Research Transparency, Reproducibility, and Basic Data Analysis in R!

Shaon Lahiri, PhD, MPH

2023-08-30

Table of contents

Preface **7**

1 Installing R and RStudio **9**

2 Getting Accustomed to R Studio **11**

3 Introduction to R **19**
3.1 What exactly is R, and why is it called *R* and not something more informative? 19
3.2 Using R as a calculator . . . 22
3.3 Comments . . . 25
3.4 Functions . . . 25
3.5 R Packages . . . 26
3.6 Assignments . . . 30
3.7 Help . . . 32
3.8 Exercises . . . 33

4 Data Types **34**
4.1 Vectors . . . 35
4.1.1 Types . . . 35
4.1.2 Creating Vectors . . . 36
4.1.3 Extracting Elements from a Vector . . . 37
4.1.4 Vector Operations . . . 39
4.2 Matrices . . . 43
4.3 Dataframes . . . 46
4.4 Lists . . . 51
4.5 Arrays . . . 54
4.6 Factors . . . 56
4.7 Exercises . . . 59

5 Strings & Comparisons **61**
5.1 Strings . . . 61
5.2 Comparisons . . . 66
5.2.1 Comparison Operators . . . 66
5.2.2 Comparisons in Conditional Statements . . . 68
5.3 Exercises . . . 72

6 Importing Data **74**
6.1 Importing a CSV or Excel file . . . 74
6.2 Importing a text file . . . 76
6.3 Importing a Stata, SAS, or SPSS file . . . 78

7 Merging (Joining) and Reshaping Data **81**
7.1 Left and Right Joins . . . 81
7.2 Inner and Full Joins . . . 85
7.3 Two Common Merging Scenarios . . . 86
7.4 Reshaping Data . . . 88
7.4.1 From long to wide . . . 89
7.4.2 From wide to long . . . 91
7.5 Exercises . . . 93

8 Data Cleaning **95**
8.1 Basic Folder Structure . . . 95
8.2 The Pipe Operator (%>%) . . . 97
8.3 Illogical Values & Typos . . . 102
8.4 Duplicate Values & Outliers . . . 103
8.5 Detecting Illogical or Extreme Values . . . 104
8.6 Variable Subsetting, Naming, Transformation & Creation . . . 108
8.6.1 Subsetting . . . 108
8.6.2 (Re)Naming . . . 111
8.6.3 Transformation & Creation . . . 113
8.7 Saving Your Cleaned Dataframe . . . 118
8.7.1 As an R Data File . . . 118
8.7.2 As an Excel or CSV File . . . 118
8.7.3 As a Stata, SPSS, or SAS File . . . 118
8.7.4 As a Text File . . . 119
8.8 Exercises . . . 120

9 Data Exploration with `dplyr` **123**
9.1 Filter() & Arrange() . . . 123
9.2 Mutate() & Rename() . . . 125
9.3 Select() & Slice() . . . 126
9.4 Relocate() and Summarise() . . . 129
9.5 Exercises . . . 133

10 Data Visualization with base R and `ggplot2` **135**
10.1 Visualizations in Base R . . . 136
10.1.1 Bar graphs . . . 136
10.1.2 Histograms . . . 143
10.1.3 Scatterplot . . . 146

10.2 Visualizations in `ggplot2` . . . 150
10.2.1 Bar Graphs . . . 150
10.2.2 Histograms . . . 158
10.2.3 Sturge's Rule . . . 160
10.2.4 Freedman-Diaconis Rule . . . 160
10.2.5 Rice's Rule . . . 160
10.2.6 Scatterplots . . . 164
10.2.7 Violin Plots . . . 170

11 A Very Brief History of Replication **175**
11.1 What is Research Transparency, Reproducibility, and Replication? . . . 175
11.2 Replication Origins . . . 176
11.2.1 Ibn al Haytham . . . 176
11.2.2 Abu Rayhan Al Biruni . . . 178
11.2.3 Roger Bacon . . . 178
11.2.4 Andries van Wesel . . . 181
11.2.5 Francis Bacon . . . 181

12 Mertonian Norms and Counter-Norms **185**
12.1 Background . . . 186
12.2 CUDOS . . . 187
12.2.1 Communalism . . . 187
12.2.2 Universalism . . . 188
12.2.3 Disinterestedness . . . 190
12.2.4 Organized Skepticism . . . 191
12.3 Counter-Norms . . . 191
12.3.1 Particularism (in contrast with Universalism) . . . 191
12.3.2 Solitariness (in contrast with Communalism) . . . 194
12.3.3 Interestedness (in contrast with Disinterestedness) . . . 194
12.3.4 Organized Dogmatism (in contrast with Organized Skepticism) . . . 195
12.4 Conclusion . . . 196

13 Some Statistical Preliminaries **197**
13.1 Null Hypothesis Significance Testing . . . 198
13.2 P-Value . . . 200
13.3 Power . . . 201
13.4 Confidence Intervals . . . 202

14 Crises of Replication **203**
14.1 Reproducibility and Replication Definitions . . . 203
14.2 Stirrings of a Crisis . . . 203
14.3 Feeling the Future . . . 205
14.3.1 An Unintentional Crisis Unfolds . . . 205

14.3.2 Where Bem Went Wrong . . . 208
14.4 The Domino Effect . . . 209
14.5 Gauging the Extent of the Damage . . . 214
14.5.1 The Open Science Collaboration . . . 214
14.5.2 Questionable Research Practices in Psychology . . . 214
14.5.3 Many Labs 2 . . . 217
14.5.4 Replicating Social Science Experiments from *Nature* and *Science* . . . 217
14.6 Retractions . . . 217
14.6.1 What is it, and Why Does it Happen? . . . 217
14.6.2 Life After Death: Continued Citations Despite Retraction . . . 219

15 Common Problems that Hamper Reproducibility 228
15.1 Summary of Some Common Problems . . . 228
15.2 HARKing . . . 228
15.3 Publication Bias & the File Drawer Problem . . . 233
15.4 Cognitive Biases . . . 236

16 Potential Solutions & the Way Forward 240
16.1 A Partial List . . . 240
16.2 Pre-registration . . . 241
16.2.1 General Description . . . 241
16.2.2 Common Apprehensions . . . 243
16.2.3 What Should Be Included & Where to Pre-register . . . 244
16.3 Improved Methodological Training . . . 245
16.4 Incentives . . . 248
16.5 Funders Hold Tremendous Power . . . 249
16.6 Reporting Guidelines . . . 253
16.7 Preprints and Registered Reports . . . 254
16.8 Conclusion . . . 254

17 References 256

Appendices 262

A Answers for Section 3.8 262

B Answers for Section 4.7 264

C Answers for Section 5.3 267

D Answers for Section 7.5 271

E Answers for Section 8.8 276

F Answers for Section 9.5 **282**

G Kahneman's Open Letter to Priming Researchers **288**
G.1 Background 288
G.2 The Letter in Full 288

Preface

I started writing this book in the summer of 2023 primarily for my students in *PPE 4000 - Research in PPE: Research Transparency, Reproducibility and Basic Data Analysis in R.* This is a small research course for advanced undergraduates in the Philosophy, Politics & Economics (PPE) Program at the University of Pennsylvania. I am currently a Postdoctoral Researcher in the PPE Program. I'm also a BITSS Catalyst, which means that I am dedicated to educating the next generation of social scientists on research transparency tools and practices.

Beyond my students, I believe that the book can be useful for just about anyone interested in learning basic data analysis in R, especially if you have never learned any coding before. You may be wondering what separates this course from other introductory R courses or books that are freely available in the public domain? While there are MANY free resources to learn R, and MANY free resources to understand issues of Research Transparency & Reproducibility (RT2), I try to bridge the two ideas in this book at an undergraduate level. I have not seen a similar book, and thus it may be useful.

The prime motivation for doing so is that combining them in this way makes understanding RT2 concepts more concrete by implementing them directly in R. Additionally, by learning some basic R skills, students can walk away not only with an understanding of RT2 concepts, but also with a marketable skill for industry and academia. I never liked separating RT2 concepts and practices from actual quantitative work, and for that reason I have attempted to bridge the two (somewhat) cohesively in this semester-long course. Eventually the bridge will be more evident in the book (at least that's my hope).

I wrote this book using Quarto Book in R, and created the illustrations using Canva and DALL-E 2. I intend to keep updating this book and adding new content when I have time. If you have any feedback on this book, please email me at shaonl@sas.upenn.edu.

1 Installing R and RStudio

Let's make sure we understand what R and RStudio are before proceeding. Firstly, R is a programming language that is commonly used for statistical analysis and data visualization. RStudio is an Integrated Development Environment (IDE) which sits on top of R. It provides a very nice graphical user interface that allows us to use R along with many other useful features.

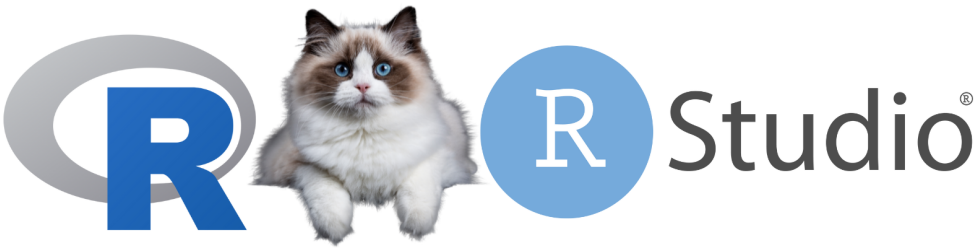
R is the engine, and RStudio is the car frame. You can use R without RStudio (known as Base R), but you cannot use RStudio without R.

Before we meet for the first class, ensure you have downloaded and installed the following:

1. R
    - Download from here: https://cran.r-project.org
2. RStudio Desktop
    - Download from here: https://posit.co/download/rstudio-desktop/

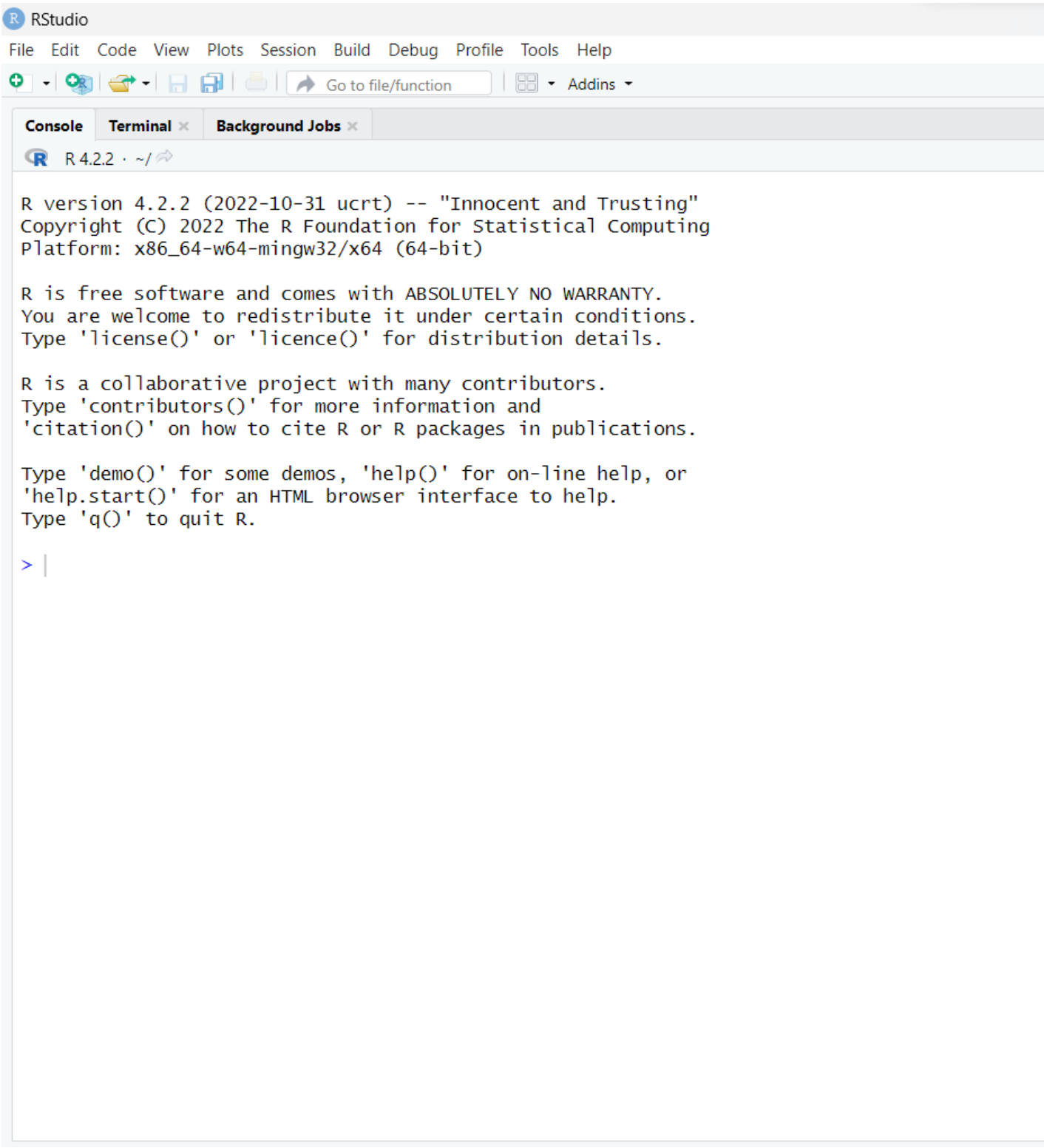
If you're using a Mac and you're having trouble, try switching browsers from Safari to Chrome. Sometimes Safari has some extra security provisions which prevent you from installing software successfully, and this can be remedied by using a different browser like Chrome.

Please email me if you have any trouble installing R and R Studio. If you've successfully installed both, head over to Chapter 2 .


R
Studio®

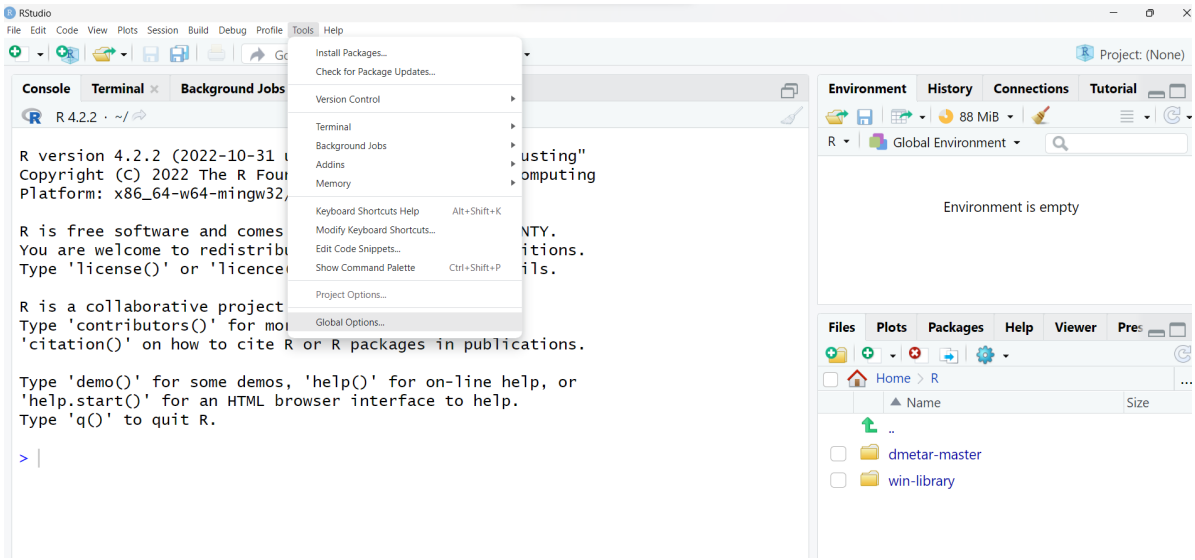
2 Getting Accustomed to R Studio

Once you've successfully installed R and RStudio, go ahead and open RStudio. It should look something like this:

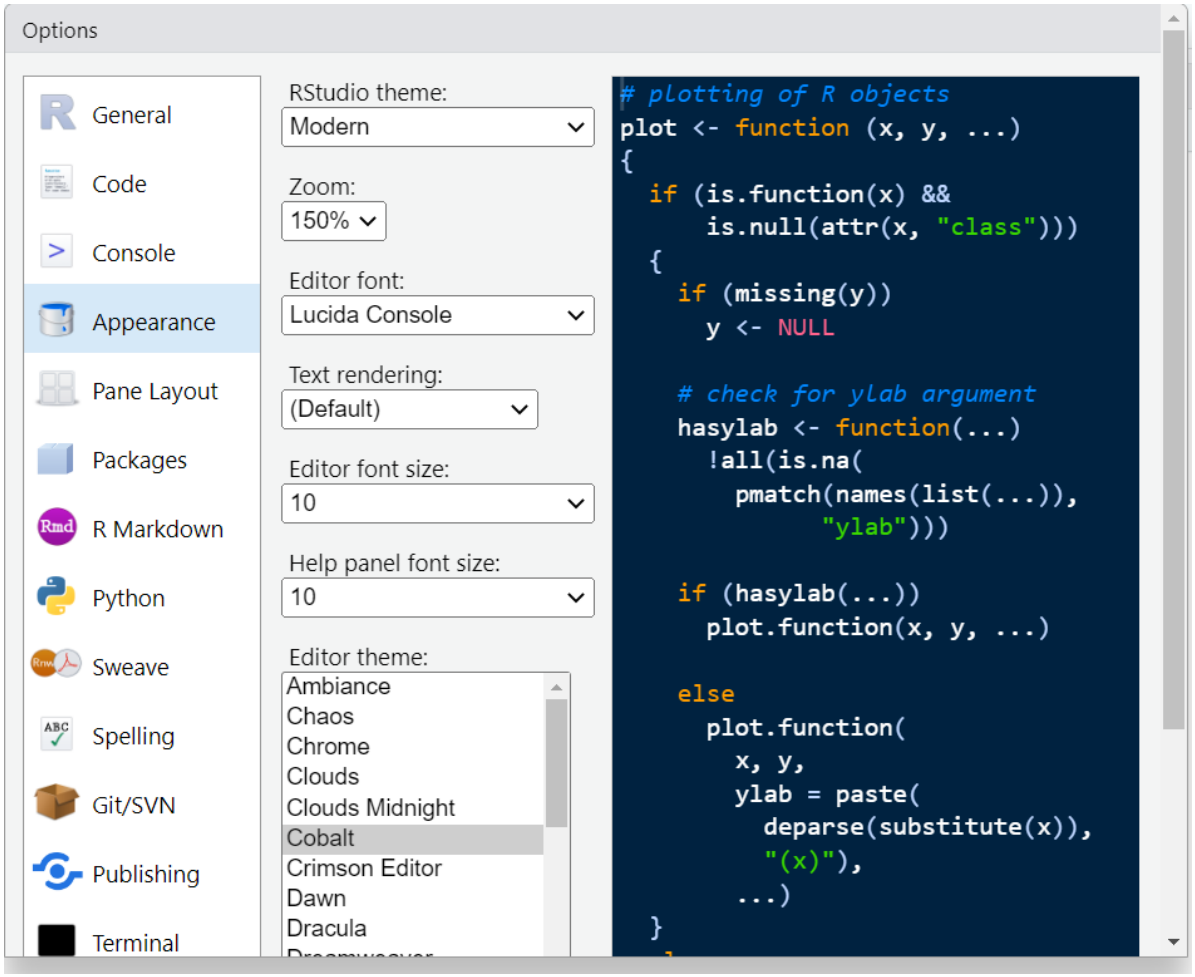

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console
Terminal
Background Jobs
R 4.2.2 · ~/
R version 4.2.2 (2022-10-31 ucrt) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>

Looks pretty plain right? No razzle dazzle?

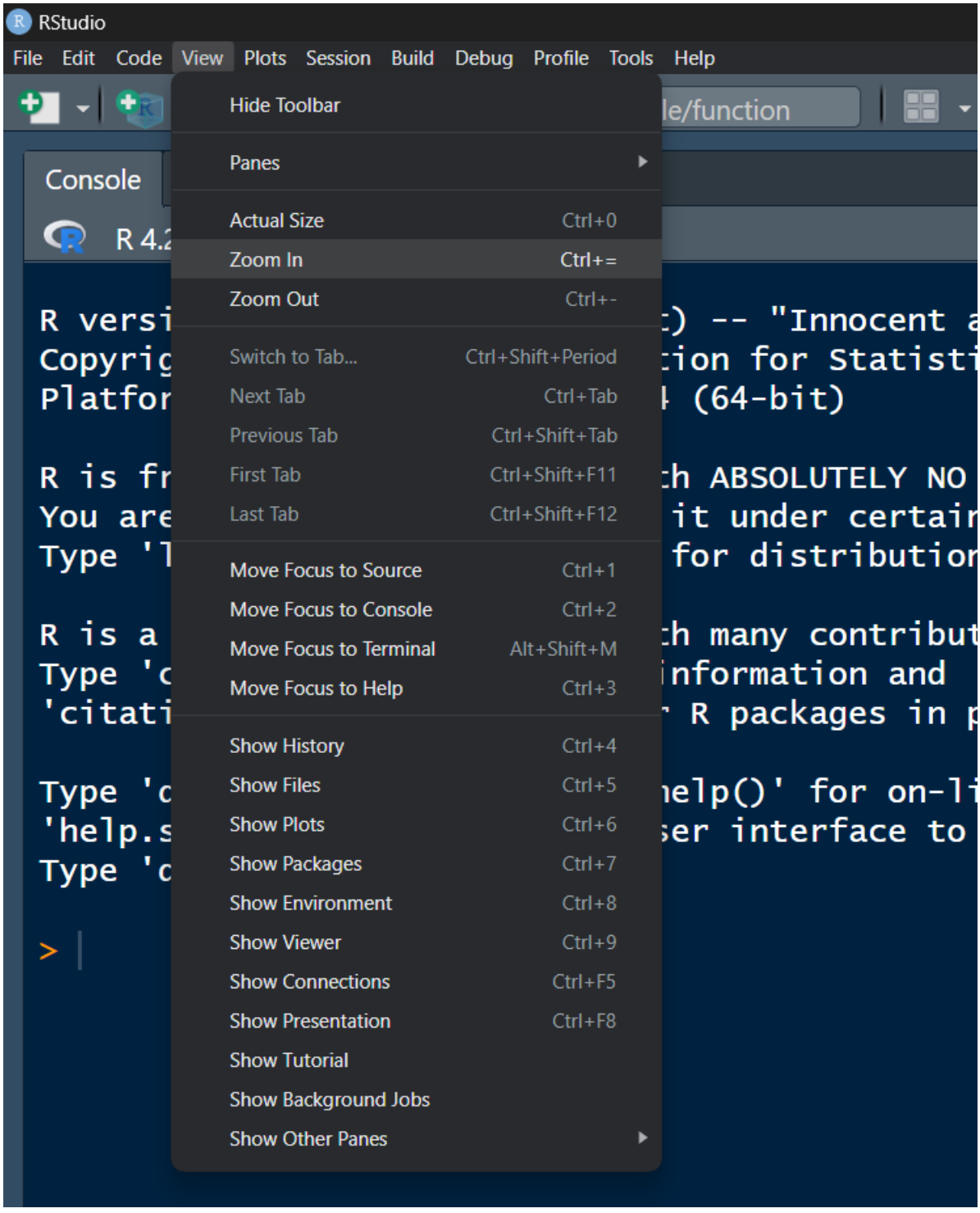
Before we do anything, let's first customize the way RStudio looks by going to Tools –> Global Options:



Next, go to Appearance and select an Editor Theme (personally I prefer dark themes like Cobalt). You can also modify the Editor font size in case you want to zoom in on the code. Click Apply to try out different themes, and Ok once you've settled on a theme.

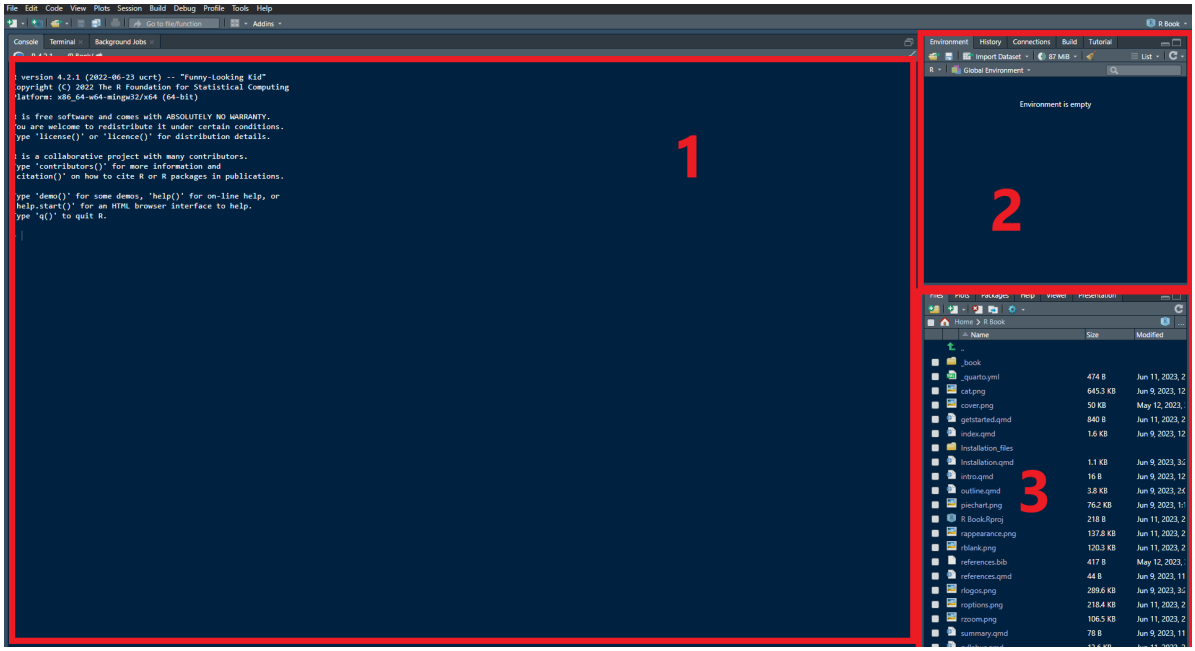


You can change the Editor font size to something larger if you have trouble seeing the code at size 10. You can also do this anytime in RStudio by going to View and then clicking Zoom In or Zoom Out:



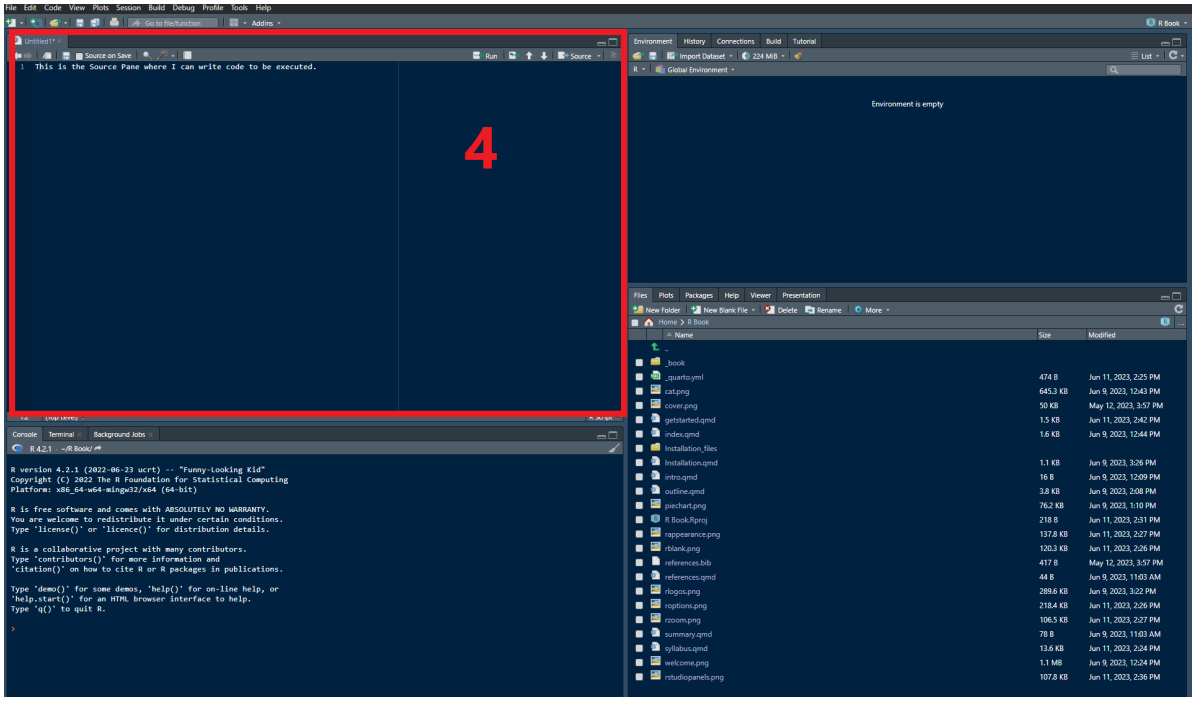
Now that you've settled on a theme and Editor font size, let's see what each of the panels in

RStudio mean:



1. This area is called the **Console**, and this is where code is executed.
2. This area is the **Environments Pane**. This is where you can see the objects stored in your R session, such as data, functions, variables, and other things.
3. This area is the **Output Pane**. This is where you can see files in your Working Directory (under the tab `Files`), see a preview Plots you might have created, see your list of **packages**, and more.

There is also a fourth pane, which is usually created by clicking File –> New File –> R Script. This will create a setup that looks like this:



4. This is the **Source Pane**. This is where you can view and edit various code files. By default, we will use this pane to write, edit, and execute **R Script** files. These files tell R what code to run and in what order to run it.

So there you have it. Those are the four main panels that we will use in R Studio to write and execute R Code. **Now, let's learn some R starting with** Chapter 3 **!**


WHO WOULD WIN?
MULTIBILLION COMPANIES THAT RULED THE STATISTICAL PROGRAMMING MARKET FOR DECADES
SOME WEIRD LETTER AND A SNAKE???
sas
STATA
R
imgflip.com
u/certified_officer
r/rstatsmemes

3 Introduction to R

By now, you should have installed R and R Studio (see Chapter 1 for details), and customized R Studio to the Editor theme and font size of your choice (see Chapter 2 for details). If you have not done these things, please do so now before proceeding.

3.1 What exactly is R, and why is it called *R* and not something more informative?

R is a computer language and run-time environment which can be used to do many things, including statistical computing and data visualization. I know, the name is weird. How can it be named after a letter?!

Here's the deal. R was created by statisticians **Ross Ihaka** and **Robert Gentleman** in the early 1990s at the University of Auckland. They developed a coding language to teach introductory statistics based on the syntax of another language at the time called *S* (programmers loved single-letter names back in the day). Since both their first names (Robert and Ross) started with the letter *R*, that's what they chose to call their new language. Seriously.

Importantly, Gentleman and Ihaka spoke with their colleagues who convinced them to release the language for free as part of a philosophy called the **Free Software Movement.** Under this philosophy, "users have the freedom to run, copy, distribute, study, change and improve the software" (see GNU for more details).

They created a mailing list to discuss all things R in 1994, which quickly became big enough to overwhelm them with many feature requests and bug reports. They then selected a core group of developers who would maintain R. They also enlisted the support of their colleagues Kurt Hornik and Fritz Leisch at the Technical University of Vienna who created the Comprehensive R Archive Network (CRAN), which is a repository of user contributions.

Today, R is a thriving language that receives regular updates by the R Core Team, and boasts about 19,657 user-contributed packages on CRAN. **R is free, flexible, powerful, and awesome.**

You might have heard of other programs for data analysis, and might be wondering which of these are open-source (meaning free!), and which are proprietary. I list some of the most common statistics programs, and whether they are open source or not in Table 3.1.

Figure 3.1: Ross Ihaka (right) and Robert Gentleman (left) - the creatoRs.

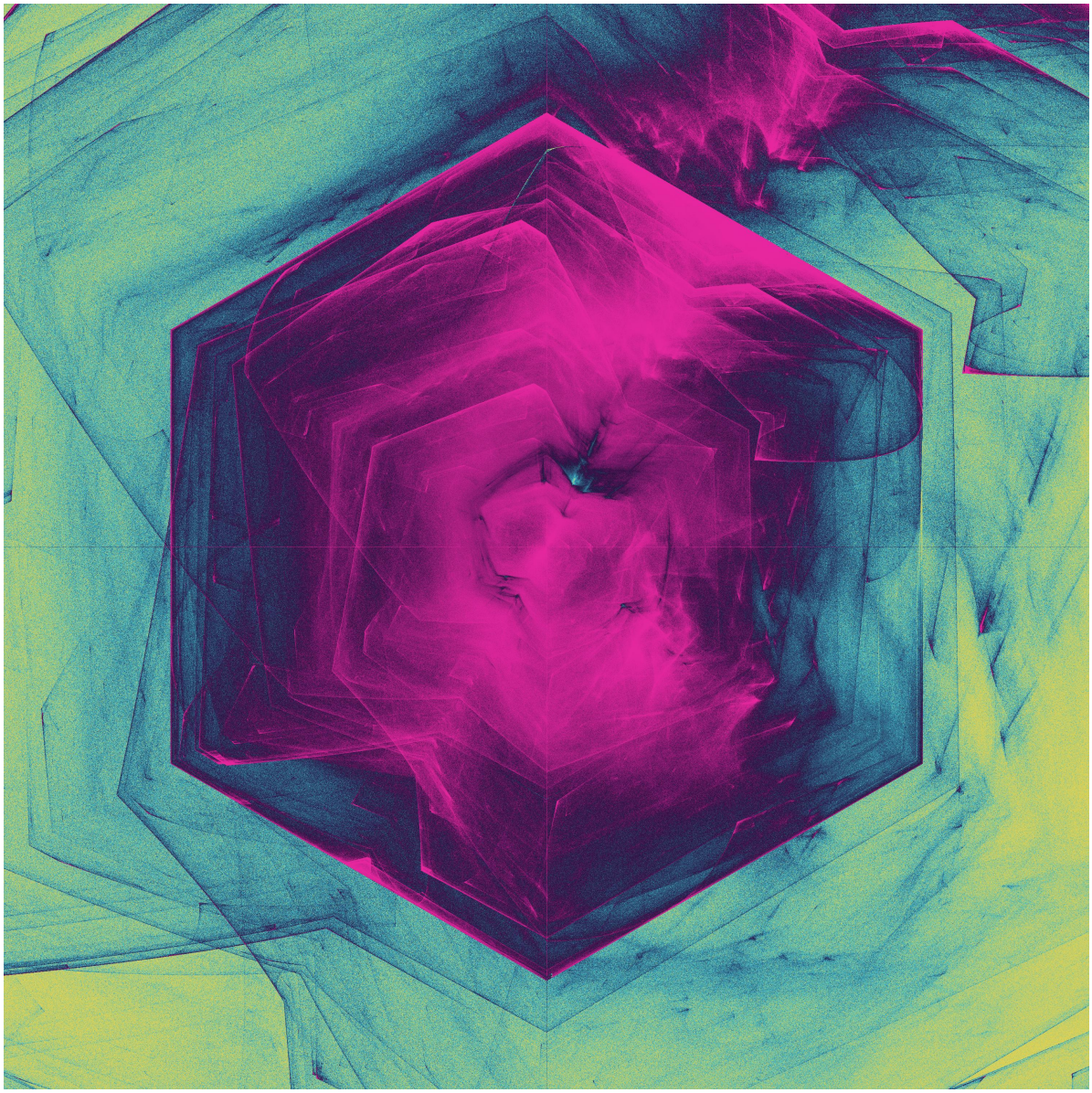


Figure 3.2: This art was made in R by Danielle Navarro. That's right, you can even make art in R. Click on the image if you want to see more of her work.

Table 3.1: Open source or not? How R compares to other programs used for data analysis.

Program/language used for data analysis	Is it Open Source (free) or not?	Company or organization maintaining the program/language
R	YES	Supported by R Core Team and R Foundation for Statistical Computing
Python	YES	Python Software Foundation
SAS	NO	SAS Institute
SPSS	NO	IBM
Stata	NO	StataCorp LLC
MATLAB	NO	MathWorks
JMP	NO	JMP Statistical Discovery LLC
MPlus	NO	Muthén & Muthén

After looking at the list in Table 3.1, it's natural to wonder "Why would I ever need to use a proprietary software when free versions exist?" That's a great question to ask. Personally, I see the main reason people use proprietary software is because they were trained to use a particular program and stuck with it. Whatever works for you! Another reason to use a proprietary software might be because it has functionality which you cannot find in the open-source versions. I have only encountered this case once with a very specific type of analytical approach, but generally speaking there are far more contributors to open-source packages and functions than you will find with proprietary software companies.

This book proudly takes the very opinionated position that R is the best.

3.2 Using R as a calculator

The most basic use of R is as a calculator. You can execute calculations in a Script file or in the Console directly. In addition to using any real number, the following symbols and operations can also be used:

Code	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation
( )	Parentheses/Brackets
sqrt()	Square Root

Code	Meaning
log()	Natural Logarithm
exp()	Exponential Value

Just enter the calculation in the Console (or Script File, though the Console is easier for quick calculations) following the appropriate order of operations (PEMDAS). As a matter of style, you want to include a space between arithmetic operators like `+`,`-`,`*`, and`/`. You also want to add a space after any commas, just like in English. This makes your code more readable.

```
  (6 * 6) + (12 / 2)
```

```
[1] 42
```

```
  sqrt(81) * sqrt(9)
```

```
[1] 27
```

```
  (2 / 3)^(4)
```

```
[1] 0.1975309
```

```
  log(100)
```

```
[1] 4.60517
```

```
  exp(4.61)
```

```
[1] 100.4841
```

```
  10^5
```

```
[1] 1e+05
```

As you see, the answer is given following the `[1]`, which just refers to the first position in a vector (to be discussed shortly). Note also that R uses scientific notation, so instead of printing `10^5=1000000`, R will print `10^5 = 1e+05`. The `e` here is completely unrelated to the number e (Euler's number). It just means "10 to the power of."

You can also easily round a number to a given place using the `round()` function, in which the first argument is the number or numeric vector to round, and the second argument `digits =` is the number of decimal places to which you want to round.

```
x <- log(100)*2.45

print(x)
```

```
[1] 11.28267
```

```
# Let's round this to two decimal places, then one decimal place, and then no decimal plac

round(x, digits = 2)
```

```
[1] 11.28
```

```
round(x, digits = 1)
```

```
[1] 11.3
```

```
round(x, digits = 0)
```

```
[1] 11
```

"We talking about practice, man" - Allen Iverson

Go ahead and try doing some calculations of your own in the Console! Especially if you're new to R, the best way to learn R is by doing R over and over again.

3.3 Comments

You can and should always write comments in your R Script file using a single hash `#`. It's a good idea to use `-` or `=` to also clearly break up the Script file into chunks. The easiest and cleanest way to do this is to use Section Headers. On a Windows, you can use Ctrl+Shift+R and on Mac, you can use Cmd+Shift+R to quickly add a section header. You can also go to Code –> Insert Section. Then you can very neatly organize your Script File, like so:

```
# Load Libraries ------------------------------------------------------

# Load Data -----------------------------------------------------------
```

It's a good idea to use comments to explain why you are doing something. For instance, if you are using a specific analytic approach, mention in a comment why you are doing that. I also recommend starting every Script File with the following information on separate commented lines: Title of file, Author, Date, Purpose. Here is an example:

```
# Data Cleaning
# By Shaon Lahiri
# June 14, 2027
# This file imports the dataset "piracy.csv" and prepares it for data analysis.
```

By having clear meta-data for each Script file like this, someone else who reads your code can better understand its purpose, provenance, and format. Additionally, comments on the purpose of a particular line of code can also help you remember why you did something, as it can be easy to forget years later what you had in mind for a particular task.

> Keyboard Shortcuts
>
> It can be very useful to know keyboard shortcuts for common tasks. You may want to bookmark this page which lists keyboard shortcuts in R Studio for Mac and Windows.

3.4 Functions

A **function** is a block of code that runs only when it is *called*. Calling a function just means you are giving the computer a single instruction to do a particular thing. In R, every function comprise a word followed immediately by parentheses `function()`. We often have information we want to pass to a function, and these are called **arguments.** Essentially, the function does the thing we want, and the arguments specify how we want them done, or what information we want the function to use.

Let's try using the simple `print()` function to display the phrase "Hello World."

```
print("Hello, World!")
```

```
[1] "Hello, World!"
```

Here we called the `print()` function and passed it the argument `"Hello World!"`. This resulted in the expected behavior of the value `"Hello, World!"` being printed (i.e. shown on screen).

> **Why "Hello, World!"?**
>
> Did you know that writing code to print "Hello World" is a programming tradition? It's often the first thing a student learns to write in a programming language, and comes to us from the 1970s in Bell Laboratories. Learn more here.

Besides using functions to accomplish various tasks, we can also easily create our own functions in R. Creating your own function is often a way to automate a series of tasks to streamline your work. We will tackle this issue later. For now, it will be sufficient to be familiar with the terms *function*, *argument*, and *call*.

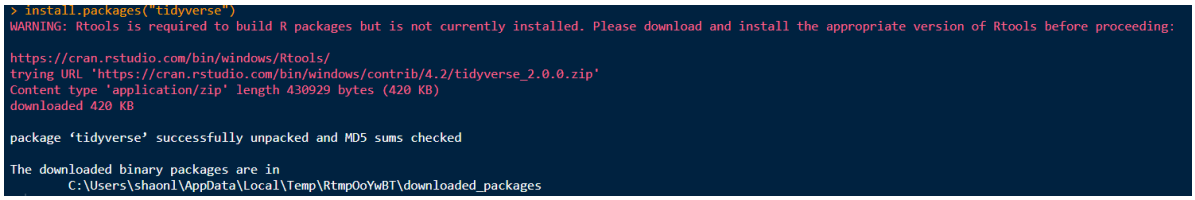
3.5 R Packages

Packages are user-written contributions which extend the functionality of R. They typically comprise a set of functions, code, documentation, and occasionally some datasets as well. Packages are stored in R as a 'library'. They only have to installed once, but must be loaded each time you intend to use them. Typically, we only have need for a few packages at a time.

Installing a package typically involves communicating with CRAN, and is easily accomplished using the `install.packages()` function in which the name of the package you want to install should be placed in the parentheses with quotation marks `" "`. Go ahead now and give it a shot with the `tidyverse` package (or set of packages rather) using the following code.

```
install.packages("tidyverse")
```

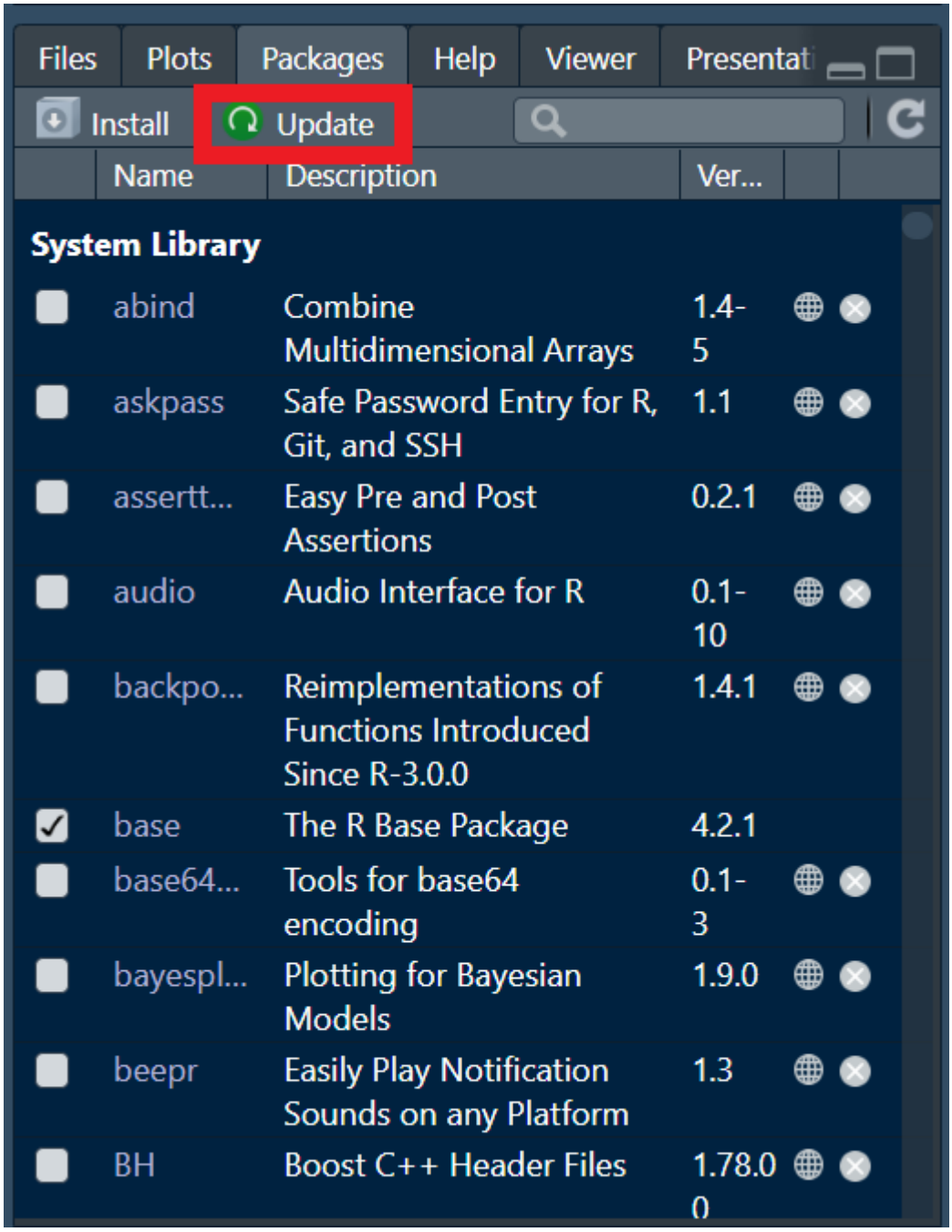
If the installation was successful, you should see a message that mentions the package has been successfully unpacked and located somewhere on your computer like this:



See how it says that the downloaded package is located on my computer? This type of message indicates that there were no problems with the installation. Now, if we want to use this package, it must be explicitly called using the `library()` function. This is ideally done in a dedicated section called "Load Libraries" or something similar.

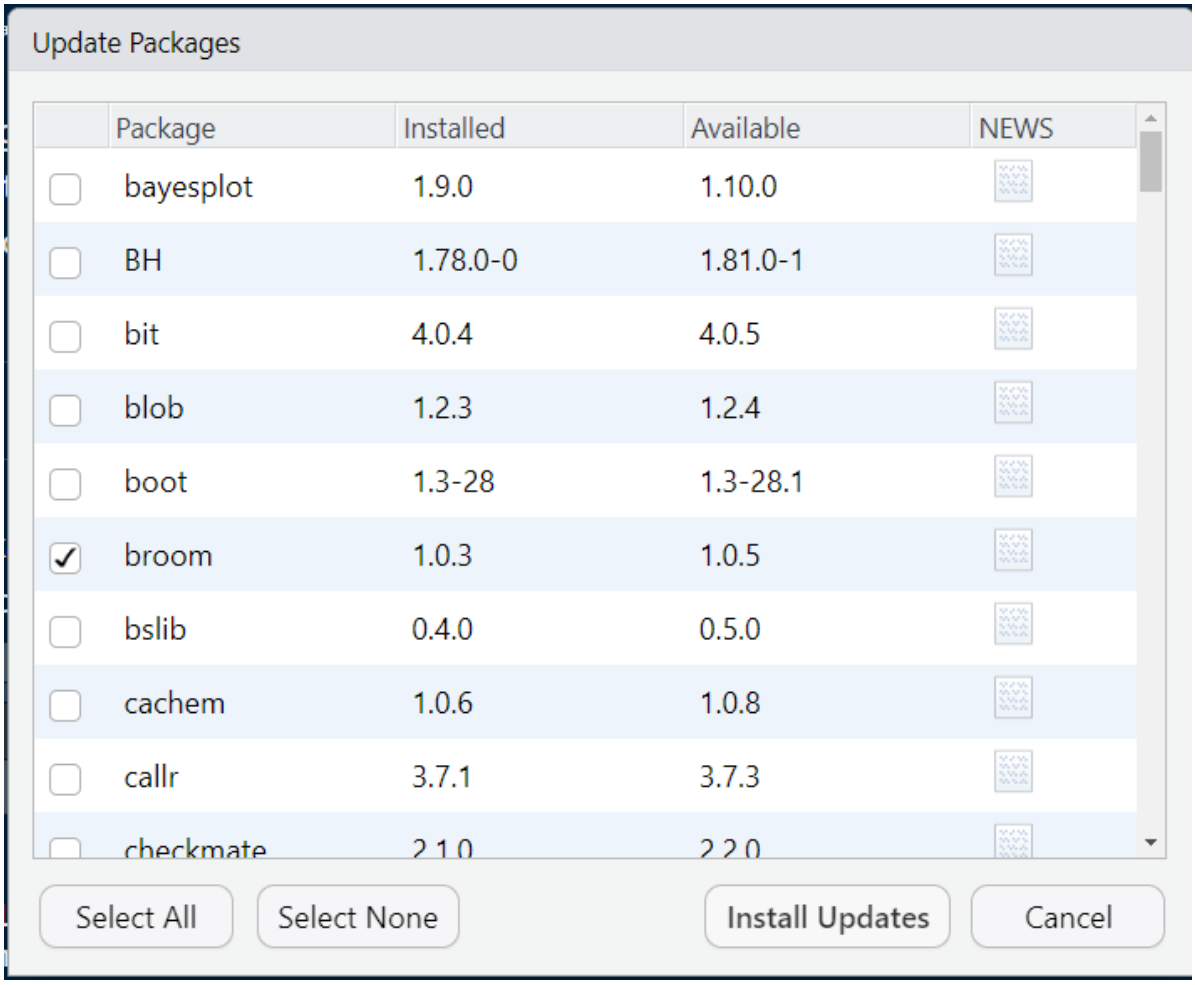
```
# Load libraries --------------------------
library(tidyverse)
```

Now you know how to install and load packages from CRAN. This is what you will mostly do in R, as CRAN packages are all tested and meet its quality standards. To update a package, I find the easiest, cleanest approach is to go to your Output Pane in the bottom right of the screen, go to the Packages tab, and then hit the Update button.



That will bring up a menu which gives you several relevant pieces of information:

- Which package(s) have an update available.
- What version of the package you have installed vs what version is available.
- What changes the update is making under 'NEWS'.



Maybe you don't want to sit through the updates of EVERY package for which an update is available. This menu allows you to pick and choose the packages you want to update, and then click "Install Updates." If you have a specific package you want to update and want to do it quickly, you can use the `install.packages()` function with the package name to be updated in quotes.

```
install.packages("bayesplot")
```

Finally, say you want to just have all your packages up to date, and have some time to let R do its thing while you get a coffee, you can just type:

```
update.packages(ask=FALSE)
```

This will update every package for which an update is available, and the `ask=FALSE` argument suppresses a prompt from appearing before every package update.

Finally, you may be wondering which packages are essential to install and have ready to go. Typically, you know which packages to use based on a particular task you want to accomplish. Thus, I wouldn't worry about this for now. Any R script file using packages will always have those listed in the file, and thus you will typically know what packages to install and load.

3.6 Assignments

In R, everything is an **object.** An object is just some data that you have stored. It can be a number, a dataset, a function, or anything else. For example, let's say we want to perform the following calculations:

```
((sqrt(81) * sqrt(9)) / log(3.6)) + 6
((sqrt(81) * sqrt(9)) / log(3.6)) - 12
((sqrt(81) * sqrt(9)) / log(3.6)) * (2/3)
```

We may not want to keep writing `((sqrt(81) * sqrt(9)) / log(3.6))` as it's cumbersome to work with and to look at. We can streamline the process by assigning `((sqrt(81) * sqrt(9)) / log(3.6))` to a simple name of our choice. Let's say we assign it to an object called `quantity1`, the resulting computation would be much easier to write. We can do this by using the assignment operator `<-` which is comprised of the symbol for *less than* followed immediately by a minus sign `-`. So, the code incorporating assignment of the quantity to an object called `quantity1` would look like this:

```
quantity1 <- ((sqrt(81) * sqrt(9)) / log(3.6))

quantity1 + 6
quantity1 - 12
quantity1 * (2/3)
```

As you'll notice, the name of the object always comes first, then the assignment operator `<-`, and finally the value we are assigning to the object.

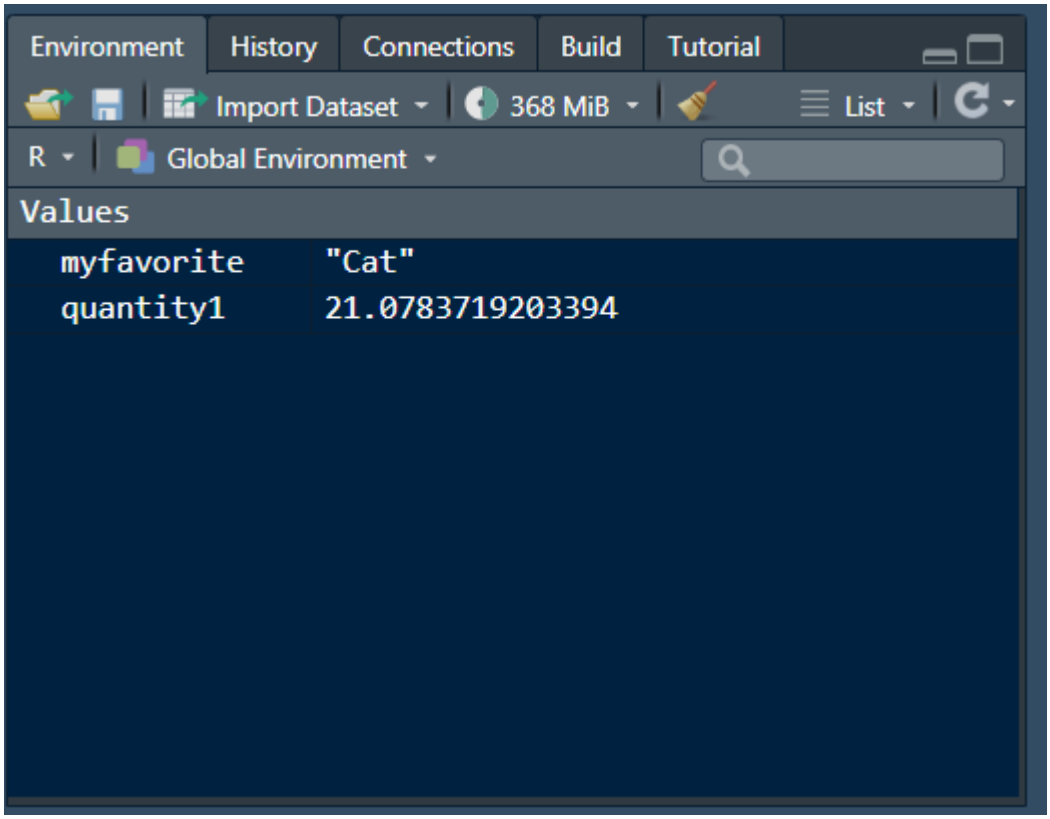
You can also assign text to an object if you enclose the text in quotes `" "`.

```
myfavorite <- "Cat"
myfavorite
```

```
[1] "Cat"
```

Here, I assigned the value `"Cat"` to an object called `myfavorite`. I then called that object, and this printed its value.

One you assign a value to an object, it will show up in your Environments Pane on the top-right of your screen. This allows you to keep track of all the objects you have created in your R session, along with their values.



You can also assign the same value to multiple variables using multiple assignment operators `<- <- <-`. For example, if I want to assign the value `male` to three names, this is what that would look like.

```
# Assign multiple variables the value male.
Brendan <- Mark <- Tyrone <- "male"

print(Brendan)
```

```
[1] "male"
```

```
print(Mark)
```

```
[1] "male"
```

```
print(Tyrone)
```

```
[1] "male"
```

If you want to remove an object from your environment, you can use the `rm()` function with the name of your object in parentheses. For example, if I wanted to remove the object `quantity1`, I would write `rm(quantity1)`. If you want to remove all objects from your environment, you can run the command `rm(list=ls())`.

> ⚠ You can't name an object just anything!
>
> There are some words in R that are reserved for particular tasks. These **Reserved Words** cannot be used to name an object. So, don't ever use these words alone to name an object.

break	NA
else	NaN
FALSE	next
TRUE	repeat
for	return
function	while
Inf	if

3.7 Help

If you ever need to know how a function works, such as what arguments it take, you can easily see documentation for it by adding a question mark `?` before the function name. For example, if you can't remember what the `rm()` function does, or what arguments it takes, you can write `?rm` and see the documentation for it. For more complex problems or troubleshooting, you can and should become familiar with Googling the issue, and then seeing the best answer on Stack Overflow in particular. Stack Overflow is one the best places to get answers for troublesome R code. You should also note that everyone, and I mean absolutely, everyone Googles code issues at some point or another. This is because coding and data science are dynamic fields, and things are often changing. It's sensible to keep up with how people are solving particular problems by seeing what people are saying on Stack Overflow or other fora.

3.8 Exercises

It's a good idea to attempt these right away after reading this section while the content is fresh. You can find the answers in Appendix A.

1. Calculate $89 + 9/(4 * 5)^2$ and assign the value to the object `solution1`. Then print `solution1`.
2. What is the difference between R and R Studio?
3. How do you add a comment to a Script file?
4. What are packages in R, and how do you install them?
5. Install and load the package `rstudioapi`.
6. How do you modify the appearance of R Studio?
7. Assign the value of `365` to an object called `year`. Then, create another object called `months` and assign it the value `year * 0.032854884083862`.
8. Create three variables named `Cowboys`, `Giants`, and `Commanders` and assign them all the value `"Inferior Team"` using multiple assignment operators.

4 Data Types

Before we can dive into data management and cleaning, we need to know what types of data can be stored in R. There are five main types of objects that one can use to store data in R. These are summarized in Figure 4.1.

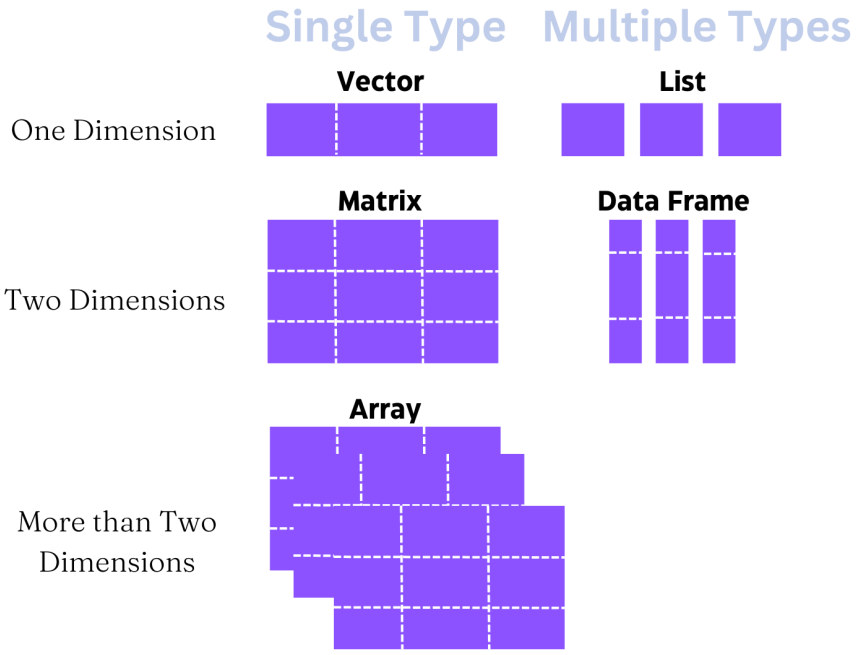


Figure 4.1: Five types of data in R. Adapted from Grolemund (2014).[1]

What do we mean by different types of data? Well, data can be many things - text, images, video, recordings, numbers, etc. For our purposes, data will comprise numbers and text. Within these two broad classes, there are five main types of objects used to store data in R, as shown in Figure 4.1. Let's break each of these down with an emphasis on vectors and dataframes, since we will work with these types most often.

4.1 Vectors

4.1.1 Types

A **vector** is a sequence of data elements of the same type. In R, there are two types of vectors - **atomic vectors** and **lists**. Within the category of atomic vectors, there are six types:

- **Logical** (TRUE, FALSE, NA)
- **Integer** (a type of numeric vector containing only integers)
- **Double** (a type of numeric vector which is the default storage type for numbers in R; represents floating point numbers which can't always be precisely represented by fixed memory)
- **Character** (a vector of *strings*, which are pieces of text. Each string is surrounded by quotes `" "`.)
- **Complex** (a vector of elements that include complex numbers)
- **Raw** (a vector containing a 'raw' sequence of bytes; very unusual data type)

You should note that Doubles and Integers are both numeric vectors, but Doubles are far more common data structures that we typically deal with in social science research. Doubles are quite flexible in that they can be written in decimal, scientific, or hexadecimal form. They also have three unique values: `Inf` (Infinity), `-Inf` (Negative Infinity), and `NaN` (not a number). In practical usage, if you get a value corresponding to any of these three values, something has probably gone wrong. For instance, let's see what happens when you try to divide certain values by 0 (which is, of course, undefined):

```
54/0
```

```
[1] Inf
```

```
-54/0
```

```
[1] -Inf
```

```
0/0
```

```
[1] NaN
```

Integers cannot contain fractional values (i.e. no decimals), and are written like Doubles but are followed by an `L` as in `2L, 3L` and so on. Generally, we won't have to worry about these since we will usually be working with Doubles, Logical, or Character vectors.

The most common vectors we will deal with are Logical, Double, and Character vectors. Atomic vectors contain elements of the same type. You can always check the type of vector with the `typeof()` or `class()` commands. Additionally, you can see the number of elements in a given vector with the `length()` command.

4.1.2 Creating Vectors

As mentioned, vectors can comprise strings (pieces of text), integers, or logical values. While they can also comprise a sequence of bytes and complex numbers, we will not discuss these as they are less relevant for our purposes. Let's start with creating a numeric vector, which can be an integer or double (it doesn't matter for our purposes). The most common way to create a vector is to use the concatenate or combine `c()` function with values in parentheses separated by commas. To create a character vector, you need to surround each element of the vector in quotes `" "` separated by commas. The `paste()` function can also be useful in combining vectors.

You can also create a vector comprising a sequence of numbers using the colon `:`.

```
# Create two numeric vectors.
a <- c(1, 2, 3, 4)
b <- c(5, 6, 7, 8)

# You can do the same thing with a colon instead.
a <- c(1:4)
b <- c(5:8)

# Create a new vector c that combines vectors a and b.
c <- c(a,b)
print(c)
```

```
[1] 1 2 3 4 5 6 7 8
```

```
# Create two character vectors
d <- c("Nirvana", "Alice in Chains")
e <- c("Soundgarden", "Pearl Jam")

# Create a new vector f that combines vectors d and e.
f <- c(d,e)
```

```
# Create three character vectors.
g <- "Four score and seven years ago our fathers brought forth on this continent,"
h <- "a new nation, conceived in Liberty, and dedicated to the"
i <- "proposition that all men are created equal."

# Combine the character vectors using the paste() function.
j <- paste(g,h,i)
print(j)
```

```
[1] "Four score and seven years ago our fathers brought forth on this continent, a new natio
```

4.1.3 Extracting Elements from a Vector

To extract an element from a vector, we use square brackets `[]` after the vector name, and write the position of the element(s) we want to extract. We can extract multiple elements by using the concatenate or combine `c()` function, such as `vector1[c(1,4,5)]`. To extract elements in a sequential range, we can use colon `:` between positions, such as `vector1[c(1:5)]`.

Let's try an example with a visual summary. First, let's create a character vector called `catbreeds` with the values `"American Bobtail"`, `"Abyssinian"`, `"Burmese"`, `"Himalayan"`, and `"Manx"`. We will create the vector with our trusty concatenate or combine `c()` function. Remember that if you have strings, or pieces of text, these are surrounded by quotes `" "`. You don't need quotes if you are creating any of the other vectors mentioned.

```
# I like one string per line to keep things organized and neat.
catbreeds <- c("American Bobtail",
               "Abyssinian",
               "Burmese",
               "Himalayan",
               "Manx")
# Now I write the name of the vector to see its elements.
catbreeds
```

```
[1] "American Bobtail" "Abyssinian"       "Burmese"          "Himalayan"
[5] "Manx"
```

Ok, so now we have a nice character vector called `catbreeds` where the different strings (or pieces of text) correspond to different cat breeds. Now, how do we extract or print different elements of the vector? This is summarized in Figure 4.2.

