Sports Data Analysis and Visualization

Code, data, visuals and the Tidyverse for journalists and other storytellers

By Matt Waite

July 29, 2019

Contents

1 Throwing cold water on hot takes **7**
1.1 Requirements and Conventions 8
1.2 About this book 8

2 The very basics **11**
2.1 Adding libraries, part 1 13
2.2 Adding libraries, part 2 14
2.3 Notebooks 14

3 Data, structures and types **17**
3.1 Rows and columns 17
3.2 Types 19
3.3 A simple way to get data 19
3.4 Cleaning the data 20

4 Aggregates **27**
4.1 Basic data analysis: Group By and Count 28
4.2 Other aggregates: Mean and median 31
4.3 Even more aggregates 33

5 Mutating data **35**
5.1 A more complex example 37

6 Filters and selections **41**
6.1 Selecting data to make it easier to read 44
6.2 Using conditional filters to set limits 44
6.3 Top list 46

7 Transforming data **49**
7.1 Making long data wide 51
7.2 Why this matters 52

8 Simulations **55**
8.1 Cold streaks 57

9 Correlations and regression **59**
9.1 A more predictive example 63

10 Multiple regression **67**

11 Residuals **81**
11.1 Penalties 85

12 Z scores **89**
12.1 Calculating a Z score in R 89

13 Intro to ggplot **93**
13.1 The bar chart 95
13.2 Scales 98
13.3 Styling 99
13.4 One last trick: coord flip 101

14 Stacked bar charts **103**

15 Waffle charts **107**
15.1 Waffle Irons 110

16 Line charts **113**
16.1 This is too simple. 115
16.2 But what if I wanted to add a lot of lines. 117

17 Step charts **121**

18 Ridge charts **127**

19 Lollipop charts **133**

20 Scatterplots **139**

21 Facet wraps **145**
21.1 Facet grid vs facet wraps 148
21.2 Other types 150

22 Tables **153**

23 Bubble charts **175**

24 Circular bar plots **185**

25 Intro to rvest **191**
25.1 A slightly more complicated example 192
25.2 An even more complicated example 193

26 Advanced rvest **251**
26.1 One last bit 254

27 Annotations **257**

28 Finishing touches, part 1 **261**
28.1 Graphics vs visual stories 263
28.2 Getting ggplot closer to output 263

29 Finishing Touches 2 **269**

30 Plotly **273**
30.1 Publishing using Plotly 277

31 Clustering **279**
31.1 Advanced metrics 286

32 Rtweet and Text Analysis **291**
32.1 Prerequisites 292
32.2 Verifying your account to access Twitter data 294
32.3 The data used here 296
32.4 Data Exploration 297
32.5 Cleaning data for analysis 299
32.6 Number of tweets throughout the game 300
32.7 Tidying the text data for analysis, applying the sentiment scores 301

33 Arranging multiple plots together **309**

34 Assignments **313**

35 Appendix **321**
35.1 How to get help in this class 321

Chapter 1

Throwing cold water on hot takes

The 2018 season started out disastrously for the Nebraska Cornhuskers. The first game against a probably overmatched opponent? Called on account of an epic thunderstorm that plowed right over Memorial Stadium. The next game? Loss. The one following? Loss. The next four? All losses, after the fanbase was whipped into a hopeful frenzy by the hiring of Scott Frost, national title winning quarterback turned hot young coach come back home to save a mythical football program from the mediocrity it found itself mired in.

All that excitement lay in tatters.

On sports talk radio, on the sports pages and across social media and cafe conversations, one topic kept coming up again and again to explain why the team was struggling: Penalties. The team was just committing too many of them. In fact, six games and no wins into the season, they were dead last in the FBS penalty yards.

Worse yet for this line of reasoning? Nebraska won game 7, against Minnesota, committing only six penalties for 43 yards, just about half their average over the season. Then they won game 8 against FCS patsy Bethune Cookman, committing only five penalties for 35 yards. That's a whopping 75 yards less than when they were losing. See? Cut the penalties, win games screamed the radio show callers.

The problem? It's not true. Penalties might matter for a single drive. They may even throw a single game. But if you look at every top-level college football team since 2009, the number of penalty yards the team racks up means absolutely nothing to the total number of points they score. There's no relationship between them. Penalty yards have no discernible influence on points beyond just random noise.

Put this another way: If you were Scott Frost, and a major college football program was paying you $5 million a year to make your team better, what should you focus on in practice? If you had growled at some press conference that you're going to work on penalties in practice until your team stops committing them, the results you'd get from all that wasted practice time would be impossible to separate from just random chance. You very well may reduce your penalty yards and still lose.

How do I know this? Simple statistics.

That's one of the three pillars of this book: Simple stats. The three pillars are:

1. Simple, easy to understand statistics …
2. … extracted using simple code …
3. … visualized simply to reveal new and interesting things in sports.

Do you need to be a math whiz to read this book? No. I'm not one either. What we're going to look at is pretty basic, but that's also why it's so powerful.

Do you need to be a computer science major to write code? Nope. I'm not one of those either. But anyone can think logically, and write simple code that is repeatable and replicable.

Do you need to be an artist to create compelling visuals? I think you see where this is going. No. I can barely draw stick figures, but I've been paid to make graphics in my career. With a little graphic design know how, you can create publication worthy graphics with code.

1.1 Requirements and Conventions

This book is all in the R statistical language. To follow along, you'll do the following:

1. Install the R language on your computer. Go to the R Project website, click download R and select a mirror closest to your location. Then download the version for your computer.

2. Install R Studio Desktop. The free version is great.

Going forward, you'll see passages like this:

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

Don't do it now, but that is code that you'll need to run in your R Studio. When you see that, you'll know what to do.

1.2 About this book

This book is the collection of class materials for the author's Sports Data Analysis and Visualization class at the University of Nebraska-Lincoln's College of

Journalism and Mass Communications. There's some things you should know about it:

- It is free for students.
- The topics will remain the same but the text is going to be constantly tinkered with.
- What is the work of the author is copyright Matt Waite 2019.
- The text is Attribution-NonCommercial-ShareAlike 4.0 International Creative Commons licensed. That means you can share it and change it, but only if you share your changes with the same license and it cannot be used for commercial purposes. I'm not making money on this so you can't either.

- As such, the whole book – authored in Bookdown – is open sourced on Github. Pull requests welcomed!

Chapter 2

The very basics

R is a programming language, one specifically geared toward statistical analysis. Like all programming languages, it has certain built-in functions and you can interact with it in multiple ways. The first, and most basic, is the console.

RStudio File Edit Code View Plots Session Build Debug Profile Tool
Go to file/function
Addins
Console
Terminal
~/
R version 3.4.3 (2017-11-30) -- "Kite-Eating Tree"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
[Workspace loaded from ~/.RData]
>

Think of the console like talking directly to R. It's direct, but it has some drawbacks and some quirks we'll get into later. For now, try typing this into the console and hit enter:

```
2+2
```

```
## [1] 4
```

Congrats, you've run some code. It's not very complex, and you knew the answer before hand, but you get the idea. We can compute things. We can also store things. **In programming languages, these are called variables**. We can assign things to variables using `<-`. And then we can do things with them. **The `<-` is a called an assignment operator**.

```
number <- 2

number * number
```

```
## [1] 4
```

Now assign a different number to the variable number. Try run `number * number` again. Get what you expected?

We can have as many variables as we can name. **We can even reuse them (but be careful you know you're doing that or you'll introduce errors)**. Try this in your console.

```
firstnumber <- 1
secondnumber <-2

(firstnumber + secondnumber) * secondnumber
```

```
## [1] 6
```

We can store anything in a variable. A whole table. An array of numbers. A single word. A whole book. All the books of the 18th century. They're really powerful. We'll explore them at length.

2.1 Adding libraries, part 1

The real strength of any given programming language is the external libraries that power it. The base language can do a lot, but it's the external libraries that solve many specific problems – even making the base language easier to use.

For this class, we're going to need several external libraries.

The first library we're going to use is called Swirl. So in the console, type `install.packages('swirl')` and hit enter. That installs swirl.

Now, to use the library, type `library(swirl)` and hit enter. That loads swirl.

Then type `swirl()` and hit enter. Now you're running swirl. Follow the directions on the screen. When you are asked, you want to install course 1 R Programming: The basics of programming in R. Then, when asked, you want to do option 1, R Programming, in that course.

When you are finished with the course – it will take just a few minutes – it will first ask you if you want credit on Coursera. You do not. Then type 0 to exit (it will not be very clear that's what you do when you are done).

2.2 Adding libraries, part 2

We'll mostly use two libraries for analysis – `dplyr` and `ggplot2`. To get them, and several other useful libraries, we can install a single collection of libraries called the tidyverse. Type this into your console: `install.packages('tidyverse')`

NOTE: This is a pattern. You should always install libraries in the console.

Then, to help us with learning and replication, we're going to use R Notebooks. So we need to install that library. Type this into your console: `install.packages('rmarkdown')`

2.3 Notebooks

For the rest of the class, we're going to be working in notebooks. In notebooks, you will both run your code and explain each step, much as I am doing here.

To start a notebook, you click on the green plus in the top left corner and go down to R Notebook. Do that now.

Go to file/function
Addins
R Script ⇧⌘N
R Notebook
R Markdown...
Shiny Web App...
Text File
C++ File
R Sweave
R HTML
R Presentation
R Documentation
Insert
anything in a variable. A whole table. An array o
book. All the books of the 18th century. They're
l semester long.
braries
ngth of any given programming language is the ext
base language can do a lot, but it's the externa
problems -- even making the base language easier
, we're going to need several external libraries
analysis -- dplyr and ggplot2. To get them, and
libraries, we can install a single collection of libraries cal
this into your console:
35
36 ```{r}
37 install.packages('tidyverse')
38 ```
39 Then, to help us with learning and replication, we're going to
need to install that library. Type this into your console:
40 ```{r}
41 install.packages('rmarkdown')
42 ```
43 You may have to quit and restart R Studio. I honestly can't re
44

You will see that the notebook adds a lot of text for you. It tells you how to work in notebooks – and you should read it. The most important parts are these:

To add text, simply type. To add code you can click on the *Insert* button on the toolbar or by pressing *Cmd+Option+I* on Mac or *Ctl+Alt+I* on Windows.

Highlight all that text and delete it. You should have a blank document. This document is called a R Markdown file – it's a special form of text, one that you can style, and one you can include R in the middle of it. Markdown is a simple markup format that you can use to create documents. So first things first, let's give our notebook a big headline. Add this:

```
# My awesome notebook
```

Now, under that, without any markup, just type This is my awesome notebook.

Under that, you can make text bold by writing `It is **really** awesome`.

If you want it italics, just do this on the next line: `No, it's _really_ awesome. I swear.`

To see what it looks like without the markup, click the Preview or Knit button in the toolbar. That will turn your notebook into a webpage, with the formatting included.

Throughout this book, we're going to use this markdown to explain what we are doing and, more importantly, why we are doing it. Explaining your thinking is a vital part of understanding what you are doing.

That explaination, plus the code, is the real power of notebooks. To add a block of code, follow the instructions from above: click on the *Insert* button on the toolbar or by pressing *Cmd+Option+I* on Mac or *Ctl+Alt+I* on Windows.

In that window, use some of the code from above and add two numbers together. To see it run, click the green triangle on the right. That runs the chunk. You should see the answer to your addition problem.

And that, just that, is the foundation you need to start this book.

Chapter 3

Data, structures and types

Data are everywhere (and data is plural of datum, thus the use of are in that statement). It surrounds you. Every time you use your phone, you are creating data. Lots of it. Your online life. Any time you buy something. It's everywhere. Sports, like life, is no different. Sports is drowning in data, and more comes along all the time.

