BADM 371 Intro to Data Analytics

BADM 371

2022-02-24

Contents

Chapter 1

Introduction

This is a book written which contains all the materials and lessons for **BADM 371 Intro to Data Analytics**. If you miss a day, want to review something we covered in class, or for any reason want to look for a worksheet, this is where you can go.

Each chapter contains a different topic we will cover during the semester. Some larger topics are split into two chapters to make accessing the materials a little more intuitive.

This book is updated automatically with any changes made to the documents during the semester, so if at any point you are told there was a change in the assignment, you can come here to get the updated version.

Also, this book has benefited greatly from lots of free, readily available resources posted on the web and we leverage these extensively. I would encourage you to review these resources in your analytics journey. Some that we specifically use with great frequency are these (and I say loud **THANK YOU** to the authors!):

- R for Data Science
- An Introduction to Statistical Learning with Applications in R
- R: A self-learn tutorial
- Data Science in a Box
- stackoverflow.com, for example

Chapter 2

Syllabus

Instructor: Tobin Turner

Office Hours: mutually convenient time arranged by email e-mail: jtturner@presby.edu

2.1 Course Objectives and Learning Outcomes

This course is designed to introduce to data science. Students will apply statistical knowledge and techniques to both business and non-business contexts.

At the end of this course students should be able to:

- Demonstrate mastery of the statistical software in R and the Rstudio IDE.
- Data wrangle (the process of cleaning and unifying messy and complex data sets for easy access and analysis)
- Demonstrate mastery of single and multiple regression.
- Demonstrate mastery of these dplyr functions: filter, select, mutate, group_by, summaize, and tally.
- Demonstrate mastery of how business analytics is related to other business functions and is important to the success of the business entity.

This course will be focused on both understanding and applying key business analytical concepts. Although the text serves as a useful foundation for the concepts covered in the class, simple memorization of the material in the text will not be sufficient. Class participation, discussion, and application are critical.

2.2 Text and Resources

- The course website (primary reource)
- An Introduction to Statistical Learning with Applications in R; by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
- R: A self-learn tutorial.
- Other free, publicly available datasets and publications.

2.3 Performance Evaluation (Grading)

- Quizzes and Assignments - 20%
- Exam 1 - 20%
- Exam 2 - 20%
- Exam 3 - 20%
- Final Exam - 20%

2.3.1 Exams

Exams will cover assigned chapters in the textbook, other assigned readings, lectures, class exercises, class discussions, videos, and guest speakers. I will typically allocate time prior to each exam to clearly identify the body of knowledge each test will cover and to answer questions about the format and objectives of the exam.

2.3.2 Quizzes – DON"T MISS CLASS

- The average of all quizzes and assignments will comprise the Quizzes and Assignments - 20% portion of your final grade
- Quizzes are designed to prepare you for your exams and to ensure you stay up wiht the course material
- Missed quizzes cannot be made up later. Be present.

Quizzes rule. **LISTEN.**

2.3.3 Final Average

- Final Average Grade
  - 90-100 A
  - 88-89 B+
  - 82-87 B+

- 80-81 B-
- 78-79 C+
- 72-77 C+
- 70-71 C-
- 60-69 D
- 59 and below F

2.4 Class Participation:

I will frequently give readings or assignments for you to complete prior to the next class meeting. I expect you to fully engage the material: answer questions, pose questions, provide insightful observations. Keep in mind that quality is an important component in "participation." Periodic cold calls will take place. I will also put students in the "hot seat" on occasion. In these class sessions, I may select a random group of students to lead us in the discussion and debate. Because the selection of participants will not be announced until class begins, everyone will be expected to prepare for the discussion. Reading the assigned chapters and articles are the best way to prepare for the discussion. If you have concerns about being called on in class, please see me to discuss. The purpose of the "hot seat" is not to stress or embarrass students, but to encourage students to actively engage the material.

2.5 Phones

Phones are not allowed to be used in class without the instructor's prior consent. If you have a need of a phone during class please let me know before class. Unauthorized use of electronic devices may result in the lowering of the grade or dismissal from the class. **I mean this.**

The phone thing? I mean this.

2.6 Attendance

You are expected to be regular and punctual in your class attendance. Students are responsible for all the material missed and homework assignments made. If class is missed, notes/homework should be obtained from another student. If I am more than 15 minutes late, class is considered cancelled. No more than 4 absences are allowed during a semester. Exceeding the absence policy may result in receiving an F for the course. The professors roll is the official roll and students not present when roll is taken will be counted as absent. If a student must miss an exam, she or he must work out an agreeable time with the instructor to take the test prior to the exam being given. If a student misses a test due to an emergency, the student must inform the instructor as soon as is possible. In special cases, the instructor may allow the student to take a make-up exam.

2.7 Accommodations

Presbyterian College is committed to providing reasonable accommodations for all students with documented disabilities. If you are seeking academic accommodations under the Americans with Disabilities Act, you must register with the Academic Success Office, located on 5th Avenue (beside Campus Police). To receive these accommodations, please obtain the proper Accommodations Approval Form from that office, and then meet with me at the beginning of the semester to discuss how we may deliver your approved accommodations. I especially encourage you to meet with me well in advance of the actual accommodations being provided, as it may not be feasible to offer immediate accommodations without sufficient advance notice (such as in the case of tests). I can assure you that all discussions will remain confidential. Disability Services information is located at this link http://bit.ly/PCdisabilityservices

Additionally, it is the student's responsibility to give the instructor one week's notice prior to each instance where accommodation will be required.

2.8 Honor Code and Plagiarism:

All assignments/exams must be your own work. Any copying or use of unauthorized assistance will be treated as a violation of PC's Honor Code. If you are unsure of what resources are allowed, please ask. Please note that all text longer than 7 words taken from ANY other source must be placed in quotations and cited. Also, summarizing ANY other source must also be cited. Using ANY other source and showing work to be your own is a violation of plagiarism and the honor code.

2.9 First-Generation Version:

I am a Presby First+ Advocate. I am here to support our current first-generation students. At Presbyterian College, first-generation students are those in which neither parent nor legal guardian graduated from a four-year higher education institution with a bachelor's degree. If you are a first-generation college student, please contact me. For more information about support for first-generation college students on our campus visit our Presby First+ webpage.

2.10 Continuing Advocate Version

I am a Presby First+ Advocate. I am committed to supporting first-generation students at Presbyterian College. At Presbyterian College, first-generation students are those in which neither parent nor legal guardian graduated from a four-year higher education institution with a bachelor's degree. If you are a first-generation college student, please contact me anytime or visit me during my office hours. For more information about support for first-generation college students on our campus visit our Presby First+ webpage.

Chapter 3

Our Class Rhythm

Monday: Wrap up previous topic and introduce what you've pre-read about. Chat. Play. Work some examples. Make sure the topics applies to real-life.

Wednesday: Work more examples. Chat as needed. **Live our best lives. :)**.

Friday: Apply what we've learned – demonstrate your mastery (typically in the form of a quiz, lab, or assignment). Rinse. Repeat.

Chapter 4

End in Mind

Dana Simmons: “Can you predict which students will enroll at PC?”

Christina Miller: ??? Well, can you? ???

Chapter 5

Schedule

This is a tentative schedule, **BUT** I will do my very best to stick to it, so that you may plan accordingly!

Spring 2022

Date	Topic
Date	A1
Monday, January 10, 2022	R basics and install
Wednesday, January 12, 2022	R basics and workflows
Friday, January 14, 2022	QUIZ 1
Monday, January 17, 2022	MLK Holiday
Wednesday, January 19, 2022	Objects and Arithmetic
Friday, January 21, 2022	QUIZ
Monday, January 24, 2022	Summaries and Subscripting
Wednesday, January 26, 2022	Matrices and `mtcars`
Friday, January 28, 2022	QUIZ
Monday, January 31, 2022	Class
Wednesday, February 2, 2022	Class
Friday, February 4, 2022	QUIZ
Monday, February 7, 2022	Social Dilemma and Review
Wednesday, February 9, 2022	**EXAM 1**
Friday, February 11, 2022	Class
Monday, February 14, 2022	Class
Wednesday, February 16, 2022	Online Class
Friday, February 18, 2022	Online QUIZ
Monday, February 21, 2022	Class
Wednesday, February 23, 2022	Class

Date	Topic
Friday, February 25, 2022	QUIZ
Monday, February 28, 2022	Class
Wednesday, March 2, 2022	Class
Friday, March 4, 2022	QUIZ
Monday, March 7, 2022	**EXAM 2**
Wednesday, March 9, 2022	Online Class
Friday, March 11, 2022	Online Class
Monday, March 14, 2022	SPRING BREAK
Wednesday, March 16, 2022	SPRING BREAK
Friday, March 18, 2022	SPRING BREAK
Monday, March 21, 2022	Class
Wednesday, March 23, 2022	Class
Friday, March 25, 2022	QUIZ
Monday, March 28, 2022	ADVISING WEEK
Wednesday, March 30, 2022	ADVISING WEEK
Friday, April 1, 2022	QUIZ
Monday, April 4, 2022	Class
Wednesday, April 6, 2022	Class
Friday, April 8, 2022	QUIZ
Monday, April 11, 2022	Class
Wednesday, April 13, 2022	**EXAM 3**
Friday, April 15, 2022	Easter Holidays
Monday, April 18, 2022	Easter Holidays
Wednesday, April 20, 2022	Class
Friday, April 22, 2022	QUIZ
Monday, April 25, 2022	Class
Wednesday, April 27, 2022	QUIZ
Friday, April 29, 2022	LAST DAY
Monday, May 2, 2022	**Final Exam** 5:30 p.m. – E period

Chapter 6

Elephant in the room: R

There are a variety of different applications for R. Yes, the more obvious ones would be things such as machine learning, artificial intelligence, and data mining. However, the possibilities with this program are honestly limitless.

6.1 The bad news

> **"The bad news is whenever you're learning a new tool, for a long time you're going to suck. It's going to be very frustrating. But, the good news is that that is typical, it's something that happens to everyone, and it's only temporary ... [T]here is no way to go from knowing nothing about a subject to knowing something about a subject and being an expert in it without going through a period of great frustration."**

– Hadley Wickham

6.2 About R

R is a software language for carrying out complicated (and simple) statistical analyses. It includes routines for data summary and exploration, graphical presentation and data modelling.

Chapter 7

R vs. Excel

7.1 Both are useful

Data analytics are increasingly important components of decision-making in any business. Whether you're a part of a marketing team that needs to generate visuals to highlight industry trends, or you're looking to generate financial statements, you will need an analytics program to help you develop your reports and effectively communicate your findings.

Both R and Excel are excellent data analytics tools, but they each have distinct functionality.

> **Please make sure you can explain the distinct functions of both R and Excel! YOU WILL NEED TO KNOW THIS!**

Excel is a well-known software program included in the Microsoft Office Suite. Used to create spreadsheets, execute calculations, produce charts, and perform statistical analysis, Excel is used by many professionals across a variety of industries. PC's **BADM 299** prepares you well for using Excel.

R is a free, open-source programming language and software environment that's frequently used in big data analysis and statistical computing. R has many advanced functions and capabilities.

7.2 Differences Between R and Excel

When choosing between R and Excel, it's important to understand how either software can get you the results you need. Here are some key differences between R and Excel to help you decide which makes the most sense to use.

7.3 Ease of Use & Learning the Software

Most people have likely already learned at least a few basic tips in Microsoft Excel. That's one substantial benefit of using Excel—the initial learning curve is quite minimal, and most analysis can be done via point-and-click on the top panel. Once a user imports their data into the program, it's not very hard to make basic graphs and charts.

R is a programming language, however, meaning the initial learning curve is steeper. It will take most at least a few weeks to familiarize themselves with the interface and master the various functions. Luckily, using R can quickly become second-nature with practice.

7.4 Replicating Analysis

R, while less user-friendly with a more intimidating user interface, has the capability to reproduce analyses repeatedly and with very different datasets. This can be incredibly helpful for large projects with multiple data sets, as you'll keep everything consistent and clean, without having to rewrite the script each time.

Since Excel's user interface is point-and-click, you'll need to rely on memory and repetition frequently. You cannot import codes and scripts as you would with R, so you'll have to "reinvent the wheel" to perform the same analysis across different data sets. This is not detrimental if you are doing basic statistics, but it may become time-consuming with more complicated analyses.

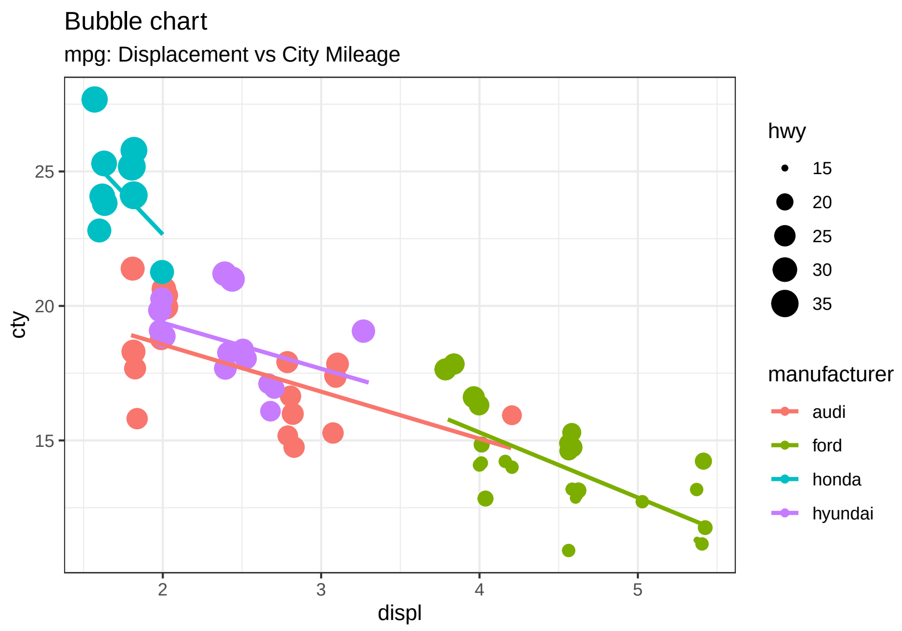
> For example, let's say you have thoroughly analyzed the analytics of 1 football season. How could R (vs Excel) help you quickly analyze a new season's data?