You can see what this looks like in practice below.

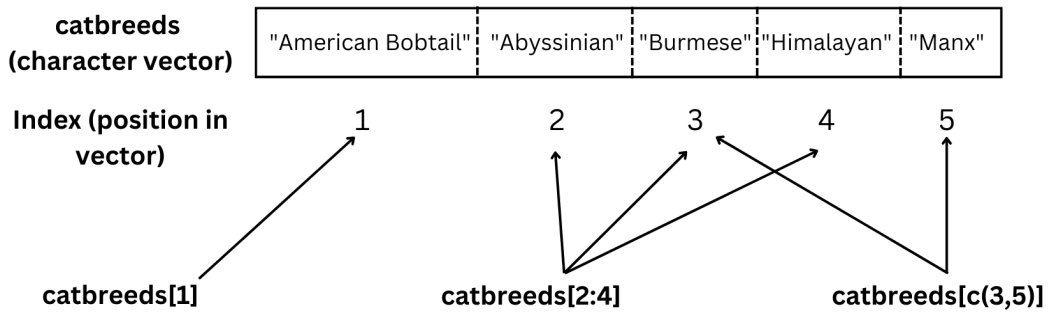


Figure 4.2: Retrieving different elements of a vector by position number.

```
# Let's extract "American Bobtail" which is the first element in the vector.
catbreeds[1]
```

```
[1] "American Bobtail"
```

```
# Now let's extract three breeds - Abyssinian, Burmese, and Himalayan.
# These are elements 2, 3, and 4 in the vector.
catbreeds[2:4]
```

```
[1] "Abyssinian" "Burmese"    "Himalayan"
```

```
# What if we want to extract multiple elements that are not sequential?
# Let's do that with Burmese (element 3) and Manx (element 5).
catbreeds[c(3,5)]
```

```
[1] "Burmese" "Manx"
```

You can also extract only positive or only negative integers in a vector by specifying the relevant rule within square brackets after the vector name. For example, let's create a new vector `vectorbeta` that comprises only the positive integers of `vectoralpha`. Then we'll create another new vector `vectorgamma` that comprises only the negative integers of `vectoralpha`.

```r
# Create a numeric vector
vectoralpha <- c(29, 149, 217, -226, 55, 64, -103, -313, 368, 189)

# Create a new vector vectorbeta that takes only positive integers of vectoralpha.
# This code says take all the integers greater than 0 in vectoralpha and assign
# the values to a new vector called vectorbeta.
vectorbeta <- vectoralpha[vectoralpha > 0]
print(vectorbeta)
```

```
[1]  29 149 217  55  64 368 189
```

```r
# Create a new vector vectorgamma that takes only the negative integers of vectoralpha.
# This code says take all the integers less than 0 in vectoralpha and assign
# the values to a new vector called vectorgamma.
vectorgamma <- vectoralpha[vectoralpha < 0]
print(vectorgamma)
```

```
[1] -226 -103 -313
```

4.1.4 Vector Operations

In a numeric vector, you can perform a number of operations such as finding the mean `mean()`, median `median()`, minimum `min()`, and maximum `max()`.

```r
# Create a numeric vector

vectoralpha <- c(29, 149, 217, -226, 55, 64, -103, -313, 368, 189)

# Find the minimum and maximum values in the vector.

min(vectoralpha)
```

```
[1] -313
```

```r
max(vectoralpha)
```

```
[1] 368
```

```r
# Find the mean and median of the vector.

mean(vectoralpha)
```

```
[1] 42.9
```

```r
median(vectoralpha)
```

```
[1] 59.5
```

You can also perform arithmetic operations on vectors. Let's create a couple of vectors and illustrate the four basic arithmetic operations. Each operation is performed element by element. This means that if both vectors are the same length (same number of elements), the operation will be conducted on the first element of both vectors, then the second element of both vectors, and so on. For example, in vectors a1 and b1 below, addition would involve the following steps: $1 + 2$, $3 + 1$, and $6 + 4$ since these number comprise the first, second, and third elements of each vector, respectively.

```r
# Create two vectors.
a1 <- c(1, 3, 6)
b1 <- c(2, 1, 4)

# Vector Addition
a1 + b1
```

```
[1]  3  4 10
```

```r
# Vector Subtraction
a1 - b1
```

```
[1] -1  2  2
```

```r
# Vector Multiplication
a1 * b1
```

```
[1]  2  3 24
```

```
# Vector Division
a1 / b1
```

```
[1] 0.5 3.0 1.5
```

Don't Just Keep Reading!

Figure 4.3: Manx cat wants you to try coding up a couple of vectors of your own before proceeding.

Try creating your own vector and extracting the elements. Once you feel like you've got it, only then move on to Matrices. Try creating a numeric vector as well, and extracting its elements using the techniques illustrated above.

4.2 Matrices

A **matrix** is a two-dimensional vector (i.e. rows and columns.) All columns in matrix should have the same type (e.g. logical, character, numeric) and same length. This means that matrices are *homogeneous* data structures in that all elements must be of the same type.

You can create a matrix from vectors using the `rbind()` function to combine rows of data, and using the `cbind()` function to combine columns of data.

```
# row bind

a <- c(.94, .92, .95)
b <- c(.25, .56, .82)
c <- c(.65, .45, .37)
d <- rbind(a, b, c)

print(d)
```

```
  [,1] [,2] [,3]
a 0.94 0.92 0.95
b 0.25 0.56 0.82
c 0.65 0.45 0.37
```

```
# column bind
e <- c(34.5, 23.6)
f <- c(22.3, 13.2)
g <- cbind(e,f)

print(g)
```

```
        e    f
[1,] 34.5 22.3
[2,] 23.6 13.2
```

You can also create a matrix using the `matrix()` function in which the main arguments are the data vector, the number of rows, and the number of columns. Here is a simple matrix with the numbers 1:9 spread over three rows and three columns:

```
matrix(1:9, nrow = 3, ncol = 3)
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

You can also change the names of the rows and columns of the matrix using the `rownames()` and `colnames()` functions. Let's try using the `matrix()` function to input a correlation matrix into R. A correlation matrix is a type of matrix which displays correlation coefficients between variables. Essentially, it shows you the strength of a relationship between pairs of variables ranging from -1 to 1. There are many types of correlation coefficients, but typically Pearson's r is used. When you look at a correlation matrix, you will notice that 1s are on the diagonal, because a variable is always perfectly correlated with itself. The upper triangle (above the diagonal) is a mirror image of the bottom triangle (below the diagonal), so you only need to focus on one. Personally, I like focusing on the lower triangle, but it doesn't much matter.

Imagine a study on exam performance and study habits with the following variables:

- Hours spent studying
- Exam Score
- IQ Score
- Hours spent sleeping
- School Rating

Now let us examine the correlation matrix of the variables of this hypothetical study illustrated in Figure 4.4.

	Hours spent studying	Exam score	IQ score	Hours spent sleeping	School rating
Hours spent studying	1.00	0.82	0.48	-0.22	0.36
Exam score	0.82	1.00	0.33	-0.04	0.23
IQ score	0.08	0.33	1.00	0.06	0.02
Hours spent sleeping	-0.22	-0.04	0.06	1.00	0.12
School rating	0.36	0.23	0.02	0.12	1.00

Figure 4.4: A correlation matrix - correlations between variables are on either side of the 1s on the diagonal.

It looks like there is a strong correlation ($r = 0.82$) between hours spent studying and exam score, which makes sense. We also have smaller correlations between IQ and exam score ($r = 0.33$) and school rating and exam score ($r = 0.23$). All other correlations with exam score are negligible. Let's now input this correlation matrix into R.

```r
# Input correlations

corrs <- c(1.00, 0.82, 0.08, -0.22, 0.36, 0.82,
           1.00, 0.33, -0.04, 0.23, 0.48, 0.33,
           1.00, 0.06, 0.02, -0.22, -0.04, 0.06,
           1.00, 0.12, 0.36, 0.23, 0.02, 0.12,
           1.00)

# Use matrix() function and specify appropriate rows and columns

corrmat1 <- matrix(corrs, nrow = 5, ncol = 5)

# Rename rows and columns

colnames(corrmat1) <- c("Hours studying",
                        "Exam Score",
                        "IQ Score",
                        "Hours sleeping",
                        "School rating")

rownames(corrmat1) <- c("Hours studying",
                        "Exam Score",
                        "IQ Score",
                        "Hours sleeping",
                        "School rating")

print(corrmat1)
```

```
               Hours studying Exam Score IQ Score Hours sleeping School rating
Hours studying           1.00       0.82     0.48          -0.22          0.36
Exam Score               0.82       1.00     0.33          -0.04          0.23
IQ Score                 0.08       0.33     1.00           0.06          0.02
Hours sleeping          -0.22      -0.04     0.06           1.00          0.12
School rating            0.36       0.23     0.02           0.12          1.00
```

To retrieve an element of a matrix, write the matrix name, followed by square brackets in which the first argument is the row and the second is the column that you want. For example, if we want to retrieve the correlation between `"Hours spent studying"` and `"Exam Score"`, we can write the matrix name `corrmat1` followed by row 2 and column 1:

```r
corrmat1[2,1]
```

```
[1] 0.82
```

This retrieves the appropriate correlation of 0.82. Finally, let's say you want to see the values for an entire column or an entire row. In this case, simply leave the row or column argument blank:

```
# See the values for column 3 only
corrmat1[, 3]
```

```
Hours studying     Exam Score       IQ Score Hours sleeping  School rating
          0.48           0.33           1.00           0.06           0.02
```

```
# See the values for row 4 only
corrmat1[4, ]
```

```
Hours studying     Exam Score       IQ Score Hours sleeping  School rating
         -0.22          -0.04           0.06           1.00           0.12
```

4.3 Dataframes

A dataframe is the most common type of data structure we typically deal with in social science. It is a two-dimensional labelled list of vectors, and can have columns of multiple data types. A dataframe's vectors must be of the same length, which gives dataframes a rectangular structure. A dataframe has `rownames()` and `colnames()` just like matrices and lists. The variable names in a dataframe also must be different from one another, meaning that the two variables cannot have the exact same name.

If you want to explore R's built-in dataframes to test out functions and analyses, you can run the `data()` command to see a list of dataframes. Let's go ahead and use the built-in dataframe `mtcars` by assigning it to an object called `df1` (dataframe 1). Then, let's look at the dataframe. If you want to have a very quick preview of the first six rows of the dataframe, invoke the `head()` function, and similarly the last six rows of the dataframe can be previewed using the `tail()` function. In both cases, the name of the dataframe should be inside the parentheses.

The `head()` and `tail()` functions are useful when you have a very large dataset and cannot feasibly examine the whole thing. However, I find the `View()` function far more useful. This actually pulls up the entire dataframe in a separate tab.

```
# Assign the built-in mtcars dataframe to an object called df1.
df1 <- mtcars

# Examine the first six rows of the dataframe.
head(df1)
```

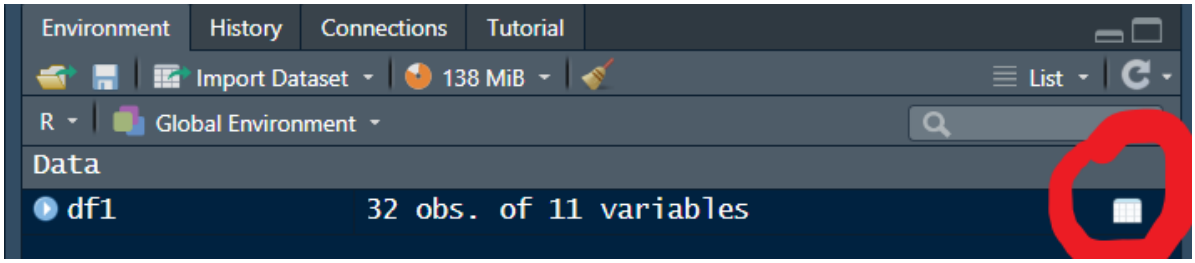
```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Examine the last six rows of the dataframe.
tail(df1)
```

```
                mpg cyl  disp  hp drat    wt qsec vs am gear carb
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```
# Have a much more detailed look at the dataframe.
View(df1)
```

You can also click the little dataframe icon next to the dataframe in your Environment tab to view the dataframe.



Whether you use the `View()` function or click the little icon, R will open another tab with the dataframe, and will look something like this:

Filter

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1

Showing 1 to 26 of 32 entries, 11 total columns

If we want to know the dimensions of the dataframe (imagine it's huge and we can't easily `View()` the whole dataset), as well as the structure of the dataset, we can use the functions: `nrow()` (to see the number of rows), `ncol()` (to see the number of columns), and `str()` (see the structure of the dataframe).

```r
# See the number of rows in a dataframe
nrow(df1)
```

```
[1] 32
```

```r
# See the number of columns in a dataframe
ncol(df1)
```

```
[1] 11
```

```r
# See the structure of the dataframe
str(df1)
```

```
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Note that the `str()` function gives you a quick overview of the type of variables and some of their values in the dataframe. You can get this information using the `View()` function, which I tend to prefer, but if you want to quickly examine the structure of multiple dataframes, for instance, you can use the `str()` function.

Notice how the dataframe comprises rows and columns, where the rows are names of cars and the columns are different attributes of those cars, such as miles per gallon, number of cylinders, displacement weight, and others. To retrieve a particular column in the dataframe, use the dollar sign `$` operator after the name of the dataframe, followed by the column name. For instance, if you want to know the mean miles per gallon across all the cars, we can use the `mean()` function in the following manner:

```
# Get the mean of the mpg variable in the df1 dataframe.
mean(df1$mpg)
```

```
[1] 20.09062
```

From the `mean()` function used above, we see that across all the cars in the dataframe, the average fuel efficiency is about 20 miles per gallon.

In general, you will use the format `dfname$variable` to access or refer to any specific variable or column of a dataframe. You can also add a new variable to the dataframe using the same format. For example, if we wanted to add a new column to our dataset `df1` in which we take the existing column `qsec` (the number of seconds it takes the car to travel one-fourth of a mile) and multiply this by four, to get an estimate of how long it takes the car to travel one mile, we can accomplish by with the following code.

```
# Create new variable in df1 called qsec2 which takes the existing variable
# qsec and multiplies it by four.
df1$qsec2 <- df1$qsec*4

# The mean of qsec2 is predictably four times the mean of qsec.
mean(df1$qsec)
```

```
[1] 17.84875
```

```
mean(df1$qsec2)
```

```
[1] 71.395
```

To get a quantitative summary of numeric variables in a dataframe, we can use the `summary()` function to obtain the minimum value, 25th percentile, median, 75th percentile, and maximum value. Let's try that with the weight `wt` variable. Then, we'll obtain the information for multiple variables in the dataframe using square brackets `[]`, the concatenation operator `c()`, and single apostrophes `' '` around the variable names.

```
# Summary statistics for only the weight variable
summary(df1$wt)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.513   2.581   3.325   3.217   3.610   5.424
```

```r
# Summary statistics for multiple variables
summary(df1[c('mpg','wt', 'qsec')])
```

```
      mpg              wt             qsec      
 Min.   :10.40   Min.   :1.513   Min.   :14.50  
 1st Qu.:15.43   1st Qu.:2.581   1st Qu.:16.89  
 Median :19.20   Median :3.325   Median :17.71  
 Mean   :20.09   Mean   :3.217   Mean   :17.85  
 3rd Qu.:22.80   3rd Qu.:3.610   3rd Qu.:18.90  
 Max.   :33.90   Max.   :5.424   Max.   :22.90  
```

4.4 Lists

While dataframes can contain data of different types (*heterogeneous data)*, lists are even more flexible at doing this. A list is like a collection of different things that don't have to play by the rules of a dataframe. Recall that though dataframes can contain heterogeneous data, the elements of a dataframe are subject to a few restrictions such as a) they have to be a labelled list of vectors, b) they have to be the same length, and c) the variables must not have the exact same name.

List elements don't have such restrictions, and you can think of them as collections stuff. For example, you can put the contents of a function, a dataframe, a piece of text, numbers, etc. Let's look at an example of a list called `stuff`. We will create this list using the `list()` function and assigning a bunch of random stuff to it. You can even put lists in a list! That's pretty meta. You can then `View()` the list to examine its contents.

```r
# Create a list called stuff comprising various random things.
stuff <- list(catbreed = "manx",
```

```
                data = mtcars,
                Euler = 2.71828,
                logical = c(TRUE, FALSE, TRUE, NA),
                random = list(alphabet = letters,
                              greeting = "Hello, World",
                              number = pi
                              )
                )
```

Once you `View()` the list, it will show you the contents as in **?@fig-stufflist**.

Name	Type	Value
stuff	list [5]	List of length 5
catbreed	character [1]	'manx'
data	list [32 x 11] (S3: data.frame)	A data.frame with 32 rows and 11 columns
mpg	double [32]	21.0 21.0 22.8 21.4 18.7 18.1 ...
cyl	double [32]	6 6 4 6 8 6 ...
disp	double [32]	160 160 108 258 360 225 ...
hp	double [32]	110 110 93 110 175 105 ...
drat	double [32]	3.90 3.90 3.85 3.08 3.15 2.76 ...
wt	double [32]	2.62 2.88 2.32 3.21 3.44 3.46 ...
qsec	double [32]	16.5 17.0 18.6 19.4 17.0 20.2 ...
vs	double [32]	0 0 1 1 0 1 ...
am	double [32]	1 1 1 0 0 0 ...
gear	double [32]	4 4 4 3 3 3 ...
carb	double [32]	4 4 1 1 2 1 ...
Euler	double [1]	2.71828
logical	logical [4]	TRUE FALSE TRUE NA
random	list [3]	List of length 3
alphabet	character [26]	'a' 'b' 'c' 'd' 'e' 'f' ...
greeting	character [1]	'Hello, World'
number	double [1]	3.141593

You can see that our list has a string ("Manx"), a dataframe (mtcars), a number (Euler), a logical vector (logical), and another list with three elements of its own (random). Lists are so flexible!

You can very easily extract elements of a list with the dollar sign `$` operator if you know the name of the element, or using double square brackets `[[ ]]` if you know the position of the element. You can also use single square brackets `[ ]` which will give you a little more information than double square brackets, such as the name of the element.

```
# Extract elements of a list by name using dollar sign operator.
stuff$Euler
```

```
[1] 2.71828
```

```
# Extract elements of a list by using double square brackets.
stuff[[3]]
```

```
[1] 2.71828
```

```
# Extract elements of a list by using single square brackets.
stuff[3]
```

```
$Euler
[1] 2.71828
```

As you see, using single brackets `[ ]` to extract an element from a list gives you the element name `Euler` as well as the value, as opposed to only the value provided by the double square brackets `[[ ]]`. In general, if you have a collection of stuff, create a list for easy reference.

4.5 Arrays

Though we will not work with Arrays in this course, it may be worthwhile to know a thing or two about them. Recall that matrices and dataframes are two-dimensional data structures. An array goes beyond two dimensions, and can hold n-dimensional data, but the data have to be of the same type. I think arrays are easiest to understand when thinking about a collection of multiple matrices.

For example, let's say we have two vectors of different lengths. One vector has three elements, while the other has six elements.

```
# Create two vectors of different lengths (three and six).
vector1 <- c(1,3,4)
vector2 <- c(32, 4, 7, 19, 23, 43)
```

Now, let's create an array where we first create a matrix of three rows and three columns based on our vectors, and then duplicate this matrix four times. We can accomplish this using the `array()` function where the first argument will comprise our vectors, and the argument `dim` (or dimension) takes the row numbers, column numbers, and number of matrices we want. So if we want three rows, three columns, and four matrices, the values would be `dim = c(3,3,4)`.

```
# Use previous vectors to create array of four 3x3 matrices.
array1 <- array(c(vector1, vector2),
                dim = c(3,3,4)
                )

print(array1)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1   32   19
[2,]    3    4   23
[3,]    4    7   43

, , 2

     [,1] [,2] [,3]
[1,]    1   32   19
[2,]    3    4   23
[3,]    4    7   43
```

```
, , 3

     [,1] [,2] [,3]
[1,]    1   32   19
[2,]    3    4   23
[3,]    4    7   43

, , 4

     [,1] [,2] [,3]
[1,]    1   32   19
[2,]    3    4   23
[3,]    4    7   43
```

See how we now have four versions of the matrix we requested based on our two vectors? If we want to name our rows, columns and matrices, we can create vectors for those and then feed them into the `dimnames` (dimension names) argument of `array()`. The `dimnames` argument only takes lists, so we'll stick our name vectors into a list.

```
# Create row names.
rownames <- c("Row 1",
              "Row 2",
              "Row 3")

# Create column names.
columnnames <- c("Column 1",
                 "Column 2",
                 "Column 3")

# Create matrix names.
matrixnames <- c("Matrix Alpha",
                 "Matrix Beta",
                 "Matrix Gamma",
                 "Matrix Delta")

# Rename the array elements.

array1 <- array(c(vector1, vector2),
                dim = c(3,3,4),
                dimnames = list(rownames,
                                columnnames,
                                matrixnames)
```

```
                )

# And Voila!
print(array1)
```

```
, , Matrix Alpha

      Column 1 Column 2 Column 3
Row 1        1       32       19
Row 2        3        4       23
Row 3        4        7       43

, , Matrix Beta

      Column 1 Column 2 Column 3
Row 1        1       32       19
Row 2        3        4       23
Row 3        4        7       43

, , Matrix Gamma

      Column 1 Column 2 Column 3
Row 1        1       32       19
Row 2        3        4       23
Row 3        4        7       43

, , Matrix Delta

      Column 1 Column 2 Column 3
Row 1        1       32       19
Row 2        3        4       23
Row 3        4        7       43
```

While more could be said about arrays, we will end here since we really don't need to worry about them in this course.

4.6 Factors

Factors are data structures used to store categorical variables in R. Categorical variables have a fixed set of possible values. Factors are *not* vectors, so they're a pretty unique class. They

have their own unique class, and can be ordered if we have an ordered categorical variable (i.e. a categorical variable with a logical and explicit order, such as finishes in a race that correspond to 'first', 'second', etc.).

For example, let's say we have a categorical variable related to the temperature of cooked meat called `done` which has five possible values (or *levels*) - *rare*, *medium rare*, *medium*, *medium well*, and *well done*. Often, we are faced with a character vector which we want to convert to factor, which can be accomplished with the `factor()` function, in which the first argument is the variable or vector to be converted. Remember, you can always use the `class()` function to check if you converted the vector or variable to the correct class.

```
# Let's create the done variable with five levels.
done <- c("rare", "medium rare", "medium", "medium well", "well done")

class(done)
```

```
[1] "character"
```

```
# Now, let's convert this character vector to factor.

done2 <- factor(done)

class(done2)
```

```
[1] "factor"
```

If we want to look at the levels in our factor variable, we can simply put the variable or vector name in parentheses within the `levels()` function. If we don't like the order in which the levels are sorted, we can reorder them by inputting a vector of strings corresponding to the order we want as the second argument of the `factor()` function.

```
# Let's examine the levels of our factor variable.
levels(done2)
```

```
[1] "medium"      "medium rare" "medium well" "rare"        "well done"
```

```
# That's not the right order! Let's reorder the levels so they go in ascending order to te

donelevels <- c("rare", "medium rare", "medium", "medium well", "well done")
```

```
done2 <- factor(done2, levels = donelevels)

# Let's check if it worked.
levels(done2)
```

```
[1] "rare"        "medium rare" "medium"      "medium well" "well done"
```

Factor levels like these have a natural order to them. In this case, the natural order is governed by increasing temperature of the meat. So we might want to properly cast them as ordered factors using the `ordered` argument in factor, which takes a logical value (i.e. `TRUE` if you want it ordered and `FALSE` if you don't).

```
# Let's create an ordered factor variable and check that it worked.
done3 <- factor(done2,
                levels = donelevels,
                ordered = TRUE)

class(done3)
```

```
[1] "ordered" "factor"
```

```
# We can also check that the levels are correctly ordered with the levels() function. We c
print(done3)
```

```
[1] rare        medium rare medium      medium well well done
Levels: rare < medium rare < medium < medium well < well done
```

Now, let's say we want to change the levels of our factor variable. Perhaps, I want the factor levels to reflect my personal meat temperature preference of *medium rare* or not. So I'll code *medium rare* as `Delicious` and everything else as `Gross`. We can use the `recode()` function from the `dplyr` package to accomplish this (more on `dplyr` later in Chapter 9).

```
library(dplyr)

done4 <- recode(done3,
                rare = "Gross",
                "medium rare" = "Delicious",
                medium = "Gross",
                "medium well" = "Gross",
```

```
                "well done" = "Gross")

print(done4)
```

```
[1] Gross     Delicious Gross     Gross     Gross
Levels: Gross < Delicious
```

4.7 Exercises

As always, it's a good idea to attempt these while the material is still fresh. You can find the answers in Appendix B.

1. Create a numeric vector called `vec1` comprising the elements `3, -12, 532, 0, -100, 55,` and `-42`. Then, find the median and lowest value in the vector.
2. Create a new vector `vec2` which contains all the elements of `vec1` that are positive integers. Then create a new vector `vec3` that contains all the elements of `vec1` that are negative integers. Then create a new vector `vec4` which is the sum of `vec2` and `vec3`.
3. Assign the built-in dataframe `mtcars` to an object named after your favorite animal. Then calculate and print the median of the variable `mpg`.
4. Create a new variable and add to the object you created above (named after your favorite animal). This variable will take the variable Miles per Gallon `mpg` and convert it to Kilometers per Liter, which you will name `kpl`. This can be accomplished by taking `mpg` and dividing it by 2.352. Once you've created this new variable and added it to the object, calculate the median of `kpl`.

5. Assign the built-in dataframe `OrchardSprays` to an object name of your choice. Then, convert the variable `treatment` to an ordered factor variable, and change the existing names of factor levels from A:H to a list of sulphur levels from `Sulpher_8` to `Sulpher_1`. Then print the dataframe.

5 Strings & Comparisons

5.1 Strings

As we discussed in Chapter 4, character values are stored in objects known as *strings* in R. Let's go over a few key things with strings.

First, let's remember that string values are surrounded by quotes, such as `x <- "Hello, World"`. These can also be single quotes, such as `y <- 'Hello, World'`. However, you CANNOT combine double quotes on one end of the value and single quotes on the other, such as `z <- 'Hello, World"`. So, make sure you are consistent. I recommend using double quotes consistently, as R will always print and store the value with double quotes, even if you store the variable using single quotes.

What if you want to store quote within a quote? In this case, you can use the standard grammar rules of American English. According to Brittney Ross from *grammarly*, in American English, we use double quotation marks for quotes, and single quotation marks for quotes within quotes. Here's an example.

```
a <- "Invoking the Bard, she replied 'To thine own self be true.'"
a
```

```
[1] "Invoking the Bard, she replied 'To thine own self be true.'"
```

Notice that simply running the object name returns its value, in the same way that `print()`does. We can check the length of a string with the `nchar()` function.

```
nchar(a)
```

```
[1] 59
```

We can also check if a character or sequence of characters exist in a given string using the `grepl()` function, in which the first argument is the character/sequence of interest, and the second argument is the string. Evaluating this expression returns a logical value.

```
grepl("Bard", a)
```

```
[1] TRUE
```

```
grepl("Z", a)
```

```
[1] FALSE
```

To combine strings, we can use the `paste()` function, where the arguments are the string objects to be combined. This is also called *concatenating* or *merging* multiple strings.

```
# Let's create two new string objects b and c.

b <- "Demurely and without hesitation, I invoked Jonson 'There is no greater hell than to

c <- "That ended the conversation rather quickly."

# Now we combine the three

d <- paste(a,b,c)

d
```

```
[1] "Invoking the Bard, she replied 'To thine own self be true.' Demurely and without hesita
```

Note that the `paste()` function concatenates strings with a space by default. If we don't want spaces by default, we can use the `paste0()` function which does not separate strings by spaces, by default.

```
paste0("Remove", "All", "Spaces", "Now", "!", "!", "!")
```

```
[1] "RemoveAllSpacesNow!!!"
```

The `stringr` package also has a number of useful functions for manipulating strings. One thing I find especially helpful in this package is the ability to convert the characters in a string to lowercase, uppercase, or title case. This is especially helpful if you have values with inconsistent punctuation. Here's an example.

```
# A string with inconsistent punctuation
library(stringr)
```

```
Warning: package 'stringr' was built under R version 4.2.2
```

```
e <- "WOW, tHiS Is qUiTE a mESsY oNe."

# Let's str_to_lower() function to make all characters lowercase.

f <- str_to_lower(e)
f
```

```
[1] "wow, this is quite a messy one."
```

```
# Now we use the str_to_upper() function to make all characters uppercase.

g <- str_to_upper(f)
g
```

```
[1] "WOW, THIS IS QUITE A MESSY ONE."
```

```r
# Finally, we use the str_to_title() function to make all characters title case.

h <- str_to_title(g)
h
```

```
[1] "Wow, This Is Quite A Messy One."
```

Sometimes you have a string or vector of strings for which you want to apply a broad-sweeping change. This commonly happens with variable names or values, and sometimes, you want to write code to make changes to a number of values without changing the values one-by-one. In this case, the `sub()` and `gsub()` functions are very useful. These functions take the first argument as the pattern to match, the second as the thing you want to replace the pattern with, and the third is the dataframe or vector you are modifying. The main difference between `sub()` and `gsub()` is that `sub()` only modifies the first match in an individual string or vector, whereas `gsub()` modifies all matches in a particular string or vector. I tend to use `gsub()` more often than `sub()`, but use what works for you. Let's see how it works.

```r
# First let's create a vector of strings.

intro <- c("Hello, my name is Jamal.")
print(intro)
```

```
[1] "Hello, my name is Jamal."
```

```r
# Let's say I want to substitute another name for Jamal here. The gsub() function helps me

gsub('Jamal', "Monica", intro)
```

```
[1] "Hello, my name is Monica."
```

```r
# If you want to add a prefix to a vector of strings based on a particular pattern, you ca

gsub(".*^", "Welcome and ", intro)
```

```
[1] "Welcome and Hello, my name is Jamal."
```

```r
# Finally, let's work with a dataframe.

df <- data.frame(id = 1:4,
                 gender = c("male", "female", "transgender", "non-binary"),
                 state = c("California", "Pennsylvania", "New York", "Georgia")
                 )
head(df)
```

```
  id      gender        state
1  1        male   California
2  2      female Pennsylvania
3  3 transgender     New York
4  4  non-binary      Georgia
```

```r
# Let's change all the gender values to abbreviations.

df$gender <- gsub("female", "F", df$gender)
df$gender <- gsub("male", "M", df$gender)
df$gender <- gsub("transgender", "T", df$gender)
df$gender <- gsub("non-binary", "NB", df$gender)

head(df)
```

```
  id gender        state
1  1      M   California
2  2      F Pennsylvania
3  3      T     New York
4  4     NB      Georgia
```

```r
# Looks good. Now let's change all the state names to their abbreviations.

df$state <- gsub("California", "CA", df$state)
df$state <- gsub("Pennsylvania", "PA", df$state)
df$state <- gsub("New York", "NY", df$state)
df$state <- gsub("Georgia", "GA", df$state)

head(df)
```

```
  id gender state
```

```
1  1      M    CA
2  2      F    PA
3  3      T    NY
4  4     NB    GA
```

```
# You can also substitute a blank space for any value with gsub()

df$state <- gsub("CA", " ", df$state)
df$state
```

```
[1] " "  "PA" "NY" "GA"
```

When it comes to substituting strings, it's better to test replacement code on a smaller set of values before expanding to all the values you want to modify. This will avoid errors.

5.2 Comparisons

Very often we are interested in comparing two quantities in R. For instance, we may compare a variable to a value, or two variables to one another. We can also use Boolean operators in constructing our comparisons. Boolean operators are words like *AND*, *NOT*, and *OR* that are used as conjunctions commonly to create advanced search terms. However, we can also use them in creating comparisons.

5.2.1 Comparison Operators

To start, let us remind ourselves of common operators used in comparisons, as well as the three common Boolean operators, seen in Table 5.1.

Table 5.1: Operators used for comparisons in R.

Operator (R code)	Description
==	Equals to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!	NOT
|	OR
&	AND

You can use these to compare values or objects. Comparisons result in a logical value being returned.

```
# Comparing values

12 > 4
```

```
[1] TRUE
```

```
4.32 == (16.03/4)
```

```
[1] FALSE
```

```
pi^pi <= (4 * pi) / (pi + 4)
```

```
[1] FALSE
```

```
# Comparing objects

a <- 453

b <- 4 * 23

a >= b
```

```
[1] TRUE
```

```
## The objects being can be vectors of the same length. This will return a vector of logic

c <- mtcars$mpg > mtcars$wt
print(c)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
[31] TRUE TRUE
```

```
# Booleans can be used to create multiple conditions for the comparison.

d <- 500
e <- 50

# This evaluates to FALSE becase d > 100.
d < 100 & e < 100
```

```
[1] FALSE
```

```
# This evaluates to TRUE because at least one of {d,e} < 100.
d < 100 | e < 100
```

```
[1] TRUE
```

```
# This evaluates to TRUE because d > e.

d != e
```

```
[1] TRUE
```

5.2.2 Comparisons in Conditional Statements

Comparisons are commonly used within `if` and `if else` statements. These are conditional statements that specify a condition that must be satisfied, and then list rules for what happens if the condition is satisfied and what happens when it is not satisfied. The logic of the `if else` statement in show in Figure 5.1.

Let's have a look at some simple If Else statements involving comparisons. Note that the If and Else blocks are surrounded by curly braces `{ }`.

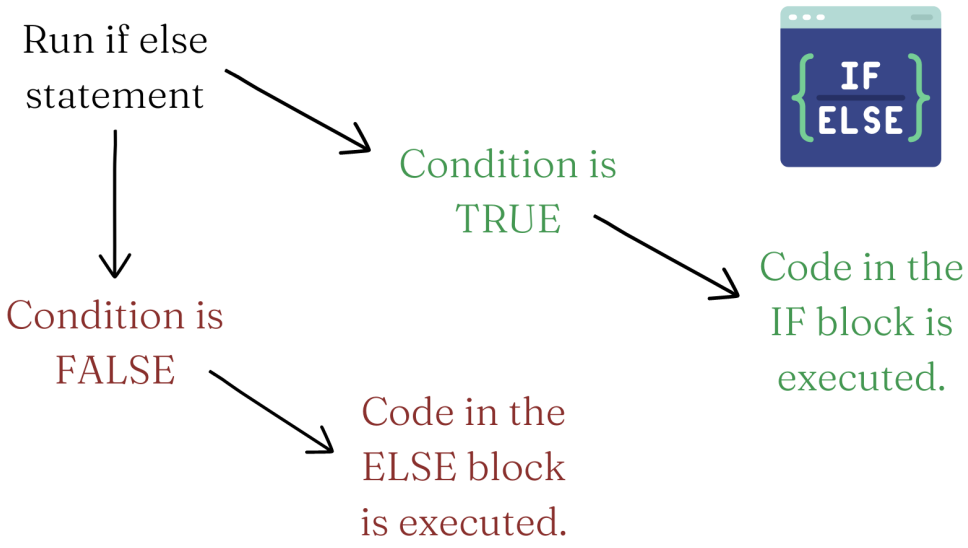


Figure 5.1: The simple and elegant logic of the If Else statement.

```
# First, we create some variables.

f <- 25
g <- 50
h <- 10

# Now let's try a simple If Else statement comparing the variables.

if (g < f) {                    # Here comes the If block
  print("g is less than f")
} else {                        # Here comes the Else block
  print("g is greater than f")
}
```

```
[1] "g is greater than f"
```

```
if (g < h) {                    # Here comes the If block
  print("g is less than h")
} else {                        # Here comes the Else block
```

```
  print("g is not less than h")
}
```

```
[1] "g is not less than h"
```

We can also add more conditions with the `else if` statement, which basically tells R "If the previous condition is not true, here's another condition with a rule." Let's add an Else If statement to the example above.

```
# First, we create some variables.

f <- 25
g <- 50
i <- 50

# Now we add some Else If statements to our IF statement.

if (g < i) {
  print("g is less than i")
} else if (g > i) {
  print("g is greater than i")
} else if (g == i) {
  print("g is equal to i")
}
```

```
[1] "g is equal to i"
```

```
# You might say that the third Else If condition above can just be an Else statement, sinc

if (g < i) {
  print("g is less than i")
} else if (g > i) {
  print("g is greater than i")
} else {
  print("g is equal to i")
}
```

```
[1] "g is equal to i"
```

We can also add the AND operator `&` and OR operator `|` to the conditional statements.

```r
# Let's create some variables.

j <- 100
k <- 200
l <- 150

# Now we use & (AND) in our if else statement.

if (j < l & j < k) {
  print("j is the lowest")
} else if (j > l & j > k) {
  print("j is the highest")
} else {
  print("j is the middle value")
}
```

```
[1] "j is the lowest"
```

```r
# Now we use | (OR) in our if else statement.

if (l < k | l < j) {
  print("l is less than at least one other value")
} else {
  print("l is the highest")
}
```

```
[1] "l is less than at least one other value"
```

That about wraps it up for Strings and Comparisons. As always, don't forget to…

5.3 Exercises

As always, it's a good idea to attempt these while the material is still fresh. You can find the answers in Appendix C.

1. Create a variable called `MarySue1` with the value `"Dr Mary Sue Coleman, former president of the University of Michigan once said"`. Then, create another variable called `MarySue2` with the value `"For today, goodbye. For tomorrow, good luck. And Forever, Go Blue!"`. Then find the number of characters in each variable using the `nchar()` function. Then, check if the letter 'r' is present in each variable, and report the results.
2. Create a variable called `MarySue3` whose value is a concatenation (combination) of `MarySue1` and `MarySue2`. Then, print the value for `MarySue3`.
3. Create a string vector called `basho` and assign it the value `"An old silent pond. A frog jumps into the pond-Splash! Silence again."` Then create and print another variable called `basho2` in which the word 'frog' has been replaced by 'buffalo', and the word 'Splash!' has been replaced by 'Yikes!'.
4. Let's do a variation of Mad Libs I will call Mad Sentences. Install and load the `keyToEnglish` package (be mindful of the capitalization in this package's name). Then

create three variables named after your three favorite cuisines. For each variable, assign the value `generate_random_sentences(n = 2, punctuate = TRUE)` to generate two random sentences per variable. This will produce a total of six sentences (two per variable). Finally, create a variable called `madsentences` whose value combines (pastes) all three variables. Print `madsentences`. If it sounds nonsensical, then it worked!

5. Let's compare the returns from simple vs compound interest after five years. First, define `p` as `1000`, `r` as `.07` and `t` as 5. Then Create a variable called `simple` with the value `p * r * t`. Next, create a variable `compound` with the value `p * (1 + r)^t - p`. Then, perform a logical test to see if `simple` is equal to `compound`, and write out the results of the test in one sentence.

6. Retain the variables you created above and write a series of conditional (If else/Else If) statements according to the following rules: 1) If `simple` is less than `compound`, print the statement "Simple interest is less than compound interest"; 2) If `simple` is greater `compound`, print the statement "Simple interest is greater than compound interest"; 3) If `simple` is equal to `compound`, print the statement "Simple interest is equal to compound interest".

6 Importing Data

6.1 Importing a CSV or Excel file

People often store data in a spreadsheet. That makes a lot of sense since a spreadsheet has rows and columns, and that's how most of our data in social science is going to be organized. It's very easy to import spreadsheet data into R.

To begin, let's be clear on the difference between **Comma Separated Values (CSV)** files and Microsoft's proprietary **Excel file formats (xls or xlsx)**. CSV files are a plain text format which separates a series of values with commas. Plain text means it cannot store formatting, macros, formulas, or other things that you might see in other file formats. You can open a CSV file in Microsoft Excel, but you cannot open an Excel file in a text editor. So, in general CSV files are probably the way to go in terms of storing data since they can be easily opened and manipulated, but Excel files are pretty great because they contain meta-data files, which can help detect instances of data tampering!

Let's go ahead and import the data using the `read.csv()` function, and assign the dataframe to an object called `mydata1`.

```
# This is how to import a CSV file from a URL
mydata1 <- read.csv("https://data.kingcounty.gov/api/views/yaai-7frk/rows.csv?accessType=D

# To import a CSV file that you have downloaded, just put the path in where the URL went a
# If you're on Windows like me, make sure the backslahes are doubled up like this \\.
# You can also change them to single forward slashes /.

mydata1 <- read.csv("~/R Book/Datasets/Lost__found__adoptable_pets.csv")
```

Let's start out by getting a sense of the scale of the dataframe using the dimensions `dim()` function. Then, let's check very quickly that the data look ok (i.e. there arent's any weird anomalies that jump out) by using the `View()` function.

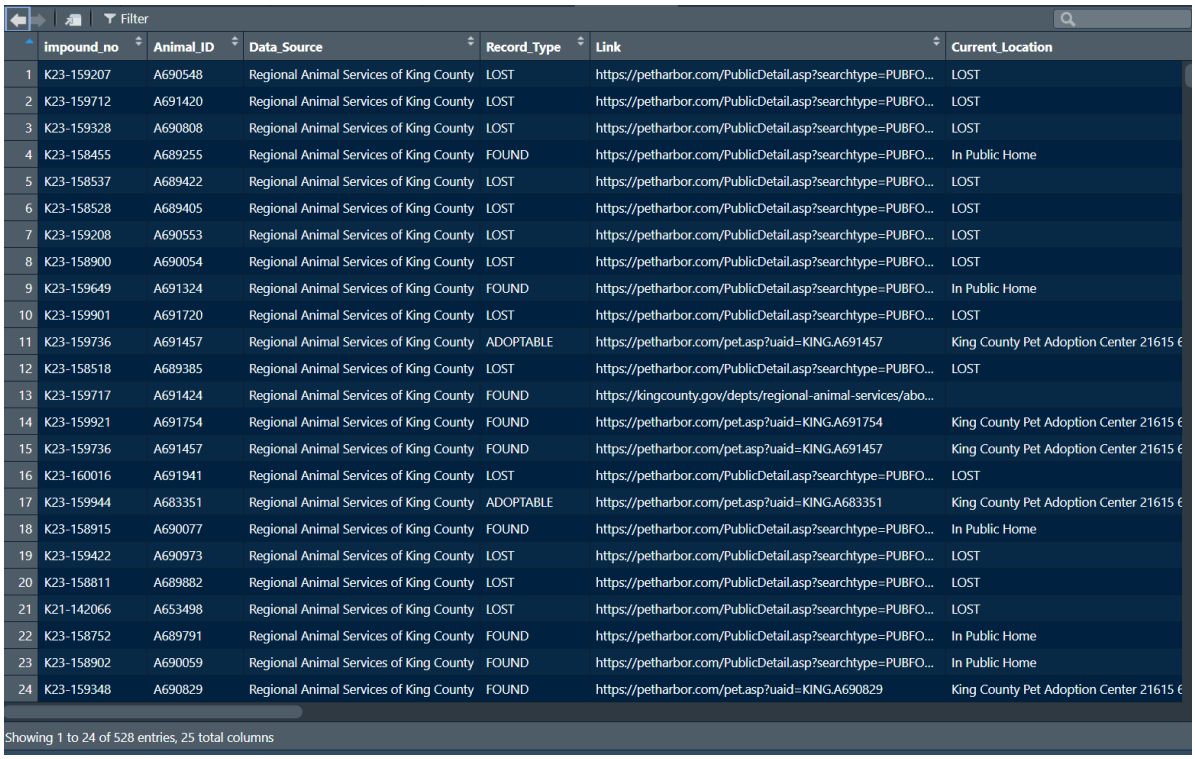
```
dim(mydata1)
```

```
[1] 540  25
```

Using the `dim()` function, we can see that there are 528 rows and 25 columns in the dataframe. Each row corresponds to a unit of observation, and each column corresponds to a different variable. So, this dataframe has 528 units of observation, and 25 variables.

Cool, now let's take a closer look at the data with the `View()` function.

```
View(mydata1)
```



Filter

	impound_no	Animal_ID	Data_Source	Record_Type	Link	Current_Location
1	K23-159207	A690548	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
2	K23-159712	A691420	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
3	K23-159328	A690808	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
4	K23-158455	A689255	Regional Animal Services of King County	FOUND	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	In Public Home
5	K23-158537	A689422	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
6	K23-158528	A689405	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
7	K23-159208	A690553	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
8	K23-158900	A690054	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
9	K23-159649	A691324	Regional Animal Services of King County	FOUND	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	In Public Home
10	K23-159901	A691720	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
11	K23-159736	A691457	Regional Animal Services of King County	ADOPTABLE	https://petharbor.com/pet.asp?uaid=KING.A691457	King County Pet Adoption Center 21615 €
12	K23-158518	A689385	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
13	K23-159717	A691424	Regional Animal Services of King County	FOUND	https://kingcounty.gov/depts/regional-animal-services/abo...	
14	K23-159921	A691754	Regional Animal Services of King County	FOUND	https://petharbor.com/pet.asp?uaid=KING.A691754	King County Pet Adoption Center 21615 €
15	K23-159736	A691457	Regional Animal Services of King County	FOUND	https://petharbor.com/pet.asp?uaid=KING.A691457	King County Pet Adoption Center 21615 €
16	K23-160016	A691941	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
17	K23-159944	A683351	Regional Animal Services of King County	ADOPTABLE	https://petharbor.com/pet.asp?uaid=KING.A683351	King County Pet Adoption Center 21615 €
18	K23-158915	A690077	Regional Animal Services of King County	FOUND	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	In Public Home
19	K23-159422	A690973	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
20	K23-158811	A689882	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
21	K21-142066	A653498	Regional Animal Services of King County	LOST	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	LOST
22	K23-158752	A689791	Regional Animal Services of King County	FOUND	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	In Public Home
23	K23-158902	A690059	Regional Animal Services of King County	FOUND	https://petharbor.com/PublicDetail.asp?searchtype=PUBFO...	In Public Home
24	K23-159348	A690829	Regional Animal Services of King County	FOUND	https://petharbor.com/pet.asp?uaid=KING.A690829	King County Pet Adoption Center 21615 €

Showing 1 to 24 of 528 entries, 25 total columns

Ok, it looks like we have a variables corresponding to whether the dog is lost, found, or adoptable (`Record_type`), the specific ID of the animal (`Animal_ID`), it's current location, and several others. It's always a good idea to have a look at the data to see if anything looks amiss. We'll get more into some specifics of data cleaning a bit later, but for now, it's worthwhile just having a look.

You can import an Excel file easily into R using the `readxl` package. The corresponding `read_xls()` and `read_xlsx()` functions will help you import .xls (older Excel files) and .xlsx files, respectively. The main argument is the file path.

```
library(readxl)

# Read a Microsoft Excel file.
```

```
df1 <- read_xlsx("~/R Book/Datasets/Lost__found__adoptable_pets_excel.xlsx")
```

6.2 Importing a text file

To import a text file (a file with the extension .txt), we first need to know how data are separated. Usually, this is done by tabs (think of the 'tab' key on your keyboard). In such cases, the `read.table()` function is useful where the arguments should be the URL or file path, followed by the type of separator using the `sep =` argument. For tabs, it should be `sep = '\t'`. For comma-separated data, it should be `sep = ','` and for period-separated data, it should be `sep = '.'`. Since, this data frame is tab-separated (I know because I saved it as such), we will use the tab separator.

```
# Read a tab-separated text file.
mydata2 <- read.table("~/R Book/Datasets/zoo.txt", sep = '\t')

dim(mydata2)
```

```
[1] 44 18
```

Using the `dim()` function, we can see that this dataframe has 44 units of observation and 18 variables. While we can always use the `View()` function to have a look at the data, let's say that we're in a real hurry and we just want to look at the first 10 rows of the data. Recall, that we can use `head()` function, and specify `n = 10` to look at the first 10 rows.

```
head(mydata2, n = 10)
```

```
           V1   V2       V3   V4   V5       V6      V7       V8      V9
1  animal_name hair feathers eggs milk airborne aquatic predator toothed
2       turtle    0        0    1    0        0       1        0       0
3    chameleon    0        0    1    0        0       0        0       1
4       iguana    0        0    1    0        0       0        1       1
5       lizard    0        0    1    0        0       0        1       1
6        gecko    0        0    1    0        0       0        0       1
7       python    0        0    1    0        0       0        1       1
8          boa    0        0    1    0        0       0        1       1
9        adder    0        0    1    0        0       0        1       1
10   crocodile    0        0    1    0        0       1        1       1
       V10      V11      V12  V13  V14  V15      V16     V17        V18
```

```
1  backbone breathes venomous fins legs tail domestic catsize class_type
2         1        1        0    0    4    1        1       1          3
3         1        1        0    0    4    1        1       0          3
4         1        1        0    0    4    1        1       1          3
5         1        1        0    0    4    1        0       0          3
6         1        1        0    0    4    1        1       0          3
7         1        1        1    0    0    1        0       1          3
8         1        1        0    0    0    1        0       1          3
9         1        1        1    0    0    1        0       1          3
10        1        1        0    0    4    1        0       1          3
```

Uh oh, it looks like the first row of data is the name of the variables! That's no good. The solution is simply to include the argument `header = TRUE` in the `read.table()` function. So let's try that data import of a text file again with the new argument.

```
# Read a tab-separated text file and keep the headers.
mydata2 <- read.table("~/R Book/Datasets/zoo.txt", sep = '\t', header = TRUE)

dim(mydata2)
```

```
[1] 43 18
```

The `dim()` function now shows me the dataframe has one less row than before (44 to 43). That's good! Now, let's use the `head()` function to check if the variable names are correctly excluded from our rows.

```
head(mydata2, n = 10)
```

```
   animal_name hair feathers eggs milk airborne aquatic predator toothed
1       turtle    0        0    1    0        0       1        0       0
2    chameleon    0        0    1    0        0       0        0       1
3       iguana    0        0    1    0        0       0        1       1
4       lizard    0        0    1    0        0       0        1       1
5        gecko    0        0    1    0        0       0        0       1
6       python    0        0    1    0        0       0        1       1
7          boa    0        0    1    0        0       0        1       1
8        adder    0        0    1    0        0       0        1       1
9    crocodile    0        0    1    0        0       1        1       1
10   alligator    0        0    1    0        0       1        1       1
   backbone breathes venomous fins legs tail domestic catsize class_type
```

1	1	1	0	0	4	1	1	1	3
2	1	1	0	0	4	1	1	0	3
3	1	1	0	0	4	1	1	1	3
4	1	1	0	0	4	1	0	0	3
5	1	1	0	0	4	1	1	0	3
6	1	1	1	0	0	1	0	1	3
7	1	1	0	0	0	1	0	1	3
8	1	1	1	0	0	1	0	1	3
9	1	1	0	0	4	1	0	1	3
10	1	1	0	0	4	1	0	1	3

That's better! Always be mindful of the "header problem" wherein the headers get mistakenly included as observational units, rather than column labels.

6.3 Importing a Stata, SAS, or SPSS file

Recall from Table 3.1 that Stata, SAS, and SPSS are different proprietary programs for statistical analysis. Stata is very commonly used by economists, and others in the social sciences. SAS is commonly used by healthcare and goverment agencies. SPSS is a commonly used by psychologists and other social scientists. Sometimes you might be working with someone who uses one of these software programs, or need to import a dataset from one of these programs.

First, it's important to know the file extension for each software program. Once you know the appropriate format, the `haven` package will allow us to import data from SAS, SPSS, or Stata into R. That's pretty cool and easy! These can be found in Table 6.1

Table 6.1: Data file extensions for Stata, SAS, or SPSS and the relevant *haven* package function.

Software	File Extension	Relevant *haven* package function
SPSS	.sav	`read_sav()`
SAS	.sas7bdat or .xpt	`read_sas()` or `read_xpt()`
Stata	.dta	`read_dta()`

Let's give each of these a shot with the relevant data format. Remember to install the `haven` package if you haven't done that already with `install.packages("haven")` and then load the package using `library(haven)`.

```
library(haven)

# Read a sas7bdat file.
```

```
mydata3 <- read_sas("~/R Book/Datasets/naws_all.sas7bdat")
```

When a SAS data file is exported, you may see the file extension .xpt. As seen in Table 6.1, this is not a problem for the formidable **haven** package. Let's import an XPT file now.

```
library(haven)

# Read a SAS xpt file.
mydata4 <- read_xpt("~/R Book/Datasets/P_DEMO.xpt")

dim(mydata4)
```

```
[1] 15560    29
```

We can see that the data frame has 15,560 observational units and 29 columns or variables. Let's have a look at the last 10 rows of data and first 6 columns.

```
tail(mydata4[, c(1:6)], n = 10)
```

```
# A tibble: 10 x 6
     SEQN SDDSRVYR RIDSTATR RIAGENDR RIDAGEYR RIDAGEMN
    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
 1 124813       66        2        2       43       NA
 2 124814       66        2        1       64       NA
 3 124815       66        2        1       52       NA
 4 124816       66        2        1        1       15
 5 124817       66        2        2       67       NA
 6 124818       66        2        1       40       NA
 7 124819       66        2        1        2       NA
 8 124820       66        2        2        7       NA
 9 124821       66        2        1       63       NA
10 124822       66        2        1       74       NA
```

We can see a number of variables with values, and things seem to have imported nicely. We can't know this for sure until we've done some more careful examination and cleaning of the data, but the fact that nothing looks terribly out of place at first glance is a good sign.

You will also notice that when we examined the first and last 10 rows and first six columns, R gave us a **tibble** of dimensions 10 * 6. A tibble is basically a well-formatted and easy to understand summary of the data in R. Tibbles print data out nicely so they're easy to glance at.

Now you know how to import a number of common data types into R! So, why not…

Figure 6.1: Manx cat has public use dataset suggestions for you if you just click on him. Credit to the University of Missouri Libraries.

7 Merging (Joining) and Reshaping Data

The data analyst's task is often to combine multiple dataframes. We've already seen how to combine rows and columns with matrices in Section 4.2. Those of course are homogenous data structures (all elements have to be the same type). With dataframes we can have multiple types of data structures, and we commonly want to combine multiple dataframes. If two dataframes are already in the same order, and have the exact same number of rows, we can still use the `cbind()` function to append multiple columns. It's usually safer to merge dataframes based on a key (usually an ID variable), but if the dataframes are sorted in the same order, and have the same number of rows, go ahead and trying the `cbind()` function.

Usually, there is some link such as an ID number for each observational unit that links the dataframes, but not always. Terminologically, combining data is often called *merging, appending, or joining.* The latter is commonly used with other programming languages like SQL, so we'll stick to calling it joining or different types of data joins. Four commonly used types of data joins are presented in Figure 7.2.

7.1 Left and Right Joins

Let's look at an example of a left join using the `left_join()` function from the `dplyr` package. We'll look at two dataframes which both have the same ID variable. So, what we want is a dataframe that takes the columns in one dataframe and adds them to the other dataframe, linked by a common ID variable.

```
library(dplyr)
library(readxl)

df1 <- read_xlsx("Datasets/marketing1.xlsx")
df2 <- read_xlsx("Datasets/marketing2.xlsx")
df3 <- read_xlsx("Datasets/marketing3.xlsx")

head(df1, n = 5)
```

```
# A tibble: 5 x 5
     ID Year_Birth Education  Marital_Status Income
```

Figure 7.1: Which built-in R dataframe am I invoking with this image?

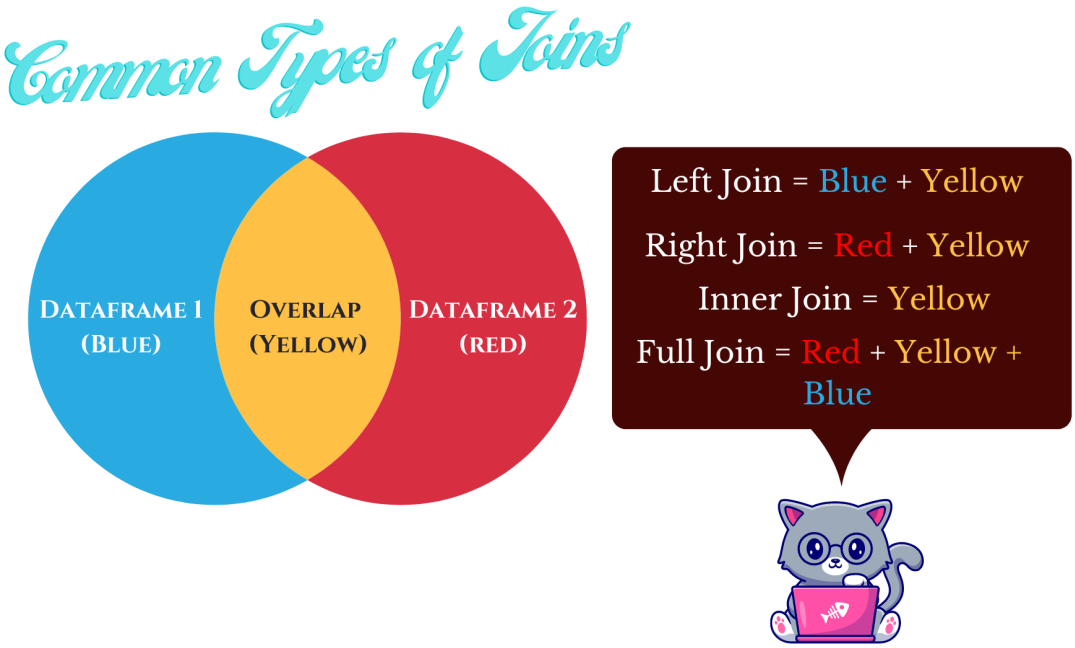


Figure 7.2: Four common data join types. There are others, but we'll leave those aside for now.

```
  <dbl>       <dbl> <chr>      <chr>           <dbl>
1  5524        1957 Graduation Single          58138
2  2174        1954 Graduation Single          46344
3  4141        1965 Graduation Together        71613
4  6182        1984 Graduation Together        26646
5  5324        1981 PhD        Married         58293
```

```
head(df2, n = 5)
```

```
# A tibble: 5 x 3
     ID MntWines MntFishProducts
  <dbl>    <dbl>            <dbl>
1  5524      635              172
2  2174       11                2
3  4141      426              111
4  6182       11               10
5  5324      173               46
```

```r
df4 <- df1 %>%
  left_join(df2,
            by = c("ID" = "ID")
            )

head(df4, n = 5)
```

```
# A tibble: 5 x 7
     ID Year_Birth Education  Marital_Status Income MntWines MntFishProducts
  <dbl>      <dbl> <chr>      <chr>           <dbl>    <dbl>            <dbl>
1  5524       1957 Graduation Single          58138      635              172
2  2174       1954 Graduation Single          46344       11                2
3  4141       1965 Graduation Together        71613      426              111
4  6182       1984 Graduation Together        26646       11               10
5  5324       1981 PhD        Married         58293      173               46
```

The `left_join()` function then returns all the rows from the first dataframe (`df1` in this case) that match up with the rows in the second dataframe (`df2` in this case). The `right_join()` function (also from the `dplyr` package) is the converse; it returns all the rows in the second dataframe (df2) that match up with the rows in first dataframe (`df1`). Functionally, it's the same as the `left_join()` function, but notice how the rows in the second dataframe are now ordered first.

```r
df5 <- df2 %>%
  right_join(df1,
             by = c("ID" = "ID")
             )

head(df5, n = 5)
```

```
# A tibble: 5 x 7
     ID MntWines MntFishProducts Year_Birth Education  Marital_Status Income
  <dbl>    <dbl>            <dbl>      <dbl> <chr>      <chr>           <dbl>
1  5524      635              172       1957 Graduation Single          58138
2  2174       11                2       1954 Graduation Single          46344
3  4141      426              111       1965 Graduation Together        71613
4  6182       11               10       1984 Graduation Together        26646
5  5324      173               46       1981 PhD        Married         58293
```

7.2 Inner and Full Joins

Let's look at an example of an inner join, where the overlap between the two dataframes is retained. Using the `inner_join()` function from the `dplyr` package, we can merge two dataframes that have some common information (based on an ID variable). However, all rows in the second dataframe that do not correspond with the first will be dropped.

To create a new dataframe `df6`, we will join two dataframes (`df3` and `df2`) that share the same nine rows (1:9), but `df3` has an additional 11 rows (10:20) which `df2` doesn't have. `df3` also has two columns that `df2` does not have. Let's see what happens when we merge them.

```
df6 <- df3 %>%
  inner_join(df2,
             by = c("ID" = "ID")
             )
```

	ID	MntWines.x	MntFishProducts.x	MntMeatProducts	MntFruits	MntWines.y	MntFishProducts.y
1	5524	635	172	546	88	635	172
2	2174	11	2	6	1	11	2
3	4141	426	111	127	49	426	111
4	6182	11	10	20	4	11	10
5	5324	173	46	118	43	173	46
6	7446	520	0	98	42	520	0
7	965	235	50	164	65	235	50
8	6177	76	3	56	10	76	3
9	4855	14	3	24	0	14	3

The merged dataframe df6 only retained the nine common rows between df2 and df3 (1:9). All other rows were deleted. Additionally, this function took all the columns in df2 and added them to df3, even if they are the same columns. In this case, R adds a little suffix to indicate the provenance of each column (.x and .y). This is not the most useful join, as you can see that it drops observations. However, it can be useful to examine the intersection of two dataframes.

Finally let's look at a full join which combines everything - the unique aspects of both dataframes as well as the overlap. We will make use of the `full_join()` function from the `dplyr` package.

```
df7 <- df3 %>%
  full_join(df2,
            by = c("ID" = "ID")
            )
```

	ID	MntWines.x	MntFishProducts.x	MntMeatProducts	MntFruits	MntWines.y	MntFishProducts.y
1	5524	635	172	546	88	635	172
2	2174	11	2	6	1	11	2
3	4141	426	111	127	49	426	111
4	6182	11	10	20	4	11	10
5	5324	173	46	118	43	173	46
6	7446	520	0	98	42	520	0
7	965	235	50	164	65	235	50
8	6177	76	3	56	10	76	3
9	4855	14	3	24	0	14	3
10	5899	28	0	6	0	*NA*	*NA*
11	1994	5	5	6	5	*NA*	*NA*
12	387	6	16	11	16	*NA*	*NA*
13	2125	194	61	480	61	*NA*	*NA*
14	8180	233	2	53	2	*NA*	*NA*
15	2569	3	14	17	14	*NA*	*NA*
16	2114	1006	22	115	22	*NA*	*NA*
17	9736	53	5	19	5	*NA*	*NA*
18	4939	84	5	38	5	*NA*	*NA*
19	6565	1012	80	498	80	*NA*	*NA*
20	2278	4	17	19	17	*NA*	*NA*

In contrast with the inner join, the full join has added all rows across both dataframes. It also added all columns with the appropriate `.x` or `.y` suffix to columns shared between dataframes. Given that it added all rows, naturally there will be some columns for which there are no values. For example, rows 10:20 are not present in `df2` and the variable `MntFishProducts` is present in `df2`. But after the merge, the values for `MntFishProducts` for rows 10:20 are missing `NA`.

7.3 Two Common Merging Scenarios

Practically, the data analyst typically faces two common data merging scenarios:

- We want to add new columns to the same observational units.
- We want to add new rows (also called *appending)* to the existing set of columns.

Let's have a look at how to deal with both these scenarios.

```
# This is a case where we want to add new columns to the same observational units. This le

df8 <- df1 %>%
```

```
  left_join(df2,
            by = c("ID" = "ID")
            )
```

	ID	Year_Birth	Education	Marital_Status	Income	MntWines	MntFishProducts
1	5524	1957	Graduation	Single	58138	635	172
2	2174	1954	Graduation	Single	46344	11	2
3	4141	1965	Graduation	Together	71613	426	111
4	6182	1984	Graduation	Together	26646	11	10
5	5324	1981	PhD	Married	58293	173	46
6	7446	1967	Master	Together	62513	520	0
7	965	1971	Graduation	Divorced	55635	235	50
8	6177	1985	PhD	Married	33454	76	3
9	4855	1974	PhD	Together	30351	14	3

Now, we have a merged dataframe where we have added two new columns (`MntWines` and `MntFishProducts` from `df2`) to `df1` and matched them based on `ID`. This is a scenario where you have, for example, multiple dataframes for same individuals.

Now, let's look at a situation where we want to append new rows to an existing dataframe. This is a scenario where, for example, you have the same exact set of variables but data for different people across dataframes. First, we'll modify our existing `df3` so that it contains the exact same three variables (`ID`, `MntWines`, and `MntFishProducts`) as `df2`, but just has different rows. We will then use the base R `rbind()` function to append rows from `df3` to `df2`.

```
# This will make df3 have the same variables as df2, but with different rows corresponding

df3 <- df3[-c(1:9), -c(4:5)]

# Now, we will merge df2 and df3 into a new dataframe df9. This will append rows from df3

df9 <- rbind(df2, df3)
```

	ID	MntWines	MntFishProducts
1	5524	635	172
2	2174	11	2
3	4141	426	111
4	6182	11	10
5	5324	173	46
6	7446	520	0
7	965	235	50
8	6177	76	3
9	4855	14	3
10	5899	28	0
11	1994	5	5
12	387	6	16
13	2125	194	61
14	8180	233	2
15	2569	3	14
16	2114	1006	22
17	9736	53	5
18	4939	84	5
19	6565	1012	80
20	2278	4	17

As we see from the screenshot above, we have now appended rows 10:20 from `df3` to rows 1:9 from `df2`.