In sports, and in this class, we'll be dealing largely with two kinds of data: event level data and summary data. It's not hard to envision event level data in sports. A pitch in baseball. A hit. A play in football. A pass in soccer. They are the events that make up the game. Combine them together – summarize them – and you'll have some notion of how the game went. What we usually see is summary data – who wants to scroll through 50 pitches to find out a player went 2-3 with a double and an RBI? Who wants to scroll through hundreds of pitches to figure out the Rays beat the Yankees?

To start with, we need to understand the shape of data.

> EXERCISE: Try scoring a child's board game. For example, Chutes and Ladders. If you were placed in charge of analytics for the World Series of Chutes and Ladders, what is your event level data? What summary data do you keep? If you've got the game, try it.

3.1 Rows and columns

Data, oversimplifying it a bit, is information organized. Generally speaking, it's organized into rows and columns. Rows, generally, are individual elements. A team. A player. A game. Columns, generally, are components of the data, sometimes called variables. So if each row is a player, the first column might be their name. The second is their position. The third is their batting average. And so on.

G	Date		Opp	W/L	
1	2018-11-06		Mississippi Valley State	W	10
2	2018-11-11		Southeastern Louisiana	W	8
3	2018-11-14		Seton Hall	W	8
4	2018-11-19	N	Missouri State	W	8
5	2018-11-20	N	Texas Tech	L	5
6	2018-11-24		Western Illinois	W	7
7	2018-11-26	@	Clemson	W	6
8	2018-12-02		Rows	W	7
9	2018-12-05	@	Minnesota	L	7
10	2018-12-08		Creighton	W	9
11	2018-12-16	N	Oklahoma State	W	7
12	2018-12-22		Cal State Fullerton	W	8
13	2018-12-29		Southwest Minnesota State	W	7
14	2019-01-02	@	Maryland	L	7
15	2019-01-06	@	Iowa	L	8
16	2019-01-10		Penn State	W	7
17	2019-01-14	@	Indiana	W	6
18	2019-01-17		Michigan State	L	6
19	2019-01-21	@	Rutgers	L	6

One of the critical components of data analysis, especially for beginners, is having a mental picture of your data. What does each row mean? What does each column in each row signify? How many rows do you have? How many columns?

3.2 Types

There are scores of data types in the world, and R has them. In this class, we're primarily going to be dealing with data frames, and each element of our data frames will have a data type.

Typically, they'll be one of four types of data:

- Numeric: a number, like the number of touchdown passes in a season or a batting average.
- Character: Text, like a name, a team, a conference.
- Date: Fully formed dates – 2019-01-01 – have a special date type. Elements of a date, like a year (ex. 2019) are not technically dates, so they'll appear as numeric data types.
- Logical: Rare, but every now and then we'll have a data type that's Yes or No, True or False, etc.

Question: Is a zip code a number? Is a jersey number a number? Trick question, because the answer is no. Numbers are things we do math on. If the thing you want is not something you're going to do math on – can you add two phone numbers together? – then make it a character type. If you don't, most every software system on the planet will drop leading zeros. For example, every zip code in Boston starts with 0. If you record that as a number, your zip code will become a four digit number, which isn't a zip code anymore.

3.3 A simple way to get data

One good thing about sports is that there's lots of interest in it. And that means there's outlets that put sports data on the internet. Now I'm going to show you a trick to getting it easily.

The site sports-reference.com takes NCAA (and other league) stats and puts them online. For instance, here's their page on Nebraska basketball's game logs, which you should open now.

Now, in a new tab, log into Google Docs/Drive and open a new spreadsheet. In the first cell of the first row, copy and paste this formula in:

```
=IMPORTHTML("https://www.sports-reference.com/cbb/schools/nebraska/2019-gamelogs.html", "table",
```

If it worked right, you've got the data from that page in a spreadsheet.

3.4 Cleaning the data

The first thing we need to do is recognize that we don't have data, really. We have the results of a formula. You can tell by putting your cursor on that field, where you'll see the formula again. This is where you'd look:

	A	B	C	D	E
1					
2	G	Date		Opp	W/L
3	1	2018-11-06		Mississippi Valley	W
4	2	2018-11-11		Southeastern Lou	W
5	3	2018-11-14		Seton Hall	W
6	4	2018-11-19	N	Missouri State	W
7	5	2018-11-20	N	Texas Tech	L
8	6	2018-11-24		Western Illinois	W
9	7	2018-11-26	@	Clemson	W
10	8	2018-12-02		Illinois	W
11	9	2018-12-05	@	Minnesota	L
12	10	2018-12-08		Creighton	W
13	11	2018-12-16	N	Oklahoma State	W
14	12	2018-12-22		Cal State Fullerto	W
15	13	2018-12-29		Southwest Minne	W
16	14	2019-01-02	@	Maryland	L
17	15	2019-01-06	@	Iowa	L
18	16	2019-01-10		Penn State	W
19	17	2019-01-14	@	Indiana	W
20	18	2019-01-17		Michigan State	L

The solution is easy:

Edit > Select All or type command/control A Edit > Copy or type command/control c Edit > Paste Special > Values Only or type command/control shift v

You can verify that it worked by looking in that same row 1 column A, where you'll see the formula is gone.

Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help
Undo ⌘Z
Redo ⌘Y
Cut ⌘X
Copy ⌘C
Paste ⌘V
Paste special
Find and replace... ⌘+Shift+H
Delete values
Clear notes
Remove checkboxes
Paste values only
Paste format only
Paste all except b
Paste column wid
Paste formula onl
Paste data validat
Paste conditional
Paste transposed
Arial
10
=IMPOR
/cbb/schools/nebraska/
W/L
sissippi Valle
W
theastern Lo
W
14 2019-01-02 @
15 2019-01-06 @
16 2019-01-10
17 2019-01-14 @ Indiana W
18 2019-01-17 Michigan State L

Now you have data, but your headers are all wrong. You want your headers to be one line – not two, like they have. And the header names repeat – first for our team, then for theirs. So you have to change each header name to be UsORB or TeamORB and OpponentORB instead of just ORB.

After you've done that, note we have repeating headers. There's two ways to deal with that – you could just hightlight it and go up to Edit > Delete Rows XX-XX depending on what rows you highlighted. That's the easy way with our data.

But what if you had hundreds of repeating headers like that? Deleting them would take a long time.

You can use sorting to get rid of anything that's not data. So click on Data > Sort Range. You'll want to check the "Data has header row" field. Then hit Sort.

Pen.
Yards
Pen./G
Yards/G
Sort range from A1 to Z100
Data has header row
sort by
Year
A → Z
Z → A
+ Add another sort column
Sort
Cancel

Now all you need to do is search through the data for where your junk data – extra headers, blanks, etc. – got sorted and delete it. After you've done that, you can export it for use in R. Go to File > Download as > Comma Separated Values. Remember to put it in the same directory as your R Notebook file so you can import the data easily.

Chapter 4

Aggregates

R is a statistical programming language that is purpose built for data analysis.

Base R does a lot, but there are a mountain of external libraries that do things to make R better/easier/more fully featured. We already installed the tidyverse – or you should have if you followed the instructions for the last assignment – which isn't exactly a library, but a collection of libraries. Together, they make up the tidyverse. Individually, they are extraordinarily useful for what they do. We can load them all at once using the tidyverse name, or we can load them individually. Let's start with individually.

The two libraries we are going to need for this assignement are `readr` and `dplyr`. The library `readr` reads different types of data in. For this assignment, we're going to read in csv data or Comma Separated Values data. That's data that has a comma between each column of data.

Then we're going to use `dplyr` to analyze it.

To use a library, you need to import it. Good practice – one I'm going to insist on – is that you put all your library steps at the top of your notebooks.

That code looks like this:

```
library(readr)
```

To load them both, you need to run that code twice:

```
library(readr)
library(dplyr)
```

You can keep doing that for as many libraries as you need. I've seen notebooks with 10 or more library imports.

4.1 Basic data analysis: Group By and Count

The first thing we need to do is get some data to work with. We do that by reading it in. In our case, we're going to read data from a csv file – a comma-separated values file.

The CSV file we're going to read from is a Nebraska Game and Parks Commission dataset of confirmed mountain lion sightings in Nebraska. There are, on occasion, fierce debates about mountain lions and if they should be hunted in Nebraska. This dataset can tell us some interesting things about that debate.

So step 1 is to import the data. The code looks *something* like this, but hold off copying it just yet:

```
mountainlions <- read_csv("~/Documents/Data/mountainlions.csv")
```

Let's unpack that.

The first part – mountainlions – is the name of your variable. A variable is just a name of a thing. In this case, our variable is a data frame, which is R's way of storing data. We can call this whatever we want. I always want to name data frames after what is in it. In this case, we're going to import a dataset of mountain lion sightings from the Nebraska Game and Parks Commission. Variable names, by convention are one word all lower case. You can end a variable with a number, but you can't start one with a number.

The <- bit is the variable assignment operator. It's how we know we're assigning something to a word. Think of the arrow as saying "Take everything on the right of this arrow and stuff it into the thing on the left." So we're creating an empty vessel called mountainlions and stuffing all this data into it.

The `read_csv` bits are pretty obvious, except for one thing. What happens in the quote marks is the path to the data. In there, I have to tell R where it will find the data. The easiest thing to do, if you are confused about how to find your data, is to put your data in the same folder as as your notebook (you'll have to save that notebook first). If you do that, then you just need to put the name of the file in there (mountainlions.csv). In my case, I've got a folder called Documents in my home directory (that's the `~` part), and in there is a folder called Data that has the file called mountainlions.csv in it. Some people – insane people – leave the data in their downloads folder. The data path then would be `~/Downloads/nameofthedatafilehere.csv` on PC or Mac.

What you put in there will be different from mine. So your first task is to import the data.

```
mountainlions <- read_csv("data/mountainlions.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
```

```
##   `Cofirm Type` = col_character(),
##   COUNTY = col_character(),
##   Date = col_character()
## )
```

Now we can inspect the data we imported. What does it look like? To do that, we use `head(mountainlions)` to show the headers and the first six rows of data. If we wanted to see them all, we could just simply enter `mountainlions` and run it.

To get the number of records in our dataset, we run `nrow(mountainlions)`

```
head(mountainlions)
```

```
## # A tibble: 6 x 4
##      ID `Cofirm Type` COUNTY       Date
##   <dbl> <chr>         <chr>        <chr>
## 1     1 Track         Dawes        9/14/91
## 2     2 Mortality     Sioux        11/10/91
## 3     3 Mortality     Scotts Bluff 4/21/96
## 4     4 Mortality     Sioux        5/9/99
## 5     5 Mortality     Box Butte    9/29/99
## 6     6 Track         Scotts Bluff 11/12/99
```

```
nrow(mountainlions)
```

```
## [1] 393
```

So what if we wanted to know how many mountain lion sightings there were in each county? To do that by hand, we'd have to take each of the 393 records and sort them into a pile. We'd put them in groups and then count them.

`dplyr` has a group by function in it that does just this. A massive amount of data analysis involves grouping like things together at some point. So it's a good place to start.

So to do this, we'll take our dataset and we'll introduce a new operator: %>%. The best way to read that operator, in my opinion, is to interpret that as "and then do this." Here's the code:

```
mountainlions %>%
  group_by(COUNTY) %>%
  summarise(
    total = n()
  )
```

```
## # A tibble: 42 x 2
##    COUNTY    total
##    <chr>     <int>
##  1 Banner        6
##  2 Blaine        3
```

```
##  3 Box Butte     4
##  4 Brown        15
##  5 Buffalo       3
##  6 Cedar         1
##  7 Cherry       30
##  8 Custer        8
##  9 Dakota        3
## 10 Dawes       111
## # ... with 32 more rows
```

So let's walk through that. We start with our dataset – `mountainlions` – and then we tell it to group the data by a given field in the data. In this case, we wanted to group together all the counties, signified by the field name COUNTY, which you could get from looking at `head(mountainlions)`. After we group the data, we need to count them up. In dplyr, we use `summarize` which can do more than just count things. Inside the parentheses in summarize, we set up the summaries we want. In this case, we just want a count of the counties: `count = n()`, says create a new field, called `total` and set it equal to `n()`, which might look weird, but it's common in stats. The number of things in a dataset? Statisticians call in n. There are n number of incidents in this dataset. So `n()` is a function that counts the number of things there are.