7.5 Visualization

When deciding between R and Excel, ask yourself, "How detailed do my visualizations need to be in order to achieve my goal(s)?" In Excel, for example, you can quickly highlight a group of cells and make a simple chart for PowerPoint. If you need a more comprehensive graph, however, R may be your best bet. R can produce incredibly attractive, detailed visuals that can help stakeholders understand your findings.

It all comes down to what you need your graphics to do. If you're just looking to cobble together a quick-and-dirty presentation to visualize data for your coworkers, then making simple straightforward charts in Excel will suffice. For those planning to publish large amounts of complicated data to various stakeholders, spending the time in R to create impressive interactive visual representations will likely be worth your while.

For example, here's and example of a pretty easy visualization in R that would be challenging to do (and update) in Excel.



7.6 Packages

In R, the fundamental unit of shareable code is the package. A package bundles together code, data, documentation, and tests, and is easy to share with others. As of June 2019, there were over 14,000 packages available on the Comprehensive R Archive Network, or CRAN, the public clearing house for R packages. This huge variety of packages is one of the reasons that R is so successful: the chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package.

But packages are useful even if you never share your code. As Hilary Parker says in her introduction to packages: "Seriously, it doesn't have to be about sharing your code (although that is an added benefit!). It is about saving yourself time." Organising code in a package makes your life easier because packages come with conventions. For example, you put R code in R/, you put tests in tests/ and you put data in data/. These conventions are helpful because:

- They save you time — you don't need to think about the best way to organise a project, you can just follow a template.
- Standardized conventions lead to standardized tools — if you buy into R's package conventions, you get many tools for free.

  We will chat more about packages, but for fun check out the links below…

7.7 Careers

Aptitude with Excel and R are incredibly valuable competencies that are in-demand across a variety of industries. Countless jobs are looking for applicants with at least some Excel experience (pivot tables look really good on a resumé), but R has a higher earning potential and is more in-demand than Excel.

R is one of the most popular programming languages and is an industry-standard for data analytics and data science. If you want to enter either field, there's a good chance you'll have a competitive advantage by knowing R. Entry-level jobs for those focusing on R also tend to make a high salary, frequently starting off earning more than $75,000.

Countless job listings also require Excel competency. From administrative assistants, marketers, academics, and more, everyone is expected to use Excel to some degree, whereas 10 to 15 years ago it was optional. Having a good background in Excel is still attractive on a resumé and will help to land a career with a high earning potential, but there are not many jobs looking for Excel skills alone.

7.8 Summary – Using R and Excel

R and Excel are beneficial in different ways. Excel starts off easier to learn and is frequently cited as the go-to program for reporting, thanks to its speed and efficiency. R is designed to handle larger data sets, to be reproducible, and to create more detailed visualizations. It's not a question of choosing between R and Excel, but deciding which program to use for different needs.

Chapter 8

R basics and workflows

8.1 Basics of working with R at the command line and RStudio goodies

Launch RStudio/R.

You will first intall R and then. RStudio.

- Installing R
- Installing RStudio
- Customizing RStudio
- RStudio Quick keys

8.2 In Rstudio - where we will live

Notice the default panes:

- Console (entire left)
- Environment/History (tabbed in upper right)
- Files/Plots/Packages/Help (tabbed in lower right)

8.3

Rstudio Console

FYI: you can change the default location of the panes, among many other things: Customizing RStudio.

Go into the Console, where we interact with the live R process.

You can make an object by assigning a value or statement to a letter or string. We use `<-` to assign objects meaning. Create and inspect the following object:

Your first analysis in R:

```
x <- 3 * 4
x
#> [1] 12
```

All R statements where you create objects – "assignments" – have this form:

```
objectName <- value
```

and in my head I hear, e.g., "x gets 12".

You will make lots of assignments and the operator `<-` is a pain to type. Don't be lazy and use `=`, although it would work, because it will just sow confusion later. Instead, utilize RStudio's keyboard shortcut: Alt + - (the minus sign).

I want to be your friend. As a friend, I implore you, learn this:

> In RStudio insert the <- assignment operator with Option + - (the minus sign)on a Mac, or Alt + - (the minus sign) on Windows.

Notice that RStudio automatically surrounds `<-` with spaces, which demonstrates a useful code formatting practice. Code is miserable to read on a good day. Give your eyes a break and use spaces.

RStudio offers many handy RStudio Quick keys. Also, Alt+Shift+K brings up a keyboard shortcut reference card.

Object names cannot start with a digit and cannot contain certain other characters such as a comma or a space. You will be wise to adopt a convention for demarcating words in names, but note that best practice is to choose ONE convention and stay true to it throughout your code.

```
i_use_snake_case
other.people.use.periods
evenOthersUseCamelCase
```

Make another assignment:

```
this_is_a_really_long_name <- 2.5
```

To inspect this, try out RStudio's completion facility: type the first few characters, press TAB, add characters until you disambiguate, then press return.

Make another assignment:

```
turner_rocks <- 2 ^ 3
```

When making assignments, it is best practice to keep the names brief, yet descriptive. For instance, while the name "this_is_a_really_long_name" is accurate, so is "long_name" and this is much more intuitive and easy to read/type over and over.

Let's try to inspect:

```
turnerrocks
#> Error in eval(expr, envir, enclos): object 'turnerrocks' not found
Turner_rocks
#> Error in eval(expr, envir, enclos): object 'Turner_rocks' not found
turner_rocks
#> [1] 8
```

Implicit contract with the computer / scripting language: Computer will do tedious computation for you. In return, you will be completely precise in your instructions. Typos matter. Case matters. Get better at typing.

R has a mind-blowing collection of built-in functions that are accessed like so:

```
functionName(arg1 = val1, arg2 = val2, and so on)
```

Let's try using `seq()` which makes regular sequences of numbers and, while we're at it, demo more helpful features of RStudio.

Type `se` and hit TAB. A pop up shows you possible completions. Specify `seq()` by typing more to disambiguate or using the up/down arrows to select. Notice the floating tool-tip-type help that pops up, reminding you of a function's arguments. If you want even more help, press F1 as directed to get the full documentation in the help tab of the lower right pane. Now open the parentheses and notice the automatic addition of the closing parenthesis and the placement of cursor in the middle. Type the arguments `1, 10` and hit return. RStudio also exits the parenthetical expression for you. IDEs are great.

```
seq(1, 10)
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

The above also demonstrates something about how R resolves function arguments. You can always specify in `name = value` form. But if you do not, R attempts to resolve by position. So above, it is assumed that we want a sequence `from = 1` that goes `to = 10`. Since we didn't specify step size, the default value of `by` in the function definition is used, which ends up being 1 in this case. For functions I call often, I might use this resolve by position for the first argument or maybe the first two. After that, I always use `name = value`.

Make this assignment and notice similar help with quotation marks.

```
yo <- "hello world"
```

If you just make an assignment, you don't get to see the value, so then you're tempted to immediately inspect.

```
y <- seq(1, 10)
y
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

This common action can be shortened by surrounding the assignment with parentheses, which causes assignment and "print to screen" to happen.

It is best practice to always attempt to "print" your assignments after creating them. This will help leviate the issue of searching 200+ lines of code for that one error causing argument.

```
(y <- seq(1, 10))
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

Not all functions have (or require) arguments:

```
date()
#> [1] "Thu Feb 24 08:59:39 2022"
```

Now look at your workspace – in the upper right pane. The workspace is where user-defined objects accumulate. You can also get a listing of these objects with commands:

```
objects()
#> [1] "this_is_a_really_long_name"
#> [2] "turner_rocks"
#> [3] "x"
#> [4] "y"
#> [5] "yo"
ls()
#> [1] "this_is_a_really_long_name"
#> [2] "turner_rocks"
#> [3] "x"
#> [4] "y"
#> [5] "yo"
```

If you want to remove the object named `y`, you can do this:

```
rm(y)
```

To remove everything:

```
rm(list = ls())
```

or click the broom in RStudio's Environment pane.

8.4 Workspace and working directory

One day you will need to quit R, go do something else and return to your analysis later (this is a very happy day).

One day you will have multiple analyses going that use R and you want to keep them separate (a not so happy day).

One day you will need to bring data from the outside world into R and send numerical results and figures from R back out into the world (the happiest of days).

To handle these real life situations, you need to make two decisions:

- What about your analysis is "real", i.e. will you save it as your lasting record of what happened?
- Where does your analysis "live"?

8.4.1 Workspace, `.RData`

As a beginning R user, it's OK to consider your workspace "real". *Very soon*, I urge you to evolve to the next level, where you consider your saved R scripts as "real". (In either case, of course the input data is very much real and requires preservation!) With the input data and the R code you used, you can reproduce *everything*. You can make your analysis fancier. You can get to the bottom of puzzling results and discover and fix bugs in your code. You can reuse the code to conduct similar analyses in new projects. You can remake a figure with different aspect ratio or save is as TIFF instead of PDF. You are ready to take questions. You are ready for the future.

If you regard your workspace as "real" (saving and reloading all the time), if you need to redo analysis … you're going to either redo a lot of typing (making mistakes all the way) or will have to mine your R history for the commands you used. Rather than [becoming an expert on managing the R history][rstudio-command-history], a better use of your time and psychic energy is to keep your "good" R code in a script for future reuse.

Because it can be useful sometimes, note the commands you've recently run appear in the History pane.

But you don't have to choose right now and the two strategies are not incompatible. Let's demo the save / reload the workspace approach.

Upon quitting R, you have to decide if you want to save your workspace, for potential restoration the next time you launch R. Depending on your set up, R or your IDE, e.g. RStudio, will probably prompt you to make this decision.

Quit R/RStudio, either from the menu, using a keyboard shortcut, or by typing `q()` in the Console. You'll get a prompt like this:

> Save workspace image to ~/.Rdata?

Note where the workspace image is to be saved and then click "Save".

Using your favorite method, visit the directory where image was saved and verify there is a file named `.RData`. You will also see a file `.Rhistory`, holding the commands submitted in your recent session.

Restart RStudio. In the Console you will see a line like this:

```
[Workspace loaded from ~/.RData]
```

indicating that your workspace has been restored. Look in the Workspace pane and you'll see the same objects as before. In the History tab of the same pane, you should also see your command history. You're back in business. This way of starting and stopping analytical work will not serve you well for long but it's a start.

8.4.2 Working directory

Any process running on your computer has a notion of its “working directory”. In R, this is where R will look, by default, for files you ask it to load. It also where, by default, any files you write to disk will go. Chances are your current working directory is the directory we inspected above, i.e. the one where RStudio wanted to save the workspace.

You can explicitly check your working directory with:

```
getwd()
```

It is also displayed at the top of the RStudio console.

As a beginning R user, it’s OK let your home directory or any other weird directory on your computer be R’s working directory. *Very soon*, I urge you to evolve to the next level, where you organize your analytical projects into directories and, when working on project A, set R’s working directory to the associated directory.

Although I do not recommend it, in case you’re curious, you can set R’s working directory at the command line like so:

```
setwd("~/myCoolProject")
```

Although I do not recommend it, you can also use RStudio’s Files pane to navigate to a directory and then set it as working directory from the menu: *Session > Set Working Directory > To Files Pane Location*. (You’ll see even more options there). Or within the Files pane, choose “More” and “Set As Working Directory”.

But there’s a better way. A way that also puts you on the path to managing your R work like an expert.

8.5 Exercises

1. Create an object called “cool_object” and assign it the number 100.
2. Create a new object called “big_brain” and multiply the object from question one by 15.
3. Print both objects.
4. Use base R functions to return today’s date and print it.
5. Create a sequence of numbers counting from 10 to 100 by 2.

6. Identify your working directory. What is it? Change it to where you want it.

7. Save the R script that answers questions 1 through 5 above. Save it; clean and close Rstudio; reopen your script and run it. Make sense?

Chapter 9

Objects and Arithmetic

9.1 Introduction

R stores information and operates on objects. The simplest objects are scalars, vectors and matrices. But there are many others: lists and dataframes for example. In advanced use of R it can also be useful to define new types of object, specific for particular application. We will stick with just the most commonly used objects here. An important feature of R is that it will do different things on different types of objects. For example, type:

```
4+6
#> [1] 10
```

So, R does scalar arithmetic returning the scalar value 10. (In actual fact, R returns a vector of length 1 - hence the [1] denoting first element of the vector. We can assign objects values for subsequent use. For example:

```
x<-6
y<-4
z<-x+y
```

would do the same calculation as above, storing the result in an object called z. We can look at the contents of the object by simply typing its name:

```
z
#> [1] 10
```

Storing things such as calculations as objects is extremely useful, especially in longer scripts. Mainly because you will likely call the same equation multiple times and if you need to revise it in any way you only need to change the initial assignment rather than every line.

9.2 Basic Functions

At any time we can list the objects which we have created:

```
ls()
#> [1] "x" "y" "z"
```

Notice that ls is actually an object itself. Typing ls would result in a display of the contents of this object, in this case, the commands of the function. The use of parentheses, ls(), ensures that the function is executed and its result - in this case, a list of the objects in the directory - displayed. More commonly a function will operate on an object, for example:

```
sqrt(16)
#> [1] 4
```

calculates the square root of 16. Objects can be removed from the current workspace with the rm function:

```
rm(x,y)
```

for example.

There are many standard functions available in R, and it is also possible to create new ones. Vectors can be created in R in a number of ways. We can describe all of the elements:

```
z<-c(5,9,1,0)
```

Note the use of the function c to concatenate or glue together individual elements. This function can be used much more widely, for example

```
x<-c(5,9)
y<-c(1,0)
z<-c(x,y)
```

would lead to the same result by gluing together two vectors to create a single vector.

Sequences can be generated as follows:

```
x<-1:10
```

while more general sequences can be generated using the seq command. For example:

```
seq(1,9,by=2)
#> [1] 1 3 5 7 9
seq(8,20,length=6)
#> [1]  8.0 10.4 12.8 15.2 17.6 20.0
```

These examples illustrate that many functions in R have optional arguments, in this case, either the step length or the total length of the sequence (it doesn't make sense to use both). If you leave out both of these options, R will make its own default choice, in this case assuming a step length of 1. So, for example,

```
seq(8,20,length=6)
#> [1]  8.0 10.4 12.8 15.2 17.6 20.0
x<-seq(1,10)
x
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

also generates a vector of integers from 1 to 10.

At this point it's worth mentioning the help facility. If you don't know how to use a function, or don't know what the options or default values are, type help(functionname) where functionname is the name of the function you are interested in. This will usually help and will often include examples to make things even clearer.