7.4 Reshaping Data

When we think about collecting data about some phenomenon over time, or of taking multiple measurements from the same observational unit, the a dataframe comes in one of two different formats: *wide* or *long*. A wide dataframe means that an observational unit's repeated responses

are found on a single row, whereas a long dataframe means that each row corresponds to just one time point for our observational unit. Wide format might be easier to read when looking at change over time, but long format is often useful for particular statistical analyses. Thus, it's helpful to know how to *reshape* a dataframe from long to wide, and vice versa.

7.4.1 From long to wide

Let's look at an example with a dataframe `ncaabball` which contains information on certain NCAA Basketball team over a three-year period from 2015 to 2017.

```
ncaabball  <- read.csv("~/R Book/Datasets/ncaabball.csv",
                       header = TRUE)

head(ncaabball, n = 15)
```

```
       Team Wins Year
1  Illinois   19 2015
2  Illinois   15 2016
3  Illinois   20 2017
4   Indiana   18 2017
5   Indiana   20 2015
6   Indiana   27 2016
7      Iowa   19 2017
8      Iowa   22 2015
9      Iowa   22 2016
10 Maryland   28 2015
11 Maryland   24 2017
12 Maryland   26 2016
13 Michigan   15 2015
14 Michigan   22 2016
15 Michigan   26 2017
```

Currently, the dataframe is in long format, because the individual teams are listed on multiple rows. Let's say we want a wide format in which each team only gets one row, and there become three columns corresponding to each of the three years with values in those cells corresponding to the number of wins per year. Let's then reshape this dataframe from long to wide using the `pivot_longer()` function from the `tidyr` package, which is part of the `tidyverse`.

The `names_from` argument specifies which variable we want to make into separate columns. The `values_from` argument specifies what should go into the cells of our new set of columns. In this case, we want the values from the `Wins` column to be included in our new set of

columns. Finally, we leverage the `names_glue` argument to specify the naming for our new columns. Here, we want the Year value to be followed by a space, and then the word `Wins` which corresponds to our values variable.

```
library(tidyverse)

# Let's reshape from long to wide so each year is on one row.

ncaabball_wide <- ncaabball %>%
  pivot_wider(
    names_from = Year,
    values_from = Wins,
    names_glue = "{Year} {.value}")          ## This is saying "use the existing column nam

## You can also customize the column names with string text outside of curly braces.

ncaabball_wide2 <- ncaabball %>%
  pivot_wider(
    names_from = Year,
    values_from = Wins,
    names_glue = "Wins in {Year}")

# The dataframe is now in wide format with each year having its own column.

head(ncaabball_wide)
```

```
# A tibble: 5 x 4
  Team     `2015 Wins` `2016 Wins` `2017 Wins`
  <chr>          <int>       <int>       <int>
1 Illinois          19          15          20
2 Indiana           20          27          18
3 Iowa              22          22          19
4 Maryland          28          26          24
5 Michigan          15          22          26
```

```
head(ncaabball_wide2) ## Notice how the column names are different from ncaabball_wide.
```

```
# A tibble: 5 x 4
  Team     `Wins in 2015` `Wins in 2016` `Wins in 2017`
  <chr>             <int>          <int>          <int>
```

```
1 Illinois              19              15              20
2 Indiana               20              27              18
3 Iowa                  22              22              19
4 Maryland              28              26              24
5 Michigan              15              22              26
```

7.4.2 From wide to long

Alright, now let's say we want to go back to long from wide. What this means is that our three columns `2015 (Wins)`, `2016 (Wins)`, and `2017 (Wins)` will now become individual rows. They will now be in their own column, which we'll have to name. Then the values under `2015 (Wins)`, `2016 (Wins)`, and `2017 (Wins)` in the `ncaabball_wide` dataframe will have to be gathered and become rows in their own column, which we will also have to name. In summary, the wide dataframe with five rows and four columns will become a long datafame with 15 rows and three columns.

To accomplish this, we will use the `pivot_longer()` function also from the `tidyr` package. The first argument `cols` specifies the columns by name which will become individual rows. Next, the `names_to` argument specifies the name of the new column that will be created for our previous column variables. The `values_to` argument species the name of the new column under which we'll gather all the values from our previous columns (these correspond to game wins).

```
# Let's reshape from wide to long so each year gets its own row.

## Because we no longer want the word 'Wins' listed after the year, we'll use a specific s

ncaabball_long <- ncaabball_wide %>%
  pivot_longer(
    cols = c("2015 Wins", "2016 Wins", "2017 Wins"),
    names_to = "Year",
    names_pattern = "(....)",
    values_to = "Wins"
    )

head(ncaabball_long, n = 15)
```

```
# A tibble: 15 x 3
   Team     Year   Wins
   <chr>    <chr> <int>
 1 Illinois 2015     19
 2 Illinois 2016     15
```

```
 3 Illinois 2017     20
 4 Indiana  2015     20
 5 Indiana  2016     27
 6 Indiana  2017     18
 7 Iowa     2015     22
 8 Iowa     2016     22
 9 Iowa     2017     19
10 Maryland 2015     28
11 Maryland 2016     26
12 Maryland 2017     24
13 Michigan 2015     15
14 Michigan 2016     22
15 Michigan 2017     26
```

And just like that, we are back to long format using the `pivot_longer()` function. Remember, reshaping is mainly about readability and preparing the data for particular statistical procedures. Some procedures require the data to be in long format, while others require the data to be in wide format. Therefore, it's useful to know how to transition from one format to another.

7.5 Exercises

As always, it's a good idea to attempt these while the material is still fresh. You can find the answers in Appendix D.

1. Assign the built-in dataframe `Orange` to an object named whatever you want. This dataframe relates to the age and circumference of orange trees. First, with respect to the specific tree, is the dataframe in long or wide format? Please explain your answer.
2. Once again with respect to individual trees, reshape the dataframe. If you believe it is in long format, reshape to wide (hint: this is the correct answer). Assign the reshaped dataframe to a new object with a name that indicates the reshaped nature of the data (e.g., "tree_wide"). What you want to see is a dataframe where each row corresponds to age, and separate columns listing the circumference of each tree. Use an approach you have seen to name each column "Tree (number) circumference". For example, the first tree column should be named "Tree 1 circumference", the second should be called "Tree 2 circumference", and so on. Once you have finished, use the `head()` function to show the first five rows of the reshaped dataframe.
3. Let's go backwards. Take the wide dataframe from Question 2, and reshape into long format, assigning this to a new object with a name noting that the new dataframe is in long format (e.g. "tree_long"). Here, we want each row to correspond to a tree, and separate columns for age and circumference, labelled as such. Then, use a command that you've seen before to make the new tree column come first in order in the dataframe. Finally, use the `head()` to show the first five rows of the new dataframe.
4. Let's go back to the original `Orange` dataframe, or the object to which you initially assigned it. Next, we're going to create two new dataframes `tree2` and `tree3` using the code below. We're also going to create an ID variable for the two dataframes with the same rows in order to have unique indices for each row.

```
# I assign the Orange dataframe to an object called tree.
tree <- Orange

# I then create a vector of values which will populate a new column.
newage <- tree$circumference/2.5

# I then create a vector corresponding to values listed in the Tree column in the original
treeid <- tree$Tree

# I then create a new dataframe with two variables. It takes the original dataframe's Tree
tree2 <- data.frame(Tree = treeid,
                    "Age_years" = newage)
```

```
# I then create a new dataframe with three columns and seven rows. These will be appended
tree3 <- data.frame(Tree = c(rep(6, 7)),
                    "Age_years" = c(10.4,
                                    21.2,
                                    30.4,
                                    41.3,
                                    55.8,
                                    66.7,
                                    71.4),
                    ID = c(seq(from = 36, to = 42)
                           )
                    )

# I then make sure the Tree column in tree3 matches the class (ordered factor) of the Tree

tree3$Tree <- ordered(tree3$Tree)

# I then create a unique ID variable in the original dataframe and in tree2 which will be
tree$ID <- 1:nrow(tree)
tree2$ID <- 1:nrow(tree2)
```

5. Create a new dataframe `tree4` that merges `tree2` with the `Orange` dataframe or whatever object you assigned it to. Note, what we want here is to add a new column `Age (Years)` to the same rows or observational units. The `ID` variable we created above should be used to link the dataframes. Once the merge is complete, print the first six rows of the merged dataframe to make sure it worked.

6. Now create a new dataframe `tree5` which appends rows from `tree3` to the `tree2` dataframe we created in Question 2. Once the merge is complete, print the *last* seven rows of the merged dataframe to make sure it worked.

8 Data Cleaning

What does it mean to *clean* data? Raw data, or the data you obtain directly from measurement, and has yet to be processed and made *digestible* for human consumption. This is because raw data can missing data, contain errors, weird characters or symbols, personally identifiable information (information that can help identify an individual and break anonymity), or other things that you'd rather not have in there. Therefore, raw data needs to be cleaned (or cleansed or scrubbed) to proceed to data analysis, inference, visualization, and reporting.

There is no standard definition of data cleaning, but in general it means that you want to fix any issues in your data before you do anything else. This makes sense right? You don't want to analyze data which has a bunch of errors in it, as that could bias your results, create a misleading interpretation, and ultimately introduce noise and distortion into the scientific literature. When it comes data analysis, remember the acronym GIGO, which is explained in Figure 8.1.

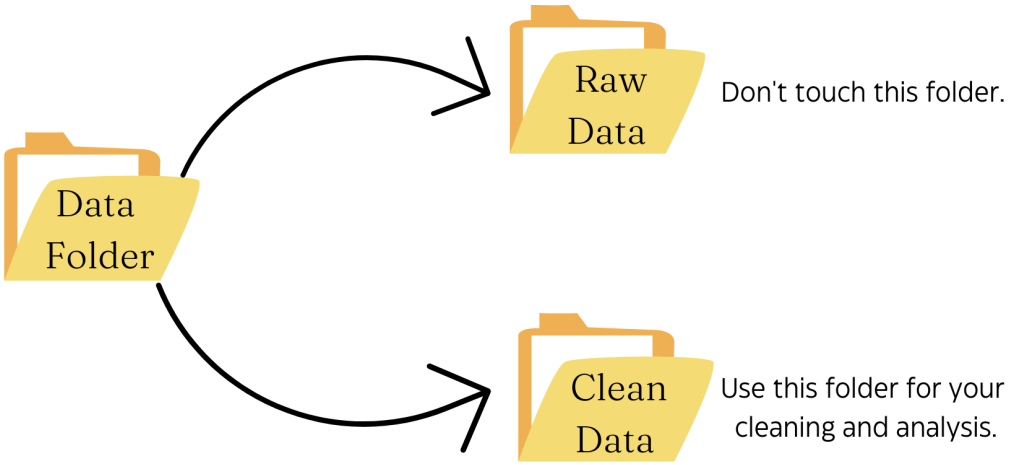
Let's now dig into some basic data cleaning issues.

8.1 Basic Folder Structure

There are several potential errors you can detect while looking at raw data. The first thing you should do is separate out raw data from the dataset you will be cleaning, and eventually produced a clean dataset. Leave the raw data in an encrypted folder or other secure location,

Figure 8.1: GIGO: Garbage In, Garbage Out. Your analyses are only as good as the quality of your data.

and work with just the cleaned data (or data that will be cleaned). This will prevent you from overwriting the original dataset and making any errors in data cleaning permanent.



Once this separation of raw and clean (or to be cleaned) data is established, you can begin working on cleaning.

8.2 The Pipe Operator (%>%)

This is a probably a good time to introduce one of the most useful collections of packages in R - the tidyverse!

According to its website:

> The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

It currently has about 30 packages which can all be installed simultaneously with `install.packages("tidyverse")`. Once you load the tidyverse, you can see a full list of its packages with the function `tidyverse_packages()`.

```
library(tidyverse)

tidyverse_packages()
```

```
 [1] "broom"         "conflicted"    "cli"           "dbplyr"
```

Figure 8.2: Source: www.tidyverse.org

```
 [5] "dplyr"         "dtplyr"        "forcats"       "ggplot2"
 [9] "googledrive"   "googlesheets4" "haven"         "hms"
[13] "httr"          "jsonlite"      "lubridate"     "magrittr"
[17] "modelr"        "pillar"        "purrr"         "ragg"
[21] "readr"         "readxl"        "reprex"        "rlang"
[25] "rstudioapi"    "rvest"         "stringr"       "tibble"
[29] "tidyr"         "xml2"          "tidyverse"
```

The tidyverse is a collection of useful tools that are commonly used by R users. In particular, the packages `dplyr`, and `ggplot2` are the stars of the show, and very useful for data wrangling and visualization, which is what we will focus on mainly in this book.

The pipe operator `%>%` is used to combine functions in a way that sets up a chain of operations. It was originally part of the `magrittr` package, and now comes standard with the `dplyr` package as well. The pipe is essential telling R "Take the output from the left side and pass it into the right side as the first argument." Put more simply, the pipe is saying "Take this [stuff on left] and put it through that [stuff on right]". This is important because it can make our code more efficient when we want to multiple things. Let's look at a simple example.

```
library(dplyr)

# First, let's randomly sample 10 numbers from a normal distribution.

x <- rnorm(10)
print(x)
```

```
[1] -0.72342147  0.85226177  0.11390658  0.44640477 -0.44549899 -0.02481577
[7] -0.01655658  0.01654529 -0.78527038  1.01613043
```

```
# Here's the slow way to 1) Add five to every number in the vector; 2) round the numbers t

## First, we add five to every number in the vector
x <- x + 5
print(x)
```

```
[1] 4.276579 5.852262 5.113907 5.446405 4.554501 4.975184 4.983443 5.016545
[9] 4.214730 6.016130
```

```
## Second, we round the vector to one decimal place.
x <- round(x, digits = 1)
```

```
print(x)
```

```
 [1] 4.3 5.9 5.1 5.4 4.6 5.0 5.0 5.0 4.2 6.0
```

```
## Third, we take the log of each number.
x <- log(x)
print(x)
```

```
 [1] 1.458615 1.774952 1.629241 1.686399 1.526056 1.609438 1.609438 1.609438
 [9] 1.435085 1.791759
```

```
## Take the mean of the resulting vector.
print(mean(x))
```

```
[1] 1.613042
```

```
# Let's now accomplish this the more efficient way with pipes!

y <- rnorm(10)
print(y)
```

```
 [1] -1.1091466 -0.2896161  0.7372577 -1.5568200  0.2410887 -0.2760042
 [7]  0.4232345  0.3116403 -2.3331220 -2.1451690
```

```
y %>%
  `+`(5) %>%              # Notice that arithmetic operators are surrounded by backticks `
  round(digits = 2) %>%   # Only the second argument is needed, since the first argument i
  log() %>%
  mean() %>%
  print()
```

```
[1] 1.448824
```

```
# The summary() function can also provide us with useful information very easily. This inf
y %>%
```

```
  summary()
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.3331 -1.4449 -0.2828 -0.5997  0.2940  0.7373
```

We can also use the pipe operator to manipulate dataframes, including operations with string vectors or columns. Importantly, note that the pipe operator puts the output from the left side as the first argument on the right side. However, this becomes a problem when we use a function whose first argument is not a dataframe or vector, such as the `gsub()` function. In such cases, you can put a period `.` in the place where the vector or dataframe should go. Let's look at an example.

```
# Let's create a small dataframe.

bball <- data.frame(team = c("San Antonio",
                             "San Antonio",
                             "New Orleans",
                             "Washington",
                             "Utah",
                             "Milwaukee",
                             "Cleveland"),
                    player_name = c("Romeo Langford",
                                    "Jakov Poeltl",
                                    "Dyson Daniels",
                                    "Will Barton",
                                    "Mike Conley",
                                    "Joe Ingles",
                                    "Raul Neto"),
                    points = c(2,
                               23,
                               35,
                               23,
                               8,
                               33,
                               28)
                    )

# Let's use the pipe operator to chain some functions together! The functions below change

bball$team %>%
  gsub("San Antonio", "San Antonio Spurs", .) %>%
```

```
    gsub("New Orleans", "New Orleans Pelicans", .) %>%
    gsub("Washington", "Washington Commanders", .) %>%
    gsub("Utah", "Utah Jazz", .) %>%
    gsub("Milwaukee", "Milwaukee Bucks", .) %>%
    gsub("Cleveland", "Cleveland Cavaliers", .) %>%
    unique()
```

```
[1] "San Antonio Spurs"     "New Orleans Pelicans"  "Washington Commanders"
[4] "Utah Jazz"             "Milwaukee Bucks"       "Cleveland Cavaliers"
```

8.3 Illogical Values & Typos

One of the first things I always do with a raw data set is to check if there are any illogical values. These are values that fall outside of the acceptable set of response options for a particular variable. For example, if a question is asking about a participant's age in years, a

value of 366 would be an illogical value since humans cannot yet live that long. However, a value of 366 for a variable about age in months makes perfect sense, as it translates to an age of about 30 years, which is a logical value. As another example, if a survey of undergraduate students conducted at Boston University asks them to state their class year (e.g. first year, freshman, second year, sophomore, etc), and you see some entries with the value "Boston University", this is an illogical value. It should either be modified to the correct class year, or deleted and left as missing data.

Commonly, typos can create illogical values. These should be fairly obvious if you a codebook, which is a document providing the name, description, and meaning of each variable, including how it is coded and what response options are valid. These could be incorrect or illogical entries, but could also be logically but with incorrect punctuation. For example, entries for a variable about one's favorite color may contain entries *red*, *RED*, *Red, or rED.* These are all logical values, but they differ in which letters are capitalized, potentially leading to issues in the analysis. Thus, they should all be standardized. I would change them all to *red.* There is no special reason for this particular format, other than consistency, but I have found that lowercase variable names introduce less potential for errors due to capitalization.

8.4 Duplicate Values & Outliers

Different participants can have the same response to particular questions or variables. However, if different participants have EXACTLY the same values across ALL variables, then you more than likely have a case of a duplicate value, which should be removed. Duplicate values are key sources of bias, and can inflate or mask detectable effects, leading to misleading interpretations. Sometimes duplicate values can arise as errors when multiple data sets are joined together. A simple approach to avoiding this problem is to count the rows and columns in each dataset before they are joined together, and then ensure the total rows and columns after joining corresponds to the sum of rows and columns from each individual dataset.

Outliers are values that are abnormally far away from other values in your dataset. Imagine you take the average net worth of five people sitting at a bar. Even though the average net worth of the first four people is $64,501, the average net worth of all five people at the bar comes to $ 58 billion! How is that possible? Because the fifth person at the bar is Bill Gates, whose 2023 net worth was $117.5 billion. Gates' net worth value is an outlier because 117.5 billion is VERY VERY far distant from $64,501.

For right now, the important thing to note is that **you should NOT remove outliers just because they are outliers.** Outliers should only be removed if they represent true data entry/collection errors. For example, if an adult's weight is entered as 12 pounds, this is likely a data entry error, but if their weight is listed as 432 pounds, this is a legitimately possible value, and if true, should not be removed. When outliers are legitimate extreme values, they represent a natural section of the population which you are studying. They are providing you information that should not be discarded simply to fit models easier, as this is a type of

cherry-picking of the data which is . Rather, one should use statistical tests that are robust to outliers in such cases.

Some people, such as professor of data science Pasquale Cirillo make a nice distinction between outliers (values that are not possible because they are too extreme in magnitude) and extremes (values that are possible and extreme in magnitude), show in Figure 8.3.

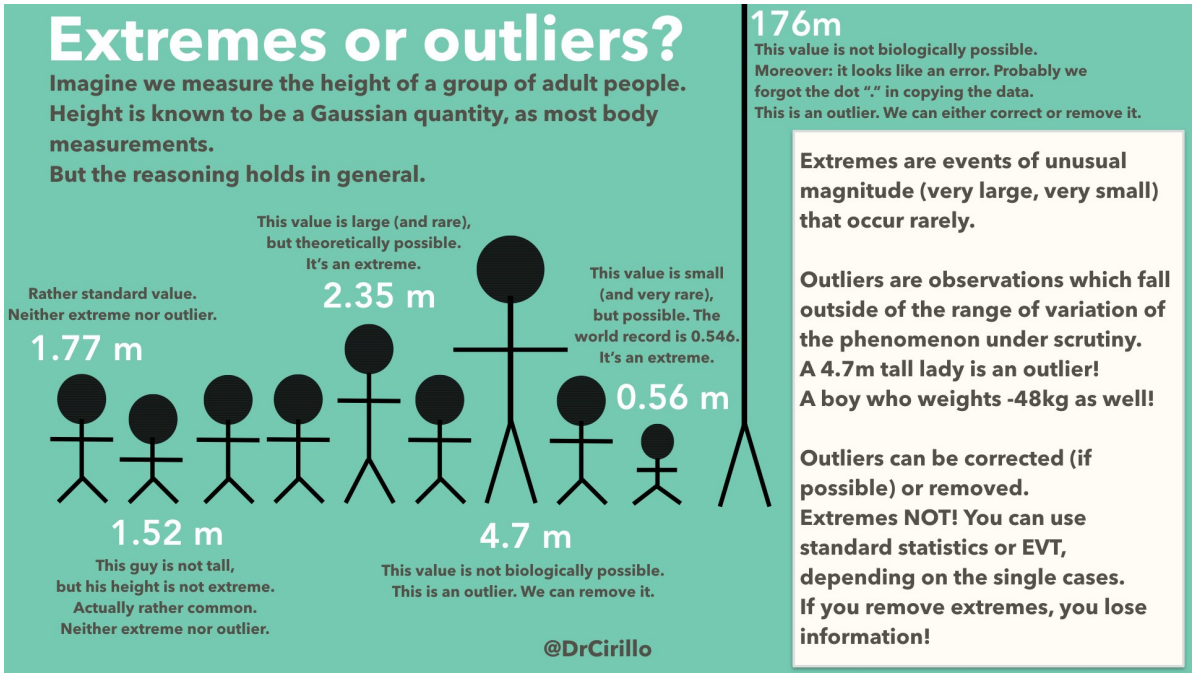


Figure 8.3: The bounds for a legitimate value depend on the variable, and should guide your choice to delete or retain the value.

8.5 Detecting Illogical or Extreme Values

One simple method of determining whether there exist illogical values in a numeric variable is to look at the maximum and minimum values (ensuring that they correspond to logical values for that variable) using the `range()` function. Similarly, boxplots `boxplot()` and histograms `hist()` are also useful. Let's look at the built-in `mtcars` data frame.

```
# Assign the mtcars dataframe to an object named df1
df1 <- mtcars

# I add a few outliers to the 'mpg' column
df1[c(2,5,7,9:11,22), 1] <- 500
```

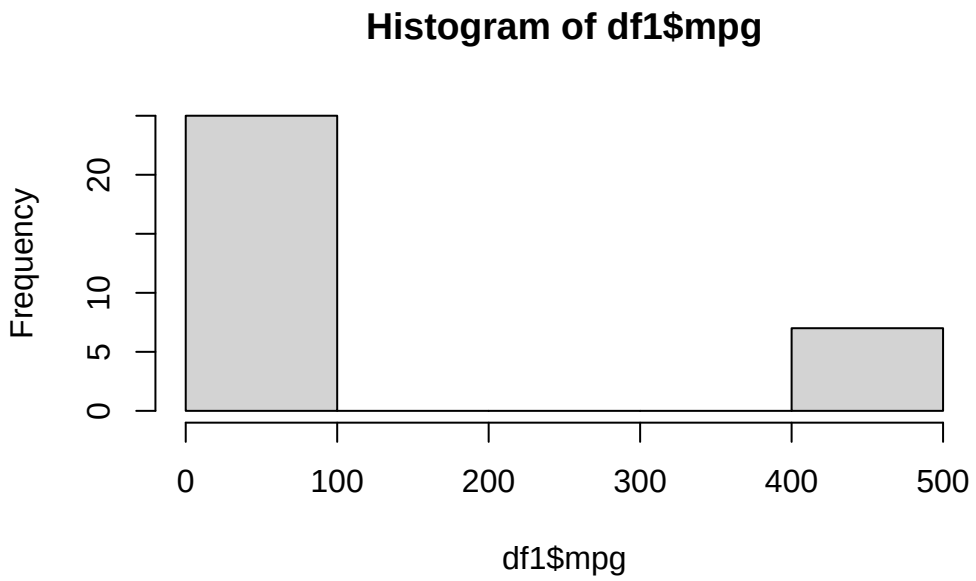
```
# Let's examine the range of values in the 'mpg' column starting with the range.
range(df1$mpg)
```

```
[1]  10.4 500.0
```

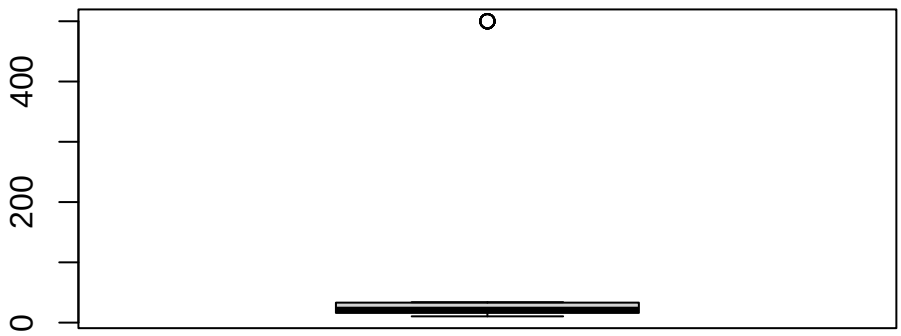
Immediately, I see that the minimum value of 10 miles per gallon appears logical (think bigger SUVs which are less fuel-efficient than smaller sedan cars). However, the maximum value of 500 miles per gallon is not logical at all. How do I know? Well, for one I put in these entries. But if I didn't know, a quick Google search tells me that the most fuel efficient gasoline car achieves 42 miles per gallon on the highway, and the most fuel efficient electric car achieves an 84 miles per gallon-equivalent. Thus, 500 miles per gallon is more of a distant dream rather than an accurate value.

Let's then visualize the variable with a histogram and a boxplot.

```
# Next, let's look at a histogram of values
hist(df1$mpg)
```




```
# Next, let's look at a boxplot of the mpg variable.
boxplot(df1$mpg)
```



The histogram shows us that the majority of values are less than 100, and a small subset of values lie around 500. This suggests that 500 might be an outlier value. The boxplot suggests that same, but provides us the **Interquartile Range (IQR)**, which refers to the 25th percentile to 75th percentile of the data. The IQR is used to build box plots, which also display 'whiskers' or lines extending above and below the box, corresponding to 1.5 times the 75th percentile and 1.5 times the 25th percentile. If data points are outside the whiskers, these can be outliers. The boxplot above is super squished, and we can see a dot at 500. This indicates that a value of 500 might be an outlier.

The `rstatix` package also has some useful functions for detecting outliers. The `identify_outliers()` function in particular returns you a data frame with two new columns indicating whether a specific value is an outlier (defined in this package as 1.5 times the IQR) or an extreme value (defined in this package is 1.5 times the IQR).

```
library(rstatix)
library(dplyr)

df2 <- df1 %>%
  identify_outliers(mpg)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	is.outlier	is.extreme
Mazda RX4 Wag	500	6	160.0	110	3.90	2.875	17.02	0	1	4	4	TRUE	TRUE
Hornet Sportabout	500	8	360.0	175	3.15	3.440	17.02	0	0	3	2	TRUE	TRUE
Duster 360	500	8	360.0	245	3.21	3.570	15.84	0	0	3	4	TRUE	TRUE
Merc 230	500	4	140.8	95	3.92	3.150	22.90	1	0	4	2	TRUE	TRUE
Merc 280	500	6	167.6	123	3.92	3.440	18.30	1	0	4	4	TRUE	TRUE
Merc 280C	500	6	167.6	123	3.92	3.440	18.90	1	0	4	4	TRUE	TRUE
Dodge Challenger	500	8	318.0	150	2.76	3.520	16.87	0	0	3	2	TRUE	TRUE

The resulting data frame shows the seven unique values that are 1.5 and 3 times the IQR (see the two columns added at the end). For our purposes, this helps identify the specific outliers or extreme values. In this case, we have already determined that 500 is an illogical value for the fuel efficiency (miles per gallon) of cars. Thus, we can delete the values and replace them with `NA` to indicate missing values. To accomplish this, let's use the `which()` function and `%in%` operator to identify the rows with a value of 500 for `mpg` from our original dataset. The `which()` function gives us the position or index which satisfies a given condition. Recall that the `%in%` operator checks whether the values in the first argument are present in the second argument, and returns a logical value. When we combine `which()` with the `%in%` operator, and add those into the first argument in square brackets after a dataframe, this will give the specific rows that satisfy a particular condition.

```
# Identify rows with mpg == 500
df1[which(df1$mpg %in% 500),]
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4 Wag     500   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Hornet Sportabout 500   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Duster 360        500   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 230          500   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280          500   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C         500   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Dodge Challenger  500   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

```
# Replace 500 with NA
df1[which(df1$mpg %in% 500), 1] <- NA
```

If we view the resulting dataframe, we can see that the `mpg` variable for the rows in question have all been replaced with `NA` to indicate missing data (indicated with red squares).

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	NA	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	NA	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	NA	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	NA	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	NA	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	NA	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	NA	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2

8.6 Variable Subsetting, Naming, Transformation & Creation

8.6.1 Subsetting

Another key component of the data cleaning process is to prepare your set of variables for analysis. In R, since it is very easy to work with multiple dataframes at the same time (just assign them to different objects), it's a good idea to create a dataframe with only the variables you need for the analysis. This is not strictly necessary, but keeps this organized, easy to view, and manageable. If you only need 20 variables for an analysis, why work with 175 variables in a data frame? It's better to keep what you need and save those to a new dataframe.

This is easily accomplished by subsetting the dataframe based on one or more conditions. This could be a particular variable value, a particular set of rows, or some other condition. Let's demonstrate this using data from the CrossFit Games from 2007-2023.

```r
# First let's import our CSV dataframe, and see the dimensions.
crossfit <- read.csv("~/R Book/Datasets/crossfit.csv", header = TRUE)
dim(crossfit)
```

```
[1] 1714   30
```

```r
# Next, let's have a quick peek at the first 10 rows and first six columns.
head(crossfit[, c(1:6)], n = 10)
```

```
   competitorId    competitorName   firstName lastName gender genderId
1          1616       Russ Greene        Russ   Greene      M        1
2          1616       Russ Greene        Russ   Greene      M        1
3          1685 Christopher Woods Christopher    Woods      M        1
4          1690      Travis Mayer      Travis    Mayer      M        1
5          1690      Travis Mayer      Travis    Mayer      M        1
6          1690      Travis Mayer      Travis    Mayer      M        1
7          1690      Travis Mayer      Travis    Mayer      M        1
8          1690      Travis Mayer      Travis    Mayer      M        1
9          1690      Travis Mayer      Travis    Mayer      M        1
10         1690      Travis Mayer      Travis    Mayer      M        1
```

Let's say I've also had a more detailed look at the data, either beforehand using a codebook, or after importing the data into R and using the `View()` function. I've decided to conduct a descriptive analysis by examining the competitors with the most competition starts, the highest ranks, as well as some of their demographic characteristics. I decide to analyze the following variables shown in Table 8.1.

Table 8.1: List of variables I will retain for analysis.

Variable Name	Description	Variable Type
competitorId	Competitor's unique identification number.	Factor
competitorName	Competitor's full name.	Factor
gender	Competitor's gender (male or female only in this dataset).	Factor
age	Competitor's age at the time of competition.	Numeric
height	Competitor's height in centimeters.	Numeric
weight	Competitor's weight in kilograms.	Numeric
countryOfOriginName	Competitor's country of origin	Factor
year	Year that competitor participated in competition.	Factor

Variable Name	Description	Variable Type
overallScore	Competitor's overall CrossFit score.	Numeric
overallRank	Competitor's overall CrossFit rank.	Numeric

You will notice that I put the type of variable in a column as well. This is not necessarily the variable type currently in the dataframe, but rather, the type of variable I *want* the variable to be before I proceed to any sort of analysis. If the variable is not in the correct class I want, I will transform it to the correct class.

To subset these variables, I can use the helpful `subset()` command from base R, where the first argument is the object to be subsetted, and the second argument can be used to `select` the variables I want to retain.

```
library(dplyr)

# Keep only the variables I want using subset().
crossfit2 <- subset(crossfit, select = c("competitorId",
                                         "competitorName",
                                         "gender",
                                         "age",
                                         "height",
                                         "weight",
                                         "countryOfOriginName",
                                         "year",
                                         "overallScore",
                                         "overallRank")
                    )

head(crossfit2[, 1:7], n = 10)
```

```
   competitorId    competitorName gender age height weight countryOfOriginName
1          1616       Russ Greene      M  20    178     83
2          1616       Russ Greene      M  21    178     83
3          1685 Christopher Woods      M  29    163     82
4          1690      Travis Mayer      M  23    181     93       United States
5          1690      Travis Mayer      M  25    181     93       United States
6          1690      Travis Mayer      M  26    181     93       United States
7          1690      Travis Mayer      M  28    181     93       United States
8          1690      Travis Mayer      M  29    181     93       United States
9          1690      Travis Mayer      M  30    181     93       United States
10         1690      Travis Mayer      M  31    181     93       United States
```

8.6.2 (Re)Naming

Alright, that's looking pretty good. I've got all the variables I want saved in a new dataframe called `crossfit2`. However, I'm not really digging some of the variable names. Some of them like `countryOfOriginName` got some weird capitalization going on, which can increase the likelihood of an error later on. Let's make simpler names for all variables. Remember, any final variable names should be reflected in an updated codebook. This applies for any dataframe you work with, because reproducibility involves careful documentation of variable transformations and choices.

```
# First, let's look at all variable names in our smaller dataframe.
names(crossfit2)
```

```
 [1] "competitorId"        "competitorName"      "gender"
 [4] "age"                 "height"              "weight"
 [7] "countryOfOriginName" "year"                "overallScore"
[10] "overallRank"
```

```
# Next, since I want to make changes to most of the names, let's create a new vector of na

newnames <- c("id",
              "name",
              "gender",
              "age",
              "height",
              "weight",
              "country",
              "year",
              "score",
              "rank")

names(crossfit2) <- newnames

# Let's check that it worked correctly.
names(crossfit2)
```

```
[1] "id"      "name"    "gender"  "age"     "height"  "weight"  "country"
[8] "year"    "score"   "rank"
```

```
# If we wanted to change a few variable names instead of writing out a whole vector, we ca

crossfit2 <- crossfit2 %>%
  rename(score_overall = score) %>%
  rename(rank_overall = rank)

names(crossfit2)
```

```
[1] "id"            "name"          "gender"        "age"
[5] "height"        "weight"        "country"       "year"
[9] "score_overall" "rank_overall"
```

Alright, so we now how to modify variable names. But what about **variable labels**? These are brief descriptions of each variable that are helpful to have in a dataframe to facilitate reproducibility. Even if you have a codebook, having variable labels can be useful for quick reference, and to know which variables are relevant to one's analyses. This can be accomplished easily using the `expss` package's `apply_labels()` function. The function takes the dataframe as the first argument, followed by a list of variables equal to their variable labels in quotes. Note: if the variable name contains spaces, then you should put it in quotes, or R won't recognize it as a variable name in this function.

Let's go ahead and add some variable labels to these variables. We can then either `View()` the dataframe to see the variable labels, or use the `str()` function to have a quick look.

```
library(expss)

# Let's add variable labels to all the variables.

crossfit2 <- apply_labels(crossfit2,
                          id = "Participant's unique identification number",
                          name = "Participant's full name",
                          gender = "Participant's gender",
                          age = "Participant's age at time of competition",
                          height = "Participant's height (in centimeters)",
                          weight = "Participant's weight (in kilograms)",
                          country = "Participant's country of origin",
                          year = "Participant's year of competition",
                          score_overall = "Participant's total CrossFit Games Score",
                          rank_overall = "Participant's overall CrossFit rank"
                          )

str(crossfit2)
```

```
'data.frame':   1714 obs. of  10 variables:
 $ id           :Class 'labelled' int  1616 1616 1685 1690 1690 1690 1690 1690 1690 1690 ...
   .. .. LABEL: Participant's unique identification number
 $ name         :Class 'labelled' chr  "Russ Greene" "Russ Greene" "Christopher Woods" "Travi
   .. .. LABEL: Participant's full name
 $ gender       :Class 'labelled' chr  "M" "M" "M" "M" ...
   .. .. LABEL: Participant's gender
 $ age          :Class 'labelled' int  20 21 29 23 25 26 28 29 30 31 ...
   .. .. LABEL: Participant's age at time of competition
 $ height       :Class 'labelled' num  178 178 163 181 181 181 181 181 181 181 ...
   .. .. LABEL: Participant's height (in centimeters)
 $ weight       :Class 'labelled' num  83 83 82 93 93 93 93 93 93 93 ...
   .. .. LABEL: Participant's weight (in kilograms)
 $ country      :Class 'labelled' chr  "" "" "" "United States" ...
   .. .. LABEL: Participant's country of origin
 $ year         :Class 'labelled' int  2007 2008 2008 2014 2016 2017 2019 2020 2021 2022 ...
   .. .. LABEL: Participant's year of competition
 $ score_overall:Class 'labelled' int  232 21 19 483 702 674 368 0 822 685 ...
   .. .. LABEL: Participant's total CrossFit Games Score
 $ rank_overall :Class 'labelled' int  11 53 32 29 10 12 12 19 12 18 ...
   .. .. LABEL: Participant's overall CrossFit rank
```

```
# Let's say we only want to add variable labels for one variable. Then we can use the var_

var_lab(crossfit2$score_overall) <- "Total CrossFit Games score"
var_lab(crossfit2$score_overall)
```

```
[1] "Total CrossFit Games score"
```

8.6.3 Transformation & Creation

Now that we know how to rename and subset the variables we want, what about transforming an old variable into a new one? Say that we wanted to derive a new height variable (in feet) from our existing height variable (in centimeters). From a quick Google search, I learn that to convert from centimeters to feet involves dividing the centimeters by 30.48.

Similarly, let's also derive a new weight variable (in pounds) from our existing weight variable (in kilograms). Once again, a quick Google search tells me that to convert from kilograms to pounds involves dividing the kilograms by 0.45359237.

We can easily derive one variable from another in this fashion using the `mutate()` function from the `dplyr` package.

```r
# Convert centimeters to feet in a new variable called height_feet.
crossfit2 <- crossfit2 %>%
  mutate(height_feet = height / 30.48)

# Let's look at these two variables side-by-side.
head(crossfit2[, c("height", "height_feet")], n = 10)
```

```
   height height_feet
1     178    5.839895
2     178    5.839895
3     163    5.347769
4     181    5.938320
5     181    5.938320
6     181    5.938320
7     181    5.938320
8     181    5.938320
9     181    5.938320
10    181    5.938320
```

```r
# Ok, it looks good generally. But, I'm not liking the many decimal places in the height_f
crossfit2$height_feet <- round(crossfit2$height_feet, digits = 1)

# Let's see if it worked.
head(crossfit2[, c("height", "height_feet")], n = 10)
```

```
   height height_feet
1     178         5.8
2     178         5.8
3     163         5.3
4     181         5.9
5     181         5.9
6     181         5.9
7     181         5.9
8     181         5.9
9     181         5.9
10    181         5.9
```

```r
# Great, now let's do the same with the new variable weight_lbs.
crossfit2 <- crossfit2 %>%
```

```
  mutate(weight_lbs = weight / 0.45359237)

# Let's check out the variables side-by-side.
head(crossfit2[, c("weight", "weight_lbs")], n = 10)
```

```
   weight weight_lbs
1      83   182.9837
2      83   182.9837
3      82   180.7791
4      93   205.0299
5      93   205.0299
6      93   205.0299
7      93   205.0299
8      93   205.0299
9      93   205.0299
10     93   205.0299
```

```
# Let's round this to the nearest unit, and see if it works.
crossfit2$weight_lbs <- round(crossfit2$weight_lbs, digits = 0)

head(crossfit2[, c("weight", "weight_lbs")], n = 10)
```

```
   weight weight_lbs
1      83        183
2      83        183
3      82        181
4      93        205
5      93        205
6      93        205
7      93        205
8      93        205
9      93        205
10     93        205
```

Before we can summarize the data with some descriptive analyses, we need to make sure the the variable classes match those in Table 8.1. To quickly look at all the variable types in the dataframe, we can use the `str()` function.

```
str(crossfit2)
```

```
'data.frame':   1714 obs. of  12 variables:
 $ id           :Class 'labelled' int  1616 1616 1685 1690 1690 1690 1690 1690 1690 1690 ...
   .. .. LABEL: Participant's unique identification number
 $ name         :Class 'labelled' chr  "Russ Greene" "Russ Greene" "Christopher Woods" "Travi
   .. .. LABEL: Participant's full name
 $ gender       :Class 'labelled' chr  "M" "M" "M" "M" ...
   .. .. LABEL: Participant's gender
 $ age          :Class 'labelled' int  20 21 29 23 25 26 28 29 30 31 ...
   .. .. LABEL: Participant's age at time of competition
 $ height       :Class 'labelled' num  178 178 163 181 181 181 181 181 181 181 ...
   .. .. LABEL: Participant's height (in centimeters)
 $ weight       :Class 'labelled' num  83 83 82 93 93 93 93 93 93 93 ...
   .. .. LABEL: Participant's weight (in kilograms)
 $ country      :Class 'labelled' chr  "" "" "" "United States" ...
   .. .. LABEL: Participant's country of origin
 $ year         :Class 'labelled' int  2007 2008 2008 2014 2016 2017 2019 2020 2021 2022 ...
   .. .. LABEL: Participant's year of competition
 $ score_overall:Class 'labelled' int  232 21 19 483 702 674 368 0 822 685 ...
   .. .. LABEL: Total CrossFit Games score
 $ rank_overall :Class 'labelled' int  11 53 32 29 10 12 12 19 12 18 ...
   .. .. LABEL: Participant's overall CrossFit rank
 $ height_feet  :Class 'labelled' num  5.8 5.8 5.3 5.9 5.9 5.9 5.9 5.9 5.9 5.9 ...
   .. .. LABEL: Participant's height (in centimeters)
 $ weight_lbs   :Class 'labelled' num  183 183 181 205 205 205 205 205 205 205 ...
   .. .. LABEL: Participant's weight (in kilograms)
```

We can see that though some variables are in the correct class (e.g. year, score_overall, etc.), others are not (e.g. id, name, gender, and country). These variables are currently stored as strings (or characters), whereas we want them to be factors. This is why people often add the argument `stringsAsFactors = TRUE` to the `read.csv()` function, which automatically converts character variables to factor, which is the appropriate class for categorical variables. However, even if you invoke this argument, there will still be some variables that are misclassed. Most commonly, ID variables are often classes as integers, whereas they should really be factors, since they are discrete identifiers.

For now, let's convert all the misclassed variables in the dataframe to factor using two methods: 1) a simple but inefficient approach, and 2) a slightly more complex but more efficient approach.

```r
# First, let's use the simple approach.
crossfit2$id <- as.factor(crossfit2$id)
```

```r
# You can always use this approach with each variable one-by-one. But let's use a more eff

catvars <- c("id", "name", "gender", "country")

crossfit2 <- crossfit2 %>%
  mutate(across(catvars, as.factor)
         )

# Let's check if it worked
str(crossfit2)
```

```
'data.frame':   1714 obs. of  12 variables:
 $ id           : Factor w/ 953 levels "1616","1685",..: 1 1 2 3 3 3 3 3 3 3 ...
 $ name         : Factor w/ 975 levels "Aaron Finley",..: 820 820 200 931 931 931 931 931 93
 $ gender       : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ age          :Class 'labelled' int  20 21 29 23 25 26 28 29 30 31 ...
   .. .. LABEL: Participant's age at time of competition
 $ height       :Class 'labelled' num  178 178 163 181 181 181 181 181 181 181 ...
   .. .. LABEL: Participant's height (in centimeters)
 $ weight       :Class 'labelled' num  83 83 82 93 93 93 93 93 93 93 ...
   .. .. LABEL: Participant's weight (in kilograms)
 $ country      : Factor w/ 116 levels "","Afghanistan",..: 1 1 1 112 112 112 112 112 112 11
 $ year         :Class 'labelled' int  2007 2008 2008 2014 2016 2017 2019 2020 2021 2022 ...
   .. .. LABEL: Participant's year of competition
 $ score_overall:Class 'labelled' int  232 21 19 483 702 674 368 0 822 685 ...
   .. .. LABEL: Total CrossFit Games score
 $ rank_overall :Class 'labelled' int  11 53 32 29 10 12 12 19 12 18 ...
   .. .. LABEL: Participant's overall CrossFit rank
 $ height_feet  :Class 'labelled' num  5.8 5.8 5.3 5.9 5.9 5.9 5.9 5.9 5.9 5.9 ...
   .. .. LABEL: Participant's height (in centimeters)
 $ weight_lbs   :Class 'labelled' num  183 183 181 205 205 205 205 205 205 205 ...
   .. .. LABEL: Participant's weight (in kilograms)
```

We can see that the four variables are now correctly classed as factor. Notice that the more efficient approach involves the `mutate()`, `across()`, and `as.factor()` functions. We've already seen `mutate()`function, but note that the `across()` function allows us to carry out an operation across multiple columns. We created a vector called `catvars` of column names we want to manipulate, and passed that into the `across()` and `mutate()` functions. Finally, the `as.factor()` function is used without parentheses in order to re-class the variables to factor. Similarly, you can convert variables to numeric using the `as.numeric()` function, and convert to character using the `as.character()` function.

8.7 Saving Your Cleaned Dataframe

8.7.1 As an R Data File

Once you're satisfied that the data are clean and ready for analysis, you can save the cleaned dataframe (or any object) as as an R Data file (.Rdata) or an RDS file (.RDS). R Data files can store multiple objects, while RDS files can only store a single R object. I tend to prefer R Data files as I can use them to store and load multiple objects at once. The `save()` function takes the R object names you want to save as the first n arguments, followed by the `file =`argument in which you specify a file name (and path) for saved object(s) in quotes and with the `.Rdata` file extension. To retrieve the object, simply use the `load()` function with the object name in quotes.

```
save(crossfit2, file = "crossfitdf.Rdata")
load("crossfitdf.Rdata")
```

8.7.2 As an Excel or CSV File

R data files are native to R, and you might want to save your dataframe in other formats. Perhaps, you want to save your data frame in Excel format (.xlsx). This can be done using the `writexl` package, and the `write_xlsx()` function where the first argument is the R object you want to save, and the second argument is the name (and path) of the Excel file with the .xlsx file extension.

To export to CSV format, you can use the built-in `write.csv()` function, where the first argument is the R object name, and the second is the file name (and path). I also recommend a third argument `row.names = FALSE` which gets rid of the row numbering which you already have when you open a spreadsheet.

```
# To export to Excel format, use the writexl package.
library(writexl)

write_xlsx(crossfit2, "crossfitdf.xlsx")

# To export to csv, use the write.csv() function.
write.csv(crossfit2, "crossfitdf.csv", row.names = FALSE)
```

8.7.3 As a Stata, SPSS, or SAS File

If you want to export to SAS, SPSS, or SAS formats, we can use the `haven` package. For SAS, you can use the `write_xpt()` function. For Stata, you can use the `write_dta()` function. For

SPSS, you can use the `write_sav()` function. The first argument of each is the R object to be saved, and the second argument is the name of the file (and path) with the appropriate file extension.

```
library(haven)

# Export to SAS format
write_xpt(crossfit2, "crossfitdf.xpt")

# Export to Stata format
write_dta(crossfit2, "crossfitdf.dta")

# Export to SPSS format
write_dta(crossfit2, "crossfitdf.sav")
```

8.7.4 As a Text File

The `write.table()` function can be used to export your dataframe to a text file. As with the other functions, the first argument is the R object to be exported, and the second is the name (and path) with the appropriate file extension. I recommend using a separator for readability using the `sep =`argument. Common choices for separator are commas `sep = ","` or tabs `sep = "\t"`.

```
# Export to text file
write.table(crossfit2, "crossfitdf.txt", sep = "\t")
```

8.8 Exercises

As always, it's a good idea to attempt these while the material is still fresh. You can find the answers in Appendix E.

Let's create a dataframe with the following code:

```
indianfood <- data.frame(name = c("Boondi",
                                  "Gajar ka halwa",
                                  "Ghevar",
                                  "Kalakand",
                                  "Misti Doi",
                                  "Aloo tikki",
                                  "Chicken tikka masala"),
                         ingredients = c("Maida flour, yogurt, oil, sugar",
                                         "Carots, milk, sugar, ghee, cashews, raisins",
                                         "Flour, ghee, kewra, milk, clarified butter, suga
                                         "Milk, cottage cheese, sugar",
                                         "Milk, jaggery",
                                         "Rice flour, potatoe, bread crumbs, garam masalaa
```

```
                                "Naan bread, tomato sauce, skinless chicken breas
                    prep_time = c(45,
                                  15,
                                  15,
                                  20,
                                  480,
                                  5,
                                  15020240),
                    state = c("West Bengal",
                              "Punjab",
                              "Rajasthan",
                              "West Bengal",
                              "West Bengal",
                              "Punjab",
                              "Punjab")
                    )
```

Next, let's fix some errors and make some changes!

1. First, let's create some more descriptive variable names. Change the existing variables names to *Name, Ingredients, Preparation Time (mins), and Origin (state)*. Then, print the names of the dataframe.

2. Next, having read through the ingredients, we can see various typos that should be fixed. Go ahead and fix the spelling mistakes (typos) you see in the `Ingredients` column. *Hint: there are four spelling mistakes in the* `Ingredients` *column, and we are using standard American English for spelling.*

3. I see that the ingredients list has the terms "cottage cheese", "clarified butter", and "naan bread." Substitute *paneer* for "cottage cheese" and remove "clarified butter" since there is already 'ghee' in the same row, which is pretty much the same thing. Also, remove 'bread' from "Naan bread", since that is unnecessary. Then print the `Ingredients` column.

4. Let's check the class of the `Origin (state)` variable. If it is not already in factor class, go ahead and convert it to factor. Make sure to save the new factor-classed variable back in the dataset. Then, check the class of the variable again to make sure it is factor.

5. Next, let's add some variable labels to each variable. The variable labels can say whatever you think might be helpful. In general, think of some description that will provide context if you are looking at these data for the first time. Then, do something to check if it worked. *Hint: you've seen two ways to do this.*

6. Finally, let's have a look at the `Preparation Time (mins)` variable to detect any outliers. Print the range of values and produce a histogram of this variable. Are any values seemingly outliers from the histogram?

7. If you detected an outlier in the previous question (and I hope that you did!), explain your decision as to remove it or retain it in the dataframe. If you choose to remove the outlier value, replace it with a more plausible value derived from a quick Google search. Then, print the variable.

8. Finally, save your cleaned `indianfood` dataframe as an R data file as well as a CSV file using the file name of your choice.

9 Data Exploration with `dplyr`

The `dplyr` package is a great tool for organizing and manipulating our dataframe. Note, I use the term *manipulate* not to convey anything nefarious or unethical, but in the sense of *data management*. The functions of this package are named after useful verbs, making them relatively easy to remember. Some common `dplyr` functions are listed in Table 9.1.

Table 9.1: dplyr's useful functions with their description.

`dplyr` Function	Description
`arrange()`	Change the order of rows.
`filter()`	Select rows based on column values.
`mutate()`	Change the values in certain columns, and create new columns.
`relocate()`	Change the order of columns.
`rename()`	Change the name of columns.
`select()`	Include or exclude a column.
`slice()`	Select rows based on location.
`summarise()`	Collapse a group into a single row.
`group_by()`	Select a grouping variable to perform an operation by group.

Let's look at some examples with the built-in `mtcars` dataframe.

9.1 Filter() & Arrange()

```
library(dplyr)

# Load mtcars dataframe. Assign it to an object.
df <- mtcars

# Let's look at cars with only eight cylinders.
df %>%
  filter(cyl == 8)
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```
# Let's add to that, and order the dataframe by mpg in descending order (highest to lowest

df %>%
  filter(cyl == 8) %>%
  arrange(desc(mpg))
```

```
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

9.2 Mutate() & Rename()

```
# Now let's add a new column called kpl (kilometers per litre) using mutate(). A quick Goo

df %>%
  filter(cyl == 8) %>%
  arrange(desc(mpg)) %>%
  mutate(kpl = mpg / 2.352) %>%
  head()
```

```
                  mpg cyl  disp  hp drat    wt  qsec vs am gear carb      kpl
Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2 8.163265
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2 7.950680
Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3 7.355442
Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3 6.972789
Ford Pantera L    15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4 6.717687
Dodge Challenger  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2 6.590136
```

```
# Let's add a new name for the wt variable called weight.
df %>%
  filter(cyl == 8) %>%
  arrange(desc(mpg)) %>%
  mutate(kpl = mpg / 2.352) %>%
  rename(weight = wt) %>%
  head()
```

```
                  mpg cyl  disp  hp drat weight  qsec vs am gear carb      kpl
Pontiac Firebird  19.2   8 400.0 175 3.08  3.845 17.05  0  0    3    2 8.163265
Hornet Sportabout 18.7   8 360.0 175 3.15  3.440 17.02  0  0    3    2 7.950680
Merc 450SL        17.3   8 275.8 180 3.07  3.730 17.60  0  0    3    3 7.355442
Merc 450SE        16.4   8 275.8 180 3.07  4.070 17.40  0  0    3    3 6.972789
Ford Pantera L    15.8   8 351.0 264 4.22  3.170 14.50  0  1    5    4 6.717687
Dodge Challenger  15.5   8 318.0 150 2.76  3.520 16.87  0  0    3    2 6.590136
```

9.3 Select() & Slice()

```
# Let's now only select a few variables using select()

df %>%
  select(mpg, cyl, wt) %>%
  head()
```

```
                   mpg cyl    wt
Mazda RX4         21.0   6 2.620
Mazda RX4 Wag     21.0   6 2.875
Datsun 710        22.8   4 2.320
Hornet 4 Drive    21.4   6 3.215
Hornet Sportabout 18.7   8 3.440
Valiant           18.1   6 3.460
```

```
## If we want to select all but a few variables, we can still use select. Let's say I want

df %>%
  select(-mpg, -cyl, -wt)
```

```
                   disp  hp drat  qsec vs am gear carb
Mazda RX4           160.0 110 3.90 16.46  0  1    4    4
Mazda RX4 Wag       160.0 110 3.90 17.02  0  1    4    4
Datsun 710          108.0  93 3.85 18.61  1  1    4    1
Hornet 4 Drive      258.0 110 3.08 19.44  1  0    3    1
Hornet Sportabout   360.0 175 3.15 17.02  0  0    3    2
Valiant             225.0 105 2.76 20.22  1  0    3    1
Duster 360          360.0 245 3.21 15.84  0  0    3    4
Merc 240D           146.7  62 3.69 20.00  1  0    4    2
Merc 230            140.8  95 3.92 22.90  1  0    4    2
Merc 280            167.6 123 3.92 18.30  1  0    4    4
Merc 280C           167.6 123 3.92 18.90  1  0    4    4
Merc 450SE          275.8 180 3.07 17.40  0  0    3    3
Merc 450SL          275.8 180 3.07 17.60  0  0    3    3
Merc 450SLC         275.8 180 3.07 18.00  0  0    3    3
Cadillac Fleetwood  472.0 205 2.93 17.98  0  0    3    4
Lincoln Continental 460.0 215 3.00 17.82  0  0    3    4
Chrysler Imperial   440.0 230 3.23 17.42  0  0    3    4
Fiat 128             78.7  66 4.08 19.47  1  1    4    1
```

```
Honda Civic          75.7  52 4.93 18.52  1  1    4    2
Toyota Corolla       71.1  65 4.22 19.90  1  1    4    1
Toyota Corona       120.1  97 3.70 20.01  1  0    3    1
Dodge Challenger    318.0 150 2.76 16.87  0  0    3    2
AMC Javelin         304.0 150 3.15 17.30  0  0    3    2
Camaro Z28          350.0 245 3.73 15.41  0  0    3    4
Pontiac Firebird    400.0 175 3.08 17.05  0  0    3    2
Fiat X1-9            79.0  66 4.08 18.90  1  1    4    1
Porsche 914-2       120.3  91 4.43 16.70  0  1    5    2
Lotus Europa         95.1 113 3.77 16.90  1  1    5    2
Ford Pantera L      351.0 264 4.22 14.50  0  1    5    4
Ferrari Dino        145.0 175 3.62 15.50  0  1    5    6
Maserati Bora       301.0 335 3.54 14.60  0  1    5    8
Volvo 142E          121.0 109 4.11 18.60  1  1    4    2
```

```
# If we want to select certain rows of a dataframe, we can do this with slice() by mention

which(df$mpg > 20)
```

```
 [1]  1  2  3  4  8  9 18 19 20 21 26 27 28 32
```

```
df %>%
  slice(1:4,
        8,
        9,
        18:21,
        26:28,
        32) %>%
  print()
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

```
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```r
# Similarly, if you want to know which rows have the value 'NA' which indicates missing da

## Let's add some NAs to the disp variable first.

df[c(5,8,21:22), 3] <- NA

## Now we will see which rows in disp have NA.

which(is.na(df$disp))
```

```
[1]  5  8 21 22
```

```r
## Often we want to know the highest and lowest values of a variable. We can use slice_min

## The five cars with the lowest mpg.

df %>%
  slice_min(mpg, n = 5)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
```

```r
## The five cars with the highest mpg.

df %>%
  slice_max(mpg, n = 5)
```

```
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
```

```
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
Fiat X1-9      27.3   4 79.0  66 4.08 1.935 18.90  1  1    4    1
```

9.4 Relocate() and Summarise()

```
# If I want to change the order of how some columns appear in the dataframe, I can do so w

## Let's move wt to the front.

df %>%
  relocate(wt, .before = everything()
           )
```

```
                     wt  mpg cyl  disp  hp drat  qsec vs am gear carb
Mazda RX4         2.620 21.0   6 160.0 110 3.90 16.46  0  1    4    4
Mazda RX4 Wag     2.875 21.0   6 160.0 110 3.90 17.02  0  1    4    4
Datsun 710        2.320 22.8   4 108.0  93 3.85 18.61  1  1    4    1
Hornet 4 Drive    3.215 21.4   6 258.0 110 3.08 19.44  1  0    3    1
Hornet Sportabout 3.440 18.7   8    NA 175 3.15 17.02  0  0    3    2
Valiant           3.460 18.1   6 225.0 105 2.76 20.22  1  0    3    1
Duster 360        3.570 14.3   8 360.0 245 3.21 15.84  0  0    3    4
Merc 240D         3.190 24.4   4    NA  62 3.69 20.00  1  0    4    2
Merc 230          3.150 22.8   4 140.8  95 3.92 22.90  1  0    4    2
Merc 280          3.440 19.2   6 167.6 123 3.92 18.30  1  0    4    4
Merc 280C         3.440 17.8   6 167.6 123 3.92 18.90  1  0    4    4
Merc 450SE        4.070 16.4   8 275.8 180 3.07 17.40  0  0    3    3
Merc 450SL        3.730 17.3   8 275.8 180 3.07 17.60  0  0    3    3
Merc 450SLC       3.780 15.2   8 275.8 180 3.07 18.00  0  0    3    3
Cadillac Fleetwood  5.250 10.4   8 472.0 205 2.93 17.98  0  0    3    4
Lincoln Continental 5.424 10.4   8 460.0 215 3.00 17.82  0  0    3    4
Chrysler Imperial   5.345 14.7   8 440.0 230 3.23 17.42  0  0    3    4
Fiat 128          2.200 32.4   4  78.7  66 4.08 19.47  1  1    4    1
Honda Civic       1.615 30.4   4  75.7  52 4.93 18.52  1  1    4    2
Toyota Corolla    1.835 33.9   4  71.1  65 4.22 19.90  1  1    4    1
Toyota Corona     2.465 21.5   4    NA  97 3.70 20.01  1  0    3    1
Dodge Challenger  3.520 15.5   8    NA 150 2.76 16.87  0  0    3    2
AMC Javelin       3.435 15.2   8 304.0 150 3.15 17.30  0  0    3    2
```

```
Camaro Z28          3.840 13.3   8 350.0 245 3.73 15.41  0  0    3    4
Pontiac Firebird    3.845 19.2   8 400.0 175 3.08 17.05  0  0    3    2
Fiat X1-9           1.935 27.3   4  79.0  66 4.08 18.90  1  1    4    1
Porsche 914-2       2.140 26.0   4 120.3  91 4.43 16.70  0  1    5    2
Lotus Europa        1.513 30.4   4  95.1 113 3.77 16.90  1  1    5    2
Ford Pantera L      3.170 15.8   8 351.0 264 4.22 14.50  0  1    5    4
Ferrari Dino        2.770 19.7   6 145.0 175 3.62 15.50  0  1    5    6
Maserati Bora       3.570 15.0   8 301.0 335 3.54 14.60  0  1    5    8
Volvo 142E          2.780 21.4   4 121.0 109 4.11 18.60  1  1    4    2
```

```r
## Let's move wt to before disp.

df %>%
  relocate(wt, .before = disp)
```

```
                   mpg cyl    wt  disp  hp drat  qsec vs am gear carb
Mazda RX4         21.0   6 2.620 160.0 110 3.90 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 2.875 160.0 110 3.90 17.02  0  1    4    4
Datsun 710        22.8   4 2.320 108.0  93 3.85 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 3.215 258.0 110 3.08 19.44  1  0    3    1
Hornet Sportabout 18.7   8 3.440    NA 175 3.15 17.02  0  0    3    2
Valiant           18.1   6 3.460 225.0 105 2.76 20.22  1  0    3    1
Duster 360        14.3   8 3.570 360.0 245 3.21 15.84  0  0    3    4
Merc 240D         24.4   4 3.190    NA  62 3.69 20.00  1  0    4    2
Merc 230          22.8   4 3.150 140.8  95 3.92 22.90  1  0    4    2
Merc 280          19.2   6 3.440 167.6 123 3.92 18.30  1  0    4    4
Merc 280C         17.8   6 3.440 167.6 123 3.92 18.90  1  0    4    4
Merc 450SE        16.4   8 4.070 275.8 180 3.07 17.40  0  0    3    3
Merc 450SL        17.3   8 3.730 275.8 180 3.07 17.60  0  0    3    3
Merc 450SLC       15.2   8 3.780 275.8 180 3.07 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 5.250 472.0 205 2.93 17.98  0  0    3    4
Lincoln Continental 10.4   8 5.424 460.0 215 3.00 17.82  0  0    3    4
Chrysler Imperial   14.7   8 5.345 440.0 230 3.23 17.42  0  0    3    4
Fiat 128          32.4   4 2.200  78.7  66 4.08 19.47  1  1    4    1
Honda Civic       30.4   4 1.615  75.7  52 4.93 18.52  1  1    4    2
Toyota Corolla    33.9   4 1.835  71.1  65 4.22 19.90  1  1    4    1
Toyota Corona     21.5   4 2.465    NA  97 3.70 20.01  1  0    3    1
Dodge Challenger  15.5   8 3.520    NA 150 2.76 16.87  0  0    3    2
AMC Javelin       15.2   8 3.435 304.0 150 3.15 17.30  0  0    3    2
Camaro Z28        13.3   8 3.840 350.0 245 3.73 15.41  0  0    3    4
Pontiac Firebird  19.2   8 3.845 400.0 175 3.08 17.05  0  0    3    2
Fiat X1-9         27.3   4 1.935  79.0  66 4.08 18.90  1  1    4    1
```

```
Porsche 914-2       26.0   4 2.140 120.3  91 4.43 16.70  0  1    5    2
Lotus Europa        30.4   4 1.513  95.1 113 3.77 16.90  1  1    5    2
Ford Pantera L      15.8   8 3.170 351.0 264 4.22 14.50  0  1    5    4
Ferrari Dino        19.7   6 2.770 145.0 175 3.62 15.50  0  1    5    6
Maserati Bora       15.0   8 3.570 301.0 335 3.54 14.60  0  1    5    8
Volvo 142E          21.4   4 2.780 121.0 109 4.11 18.60  1  1    4    2
```

```r
## Let's move wt to after qsec

df %>%
  relocate(wt, .after = qsec)
```

```
                   mpg cyl  disp  hp drat  qsec    wt vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 16.46 2.620  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 17.02 2.875  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 18.61 2.320  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 19.44 3.215  1  0    3    1
Hornet Sportabout 18.7   8    NA 175 3.15 17.02 3.440  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 20.22 3.460  1  0    3    1
Duster 360        14.3   8 360.0 245 3.21 15.84 3.570  0  0    3    4
Merc 240D         24.4   4    NA  62 3.69 20.00 3.190  1  0    4    2
Merc 230          22.8   4 140.8  95 3.92 22.90 3.150  1  0    4    2
Merc 280          19.2   6 167.6 123 3.92 18.30 3.440  1  0    4    4
Merc 280C         17.8   6 167.6 123 3.92 18.90 3.440  1  0    4    4
Merc 450SE        16.4   8 275.8 180 3.07 17.40 4.070  0  0    3    3
Merc 450SL        17.3   8 275.8 180 3.07 17.60 3.730  0  0    3    3
Merc 450SLC       15.2   8 275.8 180 3.07 18.00 3.780  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 17.98 5.250  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 17.82 5.424  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 17.42 5.345  0  0    3    4
Fiat 128          32.4   4  78.7  66 4.08 19.47 2.200  1  1    4    1
Honda Civic       30.4   4  75.7  52 4.93 18.52 1.615  1  1    4    2
Toyota Corolla    33.9   4  71.1  65 4.22 19.90 1.835  1  1    4    1
Toyota Corona     21.5   4    NA  97 3.70 20.01 2.465  1  0    3    1
Dodge Challenger  15.5   8    NA 150 2.76 16.87 3.520  0  0    3    2
AMC Javelin       15.2   8 304.0 150 3.15 17.30 3.435  0  0    3    2
Camaro Z28        13.3   8 350.0 245 3.73 15.41 3.840  0  0    3    4
Pontiac Firebird  19.2   8 400.0 175 3.08 17.05 3.845  0  0    3    2
Fiat X1-9         27.3   4  79.0  66 4.08 18.90 1.935  1  1    4    1
Porsche 914-2     26.0   4 120.3  91 4.43 16.70 2.140  0  1    5    2
Lotus Europa      30.4   4  95.1 113 3.77 16.90 1.513  1  1    5    2
Ford Pantera L    15.8   8 351.0 264 4.22 14.50 3.170  0  1    5    4
```

```
Ferrari Dino         19.7   6 145.0 175 3.62 15.50 2.770  0  1    5    6
Maserati Bora        15.0   8 301.0 335 3.54 14.60 3.570  0  1    5    8
Volvo 142E           21.4   4 121.0 109 4.11 18.60 2.780  1  1    4    2
```

```r
# Let's use the summarise() and group_by() functions to get a summary of the weight of eac

df %>%
  group_by(cyl) %>%
  summarise(mean_weight = mean(wt)) %>%
  round(digits = 2)
```

```
# A tibble: 3 x 2
    cyl mean_weight
  <dbl>       <dbl>
1     4        2.29
2     6        3.12
3     8        4
```

Thus, we can see that the `dplyr` package offers several useful functions to manage our data. Remember, that if you want any changes to be reflected in your dataframe, such as renaming a variable, remember to assign your `dplyr` code to your dataframe. For example, if I want the name change for `wt` to `weight` to stick, I would assign that to the dataframe like this:

```r
df <- df %>%
  rename(weight = wt)
```

9.5 Exercises

As always, it's a good idea to attempt these while the material is still fresh. You can find the answers in Appendix F.