And when we run that, we get a list of counties with a count next to them. But it's not in any order. So we'll add another And Then Do This %>% and use `arrange`. Arrange does what you think it does – it arranges data in order. By default, it's in ascending order – smallest to largest. But if we want to know the county with the most mountain lion sightings, we need to sort it in descending order. That looks like this:

```
mountainlions %>%
  group_by(COUNTY) %>%
  summarise(
    count = n()
  ) %>% arrange(desc(count))
```

```
## # A tibble: 42 x 2
##    COUNTY       count
##    <chr>        <int>
##  1 Dawes          111
##  2 Sioux           52
##  3 Sheridan        35
##  4 Cherry          30
##  5 Scotts Bluff    26
##  6 Keya Paha       20
##  7 Brown           15
##  8 Rock            11
##  9 Lincoln         10
```

```
## 10 Custer           8
## # ... with 32 more rows
```

We can, if we want, group by more than one thing. So how are these sightings being confirmed? To do that, we can group by County and "Cofirm Type", which is how the state misspelled Confirm. But note something in this example below:

```
mountainlions %>%
  group_by(COUNTY, `Cofirm Type`) %>%
  summarise(
    count = n()
  ) %>% arrange(desc(count))
```

```
## # A tibble: 93 x 3
## # Groups:   COUNTY [42]
##    COUNTY       `Cofirm Type`      count
##    <chr>        <chr>              <int>
##  1 Dawes        Trail Camera Photo    41
##  2 Sioux        Trail Camera Photo    40
##  3 Dawes        Track                 19
##  4 Keya Paha    Trail Camera Photo    18
##  5 Cherry       Trail Camera Photo    17
##  6 Dawes        Mortality             17
##  7 Sheridan     Trail Camera Photo    16
##  8 Dawes        Photo                 13
##  9 Dawes        DNA                   11
## 10 Scotts Bluff Trail Camera Photo    11
## # ... with 83 more rows
```

See it? When you have a field name that has two words, `readr` wraps it in backticks, which is next to the 1 key on your keyboard. You can figure out which fields have backticks around it by looking at the output of `readr`. Pay attention to that, because it's coming up again in the next section and will be a part of your homework.

4.2 Other aggregates: Mean and median

In the last example, we grouped some data together and counted it up, but there's so much more you can do. You can do multiple measures in a single step as well.

Let's look at some salary data from the University of Nebraska.

```
salaries <- read_csv("data/nusalaries1819.csv")
```

```
## Parsed with column specification:
## cols(
```

```
##   Employee = col_character(),
##   Position = col_character(),
##   Campus = col_character(),
##   Department = col_character(),
##   `Budgeted Annual Salary` = col_number(),
##   `Salary from State Aided Funds` = col_number(),
##   `Salary from Other Funds` = col_number()
## )
```

```
head(salaries)
```

```
## # A tibble: 6 x 7
##   Employee Position Campus Department `Budgeted Annua~ `Salary from St~
##   <chr>    <chr>    <chr>  <chr>                 <dbl>            <dbl>
## 1 Abbey, ~ Associa~ UNK    Kinesiolo~            61276            61276
## 2 Abbott,~ Staff S~ UNL    FM&P Faci~            37318               NA
## 3 Abboud,~ Adminis~ UNMC   Surgery-U~            76400            76400
## 4 Abdalla~ Asst Pr~ UNMC   Pathology~            74774            71884
## 5 Abdelka~ Post-Do~ UNMC   Surgery-T~            43516               NA
## 6 Abdel-M~ Researc~ UNL    Public Po~            58502               NA
## # ... with 1 more variable: `Salary from Other Funds` <dbl>
```

In summarize, we can calculate any number of measures. Here, we'll use R's built in mean and median functions to calculate … well, you get the idea.

```
salaries %>%
  summarise(
    count = n(),
    mean_salary = mean(`Budgeted Annual Salary`),
    median_salary = median(`Budgeted Annual Salary`)
  )
```

```
## # A tibble: 1 x 3
##   count mean_salary median_salary
##   <int>       <dbl>         <dbl>
## 1 13039      62065.         51343
```

So there's 13,039 employees in the database, spread across four campuses plus the system office. The mean or average salary is about $62,000, but the median salary is slightly more than $51,000.

Why?

Let's let sort help us.

```
salaries %>% arrange(desc(`Budgeted Annual Salary`))
```

```
## # A tibble: 13,039 x 7
##    Employee Position Campus Department `Budgeted Annua~ `Salary from St~
##    <chr>    <chr>    <chr>  <chr>                 <dbl>            <dbl>
```

```
##  1 Frost, ~ Head Co~ UNL    Athletics          5000000              NA
##  2 Miles, ~ Head Co~ UNL    Athletics          2375000              NA
##  3 Moos, W~ Athleti~ UNL    Athletics          1000000              NA
##  4 Gold, J~ Chancel~ UNMC   Office of~          853338          853338
##  5 Chinand~ Assista~ UNL    Athletics           800000              NA
##  6 Walters~ Assista~ UNL    Athletics           700000              NA
##  7 Cook, J~ Head Co~ UNL    Athletics           675000              NA
##  8 William~ Head Co~ UNL    Athletics           626750              NA
##  9 Bounds,~ Preside~ UNCA   Office of~          540000          540000
## 10 Austin ~ Assista~ UNL    Athletics           475000              NA
## # ... with 13,029 more rows, and 1 more variable: `Salary from Other
## #   Funds` <dbl>
```

Oh, right. In this dataset, the university pays a football coach $5 million. Extremes influence averages, not medians, and now you have your answer.

So when choosing a measure of the middle, you have to ask yourself – could I have extremes? Because a median won't be sensitive to extremes. It will be the point at which half the numbers are above and half are below. The average or mean will be a measure of the middle, but if you have a bunch of low paid people and then one football coach, the average will be wildly skewed. Here, because there's so few highly paid football coaches compared to people who make a normal salary, the number is only slightly skewed in the grand scheme, but skewed nonetheless.

4.3 Even more aggregates

There's a ton of things we can do in summarize – we'll work with more of them as the course progresses – but here's a few other questions you can ask.

Which department on campus has the highest wage bill? And what is the highest and lowest salary in the department? And how wide is the spread between salaries? We can find that with `sum` to add up the salaries to get the total wage bill, `min` to find the minumum salary, `max` to find the maximum salary and `sd` to find the standard deviation in the numbers.

```
salaries %>%
  group_by(Campus, Department) %>%
  summarise(
    total = sum(`Budgeted Annual Salary`),
    avgsalary = mean(`Budgeted Annual Salary`),
    minsalary = min(`Budgeted Annual Salary`),
    maxsalary = max(`Budgeted Annual Salary`),
    stdev = sd(`Budgeted Annual Salary`)) %>% arrange(desc(total))
```

```
## # A tibble: 804 x 7
## # Groups:   Campus [5]
```

```
##    Campus Department                  total avgsalary minsalary maxsalary  stdev
##    <chr>  <chr>                       <dbl>     <dbl>     <dbl>     <dbl>  <dbl>
##  1 UNL    Athletics                  3.56e7   118508.     12925   5000000 3.33e5
##  2 UNMC   Pathology/Microbiology     1.36e7    63158.      1994    186925 3.41e4
##  3 UNL    Agronomy & Horticulture    8.98e6    66496.      5000    208156 4.01e4
##  4 UNMC   Anesthesiology             7.90e6    78237.     10000    245174 3.59e4
##  5 UNL    School of Natural Resour~  6.86e6    65995.      2400    194254 3.28e4
##  6 UNL    College of Law             6.70e6    77953.      1000    326400 7.23e4
##  7 UNL    University Television      6.44e6    55542.     16500    221954 2.75e4
##  8 UNL    University Libraries       6.27e6    51390.      1200    215917 2.68e4
##  9 UNMC   Pharmacology/Exp Neurosc~  6.24e6    58911.      2118    248139 4.29e4
## 10 UNMC   CON-Omaha Division         6.11e6    78304.      3000    172522 4.48e4
## # ... with 794 more rows
```

So again, no surprise, the UNL athletic department has the single largest wage bill at nearly $36 million. The average salary in the department is $118,508 – more than double the univeristy as a whole, again thanks to Scott Frost's paycheck.

Chapter 5

Mutating data

One of the most common data analysis techniques is to look at change over time. The most common way of comparing change over time is through percent change. The math behind calculating percent change is very simple, and you should know it off the top of your head. The easy way to remember it is:

```
(new - old) / old
```

Or new minus old divided by old. Your new number minus the old number, the result of which is divided by the old number. To do that in R, we can use `dplyr` and `mutate` to calculate new metrics in a new field using existing fields of data.

So first we'll import the tidyverse so we can read in our data and begin to work with it.

```
library(tidyverse)
```

Now we'll import a common and simple dataset of total attendance at NCAA football games over the last few seasons.

```
attendance <- read_csv('data/attendance.csv')
```

```
## Parsed with column specification:
## cols(
##   Institution = col_character(),
##   Conference = col_character(),
##   `2013` = col_double(),
##   `2014` = col_double(),
##   `2015` = col_double(),
##   `2016` = col_double(),
##   `2017` = col_double(),
##   `2018` = col_double()
## )
```

```
head(attendance)
```

```
## # A tibble: 6 x 8
##   Institution     Conference      `2013` `2014` `2015` `2016` `2017` `2018`
##   <chr>           <chr>            <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Air Force       MWC             228562 168967 156158 177519 174924 166205
## 2 Akron           MAC             107101  55019 108588  62021 117416  92575
## 3 Alabama         SEC             710538 710736 707786 712747 712053 710931
## 4 Appalachian St. FBS Independent 149366     NA     NA     NA     NA     NA
## 5 Appalachian St. Sun Belt            NA 138995 128755 156916 154722 131716
## 6 Arizona         Pac-12          285713 354973 308355 338017 255791 318051
```

The code to calculate percent change is pretty simple. Remember, with `summarize`, we used `n()` to count things. With `mutate`, we use very similar syntax to calculate a new value using other values in our dataset. So in this case, we're trying to do (new-old)/old, but we're doing it with fields. If we look at what we got when we did `head`, you'll see there's '2018' as the new data, and we'll use '2017' as the old data. So we're looking at one year. Then, to help us, we'll use arrange again to sort it, so we get the fastest growing school over one year.

```
attendance %>% mutate(
  change = (`2018` - `2017`)/`2017`
)
```

```
## # A tibble: 150 x 9
##    Institution   Conference   `2013` `2014` `2015` `2016` `2017` `2018`   change
##    <chr>         <chr>         <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>    <dbl>
##  1 Air Force     MWC          228562 168967 156158 177519 174924 166205 -0.0498
##  2 Akron         MAC          107101  55019 108588  62021 117416  92575 -0.212
##  3 Alabama       SEC          710538 710736 707786 712747 712053 710931 -0.00158
##  4 Appalachian ~ FBS Indepen~ 149366     NA     NA     NA     NA     NA NA
##  5 Appalachian ~ Sun Belt         NA 138995 128755 156916 154722 131716 -0.149
##  6 Arizona       Pac-12       285713 354973 308355 338017 255791 318051  0.243
##  7 Arizona St.   Pac-12       501509 343073 368985 286417 359660 291091 -0.191
##  8 Arkansas      SEC          431174 399124 471279 487067 442569 367748 -0.169
##  9 Arkansas St.  Sun Belt     149477 149163 138043 136200 119538 119001 -0.00449
## 10 Army West Po~ FBS Indepen~ 169781 171310 185946 163267 185543 190156  0.0249
## # ... with 140 more rows
```