Another useful function for building vectors is the rep command for repeating things. For example:

```
rep(0,100)
#>   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#>  [28] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#>  [55] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
#>  [82] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
rep(1:3,6)
#>  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
rep(1:3,c(6,6,6))
#>  [1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3
```

which we could also simplify cleverly as:

```
rep(1:3,rep(6,3))
#>  [1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3
```

As explained above, R will often adapt to the objects it is asked to work on. For example:

```
x<-c(6,8,9)
y<-c(1,2,4)
x+y
#> [1]  7 10 13
```

and

```
x*y
#> [1]  6 16 36
```

showing that R uses componentwise arithmetic on vectors. R will also try to make sense if objects are mixed. For example:

```
x<-c(6,8,9)
x+2
#> [1]  8 10 11
```

though care should be taken to make sure that R is doing what you would like it to in these circumstances.

Two particularly useful functions worth remembering are length which returns the length of a vector (i.e. the number of elements it contains) and sum which calculates the sum of the elements of a vector.

9.3 Statistics and Summaries

One of the most useful functions of R is its ability to perform statistical analysis on large pieces of data. Later in this book we will cover how this analysis can be used to build complex models and test their accuracy. For now, the focus will be on how to perform basic statistical analysis and the definitions of the more basic terms.

9.3.1 Statistical Analysis

You've undoubtedly have heard the terms mean, median, standard deviation and variance. However, if asked to define these terms and provide examples of their usefulness, what would you say? This section is going to answer the first part of that question.

Mean is a pretty straight forward concept for most. Commonly referred to as the "average", you find this number by adding all the numbers in your data set together and then divide by the total number of points you used. This is easily done on a calculator if you are working with less then 20 digits, however this is almost never the case in data science. Often, especially with R, you will be dealing with data sets with hundreds, thousands, or maybe even hundreds of thousands of data points. Fortunately, R takes the pain away from this process and offers a very simple function that will do this for you. Start by creating an object that contains a list of numbers, and then simply use the mean function to calculate the average of your variable.

```
a <- c(3.8,4,3.1,2.1,12.6,17,8.43,11,2,3,9,5,3,0.5)

mean(a)
#> [1] 6.037857
```

The median of a data set is data point that fall in the middle of all the other points when they are ordered from lowest to highest. Again, something easily found even without a calculator if you have small data sets, however humans can miscount and some data sets are just too massive for us to do on our own. R never makes these counting mistakes and by using a simple function you save yourself hours of mundane counting. Start the same way you found mean by creating an object with various numbers and then simply use the `median` function.

```
b <- c(3.8,4,3.1,2.1,12.6,17,8.43,11,2,3,9,5,3,0.5)

median(b)
#> [1] 3.9
```

Many people have heard of standard deviation, but not many can provide an accurate definition. Standard deviation is the measure of variance (see next paragraph) between numbers in a data set and that data set's mean. Typically a lower standard deviation is what you want to see as this proves there is little variance in your data set and that typically means a higher correlation. If you wanted to find this by hand you would have to find the square root of the variance between each point and the mean. This is easy if you have all the numbers you need, but that is almost never the case and thus you would have to calculate all that variance by hand and that is too much work. Why take all those extra steps when R can do it for you? The `sd()` function does all the work without any hassle, all you have to do is identify the object you want it to find the standard deviation of.

```
d <- c(3.8,4,3.1,2.1,12.6,17,8.43,11,2,3,9,5,3,0.5)

sd(d)
#> [1] 4.821066
```

Finally, we can cover variance. Variance is the measure of distance between two numbers. This statistic is often used to measure how spread out your data is and can be used to determine your model's accuracy. Typically, a higher variance means your model is inaccurate. So, when building models and reviewing your summary statistics, you want your variance to be as low as possible. There is a var() function that will find the variance of a variable for you, however typically you will create a confusion matrix (discussed in a later section) that will help you determine the variance, and thus accuracy of your model.

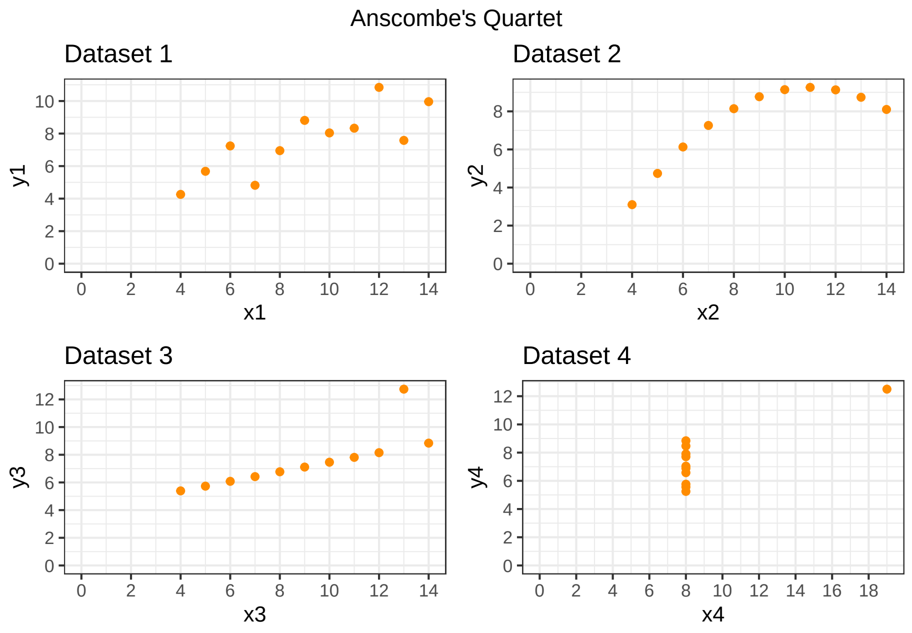
9.3.2 Anscombe's Quartet

You may be asking now, why bother visualizing data if I have all these numbers that will tell me what I need to know about my data? You are correct in saying that statistical data is very useful, and in many cases essential, however visualization is equally as important. Anscombe's Quartet is a statistical phenomenon where four sets of data have very similar statistical properties, but very are completely different when graphed. Let's take a look at the data:

```
#>    x1 x2 x3 x4    y1   y2    y3    y4
#> 1  10 10 10  8  8.04 9.14  7.46  6.58
#> 2   8  8  8  8  6.95 8.14  6.77  5.76
#> 3  13 13 13  8  7.58 8.74 12.74  7.71
#> 4   9  9  9  8  8.81 8.77  7.11  8.84
#> 5  11 11 11  8  8.33 9.26  7.81  8.47
#> 6  14 14 14  8  9.96 8.10  8.84  7.04
#> 7   6  6  6  8  7.24 6.13  6.08  5.25
#> 8   4  4  4 19  4.26 3.10  5.39 12.50
#> 9  12 12 12  8 10.84 9.13  8.15  5.56
#> 10  7  7  7  8  4.82 7.26  6.42  7.91
#> 11  5  5  5  8  5.68 4.74  5.73  6.89
```

```
#>                    x1        x2        x3        x4
#> nobs        11.000000 11.000000 11.000000 11.000000
#> NAs          0.000000  0.000000  0.000000  0.000000
#> Minimum      4.000000  4.000000  4.000000  8.000000
#> Maximum     14.000000 14.000000 14.000000 19.000000
#> 1. Quartile  6.500000  6.500000  6.500000  8.000000
#> 3. Quartile 11.500000 11.500000 11.500000  8.000000
#> Mean         9.000000  9.000000  9.000000  9.000000
#> Median       9.000000  9.000000  9.000000  8.000000
#> Sum         99.000000 99.000000 99.000000 99.000000
#> SE Mean      1.000000  1.000000  1.000000  1.000000
#> LCL Mean     6.771861  6.771861  6.771861  6.771861
#> UCL Mean    11.228139 11.228139 11.228139 11.228139
#> Variance    11.000000 11.000000 11.000000 11.000000
#> Stdev        3.316625  3.316625  3.316625  3.316625
#> Skewness     0.000000  0.000000  0.000000  2.466911
#> Kurtosis    -1.528926 -1.528926 -1.528926  4.520661
#>                    y1        y2        y3        y4
#> nobs        11.000000 11.000000 11.000000 11.000000
#> NAs          0.000000  0.000000  0.000000  0.000000
#> Minimum      4.260000  3.100000  5.390000  5.250000
#> Maximum     10.840000  9.260000 12.740000 12.500000
#> 1. Quartile  6.315000  6.695000  6.250000  6.170000
#> 3. Quartile  8.570000  8.950000  7.980000  8.190000
#> Mean         7.500909  7.500909  7.500000  7.500909
#> Median       7.580000  8.140000  7.110000  7.040000
#> Sum         82.510000 82.510000 82.500000 82.510000
#> SE Mean      0.612541  0.612568  0.612196  0.612242
#> LCL Mean     6.136083  6.136024  6.135943  6.136748
#> UCL Mean     8.865735  8.865795  8.864057  8.865070
#> Variance     4.127269  4.127629  4.122620  4.123249
#> Stdev        2.031568  2.031657  2.030424  2.030579
#> Skewness    -0.048374 -0.978693  1.380120  1.120774
#> Kurtosis    -1.199123 -0.514319  1.240044  0.628751
```

As you can see from the table, the summary statistics for the four tables look very similar, however let's graph these and see what kind of visualizations we get.



Pretty rad right? So yes, statistical data is vital, however it should not be the only thing you look at, sometimes a visualization can help you understand the data more! Later in this book, and in the upper level data analytics courses, we will cover visualizations more and how they can be useful.

9.4 Exercises

1. Define

```
x<-c(4,2,6)
y<-c(1,0,-1)
```

2. Decide what the result will be of the following:
    - length(x)
    - sum(x)
    - sum(x^2)
    - x+y
    - x*y
    - x-2
    - x^2
3. Use R to check your answers.
4. Decide what the following sequences are and use R to check your answers:
    - 7:11
    - seq(2,9)
    - seq(4,10,by=2)
    - seq(3,30,length=10)
    - seq(6,-4,by=-2)
5. Determine what the result will be of the following R expressions, and then use R to check if you are right:
    - rep(2,4)
    - rep(c(1,2),4)
    - rep(c(1,2),c(4,4))
    - rep(1:4,4)
    - rep(1:4,rep(3,4))
6. Use the rep function to define simply the following vectors in R.
    - 6,6,6,6,6,6
    - 5,8,5,8,5,8,5,8
    - 5,5,5,5,8,8,8,8

Chapter 10

Summaries and Subscripting

10.1 Introduction

Let's suppose we've collected some data from an experiment and stored them in an object x:

```
x<-c(7.5,8.2,3.1,5.6,8.2,9.3,6.5,7.0,9.3,1.2,14.5,6.2)
```

Some simple summary statistics of these data can be produced:

```
mean(x)
#> [1] 7.216667
var(x)
#> [1] 11.00879
sd(x)
#> [1] 3.317949
summary(x)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   1.200   6.050   7.250   7.217   8.475  14.500
```

which should all be self explanatory.

It may be, however, that we subsequently learn that the first 6 data points correspond to measurements made on one machine, and the second six on another machine.

This might lead us to want to summarize the two sets of data separately, so we would need to extract from x the two relevant subvectors. This is achieved by subscripting:

```
x[1:6]
#> [1] 7.5 8.2 3.1 5.6 8.2 9.3
x[7:12]
#> [1]  6.5  7.0  9.3  1.2 14.5  6.2
summary(x[1:6])
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   3.100   6.075   7.850   6.983   8.200   9.300
summary(x[7:12])
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   1.200   6.275   6.750   7.450   8.725  14.500
```

Other subsets can be created in the obvious way. For example:

```
x[c(2,4,9)]
#> [1] 8.2 5.6 9.3
```

Negative integers can be used to exclude particular elements. For example

```
x[-(1:6)]
#> [1]  6.5  7.0  9.3  1.2 14.5  6.2
```

has the same effect as

```
x[7:12]
#> [1]  6.5  7.0  9.3  1.2 14.5  6.2
```

10.2 Exercises (Summaries and Subscripting)

1. If x<- c(5,9,2,3,4,6,7,0,8,12,2,9) decide what each of the following is and use R to check your answers:

(a) x[2]
(b) x[2:4]
(c) x[c(2,3,6)]
(d) x[c(1:5,10:12)]
(e) x[-(10:12)]

2. The data y<-c(33,44,29,16,25,45,33,19,54,22,21,49,11,24,56) contain sales of milk in gallons for 5 days in three different shops (the frst 3 values are for shops 1,2 and 3 on Monday, etc.) Produce a statistical summary of the sales for each day of the week and also for each shop.

Chapter 11

Matrices

11.1 CBind and RBind

Matrices can be created in R in a variety of ways. Perhaps the simplest is to create the columns (just a couple of objects) and then glue them together with the command cbind. For example,

```
x<-c(5,7,9)
y<-c(6,3,4)
z<-cbind(x,y)
z
#>      x y
#> [1,] 5 6
#> [2,] 7 3
#> [3,] 9 4
```

The dimension of a matrix can be checked with the dim command:

```
dim(z)
#> [1] 3 2
```

[1] 3 2 i.e., three rows and two columns. There is a similar command, rbind, for building matrices by gluing rows together.

The functions cbind and rbind can also be applied to matrices themselves (provided the dimensions match) to form larger matrices. For example,

```
rbind(z,z)
#>      x y
#> [1,] 5 6
#> [2,] 7 3
#> [3,] 9 4
#> [4,] 5 6
#> [5,] 7 3
#> [6,] 9 4
```

11.1.1 Review Questions

1) Create a matrix made up of two columns showing the GPAs and number of hours studied by seven students.

2) Recreate the following matrix in R:

```
#>      x   y
#> [1,] 5 3.4
#> [2,] 7 4.0
#> [3,] 2 2.5
#> [4,] 3 3.2
#> [5,] 8 2.8
#> [6,] 4 3.1
#> [7,] 2 3.6
```

3) Using the appropriate function, combine the two matrices you created above.