1. Load the `tidyverse` package. Then assign the built-in dataframe `starwars` to an object named whatever you want. Then subset the dataframe by human species only. Save the subsetted dataframe as an object called `swhuman`. Then calculate and report the mean and median height in `swhuman`. Also report an NAs (missing data) in the height variable. *Note: the units for the height variable are centimeters.*
2. Hopefully you noticed that there are indeed some NAs in the `swhuman` dataframe! Detect which rows have NAs for the height variable, and write the names of the characters that have this. Next, let's fix these errors. Perform an internet search and populate those NAs with plausible values. If you need to convert from feet to centimeters, multiply the value in feet by 30.48. If you absolutely cannot find the height of any character substitute the median height from Question 1 for their height.
3. Once you have filled in this missing data, calculate the new mean and median for the height variable. Comment on how much of a difference the additional values made on

the mean and median compared with the values you calculated in Question 1. Then determine the three shortest characters, and three tallest characters.

4. Return to the larger `starwars` dataframe or whatever object to which you assigned it. Determine which characters have NA for height. If there are any characters with NA for height (hint: there are), enter plausible values for their heights using the approach taken in Question 2. Then report the mean and median height across everyone in this dataframe.

5. Still working with the `starwars` dataframe, convert the `species` variable to factor. Then, group and summarise the mean height by species, and print this in descending order. Report which species is the tallest, on average. Then, rearrange and report the species which is the shortest, on average.

10 Data Visualization with base R and `ggplot2`

One of the most powerful and versatile aspects of R is the ability it affords us for data visualization. More than creating pretty pictures, visualization is an important part of data analysis and science communication. Visualization can also be used to manipulate or mislead our understanding of a particular phenomenon, and can hide data. This is why it is essential to create visualizations that are clear, easy to follow, and reproducible.

A full tutorial on data visualization can comprise its own book, and in fact, does. There are a number of books and other resources for data visualization in R which can help you choose the right visualization for the right data, along with several design principles. Three of these resources are presented in Table 10.1 below.

Table 10.1: Resources for learning more about data visualization in R.

Resource name (with link)	Author
Data Visualization with R	Rob Kabacoff
R for Data Science (2e) - Visualize (Chapters 10 to 12)	Hadley Wickham, Mine Çetinkaya-Rundel & Garrett Grolemund
The R Graph Gallery	Yan Holtz

For our purposes, we'll go over how to create some basic graphs using base R, and more aesthetically pleasing graphs using the `ggplot2` package. We'll illustrate some graphs using the `gapminder` dataframe from the `gapminder` library. So go ahead and install this library if you haven't already done so.

10.1 Visualizations in Base R

10.1.1 Bar graphs

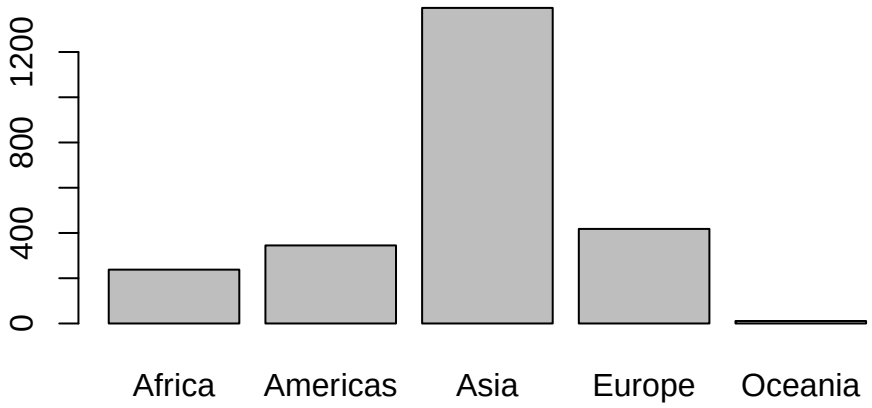
Bar graphs are commonly used to illustrate differences between groups on some numeric measure. Usually the levels of a categorical variable are on one axis, and some sort of numeric measure on the other axis. The latter can be another variable, but can also be a summary measures such as counts, sums, means, or standard deviations. In base R, we can create a bar graph using the `barplot()` function. Let's say I want to compare the total population for the five continents in the `gapminder` dataframe for the year 1952.

```
library(gapminder)
library(tidyverse)

# First, I'll create a small dataframe which filters the dataframe by the year 1952. It th

sumpop <- gapminder %>%
  filter(year == "1952") %>%
  group_by(continent) %>%
  summarise(sum_pop = sum(pop)) %>%
  mutate(sum_pop = round((sum_pop/1000000), digits = 0)) %>%
  rename("Total Population (Millions)" = sum_pop,
         "Continent" = continent)
```

```
head(sumpop)
```

```
# A tibble: 5 x 2
  Continent `Total Population (Millions)`
  <fct>                             <dbl>
1 Africa                              238
2 Americas                            345
3 Asia                               1395
4 Europe                              418
5 Oceania                              11
```

```
# Now, let's make a basic barplot.

barplot(sumpop$`Total Population (Millions)`,
        names.arg = sumpop$Continent)
```



Ok, that's not the prettiest bar graph in the world. Also the y-axis doesn't extend to the maximum value of the data. Let's change the y-axis limits with the `ylim` argument. Let's also add some color to the bars, add a title, and label the y-axis.

Colors in R

Before we proceed, let's take a moment to review how to use colors for our visualizations in R. There are a number of ways to use colors in R. The first way is to simple write the name of the color like `"red"`, `"beige"`, `"cyan"` or more exotic color names such as `"darkorchid2"`, `"goldenrod"`, and `"mistyrose"`. In fact, there are 657 colors which you can currently call by name. The full list can be seen by running the `colors()` function. Of course, seeing a list of color names is not useful unless we already know how the colors look. To see a full list of all 657 colors along with their names, check out this Data Novia page.

You might want to choose colors based on a palette, often to illustrate progress on a gradient. Or you might just find it easier to look at a palette and see which colors work well together. In such cases, I recommend choosing an established color palette from the `RColorBrewer` package. You can load this library and use the `display.brewer.all()` function to get a list of color palettes. The palettes are listed in Figure 10.1 below. You can then use the `brewer.pal()` function from the `RColorBrewer` package to add some colors to our visualizations. The first argument is the number of colors you want, and the second argument is the name of the palette.

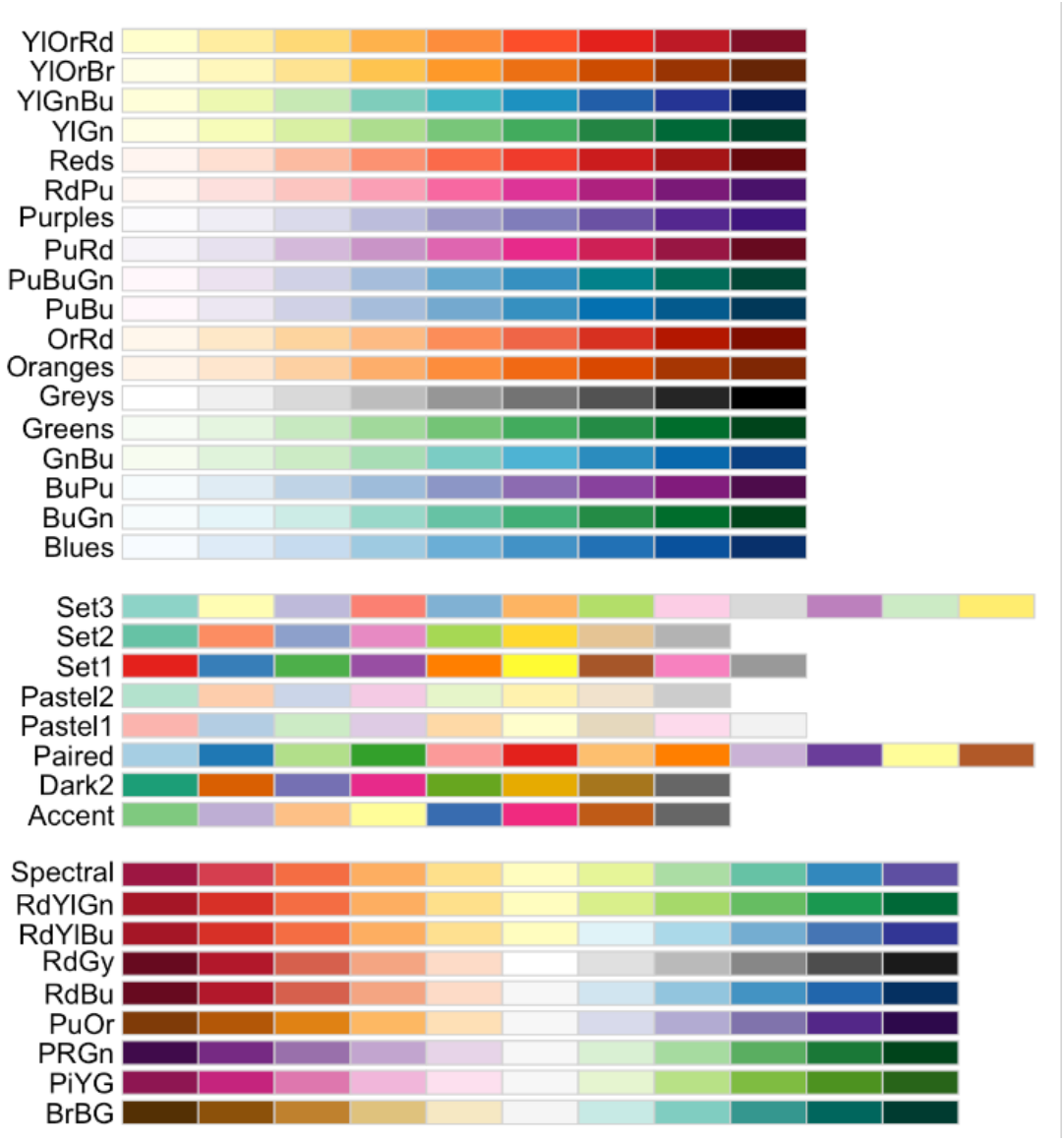


Figure 10.1: The RColorBrewer package has a number of great color palettes.

There are also great color packages available through other packages. One of my favorites is the `wesanderson` package created by Karthik Ram, which contain color palettes from the movies of famous director Wes Anderson.

Finally, you can also invoke colors by hexadecimal number. The hexadecimal system is a numerical system of base-16. It uses 16 symbols to represent numbers (10 to 15 are represented with the letters A through F). A list of hexadecimal numbers can be found

in Figure 10.2.

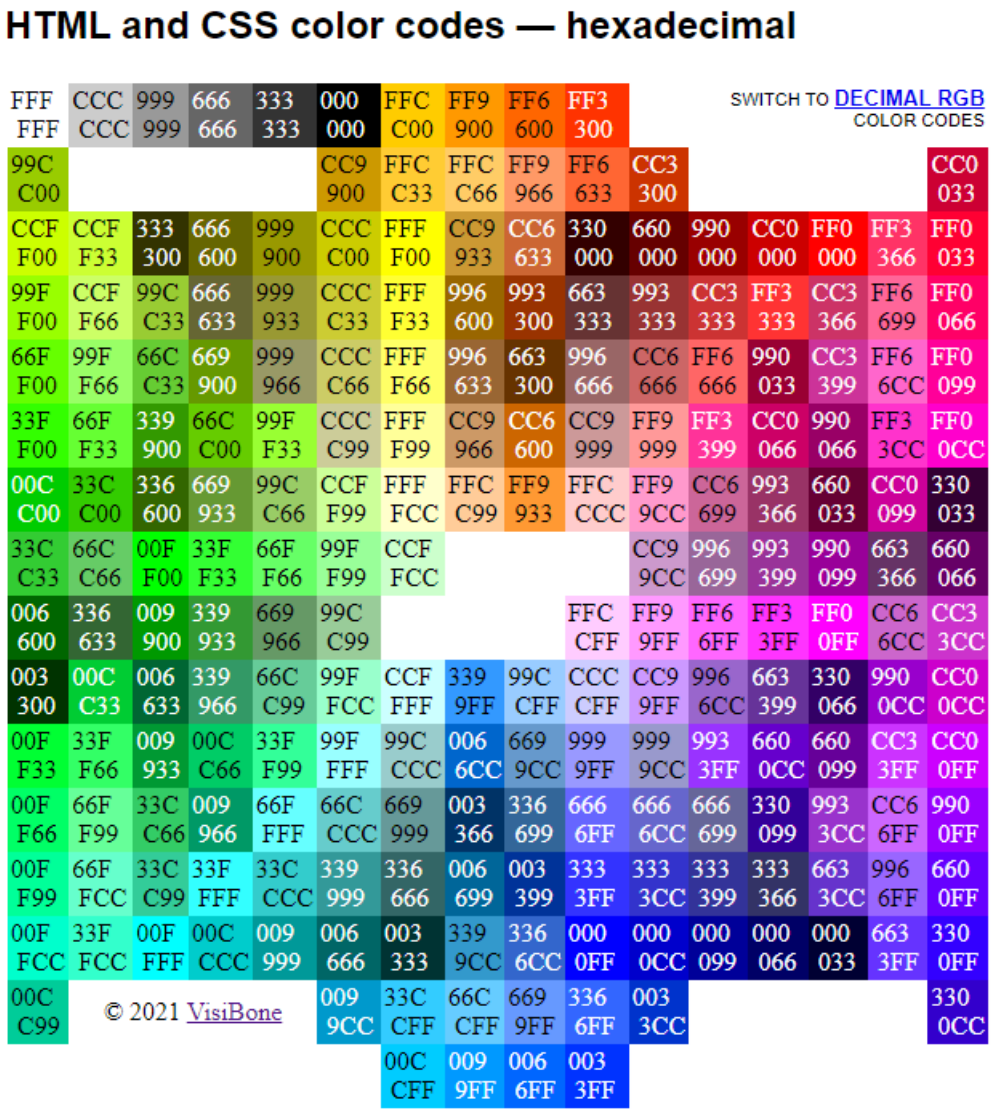
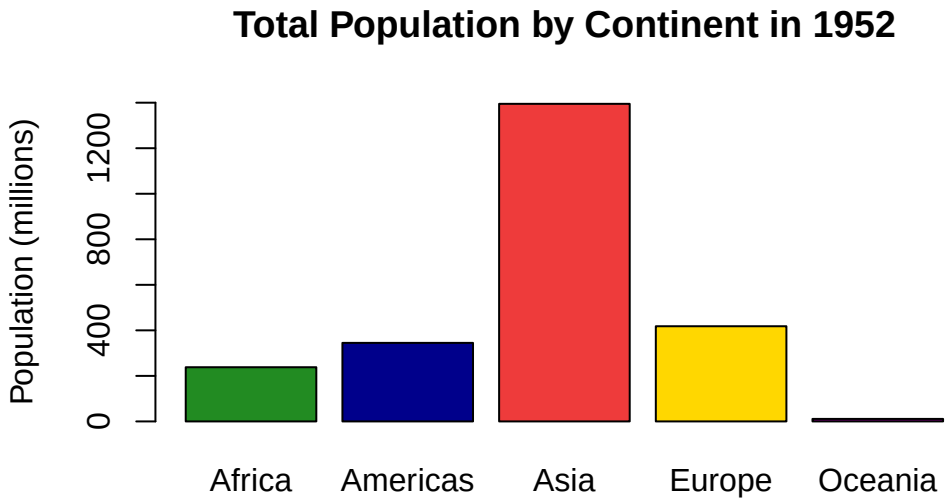


Figure 10.2: Hexadecimal color codes. Source: VisiBone.

When using the hexadecimal (or 'hex') number in the `col =` argument, make sure to add a hash # in front of the code (e.g. #00CCFF, #0033CC).

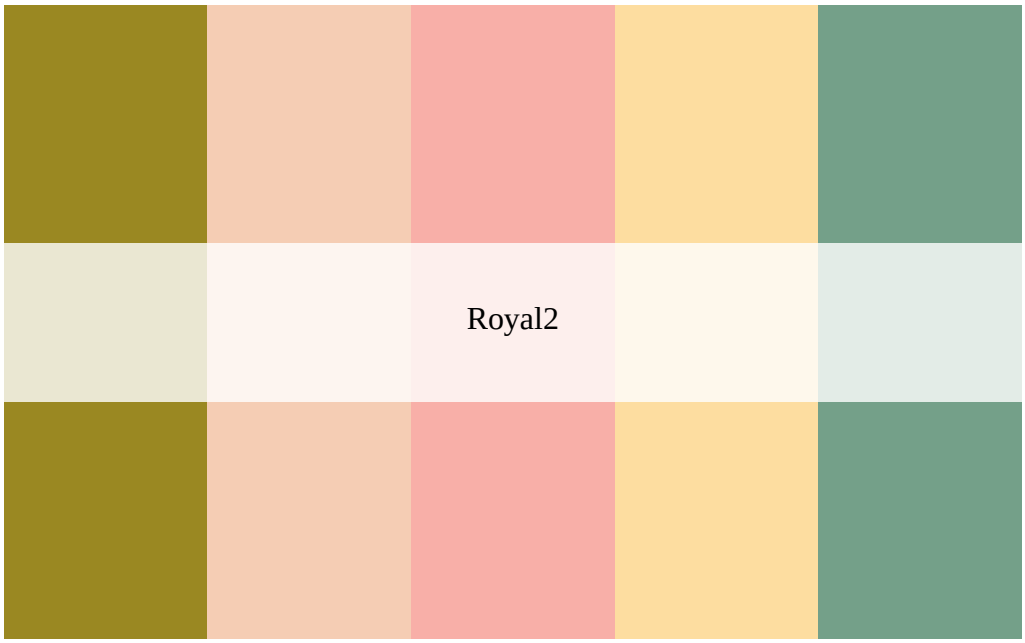
Ok, back to our barplot. Let's create one with named colors, labels, and a title.

```
barplot(sumpop$`Total Population (Millions)`,  #this is the y-axis variable.
        names.arg = sumpop$Continent,          # this is the x-axis variable.
        col = c("forestgreen", "darkblue", "brown2", "gold1", "magenta4"), # the color of
        ylim = c(0, 1400), # y-axis limit
        ylab = "Population (millions)",
        main = "Total Population by Continent in 1952"  # Title of graph.
        )
```

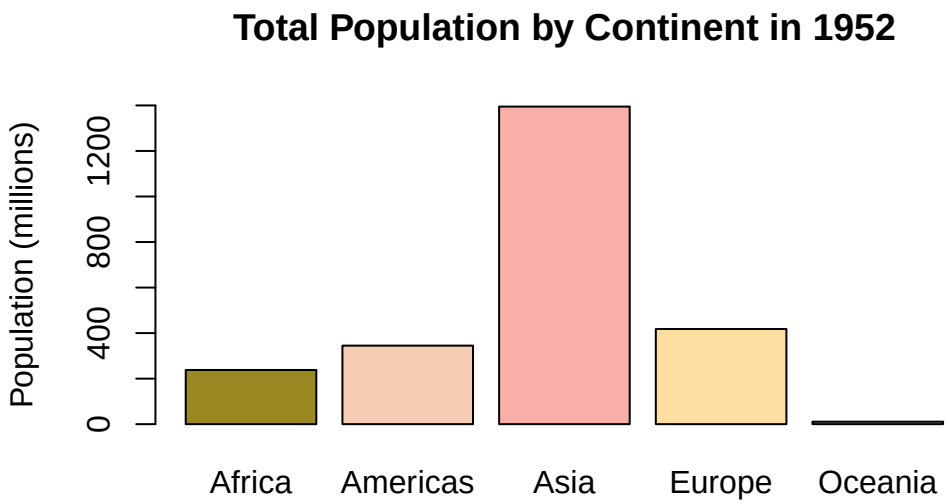


Looks good, but how about some colors from the Wes Anderson movie *The Royal Tenenbaums*? Why not!

```
library(wesanderson)

# Let's look at the second The Royal Tenenbaums palette.The first one only has four colors

wes_palette("Royal2")
```



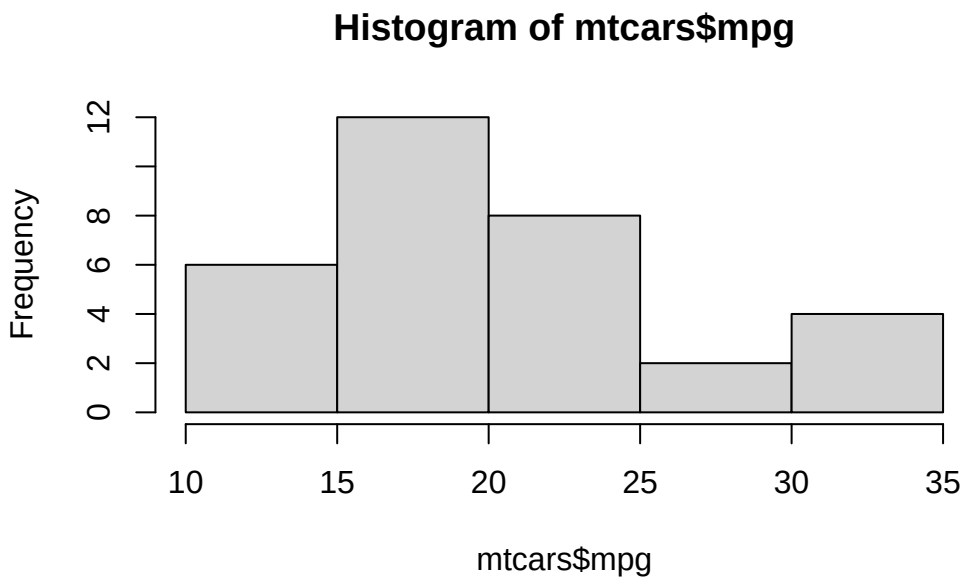

```
pal1 <- wes_palette("Royal2")

barplot(sumpop$`Total Population (Millions)`,  #this is the y-axis variable.
        names.arg = sumpop$Continent,          # this is the x-axis variable.
        col = pal1, # the color of the bars
        ylim = c(0, 1400), # y-axis limit
        ylab = "Population (millions)",
        main = "Total Population by Continent in 1952"  # Title of graph.
        )
```



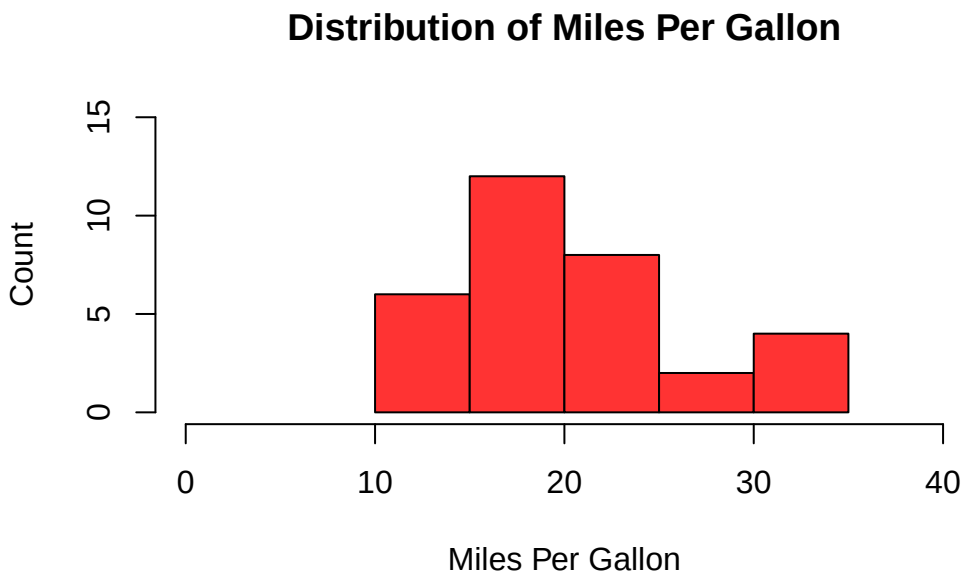
10.1.2 Histograms

If you have two numeric variables, you can examine the distribution of a variable based on values of another variable with a histogram. Even if you only want to examine the distribution of values in one variable, a histogram is a great approach. I commonly check the distribution of a variable quickly with the `hist()` function, where the first argument is the variable of interest. Other arguments can be used to add axis labels, titles, and axis limits. With a histogram, the number of bars or *bins* can be increased when you have many different values. This can be adjusted with the `breaks =` argument. Let's look at the distribution of the `mpg` variable in the `mtcars` dataframe.

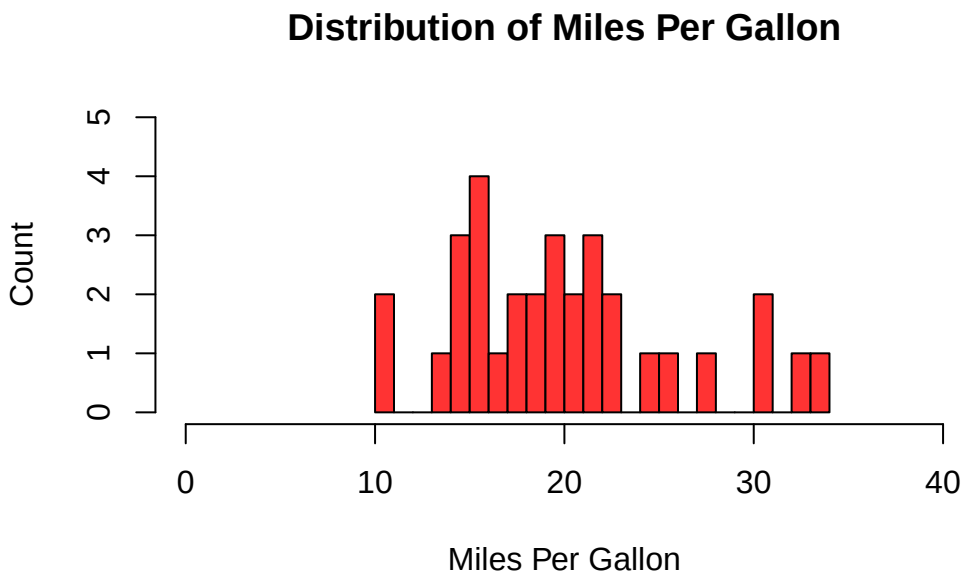
```
# A quick and basic histogram
hist(mtcars$mpg)
```




```r
# Let's add some color, labels, and axis limits.
hist(mtcars$mpg, col = "#FF3333",
     xlab = "Miles Per Gallon",
     ylab = "Count",
     main = "Distribution of Miles Per Gallon",
     xlim = c(0, 40),
     ylim = c(0, 15)
     )
```



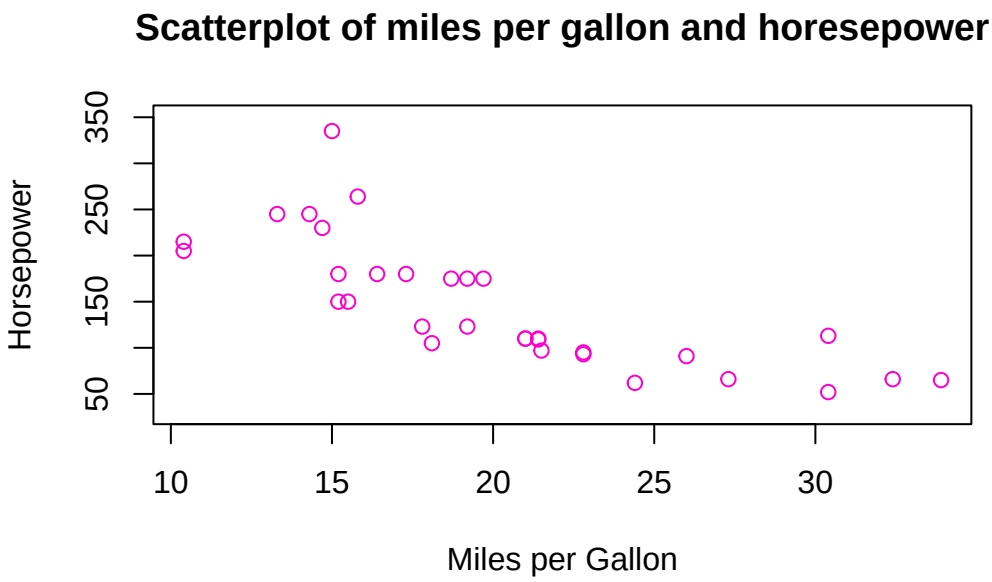

```r
# Let's now increase the number of bins. If we do this with many unique values, we may nee

hist(mtcars$mpg, col = "#FF3333",
     xlab = "Miles Per Gallon",
     ylab = "Count",
     main = "Distribution of Miles Per Gallon",
     xlim = c(0, 40),
     ylim = c(0, 5),
     breaks = 20
     )
```



10.1.3 Scatterplot

When we have two numeric variables and want to see the relationship between them, a scatterplot is ideal. This visualization is a series of dots on graph that shows the corresponding value of y for all values of x in the dataframe. This can be constructed using the `plot()` function.

```
# Let's create a scatterplot of miles per gallon and horsepower.

plot(mtcars$mpg, mtcars$hp,
     main = "Scatterplot of miles per gallon and horesepower",
     xlab = "Miles per Gallon",
     ylab = "Horsepower",
     col = "#FF00CC",
     ylim = c(30,350)
     )
```



We can also change the shape of the points to other things, listed in Figure 10.3 using the `pch =` argument. The size of the shape can be adjusted by specifying a number relative to the current size. For example `cex = 2` would double the default size of the shape, and `cex = 0.5` would reduce the size of the shape by 50%.

For instance, let's change the shape of the dots to the filled-in triangle (#17), and increase the size of the shapes by 50%.

```
plot(mtcars$mpg, mtcars$hp,
     main = "Scatterplot of miles per gallon and horesepower",
     xlab = "Miles per Gallon",
     ylab = "Horsepower",
     col = "#FF00CC",
     ylim = c(30,350),
     cex = 1.5,
     pch = 17
     )
```

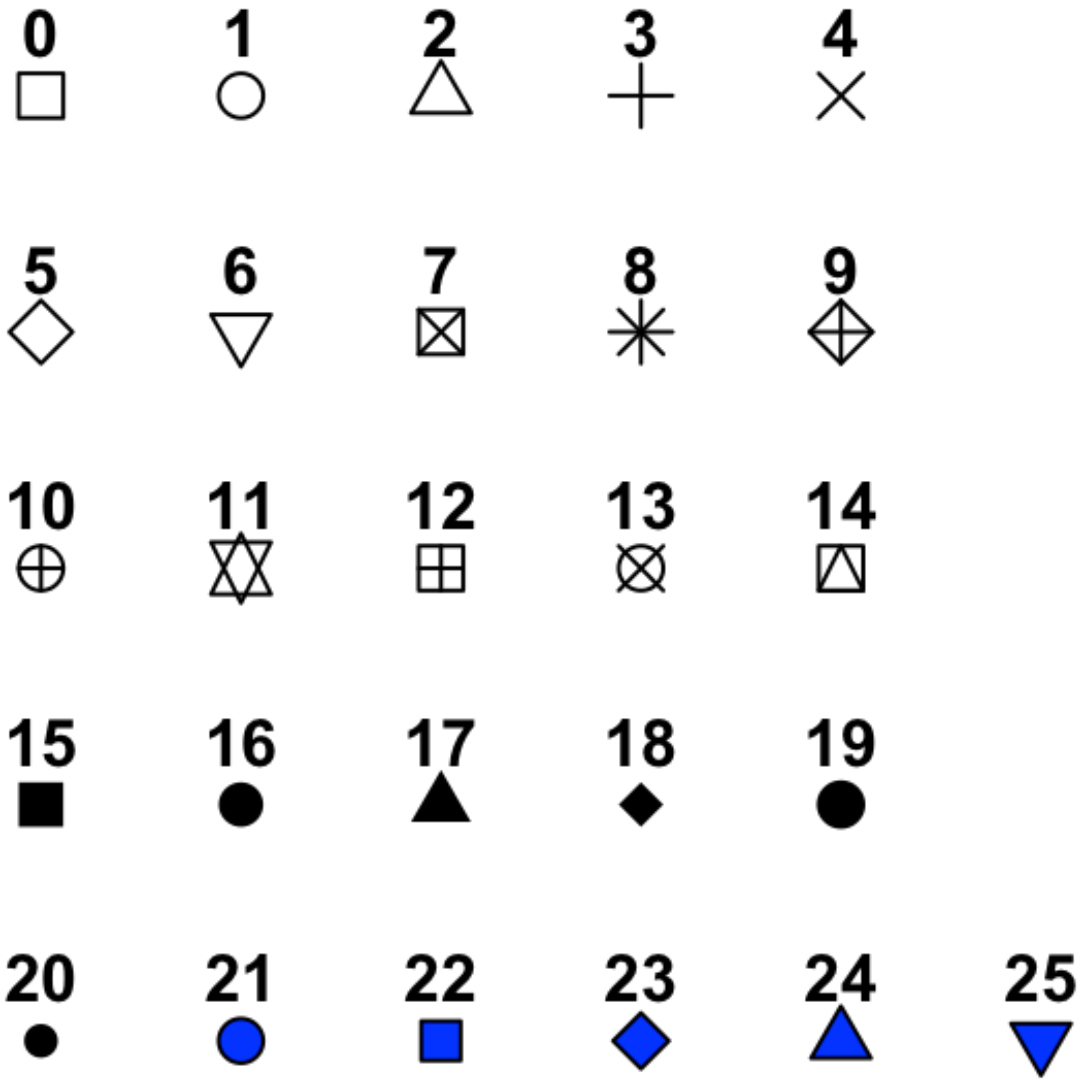
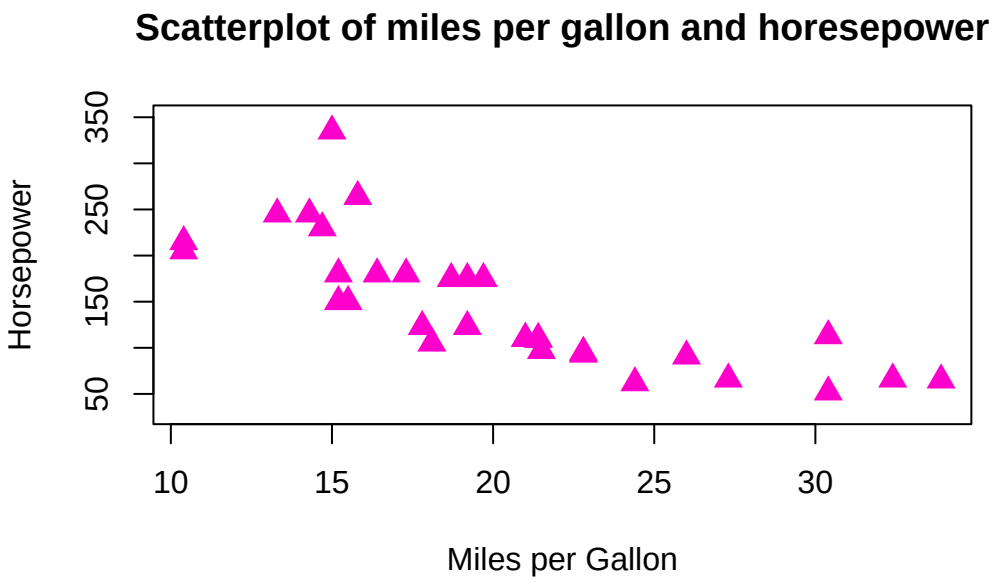


Figure 10.3: Different points that can be used on scatterplots.



You can export the images to certain formats using the corresponding functions - `jpeg()`, `png()`, `svg()`, and `pdf()`, where the first argument is the file path including the file name, and additional arguments that can be used to adjust the width and height of the image. After this command is run, we then create the plot, and then close the plot using the `dev.off()` function.

```
# Step 1: Create export file.

png("Scatterplot1.png",
    width = 750,
    height = 500)

# Step 2: Create plot.
plot(mtcars$mpg, mtcars$hp,
     main = "Scatterplot of miles per gallon and horesepower",
     xlab = "Miles per Gallon",
     ylab = "Horsepower",
     col = "LightSkyBlue",
     ylim = c(30,350),
     pch = 19
     )
```

```
# Close file.
dev.off()
```

10.2 Visualizations in `ggplot2`

One of the best parts of the tidyverse is the `ggplot2` package for data visualization. The 'gg' in `ggplot2` stands for *grammar of graphics,* the namesake of a famous book by Leland Wilkinson. The books sets out a framework for layering elements to construct visualizations. If you've ever used graphic design software like *Photoshop* or *Canva*, you will note a similar idea behind layering different elements together. The creator of `ggplot2` - Hadley Wickham - was influenced by this approach, and extended this thinking in his paper *A Layered Grammar of Graphics.*[2]

One of the main ideas behind the grammar of graphics influence on `ggplot2` is that each graphic can contain many layers of different things, but they all need at least three basic layers satisfied:

- Some **data** that you want to visualize. This is commonly one or more variables from a dataframe.
- Particular **aesthetics** (**aes** for short) which specify the axes and dimensions of the plot.
- A geometric object (**geom** for short) which refer to the type of plot we want (scatterplot, bar plot, etc).

There are other potential layers we can add to customize labels, add relevant statistics, use a different coordinate system, specify subplots, and more. Some of the main layers are presented in Figure 10.4. Note that the bottom three are the most crucial as these form the basic building blocks of a ggplot.

10.2.1 Bar Graphs

Let's return to the `gapminder` dataframe from the package of the same name. In Section 10.1.1 we created bar graphs for population by continent for the year 1952. Let's do the same thing, but for the year 2007, and using `ggplot()` function to create our bar graph. Note that with bar graphs, we need the argument `stat = "identity"` within the `geom_bar()` layer.

```
# First, let's create a smaller dataframe which filters the dataframe by the year 2007 It

sumpop2 <- gapminder %>%
  filter(year == "2007") %>%
```

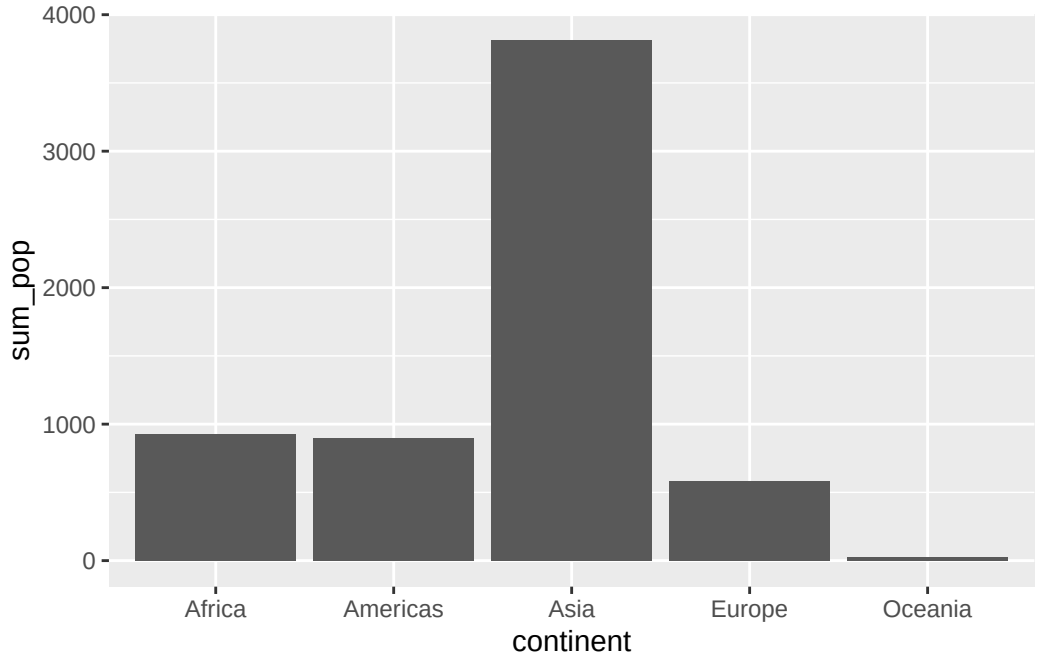
Figure 10.4: Some of the main layers of ggplot2. At a minimum, you need the bottom three layers to create a ggplot.

```
    group_by(continent) %>%
    summarise(sum_pop = sum(pop)) %>%
    mutate(sum_pop = round((sum_pop/1000000), digits = 0))

  head(sumpop2)
```

```
# A tibble: 5 x 2
  continent sum_pop
  <fct>       <dbl>
1 Africa        930
2 Americas      899
3 Asia         3812
4 Europe        586
5 Oceania        25
```

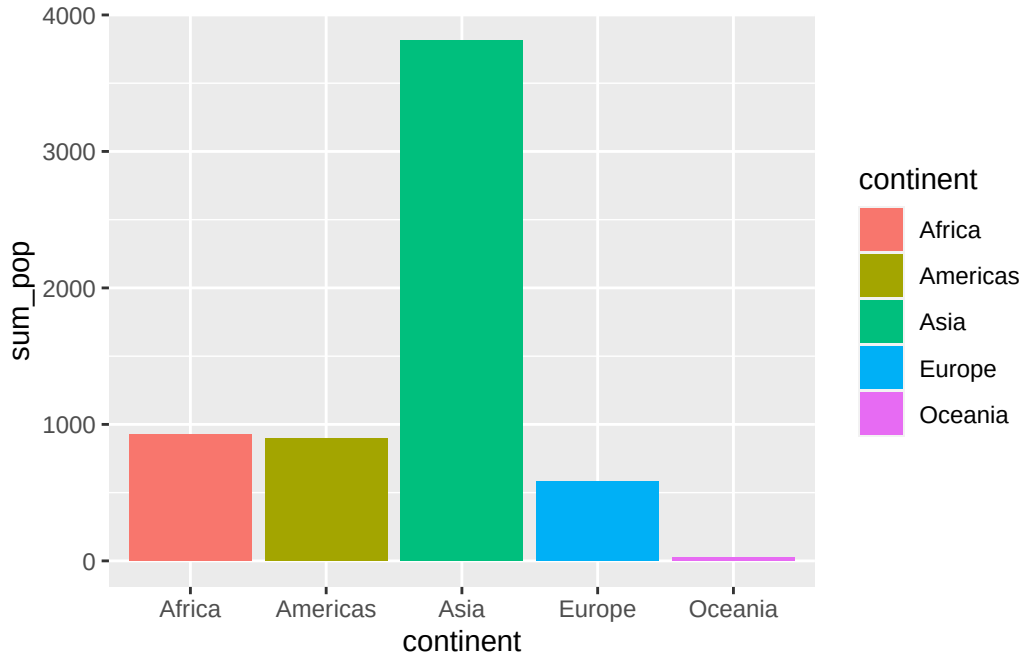
```
  # Let's now create our bar graph to examine population by continent for the year 2007.

  ggplot(sumpop2,                                  ## Data layer
         aes(x = continent, y = sum_pop)) +        ## Aesthetics layer
```

```
  geom_bar(stat = "identity")                ## Geometry layer
```

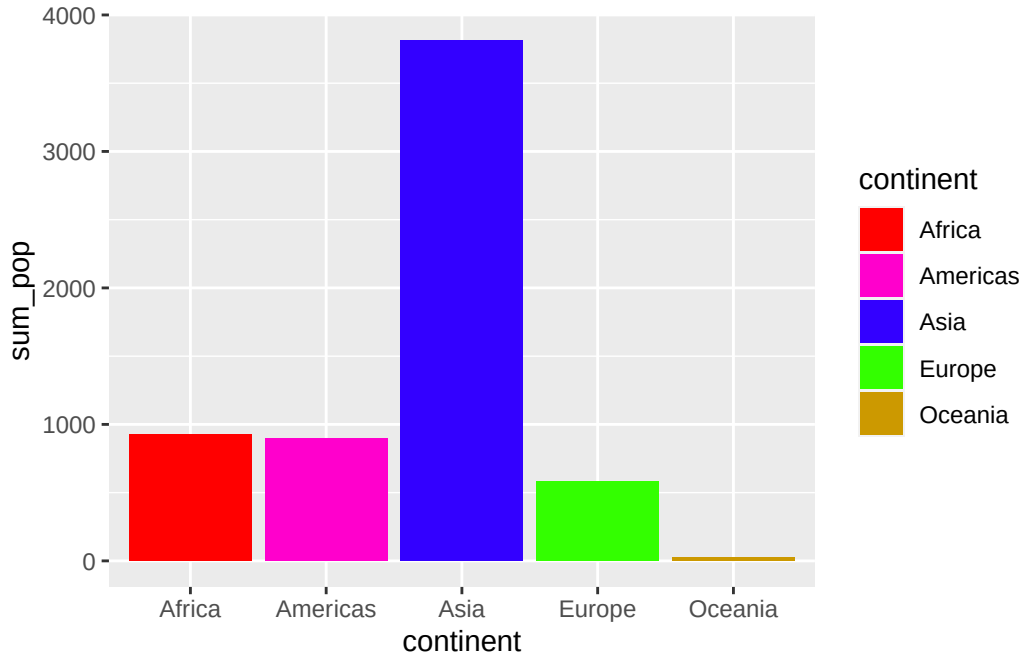


That's the basic bar graph. Let's add some colors using the `fill =` argument in the **aes** layer. We can also modify the colors by name or hex digit using the `scale_fill_manual()` function. We can also use a palette from the `RColorBrewer` package using the `scale_fill_brewer()` function. Additionally, we can also modify the legend position (including to omit it entirely) using the `theme(legend.position = )`argument. If we want to reorder the x-axis categories, we can do that using the `scale_x_discrete(limits =` function. We can also modify axis labels and the title of the plot using the `labs()` layer. Additionally, the grey background of the graph can seem pretty drab. To change the overall appearance of the plot, we can select one of a number of themes (see a list of themes on the ggplot page). Personally, I like the theme called *minimal*.

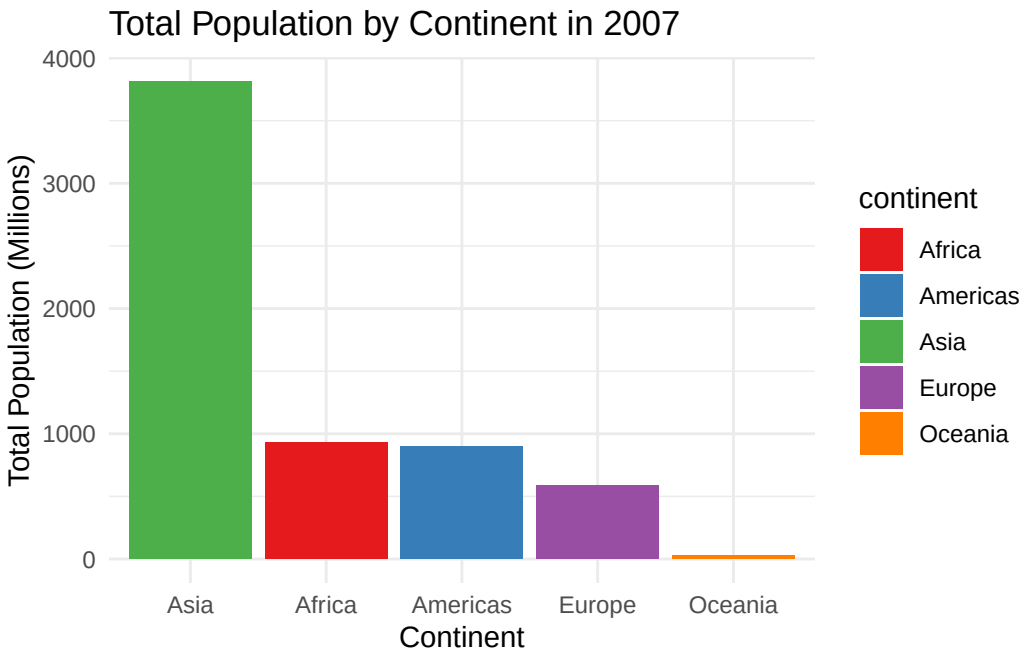
```
# Using the "fill =" argument, we enter the variable we want to color.
ggplot(sumpop2,
       aes(x = continent, y = sum_pop, fill = continent)) +
  geom_bar(stat = "identity")
```




```
# We can change the colors by name or hex digit using scale_fill_manual().
ggplot(sumpop2,
       aes(x = continent, y = sum_pop, fill = continent)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c("#FF0000", "#FF00CC", "#3300FF",
                               "#33FF00", "#CC9900"))
```




```
# Let's use a palette from the RColorBrewer package, reorder the continents on the x-axis

ggplot(sumpop2,
       aes(x = continent, y = sum_pop, fill = continent)) +
  geom_bar(stat = "identity") +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  scale_x_discrete(limits = c("Asia", "Africa", "Americas",
                              "Europe", "Oceania")) +
  labs(title = "Total Population by Continent in 2007",
       y = "Total Population (Millions)",
       x = "Continent") +
  theme_minimal()
```



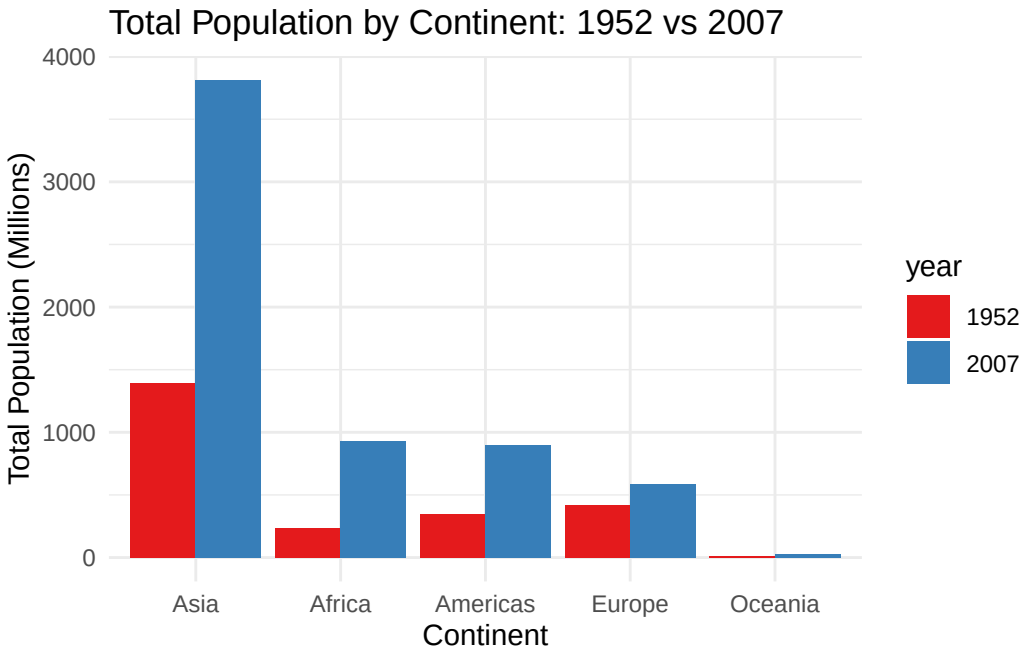
It's starting to look nicer, right? There's plenty more we can do to customize this ggplot, including adding value labels right above each bar, which can improve the interpretability of the plot. We can also another variable in there to compare changes. For instance, let's say we want to look at the population in 1952 vs 2007 for each continent.

```
# First, let's create a dataframe with population for the years 1952 and 2007 by continent
sumpop3 <- gapminder %>%
  select(country, continent, year, pop) %>%
  filter(year == "1952" | year == "2007") %>%
  group_by(continent, year) %>%
  summarise(total_pop = sum(pop)) %>%
  mutate(total_pop = round((total_pop/1000000), digits = 0))

sumpop3$year <- as.factor(sumpop3$year)

head(sumpop3)
```

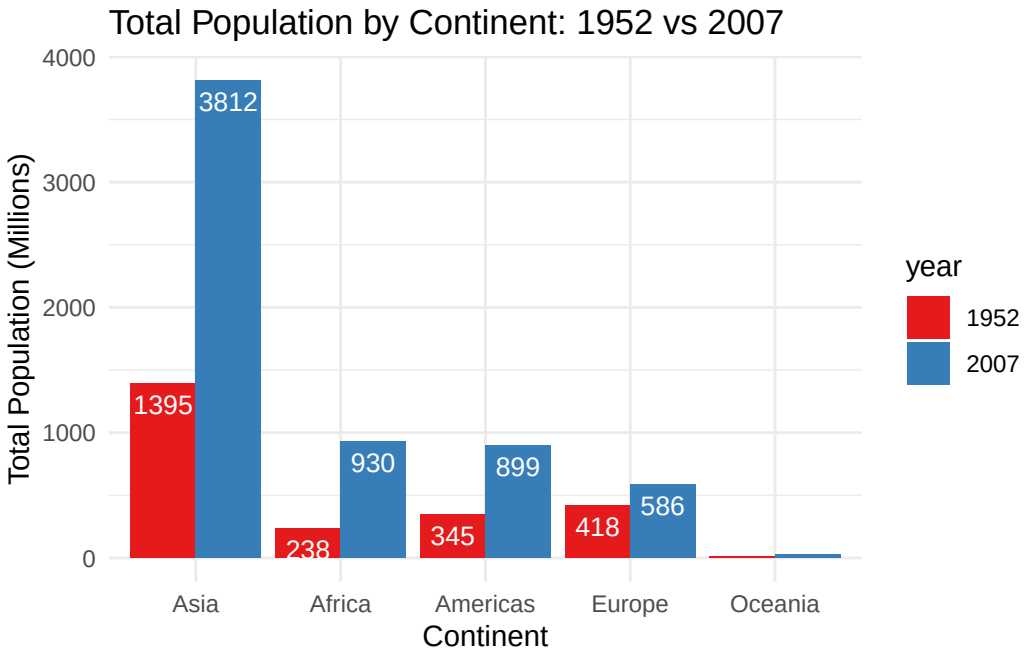
```
# A tibble: 6 x 3
# Groups:   continent [3]
  continent year  total_pop
  <fct>     <fct>     <dbl>
1 Africa    1952        238
```

```
2 Africa    2007        930
3 Americas  1952        345
4 Americas  2007        899
5 Asia      1952       1395
6 Asia      2007       3812
```

```r
# Now let's create side-by-side bar graphs (known as a dodged barplot) using the position

ggplot(sumpop3,
       aes(x = continent, y = total_pop, fill = year)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  scale_x_discrete(limits = c("Asia", "Africa", "Americas",
                              "Europe", "Oceania")) +
  labs(title = "Total Population by Continent: 1952 vs 2007",
       y = "Total Population (Millions)",
       x = "Continent") +
  theme_minimal()
```

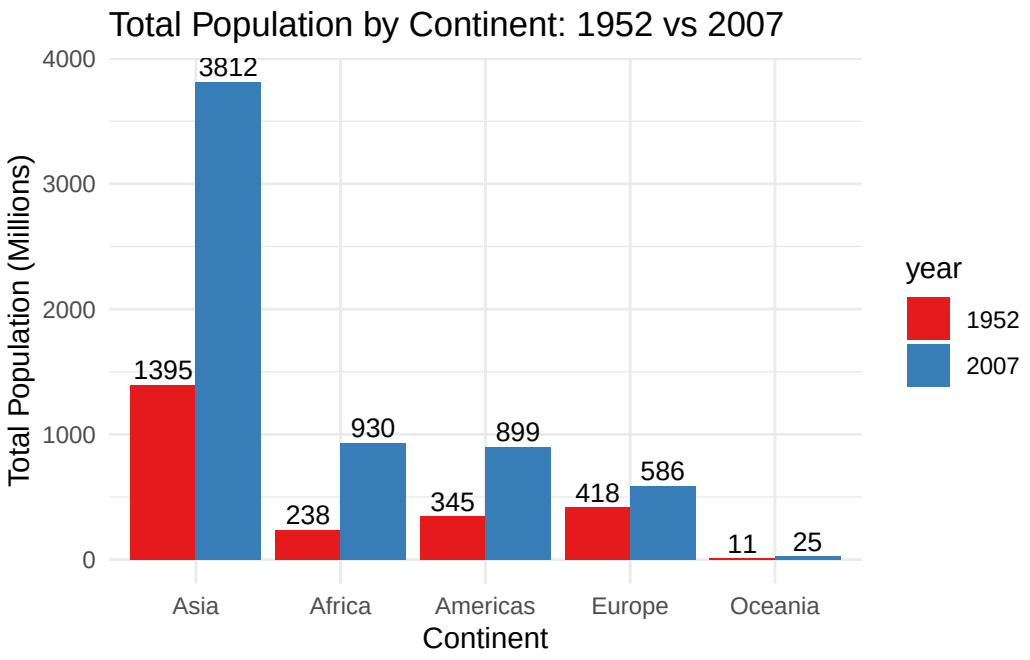


```r
# Now let's add value labels above each bar using the geom_text() layer.

ggplot(sumpop3,
       aes(x = continent, y = total_pop, fill = year)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  scale_x_discrete(limits = c("Asia", "Africa", "Americas",
                              "Europe", "Oceania")) +
  labs(title = "Total Population by Continent: 1952 vs 2007",
       y = "Total Population (Millions)",
       x = "Continent") +
  theme_minimal() +
  geom_text(aes(label = total_pop),
            vjust = 1.6,
            color = "white",
            position = position_dodge(0.9), size = 3.5)
```



Ok, that looks good but because the bars for Oceania are so small, we can't see the value labels if they are placed inside the bar. So let's place them above the bars.

```r
ggplot(sumpop3,
       aes(x = continent, y = total_pop, fill = year)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  scale_fill_brewer(palette = "Set1") +
  theme(legend.position = "none") +
  scale_x_discrete(limits = c("Asia", "Africa", "Americas",
                              "Europe", "Oceania")) +
  labs(title = "Total Population by Continent: 1952 vs 2007",
       y = "Total Population (Millions)",
       x = "Continent") +
  theme_minimal() +
  geom_text(aes(label = total_pop),
            vjust = -0.3,
            color = "black",
            position = position_dodge(0.9), size = 3.5)
```