What do we see right away? Do those numbers look like we expect them to? No. They're a decimal expressed as a percentage. So let's fix that by multiplying by 100.

```
attendance %>% mutate(
  change = ((`2018` - `2017`)/`2017`)*100
)
```

```
## # A tibble: 150 x 9
##    Institution    Conference   `2013` `2014` `2015` `2016` `2017` `2018`  change
##    <chr>          <chr>         <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>
##  1 Air Force      MWC          228562 168967 156158 177519 174924 166205  -4.98
##  2 Akron          MAC          107101  55019 108588  62021 117416  92575 -21.2
##  3 Alabama        SEC          710538 710736 707786 712747 712053 710931  -0.158
##  4 Appalachian S~ FBS Indepen~ 149366     NA     NA     NA     NA     NA  NA
##  5 Appalachian S~ Sun Belt         NA 138995 128755 156916 154722 131716 -14.9
##  6 Arizona        Pac-12       285713 354973 308355 338017 255791 318051  24.3
##  7 Arizona St.    Pac-12       501509 343073 368985 286417 359660 291091 -19.1
##  8 Arkansas       SEC          431174 399124 471279 487067 442569 367748 -16.9
##  9 Arkansas St.   Sun Belt     149477 149163 138043 136200 119538 119001  -0.449
## 10 Army West Poi~ FBS Indepen~ 169781 171310 185946 163267 185543 190156   2.49
## # ... with 140 more rows
```

Now, does this ordering do anything for us? No. Let's fix that with arrange.

```
attendance %>% mutate(
  change = ((`2018` - `2017`)/`2017`)*100
) %>% arrange(desc(change))
```

```
## # A tibble: 150 x 9
##    Institution   Conference `2013` `2014` `2015` `2016` `2017` `2018` change
##    <chr>         <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Ga. Southern  Sun Belt       NA 105510 124681 104095  61031 100814   65.2
##  2 La.-Monroe    Sun Belt    85177  90540  58659  67057  49640  71048   43.1
##  3 Louisiana     Sun Belt   129878 154652 129577 121346  78754 111303   41.3
##  4 Hawaii        MWC        185931 192159 164031 170299 145463 205455   41.2
##  5 Buffalo       MAC        136418 122418 110743 104957  80102 110280   37.7
##  6 California    Pac-12     345303 286051 292797 279769 219290 300061   36.8
##  7 UCF           AAC        252505 226869 180388 214814 257924 352148   36.5
##  8 UTSA          C-USA      175282 165458 138048 138226 114104 148257   29.9
##  9 Eastern Mich. MAC         20255  75127  29381 106064  73649  95632   29.8
## 10 Louisville    ACC            NA 317829 294413 324391 276957 351755   27.0
## # ... with 140 more rows
```

So who had the most growth last year from the year before? Something going on at Georgia Southern.

5.1 A more complex example

There's metric in basketball that's easy to understand – shooting percentage. It's the number of shots made divided by the number of shots attempted. Simple, right? Except it's a little too simple. Because what about three point shooters? They tend to be more vailable because the three point shot is worth more. What about players who get to the line? In shooting percentage, free throws are nowhere to be found.

Basketball nerds, because of these weaknesses, have created a new metric called True Shooting Percentage. True shooting percentage takes into account all aspects of a players shooting to determine who the real shooters are.

Using `dplyr` and `mutate`, we can calculate true shooting percentage. So let's look at a new dataset, one of every college basketball player's season stats in 2018-19 season. It's a dataset of 5,386 players, and we've got 59 variables – one of them is True Shooting Percentage, but we're going to ignore that.

```
players <- read_csv("data/players19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character(),
##   Conference = col_character(),
##   Player = col_character(),
##   Class = col_character(),
##   Pos = col_character(),
##   Height = col_character(),
##   Hometown = col_character(),
##   `High School` = col_character(),
##   Summary = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

The basic true shooting percentage formula is `(Points / (2*(FieldGoalAttempts + (.44 * FreeThrowAttempts)))) * 100`. Let's talk that through. Points divided by a lot. It's really field goal attempts plus 44 percent of the free throw attempts. Why? Because that's about what a free throw is worth, compared to other ways to score. After adding those things together, you double it. And after you divide points by that number, you multiply the whole lot by 100.

In our data, we need to be able to find the fields so we can complete the formula. To do that, one way is to use the Environment tab in R Studio. In the Environment tab is a listing of all the data you've imported, and if you click the triangle next to it, it'll list all the field names, giving you a bit of information about each one.

Environment
History
Connections
Import Dataset
Global Environment
players
5386 obs. of 59 vari
X1 : num 1 2 3 4 5 6 7 8 9 10 .
Team : chr "Youngstown State Pe
Conference : chr "Horizon" "Hor
Player : chr "Darius Quisenberr
: num 3 32 22 2 33 13 5 31 21
Class : chr "FR" "SO" "JR" "FR"
Files
Plots
Packages
Help
Viewe
New Folder
Delete
Rename
Home > Box > BookProjects > Sport
Name
27-finishingtouches2.Rmd
28-assignments.Rmd
ly field
a free throw
it. And

So what does True Shooting Percentage look like in code?

Let's think about this differently. Who had the best true shooting season last year?

```
players %>%
  mutate(trueshooting = (PTS/(2*(FGA + (.44*FTA))))*100) %>%
  arrange(desc(trueshooting))
```

```
## # A tibble: 5,386 x 60
##       X1 Team  Conference Player   `#` Class Pos   Height Weight Hometown
##    <dbl> <chr> <chr>      <chr>  <dbl> <chr> <chr> <chr>   <dbl> <chr>
##  1   579 Texa~ Big 12     Drayt~     4 JR    G     6-0       156 Austin,~
##  2   843 Ston~ AEC        Nick ~    42 FR    F     6-7       240 Port Je~
##  3  1059 Sout~ Southland  Patri~    22 SO    F     6-3       210 Folsom,~
##  4  4269 Dayt~ A-10       Camro~    52 SO    G     5-7       160 Country~
##  5  4681 Cali~ Pac-12     David~    21 JR    G     6-4       185 Newbury~
##  6   326 Virg~ ACC        Grant~     1 FR    G     <NA>       NA Charlot~
##  7   410 Vand~ SEC        Mac H~    42 FR    G     6-6       182 Chattan~
##  8  1390 Sain~ A-10       Jack ~    31 JR    G     6-6       205 Mattoon~
##  9  2230 NJIT~ A-Sun      Patri~     3 SO    G     5-9       160 West Or~
## 10   266 Wash~ Pac-12     Reaga~    34 FR    F     6-6       225 Santa A~
## # ... with 5,376 more rows, and 50 more variables: `High School` <chr>,
## #   Summary <chr>, Rk.x <dbl>, G <dbl>, GS <dbl>, MP <dbl>, FG <dbl>,
## #   FGA <dbl>, `FG%` <dbl>, `2P` <dbl>, `2PA` <dbl>, `2P%` <dbl>, `3P` <dbl>,
## #   `3PA` <dbl>, `3P%` <dbl>, FT <dbl>, FTA <dbl>, `FT%` <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, Rk.y <dbl>, PER <dbl>, `TS%` <dbl>, `eFG%` <dbl>, `3PAr` <dbl>,
## #   FTr <dbl>, PProd <dbl>, `ORB%` <dbl>, `DRB%` <dbl>, `TRB%` <dbl>,
## #   `AST%` <dbl>, `STL%` <dbl>, `BLK%` <dbl>, `TOV%` <dbl>, `USG%` <dbl>,
## #   OWS <dbl>, DWS <dbl>, WS <dbl>, `WS/40` <dbl>, OBPM <dbl>, DBPM <dbl>,
## #   BPM <dbl>, trueshooting <dbl>
```

You'll be forgiven if you did not hear about Texas Longhorns shooting sensation Drayton Whiteside. He played in six games, took one shot and actually hit it. It happened to be a three pointer, which is one more three pointer than I've hit in college basketball. So props to him. Does that mean he had the best true shooting season in college basketball last year? Not hardly.

We'll talk about how to narrow the pile and filter out data in the next chapter.

Chapter 6

Filters and selections

More often than not, we have more data than we want. Sometimes we need to be rid of that data. In `dplyr`, there's two ways to go about this: filtering and selecting.

Filtering creates a subset of the data based on criteria. All records where the count is greater than 10. All records that match "Nebraska". Something like that.

Selecting simply returns only the fields named. So if you only want to see School and Attendance, you select those fields. When you look at your data again, you'll have two columns. If you try to use one of your columns that you had before you used select, you'll get an error.

Let's work with our football attendance data to show some examples.

```
library(tidyverse)
```

```
attendance <- read_csv('data/attendance.csv')
```

```
## Parsed with column specification:
## cols(
##   Institution = col_character(),
##   Conference = col_character(),
##   `2013` = col_double(),
##   `2014` = col_double(),
##   `2015` = col_double(),
##   `2016` = col_double(),
##   `2017` = col_double(),
##   `2018` = col_double()
## )
```

So, first things first, let's say we don't care about all this Air Force, Akron,

Alabama crap and just want to see Dear Old Nebraska U. We do that with `filter` and then we pass it a condition.

Before we do that, a note about conditions. Most of the conditional operators you'll understand – greater than and less than are > and <. The tough one to remember is equal to. In conditional statements, equal to is == not =. If you haven't noticed, = is a variable assignment operator, not a conditional statement. So equal is == and NOT equal is !=.

So if you want to see Institutions equal to Nebraska, you do this:

```
attendance %>% filter(Institution == "Nebraska")
```

```
## # A tibble: 1 x 8
##   Institution Conference `2013` `2014` `2015` `2016` `2017` `2018`
##   <chr>       <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 Nebraska    Big Ten    727466 638744 629983 631402 628583 623240
```

Or if we want to see schools that had more than half a million people buy tickets to a football game in a season, we do the following. NOTE THE BACKTICKS.

```
attendance %>% filter(`2018` >= 500000)
```

```
## # A tibble: 17 x 8
##    Institution    Conference `2013` `2014` `2015` `2016` `2017` `2018`
##    <chr>          <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Alabama        SEC        710538 710736 707786 712747 712053 710931
##  2 Auburn         SEC        685252 612157 612157 695498 605120 591236
##  3 Clemson        ACC        574333 572262 588266 566787 565412 562799
##  4 Florida        SEC        524638 515001 630457 439229 520290 576299
##  5 Georgia        SEC        556476 649222 649222 556476 556476 649222
##  6 LSU            SEC        639927 712063 654084 708618 591034 705733
##  7 Michigan       Big Ten    781144 734364 771174 883741 669534 775156
##  8 Michigan St.   Big Ten    506294 522765 522628 522666 507398 508088
##  9 Nebraska       Big Ten    727466 638744 629983 631402 628583 623240
## 10 Ohio St.       Big Ten    734528 744075 750705 750944 752464 713630
## 11 Oklahoma       Big 12     508334 510972 512139 521142 519119 607146
## 12 Penn St.       Big Ten    676112 711358 698590 701800 746946 738396
## 13 South Carolina SEC        576805 569664 472934 538441 550099 515396
## 14 Tennessee      SEC        669087 698276 704088 706776 670454 650887
## 15 Texas          Big 12     593857 564618 540210 587283 556667 586277
## 16 Texas A&M      SEC        697003 630735 725354 713418 691612 698908
## 17 Wisconsin      Big Ten    552378 556642 546099 476144 551766 540072
```

But what if we want to see all of the Power Five conferences? We *could* use conditional logic in our filter. The conditional logic operators are `|` for OR and `&` for AND. NOTE: AND means all conditions have to be met. OR means any of the conditions work. So be careful about boolean logic.

