot based on a dataset where one of the values is missing, you will see this warning: `Warning: Removed 1 rows containing missing values (geom_point)`. R will still produce the scatterplot with all the remaining values, but it is warning you that one of the points isn’t there.

- **Messages:** When the red text doesn't start with either "Error" or "Warning", it's *just a friendly message*. You'll see these messages when you load *R packages* in the upcoming Subsection 1.3.2 or when you read data saved in spreadsheet files with the `read_csv()` function as you'll see in Chapter 4. These are helpful diagnostic messages and they don't stop your code from working. Additionally, you'll see these messages when you install packages too using `install.packages()`.

Remember, when you see red text in the console, *don't panic*. It doesn't necessarily mean anything is wrong. Rather:

- If the text starts with "Error", figure out what's causing it. Think of errors as a red traffic light: something is wrong!
- If the text starts with "Warning", figure out if it's something to worry about. For instance, if you get a warning about missing values in a scatterplot and you know there are missing values, you're fine. If that's surprising, look at your data and see what's missing. Think of warnings as a yellow traffic light: everything is working fine, but watch out/pay attention.
- Otherwise the text is just a message. Read it, wave back at R, and thank it for talking to you. Think of messages as a green traffic light: everything is working fine.

1.2.4 Tips on learning to code

Learning to code/program is very much like learning a foreign language, it can be very daunting and frustrating at first. Such frustrations are very common and it is very normal to feel discouraged as you learn. However just as with learning a foreign language, if you put in the effort and are not afraid to make mistakes, anybody can learn.

Here are a few useful tips to keep in mind as you learn to program:




- **Remember that computers are not actually that smart:** You may think your computer or smartphone are "smart," but really people spent a lot of time and energy designing them to appear "smart." Rather you have to tell a computer everything it needs to do. Furthermore the instructions you give your computer can't have any mistakes in them, nor can they be ambiguous in any way.
- **Take the "copy, paste, and tweak" approach:** Especially when learning your first programming language, it is often much easier to taking existing code that you know works and modify it to suit your ends, rather than trying to write new code from scratch. We call this the *copy, paste, and tweak* approach. So early on, we suggest not trying to write code from memory, but rather take existing examples we have provided you, then copy, paste, and tweak them to suit your goals. Don't be afraid to play around!

- **The best way to learn to code is by doing:** Rather than learning to code for its own sake, we feel that learning to code goes much smoother when you have a goal in mind or when you are working on a particular project, like analyzing data that you are interested in.
- **Practice is key:** Just as the only method to improving your foreign language skills is through practice, practice, and practice; so also the only method to improving your coding is through practice, practice, and practice. Don't worry however; we'll give you plenty of opportunities to do so!

1.3 What are R packages?

Another point of confusion with many new R users is the idea of an R package. R packages extend the functionality of R by providing additional functions, data, and documentation. They are written by a world-wide community of R users and can be downloaded for free from the internet. For example, among the many packages we will use in this book are the `ggplot2` package for data visualization in Chapter 2, the `dplyr` package for data wrangling in Chapter 3, and the `moderndive` package that accompanies this book.

A good analogy for R packages is they are like apps you can download onto a mobile phone:

R: A new phone	R Packages: Apps you can download
	 

So R is like a new mobile phone: while it has a certain amount of features when you use it for the first time, it doesn't have everything. R packages are like the apps you can download onto your phone from Apple's App Store or Android's Google Play.

Let's continue this analogy by considering the Instagram app for editing and sharing pictures. Say you have purchased a new phone and you would like to share a recent photo you have taken on Instagram. You need to:

1. *Install the app:* Since your phone is new and does not include the Instagram app, you need to download the app from either the App Store or Google Play. You do this once

and you're set. You might do this again in the future any time there is an update to the app.

2. *Open the app*: After you've installed Instagram, you need to open the app.