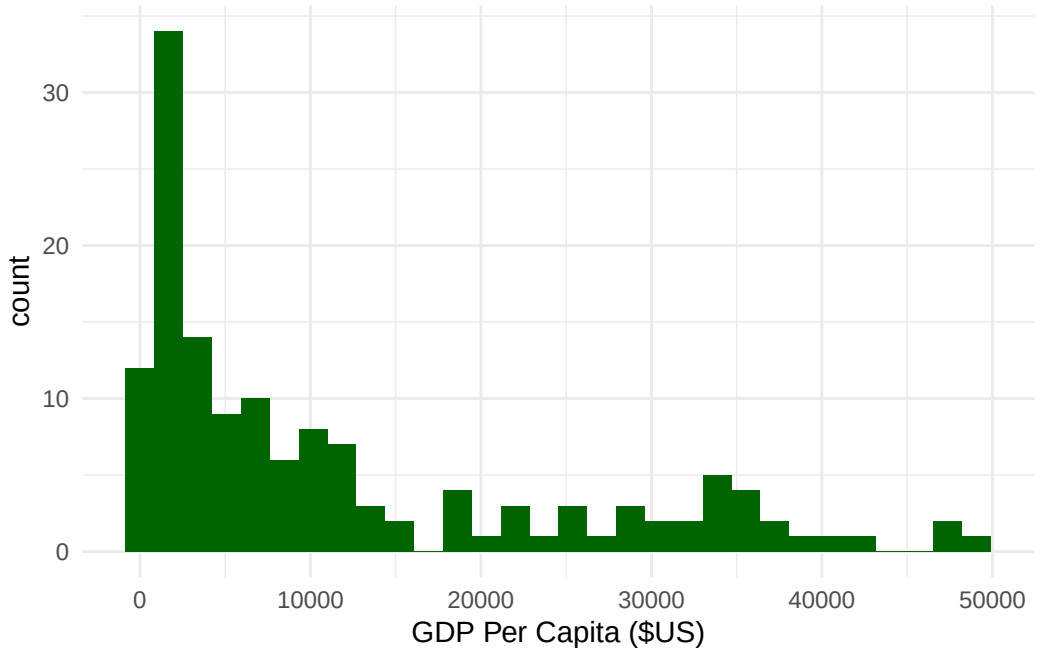


10.2.2 Histograms

Let's say we want to look at the distribution for a numeric variable such as the Gross Domestic Product per capita variable in the original `gapminder` dataframe for the year 2007.

```r
# Let's create a dataframe with GDP across all countries for the year 2007.

sumpop4 <- gapminder %>%
  filter(year == "2007")

# Now let's create a histogram.
ggplot(sumpop4,
       aes(x = gdpPercap)) +
  geom_histogram(fill = "darkgreen") +
  theme_minimal() +
  labs(x = "GDP Per Capita ($US)")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Ok, that's not bad. Notice how R gave us the message `` `stat_bin()` using `bins = 30`. Pick better value with `binwidth` `` ? This means that the default number of bins (or bars) for a histogram in `ggplot` is 30. This message shows up because we didn't specify the exact number of bins. We can get rid of this message by specifying `bins = x` in the `geom_histogram()` layer, where `x` is some number of bins. We can also specify how data is covered by a single bin using the `binwidth =` argument.

How many bins? How wide a bin?

What is the optimal number of bins you should select for a histogram? There is no universally agreed-upon standard. Generally, there is a trade-off between too much and too little detail with histograms, so people often play around by trying different numbers of bins. Some folks have specified some rules of thumb, which we might consider as well. Neither of them are perfect, but they might be a useful starting point.

10.2.3 Sturge's Rule

$$N_{bins} = 1 + 3.322(log_x)$$

where the number of bins N_{bins} is equal to one plus 3.322 times the log of the number of observations in the data log_x. For our histogram above, we have 142 observations of countries' GDP per capita for the year 2007 from the `gapminder` dataframe. Thus, the number of bins we should use according to Sturge's Rule would be:

$N_{bins} = 1+3.322(log_{142})log(142) = 4.9558273.322*4.955827 = 16.46326N_{bins} = 1+16.46326N_{bins} = 17.16 \approx 1$

10.2.4 Freedman-Diaconis Rule

$$binwidth = 2\frac{IQR(x)}{\sqrt[3]{n}}$$

where the width of a bin *binwidth* is equal to 2 times the interquartile range of the variable $IQR(x)$ divided by the cubed root of the number of observations $\sqrt[3]{n}$. For our histogram above, the IQR is 16,383.99. How did I know that? I just used the function `IQR()` with the variable `sumpop4$gdpPercap` as the argument. Thus, the width of each bin we should specify according to the Freedman-Diaconis Rule would be:

$binwidth = 2\frac{16383.99}{\sqrt[3]{142}}\sqrt[3]{142} = 5.21710316383.99/5.217103 = 3140.438binwidth = 3140.438*2 = 6280.876$

10.2.5 Rice's Rule

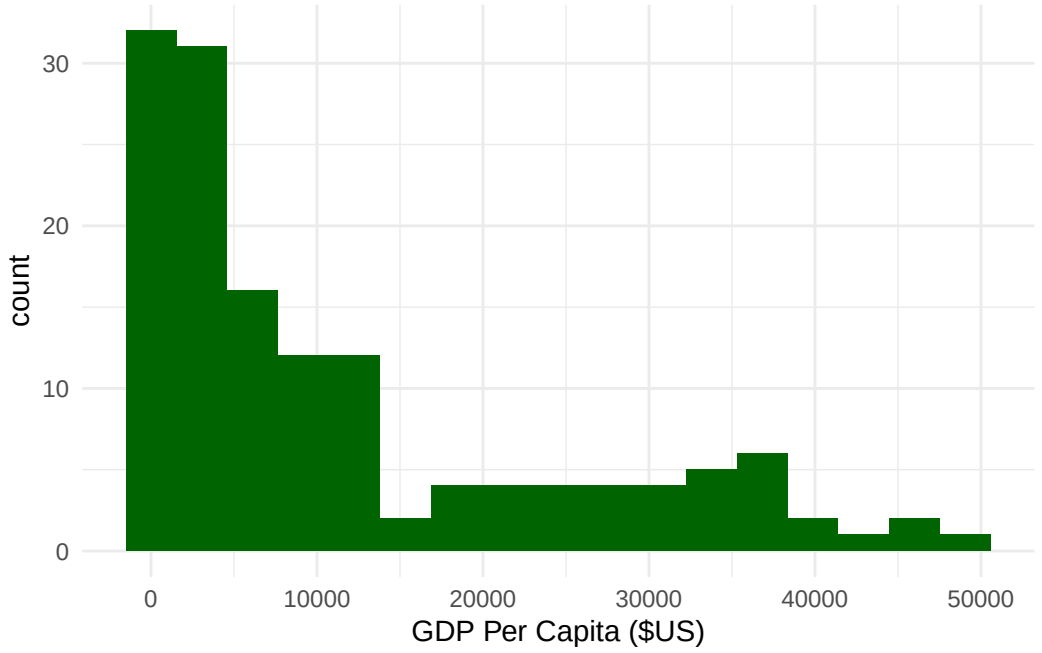
$$N_{bins} = 2 * \sqrt[3]{n}$$

where the number of bins N_{bins} is equal to twice the cubed root of the number of observations $2 * \sqrt[3]{n}$. Thus, the number of bins for our histogram above according to Rice's Rule would be:

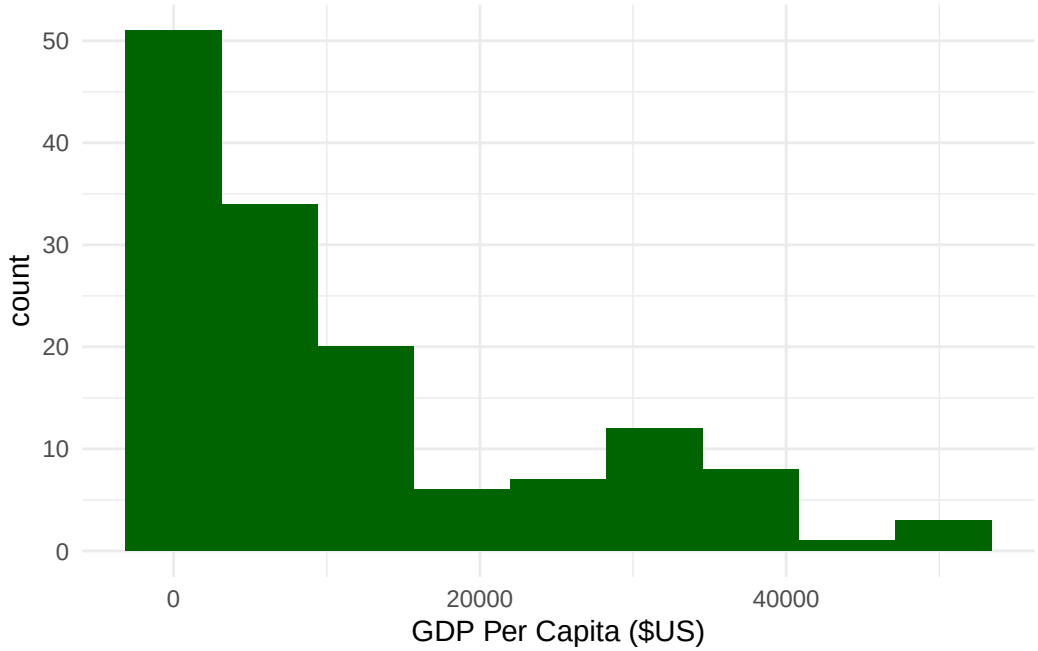
$$N_{bins} = 2 * 5.217103 = 10.43 \approx 10$$

Let's plot and compare histograms using the three rules.

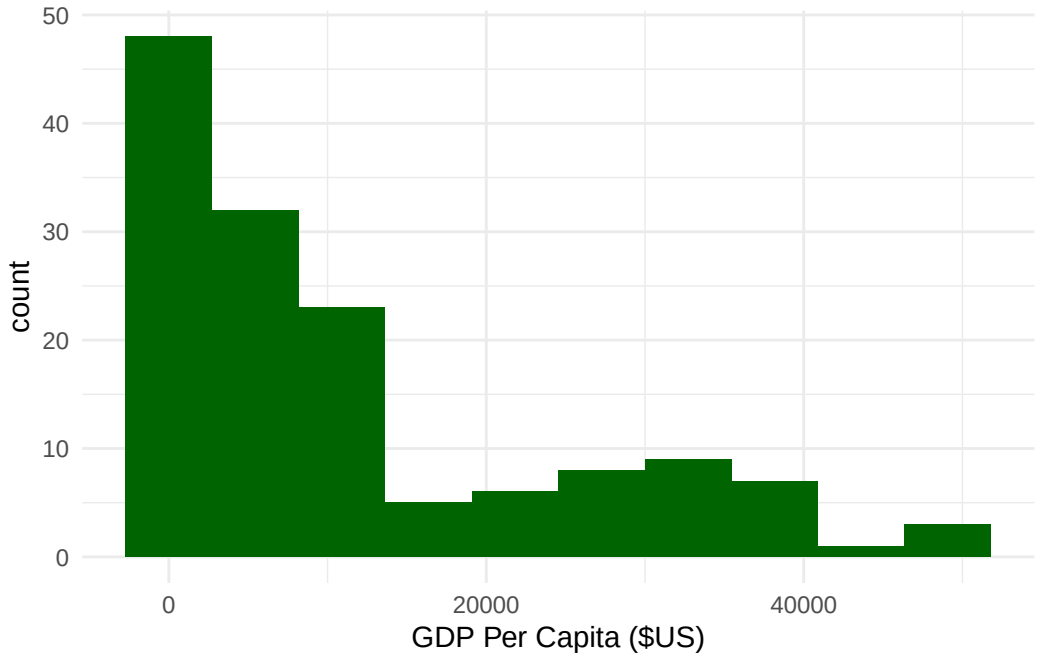
```
# Histogram using N_bins based on Sturge's Rule.
ggplot(sumpop4,
       aes(x = gdpPercap)) +
  geom_histogram(fill = "darkgreen", bins = 17) +
  theme_minimal() +
  labs(x = "GDP Per Capita ($US)")
```




```
# Histogram using bin width based on the Freedman-Diaconis Rule.
ggplot(sumpop4,
       aes(x = gdpPercap)) +
  geom_histogram(fill = "darkgreen", binwidth = 6280.876) +
  theme_minimal() +
  labs(x = "GDP Per Capita ($US)")
```



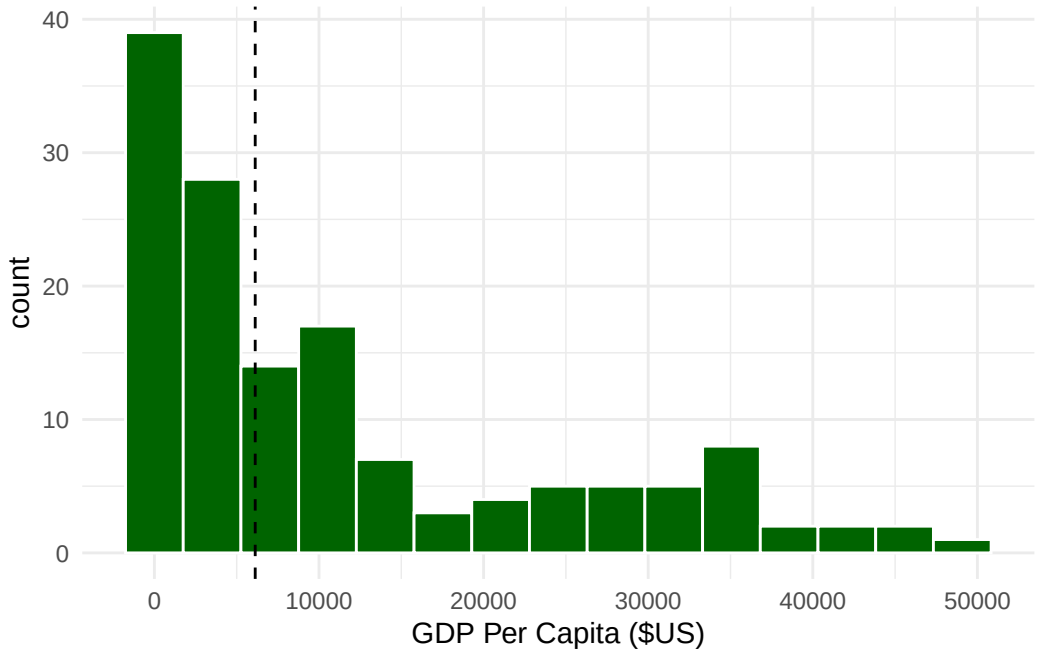

```
# Histogram using N_bins based on Rice's Rule.
ggplot(sumpop4,
       aes(x = gdpPercap)) +
  geom_histogram(fill = "darkgreen", bins = 10) +
  theme_minimal() +
  labs(x = "GDP Per Capita ($US)")
```



It looks like the histograms based on Rice's Rule and the Freedman-Diaconis Rule are fairly similar. You should try different combinations until you find one that looks right to you.

We can also add a vertical line in the histogram corresponding to the mean or median, which can be helpful to see the central tendency of the distribution using the `geom_vline()` layer.

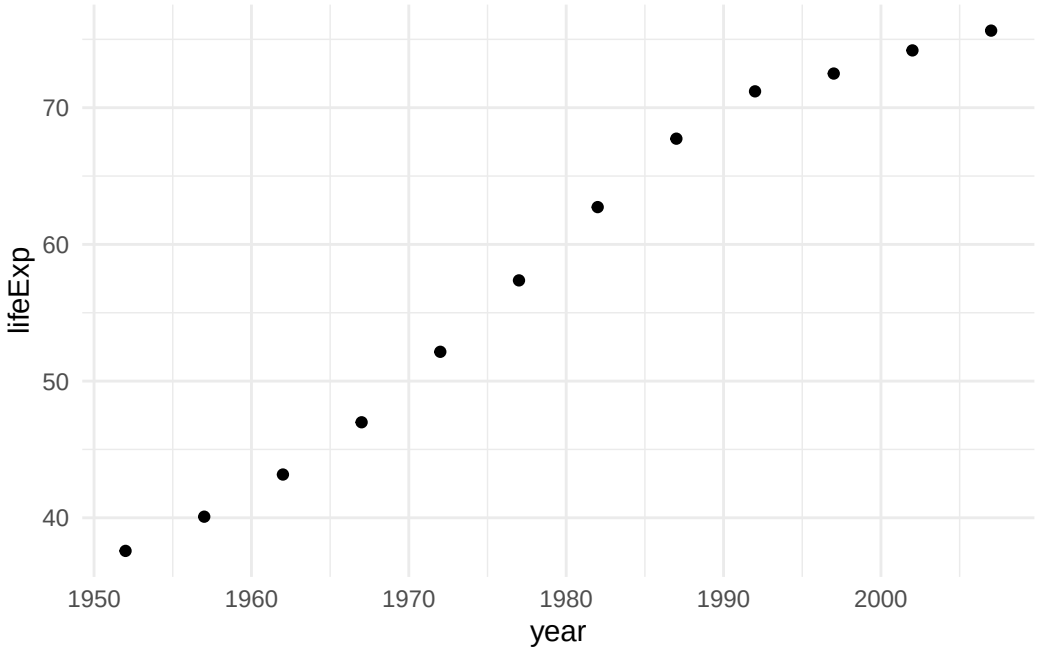
```
ggplot(sumpop4,
       aes(x = gdpPercap)) +
  geom_histogram(fill = "darkgreen",
                 bins = 15,
                 color = "white") +
  theme_minimal() +
  labs(x = "GDP Per Capita ($US)") +
  geom_vline(aes(xintercept = median(gdpPercap)),
             color = "black",
             linetype = "dashed")
```



10.2.6 Scatterplots

I quite enjoy the scatterplots produced by `ggplot2` as they can be very visually appealing. As we know from Section 10.1.3, scatterplots are used to examine the relationship between two numeric variables. Let's say I want to examine the life expectancy over time in the country of Oman from the `gapminder` dataframe.

```
# First, let's create a dataframe just for Oman
lifexpoman <- gapminder %>%
  filter(country == "Oman")

# Next, let's create a basic scatterplot.

ggplot(data = lifexpoman,
       aes(x = year,
           y = lifeExp)) +
  geom_point() +
  theme_minimal()
```



Alright, that's not bad. Let's modify our axis labels, add a title, modify our axes so they show the highest values, increase the size of the points, change the shape of the points, increase the size of the points, and add a loess smoother to the data. Loess refers to *locally weighted smoothing*, which provides a trend line to see patterns in our data more easily. We can add one using the `geom_smooth()` layer.

To change the type of shape, recall that you can specify a shape number from the list seen in Figure 10.3. BUT, You can also use a unicode character within parentheses! This opens up a lot of possibilities, as there are about 149,186 unicode characters available currently, covering different scripts, symbols, and even emojis. Wikipedia has a nice list of unicode characters. Let's get especially wild, and select a unicode character corresponding to the Arabic letter (‘Hah’), which is first letter of the Arabic word *(Hayat)*, which means *Life*. Given that we are examining life expectancy in Oman, where the official language is Arabic, using the first letter of this Arabic word seems appropriate.

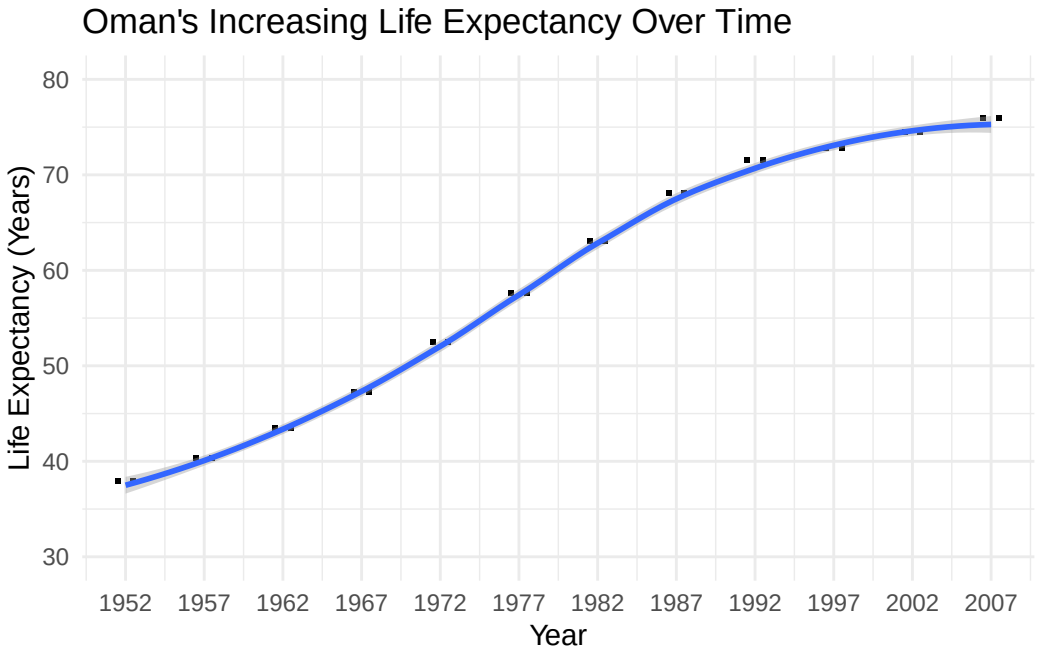
To make things even more interesting, instead of a point, let's also create a histogram where the shape corresponds to the flag of Oman. We can do this using the `ggimage` package's `geom_flag()` function which plays well with `ggplot2`. In this function, there is an `aes` argument that takes a country'stwo-letter ISO Alpha-2 code. For Oman, the ISO code is ‘OM’.

You've already seen how to change axis labels and the title using the `labs()` layer. To change the y-axis of a numeric variable, we can use the `scale_y_continuous()` function, and to change the x-axis, we can use the `scale_x_continuous()` function. These two functions have four important arguments:

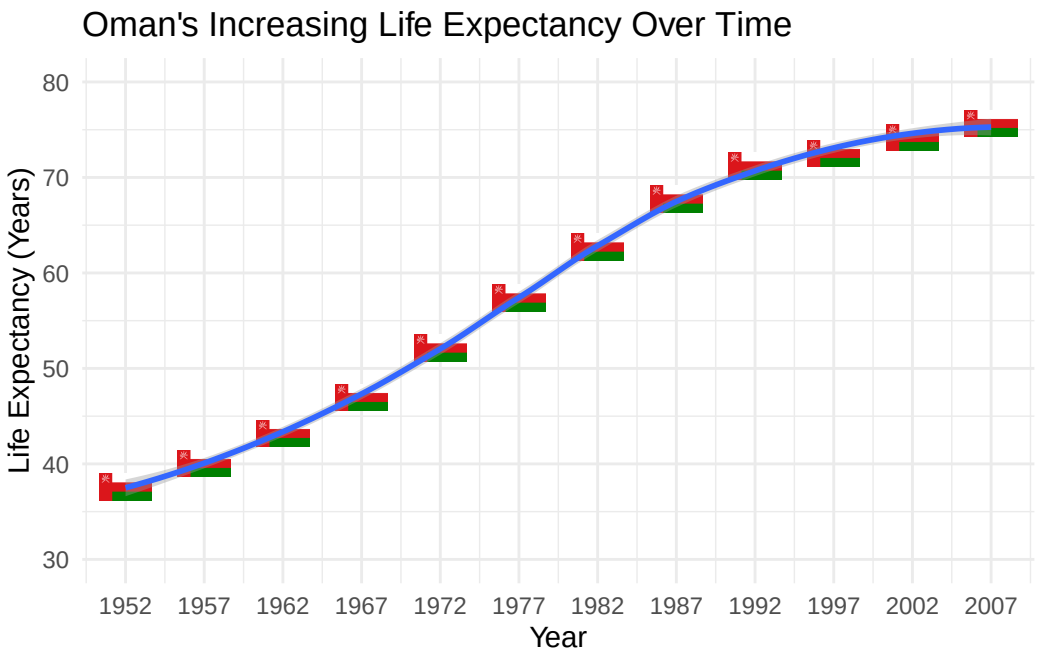
- `breaks` which specifies a vector of numbers to display on the axis.
- `n.breaks` which specifies a number of total breaks on the axis.
- `labels` which specifies a vector of labels to use on the axis.
- `limits` which specifies the range of the axis.

Let's now use these arguments to modify our scatterplot.

```
# Histogram with our changes and using a unicode symbol for the shape.
ggplot(data = lifexpoman,
       aes(x = year,
           y = lifeExp)) +
  geom_point(size = 7,
             shape = "\u062D") +   # Unicode symbol for Arabic letter   )'Hah') from Wikipe
  theme_minimal() +
  labs(title = "Oman's Increasing Life Expectancy Over Time",  # title and axis labels
       y = "Life Expectancy (Years)",
       x = "Year") +
  scale_x_continuous(breaks = seq(1952, 2007, 5)) +   # modifying x-axis
  scale_y_continuous(limits = c(30, 80)) +            # modifying y-axis
  geom_smooth()                                       # Loess smoother
```



```r
# Histogram with the flag of Oman for the shape.
library(ggimage)

ggplot(data = lifexpoman,
       aes(x = year,
           y = lifeExp)) +
  geom_flag(aes(image = "OM"), size = 0.10) +       # This replaces our geom_point layer
  theme_minimal() +
  labs(title = "Oman's Increasing Life Expectancy Over Time",  # title and axis labels
       y = "Life Expectancy (Years)",
       x = "Year") +
  scale_x_continuous(breaks = seq(1952, 2007, 5)) +   # modifying x-axis
  scale_y_continuous(limits = c(30, 80)) +            # modifying y-axis
  geom_smooth()                                       # Loess smoother
```



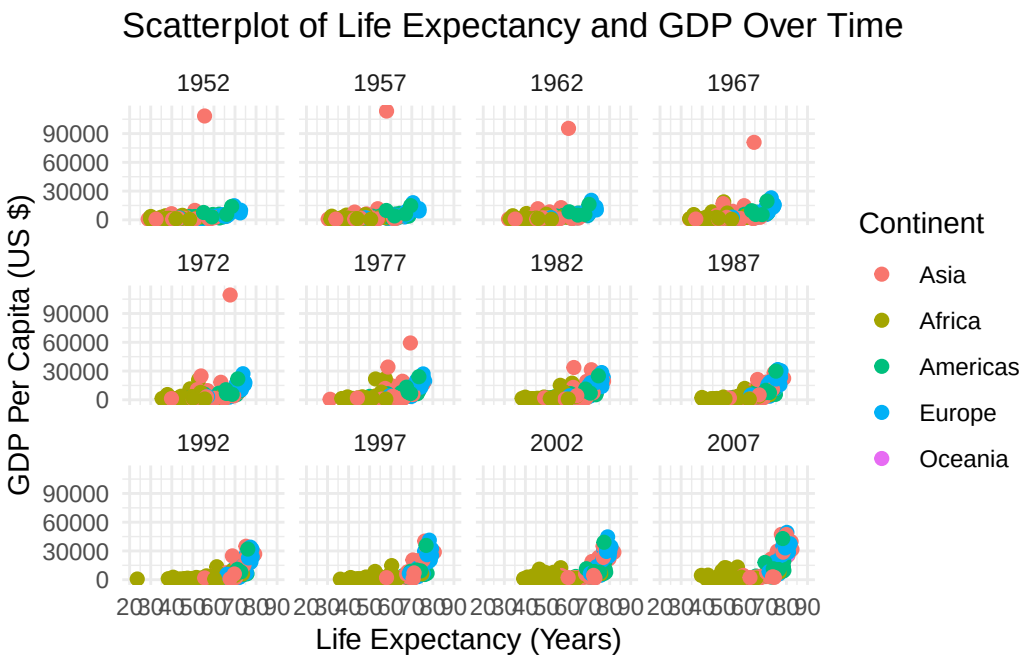
Alright, that looks interesting! We can see a clear trend toward higher life expectancy in Oman over time. We've also seen how to input unicode and use a companion package to modify the shape. Let's illustrate a final few things with the scatterplot in `ggplot2`.

If we want to examine the relationship between two numeric variables by levels of a categorical variable, we can add this as color or shape dimension. In fact, we can add two categorical variables to the scatterplot (as the shape and color dimensions), but this can be make the plot

too busy or overwhelming. Let's examine a scatterplot of life expectancy and GDP per capita among all countries and time points in the `gapminder` dataframe. We'll add a color dimension to examine the relationship by continent. Let's also specify the title for the legend using the `guides()` function. If we want to re-order the categories of the legend, we can simply re-order the categories of the factor variable first.

Additionally, we can create multiple subplots based on a categorical variable. For instance, let's create multiple scatterplots by year to look at the relationship between GDP per capita and life expectancy, with a color dimension for continent. We can create multiple subplots using the `facet_wrap()` function which takes a variable name preceded by a tilda `~` as its main argument. You can also facet on more than one variable, but this can be overwhelming so we'll stick to one variable for now. Finally, let's use `theme_bw()` to see the different subplots clearly.

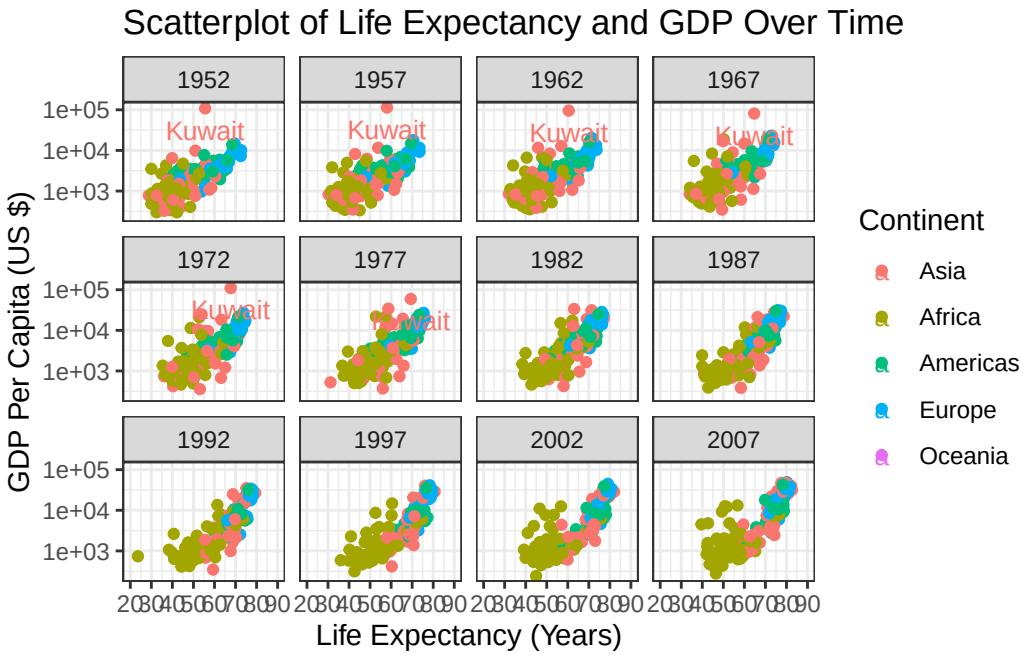
```
# First, let's re-order the factor variable continent in a particular order. Here, I choos
gapminder$continent <- factor(gapminder$continent,
                              levels = c("Asia",
                             "Africa",
                             "Americas",
                             "Europe",
                             "Oceania"))

# Next, can make our scatterplot and add continent as the color dimension.
ggplot(data = gapminder,
       aes(x = lifeExp,
           y = gdpPercap,
           color = continent)) +
    geom_point(size = 2) +            # Makes the points bigger
    theme_minimal() +
  labs(title = "Scatterplot of Life Expectancy and GDP Over Time",
       y = "GDP Per Capita (US $)",
       x = "Life Expectancy (Years)") +
  scale_x_continuous(breaks = seq(20, 90, 10), # Displays ages 20 to 90 by intervals of 10
                     limits = c(20, 90)) +     # Bound the x-axis from 20 to 90.
  guides(color = guide_legend(title = "Continent")) +  # Capitalize Continent in the legen
  facet_wrap(~year)        # Create multiple subplots
```



Alright, that doesn't look too bad. A couple of last things though. First, notice how there's an outlying point in the plots for 1952, 1957, 1962, 1967, 1972, and 1977? It's an outlier in terms of GDP per capita. Let's say we want to label that outlier on our plots. In this case, we can use the `ggrepel` package's `geom_text_repel()` function, which allows us to use tidyverse-style commands to identify outliers based on a rule. This will look clearer once we illustrate this. Second, we can also transform the scale of our axes should we wish to do so. A common transformation is to take the base 10 logarithm (or other logarithm) of axis values. Transformation can sometimes make patterns clearer. So, let's apply a base 10 log transformation to our y-axis using the `trans = 'log10'` argument within the `scale_y_continuous` function.

```
library(ggrepel)

ggplot(gapminder,
       aes(x = lifeExp,
           y = gdpPercap,
           color = continent)) +
  geom_point() +
  theme_bw() +
  labs(title = "Scatterplot of Life Expectancy and GDP Over Time",
       y = "GDP Per Capita (US $)",
       x = "Life Expectancy (Years)") +
```

```
scale_x_continuous(breaks = seq(20, 90, 10),
                   limits = c(20, 90)) +
scale_y_continuous(trans = "log10") +
facet_wrap(~year) +
guides(color = guide_legend(title = "Continent")) +
geom_text_repel(data = . %>%
                  filter(gdpPercap> 50000),
                aes(label = country),
                size = 3.5)
```



Here we see scatterplots for all the years in the dataframe where the outlier is labelled (it's Kuwait!) thanks to `ggrepel`. In general, one should always aim for simplicity and clarity with such visualizations. Aggregate trends (such as combining all countries) can also mask smaller trends, so one should interpret such visualizations cautiously, and ideally paired with statistical analysis.

10.2.7 Violin Plots

Remember how discussed using boxplots to visualize outliers and potentially illogical values in Section 8.5 ? You might be wondering why I didn't discuss boxplots in this section. The main

reason is that boxplots can be misleading because they do not show the underlying distribution. Consider Figure 10.5 from a blog post on boxplots written by the Data Visualization Society.

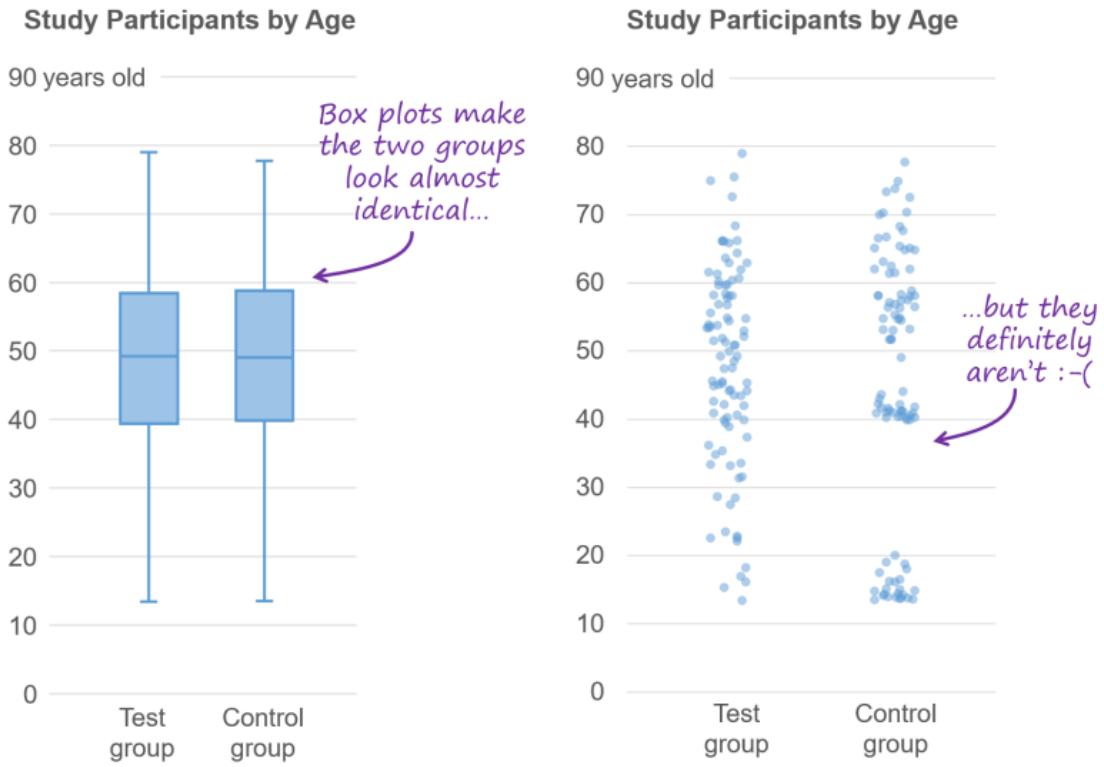
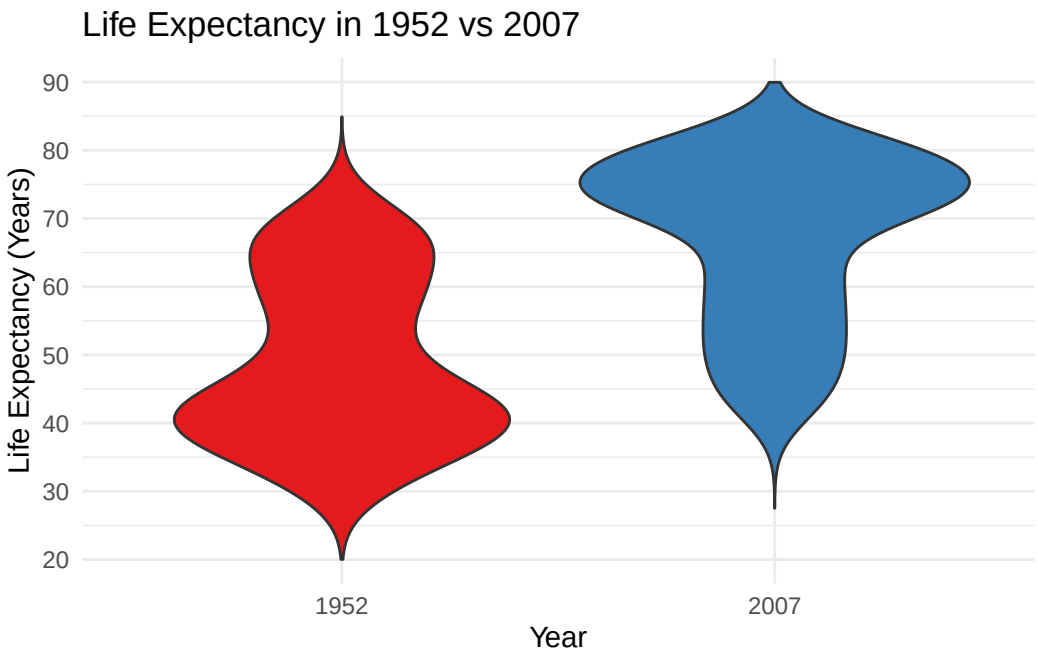


Figure 10.5: Boxplots give you some useful information but also conceal other important information.[3]

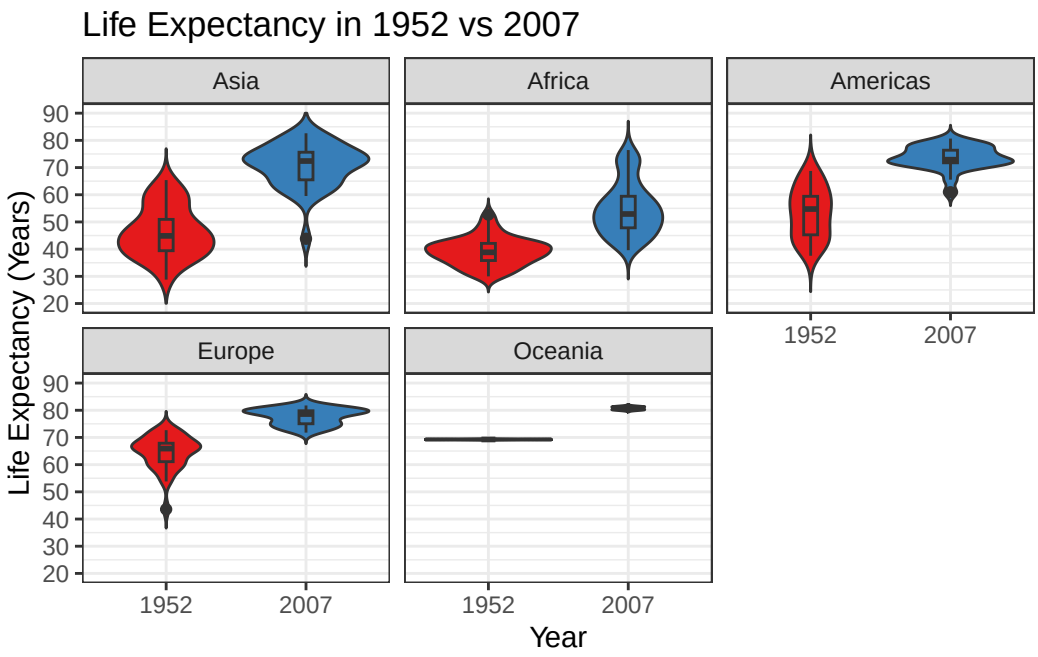
The blog post I've cited in the image caption is definitely worth a read, and explain other pitfalls with boxplots. However, if you would like the information provided by a boxplot AND want to know the underlying distribution, the violin plot is an ideal option. They illustrate the distribution of a numeric variable as a mirror image on either side of a line. They can also have a box plot embedded them. To accomplish this, we can use the `geom_violin()` and `geom_boxplot` functions. Let's create a violin plot to examine the distribution of life expectancy in 1952 and 2007 across all continents, and then separately by continent.

```
library(RColorBrewer)   # Let's use some nice palettes.

# First, let's create a dataframe for the years 1952 and 2007.
lifeexp <- gapminder %>%
  filter(year == "1952" |
```

```
           year == "2007")

# We need to make sure the x-axis variable is factor.
lifeexp$year <- as.factor(lifeexp$year)

# Let's then create violin plots for the years 1952 and 2007.
ggplot(lifeexp,
       aes(y = lifeExp,
           x = year,
           fill = year)) +
  geom_violin(trim = FALSE) +
  theme_minimal() +
  theme(legend.position = "none") +
  scale_fill_brewer(palette = "Set1") +
  labs(title = "Life Expectancy in 1952 vs 2007",
       y = "Life Expectancy (Years)",
       x = "Year") +
  scale_y_continuous(breaks = seq(20, 90, 10),
                     limits = c(20, 90))
```