```
attendance %>% filter(Conference == "Big 10" | Conference == "SEC" | Conference == "Pac-12" | Con
```

```
## # A tibble: 51 x 8
##    Institution    Conference `2013` `2014` `2015` `2016` `2017` `2018`
##    <chr>          <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Alabama        SEC        710538 710736 707786 712747 712053 710931
##  2 Arizona        Pac-12     285713 354973 308355 338017 255791 318051
##  3 Arizona St.    Pac-12     501509 343073 368985 286417 359660 291091
##  4 Arkansas       SEC        431174 399124 471279 487067 442569 367748
##  5 Auburn         SEC        685252 612157 612157 695498 605120 591236
##  6 Baylor         Big 12     321639 280257 276960 275029 262978 248017
##  7 Boston College ACC        198035 239893 211433 192942 215546 263363
##  8 California     Pac-12     345303 286051 292797 279769 219290 300061
##  9 Clemson        ACC        574333 572262 588266 566787 565412 562799
## 10 Colorado       Pac-12     230778 226670 236331 279652 282335 274852
## # ... with 41 more rows
```

But that's a lot of repetitive code. And a lot of typing. And typing is the devil. So what if we could create a list and pass it into the filter? It's pretty simple.

We can create a new variable – remember variables can represent just about anything – and create a list. To do that we use the `c` operator, which stands for concatenate. That just means take all the stuff in the parenthesis after the c and bunch it into a list.

Note here: text is in quotes. If they were numbers, we wouldn't need the quotes.

```
powerfive <- c("SEC", "Big Ten", "Pac-12", "Big 12", "ACC")
```

Now with a list, we can use the %in% operator. It does what you think it does – it gives you data that matches things IN the list you give it.

```
attendance %>% filter(Conference %in% powerfive)
```

```
## # A tibble: 65 x 8
##    Institution    Conference `2013` `2014` `2015` `2016` `2017` `2018`
##    <chr>          <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Alabama        SEC        710538 710736 707786 712747 712053 710931
##  2 Arizona        Pac-12     285713 354973 308355 338017 255791 318051
##  3 Arizona St.    Pac-12     501509 343073 368985 286417 359660 291091
##  4 Arkansas       SEC        431174 399124 471279 487067 442569 367748
##  5 Auburn         SEC        685252 612157 612157 695498 605120 591236
##  6 Baylor         Big 12     321639 280257 276960 275029 262978 248017
##  7 Boston College ACC        198035 239893 211433 192942 215546 263363
##  8 California     Pac-12     345303 286051 292797 279769 219290 300061
##  9 Clemson        ACC        574333 572262 588266 566787 565412 562799
## 10 Colorado       Pac-12     230778 226670 236331 279652 282335 274852
## # ... with 55 more rows
```

6.1 Selecting data to make it easier to read

So now we have our Power Five list. What if we just wanted to see attendance from the most recent season and ignore all the rest? Select to the rescue.

```
attendance %>% filter(Conference %in% powerfive) %>% select(Institution, Conference, `2
```

```
## # A tibble: 65 x 3
##    Institution    Conference `2018`
##    <chr>          <chr>       <dbl>
##  1 Alabama        SEC        710931
##  2 Arizona        Pac-12     318051
##  3 Arizona St.    Pac-12     291091
##  4 Arkansas       SEC        367748
##  5 Auburn         SEC        591236
##  6 Baylor         Big 12     248017
##  7 Boston College ACC        263363
##  8 California     Pac-12     300061
##  9 Clemson        ACC        562799
## 10 Colorado       Pac-12     274852
## # ... with 55 more rows
```

If you have truly massive data, Select has tools to help you select fields that start_with the same things or ends with a certain word. The documentation will guide you if you need those someday. For 90 plus percent of what we do, just naming the fields will be sufficient.

6.2 Using conditional filters to set limits

Let's return to the blistering season of Drayton Whiteside using our dataset of every college basketball player's season stats in 2018-19 season. How can we set limits in something like a question of who had the best season? Let's get our Drayton Whiteside data from the previous chapter back up.

```
players <- read_csv("data/players19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character(),
##   Conference = col_character(),
##   Player = col_character(),
##   Class = col_character(),
##   Pos = col_character(),
##   Height = col_character(),
##   Hometown = col_character(),
```

```
##   `High School` = col_character(),
##   Summary = col_character()
## )

## See spec(...) for full column specifications.
```

```
players %>%
  mutate(trueshooting = (PTS/(2*(FGA + (.44*FTA))))*100) %>%
  arrange(desc(trueshooting))
```

```
## # A tibble: 5,386 x 60
##       X1 Team  Conference Player   `#` Class Pos   Height Weight Hometown
##    <dbl> <chr> <chr>      <chr>  <dbl> <chr> <chr> <chr>   <dbl> <chr>
##  1   579 Texa~ Big 12     Drayt~     4 JR    G     6-0       156 Austin,~
##  2   843 Ston~ AEC        Nick ~    42 FR    F     6-7       240 Port Je~
##  3  1059 Sout~ Southland  Patri~    22 SO    F     6-3       210 Folsom,~
##  4  4269 Dayt~ A-10       Camro~    52 SO    G     5-7       160 Country~
##  5  4681 Cali~ Pac-12     David~    21 JR    G     6-4       185 Newbury~
##  6   326 Virg~ ACC        Grant~     1 FR    G     <NA>       NA Charlot~
##  7   410 Vand~ SEC        Mac H~    42 FR    G     6-6       182 Chattan~
##  8  1390 Sain~ A-10       Jack ~    31 JR    G     6-6       205 Mattoon~
##  9  2230 NJIT~ A-Sun      Patri~     3 SO    G     5-9       160 West Or~
## 10   266 Wash~ Pac-12     Reaga~    34 FR    F     6-6       225 Santa A~
## # ... with 5,376 more rows, and 50 more variables: `High School` <chr>,
## #   Summary <chr>, Rk.x <dbl>, G <dbl>, GS <dbl>, MP <dbl>, FG <dbl>,
## #   FGA <dbl>, `FG%` <dbl>, `2P` <dbl>, `2PA` <dbl>, `2P%` <dbl>, `3P` <dbl>,
## #   `3PA` <dbl>, `3P%` <dbl>, FT <dbl>, FTA <dbl>, `FT%` <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, Rk.y <dbl>, PER <dbl>, `TS%` <dbl>, `eFG%` <dbl>, `3PAr` <dbl>,
## #   FTr <dbl>, PProd <dbl>, `ORB%` <dbl>, `DRB%` <dbl>, `TRB%` <dbl>,
## #   `AST%` <dbl>, `STL%` <dbl>, `BLK%` <dbl>, `TOV%` <dbl>, `USG%` <dbl>,
## #   OWS <dbl>, DWS <dbl>, WS <dbl>, `WS/40` <dbl>, OBPM <dbl>, DBPM <dbl>,
## #   BPM <dbl>, trueshooting <dbl>
```

In most contests like the batting title in Major League Baseball, there's a minimum number of X to qualify. In baseball, it's at bats. In basketball, it attempts. So let's set a floor and see how it changes. What if we said you had to have played 100 minutes in a season? The top players in college basketball play more than 1000 minutes in a season. So 100 is not that much. Let's try it and see.

```
players %>%
  mutate(trueshooting = (PTS/(2*(FGA + (.44*FTA))))*100) %>%
  arrange(desc(trueshooting)) %>%
  filter(MP > 100)
```

```
## # A tibble: 3,659 x 60
##       X1 Team  Conference Player   `#` Class Pos   Height Weight Hometown
##    <dbl> <chr> <chr>      <chr>  <dbl> <chr> <chr> <chr>   <dbl> <chr>
```

```
##  1  4634 Cent~ Southland  Jorda~    33 JR    G     6-1       185 Harriso~
##  2  3623 Hart~ AEC        Max T~    20 SR    G     6-5       200 Rye, NY
##  3  2675 Mich~ Big Ten    Thoma~    15 FR    F     6-8       225 Clarkst~
##  4  5175 Litt~ Sun Belt   Kris ~    32 SO    F     6-8       194 Dewitt,~
##  5  5205 Ariz~ Pac-12     De'Qu~    32 SR    F     6-10      225 St. Tho~
##  6  4099 ETSU~ Southern   Lucas~    25 JR    C     7-0       220 De Lier~
##  7  3006 Loui~ Sun Belt   Brand~     0 SR    G     6-4       180 Hawthor~
##  8   570 Texa~ Big 12     Jaxso~    10 FR    F     6-11      220 Lovelan~
##  9  1704 Pepp~ WCC        Victo~    34 FR    C     6-9       200 Owerri,~
## 10  4056 East~ MAC        Jalen~    30 SO    F     6-9       215 Pasco, ~
## # ... with 3,649 more rows, and 50 more variables: `High School` <chr>,
## #   Summary <chr>, Rk.x <dbl>, G <dbl>, GS <dbl>, MP <dbl>, FG <dbl>,
## #   FGA <dbl>, `FG%` <dbl>, `2P` <dbl>, `2PA` <dbl>, `2P%` <dbl>, `3P` <dbl>,
## #   `3PA` <dbl>, `3P%` <dbl>, FT <dbl>, FTA <dbl>, `FT%` <dbl>, ORB <dbl>,
## #   DRB <dbl>, TRB <dbl>, AST <dbl>, STL <dbl>, BLK <dbl>, TOV <dbl>, PF <dbl>,
## #   PTS <dbl>, Rk.y <dbl>, PER <dbl>, `TS%` <dbl>, `eFG%` <dbl>, `3PAr` <dbl>,
## #   FTr <dbl>, PProd <dbl>, `ORB%` <dbl>, `DRB%` <dbl>, `TRB%` <dbl>,
## #   `AST%` <dbl>, `STL%` <dbl>, `BLK%` <dbl>, `TOV%` <dbl>, `USG%` <dbl>,
## #   OWS <dbl>, DWS <dbl>, WS <dbl>, `WS/40` <dbl>, OBPM <dbl>, DBPM <dbl>,
## #   BPM <dbl>, trueshooting <dbl>
```

Now you get Central Arkansas Bears Junior Jordan Grant, who played in 25 games and was on the floor for 152 minutes. So he played regularly. But in that time, he only attempted 16 shots, and made 68 percent of them. In other words, when he shot, he probably scored. He just rarely shot.

So is 100 minutes our level? Here's the truth – there's not really an answer here. We're picking a cutoff. If you can cite a reason for it and defend it, then it probably works.

6.3 Top list

One last little dplyr trick that's nice to have in the toolbox is a shortcut for selecting only the top values for your dataset. Want to make a Top 10 List? Or Top 25? Or Top Whatever You Want? It's easy.

So what are the top 10 Power Five schools by season attendance. All we're doing here is chaining commands together with what we've already got. We're *filtering* by our list of Power Five conferences, we're *selecting* the three fields we need, now we're going to *arrange* it by total attendance and then we'll introduce the new function: `top_n`. The `top_n` function just takes a number. So we want a top 10 list? We do it like this:

```
attendance %>% filter(Conference %in% powerfive) %>% select(Institution, Conference, `2
```

```
## Selecting by 2018
```

```
## # A tibble: 10 x 3
```

```
##    Institution Conference `2018`
##    <chr>       <chr>       <dbl>
##  1 Michigan    Big Ten    775156
##  2 Penn St.    Big Ten    738396
##  3 Ohio St.    Big Ten    713630
##  4 Alabama     SEC        710931
##  5 LSU         SEC        705733
##  6 Texas A&M   SEC        698908
##  7 Tennessee   SEC        650887
##  8 Georgia     SEC        649222
##  9 Nebraska    Big Ten    623240
## 10 Oklahoma    Big 12     607146
```

That's all there is to it. Just remember – for it to work correctly, you need to sort your data BEFORE you run top_n. Otherwise, you're just getting the first 10 values in the list. The function doesn't know what field you want the top values of. You have to do it.

Chapter 7

Transforming data

Sometimes long data needs to be wide, and sometimes wide data needs to be long. I'll explain.

You are soon going to discover that long before you can visualize data, **you need to have it in a form that the visualization library can deal with**. One of the ways that isn't immediately obvious is **how your data is cast**. Most of the data you will encounter will be **wide – each row will represent a single entity with multiple measures for that entity**. So think of states. Your row of your dataset could have the state name, population, average life expectancy and other demographic data.

But what if your visualization library needs one row for each measure? So state, data type and the data. Nebraska, Population, 1,929,000. That's one row. Then the next row is Nebraska, Average Life Expectancy, 76. That's the next row. That's where recasting your data comes in.

We can use a library called `tidyr` to `pivot_longer` or `pivot_wider` the data, depending on what we need. We'll use a dataset of college football attendance to demonstrate. First we need some libraries.