11.2 Matrix Function

Matrices can also be built by explicit construction via the function matrix. For example,

```
z<-matrix(c(5,7,9,6,3,4),nrow=3)
```

results in a matrix z identical to z above. Notice that the dimension of the matrix is determined by the size of the vector and the requirement that the number of rows is 3, as specified by the argument nrow=3. As an alternative we could have specified the number of columns with the argument ncol=2 (obviously, it is unnecessary to give both). Notice that the matrix is ‘flled up’ column-wise. If instead you wish to fill up row-wise, add the option byrow=T. For example,

```
z<-matrix(c(5,7,9,6,3,4),nr=3,byrow=T)
z
#>      [,1] [,2]
#> [1,]    5    7
#> [2,]    9    6
#> [3,]    3    4
```

Notice that the argument nrow has been abbreviated to nr. Such abbreviations are always possible for function arguments provided it induces no ambiguity - if in doubt always use the full argument name.

As usual, R will try to interpret operations on matrices in a natural way. For example, with z as above, and

```
y<-matrix(c(1,3,0,9,5,-1),nrow=3,byrow=T)
y
#>      [,1] [,2]
#> [1,]    1    3
#> [2,]    0    9
#> [3,]    5   -1
```

we obtain

```
y+z
#>      [,1] [,2]
#> [1,]    6   10
#> [2,]    9   15
#> [3,]    8    3
```

and

```
y*z
#>      [,1] [,2]
#> [1,]    5   21
#> [2,]    0   54
#> [3,]   15   -4
```

Other useful functions on matrices are to transpose a matrix:

```
z
#>      [,1] [,2]
#> [1,]    5    7
#> [2,]    9    6
#> [3,]    3    4
```

```
t(z)
#>      [,1] [,2] [,3]
#> [1,]    5    9    3
#> [2,]    7    6    4
```

As with vectors it is useful to be able to extract sub-components of matrices. In this case, wemay wish to pick out individual elements, rows or columns. As before, the [ ] notation is used to subscript. The following examples should make things clear:

```
z[1,1]
#> [1] 5
```

```
z[c(2,3),2]
#> [1] 6 4
```

```
z[,2]
#> [1] 7 6 4
```

```
z[1:2,]
#>      [,1] [,2]
#> [1,]    5    7
#> [2,]    9    6
```

So, in particular, it is necessary to specify which rows and columns are required, whilst omitting the integer for either dimension implies that every element in that dimension is selected.

11.3 Exercises

1. Create this matrix in R

```
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    7    8   11   -5
#> [2,]    3    8    6    3   -9
#> [3,]    0   11   14   14   14
```

2. Create in R these matrices:

```
x
#>      [,1] [,2]
#> [1,]    1    7
#> [2,]    8   11
#> [3,]    5    9
y
#>      [,1] [,2]
#> [1,]    6    8
#> [2,]    2    1
#> [3,]    1   -7
```

3. Calculate the following and check your answers in R:

(a) 2*x
(b) x*x
(c) t(y)

```
#>      [,1] [,2]
#> [1,]    2   14
#> [2,]   16   22
#> [3,]   10   18
#>      [,1] [,2]
#> [1,]    1   49
#> [2,]   64  121
#> [3,]   25   81
#>      [,1] [,2] [,3]
#> [1,]    6    2    1
#> [2,]    8    1   -7
```

4. With x and y as above, calculate the effect of the following subscript operations and check your answers in R.

(a) x[1,]
(b) x[2,]
(c) x[,2]
(d) y[1,2]
(e) y[,2:3]

Chapter 12

Preloaded data and `mtcars`

R comes with several built-in data sets, **which are generally used as demo data for playing with R functions.**

To see the datasets type:

```
data()
```

12.1 Practicing with `mtcars` data set

This demonstration is based on the datasset `mtcars`.

1. Read in `mtcars`

```
data(mtcars)
```

2. View first few rows and last few rows of mtcars dataframe using functions `head()` and `tail()`

```
head(mtcars)
#>                    mpg cyl disp  hp drat    wt  qsec vs am
#> Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1
#> Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1
#> Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1
#> Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0
#> Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0
#> Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0
#>                   gear carb
#> Mazda RX4            4    4
#> Mazda RX4 Wag        4    4
#> Datsun 710           4    1
#> Hornet 4 Drive       3    1
#> Hornet Sportabout    3    2
#> Valiant              3    1
tail(mtcars)
#>                 mpg cyl  disp  hp drat    wt qsec vs am
#> Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1
#> Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1
#> Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1
#> Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1
#> Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1
#> Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1
#>                gear carb
#> Porsche 914-2     5    2
#> Lotus Europa      5    2
#> Ford Pantera L    5    4
#> Ferrari Dino      5    6
#> Maserati Bora     5    8
#> Volvo 142E        4    2
```

3. Some info about mtcars dataframe using function `colnames()`, `rownames()`, `summary()`and `dim()`

```
colnames(mtcars)
#>  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"
#>  [9] "am"   "gear" "carb"
rownames(mtcars)
#>  [1] "Mazda RX4"           "Mazda RX4 Wag"
#>  [3] "Datsun 710"          "Hornet 4 Drive"
#>  [5] "Hornet Sportabout"   "Valiant"
#>  [7] "Duster 360"          "Merc 240D"
#>  [9] "Merc 230"            "Merc 280"
#> [11] "Merc 280C"           "Merc 450SE"
#> [13] "Merc 450SL"          "Merc 450SLC"
#> [15] "Cadillac Fleetwood"  "Lincoln Continental"
#> [17] "Chrysler Imperial"   "Fiat 128"
#> [19] "Honda Civic"         "Toyota Corolla"
#> [21] "Toyota Corona"       "Dodge Challenger"
#> [23] "AMC Javelin"         "Camaro Z28"
#> [25] "Pontiac Firebird"    "Fiat X1-9"
#> [27] "Porsche 914-2"       "Lotus Europa"
#> [29] "Ford Pantera L"      "Ferrari Dino"
#> [31] "Maserati Bora"       "Volvo 142E"
summary(mtcars)
```

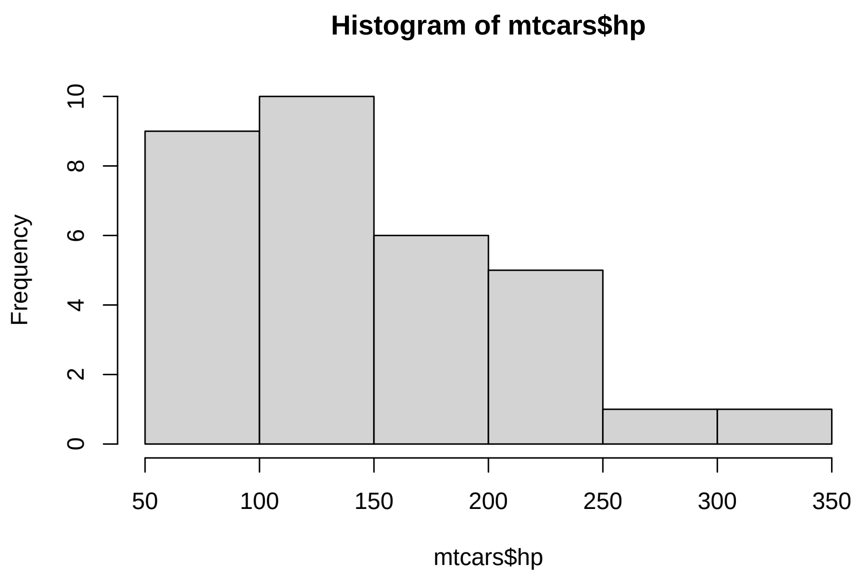
```
#>       mpg             cyl             disp      
#>  Min.   :10.40   Min.   :4.000   Min.   : 71.1  
#>  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8  
#>  Median :19.20   Median :6.000   Median :196.3  
#>  Mean   :20.09   Mean   :6.188   Mean   :230.7  
#>  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0  
#>  Max.   :33.90   Max.   :8.000   Max.   :472.0  
#>        hp             drat             wt       
#>  Min.   : 52.0   Min.   :2.760   Min.   :1.513  
#>  1st Qu.: 96.5   1st Qu.:3.080   1st Qu.:2.581  
#>  Median :123.0   Median :3.695   Median :3.325  
#>  Mean   :146.7   Mean   :3.597   Mean   :3.217  
#>  3rd Qu.:180.0   3rd Qu.:3.920   3rd Qu.:3.610  
#>  Max.   :335.0   Max.   :4.930   Max.   :5.424  
#>       qsec             vs               am        
#>  Min.   :14.50   Min.   :0.0000   Min.   :0.0000  
#>  1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000  
#>  Median :17.71   Median :0.0000   Median :0.0000  
#>  Mean   :17.85   Mean   :0.4375   Mean   :0.4062  
#>  3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000  
#>  Max.   :22.90   Max.   :1.0000   Max.   :1.0000  
#>       gear            carb      
#>  Min.   :3.000   Min.   :1.000  
#>  1st Qu.:3.000   1st Qu.:2.000  
#>  Median :4.000   Median :2.000  
#>  Mean   :3.688   Mean   :2.812  
#>  3rd Qu.:4.000   3rd Qu.:4.000  
#>  Max.   :5.000   Max.   :8.000  
dim(mtcars)
#> [1] 32 11
```

4. To calculate the variance of weight:

```
var(mtcars$wt)
#> [1] 0.957379
```

5. To get the histogram of hp, the code below will produce a histogram:

```
hist(mtcars$hp)
```

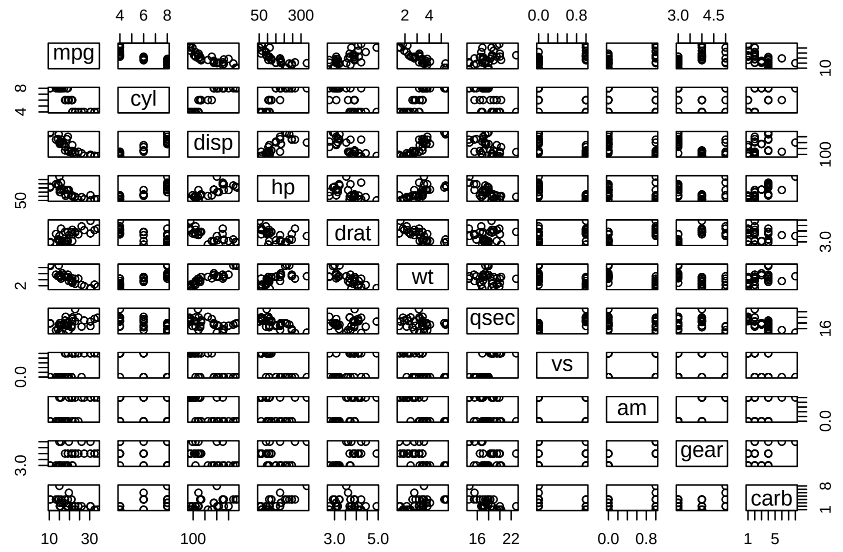


6. To calculate the quantiles by percent:

```
quantile(mtcars$wt, c(.2, .4, .8))
#>   20%   40%   80% 
#> 2.349 3.158 3.770
```

12.2 Excerises for you:

1. Find the minimum and maximum value of `mpg`
2. Find the mean and standard deviation of data variable `mpg`
3. What variable has a 3rd quartile value of 180.0?
4. Create and explain what this means



5. Create and explain what this means

```
#>              mpg        cyl       disp         hp
#> mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684
#> cyl  -0.8521620  1.0000000  0.9020329  0.8324475
#> disp -0.8475514  0.9020329  1.0000000  0.7909486
#> hp   -0.7761684  0.8324475  0.7909486  1.0000000
#> drat  0.6811719 -0.6999381 -0.7102139 -0.4487591
#> wt   -0.8676594  0.7824958  0.8879799  0.6587479
#> qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234
#> vs    0.6640389 -0.8108118 -0.7104159 -0.7230967
#> am    0.5998324 -0.5226070 -0.5912270 -0.2432043
#> gear  0.4802848 -0.4926866 -0.5555692 -0.1257043
#> carb -0.5509251  0.5269883  0.3949769  0.7498125
#>              drat         wt        qsec         vs
#> mpg   0.68117191 -0.8676594  0.41868403  0.6640389
#> cyl  -0.69993811  0.7824958 -0.59124207 -0.8108118
#> disp -0.71021393  0.8879799 -0.43369788 -0.7104159
#> hp   -0.44875912  0.6587479 -0.70822339 -0.7230967
#> drat  1.00000000 -0.7124406  0.09120476  0.4402785
#> wt   -0.71244065  1.0000000 -0.17471588 -0.5549157
#> qsec  0.09120476 -0.1747159  1.00000000  0.7445354
#> vs    0.44027846 -0.5549157  0.74453544  1.0000000
#> am    0.71271113 -0.6924953 -0.22986086  0.1683451
#> gear  0.69961013 -0.5832870 -0.21268223  0.2060233
#> carb -0.09078980  0.4276059 -0.65624923 -0.5696071
#>               am       gear        carb
#> mpg   0.59983243  0.4802848 -0.55092507
#> cyl  -0.52260705 -0.4926866  0.52698829
#> disp -0.59122704 -0.5555692  0.39497686
#> hp   -0.24320426 -0.1257043  0.74981247
#> drat  0.71271113  0.6996101 -0.09078980
#> wt   -0.69249526 -0.5832870  0.42760594
#> qsec -0.22986086 -0.2126822 -0.65624923
#> vs    0.16834512  0.2060233 -0.56960714
#> am    1.00000000  0.7940588  0.05753435
#> gear  0.79405876  1.0000000  0.27407284
#> carb  0.05753435  0.2740728  1.00000000
```

6. Create a variable called `efficiency` which is mpg divided by weight. Which car has the max `efficiency` and what is this value?
7. Which variable in this dataset has the greatest standard deviation?
8. How many cars have 3 gears?
9. How many cars get more than 17 mpg?

Chapter 13

More simple data wrangling

13.1 a nice, fun little matrix for you

```
#>      [,1] [,2] [,3] [,4] [,5]
#> [1,]    1    4    7   10   13
#> [2,]    2    5    8   11   14
#> [3,]    3    6    9   12   15
```

1. Write the code that creates this matrix:
2. Write DIFFERENT code that creates this matrix in an alternate way:
3. In the matrix above, what does `[,4]` mean?
4. What code would return the value in the 3rd column and 3rd row?
5. What single line of would give you the average of the all the numbers in columns 2, 4, and 5 and in rows 1 and 3?
6. turn `x` into a data frame.
7. How do you **know** you have turned `x` into a data frame?