```r
# Then we'll do two more things - add a boxplot within the violin plot and create subplots
ggplot(lifeexp,
       aes(y = lifeExp,
           x = year,
           fill = year)) +
  geom_violin(trim = FALSE) +
  geom_boxplot(width = 0.1) +
  theme_bw() +
  theme(legend.position = "none") +
  labs(title = "Life Expectancy in 1952 vs 2007",
       y = "Life Expectancy (Years)",
       x = "Year") +
  facet_wrap(~continent) +
  scale_fill_brewer(palette = "Set1") +
  scale_y_continuous(breaks = seq(20, 90, 10),
                     limits = c(20, 90))
```



The violin plots show how the life expectancy distribution across all continents looked quite different in 1952 compared with 2007. In 1952, we can see a clear mode around 40 years, and in 2007 the mode shifted to 75. When broken down by continent, we can see a similar pattern in Asia, and different patterns in other continents.

In this way, the violin plot is a great tool to visualize distributions between groups, and you can think of them as the ideal complement to a box plot.

11 A Very Brief History of Replication

11.1 What is Research Transparency, Reproducibility, and Replication?

By now, you should be swimming along in understanding the preliminaries of R. Hopefully, you are practicing and ensuring you have nailed down all the basics.

While it can be tempting to continue along this road, and immediately jump to the next topic in R, this is actually a good time to slow things down, and understand what we are doing in this course. This is not a course in how to code in R, at least not entirely. This course is also about the very consequential problems and solutions involved in **Research Transparency & Reproducibility (RT2)**. Let's begin with some definitions of these terms from the United States National Academies of Sciences, Engineering, and Medicine.[4]

> **i** Definitions
>
> **Reproducibility** - Obtaining consistent computational results using the sample input data, computational steps, methods, code, and conditions of analysis.
> **Replicability** - Obtaining consistent results across studies aimed at answering the same scientific question, each of which has obtained its own data.

Think of reproducibility as checking the results of the original study. If a researcher gave you their code and data, could you reproduce or recreate their analyses and have them match? That's the hope anyway, but you would be surprised how often that does not happen.

Think of replication as conducting multiple studies, using the same methods and procedures as the original study. If you arrive at similar results, that's good and shows that the underlying theory seems to hold. If not, then it could be a) a problem with the underlying theory, b) a problem with the methods or procedures, or c) both.

Finally, as stated by the National Academies, "Reproducibility is strongly associated with transparency; a study's data and code have to be available in order for others to reproduce and confirm results."[4] Thus, transparency typically refers to availability of data, code, and materials used to produce the results of a study. However, the term conveys more than that. Another definition of research transparency is as the *obligation* to "make data, analysis, methods, and interpretive choices underlying their claims visible in a way that allows others to

evaluation them - as a fundamental ethical obligation."[5] This extends beyond mere availability of data and code to a *fundamental* way of doing ethical research.

The definitions are illustrated in the Figure 11.1.

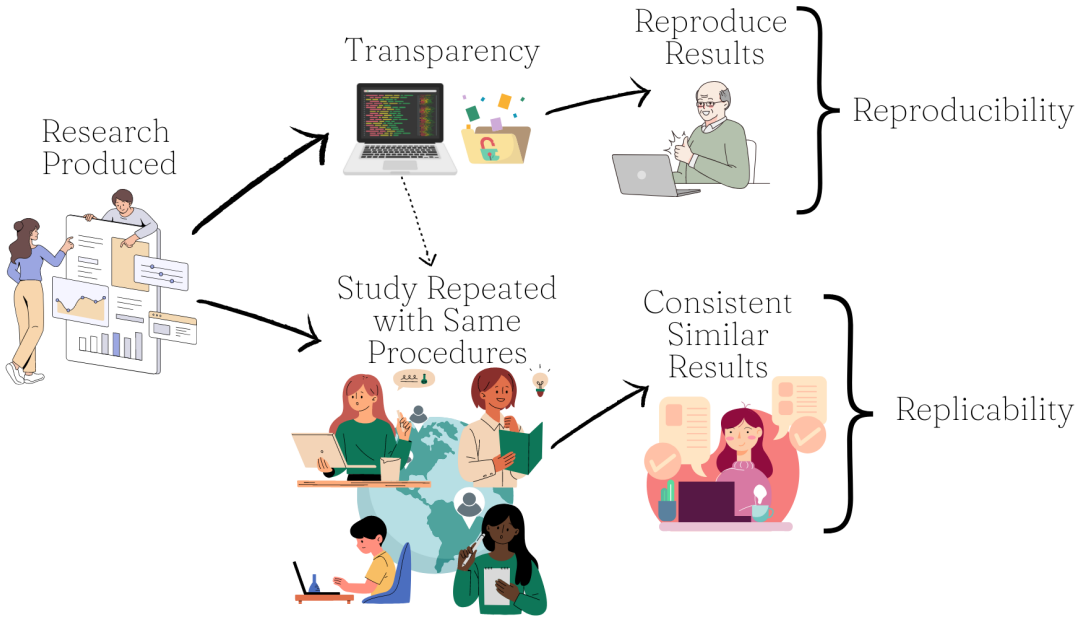


Figure 11.1: Transparency is ideal for replication efforts, but often materials are not provided by authors.

Now that we've got definitions for the terms, let's have a look at some key (and in some cases oft-forgotten) players in the history of replication and reproducibility.

11.2 Replication Origins

The history of replication is closely tied to the history of the Scientific Method. Many individuals over time likely contributed to it, but let's have a look at a few key figures throughout history. The focus here is not on the Scientific Method per se, but rather on the use of replication or repeated experimentation.

11.2.1 Ibn al Haytham

One of the earliest proponents of an approach we later would know as the Scientific Method was Ibn al-Haytham, a mathematician, astronomer, physicist, and all-around scholar who lived

Figure 11.2: Portrait of Ibn Al Haytham by Zargar Zahoor.

from 965-1040 CE. Born in Basra, modern-day Iraq, Al Haytham is known as the Father of Optics for his pioneering work on the subject in his famous work *(Kitab al-Manazir or Book of Optics)*, which provided empirical evidence that vision occurred when light entered the eyes (intromission), rather than light being emitted by the eyes (extramission), which was argued by Euclid in *Optica.*

Besides his many scientific contributions, particularly to the field of Optics, al Haytham made use of *repeated* systematic observation of natural phenomena to argue his points. Even in his magnum opus *Book of Optics*, he often ends a description of an experiment with the phrase "this is always found to be so, with no variation or change", emphasizing that repetition was a central argument of his experimental findings.[6]

While this may not seem that impressive to us today, consider that Al Haytham was working in his way before we had the term *Scientific Method*, and about 500 years before Galileo started looking through telescopes! The important takeaway from Al Haytham is that accidental conditions might distort one's observation of a phenomenon, and we're better off making multiple observations before making any grand claims.

11.2.2 Abu Rayhan Al Biruni

Abu Rayhan Muhammad ibn Ahmad al-Biruni was a Muslim polymath who lived from 973 - 1050 CE. He was known to have authored 146 books, with the majority written on mathematics, astronomy, and related subjects. He was also one of the earliest scholars to offer a formal refutation of astrology in contradistinction to astronomy, which he considered a legitimate science based on empiricism. He made contributions to many fields such as physics, astronomy, geography, and Indology (the study of India).

Importantly, al Biruni contributed to the development of the scientific method, and was one of the first documented scientists who thought about how to separate measurement of a phenomenon from errors in the measurement device. This was remarkably foresighted, as the issue of measurement error is unavoidable in the natural and social sciences. Al Biruni noted that measurement errors can come from different sources such as measurement tools and human judgment, and can compound.[7] To address the issue, he recommended taking multiple measurements of a phenomenon and then quantitatively combining them to arrive at a common-sense estimate. This is an approach we still take today with a number of applications, but especially in the use of multiple survey questions to study latent or unobserved phenomena like attitudes, emotions, and perceptions.

11.2.3 Roger Bacon

Picking up where Ibn al Haytham left off, Roger Bacon was a philosopher and Franciscan friar from England who lived from 1219 - 1292 CE. He took al Haytham's scientific method and applied it to works by Aristotle. He was the first person in Europe to describe the formula

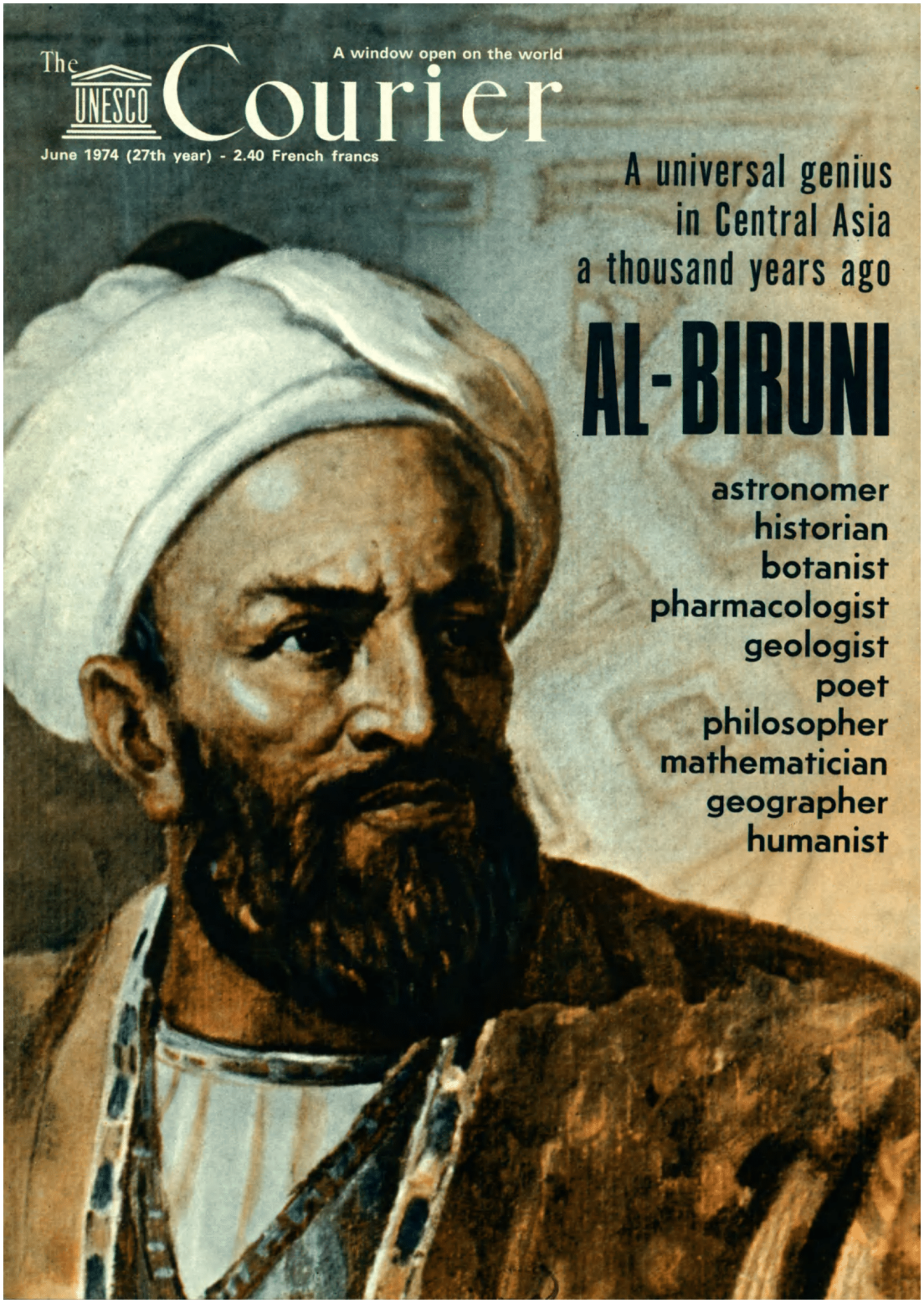


Figure 11.3: Al Biruni on the cover of the UNESCO Courier magazine in 1974.

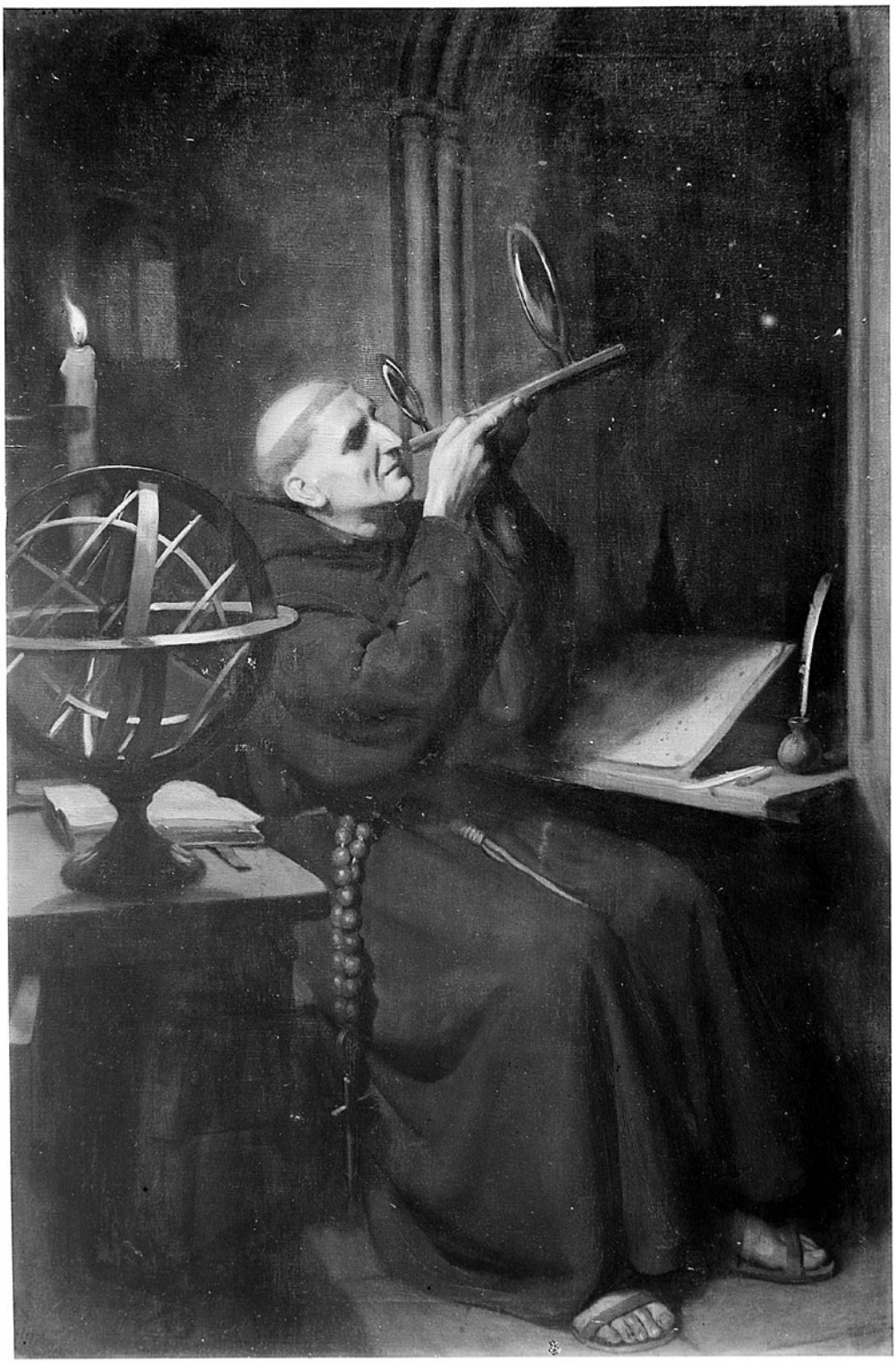


Figure 11.4: Bacon in his observatory in Merton College, Oxford by Ernest Board.

for gunpowder (it was invented and known about in China already). Crucial for our purposes, he was an early proponent of a method of observation, hypothesis, experimentation, and the need for independent verification as noted in his *Opus Tertium,* published in 1267. Regarding the latter, he kept very meticulous notes about his experiments, permitting reproducibility by others. This is exactly what we try to do today!

11.2.4 Andries van Wesel

Andries van Wesel (also known as Andreas Vesalius) as a Flemish anatomist who lived from 1514-1564 CE. He is most famous for creating detailed drawings of human anatomy that provided new insights into the subject, correcting a number of errors made by the Greek physician Galen (129-216 CE), and followed by physicians for centuries.

In the summer of 1542, he published his magnum opus *De humani coporis fabrica libri septem (On the Fabric of the Human Body in Seven Books)*, which was based on a collection of his lectures delivered at the University of Padua, where he was a professor. Based on his own experiences dissecting corpses (not a common practice among physicians at the time), he was able to show that the Galen never dissected a human corpse because he made many errors in human anatomy.[8] Galen instead had relied on dissections of animal corpses, and assumed humans had the same anatomy. Van Wesel was able to show that Galen made 300 errors in human anatomy. Finally, van Wesel was able to create detailed visualizations leveraging newer techniques of printing with woodcut engravings. This was huge, because it allowed other physicians to dissect corpses and verify what van Wesel was claiming. This is one of the earliest examples that I've seen of *evidence-based* medicine triumphing over *eminence-based* medicine. This meant that you didn't have to take his word for it, you could actually dissect a corpse and check if what you saw lined up with his illustrations. This is very much in line with how we think about replication today!

> **i** Bans on (Dead) Bodies
>
> Did you know that dissecting human corpses was strictly prohibited by the Catholic Church in the 16th century? Andries van Wesel had to secretly 'steal' the bodies of executed criminals to perform his dissections. This allowed him to correct many of Galen's errors of anatomy. Have a look at his illustrations here.

11.2.5 Francis Bacon

Sir Francis Bacon was an English philosopher and statesman who lived from 1561 - 1626 CE. He is sometimes called the father of empiricism, but of course we know that he stood on the shoulders of giants who came before him. He put forward a method of inductive reasoning in his magnum opus *Novum Organum (New Method)* which stressed the importance of making

Figure 11.5: Andries van Wesel revolutionized our understanding of human anatomy.

Figure 11.6: Portrait of Francis Bacon by Paul van Somer I.

systematic observations of phenomena, and then to generalize the observations to a few axioms. He also noted that it was important to have a skeptical and methodological approach, so that scientists would not mislead themselves, and to not generalize beyond what the facts demonstrate. The latter sentiment is particularly relevant today as authors of scientific papers frequently discuss broad-reaching implications of their findings, which really cannot generalize beyond their samples. In fact, some have suggested adding a "Constraints on Generality" section to the discussion section of every paper, in which authors clearly state the specific population to which their findings apply.[9] I feel like Sir Francis would approve.

In addition to his many contributions to the formation of the Scientific Method, Bacon also stressed the need for repeated experimentation to create an increasingly complex knowledge base that is always supported by observable facts. This is highly in line with how we think about replication today, as we aim to support or reject theories of how things work. Finally, Bacon also listed several "idols of the mind", which he noted as obscuring the path of correct scientific reasoning. These include the human tendency to perceive more order in a system than actually exists, confusion arising from the scientific use of certain words compared with their common usage, and the tendency to follow dogma and not ask questions about the world. These are what we would today refer to as **cognitive biases**, and they have an important role in inhibiting research transparency and replication, which we will discuss later.

In short, Sir Francis reaffirmed the need for repeated experimentation, careful observation, and meticulous record-keeping in the conduct of science.

While many others contributed to the refinement of the Scientific Method over the next 500 years or so, for our purposes to will skip to the 20th century.

12 Mertonian Norms and Counter-Norms

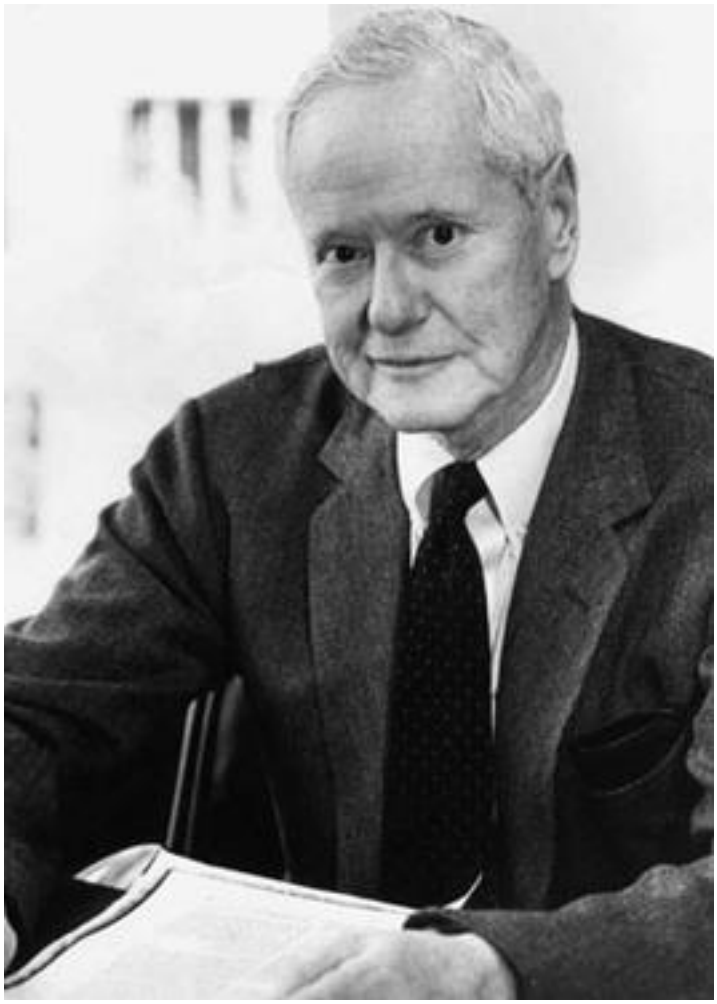


Figure 12.1: Robert K. Merton - the father of modern sociology.

12.1 Background

Robert K. Merton lived from 1910 - 2003 and made a number of major contributions to sociology, criminology, and the sociology of science. Unlike some of the early figures in the history of replication we saw in Chapter 11, Merton advocated for theorizing to begin with *middle-range theory*, or starting with aspects of a phenomenon that are clearly understood, constructed with observed data. He intended this to be a middle ground between a grand theory and a description of a phenomenon. This is remarkably contemporary, as a great deal of middle range theory is about underlying mechanisms that give rise to phenomena.[10]

Reflecting on the history of science, Merton noted that scientists could no longer isolate themselves from the concerns of society.[11] He could have been a poet, because his descriptions of this observation are loaded with powerful imagery that summarize his position nicely. Let's look at a few of these quotes with my crude interpretations interspersed.

Table 12.1: Merton, please forgive me my crude interpretations of your dense, though mellifluous, prose.

Merton's Quote	My Crude Interpretation
A tower of ivory becomes untenable when its walls are under prolonged assault.	Hey scientists, you can only pretend society doesn't exist for so long, until you can't.
A frontal assault on the autonomy of science was required to convert this sanguine isolationism into realistic participation in the revolutionary conflict of cultures.	Hey scientists, when bombs start falling, you better get in the game!
Science is a deceptively inclusive word which refers to a variety of distinct though interrelated items.	Like, what even *is* Science?
The ethos of science is that affectively toned complex of values and norms which is held to be binding on the man of science. The norms are expressed in the form of prescriptions, proscriptions, preferences, and permissions. They are legitimized in terms of institutional values.	Science is done a particular way, and you can see that in a bunch of places.

As you will note, Merton was interested in the way science was done, and with its explicit and implicit beliefs, norms, and aspirations. In short, he was interested in spelling out an **ethos** of science, or the character of ideal scientific inquiry. While Merton made many valuable contributions to social science, we are primarily concerned with his notion of **Mertonian Norms**, which are given by the acronym CUDOS.

12.2 CUDOS

Table 12.2: Mertonian Norms of Science

Component	Description
Communalism (called *Communism* by Merton)	Public ownership of scientific goods.
Universalism	Validity of findings as independent of the status of human subjects.
Disinterestedness	Scientific institutions act for the benefit of a common scientific enterprise, and not for personal gain.
Organized **S**kepticism	Scientific claims should undergo critical scrutiny before being accepted.

Let's make sure we have a good sense of each of his norms (i.e. CUDOS) before proceeding.

12.2.1 Communalism

Merton believed that science should produce goods under common ownership of the community (he called it **Communism** but scientists instead often call this **Communalism** because the former has…certain connotations unrelated to science). The idea here is that a major discovery in science in not the exclusive property of the discoverer. You might get something named after you (known as Eponymy, such as Boyle's Law or Schrödinger's cat), and maybe some recognition, but you don't get to own the discovery as this contradicts the scientific ethic. Secrecy in the antithesis of this norm, and "full and open communication" is related to putting the norm into practice. Merton notes that this norm is incompatible with the definition of technology as private property.

Real-World Applications: Communalism

A study investigated 36 top ranked apps for depression and smoking cessation for Android and iOS in the US and Australia in 2018.[12] The study found that of the 36 apps, 29 shared user data to Google and Facebook, and only 12 of these accurately disclosed this information in a privacy policy. How does sharing findings on users with commercial entities align with the Mertonian norm of Communalism?

12.2.2 Universalism

The acceptance or rejection of particular research claims should be impersonal. It should not depend on the author's race, gender, nationality, alma mater, or any personal characteristic. Merton believed that the impersonal character of science was deep and rooted in **Universalism**. He was also realistic and understood that the larger culture within which science is housed can be antagonistic to the norm of Universalism. In particular, nationalistic bias and related forces can create the situation where "the man of science may be converted into a man of way - and act accordingly."[11] Knowledge cannot be advanced and furthered if scientific careers are restricted on grounds other than competence, according to Merton. It shouldn't matter who you are as long your work is competent and adheres to the standard of quality in a field. Unfortunately, implicit and explicit can creep in during process of hiring researchers, the peer review process, and the consumption of research. This is why many journals operate using a double-blind process, whereby neither the authors nor the reviewers know the identity of the other.

Less than Universal - A Personal Anecdote

When I was an undergraduate research assistant at the University of Michigan in the US, I once brought a paper to the attention of my supervisor, who was a young PhD student educated in the US. When they asked what journal the paper came from, I replied "the

Indian Journal of Psychology." Without looking at the paper, they said "Hmm, see if you can find a paper from a better journal." I dutifully went looking for another paper, but not before reflecting "Why didn't they look at the study before making a decision on its quality?" As an Indian studying in the US, this experience illustrated the mental shortcuts frequently made in academia, illustrating a stark deviation from the norm of Universalism.

When I first read about the Mertonian norm of Universalism, my first thought was of a specific scene in Antoine de Saint-Exupéry's classic *Le Petit Prince [The Little Prince]*.[13] The scene is presented in Figure 12.2 .

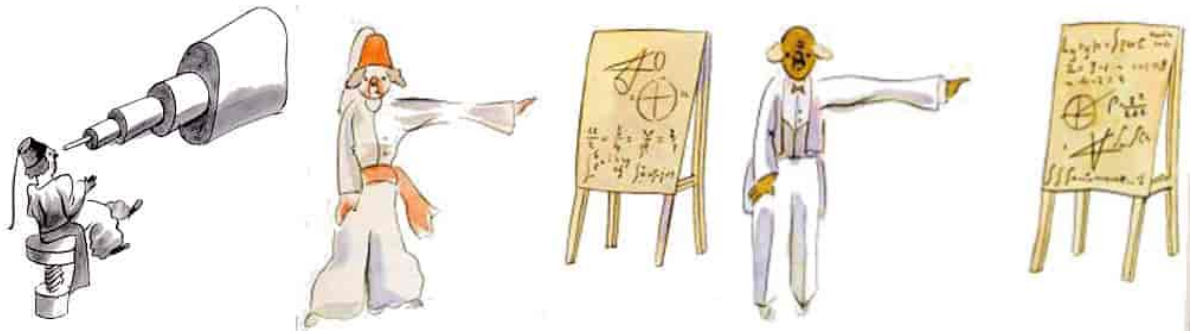


Figure 12.2: Le Petit Prince (1943). *Reynal & Hitchcock*. No one believed the astronomer when he wore non-European clothing. But when he donned a three-piece suit, they listened then. A violation of the norm of Universalism.

In this scene, a Turkish astronomer who discovered an asteroid and tried to present the results of his discovery to his peers at an international astronomy conference. However, his peers did not believe his findings because he was dressed in traditional Turkish garb of a fez and a billowing set of shalwar pants. Years later, a Turkish dictator forces his subjects to wear European-style clothing under pain of death. This time, the Turkish astronomer presented his findings at the same conference but with a more European outfit. His peers then immediately believe his findings.

12.2.3 Disinterestedness

When it comes to science, researchers shouldn't be motivated by commercial pursuits, political interests, or the desire for individual status and prestige. The norm of **Disinterestedness** is about individual motivations to do science, and that these should be for the sake of science itself. I see this more as *Purity of Intention*. Merton noted that the norm of Disinterestedness is supported by the fact that scientists are accountable to their peers. If you're going to do partisan politics under the guise of science, your professional peers are going to point that out, and illustrate how its not science. In essence, your colleagues will largely see through you and understand what you are doing.

This is the most aspirational norm, if you ask me. Consider that in academia hiring and promotion is usually heavily influenced by how many papers you publish or how much grant money you can secure. In fact, the whole purpose of this course is to show how to do ethical programming and statistical analysis, with a number of illustrative cases where this is not the case. Additionally, consider the incentives of funders of research as well, especially private organizations that have their own motives and priority areas. While this norm is not especially realistic, it's a good thing to strive for, and an important one when it comes to the interpretation of data.

12.2.4 Organized Skepticism

The norm of **Organized Skepticism** is an institutional and methodological mandate, according to Merton. Basically, for science to work, this *has* to be part of the process. Science is concerned with a *detached scrutiny of beliefs in terms of empirical and other logical criteria*. This is related to the process of peer review, in which experts provide critical scrutiny for articles submitted to scientific journals. Merton points to the fact that this norm has often involved questioning the ways things are done, and established beliefs. This can sometimes lead to negative consequences for scientists that question the power assumptions of certain institutions.

12.3 Counter-Norms

Merton's norms have been quite influential in the sociology of science field, and later work by Mitroff produced a useful list of counter-norms to Merton's norms.[14] These counter-norms are the opposite of Merton's norms, and nicely capture how science is often conducted in reality. We can refer to them when thinking about the conduct of science as falling on a spectrum with the poles being the norms and counter-norms.

12.3.1 Particularism (in contrast with Universalism)

The idea here is that the acceptance or rejection of a claim depends in large part on the characteristics of who makes the claim. Recall the Turkish astronomer in the *Petit Prince* example, and how people accepted his claims only after he donned the clothing they deemed to be acceptable. If someone *prejudges* the claims of scientist from Harvard University as being automatically more credible and valid than those of someone from Jadavpur University in Kolkatta (where my dad went to college incidentally), India without actually seeing their

Figure 12.3: Galileo's Organized Skepticism was not appreciated by the Catholic Church. Wikimedia Commons.

claims, this would be a classic case of **Particularism**. In fact, they might be implicitly biased against the claimant from Jadavpur University even if they read their study.

Sadly, given what we know about human judgment and how often it relies on quick, mental shortcuts, this counter-norm appears quite realistic. The key then is to design systems of review (such as double-blind peer review) that acknowledge the possibility of Particularism.

! Racial Discrimination as a Form of Particularism

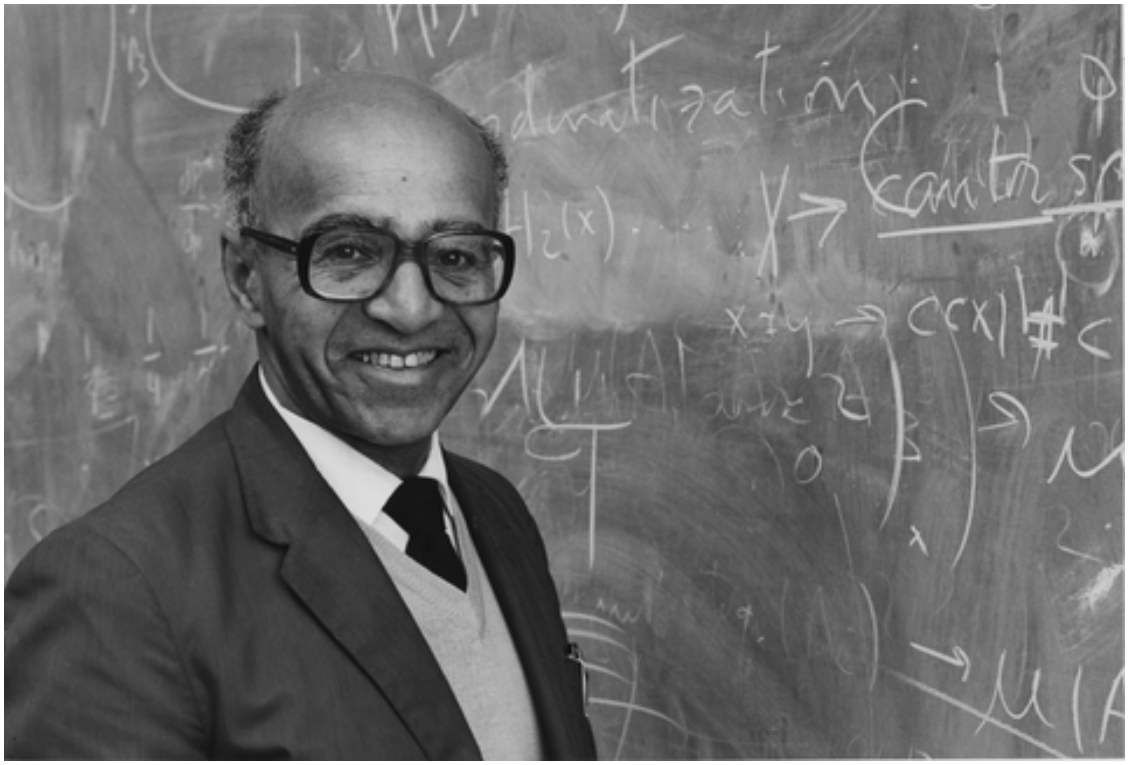


Figure 12.4: Professor David Blackwell - A Trailblazer in Statistics. The Bancroft Library, UC Berkeley Library.

Racial discrimination is a commonly-seen form of Particularism whereby scientists are judged on the basis of their race, rather than on the credulity of their claims. David Blackwell was a very smart man, and made many important contributions to statistics and mathematics including to game theory, probability theory, and Bayesian statistics. He was true luminary in the field, and blazed a trail for others as the first African American elected to the National Academy of Sciences.
Blackwell was a postdoc at the Institute of Advanced Study (IAS) in Princeton in 1941. Usually, an IAS postdoc would receive a Visiting Fellow appointment at Princeton Uni-

versity, but Blackwell was not afforded this because he was Black. The President of Princeton University was unhappy that IAS had admitted a Black man in the first place. In 1942, Blackwell had interviewed for a position at UC Berkeley. His PhD advisor Joseph Doob told Jerzy Neyman, the founder of the UC Berkeley Department of Statistics, that among his good students "***...Blackwell is the best. But of course he is black. And in spite of the fact that we are ina war that's advancing the cause of democracy, it may not have spread throughout our own land.***"[15] This was a pretty spot-on observation by Doob, as Blackwell was not hired at UC Berkeley. The reason? The wife of the department head refused to have a Black man in her house because new faculty members customarily attended dinner at the department head's house.
Blackwell went on to teach at Howard University for 10 years, and also worked at the RAND corporation. In 1954, Blackwell was hired at UC Berkeley at last, and stayed there for the rest of his long and storied career.

12.3.2 Solitariness (in contrast with Communalism)

Secrecy is the name of the game with this counter-norm. Here, property rights are extended to scientific discoveries. This is actually what one commonly sees when business interests intersect with academic ones. Think about Big Tech companies. They have a strong financial incentive to deliver value to their shareholders through monetizing and protecting their intellectual property. That these are also scientific discoveries illustrates a deviation from the norm of Communalism.

The lines are not always so stark however, and my personal opinion is that there is room for scientific communalism as well as commercial privatization in a number of cases. For example, the founders of Google, Larry Page and Sergey Brin, authored a journal article in 1998 in which they break down their famous PageRank algorithm.[16] This provided a valuable contribution to the computer science literature, while also forming the basis of Google Search, which still accounts for just over half of Google's revenues.[17]

12.3.3 Interestedness (in contrast with Disinterestedness)

In contrast with Disinterestedness, this counter-norm illustrates the temptation of self-interest and prestige as a chief motivating factor behind scientific discovery. Scientists then work not to for the sake of science, but to secure more funding, more renown, and better positions at more prestigious institutions. This is one of the more realistic counter-norms, in my opinion, as hiring and promotion tend to emphasize a scientist's output, rather than a dedication to science for the sake of science. The culture of **Publish or Perish** means that you NEED to be cranking out papers to get a tenure-track position, and certainly to achieve tenure. Consider

the implications of this incentive - the system will reward you for papers (be they fraudulent or improperly produced) rather than for producing transparent and robust unpublished work.

This incentive creeps into the academic process in a number of ways. Consider how the prestige of a scientific journal often hinges on how well-cited and influential its articles are, and how this encourages more sensational, novel papers rather than carefully-crafted replications or less *sexy*, though important, findings. Journals are more likely to publish findings that are statistically significant than findings that are not statistically significant, and this phenomenon is known as **Publication Bias**.[18] This leads to all sorts of distortions of the evidence base which we shall discuss later in the course.

! Self-Citations: Creating your own Fame

In 2018, a prominent professor of human development, Robert Sternberg, resigned as editor of the journal *Perspectives on Psychological Science*. His colleagues were not happy that he kept citing himself and publishing in the journal in which he was the Editor. In one of his articles, 10/17 citations (~59%) are his own.[19] Other articles had similarly troubling levels of self-citations.[20] He was also criticized for publishing several commentaries in the journal without peer review, and with the benefit of garnering multiple citations.
Citing your own work is not a bad thing on its own, but when the majority of your citations are your own, that's usually a problem! Consider how it may indicate an echo-chamber approach to science, where you are not actually doing the work of reviewing the literature. The case of Robert Sternberg illustrates how once a habit becomes entrenched, it can be very difficult to see the shortcomings and to stop doing it.

12.3.4 Organized Dogmatism (in contrast with Organized Skepticism)

Under this counter-norm, scientists take all the credit for their discoveries, while placing the blame for any inadequacies on previous work by others. Far from doubting their own findings, under **Organized Dogmatism** scientists promote their own findings with a deep conviction, regardless of what others are saying or doing.

i Dogmatism: The Ultimate Close-Minded Approach

A useful definition of Dogmatism is provided by the late psychologist Milton Rokeach:
A relatively closed cognitive organization of beliefs and disbeliefs about reality, organized around a central set of beliefs about absolute authority which, in turn, provides a framework for patterns of intolerance and qualified tolerance toward others.[21]
In sum, dogmatism sounds like this "I'm right. I know I'm right. Anyone who says otherwise is wrong."

Imagine working on a specific area of research for most or all of your career. You become THE leading expert in the topic, and others defer to your authority…for a time. Just as you start to wane in fame, a young hotshot is putting out research that directly critiques your fundamental findings. Will you take an organized skeptical approach and re-assess your work (you should!), or will you instead fight back vociferously and vocally, in spite of mounting evidence against you? While I would love for scientists to be detached enough to scrutinize their own work even if they are the leading expert on it, unfortunately when one's identity becomes intertwined with their work, it can be quite difficult to be objective.

How do you avoid Organized Dogmatism? Lead with intellectual humility. One excellent example of this is psychologist Julia Rohrer's Loss of Confidence Project in which she and others admit mistakes that they have made in their previous research. This is a remarkable departure from the tempting counter-norms of Organized Dogmatism and Interestedness, because instead of lauding one's own accomplishments, it publicly declares instances of one's errors. Normalizing errors and reporting them is a noble effort, and one that can fight against the counter-norms impeding the progress of science.

As pointed out by Vox science and health editor Brian Resnick,[22] there are at least three key challenges to humility (which I paraphrase):

- Our minds (even the geniuses among us) have blindspots and are more imperfect than we are willing to admit.
- We need a culture that celebrates the words "I was wrong."
- We will never achieve perfect intellectual humility, so we need to be thoughtful with our convictions.

12.4 Conclusion

In short, Mertonian Norms are aspirational and admirable. We should strive to do science in accordance with these norms. However, let's also be real and recognize that the counter-norms often motivate our behavior. We should be cognizant of them, and engage in habits and practices that actively minimize their influence on our behavior such as transparency and humility, and should strive to enact systems that are consonant with Mertonian Norms such as double-blind peer review, and keeping a lid on self-citations.

13 Some Statistical Preliminaries

Before we can proceed with understanding a pivotal moment in the history of Research Transparency & Reproducibility, let's make sure we're on the same page with some statistical preliminaries. These are some basic concepts in **classical or frequentist statistics** that are foundational. They are highly relevant to the data analysis we will be performing in R, but also to the way data analysis is conducted, and the problems therein. Even if you have taken an introductory statistics course or higher, this should still be a useful review, as many basic statistical concepts are frequently misunderstood, even by experts.

What do you mean "Frequentist Statistics?"

If you read the paragraph above, you might be wondering why I made the words **classical or frequentist statistics** in bold. This is because the way most folks have been doing statistics for the last century or so is based on a framework of inference based on long-run probabilities (also called frequencies). What does this mean in practical terms? Essentially, the philosophical approach to probability is premised on the assumption of *repeated sampling from the population,* or repeating the study again and again (to infinity).
The statistics we will be discussing in this course are all based on the frequentist paradigm, because that is still the way most scientists do statistics, and it's definitely worth knowing. If you are interested, an alternative approach to probability that pre-dates the frequentist paradigm, and is recently gaining a lot of ground across the sciences is the **Bayesian** paradigm, stemming from a theorem by English Presbyterian minister and statistician Thomas Bayes (1701 - 1761). In the Bayesian approach, probability is related to our degree of belief in something happening. If we are trying to estimate a parameter (let's say the average height of all Swedes), the Bayesian approach treats this parameter as a random variable, and estimates of this parameter are influenced by our prior knowledge of Swedes' height. In the frequentist approach, the parameter we are trying to estimate is fixed, and we can eventually estimate it given enough studies sampling from the population of Swedes.

Figure 13.1: Thomas Bayes' theorem was the catalyst for Bayesian statistics, which has since gained a lot of steam with advances in computing power.

While an introduction to Bayesian statistics is outside the scope of this book, some of my favorite books on the subject are *Doing Bayesian Data Analysis by John Kruschke*and *Statistical Rethinking by Richard McElreath*.

13.1 Null Hypothesis Significance Testing

Next, let's either remind ourselves (if you've taken a statistics course before) what is commonly referred to as **Null Hypothesis Significance Testing** (NHST). NHST is a method of statistical inference in which you test the thing you think will happen in relation to a straw-man hypothesis usually corresponding to the notion "Nothing is going to happen." Let's break down how it works.

When we talk about statistical hypothesis testing, we typically have two competing hypotheses:

- The **Null Hypothesis** (H_0) typically suggests that there is no difference between two

quantities or groups on some meaningful parameter. For example, if I'm running a study to test the efficacy of a new vaccine on a new disease, my null hypothesis is that is no effect of the vaccine on the disease.

- The **Alternative Hypothesis** (H_A) is typically what we are trying to assert or explore. To use the above example, my alternative hypothesis would be that the new vaccine has a non-zero effect on the new disease.

If the sample data provide enough evidence against the null hypothesis, we say that we *reject the null hypothesis* in favor of the alternative hypothesis. If the sample data not provide enough evidence against the null hypothesis, we say that *fail to reject the null* hypothesis. Note that we can **NEVER** say that we *accept the null hypothesis*.

How do we judge if the data provide evidence for the hypothesis? The method of NHST involves the computation a **test statistic**, or number that quantifies how close the data match the distribution implied by the null hypothesis (this is commonly a normal or Gaussian distribution). The test statistic allows for the calculation of a **p-value**, or ***probability of observing a test statistic at least as extreme or more extreme as the one observed, assuming the null hypothesis true.***[23] [24] If the p-value is less than a particular threshold (typically 0.05, or a one-in-twenty chance, and denoted by the Greek letter alpha α), then we can say that the p-value is **statistically significant.** This is illustrated in Figure 13.2.

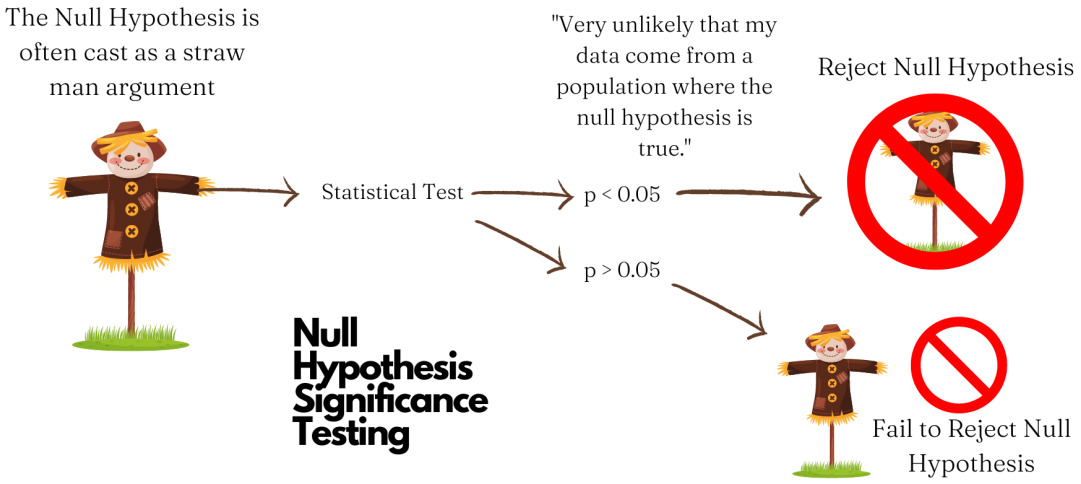


Figure 13.2: Null Hypothesis Significance Testing is akin to Proof by Contradiction.

! Why can't we ever accept the null hypothesis?

In short, we assume the null is true and are trying to find evidence against it, so our decision is always framed in terms of rejecting or failing to reject the null hypothesis. On a more fundamental level, the idea here is that *you cannot prove a negative*. If your sample data do not support the null hypothesis, you have only shown that and just that. You have not shown that null hypothesis is true. It might be true (and indeed we assume it is when doing the tests), but your study is not exploring its verity, rather your study is trying to find evidence that contradicts the null. It is a sort of proof by contradiction.

13.2 P-Value

It's important that we have a VERY clear definition of a p-value, as this is commonly misunderstood. Let's use the American Statistical Association's definition of the p-value.

! Definition of the p-value from the American Statistical Association

Informally, a p-value is the probability under a specified statistical model that a statistical summary of the data (e.g., the sample mean difference between two compared groups) would be equal to or more extreme than its observed value.[25]

Does that sound confusing to you? If so, you're not alone! Defining the p-value accurately doesn't make it very clear. I think a more intuitive definition of a p-value is an **index of surprise**. So, if the p-value is very low, that means your results are surprising assuming the null hypothesis is true, and worthy of a second-look. Something may be going on, but the p-value does not give you definitive proof of anything. In fact, that's how p-values were originally intended to be used, over a century ago (*Are my results worthy of a second look?),* and this idea has been lost over time.[26] Another idea that has been lost over time is the null hypothesis doesn't have to be a straw man hypothesis of zero effect or association between variables, but could actually be any test hypothesis.[27] It should probably not have been called *null*, as that's likely why people took it to mean zero or nothing.

As important as it is to know what the correct definition of a p-value is, it's also important to understand what is p-value is NOT.

1. **P-Values do not measure the probability that the hypothesis under study is true.** So, don't base your conclusions solely on whether an association is statistically significant or not.
2. **P-Values do not measure the probability that the data were produced by chance alone.** So, don't think that the p-value gives you the probability that the observed association was produced by chance alone.

3. **P-Values do not measure the size of an effect, nor the importance of a result.** So, don't base important decisions on whether a test of an association or effect passes an arbitrary threshold α.
4. **By themselves, p-values do not provide a good measure of evidence for a model or hypothesis.** So, don't believe an association or effect isn't there just because you didn't find it.

The p-value is also affected by the sample size and effect size. Generally, a larger sample size have a higher likelihood of detecting a statistically significant result if there is one. Additionally, a large effect is more likely to be detected than a smaller one.

Ok, so that's a lot of DON'Ts. You might be wondering, *Well what should I do then?* To answer this question, we can use the helpful acronym **ATOM: A**ccept uncertainty, be **T**houghtful, **O**pen, and **M**odest.[26] Basically, use the p-value but know its limitations.

13.3 Power

When you perform NHST, there are certain conditions of your hypothesis test that have a strong bearing on the p-values obtained, which can affect the interpretation of your results.

With any NHST, the following terms apply:

- Type I error (α) is the probability of incorrectly rejecting the null hypothesis when it is true. This is commonly known as a **false positive**. This is typically set at 5%.
- Type II error (β) is the probability of failing to reject the null hypothesis when it is false. This is commonly known as a **false negative**. This is typically set at 20%.
- Power (1 - β) is the pre-study probability of rejecting the null hypothesis, or observing an effect in the sample if there is one. In the social sciences, one usually tries to use a minimum sample size that achieves 80% power.

Statistical power calculations are performed before a study is performed, and typically to determine an appropriate sample size to detect an effect or association of interest. Besides the quantities specified above, one typically needs to also specify an effect size, or **minimum detectable effect** to be expected. This is usually specified as a 10-15% difference between groups.

Positive Predictive Value (PPV) is the probability that a positive research finding reflects a true effect (that finding is positive). Probability of research finding being true effect depends on 1) prior probability of it being true (before doing study), 2) statistical power of study, and 3) level of statistical significance. PPV is defined formally as:

$$PPV = \frac{(1-\beta) * R}{(1-\beta) * R + \alpha}$$

where: $1-\beta$ is statistical power (probability of detecting an effect if there is one), β is the probability of a Type II error (false negative), α is the probability of a Type I error (false positive), and R is the pre-study probability of observing a non-zero effect.

The lower the power and the higher the Type I error, the lower the PPV. If a study has lower power, it can only detect an effect (if there is one) when the size of that effect is large.

13.4 Confidence Intervals

Note that when we commonly deal with *estimates* when running statistical tests. We commonly estimate a **population parameter** (such as the mean), which is a descriptive measure for an entire population (such as Residents of Philadelphia). Parameters can be many things. Two of the most common population parameters we try to estimate are the mean of some variable, denoted by the Greek letter Mu μ, and the Standard Deviation (a measure how spread out the data are) of some variable denoted by the Greek letter Sigma σ.

Since we typically cannot collect data from all members of a population, we have to take samples from the population, and calculate **sample statistic** which involve the same kinds of parameters as for the population, but just for the sample. Thus, a sample statistic is a characteristic of the sample. If you draw a number from the population, the sample's characteristics will typically be similar to the population parameters. If you draw enough random samples, the sample statistics will converge on the population parameters, which is known as the **Law of Large Numbers**. Note that this only applies to the average of a variable converging to the Expected Value of the parameter in the population.

Ok, so why does all this matter? Well, the idea is that there is a degree of uncertainty associated with every estimate. The degree of uncertainty matters. If an estimate has a high degree of uncertainty, we might trust it less than an estimate with a low degree of uncertainty. How do we know how much uncertainty is associated with an estimate? Enter, the **Confidence Interval**, or a range of estimates for a parameter. The confidence level of the interval represents the proportion of confidence intervals in the long run that contain the true population parameter. For example, a confidence level of 95% (which is typical) suggests that if one were to construct many confidence intervals over and over again, 95% of them would contain the true population parameter.

14 Crises of Replication

14.1 Reproducibility and Replication Definitions

Let's remind ourselves what Reproducibility and Replicability mean. As you see in Figure 14.1, Reproducibility involves literally reproducing the results from a study using original data, code, and materials. Replicability is about carrying out the study using the same procedures in different settings and achieving consistent, not identical, results.

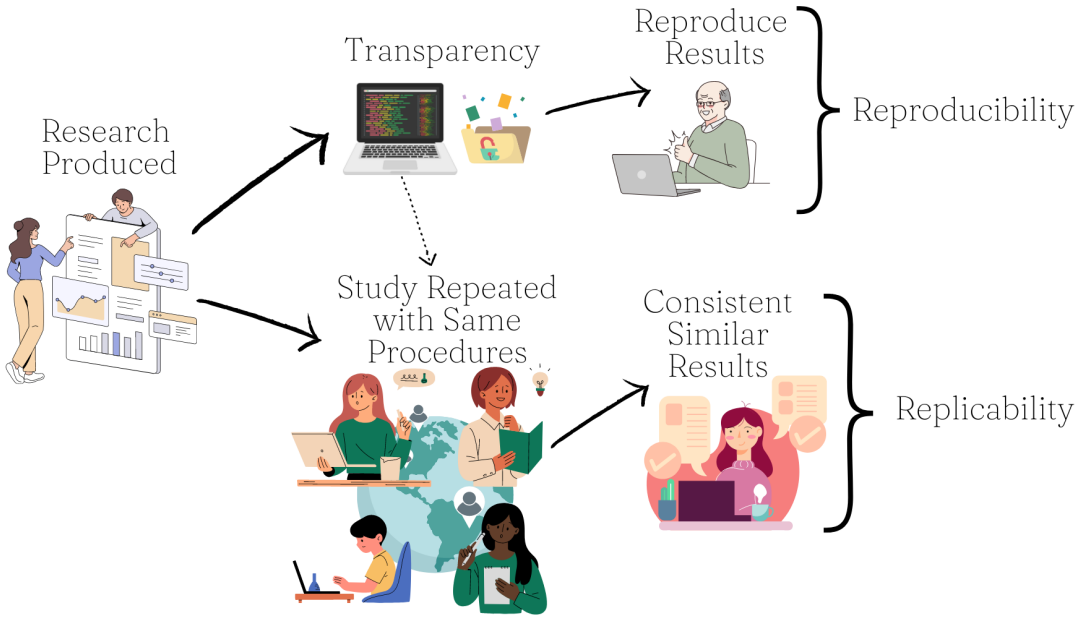


Figure 14.1: Reproducibility and Replicability.

14.2 Stirrings of a Crisis

For a long time, there was concern about the replicability and reproducibility of findings in the social sciences. In 2005, physician-scientist John Ioannidis published a provocative and

worrying essay entitled *Why Most Published Research Findings are False*, which outlined several methodological shortcomings plaguing most published research using statistics, including an over-reliance on p-values, low statistical power, and biases in study design, data collection, and analysis.

Open access, freely available online

Essay

Why Most Published Research Findings Are False

John P. A. Ioannidis

Summary

There is increasing concern that most current published research findings are false. The probability that a research claim is true may depend on study power and bias, the number of other studies on the same question, and, importantly, the ratio of true to no relationships among the relationships probed in each scientific field. In this framework, a research finding is less likely to be true when the studies conducted in a field are smaller; when effect sizes are smaller; when there is a greater number and lesser preselection of tested relationships; where there is greater flexibility in designs, definitions, outcomes, and analytical modes; when there is greater financial and other interest and prejudice; and when more teams are involved in a scientific field in chase of statistical significance. Simulations show that for most study designs and settings, it is more likely for a research claim to be false than true. Moreover, for many current scientific fields, claimed research findings may often be simply accurate measures of the prevailing bias. In this essay, I discuss the implications of these problems for the conduct and interpretation of research.

factors that influence this problem and some corollaries thereof.

Modeling the Framework for False Positive Findings

Several methodologists have pointed out [9–11] that the high rate of nonreplication (lack of confirmation) of research discoveries is a consequence of the convenient, yet ill-founded strategy of claiming conclusive research findings solely on the basis of a single study assessed by formal statistical significance, typically for a *p*-value less than 0.05. Research is not most appropriately represented and summarized by *p*-values, but, unfortunately, there is a widespread notion that medical research articles

It can be proven that most claimed research findings are false.

should be interpreted based only on *p*-values. Research findings are defined here as any relationship reaching formal statistical significance, e.g., effective interventions, informative predictors, risk factors, or associations. "Negative" research is also very useful. "Negative" is actually a misnomer, and is characteristic of the field and can vary a lot depending on whether the field targets highly likely relationships or searches for only one or a few true relationships among thousands and millions of hypotheses that may be postulated. Let us also consider, for computational simplicity, circumscribed fields where either there is only one true relationship (among many that can be hypothesized) or the power is similar to find any of the several existing true relationships. The pre-study probability of a relationship being true is $R/(R + 1)$. The probability of a study finding a true relationship reflects the power $1 - \beta$ (one minus the Type II error rate). The probability of claiming a relationship when none truly exists reflects the Type I error rate, α. Assuming that c relationships are being probed in the field, the expected values of the 2×2 table are given in Table 1. After a research finding has been claimed based on achieving formal statistical significance, the post-study probability that it is true is the positive predictive value, PPV. The PPV is also the complementary probability of what Wacholder et al. have called the false positive report probability [10]. According to the 2×2 table, one gets PPV = $(1 - \beta)R/(R$

Figure 14.2: Ioannidis laid out several factors why most published research is false.[28]

He noted that several biases in design, data analysis, and presentation factors influence the production of research findings. With greater bias in the conduct of a study, the lower the chances of the research finding being true. Additionally, the likelihood of a finding being true depends a great deal on the pre-study odds of it being true, as well as the statistical power of a study. He also points out that when a study is replicated by others in different contexts, the effect size is also likely to be smaller, and the likelihood of the finding being true is diminished.

This obviously made a lot of folks nervous, and become a highly downloaded paper. However, the concerns about replication did not truly become mainstream until one particular psychologist from Cornell published a study suggesting that humans have psychic ability…

14.3 Feeling the Future

14.3.1 An Unintentional Crisis Unfolds

In 2011, a study authored by Cornell psychologist Daryl Bem (a very respected psychologist) was published in the *Journal for Personality and Social Psychology* (a very respected psychology journal). The study appeared to shake the foundations of what we know about human beings. It suggested, through a series of nine experiments comprising more than 1,000 participants over ten years that humans have psychic ability!

Here's what happened. Bem administered College students came to a computer lab, and had to look at a computer screen. He administered many experiments to see if people have an innate psychic ability to predict the future. In one experiment, participants were presented with two curtains on a screen (represented in Figure Figure 14.4 below).

They had to then guess which curtain was concealing an image. Unbeknownst to them, the image would be randomly allocated to a curtain *after* they had made their choice. So, if they were able to guess which curtain held the image better than a 50-50 chance (for example, say they correctly guessed 60% of the time), this would be taken as evidence of an extra-sensory perception that allows one to *Feel the Future* (the admittedly catchy title of the study). Notably, some of the images were 'erotic' in nature, while others were neutral. In eight of nine studies reported in the paper, participants were significantly likely to predict which curtain contained the image. The findings in totality provided Bem ample evidence for the anomalous phenomenon of precognition, or psychic ability to see the future.

The study became **HUGE**. In general, if more than ten people read your research, I would call that a win. But Bem's study was a darling of the media. The *New York Times* ran a front-page story on it, and Bem even appeared on the popular comedy program *The Colbert Report*. Those who were already convinced of the highly questionable field of *Psi* research (research on paranormal phenomena) were thrilled at Bem's study. It seemed to provide strong evidence for psychic ability using widely accepted methods in a respected journal by a respected Ivy-league professor. Bem had large enough sample sizes to be accepted by the journal (he had been the Editor previously so he knew his methods were up to snuff), used basic one sample t-tests that one learns in introductory statistics classes, and made sure his stimuli had been randomized correctly. Many began to question whether humans had psychic ability based on Bem's findings, and I don't blame them at all. In the words of Slate author Daniel Engber, Bem's findings were both *methodologically sound and logically insane.*[29]

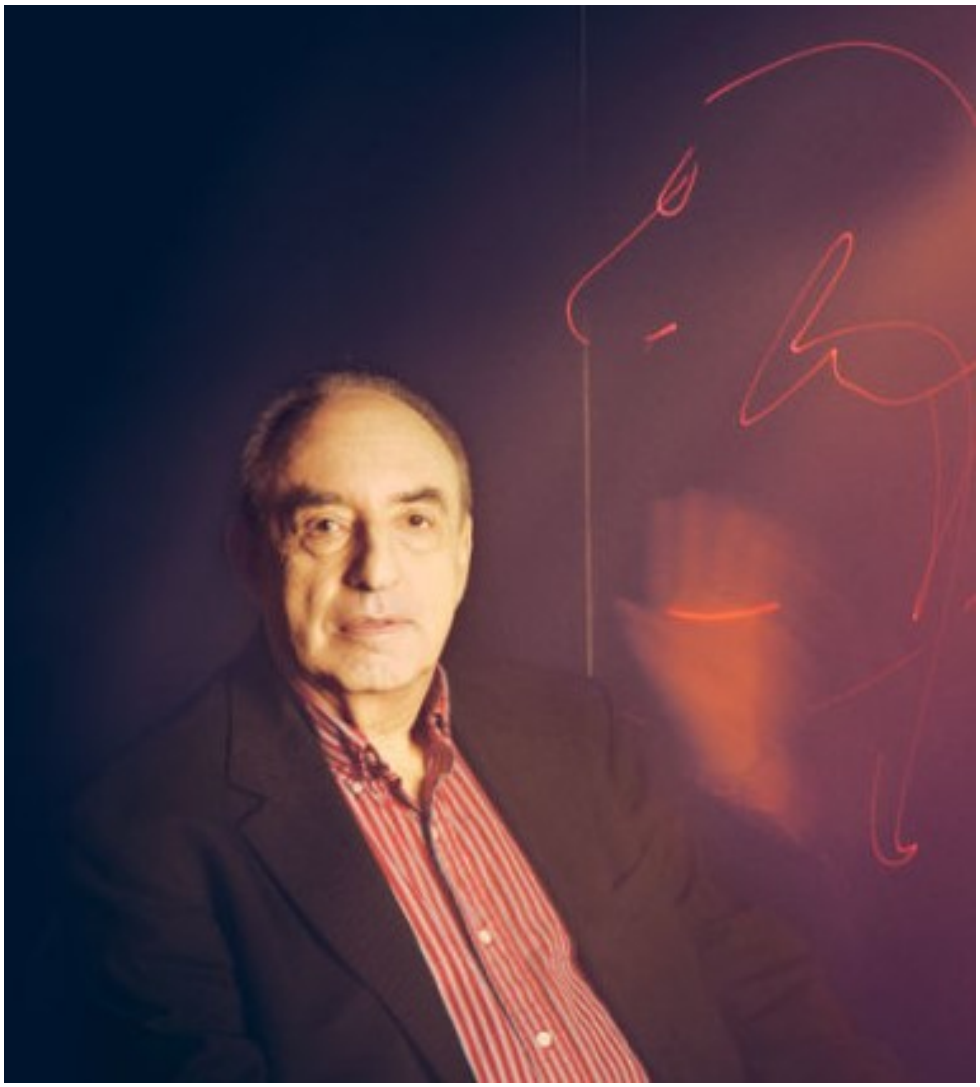


Figure 14.3: Former Cornell professor of psychology Daryl Bem was very interested in studying the paranormal. Perhaps he wanted to find evidence for the paranormal a little too much.

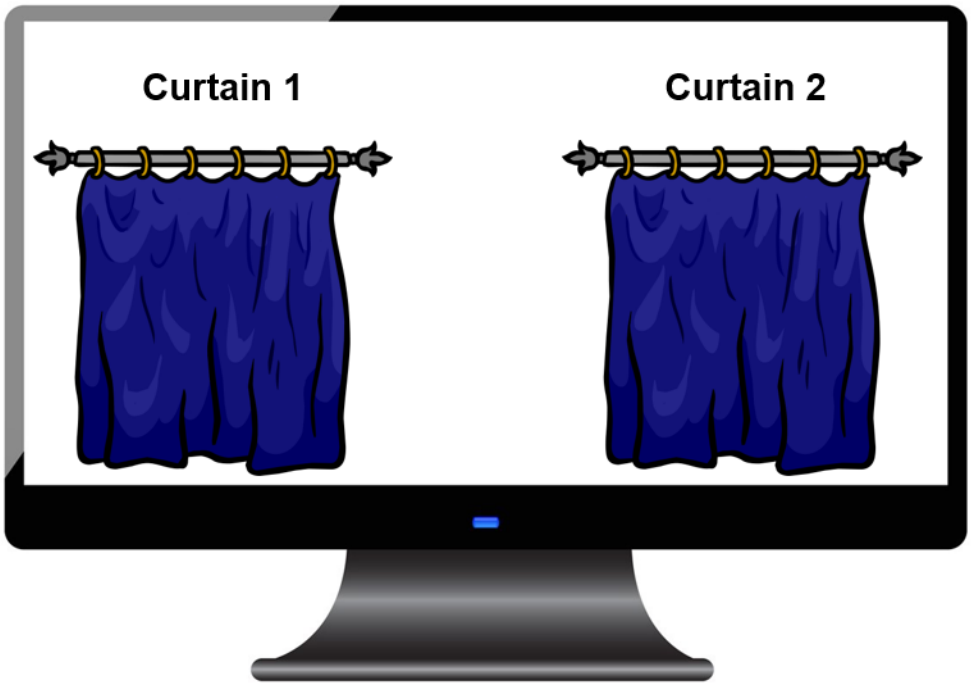


Figure 14.4: Participants had to guess which one curtain concealed an image. After they made their guess, the image was randomly assigned to one curtain or another.

Figure 14.5: Bem's ESP study riled up the field of social psychology…and beyond!

So, do humans actually have psychic ability? No. At least, we don't actually have any convincing evidence for that notion. Others tried to replicate Bem's findings and did not find the same effects. In fact, one large replication with 3,289 participants across seven experiments found an effect size of $d = 0.04$, which is statistically indistinguishable from zero. In another very thorough replication study involving 2,115 participants and ten labs from nine different countries, the researchers were unable to replicate Bem's parapsychological effects, and found support for the notion that Bem's statistically significant findings were *pure bias* in that significant findings were due to researcher and publications biases, as well as a lack of methodological rigor.[30]

> **i** Bem fires back.
>
> I should note that Bem tried to dispute the findings of the skeptics with a meta-analysis of 90 studies from 33 labs across 14 countries suggesting that this findings are in fact real.[31] However, given the myriad problems in the conduct and reporting of studies observed in the original study by Bem, I think it's fair to say most academics in the field didn't really buy these updated findings. Indeed, a breakdown of the statistics in the meta-analysis raises more questions than it answers.[32] Given the replication work by non-Bem affiliated labs, and the fact that they are more recent, it's safe to conclude that the original findings are not supported (however much some might want them to be).

14.3.2 Where Bem Went Wrong

Ok, so you may be wondering, "Well what did Bem do wrong?" He used widely accepted statistical methods, used large sample sizes, performed multiple experiments across years, and went through peer-review in a top-tier journal. It would seem he did everything right! The lesson is instructive, because you can do all of that, and still produce junk science. Bem fell prey to what we now call **Researcher Degrees of Freedom** - the flexibility inherent in the many decisions researchers have to make in data collection, analysis, and reporting that can result in selective practices producing spurious results.[33] Engaging in researcher degrees of freedom increases the chance of false positives, or Type I errors, as well as biases in selectively reporting only findings that confirm one's pre-existing conceptions. Here are just two researcher degrees of freedom that very likely led Bem to his spurious findings:

- **Bem made many tweaks to his experiments over the years in order to yield statistically significant results.** He did not document many of them. If you tweak your experiments without reporting that you did so, you make it seem like you just got the result magically, and not as a result of careful manipulation. This manipulation can results in false positives, but it also weakens your conceptual theory. For example, if a study did not show evidence of psychic ability, and Bem modified the experiment, and then DID find evidence of psychic ability, what does this say about his theory of

psychic ability. In this way, notice how you can get very flexible with your theory if you're making undocumented tweaks.

- **There were no pre-registered replications of experiments in Bem's original study.** While we'll get to pre-registration a bit more in detail later, the idea is simple: 1) Before you do your study, say what your going to do, why, and how; 2) publish this *pre-registration* in the public domain; 3) do your study and report back on the things you had planned to do. Nowadays, this is a common rigorous practice, but was not really a thing in Bem's time. This means that after doing one experiment, Bem was free to plan, adapt, abandon, and modify his hypotheses and experimental protocols in order to yield statistically significant results. Thus, if he ran a study, didn't find what he wanted and then abandoned the study, this would not be recorded. Selective reporting means that you could run 100 studies, and only report and publish 10 of those that are statistically significant, severely distorting the evidence base and hiding your 90% null results rate.

Bem's own words in an interview with Slate magazine in 2017 are quite instructive:

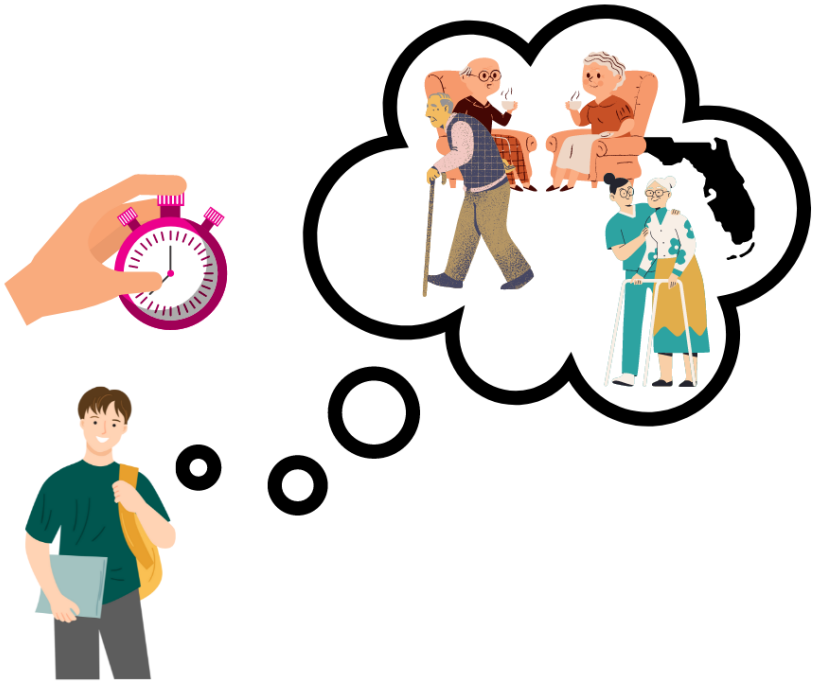
> **i** Bem in his own words.
>
> "I would start one [experiment], and if it just wasn't going anywhere, I would abandon it and restart it with changes," Bem told me recently. Some of these changes were reported in the article; others weren't. "I didn't keep very close track of which ones I had discarded and which ones I hadn't," he said. Given that the studies spanned a decade, Bem can't remember all the details of the early work. "I was probably very sloppy at the beginning," he said. "I think probably some of the criticism could well be valid. I was never dishonest, but on the other hand, the critics were correct."[29]

14.4 The Domino Effect

After the publication of Bem's ESP paper, many began questioning the foundations of the entire field of social psychology. Researchers began **trying and failing** to replicate many classic findings in psychology. The subfield of priming within social psychology was one of the first fields to undergo extensive critical scrutiny, with some dire results. Many priming studies failed to replicate, and it seemed that faulty methods, lack of statistical power, researcher biases, and publication biases were responsible.

> **i** A Prime Candidate for Replication: The Elderly Priming Study.
>
> For instance, a classic study in the field of subliminal priming led by Yale psychologist John Bargh involved one experiment in which undergraduates in a lab worked on a scrambled-sentence task. That is, they were given 30 sets of five word combinations

which they had to use to construct a sentence. In the treatment group, the scrambled words contained words previously identified to be related to stereotypes of the elderly: *Florida, old, lonely, grey, sentimental, wise,* etc. (these are from the actual study). The hidden outcome of this study was the time it took for participants to walk down a corridor to leave the study area. The results showed that those exposed to the elderly prime condition walked almost one second slower than those who were exposed to neutral words (a statistically significant result).[34]



Sounds pretty cool right? That's certainly what the authors of a psychology textbook I read in college thought when they presented the results of this study as fact. Unfortunately for Bargh and his colleagues, the unintentional crisis of replication set in motion by Bem led a team to try and replicate the classic elderly prime findings. The team followed the same protocol as Bargh and his colleagues, and found no difference in walking times between the elderly and neutral prime conditions.[35]

Notably, the replication team used infrared sensors to automate the timing process, so it didn't depend on a human's use of a stopwatch. They speculated that the people manually timing the participants may have been a source of bias in the original study. They ran another experiment in which they used people to manually time participants with a stopwatch (I'll call them the *timers*). They told the timers in one group that

participants in the treatment group would walk faster, and they told the other half the participants in the treatment group would walk slower. Unbeknownst to the timers, infrared sensors were also measuring participants' objective walking speeds. The results showed that when experimenters were led to believe participants would walk slower as a result of the intervention, the walking times were significantly higher in the Prime condition compared to the Neutral condition. Interestingly, when experimenters were led to believe that participants would walk *faster* as a result of the intervention, the walking times were significantly lower in the Prime condition compared to the Neutral condition. The results suggest that priming effects may reflect experimenter bias, rather than an actual induced effect of a prime stimulus on participants.
I should note that some of Bargh's other priming studies have also failed to replicate.[36] For his part, Bargh dismissed the replication of his elderly prime study, and responded to the replicators with many objections to their replications as well as a scathing personal attack on them.[37]

It seems that 2011 was a watershed moment in the history of reproducibility and replication. A few key moments are worth mentioning. First, just about two months after *The New York Times* reported on Bem's ESP study, a group of researchers submitted a study for publication entitled *False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant.*[33]

This paper demonstrated with real experimental data how easy it was to achieve statistical significance for an impossible hypothesis. In their study, the authors experimentally demonstrate that listening to the song *When I'm Sixty-Four* by the Beatles, compared to the song *Kalimba* which came free with Windows 7, actually MADE participants younger. Think about that for a second. Obviously, listening to any song might make you feel younger or older, but the outcome I mean here is *actual* chronological age measured by one's birth date. Even though the premise that listening to the Beatles will make you younger is obviously false, the authors show, with standard statistical methods, that it is possible to achieve a statistically significant result supporting the impossible claim at $p < .05$. There was no magic associated with this finding, it was achieved deliberately by engaging in researcher degrees of freedom such as using multiple dependent variables but reporting only the covariates that resulted in statistical significance, analyzing data before all data were collected, and not using a rule for when to stop collecting data.

As the field of priming research began to come into question, prominent psychologist Daniel Kahneman wrote an open letter to Bargh and others in the field of priming studies asking them to clean up their act. The entire letter can be found in Appendix G. The letter is prescient and instructive. Kahenman correctly predicting a *train wreck looming* which became the *Replication Crisis*, and his proposal to have a daisy chain setup of different labs replicating the same effect foreshadowed many large replication efforts to come.

Kahneman was not and is not a priming researcher, but he had some skin in the game vis-à-vis a popular book he published in 2011 entitled *Thinking Fast and Slow* which went on to become

Figure 14.6: Listening to the Beatles can make you younger! Not really, but with so much analytic flexibility, you can make an impossible hypothesis appear statistically significant.

a *New York Times* bestseller. The book was a popular psychology piece in which Kahneman reviewed some of his major findings, as well as those of others, some of whom included priming researchers. As the replication crisis unfolded, it became apparent that many of the chapters referenced studies which later failed to replicate, his fourth chapter representing the chapter most riddled with references to spurious studies.[38]

To his credit, Kahneman provided a thoughtful response to a blog scrutinizing the shaky studies he cited in his book. Kahneman replied as a comment to the blog admitting that he placed too much faith in underpowered studies, despite having published a paper previously about how researchers are often reliant on underpowered studies.

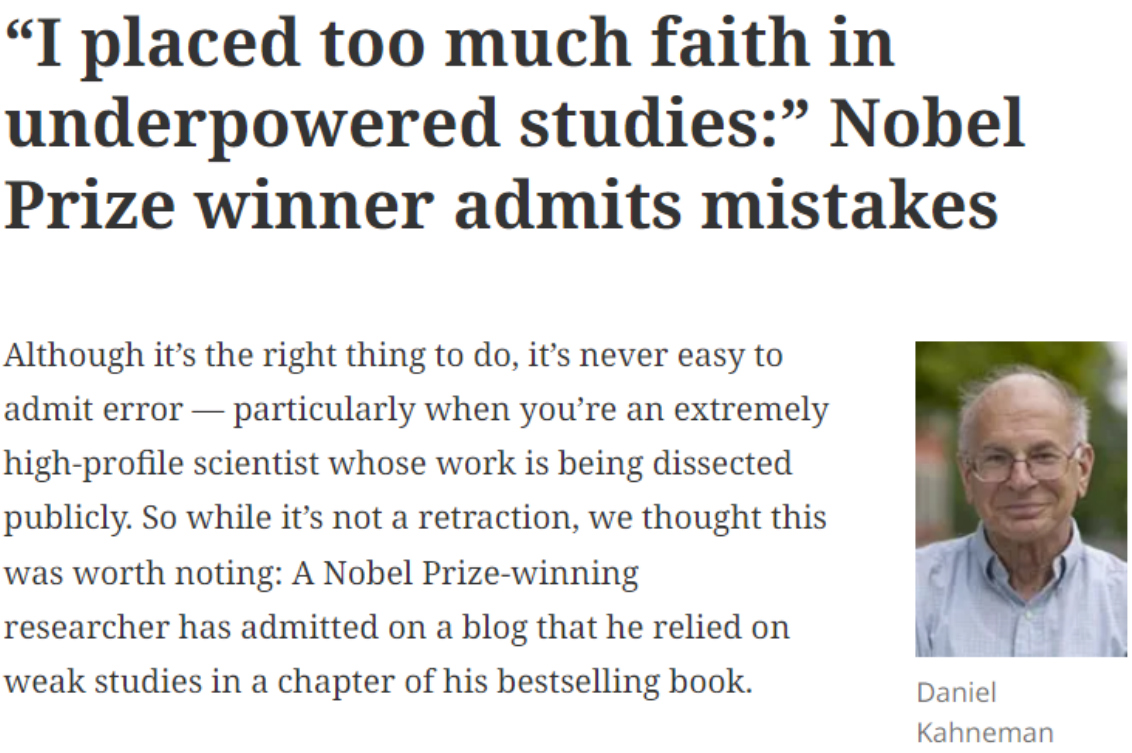


Figure 14.7: Kahneman lauded a number of priming studies in his bestselling book that turned out to not replicate. To his credit, he admitted this publicly. *Retraction Watch.*

The case of Kahneman, a prominent researcher who knew well enough the dangers of relying on small sample sizes, falling prey to the very thing he had critiqued is illustrative of the importance of *evidence-based* rather than *eminence-based* practice. It also showed the extent of the problem. Even Nobel Prize winners could make mistakes in promoting weakly supported or spurious findings.

The replication crisis was not confined to priming studies, nor to the field of psychology alone. The crisis became evident in fields such as economics, cancer biology, finance, artificial

intelligence, nutrition, and more. Indeed, it continues to this day, as more and more fields are critically scrutinizing key findings.

14.5 Gauging the Extent of the Damage

The replication crisis has resulted in major efforts to replicate key findings across different fields. This process continues to unfold, but we can examine some salient examples.

14.5.1 The Open Science Collaboration

A huge replication study was convened by a group called the Open Science Collaboration involving 270 scientists from 17 countries, who selected 100 studies published in 2008 from top-tier psychology journals.[39] Of the 100 studies, 97 had a finding that was statistically significant at the 5% level ($p < .05$), and of the replications 35 of 97 studies had a finding at $p < .05$. The replication study effects were also about half as big as the original study effects. The distribution of original study effect sizes and replicated study effect sizes in presented in Figure 14.8.

This was a massive effort that took four years to painstakingly gather data and replicate these experiments. The fact that about two-thirds of findings published in top-tier psychology journals did not replicate was concerning to many who took these findings as well-established. What did it mean for other findings? What did it mean for the entire field of psychology? Could other fields be affected?

14.5.2 Questionable Research Practices in Psychology

To better understand the extent of Questionable Research Practices (QRPs; researcher degrees of freedom) in psychology, a group of researchers surveyed over 2,000 academic psychologists at major US universities.[40] Respondents were asked about a) whether they had engaged in a number of QRPs (self-admission rate), b) the percentage of other psychologists they believed had engaged in the QRP (prevalence estimate), and c) the percentage of psychologists committing QRP who would admit to doing so. The main results are presented in Figure 14.9.

As we see, even with a select sample of psychologists, many admitted to engaging in QRPs. We can imagine that the self-admission rate represents a potential underestimate of the true extent of the problem.

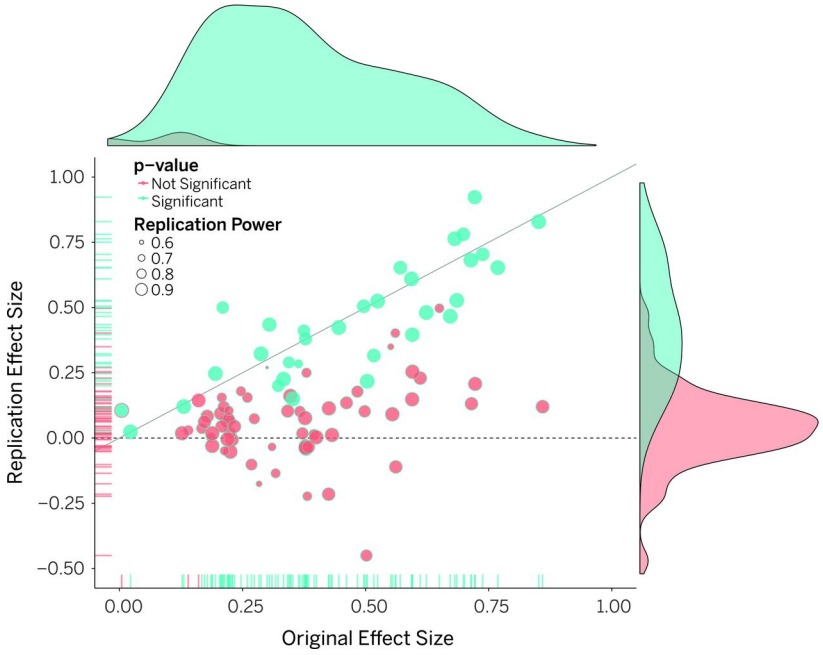


Figure 14.8: The diagonal line represents the case where the original effect size = the replicated effect size. Red dots represent non-significant results ($p > .05$), and green dots represent significant results ($p < .05$). The dotted line represents zero effect, and points below represent replicated effects in the opposite direction of the original study.[39]

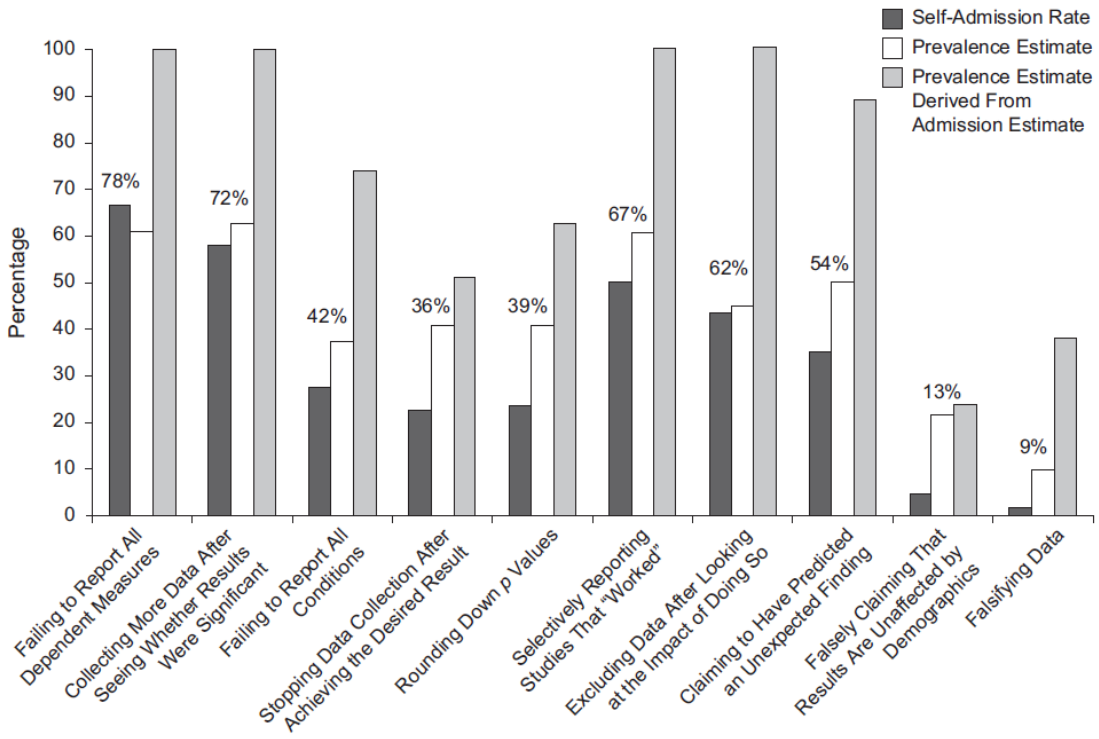


Figure 14.9: For each QRP, the authors present the self-admission rate, the perceived prevalence of the QRP among other psychologists, and a prevalence estimate derived by the researchers taking the self-admission rate and dividing it by the estimate of how likely other psychologists would admit to engaging in QRPs.[40]

14.5.3 Many Labs 2

Imagine you replicate a study and find that the effects are different from the original. Immediately, the original authors may say that your protocol deviated from the original study, explaining the discrepancy in effect. To address this issue, a team carried out replications of 28 findings in psychology using protocols that were peer reviewed in advance.[41] Each protocol was administered on about 15,305 participants from 36 countries and territories. So, this was a series of high-powered replications. Remember that statistical power relates to having the ability to detect an effect if it's actually there.

The team found that just 15 of 28 findings replicated in the same direction as the original at $p < .05$. Of these, 75% of the replicated findings were smaller than the original study effect. The authors concluded that the variation in effect sizes had more to do with the effect being studied than the sample or setting of the study.

14.5.4 Replicating Social Science Experiments from *Nature* and *Science*

Moving from the field of psychology to social science broadly, a team replicated 21 experimental studies published in the top journals *Nature* and *Science* between 2010 and 2015.[42] By being published in these journals, one would assume that these experiments are rigorous and well conducted, as these journals are supposed to be at the forefront of science. The main results are presented in Figure 14.10.

Despite being published in the BEST journals, eight of 21 studies did not replicate in the original direction, and the studies that did replicate were, on average, half the size of the original effect. Remember, getting published in these journals is very hard, and typically we expect robust effects to be found in their pages. The fact that a full eight of the 21 studies did not replicate, and that the replicated effects are much smaller than the original tells us that journal quality is not a failsafe for robust replicated findings. The authors conclude that both false positives, and inflated effect sizes of true positives contribute to *imperfect reproducibility*.

14.6 Retractions

14.6.1 What is it, and Why Does it Happen?

A journal will **retract** a paper when it removes a paper from its records that it already published. Retractions do not occur over small issues like typos. It's usually something quite seriously wrong about the paper or the peer review process. Usually, retractions occur due to major errors in the research, plagiarism, data falsification, or something else quite serious. Authors can self-retract if they later discover a serious error in their work, and that's a very

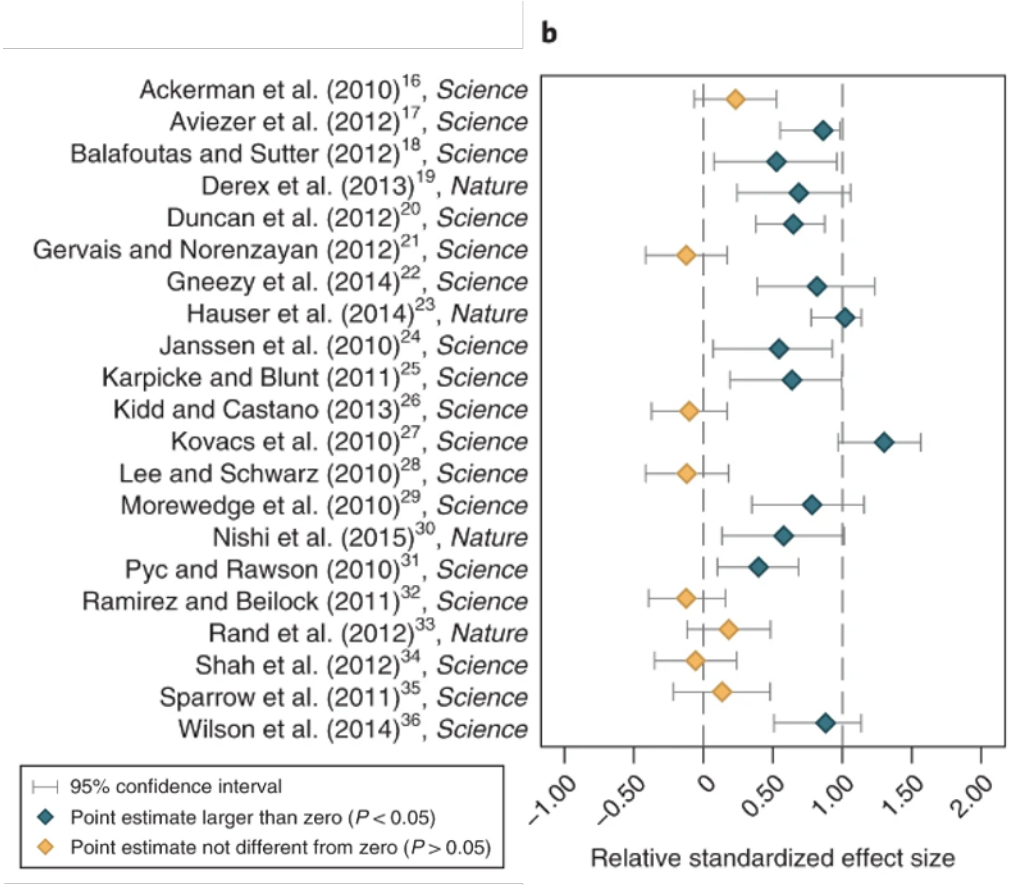


Figure 14.10: Standardized effect sizes presented such that 1 equals the original effect size. Yellow diamonds represent a zero effect, and green diamonds represent a statistically significant effect. Thirteen of 21 effects replicated in the same direction as the original study.[42]

laudable thing. However, quite often retraction decisions are made by the editor or editorial board of the journal.

Lack of reproducibility is usually not grounds for retraction on its own. Consider the Bem ESP study. Since its publication, we now have a pretty decent understanding of Bem's spurious results, including testimony from Bem himself in committing what we now call researcher degrees of freedom. However, when the editor of the *Journal of Personality and Social Psychology* was asked to retract the study in 2018, it took him two years to respond to the letter and inform the requester that the paper would not be retracted. Thus, it's not a done deal even when you have a false hypothesis, evidence of bad research practice, and a lack of reproducibility.