```
library(tidyverse)
```

Now we'll load the data.

```
attendance <- read_csv('data/attendance.csv')
```

```
## Parsed with column specification:
## cols(
##   Institution = col_character(),
##   Conference = col_character(),
##   `2013` = col_double(),
##   `2014` = col_double(),
```

```
##   `2015` = col_double(),
##   `2016` = col_double(),
##   `2017` = col_double(),
##   `2018` = col_double()
## )
```

```
attendance
```

```
## # A tibble: 150 x 8
##    Institution     Conference      `2013` `2014` `2015` `2016` `2017` `2018`
##    <chr>           <chr>            <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 Air Force       MWC             228562 168967 156158 177519 174924 166205
##  2 Akron           MAC             107101  55019 108588  62021 117416  92575
##  3 Alabama         SEC             710538 710736 707786 712747 712053 710931
##  4 Appalachian St. FBS Independent 149366     NA     NA     NA     NA     NA
##  5 Appalachian St. Sun Belt            NA 138995 128755 156916 154722 131716
##  6 Arizona         Pac-12          285713 354973 308355 338017 255791 318051
##  7 Arizona St.     Pac-12          501509 343073 368985 286417 359660 291091
##  8 Arkansas        SEC             431174 399124 471279 487067 442569 367748
##  9 Arkansas St.    Sun Belt        149477 149163 138043 136200 119538 119001
## 10 Army West Point FBS Independent 169781 171310 185946 163267 185543 190156
## # ... with 140 more rows
```

So as you can see, each row represents a school, and then each column represents a year. This is great for calculating the percent change – we can subtract a column from a column and divide by that column. But later, when we want to chart each school's attendance over the years, we have to have each row be one team for one year. Nebraska in 2013, then Nebraska in 2014, and Nebraska in 2015 and so on.

To do that, we use `pivot_longer` because we're making wide data long. Since all of the columns we want to make rows start with 20, we can use that in our `cols` directive. Then we give that column a name – Year – and the values for each year need a name too. Those are the attendance figure. We can see right away how this works.

```
attendance %>% pivot_longer(cols = starts_with("20"), names_to = "Year", values_to = "A
```

```
## # A tibble: 900 x 4
##    Institution Conference Year  Attendance
##    <chr>       <chr>      <chr>      <dbl>
##  1 Air Force   MWC        2013      228562
##  2 Air Force   MWC        2014      168967
##  3 Air Force   MWC        2015      156158
##  4 Air Force   MWC        2016      177519
##  5 Air Force   MWC        2017      174924
##  6 Air Force   MWC        2018      166205
##  7 Akron       MAC        2013      107101
```

```
##  8 Akron       MAC        2014       55019
##  9 Akron       MAC        2015      108588
## 10 Akron       MAC        2016       62021
## # ... with 890 more rows
```

We've gone from 150 rows to 900, but that's expected when we have 6 years for each team.

7.1 Making long data wide

We can reverse this process using `pivot_wider`, which makes long data wide.

Why do any of this?

In some cases, you're going to be given long data and you need to calculate some metric using two of the years – a percent change for instance. So you'll need to make the data wide to do that. You might then have to re-lengthen the data now with the percent change. Some project require you to do all kinds of flexing like this. It just depends on the data.

So let's take what we made above and turn it back into wide data.

```
longdata <- attendance %>% pivot_longer(cols = starts_with("20"), names_to = "Year", values_to =

longdata
```

```
## # A tibble: 900 x 4
##    Institution Conference Year  Attendance
##    <chr>       <chr>      <chr>      <dbl>
##  1 Air Force   MWC        2013      228562
##  2 Air Force   MWC        2014      168967
##  3 Air Force   MWC        2015      156158
##  4 Air Force   MWC        2016      177519
##  5 Air Force   MWC        2017      174924
##  6 Air Force   MWC        2018      166205
##  7 Akron       MAC        2013      107101
##  8 Akron       MAC        2014       55019
##  9 Akron       MAC        2015      108588
## 10 Akron       MAC        2016       62021
## # ... with 890 more rows
```

To `pivot_wider`, we just need to say where our column names are coming from – the Year – and where the data under it should come from – Attendance.

```
longdata %>% pivot_wider(names_from = Year, values_from = Attendance)
```

```
## # A tibble: 150 x 8
##    Institution     Conference      `2013` `2014` `2015` `2016` `2017` `2018`
##    <chr>           <chr>            <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
```

```
##  1 Air Force       MWC             228562 168967 156158 177519 174924 166205
##  2 Akron           MAC             107101  55019 108588  62021 117416  92575
##  3 Alabama         SEC             710538 710736 707786 712747 712053 710931
##  4 Appalachian St. FBS Independent 149366     NA     NA     NA     NA     NA
##  5 Appalachian St. Sun Belt            NA 138995 128755 156916 154722 131716
##  6 Arizona         Pac-12          285713 354973 308355 338017 255791 318051
##  7 Arizona St.     Pac-12          501509 343073 368985 286417 359660 291091
##  8 Arkansas        SEC             431174 399124 471279 487067 442569 367748
##  9 Arkansas St.    Sun Belt        149477 149163 138043 136200 119538 119001
## 10 Army West Point FBS Independent 169781 171310 185946 163267 185543 190156
## # ... with 140 more rows
```

And just like that, we're back.

7.2 Why this matters

This matters because certain visualization types need wide or long data. A significant hurdle you will face for the rest of the semester is getting the data in the right format for what you want to do.

So let me walk you through an example using this data.

Let's look at Nebraska's attendance over the time period. In order to do that, I need long data because that's what the charting library, `ggplot2`, needs. You're going to learn a lot more about ggplot later.

```
nebraska <- longdata %>% filter(Institution == "Nebraska")
```

Now that we have long data for just Nebraska, we can chart it.

```
ggplot(nebraska, aes(x=Year, y=Attendance, group=1)) +
  geom_line() +
  scale_y_continuous(labels = scales::comma) +
  labs(x="Year", y="Attendance", title="We'll all stick together?", subtitle="It's not
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank(),
    axis.text = element_text(size = 7),
    axis.ticks = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank(),
    legend.position="bottom"
  )
```

We'll all stick together?
It's not as bad as you think −− they widened the seats, cutting the number.
725,000
700,000
675,000
650,000
625,000
2013
2014
2015
2016
2017
2018
Year
Source: NCAA | By Matt Waite

Chapter 8

Simulations

Two seasons ago, James Palmer Jr. shot 139 three point attempts and made 43 of them for a .309 shooting percentage last year. A few weeks into last season, he was 7 for 39 – a paltry .179. Is something wrong or is this just bad luck?

Luck is something that comes up a lot in sports. Is a team unlucky? Or a player? One way we can get to this, we can get to that is by simulating things based on their typical percentages. Simulations work by choosing random values within a range based on a distribution. The most common distribution is the normal or binomial distribution. The normal distribution is where the most cases appear around the mean, 66 percent of cases are within one standard deviation from the mean, and the further away from the mean you get, the more rare things become.

Let's simulate 39 three point attempts 1000 times with his season long shooting percentage and see if this could just be random chance or something else.

We do this using a base R function called `rbinom` or binomial distribution. So what that means is there's a normally distrubuted chance that James Palmer Jr. is going to shoot above and below his career three point shooting percentage.

If we randomly assign values in that distribution 1000 times, how many times will it come up 7, like this example?

```
set.seed(1234)

simulations <- rbinom(n = 1000, size = 39, prob = .309)

table(simulations)
```

```
## simulations
##   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22
##   1   4   5  12  35  44  76 117 134 135 135  99  71  53  37  21  15   2   3   1
```

How do we read this? The first row and the second row form a pair. The top row is the number of shots made. The number immediately under it is the number of simulations where that occurred.

So what we see is given his season long shooting percentage, it's not out of the realm of randomness that with just 39 attempts for Palmer, he's only hit only 7. In 1000 simulations, it comes up 35 times. Is he below where he should be? Yes. Will he likely improve and soon? Unless something is very wrong, yes. And indeed, by the end of the season, he finished with a .313 shooting percentage from 3 point range. So we can say he was just unlucky.

8.1 Cold streaks

During the Western Illinois game, the team, shooting .329 on the season from behind the arc, went 0-15 in the second half. How strange is that?

```
set.seed(1234)

simulations <- rbinom(n = 1000, size = 15, prob = .329)

hist(simulations)
```

```
table(simulations)
```

```
## simulations
##   0   1   2   3   4   5   6   7   8   9  10  11
##   5  16  59 132 200 218 172  92  65  34   4   3
```

Short answer: Really weird. If you simulate 15 threes 1000 times, sometimes you'll see them miss all of them, but only a few times – five times, in this case. Most of the time, the team won't go 0-15 even once. So going ice cold is not totally out of the realm of random chance, but it's highly unlikely.

Chapter 9

Correlations and regression

Throughout sports, you will find no shortage of opinions. From people yelling at their TV screens to an entire industry of people paid to have opinions, there are no shortage of reasons why this team sucks and that player is great. They may have their reasons, but a better question is, does that reason really matter?

Can we put some numbers behind that? Can we prove it or not?

This is what we're going to start to answer. And we'll do it with correlations and regressions.

First, we need libraries and data.

```
library(tidyverse)
```

```
correlations <- read_csv("data/correlations.csv")
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   OffPoints = col_double(),
##   OffPointsG = col_double(),
##   DefPoints = col_double(),
##   DefPointsG = col_double(),
##   Pen. = col_double(),
##   Yards = col_double(),
##   `Pen./G` = col_double(),
##   OffConversions = col_double(),
##   OffConversionPct = col_double(),
##   DefConversions = col_double(),
##   DefConversionPct = col_double()
## )
```

To do this, we need all FBS college football teams and their season stats from last year. How much, over the course of a season, does a thing matter? That's the question you're going to answer.

In our case, we want to know how much does a team's accumulated penalties influence the number of points they score in a season? How much difference can we explain in points with penalties?

We're going to use two different methods here and they're closely related. Correlations – specifically the Pearson Correlation Coefficient – is a measure of how related two numbers are in a linear fashion. In other words – if our X value goes up one, what happens to Y? If it also goes up 1, that's a perfect correlation. X goes up 1, Y goes up 1. Every time. Correlation coefficients are a number between 0 and 1, with zero being no correlation and 1 being perfect correlation **if our data is linear**. We'll soon go over scatterplots to visually determine if our data is linear, but for now, we have a hypothesis: More penalties are bad. Penalties hurt. So if a team gets lots of them, they should have worse outcomes than teams that get few of them. That is an argument for a linear relationship between them.

But is there one?

We're going create a new dataframe called newcorrelations that takes our data that we imported and adds a column called `differential` because we don't have separate offense and defense penalties, and then we'll use correlations to see how related those two things are.