13.2 More fun (this class is really awesome isn't it?)

```
df
#>    X1 X2 X3 X4 X5 X6 X7 X8 X9 X10
#> 1   1 11 21 31 41 51 61 71 81  91
#> 2   2 12 22 32 42 52 62 72 82  92
#> 3   3 13 23 33 43 53 63 73 83  93
#> 4   4 14 24 34 44 54 64 74 84  94
#> 5   5 15 25 35 45 55 65 75 85  95
#> 6   6 16 26 36 46 56 66 76 86  96
#> 7   7 17 27 37 47 57 67 77 87  97
#> 8   8 18 28 38 48 58 68 78 88  98
#> 9   9 19 29 39 49 59 69 79 89  99
#> 10 10 20 30 40 50 60 70 80 90 100
```

1. Consider the dataframe above called `df`. What would running this code return `sum(df[7,7:10])`
2. How can you tell if an object in R is a dataframe?
3. How could you create the dataframe above called `df`?
4. What code would return the average of row 2 of `df`?
5. Consider `mtcars` dataset that comes preloaded with R that looks like this:

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

6. Why do I get this error when I run the code below: `Error in plot(hp, mpg) : object 'hp' not found`?

```
plot(hp,mpg)
```

```
Error in plot(hp, mpg) : object 'hp' not found
```

Bonus: What is a topic that you find confusing at this point in class?

Chapter 14

GDH Ice Cream

14.1 Problem Introduction

GDH provides ice cream for its wonderful customers. I LOVE GDH. Do you love it as much as me (let's discuss)?

In the last three years GDH used ice cream, in pounds, by month, as shown in the attached file.

```
#>   Month.Name year1 year2 year3
#>          Jan    60    67    64
#>          Feb    68    67    72
#>          Mar    83    62    61
#>          Apr   102    95   107
#>          May    95   105   101
#>          Jun    57    89    75
#>          Jul    61    57    81
#>          Aug   109   109   104
#>          Sep    56    86    88
#>          Oct    53    53    65
#>          Nov    74    72    72
#>          Dec    73    64    65
```

14.2 Assignment

Please answer the following questions *using R*:.

GDH provides ice cream cones for its customers. In the last three years GDH used ice cream, in pounds, by month, as shown in the attached file.

1. In R, create the above data frame and name it ice.cream
2. What is another way you could have created the same data set?
3. Using R, what is the mean using for the months of Feb and Oct?
4. Create a chart showing ice cream use over time.
5. Which year used the most ice cream?
6. Which month has the highest standard deviation of ice cream use?
7. Which year has the highest standard deviation of ice cream use?
8. Also, you May want to check out this link to look at something called dataframes that may help with this assignment (but is not absolutely necessary) https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/data.frame
9. Can you transpose your matrix?
10. Can you add meaningful row names and column names?

#Machine Learning?

Have we talked about the Target Pregnancy Story yet?

How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did

This is where the fun stuff begins! What we have learned up to this point has barely scratched the surface of what R is capable of. In the world of data science, R is used for three primary purposes, those purposes are **(1) data transformation, (2) data wrangling, (3) machine learning**. The other two purposes have been covered in earlier chapters of this book. The reason we covered the other topics first is that that lay the foundation. In the real world, it is likely you will never be given a clean data set, and you will have to do some wrangling and transformation before anything else is possible. After all, in the experience of many data science students, cleaning the data is the most tedious and time consuming process of a project.

Enough of the old stuff, what is machine learning? According to the Merriam-Webster Dictionary, machine learning is "the process by which a computer is able to improve its own performance by continuously incorporating new data into an existing statistical model." Let's take a trip back to the Target story discussed in the introductory chapter. The data scientist, or more likely data scientists (collaborative work is essential in this field), that worked on that model were likely experts in machine learning. They were able to train the computer to look through thousands (probably more) of customers' data and the computer, based off the algorithms written by these "data nerds," was able to predict whether a customer was pregnant! Think of other possible applications of this technology? We could predict how well a student will preform on an exam, the risk of someone suffering from a heart attack, or the likelihood that someone will default on a loan. Every field in existence today could find a way to implement machine learning to optimize their business.

There are two branches of machine learning, supervised and unsupervised. Both have their own unique uses, however in this course we will focus on supervised machine learning. Supervised machine learning required us to provide a clean data set with clearly defined variables and instructions. Essentially, we give the computer the information it needs and provide it with specific instructions detailing what we would like to see happen, and it does the rest. Linear regression is typically the first method of supervised learning people are introduced to, and it will be the focus of this chapter.

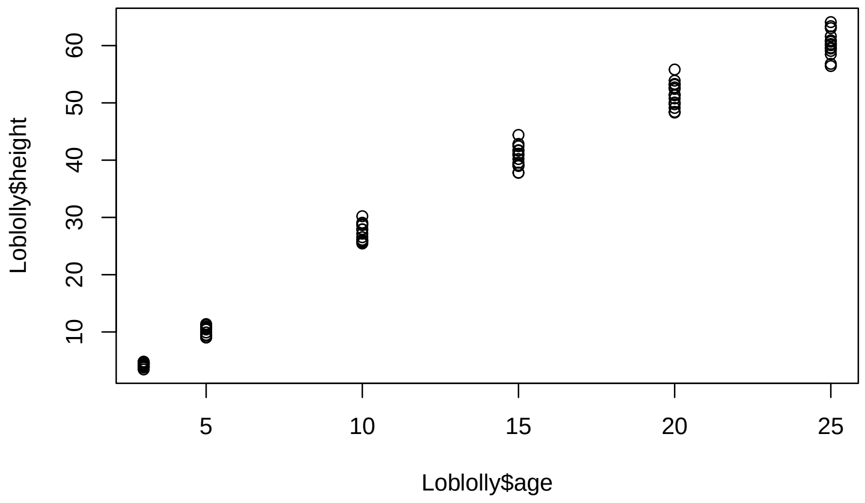
Note, these concepts are not all common sense and can be difficult to wrap your head around at times. Be sure to constantly turn to your instructor or peers for assistance and remember that there are hundreds of online resources at your disposal. As with anything though, practice makes perfect. The popular rule states that mastering a skill can take upwards of 10,000 hours! Now, this course is not going to take you 10 years to complete, however the goal is that by the end of this chapter you will know your way around the basics of linear regression.

Chapter 15

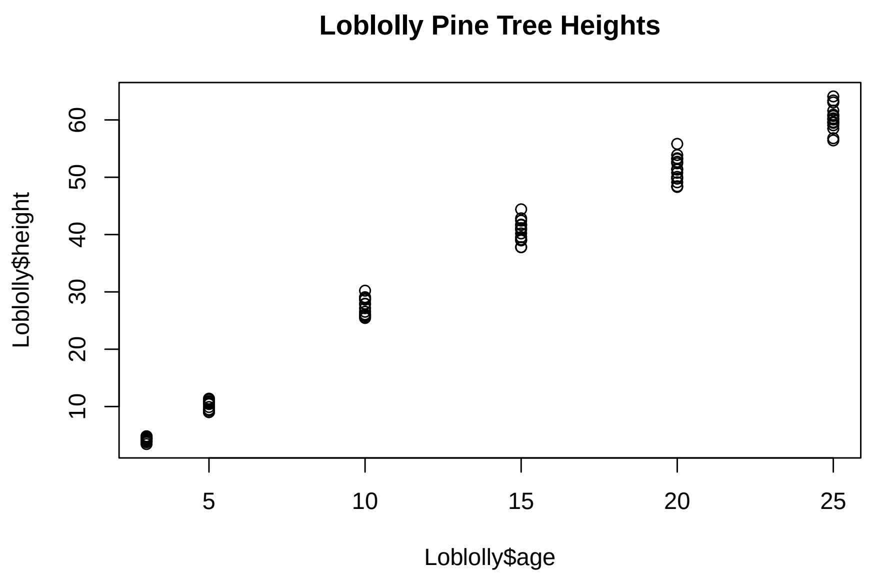
Quick Linear Regression

15.1 Quick Linear regression using `Loblolly`

1. load `Loblolly` and create a scatter plot of the data so plot so that age is the independent variable and height is the dependent variable.



2. Notice that R automatically labeled the x- and y-axes, but we also want our scatter plot to have a main title. To add a title, use the command `title(main = "Loblolly Pine Tree Heights")`.



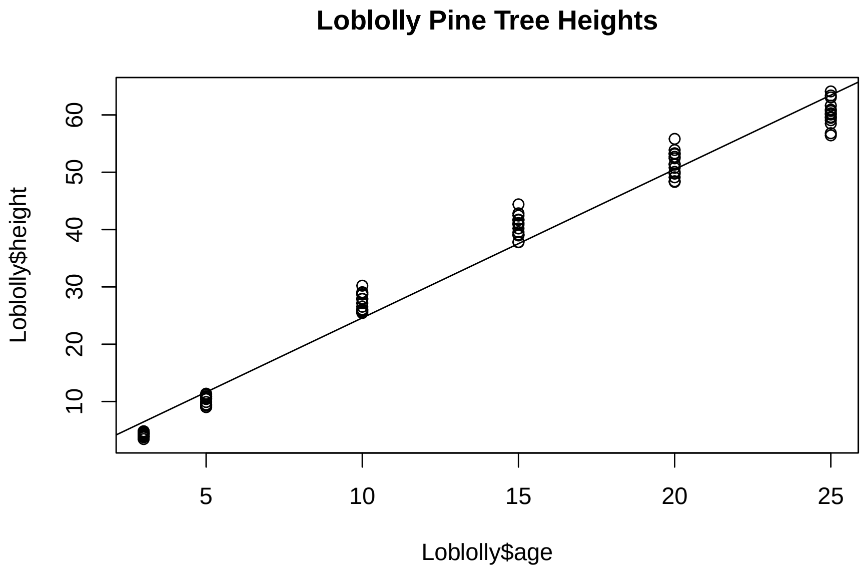
3. To find a linear model that relates the age and height of the loblolly pine trees, we will use the command `fit1<-lm(Loblolly$height~Loblolly$age)`.

   Think of `lm(Loblolly$height~Loblolly$age)` as the slope-intercept form (y=mx+b).

4. To see the model, type `fit1`

```
fit1 <- lm(Loblolly$height~Loblolly$age)
fit1
#>
#> Call:
#> lm(formula = Loblolly$height ~ Loblolly$age)
#>
#> Coefficients:
#>  (Intercept)  Loblolly$age
#>       -1.312         2.591
```

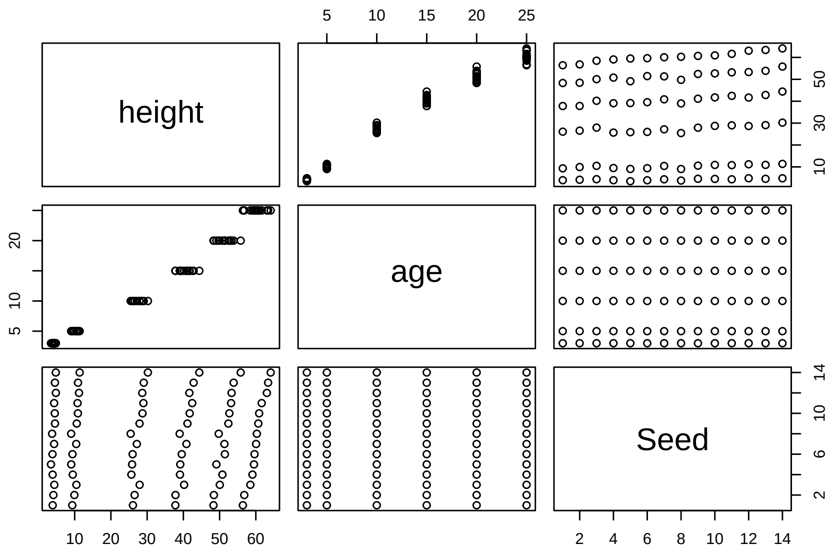
5. Now we want to add the graph of this line of best fit to our scatter plot. To do this, use the command `abline(fit1)`. .



9. The final piece of information we want about our data is the correlation of the age and height of the Loblolly pine trees. To find the correlation coefficient, use the command `cor(Loblolly$height, Loblolly$age)`

```
#> [1] 0.9899132
#>
#>  Pearson's product-moment correlation
#>
#> data:  Loblolly$height and Loblolly$age
#> t = 63.272, df = 82, p-value < 2.2e-16
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.9844505 0.9934631
#> sample estimates:
#>       cor
#> 0.9899132
```

10. What does this command do and mean: `plot(Loblolly)`?


height
age
Seed

Chapter 16

Linear Regression with mtcars

Remember: `~` here means "explained by", so the formula mpg ~ wt means we are predicting mpg as explained by wt. The most helpful way to view the output is with:

16.1 Excercises for you

16.1.1 `mtcars`

1. Which variable in the `mtcars` dataset do you think best predicts `mpg` and why?
2. What `mpg` would you predict for a car with a displacement of 333?
3. What `mpg` would you predict for a car with a displacement of 12 cylinders?
4. What `mpg` would you predict for a car with a displacement of 333 and 12 cylinders?
5. What `mpg` would you predict for a car with a displacement of 333, 12 cylinders, and weighs 4,000 pounds?

16.1.2 `trees`

Open the `trees` dataset in R.

1. What are the variables and what do they mean?
2. Make a plot with Volume on the x axis and Height on the Y and add a best fit line.

3. Use Girth and Height to predict Volume. What would you predict for a tree with a Girth of 10 and a Height of 100 feet?
4. Use Girth and Height to predict Volume. What would you predict for a tree with a Girth of 10 and a Height of 15 meters?
5. What is the maximum circumference of a tree in this dataset?

16.2 More Excercises for you

1.Open the `women` data set. Add a new variable (column) to the `women` dataframe called GPA which is these 15 numbers: 1.5, 3.7, 4,1, 3, 2.5, 3.8, 0.8, 2, 4, 1, 3, 2.5, 3.0, 4.0. You shoud get something that looks similar to mine.

```
FALSE    height weight GPA
FALSE 1      58    115 1.5
FALSE 2      59    117 3.7
FALSE 3      60    120 4.0
FALSE 4      61    123 1.0
FALSE 5      62    126 3.0
FALSE 6      63    129 2.5
FALSE 7      64    132 3.8
FALSE 8      65    135 0.8
FALSE 9      66    139 2.0
FALSE 10     67    142 4.0
FALSE 11     68    146 1.0
FALSE 12     69    150 3.0
FALSE 13     70    154 2.5
FALSE 14     71    159 3.0
FALSE 15     72    164 4.0
```

2. Use GPA and weight to predict the height of a person who is 155 pounds and has a GPA if 3.33. What is your prediction?