Once Instagram is open on your phone, you can then proceed to share your photo with your friends and family. The process is very similar for using an R package. You need to:

1. *Install the package*: This is like installing an app on your phone. Most packages are not installed by default when you install R and RStudio. Thus if you want to use a package for the first time, you need to install it first. Once you've installed a package, you likely won't install it again unless you want to update it to a newer version.
2. *"Load" the package*: "Loading" a package is like opening an app on your phone. Packages are not "loaded" by default when you start RStudio on your computer; you need to "load" each package you want to use every time you start RStudio.

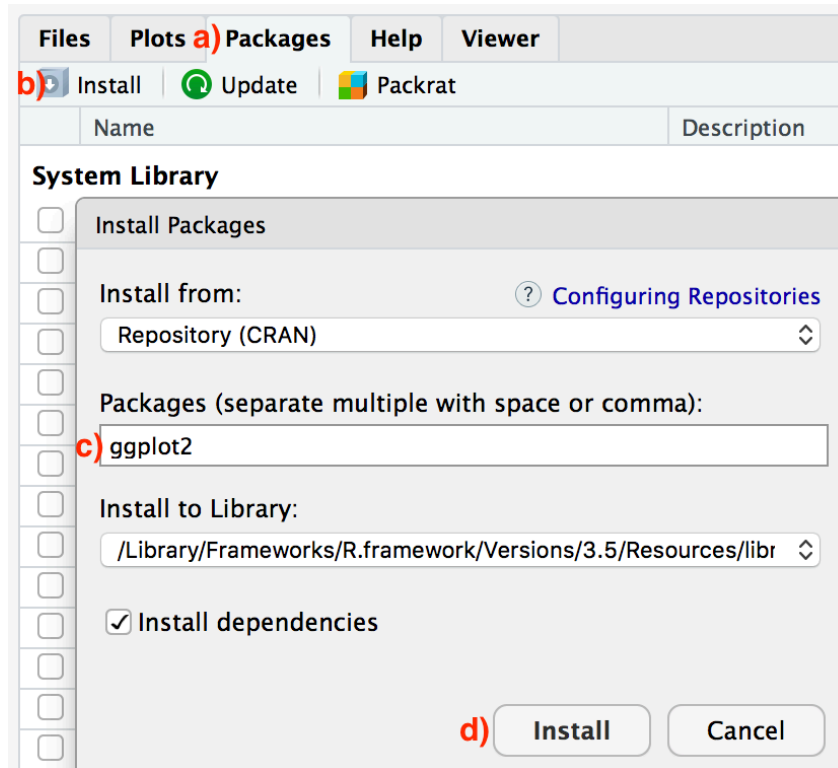
Let's now show you how to perform these two steps for the `ggplot2` package for data visualization.

1.3.1 Package installation

Note about RStudio Server: If your instructor has provided you with a link and access to RStudio Server, you probably will not need to install packages, as they have likely been pre-installed for you by your instructor. That being said, it is still a good idea to know this process for later on when you are not using RStudio Server, but rather RStudio Desktop on your own computer.

There are two ways to install an R package. For example, to install the `ggplot2` package:

1. **Easy way**: In the Files pane of RStudio:
 - a) Click on the "Packages" tab
 - b) Click on "Install"
 - c) Type the name of the package under "Packages (separate multiple with space or comma):" In this case, type `ggplot2`
 - d) Click "Install"



2. **Slightly harder way:** An alternative but slightly less convenient way to install a package is by typing `install.packages("ggplot2")` in the Console pane of RStudio and hitting enter. Note you must include the quotation marks.

Much like an app on your phone, you only have to install a package once. However, if you want to update an already installed package to a newer version, you need to re-install it by repeating the above steps.

Learning Check 1.1

Repeat the above installing steps for the `dplyr`, `nycflights13`, and `knitr` packages. This will install the earlier mentioned `dplyr` package, the `nycflights13` package containing data on all domestic flights leaving a NYC airport in 2013, and the `knitr` package for writing reports in R.