That being said, as editors have become more aware of the many issues with research misconduct, and with the growth of plagiarism detection software, the number of retractions across all of science have risen many fold. In the year 2000, there were about 100 retractions annually across all of science; in 2014 that number grew to 1,000; and in 2022 it was up to about 3,600.[43] [44] The largest and most comprehensive database of retractions (and one of my favorite websites) is Retraction Watch. If you're reading this, stop reading right now and click on the link and check out Retraction Watch (consider also making a tax-deductible donation if you like the vibe).

So, why are so many papers being retracted? Though the graphic in Figure 14.11 is a little older, it is still instructive.

Plagiarism from other papers or even from one's own previously published papers is one of the main culprits of retraction. Importantly, fake peer review has also been a driving factor behind retractions. Fake peer review happens when an author gives a journal an email address ostensibly as the contact for a potential reviewer, but in reality they control the email address. This means that they can control their own peer review, which defeats the purpose of the whole exercise. Figure 14.12 is what it can look like like when a paper is retracted because the editor discovered that the peer review process had been manipulated.

14.6.2 Life After Death: Continued Citations Despite Retraction

Bad papers are being retracted at an increasing pace. That's good, right? Yes. But, there is another problem. Many papers continue to be cited even years after they've been retracted, and most of the citations don't mention that the paper has been retracted. Let's look at the top five of the most highly cited retracted papers from Retraction Watch's database, shown in Table 14.1.[45]

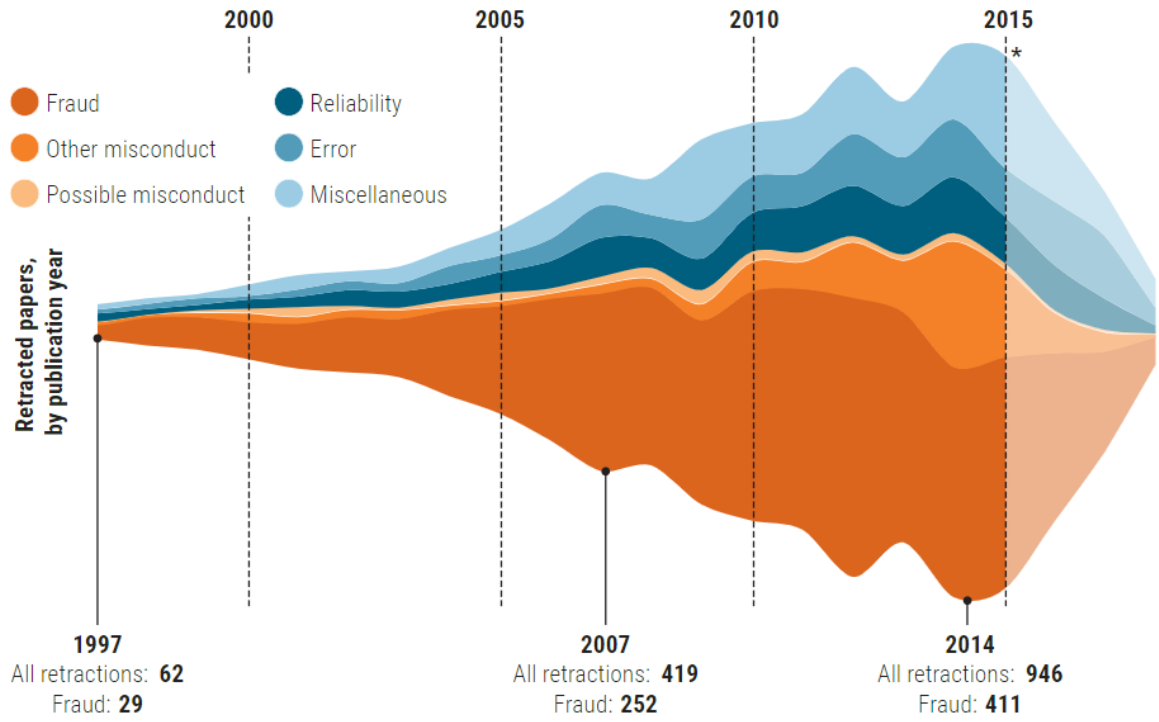


Figure 14.11: The majority of retractions between 1997 and 2015 have been due to fraud. These days, image fabrication is a particular problem in the STEM disciplines. *Science*.[44]

Afsar, B, Al-Ghazali, B, Umrani, W. Corporate social responsibility, work meaningfulness, and employee engagement: The joint moderating effects of incremental moral belief and moral identity centrality. Corp Soc Resp Env Ma. 2020; 27: 1264–1278. https://doi.org/10.1002/csr.1882

The above from Corporate Social Responsibility and Environmental Management published online on 06 February 2020 in Wiley Online Library (wileyonlinelibrary.com), has been retracted by agreement between the journal Editor-in-Chief, Richard Welford, and John Wiley & Sons Ltd. The retraction has been agreed due to intentionally manipulated peer-review by people undertaking work on behalf of the authors.

Figure 14.12: See the last sentence? This paper was retracted due to fake peer review.

Table 14.1: I've put links to the original articles instead of citing them, for obvious reasons.

Article title, journal, and year	Year of retraction	Citing articles before retraction	Citing articles after retraction	Total citations
1. Primary Prevention of Cardiovascular Disease with a Mediterranean Diet. *New England Journal of Medicine*; 2013.	2018	1,905	950	2,855
2. Ileal-lymphoid-nodular hyperplasia, non-specific colitis, and pervasive developmental disorder in children. *Lancet*; 1998.	2010	643	940	1,583
3. Visfatin: A protein secreted by visceral fat that mimics the effects of insulin. *Science*; 2005.	2007	232	1,232	1,464
4. An enhanced transient expression system in plants based on suppression of gene silencing by the p19 protein of tomato bushy stunt virus. *The Plant Journal*; 2003.	2015	895	421	1,316
5. Lysyl oxidase is essential for hypoxia-induced metastasis. *Nature*; 2006.	2020	977	105	1,082

Do you see how entries 2 and 3 have more citations after being retracted, than they did before retraction? That's a problem! Additionally, entry 2 is fraudulent study claiming that vaccines cause autism. While this study has been thoroughly debunked, it has fueled the anti-vaccination movement, and its effects continue to be seen today. In this way, research misconduct does not just affect science, but can have massive downstream effects upon society, and the health and well-being of many.

Retraction does appear to decrease citation frequency, but not by as much as we might like. One study examined compared the citation counts for 3,000 retracted papers to 3,000 non-retracted papers.[46] They found that retraction decreased citation frequency by about 60%, and that many retracted papers continued to be cited.

Perpetuation of Fraud: The Case of Scott Reuben

On February 21, 2010 American anesthesiologist Scott Reuben formally pled guilty to one count of health care fraud. He was sentenced to six months in prison, followed by three years of supervised release. He also had to pay a $5,000 fine, forfeit $50,000 to the government, and make restitution to pharmaceutical companies he had defrauded to the

tune of $360,000.
Reuben was formerly a professor of anesthesiology and pain medicine at Tufts University. He admitted to having faking data underlying his research, and lied about conducting 21 clinical trials. The fake results from these trials were published in many journals. When he was outed as a fraud, his publications had been cited almost 1,200 times, and his work was quoted in clinical guidelines. An analysis of his published work in 2014 revealed that 45% of his retracted articles had been cited at least once, and of these, only a quarter correctly mentioned the work as being retracted. Thus, even five years after his articles were retracted, they were still being quoted and cited.[47]
Reuben was also able to slip past peer review and maintain fraudulent practices for 13 years, and wasted millions of dollars of funding. Consider also the impact his bogus findings had on the field of anesthesiology. At the time of the scandal, the editor-in-chief of the journal *Anesthesia and Analgesia* said of Reuben's articles:

> We are left with a large hole in our understanding of this field. There are substantial tendrils from this body of work that reach throughout the discipline of postoperative pain management. Those tendrils mean that almost every aspect will need to be carefully thought through. What do we still believe to be true? Do the conclusions hold up to scrutiny?[48]

You may be wondering, why do people cite retracted studies? It's usually not to call them out as retracted. The reality is that most folks don't care to check or don't know that a study is retracted. Retraction notices on journals also vary widely, with some being more salient than others. Let's have a look at some retraction notices.

First up, we have the prestigious *New England Journal of Medicine* in Figure 14.13. This isn't a great retraction notice because it's just a narrow banner. These days, someone might mistake it for a pop-up about cookies, which has become ubiquitous. If someone downloads this in a hurry, I would worry that they might not see the notice.

Next, we've got *Science* in Figure 14.14. This one is bit more prominent, but could still be missed by someone in a hurry. We may also wonder, why not splash it against the title of the study?

Finally, my favorite one so far is the retraction notice of the *Lancet*, shown in Figure 14.15. This is nice! It's a huge notice splashed against the entire page. You cannot miss it. To me, this is the gold standard of retraction notices.

Despite journal websites having retraction notices, some may still not be aware of the retractions because they might have downloaded the paper earlier pre-retraction. Thus, they might not need to ever visit the paper's journal page, and may miss the retraction notice.

This is a big problem related to continued citation of retracted papers. One excellent solution to the problem exists IF you use Zotero for reference management. Zotero is a free reference management software, and now partners with Retraction Watch. This means that if a paper

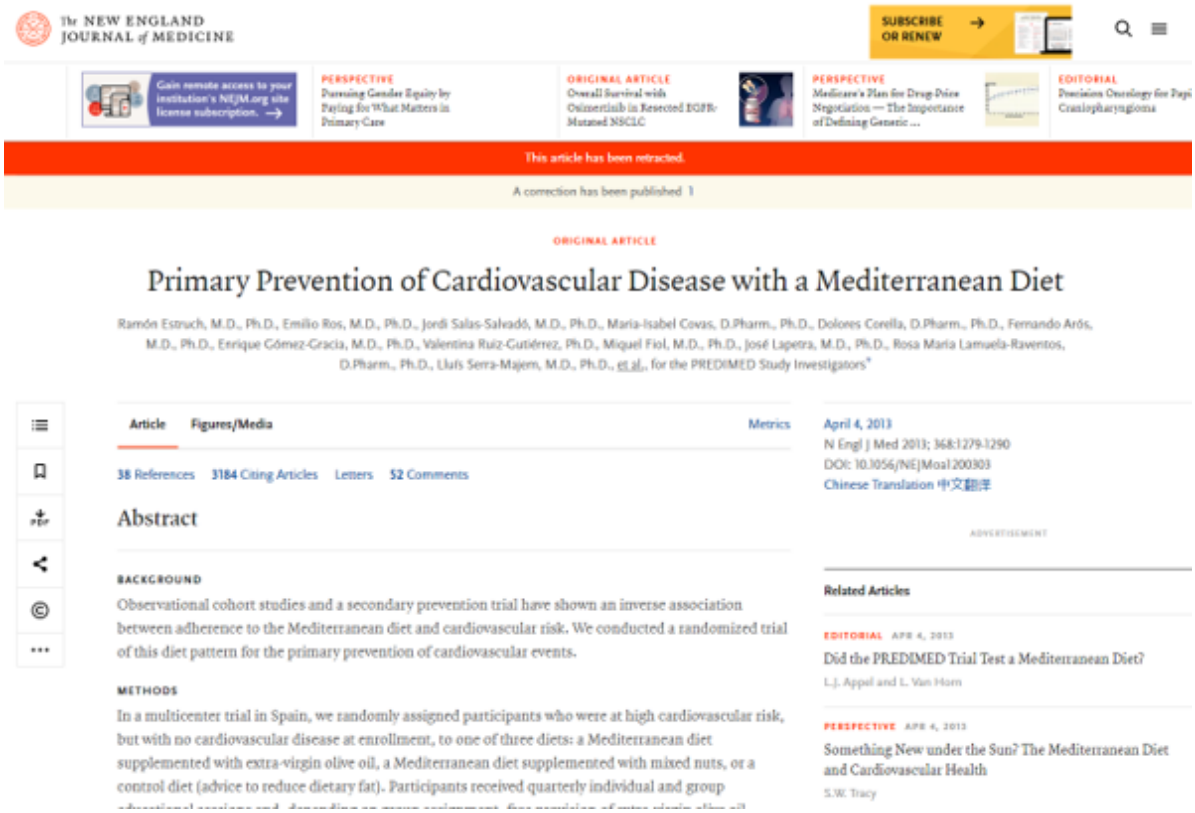


Figure 14.13: The retraction notice isn't very salient. If someone is downloading this in a hurry, will they notice?

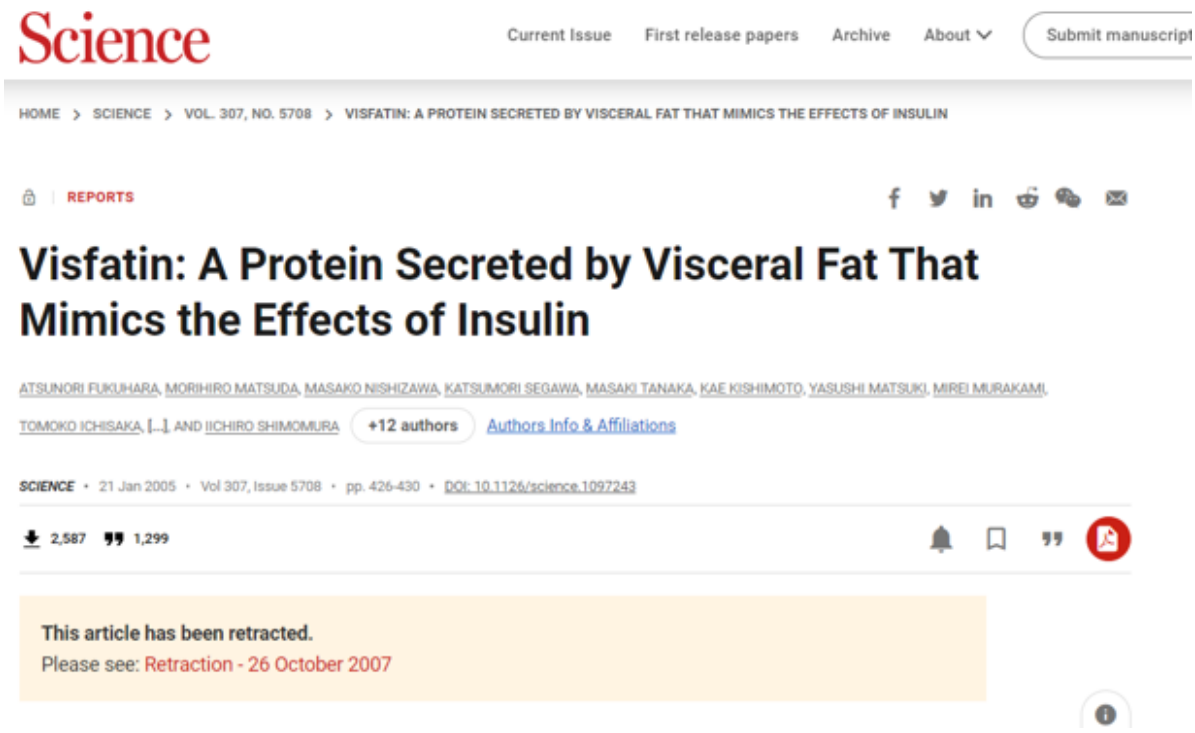


Figure 14.14: Better than NEJM's retraction notice maybe? Still, could be better and more prominent.

EARLY REPORT

Early report

Ileal-lymphoid-nodular hyperplasia, non-specific colitis, and pervasive developmental disorder in children

A J Wakefield, S H Murch, A Anthony, J Linnell, D M Casson, M Malik, M Berelowitz, A P Dhillon, M A Thomson, P Harvey, A Valentine, S E Davies, J A Walker-Smith

Summary

Background We investigated a consecutive series of children with chronic enterocolitis and regressive developmental disorder.

Methods 12 children (mean age 6 years [range 3–10], 11 boys) were referred to a paediatric gastroenterology unit with a history of normal development followed by loss of acquired skills, including language, together with diarrhoea and abdominal pain. Children underwent gastroenterological, neurological, and developmental assessment and review of developmental records. Ileocolonoscopy and biopsy sampling, magnetic-resonance imaging (MRI), electroencephalography (EEG), and lumbar puncture were done under sedation. Barium follow-through radiography was done where possible. Biochemical, haematological, and immunological profiles were examined.

Findings Onset of behavioural symptoms was associ... by the parents, with measles, mumps, and rub... vaccination in eight of the 12 children, with measl... infection in one child, and otitis media in s... All 1... children had intestinal abnormalities ranging from lymphoid nodular hyperplasia to g... oid u...ation. Histology showed patchy chronic infla...tion ... in 11 children and reactive ileal lymphoid hyperplasia in seven, but no granulomas. B... oural disor...s included autism (nine), disintegrati... ...y (one), a... possible postviral or vaccinal encephalitis ...e). There were no focal neurological ab...malities and ... and EEG tests were normal. Abno... laboratory results ...re significantly raised urinary ...thylmal... acid compared with age-matched control... (n=103), low haemoglobin in four children, ... low ... IgA in ... children.

Interp...tation ...e identi... associated gastrointestinal di...se and ...evelopmental regression in a group of previ... ...ise ... which was generally associated in time ... possible environmental triggers.

Lancet 199...; **351**: 637–41
See Commentary page

Inflammatory Bowel Disease Study Group, University Departments of Medicine and Histopathology (A J Wakefield FRCS, A Anthony MB, J Linnell PHD, A P Dhillon MRCPath, S E Davies MRCPath) **and the University Departments of Paediatric Gastroenterology** (S H Murch MB, D M Casson MRCP, M Malik MRCP, M A Thomson FRCP, J A Walker-Smith FRCP), **Child and Adolescent Psychiatry** (M Berelowitz FRCPsych), **Neurology** (P Harvey FRCP), **and Radiology** (A Valentine FRCR), **Royal Free Hospital and School of Medicine, London NW3 2QG, UK**

Correspondence to: Dr A J Wakefield

Introduction

We saw several children who, after a ... of apparent normality, lost acquired skills, includ... ...munication. They all had gastrointestinal ... symptoms, ...luding abdominal pain, diarrhoea, and ...iting and, ...some cases, food intolerance. We ...scribe ... clinical f...ings, and gastrointestinal featur... these ch...ren.

Patients and meth...s

12 children, con...secutively ...ed to ... department of paediatric gast...erology ... a history of a pervasive developmental ...der with los... ...d skills and intestinal symptoms ...rrhoea, abdominal ...n, bloating and food intolerance), were inv...ated. All children were admitted to the ward ...week, accomp...d by their parents.

Clinical investigations

... took histo... including details of immunisations and ...ure to infec...s diseases, and assessed the children. In 11 ca...he history ...as obtained by the senior clinician (JW-S). Neur... and psychiatric assessments were done by ...sultant staff (PH, MB) with HMS-4 criteria.[7] Developmental ... included a review of prospective developmental records ...rom parents, health visitors, and general practitioners. Four children did not undergo psychiatric assessment in hospital; all had been assessed professionally elsewhere, so these assessments were used as the basis for their behavioural diagnosis.

After bowel preparation, ileocolonoscopy was performed by SHM or MAT under sedation with midazolam and pethidine. Paired frozen and formalin-fixed mucosal biopsy samples were taken from the terminal ileum; ascending, transverse, descending, and sigmoid colons, and from the rectum. The procedure was recorded by video or still images, and were compared with images of the previous seven consecutive paediatric colonoscopies (four normal colonoscopies and three on children with ulcerative colitis), in which the physician reported normal appearances in the terminal ileum. Barium follow-through radiography was possible in some cases.

Also under sedation, cerebral magnetic-resonance imaging (MRI), electroencephalography (EEG) including visual, brain stem auditory, and sensory evoked potentials (where compliance made these possible), and lumbar puncture were done.

Laboratory investigations

Thyroid function, serum long-chain fatty acids, and cerebrospinal-fluid lactate were measured to exclude known causes of childhood neurodegenerative disease. Urinary methylmalonic acid was measured in random urine samples from eight of the 12 children and 14 age-matched and sex-matched normal controls, by a modification of a technique described previously.[8] Chromatograms were scanned digitally on computer, to analyse the methylmalonic-acid zones from cases and controls. Urinary methylmalonic-acid concentrations in patients and controls were compared by a two-sample *t* test. Urinary creatinine was estimated by routine spectrophotometric assay.

Children were screened for antiendomysial antibodies and boys were screened for fragile-X if this had not been done

Figure 14.15: Now THAT'S a retraction notice!

is in Retraction Watch's database, it will show up as retracted in your Zotero. In Figure 14.16 you can see a screenshot of my Zotero, showing two papers that were retracted. The panel on the right explains why it was retracted, and a simple red cross gives me a quick indication that these papers have been retracted.

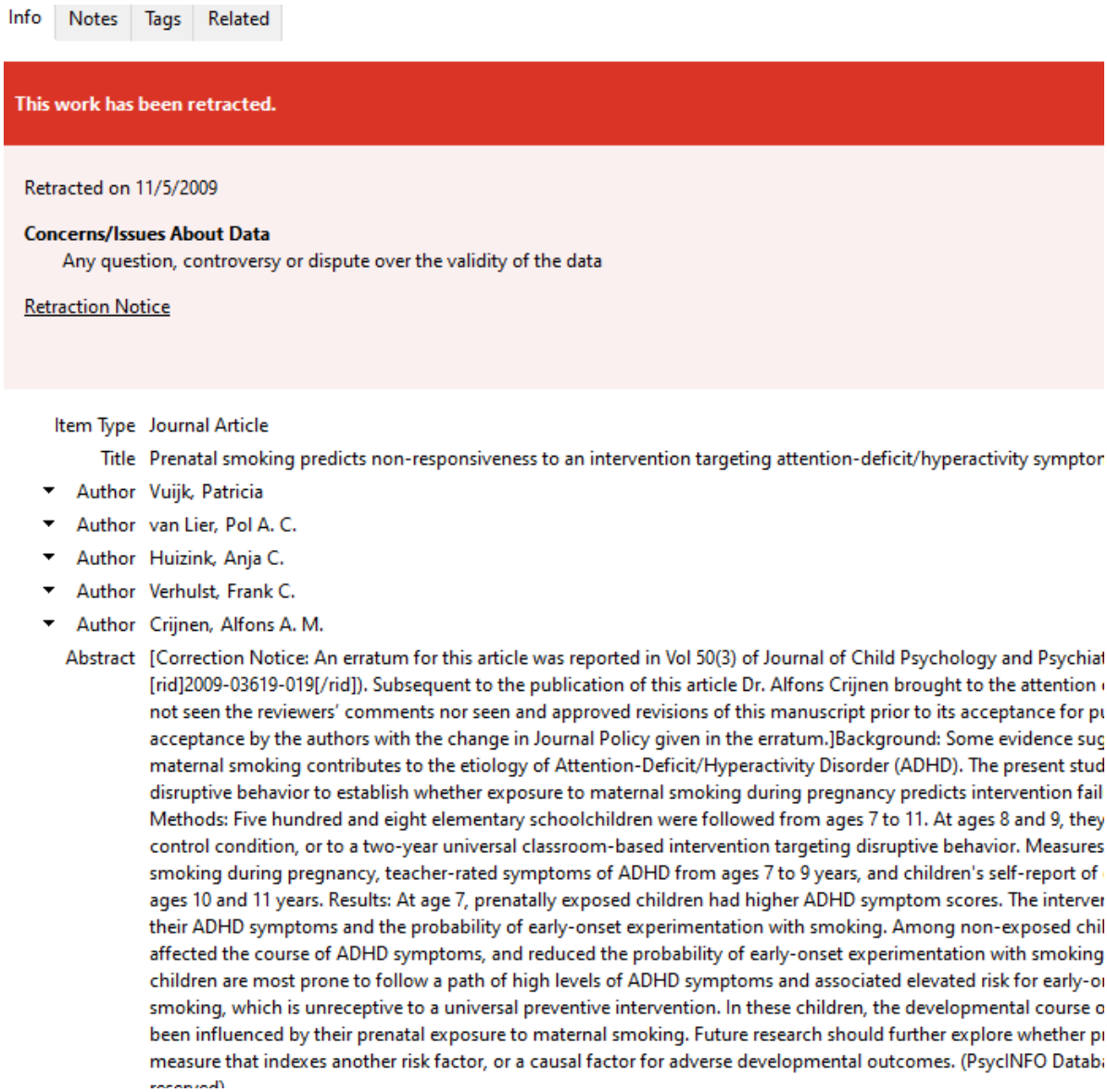


Figure 14.16: A nice and easy notice of retraction. Thanks Zotero & Retraction Watch!

15 Common Problems that Hamper Reproducibility

15.1 Summary of Some Common Problems

We've now seen the issues stemming from crises of reproducibility across multiple academic fields. Before we can discuss potential solutions to the many problems, let's take stock of some common problems that hamper reproducibility in Figure 15.1.

Notice how the issues are arranged in a circle. This is not a coincidence, as such researcher degrees of freedom are present at all steps in the research cycle. Consider the image in Figure 15.2, which comes from a 2017 paper in *Nature*.[49]

15.2 HARKing

Besides illustrating the many possibilities for researcher degrees of freedom to introduce bias into the research process, Figure 15.2 it also points to the pernicious problem of **HARKing** (Hypothesizing After Results are Known),[50] which is tantamount to circular reasoning. In practical terms, this process often looks like this:

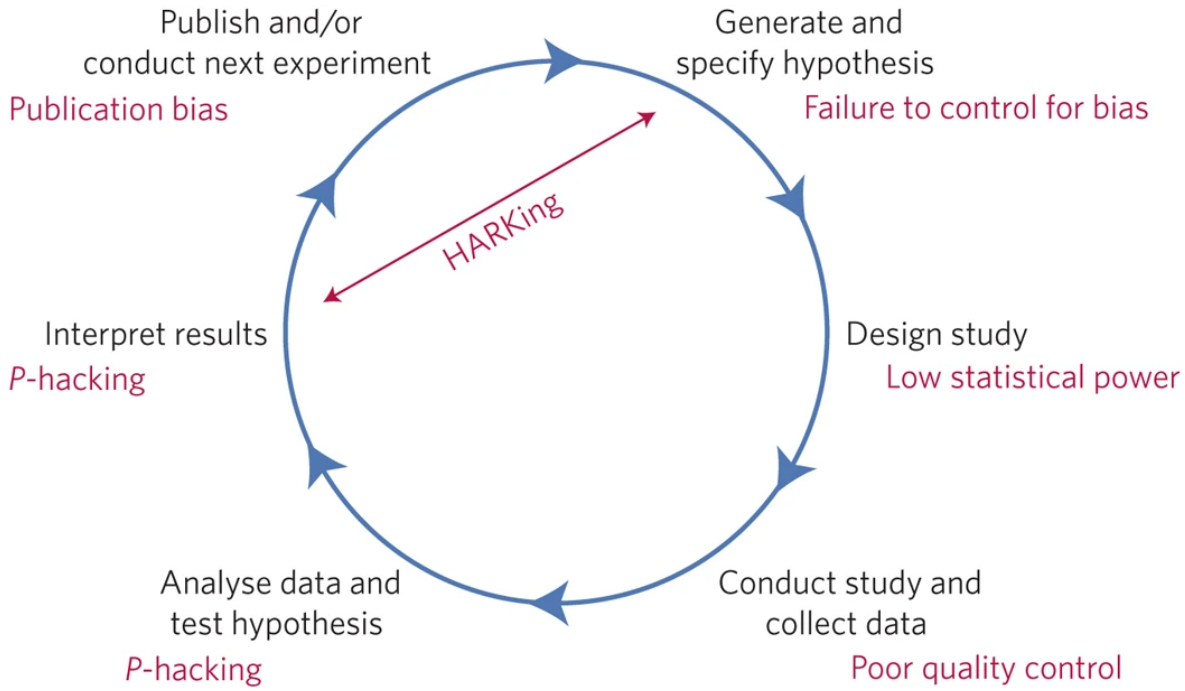
1. The researcher conducts a hypothesis test based on a hunch H.
2. They don't find what they expect, but find something statistically significant U.
3. The researcher writes up the results as $H = U$using the same data.

You may wonder, what's the big deal? If you found something unexpected and interesting, isn't that the point of science? To be clear, if the researcher starts with a hunch H and finds something unexpected and statistically significant U, it's absolutely OK and even a good idea to explore U in a new study with different data. In fact, some suggest that this form of transparent HARKing (also known as THARKing) in which authors present new hypotheses derived from the data in the Discussion section of an article can be beneficial for knowledge creation.[51] I agree with this position, as transparently stating your thoughts on potential new hypotheses can generate new ideas for future research, and also clearly bounds the discussion in an exploratory sense. But using the same data for H and U and pretending like H never existed is a problem.

Figure 15.1: Figure taken from the report of the symposium, 'Reproducibility and reliability of biomedical research', organised by the Academy of Medical Sciences, BBSRC, MRC and Wellcome Trust in April 2015. Full report here.

Figure 1: Threats to reproducible science.

From: A manifesto for reproducible science



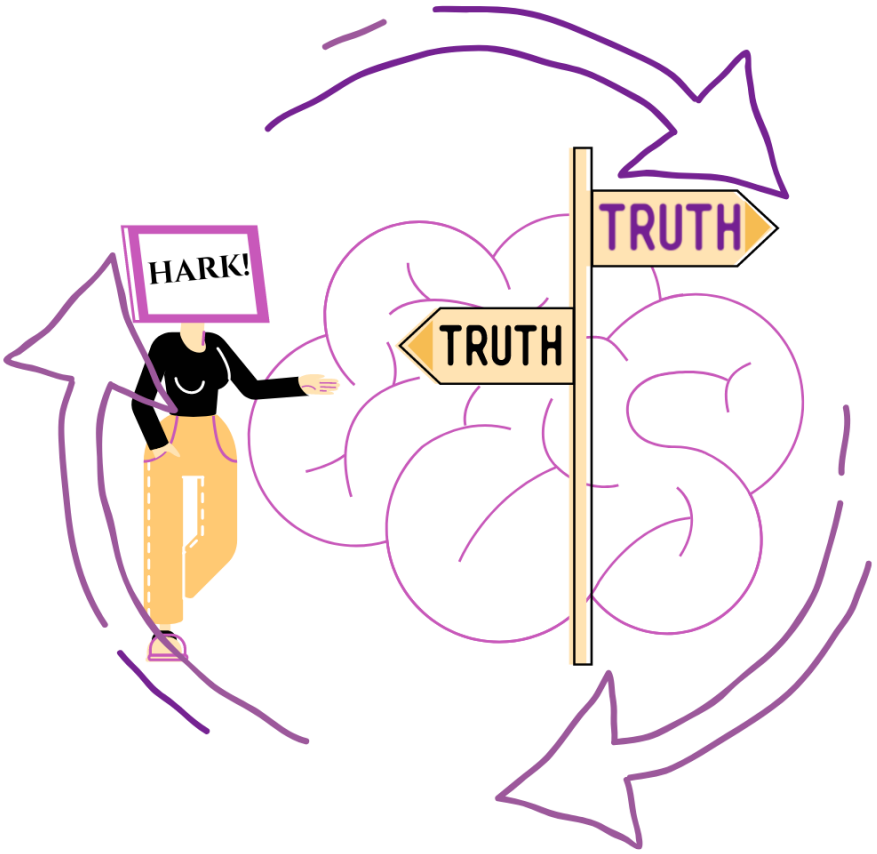
An idealized version of the hypothetico-deductive model of the scientific method is shown. Various potential threats to this model exist (indicated in red) including lack of replication[5], hypothesizing after the results are known (HARKing)[7], poor study design, low statistical power[2], analytical flexibility[5,1], *P*-hacking[4], publication bias[3] and lack of data sharing[6]. Together these will serve to undermine the robustness of published research, and may also impact o the ability of science to self-correct.

Figure 15.2: Researcher degrees of freedom can happen at any stage of the research process.

For one, if hypotheses that are falsified are never reported, that distorts the evidence base for what hypotheses are supported and which hypotheses are not supported. Imagine 10 different researchers all investigate the same hypothesis H, and find that the hypothesis is not supported by the data, but don't report it, then other researchers may continue to pursue H. This is a waste of resources. Instead, if all researchers reported that H is not supported by their data, this tells us that H might not be the right avenue to pursue.

Additionally, if the unexpected hypothesis U is reported as the *a priori* hypothesis, this reduces reproducibility as U may be unique to the sample, and not generalizable. It may represent a false positive or false negative. This is especially bad if the researcher is cherry-picking their results to support unexpected finding U because it is a straight-forward distortion of their findings. Reporting only statistically significant results and concealing non-significant results leads to a distortion in the evidence base and potentially wastes resources, as described above.

There are always chance variations in effect size due to sample size, the underlying odds in the effect being true before the study is even conducted, the composition of the sample, the variation in population effects, and more. Therefore, reporting U as H can be a case of a researcher being fooled by randomness, and reporting spurious effects as real ones.



It's also worth noting that authors have since noting some variants of HARKing, which are shown in Table 15.1.[52]

Table 15.1: Variations of HARKing.

Acronym	Full Form	Description
CHARKing	Constructing Hypotheses After Results are Known	This is like the description of HARKing above. Creating hypotheses after the fact and presenting them as having been constructed *a priori*.
RHARKing	Retrieving Hypotheses After Results are Known	Let's say the researchers had previously proposed some hypotheses but then dropped them. After the fact, they might revive those hypotheses and present them as having been known *a priori*.

Acronym	Full Form	Description
SHARKing	Suppressing Hypotheses After Results are Known	Not reporting *a priori* hypotheses that were not supported by the results. This could also be not reporting *a priori* hypotheses supported by results just because the researchers didn't like these hypotheses.

A key problem of all variations of HARKing is that they can lead to theories that are *overfit* by the results.[52] This means that the theories can become too bound up in the results of individual sample results, and thus are less useful at more general explanations of a phenomenon when applied to other samples and contexts.

15.3 Publication Bias & the File Drawer Problem

You may be wondering what motivates researcher degrees of freedom such as HARKing? One important consideration is the incentives that researchers have for engaging in researcher de-

grees of freedom. A key incentive is to be able to publish one's results. In a perfect world perhaps, all results would have an equal opportunity to be published regardless of statistical significance. Unfortunately, this is not the case.

Studies with statistically significant results are more likely to be written up and published than studies with non-statistically significant results (i.e. *null* results as they are often called).[53] [54][55] [56] Publication bias can be defined as *An editorial predilection for publishing particular findings, e.g., positive results, which leads to the failure of authors to submit negative findings for publication.*[57] The editors of journals want to publish ground-breaking research so that the prestige, relevance, and citation frequency for their journal is increased. Sadly, null or negative results are not as attractive to publish than statistically significant results, despite being as important for providing a more complete picture of the evidence base for a given phenomenon. This can lead to substantial distortions of the evidence base.

Consider, for example, the findings of a 2021 meta-analytic study published in the prestigious journal *Proceedings of the National Academy of Sciences* which suggested that nudge interventions (light-touch behavioral interventions which sway an individual toward a particular choice without restricting their ability to make a different choice) have an average effect size of $d = 0.43$.[58] This is a pretty large effect in the social science literature! It's also a remarkable headline-worthy finding. However, even meta-analyses can be prone to publication bias, because any evidence syntheses is only as good as the evidence it synthesizes. Thus, when another research team applied a novel technique to account for publication bias, they found that the average effect size of nudge interventions dropped to about $d = 0.04 - 0.11$, which the authors concluded was suggestive of no evidence of such interventions.[59] While the final word on nudge interventions is far from being written, this back-and-forth shows how the severity of publication bias can lead to drastically different estimates of overall effect.

Publication bias thus naturally leads researchers to suppress null or non-statistically significant findings. Many researchers don't even write up their null result studies, and simply abandon them forever. This is known as the **file drawer problem**, where null results are relegated to a virtual or actual file drawer, never to see the light of day.[60] The extreme case of the problem is that 5% of studies that have false positives (Type I errors) are published in journals, and 95% of studies with non-statistically significant results are relegated to the file drawer.[60] However, consider that p-hacking increases the likelihood of false positives in the literature as well.

When examining the problem empirically based on studies supported by a National Science Foundation program, a group of authors found some troubling results,[55] seen in Figure 15.3.

Publication bias is such an extensive problem (and having been recognized in the literature for over 40 years), that researchers are continually developing new methods to detect publication bias in evidence syntheses.[61] [62][63] Of course, there are other potential solutions to the problem which we'll discuss in the next chapter, but suffice it to say, we must always be cognizant of publication bias when evaluating a body of evidence.

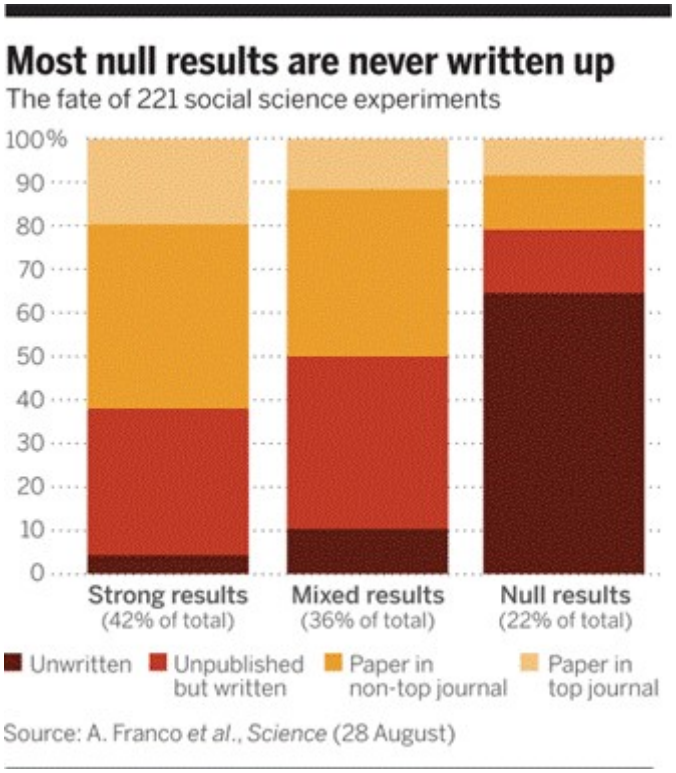
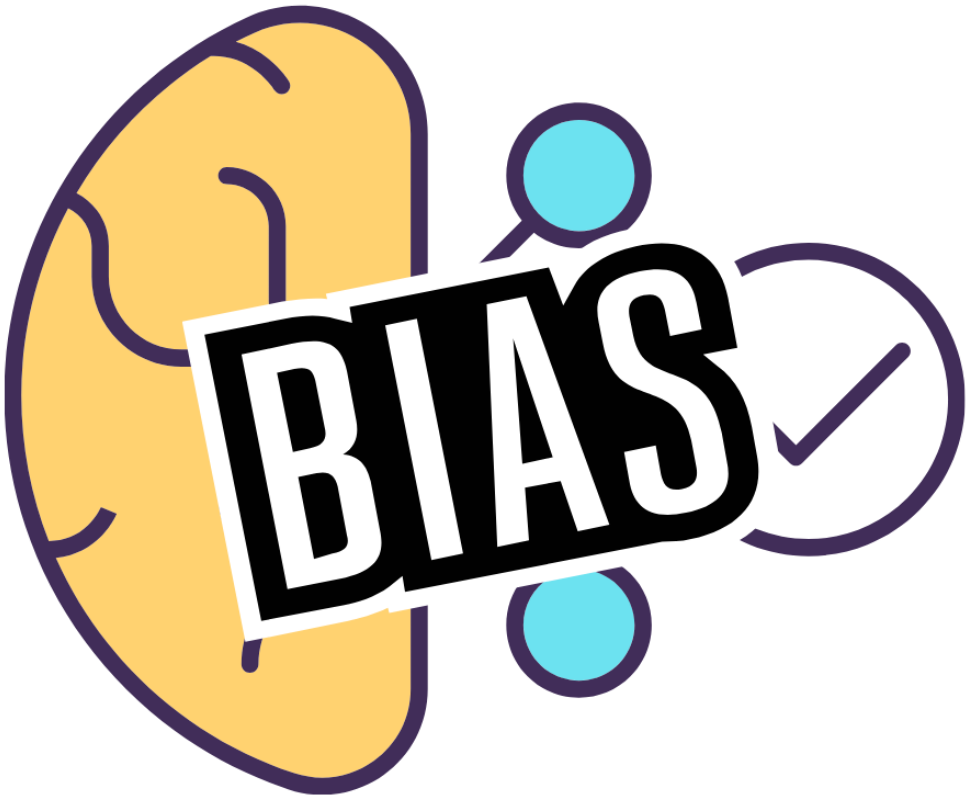


Figure 15.3: Publication bias results in fewer results seeing the light of day!

15.4 Cognitive Biases



You might be thinking at this point, *Why do researchers engage in researcher degrees of freedom if they know the consequences to the evidence base?* That's just the thing. They may willfully engage in unethical research practices, but they just as well be fooled by various cognitive biases to which we can all fall prey. For one, researchers have a strong incentive to publish papers, as their tenure and promotion is often heavily dependent on their publication output (hence the phrase *Publish or Perish*). Then given that journals are more likely to publish studies with statistically significant findings, researchers have a stake in the results of their quantitative research. Additionally, the salaries of researchers in many fields is also partly (and sometimes entirely) contingent on acquiring research funding, where a strong publication record is expected.

These conditions work synergistically with our inherent cognitive biases. For one, consider **confirmation bias**, or our tendency to look for evidence and conduct research in a way that aligns with our existing beliefs and hypotheses.[64] In essence, confirmation bias pushes us to find what we want to find, rather than what is actually there. Though the term *confirmation*

bias is relatively new, the phenomenon has been observed in research for quite some time. Consider this quote from 1869 by journalist Charles Mackay:

> *When men wish to construct or support a theory, how they torture facts into their service!*[65]

We should be clear that confirmation bias need not be a conscious, malicious choice, but rather a subconscious motive which can affect even the most principled open scientists. A key consideration is that a research question differs from a hypothesis.[66] The differences lies in how variables within a study are measured and the relationships between them postulated by hypotheses. Think of the research question as the big picture investigation of the phenomenon, and the hypotheses as being specific relationships of interest. If a researcher conducts a study and does not find evidence consistent with their hypothesis, a natural question would be if they constructed (or *operationalized*) their conceptual constructs in the right way. They may ask, *Is this a problem with the implementation, measurement, or underlying theory?* This is a valid question, and there are often no clear answers. However, there are solutions we will discuss in the next chapter that can mitigate biases such as this.

Similar to confirmation bias is **hindsight bias**, or the tendency to make a prediction after the fact (a *postdiction*) where a person thinks they knew it all along.[67] For example, a researcher makes some vague prediction in their mind before conducting a study, conducts the study, and then selects one of many outcomes as something they predicted all along. This leads to overconfidence in the obtained results, and conflates prediction with postdiction.[68] This can go hand-in-hand with HARKing, whereby the researcher creates hypotheses after obtaining results, and espouses the view that they knew this would happen all along.

Another important cognitive bias which influences reproducibility and research transparency is **apophenia**, or the human tendency to see patterns in randomness. A common example of this is seeing a face among a collection of inanimate objects, or seeing animal shapes in clouds. Consider the image in Figure 15.4.

The image shows the same mesa on Mars - one captured in 1976 (on the left) and another captured in 2001 (on the right). The picture on the right has an image resolution 10 times higher than the image on the left. However, when NASA released the image on the left to the public in 1976, they thought the apparent 'face,' resembling an ancient Egyptian planet, would drum up public interest. What they didn't realize is that it led to a lot of speculation about ancient civilizations on Mars, and conspiracy theories that NASA was hiding the facts.[69] Even after the 2001 image was released, it didn't satisfy everyone. Some still believed that this 'face' is evidence of a civilization on Mars. This reflects a natural tendency to see patterns where there are none.

When applied to research, apophenia can lead researchers to infer meaningful patterns based on noisy (or meaningless) data. For instance, running many hypothesis tests increases the likelihood of observing a false-positive, which researchers may perceive as a valid finding.

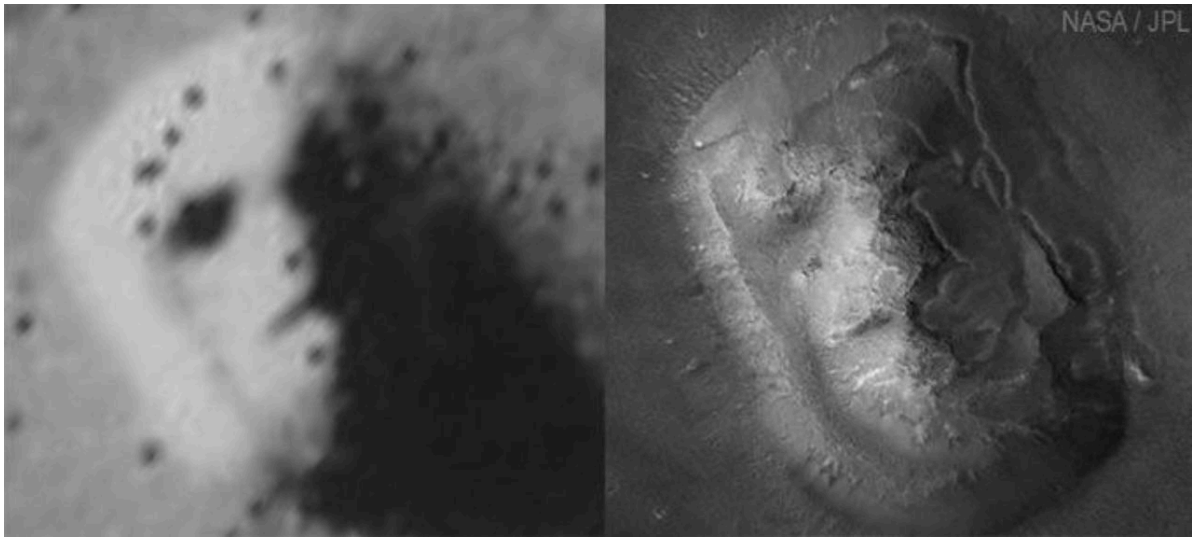


Figure 15.4: The "Face on Mars" (left) photographed in 1976 by the NASA Viking 1 orbiter. On the right is the same image taken in 2001 by the NASA Mars Global Surveyor. *Source: NASA/JPL.*

Then, one or more cognitive biases can result in hypotheses and theories based on noise and spurious findings.

Have you ever heard the expression, *Correlation does not necessarily equal causation*? This is because just because two things are statistically related or similar does not mean that there is any meaningful relationship between them. However, our tendency toward apophenia can lead us to conclude a meaningful correlation when there isn't one. Consider the image in Figure 15.5 from the website *Spurious Correlations* created by Tyler Vigen. The image shows a remarkably strong (and by chance) correlation between the number of letters in the winning word of the Scripps National Spelling Bee and the number of people killed by venomous spiders each year.

In sum, we should be mindful of the many cognitive biases that can shape our conduct of research. While the situation may seem hopeless, know that there is hope! In the next chapter, we'll discuss many advances in Open Science that can help mitigate these biases in the conduct of research.

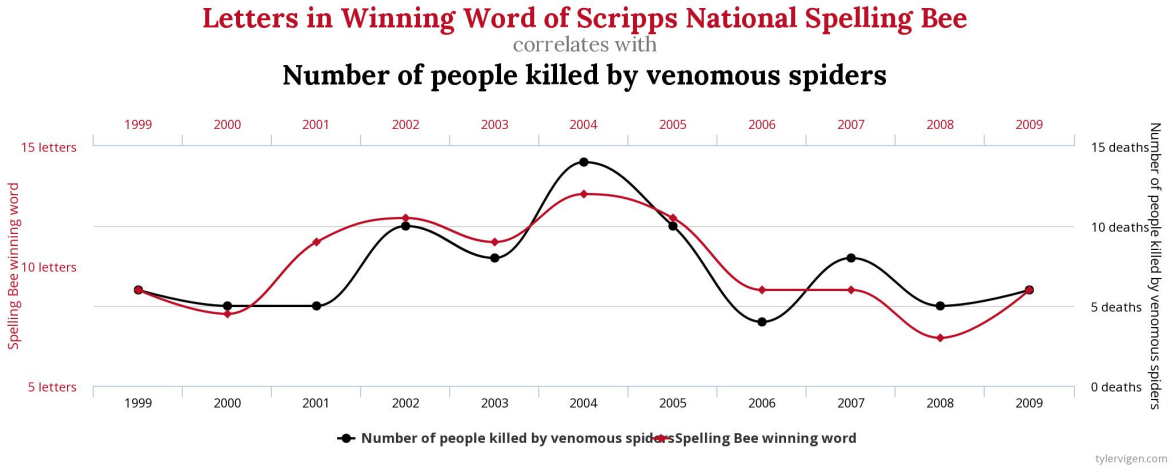


Figure 15.5: Apophenia leads us to infer patterns within random noise. *Credit: Tylervigen.com.*

16 Potential Solutions & the Way Forward

If you've made it this far having read all the previous chapters, I hope you are not dismayed at the state of research transparency and reproducibility (RT2) in science. If you are, I don't blame you. It's easy to feel helpless at the state of research. However, there are many advances that have been made, and that will be made to improve things. Consider this, about 15 years ago, RT2 was not really a thing. Most folks had no idea about problems with replicating and reproducing results of empirical studies. Sure, some folks like John Ioannidis were already sounding the alarms for a while as we say in Section 14.2, but I think it's safe to say most researchers either didn't know or didn't care about RT2 problems (many still don't).

Since that time, there have been a number of changes to the RT2 landscape, and a push toward Open Science has been increasing. Let's review some of the potential solutions to the problems we've seen.

16.1 A Partial List

First, let's look at a table of potential solutions advanced in an excellent *Nature* article entitled *A manifesto for reproducible science*[49] in Table 16.1.

Table 16.1: A partial list of potential solutions. Adapted from Table 1 in the original article by Munafò et al.[49]

Potential Solution Area	Example(s) of Potential Solutions
Protecting against cognitive biases	Blinding
Improved methodological training	Rigorous training in statistics and research methods for future researchers. Rigorous continuing education in statistics and methods for current researchers.
Independent methodological support	Involvement of methodologists in research. Independent oversight.
Collaboration and team science	Multi-site studies/distributed data collection. Team-science consortia.
Promoting study pre-registration	Registered Reports. Open Science Framework use.

Potential Solution Area	Example(s) of Potential Solutions
Improving the quality of reporting	Using reporting checklists. Protocol checklists.
Protecting against conflicts of interest	Disclosure of conflicts of interest. Exclusion/containment of financial and non-financial conflicts of interest.
Encouraging transparency and open science	Open data, code, and related materials. Pre-registration.
Diversifying peer review	Preprints. Pre- and post-publication peer review such as Publons and PubMed Commons.
Rewarding open and reproducible practices	Badges. Transparency and Openness Promotion Guidelines. Funding replication studies. Registered Reports. Open science practices in hiring and promotion.

That's a pretty good list. It's not complete, nor could it be, as advances are continually being made, but it's summarizes potential solutions across a number of areas nicely. Let's now explore some potential solutions in detail.

16.2 Pre-registration

16.2.1 General Description

Pre-registration refers to *specifying your research plan in advance of your study and submitting it to a registry.*[70] It's basically saying what you plan to do and how you plan to do it, and then publishing those details in the public domain before you conduct your study. In this way, pre-registration separates **hypothesis-testing** (confirmatory) research from **hypothesis-generating** (exploratory) research as shown in Figure 16.2.

As we saw in Chapter 15, humans are prone to several cognitive biases which can affect our ability to separate *a priori* from *a posteriori* thinking, even if we do not mean to conflate the two. Pre-registration can mitigate these biases by clearly stating at the outset what analyses will be used to confirm or falsify hypotheses, and which analyses will be exploratory. Then, the reader can judge the merit of the work for themselves. Ideally before a researcher collects any data, they should pre-register their confirmatory hypotheses and related analyses in the public domain. There are several benefits of doing so in terms of reducing bias due to:

- Selective reporting of outcomes.
- Different analytic decisions that can lead to different results.

Figure 16.1: Wise capybaras feel motivated and encouraged by advances in Open Science, and I agree with them!

Figure 16.2: Pre-registration is not a straitjacket; you can still conduct exploratory analyses and try things out. It's just that you should be clear which analyses will be confirmatory and which will be exploratory.

- Publication bias, because it corrects the denominator of studies and hypotheses out there.
- Cognitive biases like apophenia, hindsight bias, and confirmation bias.

16.2.2 Common Apprehensions

Now, let's say you followed my advice above and pre-registered your study in the public domain prior to data collection. Then, in the course of the research process, you discover that something you said you would do in your pre-registration isn't working, and you have to pivot to a different strategy. Is that OK? Yes! As long as you clearly state any deviations from your pre-registered plan, it's totally fine if you need to adjust something. It's all about being transparent about what you did. Additionally, even if you already have a dataframe you want to analyze, you can still pre-register your analysis plan prior to analyzing the data. This is not the ideal scenario as it still leaves a researcher degree of freedom to cheat and peek at the data, but at least it shows people exactly what you did and readers can judge the veracity of the pre-registration for themselves.

You may wonder, *Won't someone steal my ideas if I pre-register my ideas?* This is a valid concern, and though I don't believe this happens frequently, the point is that it *can* happen. The solution is therefore to **embargo** the pre-registration, which keeps it private for a certain amount of time (up to four years on some platforms) before it is made public. This allows researchers plenty of time to conduct a pre-registered study without worrying about their ideas being stolen (also known as **getting scooped**). Additionally, once the researcher's ideas are pre-registered and public, that provides a timestamp to lay claim to their ideas. That is, if

someone else steals your exact set of analyses, data, and code, you have a record of having registered these in the public domain prior to their study being published. In my experience, getting your ideas stolen due to preregistration doesn't really happen. In fact, some believe that the fear of being scooped is not a legitimate drawback against pre-registration as papers are unlikely to be rejected due to scooping alone, no two scooped studies can be the same, and if someone uses your methods on a different sample, that's actually a good thing for science![71] Of course, if one is still afraid, the embargo period is an elegant solution.

16.2.3 What Should Be Included & Where to Pre-register

You can be as detailed as you want in a pre-registration, but there are a few elements that should be described as a minimum. For instance, you should describe the hypotheses, planned analyses, and criteria for falsifying or supporting the hypotheses. In general, a reader should be able to replicate the methodology used, and the pre-registration should minimize as many researcher degrees of freedom as possible for confirmatory analyses. Personally, I treat the pre-registration as the bulk of my Methods section that will eventually end up in the published manuscript (save any deviations from the pre-registration which I include later in the final manuscript). This includes information about the variables being measured, a detailed analysis plan, and several back-up plans if what I want to do fails for some reason. If possible, the actual code that will be used to conduct the analyses should be included as well.

The pre-registration platform *aspredicted.org* provides perhaps the simplest template for create a pre-registration, wherein researchers need to answer the following eight questions:

1. Have any data been collected for this study already?
2. What's the main question being asked or hypothesis being tested in this study?
3. Describe the key dependent variable(s) specifying how they will be measured.
4. How many and which conditions will participants be assigned to?
5. Specify exactly which analyses you will conduct to examine the main question/hypothesis.
6. Any secondary analyses?
7. How many observations will be collected or what will determine sample size?
8. Anything else you would like to pre-register (e.g., data exclusions, variables collected for exploratory purposes, unusual analyses planned)?

My favorite platform for pre-registration and posting all research materials in the public domain is the **Open Science Framework (OSF)**.You can pre-register any study on the OSF, and it's a great place to also put all your de-identified data, code, and materials used to

generate your analyses. Besides having an excellent repository of study materials and pre-registrations, the site also provides several resources for learning more about pre-registration and Open Science. It also has an embargo period of four years should you wish to maintain privacy of the pre-registration for a certain period of time.

There are also study design-specific registries. For instance, the American Economic Association has a registry for Randomized Controlled Trials only. The PROSPERO registry hosted by York University in the UK is THE go-to place to register systematic reviews, meta-analyses, rapid reviews, and umbrella reviews. Clinical trials in the US have to registered on clinicaltrials.gov, but this registry is also applicable to any social science randomized controlled trial. The International Initiative for Impact Evaluation (3ie) hosts the Registry for International Development Impact Evaluations. Thus, there are many places to pre-register a study! If you're at a loss for where to register, I suggest trying the Open Science Framework.

> **A Pre-Registered Analysis Plan**
>
> Want to see an example of a real study pre-registration? Here's one of mine on the Open Science Framework. It's got hypotheses, analysis plans, diagrams, equations, references, and all kinds of good stuff! This study analyzed restricted-access longitudinal secondary data. This meant that I had to apply to get this data from a data vendor. Only after receiving the data could I gauge what variables I could include in an analysis plan. As a result, I create two pre-registrations: one before I received the data, and another after receiving the data but before conducting my main analyses. The link is to the second pre-registration, but you can also find a link to the first pre-registration on the same page. Check it out!

16.3 Improved Methodological Training

Any scientific discipline that makes use of statistical analysis can be open to researcher degrees of freedom, and consequently, spurious research findings. Thus, appropriate and rigorous statistical training is needed for robust quantitative science. Additionally, researchers need robust training in study design and implementation. Research design and statistical analyses are mutually dependent and reinforcing.[49] Even if you've received a PhD with solid training in statistics, keeping up the latest advances requires a concerted effort, and is necessary. In particular the following areas of methodological training have been cited as especially crucial to RT2,[49] seen in Figure 16.3.

The use of statistics in quantitative research is so ubiquitous that you'd be hard-pressed to find a scientific field that DOES NOT use statistics for any research involving numbers. Thus, most researchers may not be specialists in statistics (I know I'm certainly not), but need to know they key components in order to use them appropriately. This is why it is crucial to consult methodologists (those whose primary work concerns methodology development and training)

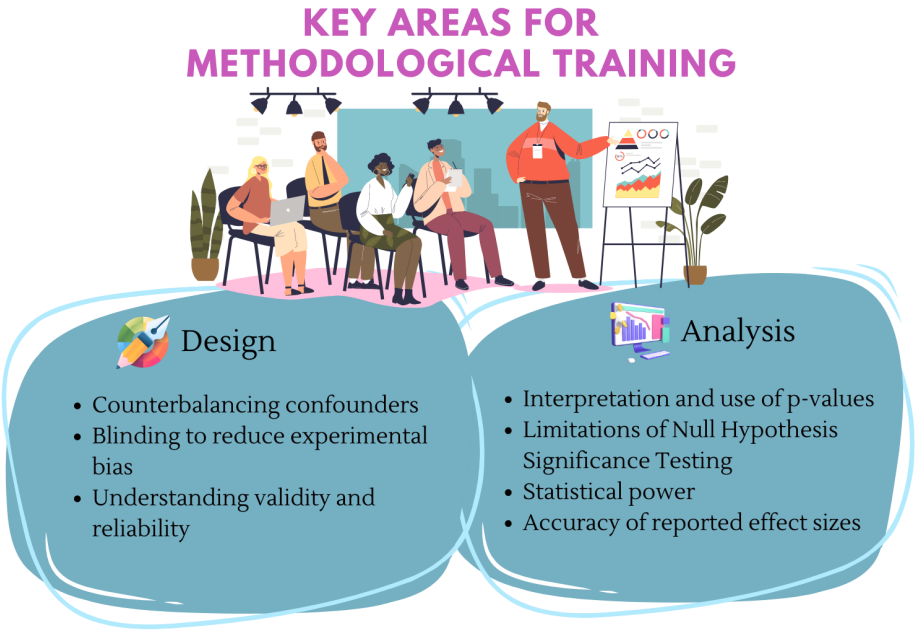


Figure 16.3: Methodological training at all stages of one's career is crucial to keep up with the latest advances in rigorous design and analysis.

and statistical experts whenever possible. However, non-specialists who use statistics should be appropriately trained so we don't make the same mistakes as researchers in the past. When reporting the results of a study, some suggest a simple 21-word 'solution':

> *We report how we determined our sample size, all data exclusions (if any), all manipulations, and all measures in the study."*[72]

This isn't a 'solution' in the mathematical sense or in the sense we are discussing here. Rather, it's a sentence that prompts researchers to clearly describe important elements of their study.

One useful way to improve methodological training is to train students to replicate a given study. This can apply to students in formal degree programs, but could just as well be applied to participants at a workshop who may be continuing their education. This is commonly done in certain fields currently, but there's no reason why it cannot be applied broadly to any fields who engage in quantitative research. In fact, there is an increasing push toward teaching replication at the undergraduate and graduate levels.[73] [74][75] An example of a replication pedagogy workflow for a graduate-level experimental methods courses is shown in Figure 16.4.

The schematic above illustrates one possible pedagogical workflow for a graduate-level course over 10 weeks. It's not the only approach, but a good place to start. In this workflow, students begin with writing brief proposals for determining which studies they want to replicate, which

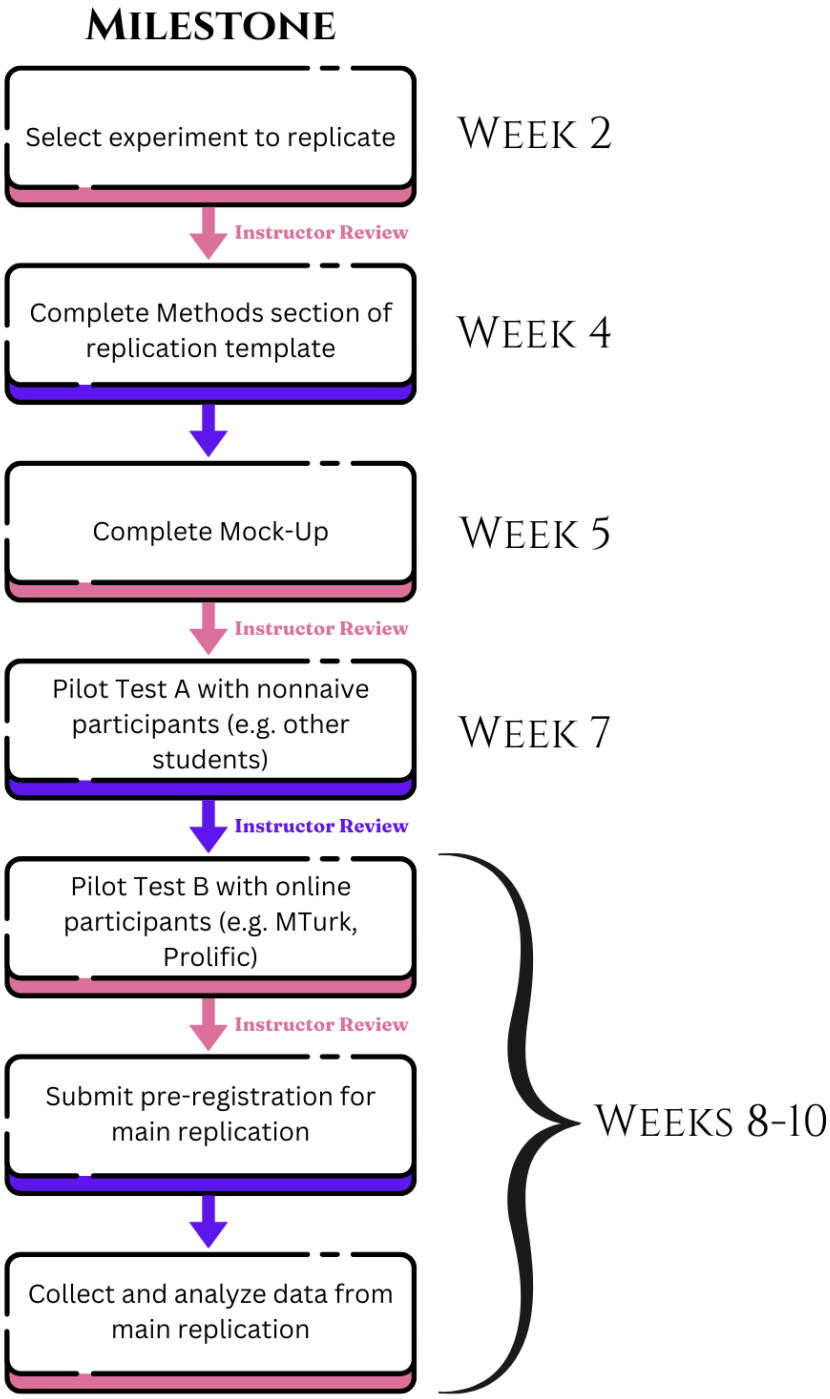


Figure 16.4: An example of a graduate-level replication pedagogy workflow. Adapted from Hawinks et al.[75]

are reviewed by an instructor for feasibility. Next, they complete the Methods section of a replication template developed by the Open Science Collaboration.[39] They then work on outlining their proposed replication experiment, which is reviewed by the instructor. Once approved, students proceed with their experimental protocol in two pilot tests. The first test is conducted with nonnaive participants, meaning the people in the class. This is to make sure the data are logged accurately, and confirmatory analysis code is working properly. After an instructor reviews the results, students proceed to a pilot test with real-world participants who are not in the class on an online platform such as MTurk or Prolific. This is done to make sure participants don't experience any issues with completing the experiment, and to solicit feedback to determine if any design changes should be made. Following instructor review of the second pilot test, students pre-register and run their main replication.

By making replication and pre-registration familiar to students, the hope is that this will transfer into actual research practice. At the very least, it can help normalize these processes by including them within curricula.

16.4 Incentives

How does one incentivize Open Science practices like sharing data, code, and related materials in the public domain? One promising approach is inspired by the world of video games! Really! If you've ever played a video in the past 10 years or so, you have likely encountered some variation of an achievement badge for some task (e.g., leveling-up, defeating a boss, etc). The Center for Open Science developed several digital badges for use by journals to award researchers for pre-registering their study, sharing their data, and sharing their materials. The current suite of badges are shown in Figure 16.5.

Over 100 journals currently award these open science badges to authors who do the thing associated with each badge. The meaning of each badge is summarized in Table 16.2.

Table 16.2: Description of Open Science Badges. *Source: American Psychological Association.*[77]

Badge	Description
Preregistered	At least one study has been preregistered with descriptions of the research design and study materials, including the planned sample size, research question(s) or hypothesis(ses), outcome variable(s), predictor variable(s). Results must be fully disclosed.
Preregistered	At least one study has been preregistered along with the analysis plan for the research, and the results are reported according to that plan.

Badge	Description
Open Materials	All digitally-shareable materials necessary to reproduce the reported results are provided, and description of non-digitally-shareable materials needed for replication are made publicly available.
Open Data	All data needed to reproduce results are made publicly available. Necessary metadata or codebooks are also provided. A list of such repositories is available here.
Open Data (PA)	PA stands for *Protected Access*. This refers to data that are sensitive or only available from an approved third-party repository that manages access to the data only for qualified researchers.

While the badges might seem silly, they can actually make a difference to Open Science practices. To see if the badges encouraged data sharing, a group of researchers examined 2,478 experimental or observational study articles published from 2012 to the first half of 2015 in five psychology journals.[78] One of these journals *Psychological Science* began awarding open science badges in late 2013. The other four journals were all respected journals in psychology that did not award open science badges during the 2012 to 2015 study period. The results are quite telling. Check out their findings related to the effect of badges on data sharing in Figure 16.6.

The findings suggest that the badges increased data sharing, as well as the usability and accuracy of the data shared. The rate of data-sharing in *Psychological Science* increased from 1.5% to 39%, while the other journals stayed at 5% or less. However, badges are still not going to create more open practices on their own. An evaluation of badge use by the *British Medical Journal* for instance found that the badges did little to increase data sharing.[79] A similar evaluation of badge use by the journal *Biostatistics* showed an increase in data sharing due to badges, but of only about 7.6%, with no impact on code-sharing.[80] Thus, badges alone will not solve the problems in the field, but they are a simple, low- (or no) cost incentive that can improve things, even if only a little bit.

16.5 Funders Hold Tremendous Power

Perhaps better than an incentives for researchers to engage in Open Science practices are requirement to do by funders. For instance, cOAlition S is an initiative by various international organizations and funders to make all research funded by national, regional or international research councils and funding bodies made available immediately in Open Access repositories and journals. This took effect from 2021 and continues to this day among funders who subscribe to this organization's Plan S Principles presented below.[81]

Figure 16.5: Current Center of Open Science badges.[76] [77]

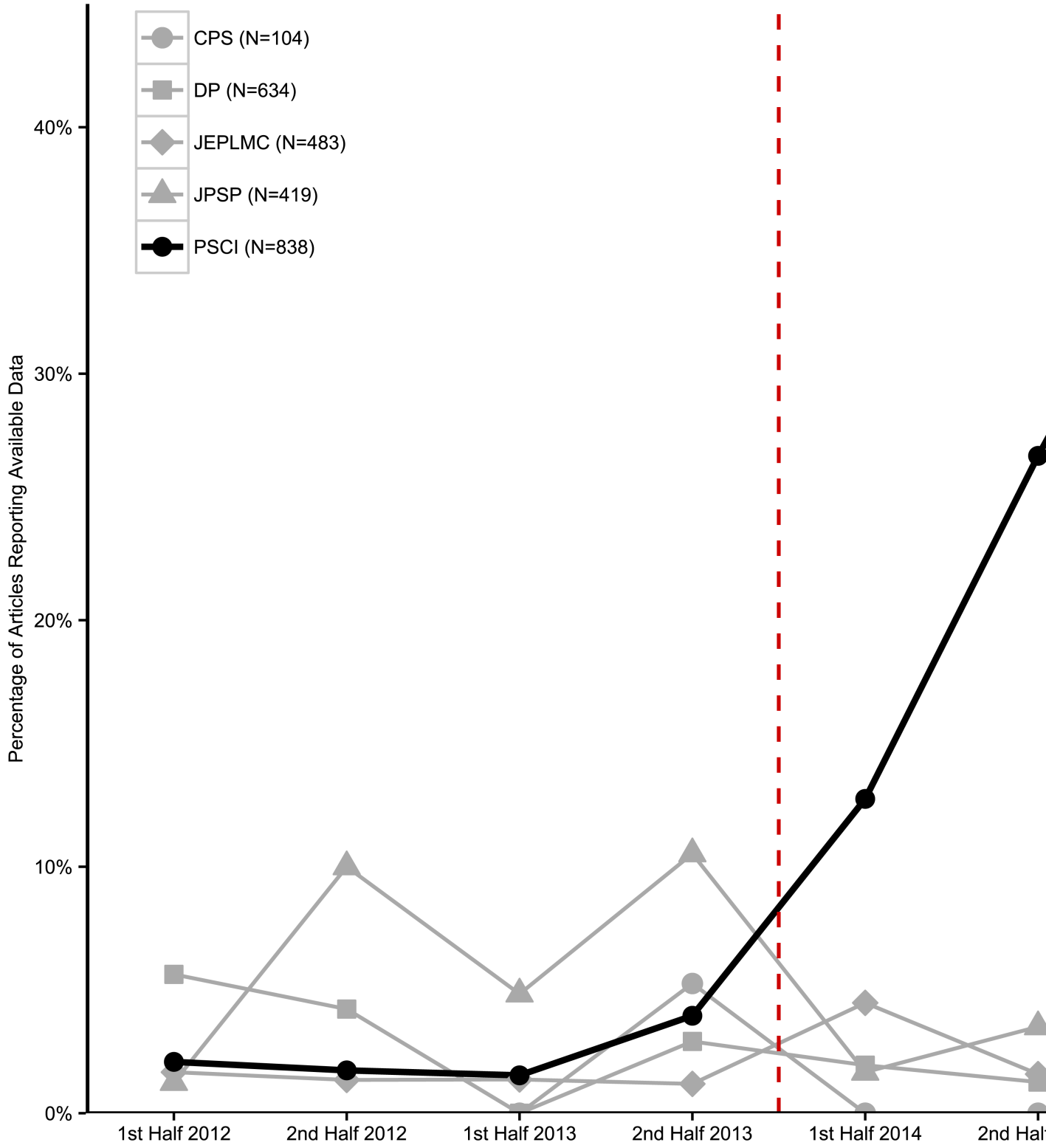


Figure 16.6: After the journal Psychological Science started awarding badges, the rate of data sharing went way up. The dotted red line indicates when Psychological Science started awarding Open Science badges. Other journals who didn't award badges had the same low level of data sharing. From Figure 2 of the original study by Kidwell et al.

1. Authors or their institutions retain copyright to their publications. All publications must be published under an open license, preferably the Creative Commons Attribution license (CC BY), in order to fulfil the requirements defined by the Berlin Declaration.
2. The Funders will develop robust criteria and requirements for the services that high-quality Open Access journals, Open Access platforms, and Open Access repositories must provide.
3. In cases where high-quality Open Access journals or platforms do not yet exist, the Funders will, in a coordinated way, provide incentives to establish and support them when appropriate; support will also be provided for Open Access infrastructures where necessary.
4. Where applicable, Open Access publication fees are covered by the Funders or research institutions, not by individual researchers; it is acknowledged that all researchers should be able to publish their work Open Access.
5. The Funders support the diversity of business models for Open Access journals and platforms. When Open Access publication fees are applied, they must be commensurate with the publication services delivered and the structure of such fees must be transparent to inform the market and facilitate the potential standardisation and capping of payments of fees.
6. The Funders encourage governments, universities, research organisations, libraries, academies, and learned societies to align their strategies, policies, and practices, notably to ensure transparency.
7. The above principles shall apply to all types of scholarly publications, but it is understood that the timeline to achieve Open Access for monographs and book chapters will be longer and requires a separate and due process.
8. The Funders do not support the 'hybrid' model of publishing. However, as a transitional pathway towards full Open Access within a clearly defined timeframe, and only as part of transformative arrangements. Funders may contribute to financially supporting such arrangements.
9. The Funders will monitor compliance and sanction non-compliant beneficiaries/grantees.
10. The Funders commit that when assessing research outputs during funding decisions they will value the intrinsic merit of the work and not consider the publication channel, its impact factor (or other journal metrics), or the publisher.

Many international funders have already endorsed the Plan S Principles such as the Bill & Melinda Gates Foundation, the World Health Organization, Templeton World, the Wellcome Fund, and others. Additionally, a number of journals have also begun adopting the Plan S Principles. Specifically, a Transformative Journal is one that is committed to transitioning to a fully Open Access journal (one that does not place access restrictions on its articles).[81] These

are small steps, but when funders require Open Science practices, that can be a powerful force for change.

16.6 Reporting Guidelines

Reporting guidelines are simple tools that provide a minimum list of information that should be included when writing a study. Having a common minimum set of guidelines helps create consistency across the field, and improve the quality of reporting. For instance, the Equator network is an organization that brings together researchers, funders, journal editors, and others in service of improving the quality of published research. They have links to 577 different reporting guidelines across diverse study designs, research areas, and fields. Some of the main guidelines that researchers use are listed in Figure 16.7. The use of such guidelines is often required by particular journals, and can be a great way to obtain relevant information about key aspects of a study.

Figure 16.7: The Equator network has links to 577 reporting guidelines!

In fact, we now even have Transparency and Openness Promotion (TOP) guidelines which specify eight standard toward greater scientific openness.[82] The complete list of guidelines can be found here. There are now thousands of journals that have implemented at least two of the TOP guidelines, and journals can be checked for their *TOP Factor* which provides a numeric rating of how well they are implementing Open Science practices. The directory of journal TOP factors can be found here.

16.7 Preprints and Registered Reports

Preprints are research papers that are shared in the public domain prior to peer review. They typically have a digital object identifier, so they can be cited. The benefits of doing so are typically to receive credit for one's ideas and findings without waiting for the often-long peer review process to conclude, to receive feedback from other experts in the field, and to increase visibility of one's research.[83] They can also be a great place to deposit null results that may not be accepted by journals due to publication bias. In this way, they can correct the denominator of studies and hypotheses on a topic, which can be particularly useful for evidence syntheses. There are several places to publish preprints, and an entire director of preprint servers is available here. Some examples of preprint servers include SSRN, ArXiv (pronounced 'archive'), Preprints.org, medRXiv, and many more.

Registered reports are a relatively newer publishing format that involve two stages of peer review: one on the methodology of the planned study before data are collection, and another after the data are collected. The idea is to eliminate publication bias by agreeing to publish work that is methodologically sound, regardless of statistically significant outcomes. Journals that allow this format review the methodology and analysis plan in the first stage, and can choose to accept it, reject it, and accept it conditional on revisions. If it accepted, then the journal grants *in-principle* acceptance to the authors, meaning that they will publish the study regardless of the outcome as long as the authors stick to the protocol they described. The basic workflow of a Registered Report is presented in Figure 16.8.

There are now over 200 published Registered Reports, up from just a handful in 2014.[85] They are not appropriate for every type of research, such as mainly exploratory work, but can be a valuable mechanism to mitigate researcher degrees of freedom and publication bias.

16.8 Conclusion

In sum, we have a lot of work left to do to clean up the research landscape. New advances are constantly being made, and an entire field of *Meta-Science* (science about science) or *Meta-Research* (research about research) has sprung up in the last decade or so to formally study many of the issues we have seen. Far from feeling hopeless, there are plenty of reasons to be excited about a better future for RT2!

Figure 16.8: Two-stage peer-review process to eliminate publication bias. *Source: Center for Open Science.*[84]

17 References

1. Grolemund G. *Hands-on Programming with R: Write Your Own Functions and Simulations.* " O'Reilly Media, Inc."; 2014.
2. Wickham H. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics.* 2010;19(1):3-28. doi:10.1198/jcgs.2009.07098
3. Desbarats N. I've Stopped Using Box Plots. Should You?, Nightingale. *Nightingale.* Published online November 2021. Accessed August 26, 2023. https://nightingaledvs.com/ive-stopped-using-box-plots-should-you/
4. National Academies of Sciences E. *Reproducibility and Replicability in Science.*; 2019. doi:10.17226/25303
5. Moravcsik A. *Transparency in Qualitative Research.* SAGE Publications Limited; 2020.
6. Steinle F. Stability and Replication of Experimental Results: A Historical Perspective. In: *Reproducibility : Principles, Problems, Practices, and Prospects.* John Wiley & Sons, Incorporated; 2016. http://ebookcentral.proquest.com/lib/upenn-ebooks/detail.action?docID=4547409
7. Glick TF, Livesey SJ, Wallis F. *Routledge Revivals: Medieval Science, Technology and Medicine (2006): An Encyclopedia.* Taylor & Francis; 2017.
8. Zampieri F, ElMaghawry M, Zanatta A, Thiene G. Andreas Vesalius: Celebrating 500 years of dissecting nature. *Global Cardiology Science & Practice.* 2015;2015(5):66. doi:10.5339/gcsp.2015.66
9. Simons DJ, Shoda Y, Lindsay DS. Constraints on Generality (COG): A Proposed Addition to All Empirical Papers. *Perspectives on Psychological Science.* 2017;12(6):1123-1128. doi:10.1177/1745691617708630
10. Cartwright N. Middle-range theory: Without it what could anyone do? *THEORIA An International Journal for Theory, History and Foundations of Science.* Published online September 2020. Accessed June 13, 2023. https://ojs.ehu.eus/index.php/THEORIA/article/view/21479
11. Merton RK. Science and technology in a democratic order, reprinted as The normative structure of science. Published online 1942. https://sciencepolicy.colorado.edu/students/envs_5110/merton_sociology_science.pdf
12. Huckvale K, Torous J, Larsen ME. Assessment of the Data Sharing and Privacy Practices of Smartphone Apps for Depression and Smoking Cessation. *JAMA Network Open.* 2019;2(4):e192542. doi:10.1001/jamanetworkopen.2019.2542

13. Saint-Exupéry A de. Le petit prince [The little prince]. *Verenigde State van Amerika: Reynal & Hitchkock (US), Gallimard (FR)*. Published online 1943.

14. Mitroff II. Norms and Counter-Norms in a Select Group of the Apollo Moon Scientists: A Case Study of the Ambivalence of Scientists. *American Sociological Review*. 1974;39(4):579-595. doi:10.2307/2094423

15. Cattau D. David Blackwell, 'Superstar'. *The University of Illinois Alumni Association – Illinois Alumni Magazine*. Published online 2010.

16. Brin S, Page L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks*. 1998;30:107-117. Accessed June 21, 2023. http://www-db.stanford.edu/~backrub/google.html

17. Zaveri B. Google's Revenue By Segment (2016-2023). *Business Quant*. Published online February 2020. Accessed June 21, 2023. https://businessquant.com/google-revenue-by-segment

18. Easterbrook PJ, Gopalan R, Berlin J, Matthews DR. Publication bias in clinical research. *The Lancet*. 1991;337(8746):867-872.

19. Sternberg RJ. "Am I Famous Yet?" Judging Scholarly Merit in Psychological Science: An Introduction. *Perspectives on Psychological Science: A Journal of the Association for Psychological Science*. 2016;11(6):877-881. doi:10.1177/1745691616661777

20. Flaherty C. Revolt Over an Editor. *Inside Higher Ed*. Published online 2018. Accessed June 21, 2023. https://www.insidehighered.com/news/2018/04/30/prominent-psychologist-resigns-journal-editor-over-allegations-over-self-citation

21. Rokeach M. The nature and meaning of dogmatism. Published online 1954.

22. Resnick B. Intellectual humility: The importance of knowing you might be wrong. *Vox*. Published online January 2019. Accessed June 21, 2023. https://www.vox.com/science-and-health/2019/1/4/17989224/intellectual-humility-explained-psychology-replication

23. Harrell F. Statistical Thinking - A Litany of Problems With p-values. Published online February 2017. Accessed June 27, 2023. https://www.fharrell.com/post/pval-litany/

24. Lakens D. The Practical Alternative to the p Value Is the Correctly Used p Value. *Perspectives on Psychological Science*. 2021;16(3):639-648. doi:10.1177/1745691620958012

25. Wasserstein RL, Lazar NA. The ASA Statement on p-Values: Context, Process, and Purpose. *The American Statistician*. 2016;70(2):129-133. doi:10.1080/00031305.2016.1154108

26. Wasserstein RL, Schirm AL, Lazar NA. Moving to a World Beyond " $p < 0.05$." *The American Statistician*. 2019;73(sup1):1-19. doi:10.1080/00031305.2019.1583913

27. Greenland S, Senn SJ, Rothman KJ, et al. Statistical tests, P values, confidence intervals, and power: A guide to misinterpretations. *European Journal of Epidemiology*. 2016;31(4):337-350. doi:10.1007/s10654-016-0149-3

28. Ioannidis JPA. Why Most Published Research Findings Are False. *PLOS Medicine*. 2005;2(8):e124. doi:10.1371/journal.pmed.0020124

29. Engber D. Daryl Bem Proved ESP Is Real. *Slate*. Published online June 2017. Accessed July 13, 2023. https://slate.com/health-and-science/2017/06/daryl-bem-proved-esp-is-real-showed-science-is-broken.html

30. Kekecs Z, Palfi B, Szaszi B, et al. Raising the value of research studies in psychological science by increasing the credibility of research reports: The transparent Psi project. *Royal Society Open Science*. 2023;10(2):191375. doi:10.1098/rsos.191375

31. Bem D, Tressoldi P, Rabeyron T, Duggan M. Feeling the future: A meta-analysis of 90 experiments on the anomalous anticipation of random future events. *F1000Research*. 2016;4:1188. doi:10.12688/f1000research.7177.2

32. Lakens D. The 20% Statistician: A pre-publication peer-review of the 'Feeling The Future' meta-analysis. *The 20% Statistician*. Published online May 2014. Accessed July 14, 2023. http://daniellakens.blogspot.com/2014/05/a-pre-publication-peer-review-of-meta.html

33. Simmons JP, Nelson LD, Simonsohn U. False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant. *Psychological Science*. 2011;22(11):1359-1366. doi:10.1177/0956797611417632

34. Bargh JA, Chen M, Burrows L. Automaticity of social behavior: Direct effects of trait construct and stereotype activation on action. *Journal of personality and social psychology*. 1996;71(2):230.

35. Doyen S, Klein O, Pichon CL, Cleeremans A. Behavioral Priming: It's All in the Mind, but Whose Mind? *PLOS ONE*. 2012;7(1):e29081. doi:10.1371/journal.pone.0029081

36. Schimmack U. Replicability Audit of John A. Bargh. *Replicability-Index*. Published online March 2019. Accessed July 14, 2023. https://replicationindex.com/2019/03/17/raudit-bargh/

37. Yong E. A failed replication draws a scathing personal attack from a psychology professor. *Science*. Published online March 2012. Accessed July 14, 2023. https://www.nationalgeographic.com/science/article/failed-replication-bargh-psychology-study-doyen

38. Schimmack U. A Meta-Scientific Perspective on "Thinking: Fast and Slow. *Replicability-Index*. Published online December 2020. Accessed July 18, 2023. https://replicationindex.com/2020/12/30/a-meta-scientific-perspective-on-thinking-fast-and-slow/

39. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*. 2015;349(6251):aac4716. doi:10.1126/science.aac4716

40. John LK, Loewenstein G, Prelec D. Measuring the Prevalence of Questionable Research Practices With Incentives for Truth Telling. *Psychological Science*. 2012;23(5):524-532. doi:10.1177/0956797611430953

41. Klein RA, Vianello M, Hasselman F, et al. Many Labs 2: Investigating Variation in Replicability Across Samples and Settings. *Advances in Methods and Practices in Psychological Science*. 2018;1(4):443-490. doi:10.1177/2515245918810225

42. Camerer CF, Dreber A, Holzmeister F, et al. Evaluating the replicability of social science experiments in Nature and Science between 2010 and 2015. *Nature Human Behaviour*. 2018;2(9):637-644. doi:10.1038/s41562-018-0399-z

43. Oransky I. Retractions are increasing, but not enough. *Nature*. 2022;608(7921):9-9. doi:10.1038/d41586-022-02071-6

44. Brainard J, You J. What a massive database of retracted papers reveals about science publishing's "death penalty." *Science*. 2018;25(1):1-5.

45. Retraction Watch. Top 10 most highly cited retracted papers. *Retraction Watch*. Published online December 2015. Accessed July 19, 2023. https://retractionwatch.com/the-retraction-watch-leaderboard/top-10-most-highly-cited-retracted-papers/

46. Kühberger A, Streit D, Scherndl T. Self-correction in science: The effect of retraction on the frequency of citations. *PLOS ONE*. 2022;17(12):e0277814. doi:10.1371/journal.pone.0277814

47. Bornemann-Cimenti H, Szilagyi IS, Sandner-Kiesling A. Perpetuation of Retracted Publications Using the Example of the Scott S. Reuben Case: Incidences, Reasons and Possible Improvements. *Science and Engineering Ethics*. 2016;22(4):1063-1072. doi:10.1007/s11948-015-9680-y

48. Gorski D. When fraud undermines science-based medicine. *Science-Based Medicine*. Published online March 2009. Accessed July 19, 2023. https://sciencebasedmedicine.org/when-fraud-undermines-science-based-medicine/

49. Munafò MR, Nosek BA, Bishop DVM, et al. A manifesto for reproducible science. *Nature Human Behaviour*. 2017;1(1):1-9. doi:10.1038/s41562-016-0021

50. Kerr NL. HARKing: Hypothesizing after the results are known. *Personality and social psychology review*. 1998;2(3):196-217.

51. Hollenbeck JR, Wright PM. Harking, sharking, and tharking: Making the case for post hoc analysis of scientific data. *Journal of Management*. 2017;43(1):5-18.

52. Lishner DA. HARKing: Conceptualizations, harms, and two fundamental remedies. *Journal of Theoretical and Philosophical Psychology*. 2021;41(4):248.

53. Møller AP, Jennions MD. Testing and adjusting for publication bias. *Trends in Ecology & Evolution*. 2001;16(10):580-586. doi:10.1016/S0169-5347(01)02235-2

54. Mlinarić A, Horvat M, Šupak Smolčić V. Dealing with the positive publication bias: Why you should really publish your negative results. *Biochemia Medica*. 2017;27(3):447-452. doi:10.11613/BM.2017.030201

55. Franco A, Malhotra N, Simonovits G. Publication bias in the social sciences: Unlocking the file drawer. *Science*. 2014;345(6203):1502-1505.

56. Thornton A, Lee P. Publication bias in meta-analysis: Its causes and consequences. *Journal of Clinical Epidemiology*. 2000;53(2):207-216. doi:10.1016/S0895-4356(99)00161-4

57. Porta M. *A Dictionary of Epidemiology*. Oxford university press; 2014.

58. Mertens S, Herberz M, Hahnel UJJ, Brosch T. The effectiveness of nudging: A meta-analysis of choice architecture interventions across behavioral domains. *Proceedings of the National Academy of Sciences*. 2022;119(1):e2107346118. doi:10.1073/pnas.2107346118

59. Maier M, Bartoš F, Stanley TD, Shanks DR, Harris AJL, Wagenmakers EJ. No evidence for nudging after adjusting for publication bias. *Proceedings of the National Academy of Sciences*. 2022;119(31):e2200300119. doi:10.1073/pnas.2200300119

60. Rosenthal R. The file drawer problem and tolerance for null results. *Psychological Bulletin*. 1979;86(3):638-641. doi:10.1037/0033-2909.86.3.638

61. Shi L, Lin L. The trim-and-fill method for publication bias: Practical guidelines and recommendations based on a large database of meta-analyses. *Medicine*. 2019;98(23):e15987. doi:10.1097/MD.0000000000015987

62. Maier M, Bartoš F, Wagenmakers EJ. Robust Bayesian meta-analysis: Addressing publication bias with model-averaging. *Psychological Methods*. 2023;28(1):107-122. doi:10.1037/met0000405

63. Rodgers MA, Pustejovsky JE. Evaluating meta-analytic methods to detect selective reporting in the presence of dependent effect sizes. *Psychological methods*. 2021;26(2):141.

64. Nickerson RS. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*. 1998;2(2):175-220.

65. Mackay C. *Memoirs of Extraordinary Popular Delusions and the Madness of Crowds*. George Routledge; sons; 1869.

66. Pusztai L, Hatzis C, Andre F. Reproducibility of research and preclinical validation: Problems and solutions. *Nature Reviews Clinical Oncology*. 2013;10(12):720-724.

67. Fischhoff B. Hindsight is not equal to foresight: The effect of outcome knowledge on judgment under uncertainty. *Journal of Experimental Psychology: Human perception and performance*. 1975;1(3):288.

68. Nosek BA, Lindsay DS. Preregistration becoming the norm in psychological science. *APS Observer*. 2018;31.

69. NASA Science. Unmasking the Face on Mars | Science Mission Directorate. *Unmasking the Face on Mars*. Published online May 2001. Accessed August 29, 2023. https://science.nasa.gov/science-news/science-at-nasa/2001/ast24may_1

70. Center for Open Science. Preregistration. Accessed August 29, 2023. https://www.cos.io/initiatives/prereg

71. Schwarzkopf S. It's not the end of the world if your research gets "scooped." *Times Higher Education (THE)*. Published online April 2016. Accessed August 29, 2023. https://www.timeshighereducation.com/blog/its-not-end-world-if-your-research-gets-scooped

72. Simmons JP, Nelson LD, Simonsohn U. A 21 word solution. *Available at SSRN 2160588*. Published online 2012.

73. Stojmenovska D, Bol T, Leopold T. Teaching replication to graduate students. *Teaching Sociology*. 2019;47(4):303-313.

74. Janz N. Bringing the gold standard into the classroom: Replication in university teaching. *International Studies Perspectives*. 2016;17(4):392-407.

75. Hawkins RXD, Smith EN, Au C, et al. Improving the Replicability of Psychological Science Through Pedagogy. *Advances in Methods and Practices in Psychological Science*. 2018;1(1):7-18. doi:10.1177/2515245917740427

76. Center for Open Science. Open Science Badges. Accessed August 29, 2023. https://www.cos.io/initiatives/badges

77. American Psychological Association. Open Science Badges. *https://wwwapaorg*. Published online 2023. Accessed August 29, 2023. https://www.apa.org/pubs/journals/resources/open-science-badges

78. Kidwell MC, Lazarević LB, Baranski E, et al. Badges to acknowledge open practices: A simple, low-cost, effective method for increasing transparency. *PLoS biology*. 2016;14(5):e1002456.

79. Rowhani-Farid A, Aldcroft A, Barnett AG. Did awarding badges increase data sharing in BMJ Open? A randomized controlled trial. *Royal Society Open Science*. 2020;7(3):191818. doi:10.1098/rsos.191818

80. Rowhani-Farid A, Barnett AG. *Badges for Sharing Data and Code at* Biostatistics: *An Observational Study*. F1000Research; 2018. doi:10.12688/f1000research.13477.2

81. European Science Foundation. Principles and Implementation | Plan S. Published online 2023. Accessed August 29, 2023. https://www.coalition-s.org/addendum-to-the-coalition-s-guidance-on-the-implementation-of-plan-s/principles-and-implementation/

82. Nosek BA, Alter G, Banks GC, et al. Promoting an open research culture. *Science*. 2015;348(6242):1422-1425. doi:10.1126/science.aab2374

83. Mudrak B. What Are Preprints, and How Do They Benefit Authors? | AJE. Published online 2018. Accessed August 29, 2023. https://www.aje.com/arc/benefits-of-preprints-for-researchers/

84. Center for Open Science. Registered Reports. Accessed August 29, 2023. https://www.cos.io/initiatives/registered-reports

85. Chambers C. What's next for Registered Reports? *Nature*. 2019;573(7773):187-189. doi:10.1038/d41586-019-02674-6

A Answers for Section 3.8

1. Calculate $89 + 9/(4 * 5)^2$ and assign the value to the object `solution1`. Then print `solution1`.