3. Is GPA a significant predictor of height and how do you know?

4. Create a figure showing a best fit line on of height and GPA.

5. Install the dplyr package into your Rstudio session.

Chapter 17

Deeper Linear Regression

Let's chat about why understaning linear regression is so important.

While there may always seem to be something new, cool, and shiny in the field of AI/ML, classic statistical methods that leverage machine learning techniques remain powerful and practical for solving many real-world business problems.

Let's look at a very simple model first. For this example, we will need to import the Introduction to Statistical Learning package (ISLR). We will use the "credit" data set that is part of the ISLR package.

```r
library(ISLR)
data("Credit")
attach(Credit)

M1 <- lm(Balance ~ Limit + Ethnicity)
```

lm is the function we use to create linear regression models. Now, before we discuss interpreting the results we get from this function, we will discuss the different parts of the model. The "~" symbol is the key to this entire equation. We are telling R to predict whatever is on the left side of the tilde using the variables on the right.

17.1 Interpretation of the Model

Let's run a summary on this model and see what we get.

```
summary(M1)
#> 
#> Call:
#> lm(formula = Balance ~ Limit + Ethnicity)
#> 
#> Residuals:
#>     Min      1Q  Median      3Q     Max 
#> -677.39 -145.75   -8.75  139.56  776.46 
#> 
#> Coefficients:
#>                      Estimate Std. Error t value Pr(>|t|)
#> (Intercept)        -3.078e+02  3.417e+01  -9.007   <2e-16
#> Limit               1.718e-01  5.079e-03  33.831   <2e-16
#> EthnicityAsian      2.835e+01  3.304e+01   0.858    0.391
#> EthnicityCaucasian  1.381e+01  2.878e+01   0.480    0.632
#> 
#> (Intercept)        ***
#> Limit              ***
#> EthnicityAsian        
#> EthnicityCaucasian    
#> ---
#> Signif. codes:  
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#> 
#> Residual standard error: 234 on 396 degrees of freedom
#> Multiple R-squared:  0.743,  Adjusted R-squared:  0.7411 
#> F-statistic: 381.6 on 3 and 396 DF,  p-value: < 2.2e-16
```

There is a lot of statistical jargon included in our summary that may be unfamiliar to those who have not taken statistics before. That is okay, however, because we are going to breakdown the main statistics we are interested in. Let's start with our variables and their significance in the model.

17.1.1 P-Values

The p-value of our model helps us either prove or disprove the null-hypothesis of our test. In the case of this class, the null-hypothesis is that there is no relationship between the variables we are using to make the predictions and the actual variable we are predicting. In other words, the smaller our p-value the higher the level of significance there is between our variables. When we run a summary of our linear regression model we are give multiple p-values.

First, under coefficients, they are listed for each variable. This can help us optimize our model because we can see what variables are helping make the model more accurate versus those that may be hindering its performance. Also notice the asterisks next to our p-values. R kindly puts up to three stars next to each variable to help us visually tell if they are significant, essentially more stars means a lower p-value and thus a higher correlation. The second place we see a p-value is at the bottom of our summary. This p-value will give us the overall correlation that exists in our model. As we see in this case, our p-values for this model is $<$ .00000000000000022, that is a tiny number and frankly a great p-value. Typically we want our p-value to be .05 or smaller. A p-value of .05 tells us that we have a confidence level of 95%.

17.1.2 Multiple R-Squared

R-squared tells us how well our model explains the variance in our variable. In other words, is the reason for the change in the independent variable actually due to our model's prediction? The higher the r-squared, the more accurate our model is because the better the data fits it. The maximum value r-squared can be is 1.

In our model's case, we have a multiple r-squared of .743, this means our model is approximately 74.3% accurate as this is the amount of variance in the data caused by our dependent variable. Our r-squared could certainly be better. In fact, in the real world you typically are aiming for an r-squared above .9 or .95, which means you would have 90%-95% accuracy.

17.2 Applying the Model to Make Predictions

This type of regression is refereed to as linear for a reason. If we were to visualize our model on a quadratic plane, we would see a line of best fit that would travel along through our data. This means we can simplify the model to fit the slope-intercept equation:

y = m(x)+b

In our case the slope of our line is related to the independent variables. The sum of these slopes will give us the overall slope of our line and the intercept is provided by the equation summary. If we modify this equation to be more applicable to our situation we would get something like this:

y = m1x1 + m2x2 … + b

Let's look back at our example model from before

```
M1 <- lm(Balance ~ Limit + Ethnicity)
summary(M1)
#>
#> Call:
#> lm(formula = Balance ~ Limit + Ethnicity)
```

```
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -677.39 -145.75   -8.75  139.56  776.46
#>
#> Coefficients:
#>                      Estimate Std. Error t value Pr(>|t|)
#> (Intercept)        -3.078e+02  3.417e+01  -9.007   <2e-16
#> Limit               1.718e-01  5.079e-03  33.831   <2e-16
#> EthnicityAsian      2.835e+01  3.304e+01   0.858    0.391
#> EthnicityCaucasian  1.381e+01  2.878e+01   0.480    0.632
#>
#> (Intercept)        ***
#> Limit              ***
#> EthnicityAsian
#> EthnicityCaucasian
#> ---
#> Signif. codes:
#> 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 234 on 396 degrees of freedom
#> Multiple R-squared:  0.743,  Adjusted R-squared:  0.7411
#> F-statistic: 381.6 on 3 and 396 DF,  p-value: < 2.2e-16
```

We see that our limit variable has an estimate of 1.718e-01, this is our slope. When dealing with quantitative variables, we simply multiply our slope by the intended independent variable. So, if we wanted to find the balance of someone with a limit of 400, we would multiply 1.718e-01 by 400.

With the qualitative variables, in this case ethnicity, we multiply the estimate of the TRUE values by 1 and FALSE values by 0, thus cancelling the FALSE values out.

Let's look at an example. If we used our above equation to predict the balance of someone who was Caucasian and has a credit limit of 500, here is the equation we would set up:

```
y = (1.718e-1*500) + (1.381e+01*1) + (2.835e+01*0) + (-3.078e+02)
y
#> [1] -208.09
```

So, according to our model our customer would have a balance of -208.09. This number may seem funny, but keep in mind that our r-squared was not the best for this model making it inaccurate and the ethnicity of the customer was not highly correlated with the balance. Both of these facts may cause our prediction to be off. If we were actually creating a model that could predict balance, we would want to look at some of the more correlated variables in the data set.

17.3 Review Questions

1) Create a linear model predicting using the ISLR data set that predicts a customer's credit limit based on their age, current balance, and the number of cards they have.
2) What is the p-value of this model? What does this tell us?
3) List the variables in order from most correlated to least. How do you know that they are correlated?
4) What is the multiple r-squared of the model? What does this tell us? Is this good or bad?
5) What would be the credit limit of a 29 year old with 5 cards and a total balance of 1500?
6) Explain what the following piece of code does

```
library(ISLR)
data("Credit")
attach(Credit)
q1 <- lm(Cards ~ Limit + Balance + Education)
```

Chapter 18

Linear Regression Practice

18.1 Your homework is to watch these videos which are posted under the linear regression header on Brightspace and then do the following homework:

18.1.1 Videos to watch:

1. Linear regression women
2. Best fit line women

18.2 Problems

Once you have watched these videos, and you can refer to them as often as you would like, please answer and do the following:

1. Use linear regression to predict the weight of a woman who is 100 inches tall.
2. Use linear regression to predict the height of the woman who weighs 200 pounds.
3. Use linear regression to predict the height of a woman who weighs 5 pounds.
4. Use linear regression to predict the weight of a woman who is 200 inches tall.
5. Plot weight on the X axes and height on the y-axes and create a best fit line on your plot.

6. Plot height on the y-axes and wait on the X axes and create a best fit line on your plot.
7. Add a another column to the women dataframe called GPA which is these 15 numbers: 1.5,4,2,3.7,4,1, 3, 2.5, 3.8, 0.8, 2, 4, 1, 3, 2.
8. Use GPA to predict height. Is GPA a significant predictor and how do you know? Draw a best fit line on this relationship.
9. Use GPA to predict a weight. Is GPA a significant predictor and how do you know? Draw a bested line on this relationship, too.
10. Predict the height of a person with a GPA of 4.0.

18.3 Multivariate Regression

I have posted a short video walking you through how to perform multiple linear regression – where you have more than one variable predicting another.

Using the data set mtcars data set:

1. Which variable predicts miles per gallon better gear or qsec? How can you tell?
2. Which two variables out of these four (qsec, vs, am, gear) together best predict miles per gallon?
3. Using only the number of cylinders, displacement, and weight what would mpg you would you predict for a car with a displacement of 400 inches, eight cylinders, and weighing 2000 pounds?
4. Be able to explain in a model which variables are significantly significant.
5. Be able to explain what adjusted R squared means.

Chapter 19

Filters and packages

Filtering data is one of the very basic operation when you work with data. You want to remove a part of the data that is invalid or simply you're not interested in. Or, you want to zero in on a particular part of the data you want to know more about

For example, in the `randu` dataset, how many `y` variables are greater than 0.5? 0.6?

```
length(randu$y[randu$y>0.5])
#> [1] 191
new.randu <- randu$y[randu$y>0.6]
head(new.randu)
#> [1] 0.873416 0.648545 0.826873 0.926590 0.741526 0.846041
length(new.randu)
#> [1] 161
```

In the `randu` dataset, how many `z` variables are greater than 0.9? Less than 0.1? Greater than 0.9 or less than 0.1?

```
length(randu$z[randu$z>0.9])
#> [1] 29
length(randu$z[randu$z<0.1])
#> [1] 37
```

19.1 R packages

From Wikipedia, the free encyclopedia, and fount of all knowledge

R packages are extensions to the R statistical programming language. R packages contain code, data, and documentation in a standardised collection format that can be installed by users of R, typically via a centralised software repository such as CRAN (the Comprehensive R Archive Network).

The large number of packages available for R, and the ease of installing and using them, has been cited as a major factor in driving the widespread adoption of the language in data science.

19.2 You can install the latest released version from CRAN with:

```
install.packages("dplyr")
```

19.3 In RStudio

Installing Packages

- Open RStudio. …
- In the lower-right pane of RStudio, select the Packages tab and the Install button.
- Type the name of the packages to be installed in the "Packages (separate multiple packages with a space or comma):" box. …
- Press Install .

19.4 Check this out

dplyr link

https://dplyr.tidyverse.org/

Chapter 20

DPLYR

20.1 Introduction

For more help **PLEASE** check out Introduction to dplyr introducing the key functionality of the dplyr package.

> Your life is about to change. For the better, even.

20.2 A Neat Resource

- RStudio's Data Wrangling Cheat Sheet for dplyr and tidyr

20.3 Single table verbs

`dplyr` aims to provide a function for each basic verb of data manipulation. These verbs can be organised into three categories based on the component of the dataset that they work with:

Rows:

- `filter()` chooses rows based on column values.
- `slice()` chooses rows based on location.
- `arrange()` changes the order of the rows.

Columns:

- `select()` changes whether or not a column is included.
- `rename()` changes the name of columns. `mutate()` changes the values of columns and creates new columns.
- `relocate()` changes the order of the columns. Groups of rows:
- `summarise()` collapses a group into a single row. It's not that useful until we learn the `group_by()` verb below.

20.4 The pipe

All of the `dplyr` functions take a data frame (or tibble) as the first argument. Rather than forcing the user to either save intermediate objects or nest functions, dplyr provides the `%>%` operator from magrittr. x %>% f(y) turns into f(x, y) so the result from one step is then "piped" into the next step. You can use the pipe to rewrite multiple operations that you can read left-to-right, top-to-bottom (reading the pipe operator as "then").

> What is this: `%>%`?

20.5 Loading `dplyr` and the `starwars` dataset

```
# You should already have done this but you'll need it
install.packages("dplyr")
```

```
library(dplyr)

starwars %>%
  filter(species == "Droid")
#> # A tibble: 6 x 14
#>   name   height  mass hair_color skin_color  eye_color
#>   <chr>   <int> <dbl> <chr>      <chr>       <chr>
#> 1 C-3PO     167    75 <NA>       gold        yellow
#> 2 R2-D2      96    32 <NA>       white, blue red
#> 3 R5-D4      97    32 <NA>       white, red  red
#> 4 IG-88     200   140 none       metal       red
#> 5 R4-P17     96    NA none       silver, red red, blue
#> 6 BB8        NA    NA none       none        black
#> # ... with 8 more variables: birth_year <dbl>, sex <chr>,
#> #   gender <chr>, homeworld <chr>, species <chr>,
#> #   films <list>, vehicles <list>, starships <list>
```

```
starwars %>%
  select(name, ends_with("color"))
#> # A tibble: 87 x 4
#>    name               hair_color    skin_color  eye_color
#>    <chr>              <chr>         <chr>       <chr>
#>  1 Luke Skywalker     blond         fair        blue
#>  2 C-3PO              <NA>          gold        yellow
#>  3 R2-D2              <NA>          white, blue red
#>  4 Darth Vader        none          white       yellow
#>  5 Leia Organa        brown         light       brown
#>  6 Owen Lars          brown, grey   light       blue
#>  7 Beru Whitesun lars brown         light       blue
#>  8 R5-D4              <NA>          white, red  red
#>  9 Biggs Darklighter  black         light       brown
#> 10 Obi-Wan Kenobi     auburn, white fair        blue-gray
#> # ... with 77 more rows
```

```
starwars %>%
  mutate(name, bmi = mass / ((height / 100)  ^ 2)) %>%
  select(name:mass, bmi)
#> # A tibble: 87 x 4
#>    name               height  mass   bmi
#>    <chr>               <int> <dbl> <dbl>
#>  1 Luke Skywalker        172    77  26.0
#>  2 C-3PO                 167    75  26.9
#>  3 R2-D2                  96    32  34.7
#>  4 Darth Vader           202   136  33.3
#>  5 Leia Organa           150    49  21.8
#>  6 Owen Lars             178   120  37.9
#>  7 Beru Whitesun lars    165    75  27.5
#>  8 R5-D4                  97    32  34.0
#>  9 Biggs Darklighter     183    84  25.1
#> 10 Obi-Wan Kenobi        182    77  23.2
#> # ... with 77 more rows
```

```
starwars %>%
  arrange(desc(mass))
#> # A tibble: 87 x 14
#>    name       height  mass hair_color skin_color  eye_color
#>    <chr>       <int> <dbl> <chr>      <chr>       <chr>
#>  1 Jabba Des~    175  1358 <NA>       green-tan,~ orange
#>  2 Grievous      216   159 none       brown, whi~ green, ye~
#>  3 IG-88         200   140 none       metal       red
#>  4 Darth Vad~    202   136 none        white       yellow
#>  5 Tarfful       234   136 brown       brown       blue
#>  6 Owen Lars     178   120 brown, gr~ light        blue
#>  7 Bossk         190   113 none        green       red
#>  8 Chewbacca     228   112 brown       unknown     blue
#>  9 Jek Tono ~    180   110 brown       fair        blue
#> 10 Dexter Je~    198   102 none        brown       yellow
#> # ... with 77 more rows, and 8 more variables:
#> #   birth_year <dbl>, sex <chr>, gender <chr>,
#> #   homeworld <chr>, species <chr>, films <list>,
#> #   vehicles <list>, starships <list>

starwars %>%
  group_by(species) %>%
  summarise(
    n = n(),
    mass = mean(mass, na.rm = TRUE)
  ) %>%
  filter(
    n > 1,
    mass > 50
  )
#> # A tibble: 8 x 3
#>   species       n  mass
#>   <chr>     <int> <dbl>
#> 1 Droid         6  69.8
#> 2 Gungan        3  74
#> 3 Human        35  82.8
#> 4 Kaminoan      2  88
#> 5 Mirialan      2  53.1
#> 6 Twi'lek       2  55
#> 7 Wookiee       2 124
#> 8 Zabrak        2  80
```