1.3.2 Package loading

Recall that after you've installed a package, you need to "load" it, in other words open it. We do this by using the `library()` command. For example, to load the `ggplot2` package, run the

following code in the Console pane. What do we mean by “run the following code”? Either type or copy & paste the following code into the Console pane and then hit the enter key.

```
library(ggplot2)
```

If after running the above code, a blinking cursor returns next to the > “prompt” sign, it means you were successful and the `ggplot2` package is now loaded and ready to use. If however, you get a red “error message” that reads...

```
Error in library(ggplot2) : there is no package called 'ggplot2'
```

... it means that you didn’t successfully install it. In that case, go back to the previous subsection “Package installation” and install it.

Learning Check 1.2

“Load” the `dplyr`, `nycflights13`, and `knitr` packages as well by repeating the above steps.

1.3.3 Package use

One extremely common mistake new R users make when wanting to use particular packages is that they forget to “load” them first by using the `library()` command we just saw. Remember: *you have to load each package you want to use every time you start RStudio*. If you don’t first “load” a package, but attempt to use one of its features, you’ll see an error message similar to:

```
Error: could not find function
```

R is telling you that you are trying to use a function in a package that has not yet been “loaded.” Almost all new users forget to do this when starting out, and it is a little annoying to get used to. However, you’ll remember with practice.

1.4 Explore your first dataset

Let’s put everything we’ve learned so far into practice and start exploring some real data! Data comes to us in a variety of formats, from pictures to text to numbers. Throughout this book, we’ll focus on datasets that are saved in “spreadsheet”-type format; this is probably the most common way data are collected and saved in many fields. Remember from Subsection [1.2.2](#)

that these “spreadsheet”-type datasets are called *data frames* in R; we will focus on working with data saved as data frames throughout this book.

Let’s first load all the packages needed for this chapter, assuming you’ve already installed them. Read Section 1.3 for information on how to install and load R packages if you haven’t already.

```
library(nycflights13)
library(dplyr)
library(knitr)
```

At the beginning of all subsequent chapters in this text, we’ll always have a list of packages that you should have installed and loaded to work with that chapter’s R code.

1.4.1 nycflights13 package

Many of us have flown on airplanes or know someone who has. Air travel has become an ever-present aspect in many people’s lives. If you live in or are visiting a relatively large city and you walk around that city’s airport, you see gates showing flight information from many different airlines. And you will frequently see that some flights are delayed because of a variety of conditions. Are there ways that we can avoid having to deal with these flight delays?

We’d all like to arrive at our destinations on time whenever possible. (Unless you secretly love hanging out at airports. If you are one of these people, pretend for the moment that you are very much anticipating being at your final destination.) Throughout this book, we’re going to analyze data related to flights contained in the **nycflights13** package (Wickham 2021). Specifically, this package contains five data sets saved in five separate data frames with information about all domestic flights departing from New York City in 2013. These include Newark Liberty International (EWR), John F. Kennedy International (JFK), and LaGuardia (LGA) airports:

- **flights**: Information on all 336,776 flights
- **airlines**: A table matching airline names and their two letter IATA airline codes (also known as carrier codes) for 16 airline companies
- **planes**: Information about each of 3,322 physical aircraft used.
- **weather**: Hourly meteorological data for each of the three NYC airports. This data frame has 26,115 rows, roughly corresponding to the $365 \times 24 \times 3 = 26,280$ possible hourly measurements one can observe at three locations over the course of a year.
- **airports**: Airport names, codes, and locations for 1,458 destination airports.