```
newcorrelations <- correlations %>%
  mutate(differential = OffPoints - DefPoints)
```

In R, there is a `cor` function, and it works much the same as `mean` or `median`. So we want to see if `differential` is correlated with `Yards`, which is the yards of penalties a team gets in a game. We do that by referenceing `differential` and `Yards` and specifying we want a `pearson` correlation. The number we get back is the correlation coefficient.

```
newcorrelations %>% summarise(correlation = cor(differential, Yards, method="pearson"))
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1       0.201
```

So on a scale of 0 to 1, penalty yards and whether or not the team scores more points than it give up are at .2. You could say they're 20 percent related. Another way to say it? They're 80 percent not related.

What about the number of penalties instead of the yards?

```r
newcorrelations %>%
  summarise(correlation = cor(differential, `Pen.`, method="pearson"))
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1       0.153
```

Even less related. What about looking at the average? Penalty yards per game?

```r
newcorrelations %>% summarise(correlation = cor(differential, `Pen./G`, method="pearson"))
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1     -0.0331
```

Not only is it less related, but the relationship is inverted.

So wait, what does that mean?

It means that the number of penalty yards and penalties is actually positively related to differential. Put another way, teams that have more penalties and penalty yards tend to have better outcomes. The average is barely – 3 percent – negatively correlated, meaning that teams with higher averages score fewer points.

What? That makes no sense. How can that be?

Enter regression. Regression is how we try to fit our data into a line that explains the relationship the best. Regressions will help us predict things as well – if we have a team that has so many penalties, what kind of point differential could we expect, given every FBS team? So regressions are about prediction, correlations are about description. Correlations describe a relationship. Regressions help us predict what that relationship means. Specifically, it tells us how much of the change in a dependent variable can be explained by the independent variable.

Another thing regressions do is give us some other tools to evaluate if the relationship is real or not.

Here's an example of using linear modeling to look at yards. Think of the ~ character as saying "is predicted by". The output looks like a lot, but what we need is a small part of it.

```r
fit <- lm(differential ~ Yards, data = newcorrelations)
summary(fit)
```

```
##
## Call:
## lm(formula = differential ~ Yards, data = newcorrelations)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -351.02  -93.49    2.67  107.88  444.42
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -108.73848   59.70868  -1.821   0.0709 .
## Yards          0.19484    0.08399   2.320   0.0219 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 140 on 128 degrees of freedom
## Multiple R-squared:  0.04035,    Adjusted R-squared:  0.03285
## F-statistic: 5.382 on 1 and 128 DF,  p-value: 0.02193
```

There's three things we need here:

1. First we want to look at the p-value. It's at the bottom right corner of the output. In the case of Yards, the p-value is .02193. The threshold we're looking for here is .05. If it's less than .05, then the relationship is considered to be *statistically significant*. Significance here does not mean it's a big deal. It means it's not random. That's it. Just that. Not random. So in our case, the relationship between penalty yards and a team's aggregate point differential are not random. It's a real relationship.
2. Second, we look at the Adjusted R-squared value. It's right above the p-value. Adjusted R-squared is a measure of how much of the difference between teams aggregate point values can be explained by penalty yards. Our correlation coefficient said they're 20 percent related to each other, but penalty yard's ability to explain the difference between teams? About 3.3 percent. That's … not much. It's really nothing.
3. The third thing we can look at, and we only bother if the first two are meaningful, is the coefficients. In the middle, you can see the (Intercept) is -108.73848 and the Yards coefficient is .19484. Remember high school algebra? Remember learning the equation of a line? Remember swearing that learning `y=mx+b` is stupid because you'll never need it again? Surprise. It's useful again. In this case, we could try to predict a team's aggregate score in a season – will they score more than they give up – by using `y=mx+b`. In this case, y is the aggregate score, m is .19484 and b is -108.73848. So we would multiply a teams total penalty yards by .19484 and then subtract 108.73848 from it. The result would tell you what the total aggregate score in the season would be. Chance that your even close with this? About 3 percent.

You can see the problem in a graph. On the X axis is penalty yards, on the y is aggregate score. If these elements had a strong relationship, we'd see a clear pattern moving from right to left, sloping down. On the left would be the teams with lots of penalty yards and a negative point differential. On right would

be teams with low penalty yards and high point differentials. Do you see that below?

Your turn: Try it with the other penalty measures. Total penalties and penalty yards per game. Does anything change? Do either of these meet the .05 threshold for randomness? Are either of these any more predictive?

9.1 A more predictive example

So we've firmly established that penalties aren't predictive. But what is? One measure I've found to be highly predictive of a team's success is how well do they do on third down. It's simple really: Succeed on third down, you get to stay on offense. Fail on third down, you are punting (most likely) or settling for a field goal. Either way, you're scoring less than you would by scoring touchdowns. How related are points per game and third down conversion percentage?

```
newcorrelations %>%
  summarise(correlation = cor(OffPointsG, OffConversionPct, method="pearson"))
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1       0.666
```

Answer: 67 percent. More than three times more related than penalty yards. But how meaningful is that relationship and how predictive is it?

```r
third <- lm(OffPointsG ~ OffConversionPct, data = newcorrelations)
summary(third)
```

```
##
## Call:
## lm(formula = OffPointsG ~ OffConversionPct, data = newcorrelations)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.3861  -3.5411  -0.5885   2.9011  13.5188
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -4.74024    3.41041   -1.39    0.167
## OffConversionPct  0.85625    0.08479   10.10   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.111 on 128 degrees of freedom
## Multiple R-squared:  0.4434, Adjusted R-squared:  0.4391
## F-statistic:   102 on 1 and 128 DF,  p-value: < 2.2e-16
```

First we check p-value. See that e-16? That means scientific notation. That means our number is 2.2 times 10 to the -16 power. So -.000000000000000022. That's sixteen zeros between the decimal and 22. Is that less than .05? Uh, yeah. So this is really, really, really not random. But anyone who has watched a game of football knows this is true. It makes intuitive sense.

Second, Adjusted R-squared: .4391. So we can predict 44 percent of the difference in the total offensive points per game a team scores by simply looking at their third down conversion percentage.

Third, the coefficients: In this case, our `y=mx+b` formula looks like `y = .85625x-4.74024`. So if we were applying this, let's look at Nebraska's 31-28 loss to Iowa on Black Friday in 2018. Nebraska was 6-15 on third down in that game, or 40 percent (Iowa was 7 of 13 or 54 percent). Given those numbers, our formula predicts Nebraska should have scored how many points?

```r
(0.85625 * 40) - 4.74024
```

```
## [1] 29.50976
```

That's really close to the 28 they did score. And Iowa?

```r
(0.85625 * 54) - 4.74024
```

```
## [1] 41.49726
```

By our model, Iowa should have scored 10 more points than they did. But they

didn't. Why, besides Iowa is terrible and deserves punishment from the football gods for being Iowa? Remember our model can only explain 44 percent of the points. There's more to football than one metric.

Chapter 10

Multiple regression

Last chapter, we looked at correlations and linear regression to predict how one element of a game would predict the score. But we know that a single variable, in all but the rarest instances, are not going to be that predictive. We need more than one. Enter multiple regression. Multiple regression lets us add – wait for it – multiple predictors to our equation to help us get a better

That presents it's own problems. So let's get our libraries and our data, this time of every college basketball game since the 2014-15 season loaded up.

```
library(tidyverse)
```

```
logs <- read_csv("data/logs1519.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

So one way to show how successful a basketball team was for a game is to show the differential between the team's score and the opponent's score. Score a lot

more than the opponent = good, score a lot less than the opponent = bad. And, relatively speaking, the more the better. So let's create that differential.

```
logs <- logs %>% mutate(Differential = TeamScore - OpponentScore)
```

The linear model code we used before is pretty straight forward. Its `field` is predicted by `field`. Here's a simple linear model that looks at predicting a team's point differential by looking at their offensive shooting percentage.

```
shooting <- lm(TeamFGPCT ~ Differential, data=logs)
summary(shooting)
```

```
##
## Call:
## lm(formula = TeamFGPCT ~ Differential, data = logs)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.260485 -0.040230 -0.001096  0.039038  0.267457
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.399e-01  2.487e-04  1768.4   <2e-16 ***
## Differential 2.776e-03  1.519e-05   182.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05949 on 57514 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.3675, Adjusted R-squared:  0.3674
## F-statistic: 3.341e+04 on 1 and 57514 DF,  p-value: < 2.2e-16
```

Remember: There's a lot here, but only some of it we care about. What is the Adjusted R-squared value? What's the p-value and is it less than .05? In this case, we can predict 37 percent of the difference in differential with how well a team shoots the ball.

To add more predictors to this mix, we merely add them. But it's not that simple, as you'll see in a moment. So first, let's look at adding how well the other team shot to our prediction model:

```
model1 <- lm(Differential ~ TeamFGPCT + OpponentFGPCT, data=logs)
summary(model1)
```

```
##
## Call:
## lm(formula = Differential ~ TeamFGPCT + OpponentFGPCT, data = logs)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -49.591  -6.185  -0.198   5.938  68.344
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)      1.1195     0.3483    3.214  0.00131 **
## TeamFGPCT      118.5211     0.5279  224.518  < 2e-16 ***
## OpponentFGPCT -119.9369     0.5252 -228.372  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.407 on 57513 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.6683, Adjusted R-squared:  0.6683
## F-statistic: 5.793e+04 on 2 and 57513 DF,  p-value: < 2.2e-16
```

First things first: What is the adjusted R-squared?

Second: what is the p-value and is it less than .05?

Third: Compare the residual standard error. We went from .05949 to 9.4. The meaning of this is both really opaque and also simple – we added a lot of error to our model by adding more measures – 158 times more. Residual standard error is the total distance between what our model would predict and what we actually have in the data. So lots of residual error means the distance between reality and our model is wider. So the width of our predictive range in this example grew pretty dramatically, but so did the amount of the difference we could predict. It's a trade off.

One of the more difficult things to understand about multiple regression is the issue of multicollinearity. What that means is that there is significant correlation overlap between two variables – the two are related to each other as well as to the target output – and all you are doing by adding both of them is adding error with no real value to the R-squared. In pure statistics, we don't want any multicollinearity at all. Violating that assumption limits the applicability of what you are doing. So if we have some multicollinearity, it limits our scope of application to college basketball. We can't say this will work for every basketball league and level everywhere. What we need to do is see how correlated each value is to each other and throw out ones that are highly co-correlated.

So to find those, we have to create a correlation matrix that shows us how each value is correlated to our outcome variable, but also with each other. We can do that in the `Hmisc` library. We install that in the console with `install.packages("Hmisc")`

```r
library(Hmisc)
```

We can pass in every numeric value to the Hmisc library and get a correlation matrix out of it, but since we have a large number of values – and many of them

character values – we should strip that down and reorder them. So that's what I'm doing here. I'm saying give me differential first, and then columns 9-24, and then 26-41. Why the skip? There's a blank column in the middle of the data – a remnant of the scraper I used.

```
simplelogs <- logs %>% select(Differential, 9:24, 26:41)
```

Before we proceed, what we're looking to do is follow the Differential column down, looking for correlation values near 1 or -1. Correlations go from -1, meaning perfect negative correlation, to 0, meaning no correlation, to 1, meaning perfect positive correlation. So we're looking for numbers near 1 or -1 for their predictive value. BUT: We then need to see if that value is also highly correlated with something else. If it is, we have a decision to make.