```
solution1 <- (89+9)/(4*5)^2
print(solution1)
```

2. What is the difference between R and R Studio?

Base R refers to the statistical programming language and application installed on your computer to process the R programming language. RStudio is an Integrated Development Environment (IDE) that integrates with R to provide much more functionality. You can use base R without RStudio, but not the other way around.

3. How do you add a comment to a Script file?

```
# Just add a hash/pound sign to the left.
#### You can add more hashes for aesthetic purposes ####

# Multi-line comments require a hash
# starting on the left of each line.
```

4. What are packages in R, and how do you install them?

R packages are user-written collections of functions, compiled code, and sample data. There are over 9000+ packages in R and counting. We use packages for specific things we want to do that we cannot accomplish with the functions in base R, or to do things easier or more efficiently than base R functions.

Most packages that have been vetted and checked are available on the Comprehensive R Archive Network (CRAN), which is the central R package repository." "In most cases, installing a package in R is accomplished with the following code `install.packages("name of package")`.

5. Install and load the package `rstudioapi`.

```
install.packages("rstudioapi")
library(rstudioapi)
```

6. How do you modify the appearance of R Studio?

Tools –> Global Options –> Appearance –> Editor Theme.

7. Assign the value of `365` to an object called `year`. Then, create another object called `months` and assign it the value `year * 0.032854884083862`.

```
year <- 365
month <- year * 0.032854884083862
```

8. Create three variables named `Cowboys`, `Giants`, and `Commanders` and assign them all the value `"Inferior Team"` using multiple assignment operators.

```
Cowboys <- Giants <- Commanders <- "Inferior Team"
```

B Answers for Section 4.7

1. Create a numeric vector called `vec1` comprising the elements `3, -12, 532, 0, -100, 55,` and `-42`. Then, find the median and lowest value in the vector.

```
vec1 <- c(3, -12, 532, 0, -100, 55, -42)

median(vec1)
```

```
[1] 0
```

```
min(vec1)
```

```
[1] -100
```

2. Create a new vector `vec2` which contains all the elements of `vec1` that are positive integers. Then create a new vector `vec3` that contains all the elements of `vec1` that are negative integers. Then create a new vector `vec4` which is the sum of `vec2` and `vec3`.

```
vec2 <- vec1[vec1 > 0]
vec3 <- vec1[vec1 < 0]
vec4 <- vec2 + vec3
```

3. Assign the built-in dataframe `mtcars` to an object named after your favorite animal. Then calculate and print the median of the variable `mpg`.

```
# I'm a big donkey guy, but you can put your favorite animal for the name of the object.
donkey <- mtcars

median(donkey$mpg)
```

```
[1] 19.2
```

4. Create a new variable and add to the object you created above (named after your favorite animal). This variable will take the variable Miles per Gallon `mpg` and convert it to Kilometers per Liter, which you will name `kpl`. This can be accomplished by taking `mpg` and dividing it by 2.352. Once you've created this new variable and added it to the object, calculate the median of `kpl`.

```
donkey$kpl <- donkey$mpg / 2.352
median(donkey$kpl)
```

```
[1] 8.163265
```

5. Assign the built-in dataframe `OrchardSprays` to an object name of your choice. Then, convert the variable `treatment` to an ordered factor variable, and change the existing names of factor levels from A:H to a list of sulphur levels from `Sulpher_8` to `Sulpher_1`. Then, print the new `treatment` variable.

```
library(dplyr)

# I'll assign the built-in OrchardSprays dataframe to an object called sprays.
sprays <- OrchardSprays

# Next, I'll convert the treatment variable to an ordered factor variable.

sprays$treatment <- factor(sprays$treatment,
                           ordered = TRUE)

summary(sprays$treatment)
```

```
A B C D E F G H
8 8 8 8 8 8 8 8
```

```
# Next, I'll replace the existing level names with new ones.
sprays$treatment <- recode(sprays$treatment,
                           A = "Sulphur_8",
                           B = "Sulphur_7",
                           C = "Sulphur_6",
                           D = "Sulphur_5",
                           E = "Sulphur_4",
                           F = "Sulphur_3",
                           G = "Sulphur_2",
```

```
                          H = "Sulphur_1")

print(sprays$treatment)
```

```
 [1] Sulphur_5 Sulphur_4 Sulphur_7 Sulphur_1 Sulphur_2 Sulphur_3 Sulphur_6
 [8] Sulphur_8 Sulphur_6 Sulphur_7 Sulphur_1 Sulphur_5 Sulphur_4 Sulphur_8
[15] Sulphur_3 Sulphur_2 Sulphur_3 Sulphur_1 Sulphur_8 Sulphur_4 Sulphur_5
[22] Sulphur_6 Sulphur_2 Sulphur_7 Sulphur_1 Sulphur_8 Sulphur_4 Sulphur_6
[29] Sulphur_3 Sulphur_2 Sulphur_7 Sulphur_5 Sulphur_4 Sulphur_5 Sulphur_2
[36] Sulphur_8 Sulphur_6 Sulphur_7 Sulphur_1 Sulphur_3 Sulphur_8 Sulphur_6
[43] Sulphur_3 Sulphur_2 Sulphur_7 Sulphur_5 Sulphur_4 Sulphur_1 Sulphur_7
[50] Sulphur_2 Sulphur_6 Sulphur_3 Sulphur_8 Sulphur_1 Sulphur_5 Sulphur_4
[57] Sulphur_2 Sulphur_3 Sulphur_5 Sulphur_7 Sulphur_1 Sulphur_4 Sulphur_8
[64] Sulphur_6
8 Levels: Sulphur_8 < Sulphur_7 < Sulphur_6 < Sulphur_5 < ... < Sulphur_1
```

C Answers for Section 5.3

1. Create a variable called `MarySue1` with the value `"Dr Mary Sue Coleman, former president of the University of Michigan once said"`. Then, create another variable called `MarySue2` with the value `"For today, goodbye. For tomorrow, good luck. And Forever, Go Blue!"`. Then find the number of characters in each variable using the `nchar()` function. Then, check if the letter 'r' is present in each variable, and report the results.

```
MarySue1 <- "Dr Mary Sue Coleman, former president of the University of Michigan once said

MarySue2 <- "For today, goodbye. For tomorrow, good luck. And Forever, Go Blue!"

nchar(MarySue1)
```

```
[1] 77
```

```
nchar(MarySue2)
```

```
[1] 66
```

```
# There are 77 characters in MarySue1 and 66 characters in MarySue2.


grepl('r', MarySue1)
```

```
[1] TRUE
```

```
grepl('r', MarySue2)
```

```
[1] TRUE
```

```
# The letter r is present in both variables MarySue1 and MarySue2.
```

2. Create a variable called `MarySue3` whose value is a concatenation (combination) of `MarySue1` and `MarySue2`. Then, print the value for `MarySue3`.

```
MarySue3 <- paste(MarySue1, MarySue2)
print(MarySue3)
```

```
[1] "Dr Mary Sue Coleman, former president of the University of Michigan once said For today
```

3. Create a string vector called `basho` and assign it the value `"An old silent pond. A frog jumps into the pond-Splash! Silence again."` Then create and print another variable called `basho2` in which the word 'frog' has been replaced by 'buffalo', and the word 'Splash!' has been replaced by 'Yikes!'.

```
basho <- "An old silent pond. A frog jumps into the pond-Splash! Silence again."

basho2 <- basho

gsub('frog', 'buffalo', basho2)
```

```
[1] "An old silent pond. A buffalo jumps into the pond-Splash! Silence again."
```

```
gsub('Splash!', 'Yikes!', basho2)
```

```
[1] "An old silent pond. A frog jumps into the pond-Yikes! Silence again."
```

```
print(basho2)
```

```
[1] "An old silent pond. A frog jumps into the pond-Splash! Silence again."
```

4. Let's do a variation of Mad Libs I will call Mad Sentences. Install and load the `keyToEnglish` package (be mindful of the capitalization in this package's name). Then create three variables named after your three favorite cuisines. For each variable, assign the value `generate_random_sentences(n = 2, punctuate = TRUE)` to generate two random sentences per variable. This will produce a total of six sentences (two per variable). Finally, create a variable called `madsentences` whose value combines (pastes) all three variables. Print `madsentences`. If it sounds nonsensical, then it worked!

```
library(keyToEnglish)

indian <- generate_random_sentences(n = 2, punctuate = TRUE)
chinese <- generate_random_sentences(n = 2, punctuate = TRUE)
korean <- generate_random_sentences(n = 2, punctuate = TRUE)

madsentence <- paste(indian, chinese, korean)
print(madsentence)
```

```
[1] "Dependable synthetic sewer collects osmium epees. Powerful filthy alletrope ransacks gr
[2] "Potable coal snack boils dotted vents. Metaphysical grainy biophysicist slaughters sued
```

5. Let's compare the returns from simple vs compound interest after five years. First, define `p` as `1000`, `r` as `.07` and `t` as 5. Then Create a variable called `simple` with the value `p * r * t`. Next, create a variable `compound` with the value `p * (1 + r)^t - p`. Then, perform a logical test to see if `simple` is equal to `compound`, and write out the results of the test in one sentence.

```
p <- 1000
r <- 7 / 100
t <- 5

simple <- p * r * t

compound <- p * (1 + r)^t - p

simple == compound
```

```
[1] FALSE
```

```
# Simple interest is not equal to compound interest in five years at a principal of 1000 (
```

6. Retain the variables you created above and write a series of conditional (If else/Else If) statements according to the following rules: 1) If `simple` is less than `compound`, print the statement "Simple interest is less than compound interest."; 2) If `simple` is greater `compound`, print the statement "Simple interest is greater than compound interest."; 3) If `simple` is equal to `compound`, print the statement "Simple interest is equal to compound interest."

```
if (simple < compound) {
  print("Simple interest is less than compound interest")
} else if (simple > compound) {
  print("Simple interest is greater than compound interest")
} else {                            # could also be else if (simple == compound)
  print("Simple interest is equal to compound interest")
}
```

```
[1] "Simple interest is less than compound interest"
```

D Answers for Section 7.5

The following are answers to the exercises in Section 7.5.

1. Assign the built-in dataframe `Orange` to an object named whatever you want. This dataframe relates to the age and circumference of orange trees. First, with respect to the specific tree, is the dataframe in long or wide format? Please explain your answer.

```
tree <- Orange

head(tree)
```

```
  Tree  age circumference
1    1  118            30
2    1  484            58
3    1  664            87
4    1 1004           115
5    1 1231           120
6    1 1372           142
```

```
# The dataframe appears to be in long format, as each tree has multiple rows.
```

2. Once again with respect to individual trees, reshape the dataframe. If you believe it is in long format, reshape to wide (hint: this is the correct answer). Assign the reshaped dataframe to a new object with a name that indicates the reshaped nature of the data (e.g., "tree_wide"). What you want to see is a dataframe where each row corresponds to age, and separate columns listing the circumference of each tree. Use an approach you have seen to name each column "Tree (number) circumference". For example, the first tree column should be named "Tree 1 circumference", the second should be called "Tree 2 circumference", and so on. Once you have finished, use the `head()` function to show the first five rows of the reshaped dataframe.

```
library(tidyverse)

tree_wide <- tree %>%
   pivot_wider(
```

```
    names_from = Tree,
    values_from = circumference,
    names_glue = "Tree {Tree} {.value}"
    )

head(tree_wide, n = 5)
```

```
# A tibble: 5 x 6
    age `Tree 1 circumference` `Tree 2 circumference` `Tree 3 circumference`
  <dbl>                  <dbl>                  <dbl>                  <dbl>
1   118                     30                     33                     30
2   484                     58                     69                     51
3   664                     87                    111                     75
4  1004                    115                    156                    108
5  1231                    120                    172                    115
# i 2 more variables: `Tree 4 circumference` <dbl>,
#   `Tree 5 circumference` <dbl>
```

3. Let's go backwards. Take the wide dataframe from Question 2, and reshape into long format, assigning this to a new object with a name noting that the new dataframe is in long format (e.g. "tree_long"). Here, we want each row to correspond to a tree, and separate columns for age and circumference, labelled as such. Then, use a command that you've seen before to make the new tree column come first in order in the dataframe. Finally, use the `head()` to show the first five rows of the new dataframe.

```
tree_long <- tree_wide %>%
  pivot_longer(
    cols = c("Tree 1 circumference",
             "Tree 2 circumference",
             "Tree 3 circumference",
             "Tree 4 circumference",
             "Tree 5 circumference"),
    names_to = "Tree",
    names_pattern = "[^Tree](.)",   ## This is basically saying take the former column lab
    values_to = "Circumference"
    ) %>%
  relocate(Tree, .before = everything()  ## This puts the Tree column first in order.
           )

head(tree_long, n = 5)
```

```
# A tibble: 5 x 3
```

```
  Tree    age Circumference
  <chr> <dbl>         <dbl>
1 1       118            30
2 2       118            33
3 3       118            30
4 4       118            32
5 5       118            30
```

4. Let's go back to the original `Orange` dataframe, or the object to which you initially assigned it. Next, we're going to create two new dataframes `tree2` and `tree3` using the code below. We're also going to create an ID variable for the two dataframes with the same rows in order to have unique indices for each row.

```
tree <- Orange

newage <- tree$circumference/2.5
treeid <- tree$Tree

tree2 <- data.frame(Tree = treeid,
                    "Age_years" = newage)


tree3 <- data.frame(Tree = c(rep(6, 7)),
                    "Age_years" = c(10.4,
                                    21.2,
                                    30.4,
                                    41.3,
                                    55.8,
                                    66.7,
                                    71.4),
                    ID = c(seq(from = 36, to = 42)
                           )
                    )
tree3$Tree <- ordered(tree3$Tree) ## This will make the Tree column in tree3 the same clas

## This creates an ID variable in the original dataframe and in tree2.
tree$ID <- 1:nrow(tree)
tree2$ID <- 1:nrow(tree2)
```

5. Create a new dataframe `tree4` that merges `tree2` with the `Orange` dataframe or whatever object you assigned it to. Note, what we want here is to add a new column `Age (Years)` to the same rows or observational units. The `ID` variable we created above should be

used to link the dataframes. Once the merge is complete, print the first six rows of the merged dataframe to make sure it worked.

```
tree4 <- tree %>%
  left_join(tree2,
            by = c("ID" = "ID")
            )

head(tree4, n = 6)
```

```
  Tree.x  age circumference ID Tree.y Age_years
1      1  118            30  1      1      12.0
2      1  484            58  2      1      23.2
3      1  664            87  3      1      34.8
4      1 1004           115  4      1      46.0
5      1 1231           120  5      1      48.0
6      1 1372           142  6      1      56.8
```

```
# While this is fine and links the data well, we do have duplicate Tree columns. Since the

tree4 <- cbind(tree, tree2$Age_years)

## Let's move ID to the front and clean up the Age_years variable name.

tree4 <- tree4 %>%
  relocate(ID, .before = everything()) %>%
  rename("Age (years)" = `tree2$Age_years`)

head(tree4, n = 6)
```

```
  ID Tree  age circumference Age (years)
1  1    1  118            30        12.0
2  2    1  484            58        23.2
3  3    1  664            87        34.8
4  4    1 1004           115        46.0
5  5    1 1231           120        48.0
6  6    1 1372           142        56.8
```

6. Now create a new dataframe `tree5` which appends rows from `tree3` to the `tree2` dataframe we created in Question 2. Once the merge is complete, print the *last* seven rows of the merged dataframe to make sure it worked.

```r
tree5 <- rbind(tree2, tree3)

tail(tree5, n = 7)
```

```
   Tree Age_years ID
36    6      10.4 36
37    6      21.2 37
38    6      30.4 38
39    6      41.3 39
40    6      55.8 40
41    6      66.7 41
42    6      71.4 42
```

E Answers for Section 8.8

The following are answers to the exercises in Section 8.8.

Let's create a dataframe with the following code:

```
indianfood <- data.frame(name = c("Boondi",
                                  "Gajar ka halwa",
                                  "Ghevar",
                                  "Kalakand",
                                  "Misti Doi",
                                  "Aloo tikki",
                                  "Chicken tikka masala"),
                         ingredients = c("Maida flour, yogurt, oil, sugar",
                                         "Carots, milk, sugar, ghee, cashews, raisins",
                                         "Flour, ghee, kewra, milk, clarified butter, suga
                                         "Milk, cottage cheese, sugar",
                                         "Milk, jaggery",
                                         "Rice flour, potatoe, bread crumbs, garam masalaa
                                         "Naan bread, tomato sauce, skinless chicken breas
                         prep_time = c(45,
                                       15,
                                       15,
                                       20,
                                       480,
                                       5,
                                       15020240),
                         state = c("West Bengal",
                                   "Punjab",
                                   "Rajasthan",
                                   "West Bengal",
                                   "West Bengal",
                                   "Punjab",
                                   "Punjab")
                         )
```

Next, let's fix some errors and make some changes!

1. First, let's create some more descriptive variable names. Change the existing variables names to *Name, Ingredients, Preparation Time (mins), and Origin (state).* Then, print the names of the dataframe.

```
newnames <- c("Name", "Ingredients", "Preparation Time (mins)", "Origin (state)")

names(indianfood) <- newnames
names(indianfood)
```

```
[1] "Name"                     "Ingredients"
[3] "Preparation Time (mins)" "Origin (state)"
```

2. Next, having read through the ingredients, we can see various typos that should be fixed. Go ahead and fix the spelling mistakes (typos) you see in the `Ingredients` column. *Hint: there are four spelling mistakes in the* `Ingredients` *column, and we are using standard American English for spelling.*

```
# I see typos in Rows 2, 3, and 6. In row 2, carrot is spelled incorrectly. In row 3, card

indianfood[2,2] <- "Carrots, milk, sugar, ghee, cashews, raisins"
indianfood[3,2] <- "Flour, ghee, kewra, milk, clarified butter, sugar, almonds, pistachio,
indianfood[6,2] <- "Rice flour, potato, bread crumbs, garam masala, salt"
```

3. I see that the ingredients list has the terms "cottage cheese", "clarified butter", and "naan bread." Substitute *paneer* for "cottage cheese" and remove "clarified butter" since there is already 'ghee' in the same row, which is pretty much the same thing. Also, remove 'bread' from "Naan bread", since that is unnecessary. Then print the `Ingredients` column.

```
indianfood$Ingredients <- gsub("cottage cheese", "paneer", indianfood$Ingredients)
indianfood$Ingredients <- gsub(" clarified butter,", " ", indianfood$Ingredients)
indianfood$Ingredients <- gsub("Naan bread", "Naan", indianfood$Ingredients)


print(indianfood$Ingredients)
```

```
[1] "Maida flour, yogurt, oil, sugar"
[2] "Carrots, milk, sugar, ghee, cashews, raisins"
[3] "Flour, ghee, kewra, milk,  sugar, almonds, pistachio, saffron, green cardamom"
[4] "Milk, paneer, sugar"
[5] "Milk, jaggery"
```

```
[6] "Rice flour, potato, bread crumbs, garam masala, salt"
[7] "Naan, tomato sauce, skinless chicken breasts, heavy cream, garam masala"
```

4. Let's check the class of the `Origin (state)` variable. If it is not already in factor class, go ahead and convert it to factor. Make sure to save the new factor-classed variable back in the dataset. Then, check the class of the variable again to make sure it is factor.

```
# Check the class of the Origin (state) variable.
class(indianfood$`Origin (state)`)
```

```
[1] "character"
```

```
# It's currently in character class, so let's make it factor.
indianfood$`Origin (state)` <- as.factor(indianfood$`Origin (state)`)

# Let's check the new class.
class(indianfood$`Origin (state)`)
```

```
[1] "factor"
```

5. Next, let's add some variable labels to each variable. The variable labels can say whatever you think might be helpful. In general, think of some description that will provide context if you are looking at these data for the first time. Then, do something to check if it worked. *Hint: you've seen two ways to do this.*

```
library(expss)

indianfood <- apply_labels(indianfood,
                           Name = "Name of the Indian dish",
                           Ingredients = "Required components to make the Indian dish",
                           "Preparation Time (mins)" = "The time in minutes it takes to pr
                           "Origin (state)" = "The provenance of the Indian dish by state.

str(indianfood) # you can also View(indianfood) to see the variable labels under the varia
```

```
'data.frame':   7 obs. of  4 variables:
 $ Name                   :Class 'labelled' chr  "Boondi" "Gajar ka halwa" "Ghevar" "Kalakand
   .. .. LABEL: Name of the Indian dish
 $ Ingredients            :Class 'labelled' chr  "Maida flour, yogurt, oil, sugar" "Carrots,
   .. .. LABEL: Required components to make the Indian dish
```

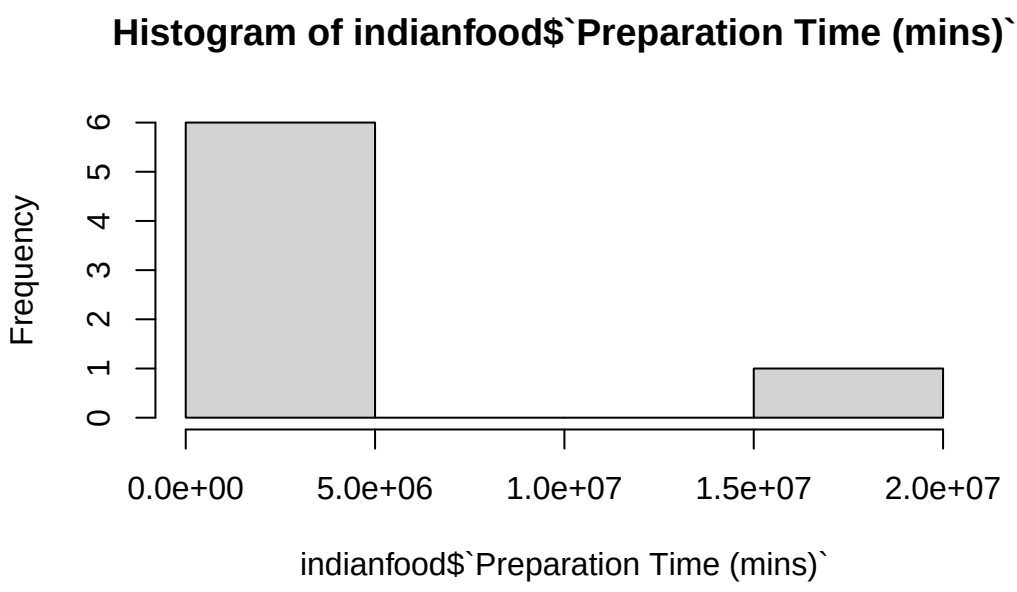
```
$ Preparation Time (mins):Class 'labelled' num  45 15 15 20 480 ...
  .. .. LABEL: The time in minutes it takes to prepare to cook the Indian dish
$ Origin (state)         :Class 'labelled' Factor w/ 3 levels "Punjab","Rajasthan",..: 3 1 1
  .. .. LABEL: The provenance of the Indian dish by state.
```

6. Finally, let's have a look at the `Preparation Time (mins)` variable to detect any outliers. Print the range of values and produce a histogram of this variable. Are any values seemingly outliers from the histogram?

```
# Look at the range of the values.
range(indianfood$`Preparation Time (mins)`)
```

```
[1]        5 15020240
```

```
# Alright, the highest value is definitely suspect. Let's look at the histogram.
hist(indianfood$`Preparation Time (mins)`)
```




```
# The value 15020240 definitely seems like an outlier because it is SO far away from the o
```

7. If you detected an outlier in the previous question (and I hope that you did!), explain your decision as to remove it or retain it in the dataframe. If you choose to remove the

outlier value, replace it with a more plausible value derived from a quick Google search. Then, print the variable.

```
# Though you don't need to do this since we already see that 15020240 is way beyond the IQ

library(rstatix)
library(dplyr)

indianfoodoutliers <- indianfood %>%
  identify_outliers(`Preparation Time (mins)`)

# If you want to see the actual row with the outlier, you can run View(indianfoodoutliers)

# Does a value of 15020240 minutes make sense as the preparation time for chicken tikka ma
15020240 / 60
```

```
[1] 250337.3
```

```
## Ok, so it equates to 250,337.3 hours. This seems obviously false. Just to be sure, I Go

# Find the row number corresponding to the outlying value.
which(indianfood$`Preparation Time (mins)` == 15020240)
```

```
[1] 7
```

```
# Now, change the value for Preparation Time (mins) in row 7 to 20, and check that it work
indianfood[7, 3] <- 20

print(indianfood$`Preparation Time (mins)`)
```

```
LABEL: The time in minutes it takes to prepare to cook the Indian dish
VALUES:
45, 15, 15, 20, 480, 5, 20
```

8. Finally, save your cleaned **indianfood** dataframe as an R data file as well as a CSV file using the file name of your choice.

```
# Save as an R Data file
save(indianfood, file = "indianfood.Rdata")
```

```r
# Save as a CSV file
write.csv(indianfood, "indianfood.csv", row.names = FALSE)
```

F Answers for Section 9.5

The following are answers to the exercises in Section 9.5.

1. Load the `tidyverse` package. Then assign the built-in dataframe `starwars` to an object named whatever you want. Then subset the dataframe by human species only. Save the subsetted dataframe as an object called `swhuman`. Then calculate and report the mean and median height in `swhuman`. Also report an NAs (missing data) in the height variable. *Note: the units for the height variable are centimeters.*

```
library(tidyverse)

stardf <- starwars

swhuman <- stardf %>%
  filter(species == "Human")

swhuman %>%
  select(height) %>%
  summary()
```

```
    height
Min.   :150.0
1st Qu.:168.5
Median :180.0
Mean   :176.6
3rd Qu.:184.0
Max.   :202.0
NA's   :4
```

```
# The mean height is 176.6 centimeters, and the median height is 180 centimeters.There are
```

2. Hopefully you noticed that there are indeed some NAs in the `swhuman` dataframe! Detect which rows have NAs for the height variable, and write the names of the characters that have this. Next, let's fix these errors. Perform an internet search and populate those NAs with plausible values. If you need to convert from feet to centimeters, multiply the

value in feet by 30.48. If you absolutely cannot find the height of any character substitute the median height from Question 1 for their height.

```
# Which rows have NA for the height variable?

which(is.na(swhuman$height))
```

```
[1] 18 32 33 34
```

```
swhuman %>%
  slice(18,
        32,
        33,
        34) %>%
  print()
```

```
# A tibble: 4 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Arvel Cr~     NA    NA brown      fair       brown             NA male  mascu~
2 Finn          NA    NA black      dark       dark              NA male  mascu~
3 Rey           NA    NA brown      light      hazel             NA fema~ femin~
4 Poe Dame~     NA    NA brown      light      brown             NA male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
## It looks like Arvel Crynyd, Finn, Rey, and Poe Dameron all have NA values for height. F

swhuman$height[18] <- 180

swhuman$height[32:34] <- c(176.8, 170.7, 172)
```

3. Once you have filled in this missing data, calculate the new mean and median for the height variable. Comment on how much of a difference the additional values made on the mean and median compared with the values you calculated in Question 1. Then determine the three shortest characters, and three tallest characters.

```
summary(swhuman$height)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  150.0   170.0   178.0   176.4   183.0   202.0
```

```
# The new mean is 176.4. This is 0.2 centimeters less than the mean from Question 1.
# The new median is 178. This is 2 centimeters less than the median from Question 1.

# Next, let's print the three shortest and three tallest characters.

swhuman %>%
  slice_min(height, n = 3)
```

```
# A tibble: 3 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <dbl> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Leia Org~    150    49 brown      light      brown             19 fema~ femin~
2 Mon Moth~    150    NA auburn     fair       blue              48 fema~ femin~
3 Cordé        157    NA brown      light      brown             NA fema~ femin~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
swhuman %>%
  slice_max(height, n = 3)
```

```
# A tibble: 3 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <dbl> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
1 Darth Va~    202   136 none       white      yellow          41.9 male  mascu~
2 Qui-Gon ~    193    89 brown      fair       blue            92   male  mascu~
3 Dooku        193    80 white      fair       brown          102   male  mascu~
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```
## The three shortest characters are Leia Organa, Mon Mothma, and Cordé.
## The three tallest characters are Darth Vader, Qui-Gon Jinn, and Dooku.
```

4. Return to the larger **starwars** dataframe or whatever object to which you assigned it. Determine which characters have NA for height. If there are any characters with NA for height (hint: there are), enter plausible values for their heights using the approach taken in Question 2. Then report the mean and median height across everyone in this dataframe.

```r
# Both approaches tell you the rows with NA in height. Use the approach you like.

which(is.na(stardf$height))
```

```
[1] 28 82 83 84 85 86
```

```r
stardf$height %>%
  is.na() %>%
  which()
```

```
[1] 28 82 83 84 85 86
```

```r
# Determine which characters have NA for height.

stardf %>%
  slice(28,
        82:86) %>%
  print()
```

```
# A tibble: 6 x 14
  name      height  mass hair_color skin_color eye_color birth_year sex   gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>           <dbl> <chr> <chr>
1 Arvel Cr~     NA    NA brown      fair       brown              NA male  mascu~
2 Finn          NA    NA black      dark       dark               NA male  mascu~
3 Rey           NA    NA brown      light      hazel              NA fema~ femin~
4 Poe Dame~     NA    NA brown      light      brown              NA male  mascu~
5 BB8           NA    NA none       none       black              NA none  mascu~
6 Captain ~     NA    NA unknown    unknown    unknown            NA <NA>  <NA>
# i 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

```r
# I already have heights for the first four. BB8's height is 67.1, and Captain Phasma's he

stardf$height[28] <- 180

stardf$height[82:86] <- c(176.8, 170.7, 172, 67.1, 200.1)

# Now calculate mean and median height.
```

```r
summary(stardf$height)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   66.0   167.0   180.0   173.4   191.0   264.0
```

```r
## The mean height is 173.4 cm and the median height is 180 cm.
```

5. Still working with the `starwars` dataframe, convert the `species` variable to factor. Then, group and summarise the mean height by species, and print this in descending order. Report which species is the tallest, on average. Then, rearrange and report the species which is the shortest, on average.

```r
# Convert species to factor

stardf$species <- as.factor(stardf$species)

# Which is the tallest species, on average?

stardf %>%
  group_by(species) %>%
  summarise(mean_height = mean(height)) %>%
  arrange(desc(mean_height))
```

```
# A tibble: 38 x 2
   species  mean_height
   <fct>          <dbl>
 1 Quermian        264
 2 Wookiee         231
 3 Kaminoan        221
 4 Kaleesh         216
 5 Gungan          209.
 6 Pau'an          206
 7 Besalisk        198
 8 Cerean          198
 9 Chagrian        196
10 Nautolan        196
# i 28 more rows
```

```r
## The Quermian species is the tallest on average.

# Which is the shortest species on average?

stardf %>%
  group_by(species) %>%
  summarise(mean_height = mean(height)) %>%
  arrange(mean_height)
```

```
# A tibble: 38 x 2
   species        mean_height
   <fct>                <dbl>
 1 Yoda's species        66
 2 Aleena                79
 3 Ewok                  88
 4 Vulptereen            94
 5 Dug                  112
 6 Droid                121.
 7 Xexto                122
 8 Toydarian            137
 9 Sullustan            160
10 Toong                163
# i 28 more rows
```

```r
## Yoda's species is the shortest on average.
```

G Kahneman's Open Letter to Priming Researchers

G.1 Background

This letter has been archived by Nature, and can be currently found here. If the link stops working, please email me and I'll fix it.

As mentioned in Chapter 14 , this letter was sent by psychologist Daniel Kahneman to psychologist John Bargh and others in the field of priming research. The letter was sent at a time where the entire field of not only priming, but psychology in general came under intense scrutiny in the form of a *Replication Crisis*, which since encouraged other fields to deal with their own crises of reproducibility, transparency, and replication. Today, the STEM fields as well are dealing with their own similar crises, particularly around fabrication or manipulation of images, which have become easier to detect with image recognition software and keen scientific integrity consultants like the formidable Dr Elisabeth Bik.

The letter sent by Kahneman is instructive and prescient. He correctly predicted the train wreck that would become the *Replication Crisis*, and while his daisy chain proposal may not have been realized among labs studying priming, it was reflected in similar large replication efforts such as Many Labs and Many Economists that continue to this day. Below, you will find the letter in full.

G.2 The Letter in Full

From: Daniel Kahneman

Sent: Wednesday, September 26, 2012 9:32 AM

Subject: A proposal to deal with questions about priming effects

Dear colleagues,

I write this letter to a collection of people who were described to me (mostly by John Bargh) as students of social priming. There were names on the list that I could not match to an email. Please pass it on to anyone else you think might be relevant.

As all of you know, of course, questions have been raised about the robustness of priming results. The storm of doubts is fed by several sources, including the recent exposure of fraudulent researchers, general concerns with replicability that affect many disciplines, multiple reported failures to replicate salient results in the priming literature, and the growing belief in the existence of a pervasive file drawer problem that undermines two methodological pillars of your field: the preference for conceptual over literal replication and the use of meta-analysis. Objective observers will point out that the problem could well be more severe in your field than in other branches of experimental psychology, because every priming study involves the invention of a new experimental situation.

For all these reasons, right or wrong, your field is now the poster child for doubts about the integrity of psychological research. Your problem is not with the few people who have actively challenged the validity of some priming results. It is with the much larger population of colleagues who in the past accepted your surprising results as facts when they were published. These people have now attached a question mark to the field, and it is your responsibility to remove it.

I am not a member of your community, and all I have personally at stake is that I recently wrote a book that emphasizes priming research as a new approach to the study of associative memory – the core of what dual-system theorists call System 1. Count me as a general believer. I also believe in a point that John Bargh made in his response to Cleeremans, that priming effects are subtle and that their design requires high-level skills. I am skeptical about replications by investigators new to priming research, who may not be attuned to the subtlety of the conditions under which priming effects are observed, or to the ease with which these effects can be undermined.

My reason for writing this letter is that I see a train wreck looming. I expect the first victims to be young people on the job market. Being associated with a controversial and suspicious field will put them at a severe disadvantage in the competition for positions. Because of the high visibility of the issue, you may already expect the coming crop of graduates to encounter problems. Another reason for writing is that I am old enough to remember two fields that went into a prolonged eclipse after similar outsider attacks on the replicability of findings: subliminal perception and dissonance reduction.

I believe that you should collectively do something about this mess. To deal effectively with the doubts you should acknowledge their existence and confront them straight on, because a posture of defiant denial is self-defeating. Specifically, I believe that you should have an association, with a board that might include prominent social psychologists from other field. The first mission of the board would be to organize an effort to examine the replicability of priming results, following a protocol that avoids the questions that have been raised and guarantees credibility among colleagues outside the field.

The following is just an example of such a protocol:

- Assemble a group of five labs, where the leading investigators have an established reputation (tenure should perhaps be a requirement). Substantial labs with several students are the most desirable participants.
- Each lab selects a recent demonstration of a priming effect, which they consider robust and most likely to replicate.
- The board makes a public commitment to these five specific effects
- Set up a daisy chain of labs A-B-C-D-E-A, where each lab will replicate the study selected by its neighbor: B replicates A, C replicates B etc.
- Have the replicating lab send someone to see how subjects are run (hence the emphasis on recency – the experiments should be in the active repertoire of the original lab, so that additional subjects can be run with confidence that the same procedure is followed).
- Have the replicated lab send someone to vet the procedure of the replicating lab as it starts its work
- Run enough subjects to guarantee power (probably more than in the original study)
- Use technology (e.g. video) to ensure that every detail of the method is documented and can be copied by others.
- Pre-commit to publish the results, letting the chips fall where they may, and make all data available for analysis by others.

This is something you could do quickly, and relatively cheaply. The main costs are 10 trips, and funds to cover these costs would be easy to get (I have checked). You would have to be careful in selecting laboratories and results to maximize credibility, and every step of the procedure should be open and documented. The unusually high openness to scrutiny may be annoying and even offensive, but it is a small price to pay for the big prize of restored credibility.

Success (say, replication of four of the five positive priming results) would immediately rehabilitate the field. Importantly, success would also provide an effective challenge to the adequacy of outsiders' replications. A publicly announced and open effort would be credible among colleagues at large, because it would show that you are sufficiently confident in your results to take a risk.

More ambiguous results would be painful, of course, but they would still protect the reputations of scholars who sincerely believe in their work – even if they are sometimes wrong.

The protocol I outlined is just an example of something you might do. The main point of my letter is that you should do something, and that you must do it collectively. No single individual will be able to overcome the doubts, but if you act as a group and avoid defensiveness you will be credible.

All best,