1.4.2 flights data frame

We will begin by exploring the `flights` data frame that is included in the `nycflights13` package and getting an idea of its structure. Run the following code in your console (either by typing it or cutting & pasting it): it loads in the `flights` dataset into your Console. Note depending on the size of your monitor, the output may vary slightly.

```
flights

# A tibble: 336,776 x 19
   year month   day dep_time sched_de~1 dep_d~2 arr_t~3 sched~4 arr_d~5 carrier
   <int> <int> <int>   <int>      <int>    <dbl>   <int>   <int>    <dbl> <chr>
1  2013     1     1     517        515         2     830     819        11 UA
2  2013     1     1     533        529         4     850     830        20 UA
3  2013     1     1     542        540         2     923     850        33 AA
4  2013     1     1     544        545        -1    1004    1022       -18 B6
5  2013     1     1     554        600        -6     812     837       -25 DL
6  2013     1     1     554        558        -4     740     728        12 UA
7  2013     1     1     555        600        -5     913     854        19 B6
8  2013     1     1     557        600        -3     709     723       -14 EV
9  2013     1     1     557        600        -3     838     846        -8 B6
10 2013     1     1     558        600        -2     753     745         8 AA
# ... with 336,766 more rows, 9 more variables: flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dtm>, and abbreviated variable names
#   1: sched_dep_time, 2: dep_delay, 3: arr_time, 4: sched_arr_time,
#   5: arr_delay
```

Let's unpack this output:

- A tibble: 336,776 x 19: A tibble is a kind of data frame used in R. This particular data frame has
 - 336,776 rows
 - 19 columns corresponding to 19 variables describing each observation
- `year month day dep_time sched_dep_time dep_delay arr_time` are different columns, in other words variables, of this data frame.
- We then have the first 10 rows of observations corresponding to 10 flights.
- ... with 336,766 more rows, and 11 more variables: indicating to us that 336,766 more rows of data and 11 more variables could not fit in this screen.

Unfortunately, this output does not allow us to explore the data very well. Let's look at different tools to explore data frames.

1.4.3 Exploring data frames

Among the many ways of getting a feel for the data contained in a data frame such as `flights`, we present three functions that take as their “argument”, in other words their input, the data frame in question. We also include a fourth method for exploring one particular column of a data frame:

1. Using the `View()` function built for use in RStudio. We will use this the most.
2. Using the `glimpse()` function, which is included in the `dplyr` package.
3. Using the `kable()` function, which is included in the `knitr` package.
4. Using the `$` operator to view a single variable in a data frame.

1. `View()`:

Run `View(flights)` in your Console in RStudio, either by typing it or cutting & pasting it into the Console pane, and explore this data frame in the resulting pop-up viewer. You should get into the habit of always **Viewing** any data frames that come your way. Note the capital “V” in `View`. R is case-sensitive so you'll receive an error if you run `view(flights)` instead of `View(flights)`.

Learning Check 1.3

What does any *ONE* row in this `flights` dataset refer to?

- a. Data on an airline
- b. Data on a flight
- c. Data on an airport
- d. Data on multiple flights

By running `View(flights)`, we see the different *variables* listed in the columns and we see that there are different types of variables. Some of the variables like `distance`, `day`, and `arr_delay` are what we will call *quantitative* variables. These variables are numerical in nature. Other variables here are *categorical*.

Note that if you look in the leftmost column of the `View(flights)` output, you will see a column of numbers. These are the row numbers of the dataset. If you glance across a row with the same number, say row 5, you can get an idea of what each row corresponds to. In other words, this will allow you to identify what object is being referred to in a given row. This is often called the *observational unit*. The *observational unit* in this example is an individual flight departing New York City in 2013. You can identify the observational unit by determining what “thing” is being measured or described by each of the variables.

2. `glimpse()`:

The second way to explore a data frame is using the `glimpse()` function included in the `dplyr` package. Thus, you can only use the `glimpse()` function after you’ve loaded the `dplyr` package. This function provides us with an alternative method for exploring a data frame:

```
glimpse(flights)
```

```
Rows: 336,776
```

```
Columns: 19
```

```
$ year      <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2~
$ month     <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ day       <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ dep_time  <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558, ~
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600, ~
$ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -1~
$ arr_time  <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849,~
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851,~
$ arr_delay <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -1~
$ carrier   <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", "~
$ flight    <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, 4~
$ tailnum   <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N394~
$ origin    <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA",~
$ dest      <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD",~
$ air_time  <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, 1~
$ distance  <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733, ~
$ hour      <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6~
$ minute    <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 0~
$ time_hour <dtm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 0~
```