Chapter 21

`starwars` Excercises

Please use the `starwars` dataset from the `dplyr` package to answer the following questions:

1. How may humans are in this dataset?
2. How many characters are taller than 89 cm?
3. How many characters are taller than 37 inches?
4. How many characters are taller than 37 inches and weigh more than 55 pounds?
5. How many characters are not human or droid?
6. How many characters are not human or droid and are taller than 47 inches?
7. Which species has the most individuals included in this data set?
8. Which species has the tallest individuals on average?
9. What is the tallest individual for each species?
10. Calculate the BMI for each individual and determine which individual has the highest BMI. Use the formula `bmi = mass/((height/100)^2)` to calculate bmi.
11. Which homeworld has the most individuals included in this data set?
12. Which homeworld has the tallest individuals on average?
13. What is the tallest individual for each eye color?

Chapter 22

Loading dplyr and the nycflights13 dataset

```r
# load packages
suppressMessages(library(dplyr))
library(nycflights13)

# print the flights dataset from nycflights13
head(flights)
#> # A tibble: 6 x 19
#>    year month   day dep_time sched_dep_time dep_delay
#>   <int> <int> <int>    <int>          <int>     <dbl>
#> 1  2013     1     1      517            515         2
#> 2  2013     1     1      533            529         4
#> 3  2013     1     1      542            540         2
#> 4  2013     1     1      544            545        -1
#> 5  2013     1     1      554            600        -6
#> 6  2013     1     1      554            558        -4
#> # ... with 13 more variables: arr_time <int>,
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

22.1 Choosing columns: select, rename

```
# besides just using select() to pick columns...
flights %>% select(carrier, flight)
#> # A tibble: 336,776 x 2
#>    carrier flight
#>    <chr>    <int>
#>  1 UA        1545
#>  2 UA        1714
#>  3 AA        1141
#>  4 B6         725
#>  5 DL         461
#>  6 UA        1696
#>  7 B6         507
#>  8 EV        5708
#>  9 B6          79
#> 10 AA         301
#> # ... with 336,766 more rows

# ...you can use the minus sign to hide columns
flights %>% select(-month, -day)
#> # A tibble: 336,776 x 17
#>     year dep_time sched_dep_time dep_delay arr_time
#>    <int>    <int>          <int>     <dbl>    <int>
#>  1  2013      517            515         2      830
#>  2  2013      533            529         4      850
#>  3  2013      542            540         2      923
#>  4  2013      544            545        -1     1004
#>  5  2013      554            600        -6      812
#>  6  2013      554            558        -4      740
#>  7  2013      555            600        -5      913
#>  8  2013      557            600        -3      709
#>  9  2013      557            600        -3      838
#> 10  2013      558            600        -2      753
#> # ... with 336,766 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

```
# hide a range of columns
flights %>% select(-(dep_time:arr_delay))

# hide any column with a matching name
```

```
flights %>% select(-contains("time"))
```

```
# pick columns using a character vector of column names
cols <- c("carrier", "flight", "tailnum")
flights %>% select(one_of(cols))
#> # A tibble: 336,776 x 3
#>    carrier flight tailnum
#>    <chr>    <int> <chr>
#>  1 UA        1545 N14228
#>  2 UA        1714 N24211
#>  3 AA        1141 N619AA
#>  4 B6         725 N804JB
#>  5 DL         461 N668DN
#>  6 UA        1696 N39463
#>  7 B6         507 N516JB
#>  8 EV        5708 N829AS
#>  9 B6          79 N593JB
#> 10 AA         301 N3ALAA
#> # ... with 336,766 more rows
```

```
# select() can be used to rename columns, though all columns not mentioned are dropped
flights %>% select(tail = tailnum)
#> # A tibble: 336,776 x 1
#>    tail
#>    <chr>
#>  1 N14228
#>  2 N24211
#>  3 N619AA
#>  4 N804JB
#>  5 N668DN
#>  6 N39463
#>  7 N516JB
#>  8 N829AS
#>  9 N593JB
#> 10 N3ALAA
#> # ... with 336,766 more rows

# rename() does the same thing, except all columns not mentioned are kept
flights %>% rename(tail = tailnum)
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      517            515         2
#>  2  2013     1     1      533            529         4
#>  3  2013     1     1      542            540         2
#>  4  2013     1     1      544            545        -1
#>  5  2013     1     1      554            600        -6
#>  6  2013     1     1      554            558        -4
#>  7  2013     1     1      555            600        -5
#>  8  2013     1     1      557            600        -3
#>  9  2013     1     1      557            600        -3
#> 10  2013     1     1      558            600        -2
#> # ... with 336,766 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tail <chr>, origin <chr>,
#> #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

22.2 Choosing rows: filter, between, slice, sample_n, top_n, distinct

```
# filter() supports the use of multiple conditions
flights %>% filter(dep_time >= 600, dep_time <= 605)
#> # A tibble: 2,460 x 19
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      600            600         0
#>  2  2013     1     1      600            600         0
#>  3  2013     1     1      601            600         1
#>  4  2013     1     1      602            610        -8
#>  5  2013     1     1      602            605        -3
#>  6  2013     1     2      600            600         0
#>  7  2013     1     2      600            605        -5
#>  8  2013     1     2      600            600         0
#>  9  2013     1     2      600            600         0
#> 10  2013     1     2      600            600         0
#> # ... with 2,450 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

```
# between() is a concise alternative for determing if numeric values fall in a range
flights %>% filter(between(dep_time, 600, 605))
```

```
# side note: is.na() can also be useful when filtering
flights %>% filter(!is.na(dep_time))
```

```
# slice() filters rows by position
flights %>% slice(1000:1005)
#> # A tibble: 6 x 19
#>    year month   day dep_time sched_dep_time dep_delay
#>   <int> <int> <int>    <int>          <int>     <dbl>
#> 1  2013     1     2      809            810        -1
#> 2  2013     1     2      810            800        10
#> 3  2013     1     2      811            815        -4
#> 4  2013     1     2      811            815        -4
#> 5  2013     1     2      811            820        -9
#> 6  2013     1     2      815            815         0
#> # ... with 13 more variables: arr_time <int>,
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>

# keep the first three rows within each group
flights %>% group_by(month, day) %>% slice(1:3)
#> # A tibble: 1,095 x 19
#> # Groups:   month, day [365]
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      517            515         2
#>  2  2013     1     1      533            529         4
#>  3  2013     1     1      542            540         2
#>  4  2013     1     2       42           2359        43
#>  5  2013     1     2      126           2250       156
#>  6  2013     1     2      458            500        -2
#>  7  2013     1     3       32           2359        33
#>  8  2013     1     3       50           2145       185
#>  9  2013     1     3      235           2359       156
#> 10  2013     1     4       25           2359        26
#> # ... with 1,085 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>

# sample three rows from each group
flights %>% group_by(month, day) %>% sample_n(3)
```

```
#> # A tibble: 1,095 x 19
#> # Groups:   month, day [365]
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      811            630       101
#>  2  2013     1     1      858            900        -2
#>  3  2013     1     1      905            905         0
#>  4  2013     1     2     1341           1200       101
#>  5  2013     1     2     1459           1500        -1
#>  6  2013     1     2     1833           1745        48
#>  7  2013     1     3       NA            920        NA
#>  8  2013     1     3     1549           1545         4
#>  9  2013     1     3     1737           1725        12
#> 10  2013     1     4     1240           1200        40
#> # ... with 1,085 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>

# keep three rows from each group with the top dep_delay
flights %>% group_by(month, day) %>% top_n(3, dep_delay)
#> # A tibble: 1,108 x 19
#> # Groups:   month, day [365]
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      848           1835       853
#>  2  2013     1     1     1815           1325       290
#>  3  2013     1     1     2343           1724       379
#>  4  2013     1     2     1412            838       334
#>  5  2013     1     2     1607           1030       337
#>  6  2013     1     2     2131           1512       379
#>  7  2013     1     3     2008           1540       268
#>  8  2013     1     3     2012           1600       252
#>  9  2013     1     3     2056           1605       291
#> 10  2013     1     4     2058           1730       208
#> # ... with 1,098 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>

# also sort by dep_delay within each group
```

```
flights %>% group_by(month, day) %>% top_n(3, dep_delay) %>% arrange(desc(dep_delay))
#> # A tibble: 1,108 x 19
#> # Groups:   month, day [365]
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     9      641            900      1301
#>  2  2013     6    15     1432           1935      1137
#>  3  2013     1    10     1121           1635      1126
#>  4  2013     9    20     1139           1845      1014
#>  5  2013     7    22      845           1600      1005
#>  6  2013     4    10     1100           1900       960
#>  7  2013     3    17     2321            810       911
#>  8  2013     6    27      959           1900       899
#>  9  2013     7    22     2257            759       898
#> 10  2013    12     5      756           1700       896
#> # ... with 1,098 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

```
# unique rows can be identified using unique() from base R
flights %>% select(origin, dest) %>% unique()
#> # A tibble: 224 x 2
#>    origin dest
#>    <chr>  <chr>
#>  1 EWR    IAH
#>  2 LGA    IAH
#>  3 JFK    MIA
#>  4 JFK    BQN
#>  5 LGA    ATL
#>  6 EWR    ORD
#>  7 EWR    FLL
#>  8 LGA    IAD
#>  9 JFK    MCO
#> 10 LGA    ORD
#> # ... with 214 more rows
```

```
# dplyr provides an alternative that is more "efficient"
flights %>% select(origin, dest) %>% distinct()

# side note: when chaining, you don't have to include the parentheses if there are no arguments
flights %>% select(origin, dest) %>% distinct
```

22.3 Excercies

Using the nycflights13 dataset and the dplyr package, answer these questions. Some answers are given in square brackets for you to check your answers.

1. How many flights in Sept were late departing flights? [7815]
2. How many flights in Sept were late departing flights that originated at JFK airport? [2649]
3. How many flights in Sept were late departing flights with an origin of JFK airport and had an destination of anywhere except MIA? [2572]
4. Which carrier had the most flights in this data set? [UA with 58665]
5. Which destination had the most flights in this data set? [ORD with 17283]
6. Which destination had the most flights with departure delays of greater than 60 minutes in this data set? [ORD with 1480]
7. What was the longest arrival delay in this dataset? [1272]
8. Which carrier in September had the most late departing flights? [UA with 1559]
9. Create a variable called total.annoyance which arrival delay plus the departure delay for each flight.
10. Which carrier with more than 10 flights in September had greatest % late departing flights?

22.4 Adding new variables: mutate, transmute, add__rownames

```
# mutate() creates a new variable (and keeps all existing variables)
flights %>% mutate(speed = distance/air_time*60)
#> # A tibble: 336,776 x 20
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      517            515         2
#>  2  2013     1     1      533            529         4
#>  3  2013     1     1      542            540         2
#>  4  2013     1     1      544            545        -1
#>  5  2013     1     1      554            600        -6
#>  6  2013     1     1      554            558        -4
#>  7  2013     1     1      555            600        -5
#>  8  2013     1     1      557            600        -3
#>  9  2013     1     1      557            600        -3
#> 10  2013     1     1      558            600        -2
#> # ... with 336,766 more rows, and 14 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>, speed <dbl>

# transmute() only keeps the new variables
flights %>% transmute(speed = distance/air_time*60)
#> # A tibble: 336,776 x 1
#>    speed
#>    <dbl>
#>  1  370.
#>  2  374.
#>  3  408.
#>  4  517.
#>  5  394.
#>  6  288.
#>  7  404.
#>  8  259.
#>  9  405.
#> 10  319.
#> # ... with 336,766 more rows
```

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

# add_rownames() turns row names into an explicit variable
mtcars %>% add_rownames("model") %>% head()
#> Warning: `add_rownames()` was deprecated in dplyr 1.0.0.
#> Please use `tibble::rownames_to_column()` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
#> # A tibble: 6 x 12
#>   model         mpg   cyl  disp    hp  drat    wt  qsec    vs
#>   <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 Mazda RX4    21       6   160   110  3.9   2.62  16.5     0
#> 2 Mazda RX4~   21       6   160   110  3.9   2.88  17.0     0
#> 3 Datsun 710   22.8     4   108    93  3.85  2.32  18.6     1
#> 4 Hornet 4 ~   21.4     6   258   110  3.08  3.22  19.4     1
#> 5 Hornet Sp~   18.7     8   360   175  3.15  3.44  17.0     0
#> 6 Valiant      18.1     6   225   105  2.76  3.46  20.2     1
#> # ... with 3 more variables: am <dbl>, gear <dbl>,
#> #   carb <dbl>

# side note: dplyr no longer prints row names (ever) for local data frames
mtcars %>% tbl_df()
#> Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
#> Please use `tibble::as_tibble()` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated
#> # A tibble: 32 x 11
#>      mpg   cyl  disp    hp  drat    wt  qsec    vs    am
#>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#>  1  21       6  160    110  3.9   2.62  16.5     0     1
#>  2  21       6  160    110  3.9   2.88  17.0     0     1
#>  3  22.8     4  108     93  3.85  2.32  18.6     1     1
#>  4  21.4     6  258    110  3.08  3.22  19.4     1     0
#>  5  18.7     8  360    175  3.15  3.44  17.0     0     0
#>  6  18.1     6  225    105  2.76  3.46  20.2     1     0
#>  7  14.3     8  360    245  3.21  3.57  15.8     0     0
#>  8  24.4     4  147.    62  3.69  3.19  20       1     0
#>  9  22.8     4  141.    95  3.92  3.15  22.9     1     0
#> 10  19.2     6  168.   123  3.92  3.44  18.3     1     0
#> # ... with 22 more rows, and 2 more variables: gear <dbl>,
#> #   carb <dbl>
```