We see that `glimpse()` will give you the first few entries of each variable in a row after the variable. In addition, the *data type* (see Subsection 1.2.2) of the variable is given immediately after each variable’s name inside `< >`. Here, `int` and `dbl` refer to “integer” and “double”, which are computer coding terminology for quantitative/numerical variables. In contrast, `chr` refers to “character”, which is computer terminology for text data. Text data, such as the `carrier` or `origin` of a flight, are categorical variables. The `time_hour` variable is an example of one more type of data type: `dtm`. As you may suspect, this variable corresponds to a specific date and time of day. However, we won’t work with dates in this class and leave it to a more advanced book on data science.

Learning Check 1.4

What are some examples in this dataset of **categorical** variables? What makes them different than **quantitative** variables?

3. `kable()`:

Another way to explore the entirety of a data frame is using the `kable()` function from the `knitr` package. Let's explore the different carrier codes for all the airlines in our dataset two ways. Run both of these lines of code in your Console:

```
airlines
kable(airlines)
```

At first glance, it may not appear that there is much difference in the outputs. However when using tools for document production such as [R Markdown](#), the latter code produces output that is much more legible and reader-friendly.

4. `$` operator

Lastly, the `$` operator allows us to explore a single variable within a data frame. For example, run the following in your console

```
airlines
airlines$name
```

We used the `$` operator to extract only the `name` variable and return it as a vector of length 16. We will only be occasionally exploring data frames using this operator, instead favoring the `View()` and `glimpse()` functions.

1.4.4 Help files

Another nice feature of R is the help system. You can get help in R by entering a `?` before the name of a function or data frame in question and you will be presented with a page showing the documentation. For example, let's look at the help file for the `flights` data frame:

```
?flights
```

A help file should pop-up in the Help pane of RStudio. If you have questions about a function or data frame included in an R package, you should get in the habit of consulting the help file right away.

1.5 Conclusion

We've given you what we feel are the most essential concepts to know before you can start exploring data in R. Is this chapter exhaustive? Absolutely not. To try to include everything in this chapter would make the chapter so large it wouldn't be useful!

1.5.1 Additional resources

If you are completely new to the world of coding, R, and RStudio and feel you could benefit from a more detailed introduction, we suggest you check out Chester Ismay's short book [Getting used to R, RStudio, and R Markdown](#) (Ismay 2016), which includes screencast recordings that you can follow along and pause as you learn. Furthermore, there is an introduction to R Markdown, a tool used for reproducible research in R.

1 Introduction

2 Why R?

3 R and RStudio Basics

4 R Markdown

5 Intro to R using R Markdown

6 Deciphering Common R Errors

7 Concluding Remarks

8 References

Published with bookdown

Create a GitHub Issue

Email Chester

Getting used to R, RStudio, and R Markdown

Chester Ismay

Patrick C. Kennedy

2018-05-23

1 Introduction

This book was written to give people who are new to [R](#), [RStudio](#), and [R Markdown](#) the tools they need to begin making their own research reproducible. R is an open-source programming language that has seen its popularity grow tremendously in recent years, with developers adding new functionality via packages on a daily basis. RStudio is a graphical development environment that makes it easier to write and view the results of R code, and R Markdown provides an easy way to produce rich, fully-documented, reproducible analyses.

Part II

Data Exploration via the tidyverse

2 Data Visualization

We begin the development of your data science toolbox with data visualization. By visualizing our data, we gain valuable insights that we couldn't initially see from just looking at the raw data in spreadsheet form. We will use the `ggplot2` package as it provides an easy way to customize your plots. `ggplot2` is rooted in the data visualization theory known as *The Grammar of Graphics* (Wilkinson 2005).

At the most basic level, graphics/plots/charts (we use these terms interchangeably in this book) provide a nice way for us to get a sense for how quantitative variables compare in terms of their center (where the values tend to be located) and their spread (how they vary around the center). Graphics should be designed to emphasize the findings and insight you want your audience to understand. This does however require a balancing act. On the one hand, you want to highlight as many meaningful relationships and interesting findings as possible; on the other you don't want to include so many as to overwhelm your audience.