Chapter 23

Grouping and counting: summarise, tally, count, group_size, n_groups, ungroup

```
# summarise() can be used to count the number of rows in each group
flights %>% group_by(month) %>% summarise(cnt = n())
#> # A tibble: 12 x 2
#>    month   cnt
#>    <int> <int>
#>  1     1 27004
#>  2     2 24951
#>  3     3 28834
#>  4     4 28330
#>  5     5 28796
#>  6     6 28243
#>  7     7 29425
#>  8     8 29327
#>  9     9 27574
#> 10    10 28889
#> 11    11 27268
#> 12    12 28135
```

```
# tally() and count() can do this more concisely
flights %>% group_by(month) %>% tally()
flights %>% count(month)
```

```
# you can sort by the count
flights %>% group_by(month) %>% summarise(cnt = n()) %>% arrange(desc(cnt))
#> # A tibble: 12 x 2
#>    month   cnt
#>    <int> <int>
#>  1     7 29425
#>  2     8 29327
#>  3    10 28889
#>  4     3 28834
#>  5     5 28796
#>  6     4 28330
#>  7     6 28243
#>  8    12 28135
#>  9     9 27574
#> 10    11 27268
#> 11     1 27004
#> 12     2 24951
```

```
# tally() and count() have a sort parameter for this purpose
flights %>% group_by(month) %>% tally(sort=TRUE)
flights %>% count(month, sort=TRUE)
```

```
# you can sum over a specific variable instead of simply counting rows
flights %>% group_by(month) %>% summarise(dist = sum(distance))
#> # A tibble: 12 x 2
#>    month     dist
#>    <int>    <dbl>
#>  1     1 27188805
#>  2     2 24975509
#>  3     3 29179636
#>  4     4 29427294
#>  5     5 29974128
#>  6     6 29856388
#>  7     7 31149199
#>  8     8 31149334
#>  9     9 28711426
#> 10    10 30012086
#> 11    11 28639718
#> 12    12 29954084
```

```
# tally() and count() have a wt parameter for this purpose
flights %>% group_by(month) %>% tally(wt = distance)
flights %>% count(month, wt = distance)
```

```
# group_size() returns the counts as a vector
flights %>% group_by(month) %>% group_size()
#>  [1] 27004 24951 28834 28330 28796 28243 29425 29327 27574
#> [10] 28889 27268 28135

# n_groups() simply reports the number of groups
flights %>% group_by(month) %>% n_groups()
#> [1] 12
```

```
# group by two variables, summarise, arrange (output is possibly confusing)
flights %>% group_by(month, day) %>% summarise(cnt = n()) %>% arrange(desc(cnt)) %>% print(n = 40
#> `summarise()` has grouped output by 'month'. You can override using the `.groups` argument.
#> # A tibble: 365 x 3
#> # Groups:   month [12]
#>    month   day   cnt
#>    <int> <int> <int>
#>  1    11    27  1014
#>  2     7    11  1006
#>  3     7     8  1004
#>  4     7    10  1004
#>  5    12     2  1004
#>  6     7    18  1003
#>  7     7    25  1003
#>  8     7    12  1002
#>  9     7     9  1001
#> 10     7    17  1001
#> 11     7    31  1001
#> 12     8     7  1001
#> 13     8     8  1001
#> 14     8    12  1001
#> 15     7    22  1000
#> 16     7    24  1000
#> 17     8     1  1000
#> 18     8     5  1000
#> 19     8    15  1000
#> 20    11    21  1000
#> 21     7    15   999
#> 22     7    19   999
#> 23     7    26   999
#> 24     7    29   999
#> 25     8     2   999
#> 26     8     9   999
#> 27    11    22   999
#> 28     8    16   998
#> 29     7    23   997
#> 30     7    30   997
#> 31     8    14   997
#> 32     7    16   996
#> 33     8     6   996
#> 34     8    19   996
#> 35     9    13   996
#> 36     9    26   996
#> 37     9    27   996
#> 38     4    15   995
#> 39     6    20   995
#> 40     6    26   995
#> # ... with 325 more rows

# ungroup() before arranging to arrange across all groups
flights %>% group_by(month, day) %>% summarise(cnt = n()) %>% ungroup() %>% arrange(des
#> `summarise()` has grouped output by 'month'. You can override using the `.groups` a
#> # A tibble: 365 x 3
#>    month   day   cnt
#>    <int> <int> <int>
#>  1    11    27  1014
#>  2     7    11  1006
#>  3     7     8  1004
#>  4     7    10  1004
#>  5    12     2  1004
#>  6     7    18  1003
#>  7     7    25  1003
#>  8     7    12  1002
#>  9     7     9  1001
#> 10     7    17  1001
#> # ... with 355 more rows
```

Chapter 24

Creating data frames: data_frame

`data_frame()` is a better way than `data.frame()` for creating data frames. Benefits of `data_frame()`:

- You can use previously defined columns to compute new columns.
- It never coerces column types.
- It never munges column names.
- It never adds row names.
- It only recycles length 1 input.
- It returns a local data frame (a tbl_df).

```
# data_frame() example
data_frame(a = 1:6, b = a*2, c = 'string', 'd+e' = 1) %>% glimpse()
#> Warning: `data_frame()` was deprecated in tibble 1.1.0.
#> Please use `tibble()` instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
#> Rows: 6
#> Columns: 4
#> $ a     <int> 1, 2, 3, 4, 5, 6
#> $ b     <dbl> 2, 4, 6, 8, 10, 12
#> $ c     <chr> "string", "string", "string", "string", "str~
#> $ `d+e` <dbl> 1, 1, 1, 1, 1, 1

# data.frame() example
data.frame(a = 1:6, c = 'string', 'd+e' = 1) %>% glimpse()
#> Rows: 6
#> Columns: 3
#> $ a   <int> 1, 2, 3, 4, 5, 6
#> $ c   <chr> "string", "string", "string", "string", "strin~
#> $ d.e <dbl> 1, 1, 1, 1, 1, 1
```

Chapter 25

Joining (merging) tables: left_join, right_join, inner_join, full_join, semi_join, anti_join

```
# create two simple data frames
(a <- data_frame(color = c("green","yellow","red"), num = 1:3))
#> # A tibble: 3 x 2
#>   color    num
#>   <chr>  <int>
#> 1 green      1
#> 2 yellow     2
#> 3 red        3
(b <- data_frame(color = c("green","yellow","pink"), size = c("S","M","L")))
#> # A tibble: 3 x 2
#>   color  size
#>   <chr>  <chr>
#> 1 green  S
#> 2 yellow M
#> 3 pink   L

# only include observations found in both "a" and "b" (automatically joins on variables that appe
inner_join(a, b)
#> Joining, by = "color"
#> # A tibble: 2 x 3
#>   color    num size
#>   <chr>  <int> <chr>
#> 1 green      1 S
#> 2 yellow     2 M

# include observations found in either "a" or "b"
full_join(a, b)
#> Joining, by = "color"
#> # A tibble: 4 x 3
#>   color    num size
#>   <chr>  <int> <chr>
#> 1 green      1 S
#> 2 yellow     2 M
#> 3 red        3 <NA>
#> 4 pink      NA L

# include all observations found in "a"
left_join(a, b)
#> Joining, by = "color"
#> # A tibble: 3 x 3
#>   color    num size
#>   <chr>  <int> <chr>
#> 1 green      1 S
#> 2 yellow     2 M
#> 3 red        3 <NA>

# include all observations found in "b"
right_join(a, b)
#> Joining, by = "color"
#> # A tibble: 3 x 3
#>   color    num size
#>   <chr>  <int> <chr>
#> 1 green      1 S
#> 2 yellow     2 M
#> 3 pink      NA L

# right_join(a, b) is identical to left_join(b, a) except for column ordering
left_join(b, a)
#> Joining, by = "color"
#> # A tibble: 3 x 3
#>   color  size    num
#>   <chr>  <chr> <int>
#> 1 green  S         1
#> 2 yellow M         2
#> 3 pink   L        NA
```

```
# filter "a" to only show observations that match "b"
semi_join(a, b)
#> Joining, by = "color"
#> # A tibble: 2 x 2
#>   color    num
#>   <chr>  <int>
#> 1 green      1
#> 2 yellow     2

# filter "a" to only show observations that don't match "b"
anti_join(a, b)
#> Joining, by = "color"
#> # A tibble: 1 x 2
#>   color   num
#>   <chr> <int>
#> 1 red       3
```

```
# sometimes matching variables don't have identical names
b <- b %>% rename(col = color)

# specify that the join should occur by matching "color" in "a" with "col" in "b"
inner_join(a, b, by=c("color" = "col"))
#> # A tibble: 2 x 3
#>   color    num size
#>   <chr>  <int> <chr>
#> 1 green      1 S
#> 2 yellow     2 M
```

Chapter 26

Viewing more output: print, View

```
# specify that you want to see more rows
flights %>% print(n = 15)
#> # A tibble: 336,776 x 19
#>      year month   day dep_time sched_dep_time dep_delay
#>     <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      517            515         2
#>  2  2013     1     1      533            529         4
#>  3  2013     1     1      542            540         2
#>  4  2013     1     1      544            545        -1
#>  5  2013     1     1      554            600        -6
#>  6  2013     1     1      554            558        -4
#>  7  2013     1     1      555            600        -5
#>  8  2013     1     1      557            600        -3
#>  9  2013     1     1      557            600        -3
#> 10  2013     1     1      558            600        -2
#> 11  2013     1     1      558            600        -2
#> 12  2013     1     1      558            600        -2
#> 13  2013     1     1      558            600        -2
#> 14  2013     1     1      558            600        -2
#> 15  2013     1     1      559            600        -1
#> # ... with 336,761 more rows, and 13 more variables:
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>,
#> #   time_hour <dttm>
```

```
# specify that you want to see ALL rows (don't run this!)
flights %>% print(n = Inf)
```

```
# specify that you want to see all columns
flights %>% print(width = Inf)
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay
#>    <int> <int> <int>    <int>          <int>     <dbl>
#>  1  2013     1     1      517            515         2
#>  2  2013     1     1      533            529         4
#>  3  2013     1     1      542            540         2
#>  4  2013     1     1      544            545        -1
#>  5  2013     1     1      554            600        -6
#>  6  2013     1     1      554            558        -4
#>  7  2013     1     1      555            600        -5
#>  8  2013     1     1      557            600        -3
#>  9  2013     1     1      557            600        -3
#> 10  2013     1     1      558            600        -2
#>    arr_time sched_arr_time arr_delay carrier flight tailnum
#>       <int>          <int>     <dbl> <chr>    <int> <chr>
#>  1      830            819        11 UA        1545 N14228
#>  2      850            830        20 UA        1714 N24211
#>  3      923            850        33 AA        1141 N619AA
#>  4     1004           1022       -18 B6         725 N804JB
#>  5      812            837       -25 DL         461 N668DN
#>  6      740            728        12 UA        1696 N39463
#>  7      913            854        19 B6         507 N516JB
#>  8      709            723       -14 EV        5708 N829AS
#>  9      838            846        -8 B6          79 N593JB
#> 10      753            745         8 AA         301 N3ALAA
#>    origin dest  air_time distance  hour minute
#>    <chr>  <chr>    <dbl>    <dbl> <dbl>  <dbl>
#>  1 EWR    IAH        227     1400     5     15
#>  2 LGA    IAH        227     1416     5     29
#>  3 JFK    MIA        160     1089     5     40
#>  4 JFK    BQN        183     1576     5     45
#>  5 LGA    ATL        116      762     6      0
#>  6 EWR    ORD        150      719     5     58
#>  7 EWR    FLL        158     1065     6      0
#>  8 LGA    IAD         53      229     6      0
#>  9 JFK    MCO        140      944     6      0
#> 10 LGA    ORD        138      733     6      0
#>    time_hour
#>    <dttm>
#>  1 2013-01-01 05:00:00
#>  2 2013-01-01 05:00:00
#>  3 2013-01-01 05:00:00
#>  4 2013-01-01 05:00:00
#>  5 2013-01-01 06:00:00
#>  6 2013-01-01 05:00:00
#>  7 2013-01-01 06:00:00
#>  8 2013-01-01 06:00:00
#>  9 2013-01-01 06:00:00
#> 10 2013-01-01 06:00:00
#> # ... with 336,766 more rows
```

```
# show up to 1000 rows and all columns
flights %>% View()

# set option to see all columns and fewer rows
options(dplyr.width = Inf, dplyr.print_min = 6)

# reset options (or just close R)
options(dplyr.width = NULL, dplyr.print_min = 10)
```

Chapter 27

DPLYR Excercies

Using the nycflights13 dataset and the dplyr package, answer these questions. Some answers are given in square brackets for you to check your answers.

1. How many flights in Sept were late departing flights? [7815]
2. How many flights in Sept were late departing flights that originated at JFK airport? [2649]
3. How many flights in Sept were late departing flights with an origin of JFK airport and had an destination of anywhere except MIA? [2572]
4. Which carrier had the most flights in this data set? [UA with 58665]
5. Which destination had the most flights in this data set? [ORD with 17283]
6. Which destination had the most flights with departure delays of greater than 60 minutes in this data set? [ORD with 1480]
7. What was the longest arrival delay in this dataset? [1272]
8. Which carrier in September had the most late departing flights? [UA with 1559]
9. Create a variable called total.annoyance which arrival delay plus the departure delay for each flight.
10. Which carrier with more than 10 flights in September had greatest % late departing flights?