As we will see, plots/graphics also help us to identify patterns and outliers in our data. We will see that a common extension of these ideas is to compare the *distribution* of one quantitative variable (i.e., what the spread of a variable looks like or how the variable is *distributed* in terms of its values) as we go across the levels of a different categorical variable.

Packages Needed

Let's load all the packages needed for this chapter (this assumes you've already installed them). Read Section 1.3 for information on how to install and load R packages.

```
library(nycflights13)
library(ggplot2)
library(dplyr)
```

2.1 The Grammar of Graphics

We begin with a discussion of a theoretical framework for data visualization known as “The Grammar of Graphics,” which serves as the foundation for the `ggplot2` package. Think of how we construct sentences in English to form sentences by combining different elements,

like nouns, verbs, particles, subjects, objects, etc. However, we can't just combine these elements in any arbitrary order; we must do so following a set of rules known as a linguistic grammar. Similarly to a linguistic grammar, "The Grammar of Graphics" define a set of rules for constructing *statistical graphics* by combining different types of *layers*. This grammar was created by Leland Wilkinson (Wilkinson 2005) and has been implemented in a variety of data visualization software including R.

2.1.1 Components of the Grammar

In short, the grammar tells us that:

A statistical graphic is a mapping of data variables to aesthetic attributes of geometric objects.

Specifically, we can break a graphic into three essential components:

1. **data**: the data set composed of variables that we map.
2. **geom**: the geometric object in question. This refers to the type of object we can observe in a plot. For example: points, lines, and bars.
3. **aes**: aesthetic attributes of the geometric object. For example, x-position, y-position, color, shape, and size. Each assigned aesthetic attribute can be mapped to a variable in our data set.

You might be wondering why we wrote the terms **data**, **geom**, and **aes** in a computer code type font. We'll see very shortly that we'll specify the elements of the grammar in R using these terms. However, let's first break down the grammar with an example.

2.1.2 Gapminder data

In February 2006, a statistician named Hans Rosling gave a TED talk titled "[The best stats you've ever seen](#)" where he presented global economic, health, and development data from the website [gapminder.org](#). For example, for the 142 countries included from 2007, let's consider only the first 6 countries when listed alphabetically in `?@tbl-gapminder-2007`.

3 Data Wrangling

 Under Construction

Currently working on content transfer from previous version of the book.

4 Data Importing & “Tidy Data”

 Under Construction

Currently working on content transfer from previous version of the book.

Part III

Data Modeling

5 Basic Regression

 Under Construction

Currently working on content transfer from previous version of the book.

6 Multiple Regression

 Under Construction

Currently working on content transfer from previous version of the book.

Part IV

Statistical Theory

7 Randomization and Causality

 Under Construction

Currently working on content transfer from previous version of the book.

8 Populations and Generalizability

 Under Construction

Currently working on content transfer from previous version of the book.

9 Sampling Distributions

 Under Construction

Currently working on content transfer from previous version of the book.

Part V

Statistical Inference

10 Confidence Intervals

 Under Construction

Currently working on content transfer from previous version of the book.

11 P-values

 Under Construction

Currently working on content transfer from previous version of the book.

12 Hypothesis tests

 Under Construction

Currently working on content transfer from previous version of the book.

13 Putting it all together

 Under Construction

Currently working on content transfer from previous version of the book.

References

- Ismay, Chester. 2016. *Getting Used to r, RStudio, and r Markdown*. <http://ismayc.github.io/rbasics-book>.
- Wickham, Hadley. 2021. *Nycflights13: Flights That Departed NYC in 2013*. <https://github.com/hadley/nycflights13>.
- Wilkinson, Leland. 2005. *The Grammar of Graphics (Statistics and Computing)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

A Statistical Background

 Under Construction

Currently working on content transfer from previous version of the book.