We get our correlation matrix like this:

```
cormatrix <- rcorr(as.matrix(simplelogs))

cormatrix$r
```

```
##                      Differential       TeamFG      TeamFGA    TeamFGPCT
## Differential          1.000000000  0.584766682  0.107389235  0.606178206
## TeamFG                0.584766682  1.000000000  0.563220974  0.751715176
## TeamFGA               0.107389235  0.563220974  1.000000000 -0.109620267
## TeamFGPCT             0.606178206  0.751715176 -0.109620267  1.000000000
## Team3P                0.318300418  0.408787900  0.213352219  0.322872202
## Team3PA               0.056680627  0.179527313  0.426011924 -0.119421368
## Team3PPCT             0.367934059  0.380235821 -0.101463821  0.545986963
## TeamFT                0.238182740 -0.022308582 -0.137853824  0.084649669
## TeamFTA               0.206075949 -0.027927391 -0.129851346  0.070632302
## TeamFTPCT             0.138833800  0.016247282 -0.044394472  0.056887587
## TeamOffRebounds       0.136095147  0.161626257  0.545231683 -0.234244567
## TeamTotalRebounds     0.470722398  0.328460524  0.470719037  0.018581908
## TeamAssists           0.540398009  0.664057724  0.284659104  0.566152928
## TeamSteals            0.277670288  0.210221346  0.208743124  0.080191710
## TeamBlocks            0.257608076  0.140856644  0.074555286  0.107327505
## TeamTurnovers        -0.180578328 -0.143210529 -0.223971265  0.001901048
## TeamPersonalFouls    -0.194427271 -0.014722266  0.107325560 -0.094653222
## OpponentFG           -0.538515115  0.144061400  0.256737262 -0.020183466
## OpponentFGA           0.001768386  0.302143806  0.301593528  0.126415534
## OpponentFGPCT        -0.614427717 -0.058571888  0.068034775 -0.114791403
## Opponent3P           -0.283754971  0.131517138  0.135290090  0.053105214
## Opponent3PA           0.013910296  0.191131927  0.138445785  0.118723805
## Opponent3PPCT        -0.382427841  0.008026622  0.057261756 -0.031370545
## OpponentFT           -0.269300868  0.019511923  0.157025930 -0.091558712
## OpponentFTA          -0.226064714  0.012937366  0.159529646 -0.101685664
## OpponentFTPCT        -0.175223632  0.007923359  0.023732217 -0.006190565
## OpponentOffRebounds  -0.089347536 -0.036316958  0.002848058 -0.042399744
```

```
## OpponentTotalRebounds -0.420010794 -0.225202127  0.316139528 -0.512983306
## OpponentAssists       -0.491676030  0.004558539  0.149320067 -0.106252682
## OpponentSteals        -0.187754380 -0.102436608 -0.131734964 -0.021724636
## OpponentBlocks        -0.262252627 -0.160469663  0.218483865 -0.356255034
## OpponentTurnovers      0.274326954  0.155293275  0.198127970  0.024254833
## OpponentPersonalFouls  0.169025733 -0.023116620 -0.107189301  0.060150658
##                             Team3P     Team3PA     Team3PPCT       TeamFT
## Differential           0.318300418  0.05668063  0.3679340589  0.238182740
## TeamFG                 0.408787900  0.17952731  0.3802358207 -0.022308582
## TeamFGA                0.213352219  0.42601192 -0.1014638212 -0.137853824
## TeamFGPCT              0.322872202 -0.11942137  0.5459869634  0.084649669
## Team3P                 1.000000000  0.70114773  0.7073663404 -0.106344056
## Team3PA                0.701147726  1.00000000  0.0407645751 -0.160515313
## Team3PPCT              0.707366340  0.04076458  1.0000000000  0.005129556
## TeamFT                -0.106344056 -0.16051531  0.0051295561  1.000000000
## TeamFTA               -0.137499074 -0.18150913 -0.0180696209  0.927525817
## TeamFTPCT              0.048777304  0.01119250  0.0553684315  0.387017653
## TeamOffRebounds       -0.062026229  0.12484929 -0.1968568361  0.087168289
## TeamTotalRebounds      0.038344971  0.12095682 -0.0628970009  0.190691619
## TeamAssists            0.519530086  0.28786139  0.4326950943 -0.016343370
## TeamSteals             0.016545254  0.04598400 -0.0246657289  0.088535320
## TeamBlocks             0.004747719 -0.02895321  0.0294277389  0.092392379
## TeamTurnovers         -0.088374940 -0.10883919 -0.0209433827  0.051609207
## TeamPersonalFouls     -0.024028303  0.02499520 -0.0498165852  0.217846416
## OpponentFG             0.123800594  0.15638030  0.0296913406  0.057853338
## OpponentFGA            0.148931744  0.13062824  0.0812237901  0.193116094
## OpponentFGPCT          0.029908235  0.08057726 -0.0264843759 -0.075399282
## Opponent3P             0.079455775  0.07482590  0.0402012413  0.024228311
## Opponent3PA            0.085704376  0.05927299  0.0601150176  0.079894905
## Opponent3PPCT          0.029666235  0.04634676  0.0005076038 -0.035478488
## OpponentFT             0.009796521  0.06316300 -0.0390873639  0.161311559
## OpponentFTA           -0.002503282  0.05474884 -0.0480732723  0.183801456
## OpponentFTPCT          0.022780414  0.02587876  0.0086512859 -0.015688533
## OpponentOffRebounds   -0.007870292 -0.01895081  0.0086776821  0.064938518
## OpponentTotalRebounds -0.062384273  0.20289676 -0.2638845414 -0.064969878
## OpponentAssists        0.029413582  0.08254506 -0.0320289494 -0.057730062
## OpponentSteals        -0.053878305 -0.05298037 -0.0251316716 -0.001883349
## OpponentBlocks        -0.111782062 -0.05804217 -0.0965607977 -0.065055523
## OpponentTurnovers      0.009284106  0.06383515 -0.0488449748  0.136922084
## OpponentPersonalFouls -0.127197007 -0.15536393 -0.0268876881  0.793539202
##                            TeamFTA    TeamFTPCT TeamOffRebounds
## Differential           0.206075949  0.138833800    0.1360951470
## TeamFG                -0.027927391  0.016247282    0.1616262575
## TeamFGA               -0.129851346 -0.044394472    0.5452316831
## TeamFGPCT              0.070632302  0.056887587   -0.2342445674
## Team3P                -0.137499074  0.048777304   -0.0620262290
```

```
## Team3PA               -0.181509133  0.011192503    0.1248492948
## Team3PPCT             -0.018069621  0.055368431   -0.1968568361
## TeamFT                 0.927525817  0.387017653    0.0871682888
## TeamFTA                1.000000000  0.053233778    0.1415933172
## TeamFTPCT              0.053233778  1.000000000   -0.0948040467
## TeamOffRebounds        0.141593317 -0.094804047    1.0000000000
## TeamTotalRebounds      0.231278690 -0.037356471    0.6373027887
## TeamAssists           -0.028289202  0.025948025    0.0509277222
## TeamSteals             0.111199125 -0.025969502    0.1195581042
## TeamBlocks             0.104112579 -0.001425412    0.1060163877
## TeamTurnovers          0.072070652 -0.034614485    0.0371728710
## TeamPersonalFouls      0.250787085 -0.025827923    0.0542337992
## OpponentFG             0.043602296  0.036986356   -0.0464694335
## OpponentFGA            0.193466766  0.040334507    0.0242353640
## OpponentFGPCT         -0.091897172  0.012864509   -0.0688833747
## Opponent3P             0.009600704  0.031763685   -0.0063710321
## Opponent3PA            0.071193179  0.032554796    0.0003753868
## Opponent3PPCT         -0.047136861  0.013996880   -0.0056578317
## OpponentFT             0.180010001 -0.009352580    0.0434399899
## OpponentFTA            0.213209437 -0.025707797    0.0584669041
## OpponentFTPCT         -0.032862991  0.028078614   -0.0319032781
## OpponentOffRebounds    0.077003661 -0.016936223   -0.0143325753
## OpponentTotalRebounds  0.004736343 -0.177541483   -0.0603891339
## OpponentAssists       -0.063875391 -0.007401206   -0.0386521955
## OpponentSteals         0.006758108 -0.022033431    0.0326977763
## OpponentBlocks        -0.053973588 -0.041175463    0.1571812909
## OpponentTurnovers      0.169704736 -0.035463921    0.1154717115
## OpponentPersonalFouls  0.866395092  0.018757079    0.1240631120
##                       TeamTotalRebounds   TeamAssists   TeamSteals   TeamBlocks
## Differential                 0.470722398  0.5403980088  0.277670288  0.257608076
## TeamFG                       0.328460524  0.6640577242  0.210221346  0.140856644
## TeamFGA                      0.470719037  0.2846591045  0.208743124  0.074555286
## TeamFGPCT                    0.018581908  0.5661529279  0.080191710  0.107327505
## Team3P                       0.038344971  0.5195300862  0.016545254  0.004747719
## Team3PA                      0.120956819  0.2878613903  0.045984003 -0.028953212
## Team3PPCT                   -0.062897001  0.4326950943 -0.024665729  0.029427739
## TeamFT                       0.190691619 -0.0163433697  0.088535320  0.092392379
## TeamFTA                      0.231278690 -0.0282892019  0.111199125  0.104112579
## TeamFTPCT                   -0.037356471  0.0259480253 -0.025969502 -0.001425412
## TeamOffRebounds              0.637302789  0.0509277222  0.119558104  0.106016388
## TeamTotalRebounds            1.000000000  0.2321524530  0.027446991  0.265518873
## TeamAssists                  0.232152453  1.0000000000  0.164837110  0.144764562
## TeamSteals                   0.027446991  0.1648371104  1.000000000  0.065539758
## TeamBlocks                   0.265518873  0.1447645615  0.065539758  1.000000000
## TeamTurnovers                0.109155292 -0.0789200586  0.078278779  0.032775757
## TeamPersonalFouls           -0.007423332 -0.1050900267  0.005151965 -0.054105029
```

```
## OpponentFG                -0.229331788 -0.0022308763 -0.138728115 -0.143969401
## OpponentFGA                0.360268614  0.1863368268 -0.120696505  0.257245080
## OpponentFGPCT             -0.530432484 -0.1397140493 -0.068951590 -0.353110391
## Opponent3P                -0.053371243  0.0354785684 -0.062074442 -0.103465578
## Opponent3PA                0.232049186  0.1116023406 -0.039184667 -0.042234814
## Opponent3PPCT             -0.273572339 -0.0502063543 -0.047114732 -0.099440199
## OpponentFT                -0.095266106 -0.0835716395 -0.034152581 -0.070920662
## OpponentFTA               -0.022971823 -0.0841605708 -0.022178476 -0.056095076
## OpponentFTPCT             -0.194279344 -0.0278263543 -0.041125993 -0.052504157
## OpponentOffRebounds       -0.052416263 -0.0333847454  0.016707012  0.178200671
## OpponentTotalRebounds     -0.059965631 -0.2225952122  0.035155522  0.037788375
## OpponentAssists           -0.218597433  0.0006884142 -0.053327136 -0.151146052
## OpponentSteals             0.066119486 -0.0288668673  0.055697260  0.028453380
## OpponentBlocks             0.013924890 -0.1657235463 -0.002230784 -0.038978593
## OpponentTurnovers         -0.034355689  0.1314533533  0.730885169  0.031375703
## OpponentPersonalFouls      0.189144014 -0.0267820830  0.071442012  0.080582762
##                       TeamTurnovers TeamPersonalFouls   OpponentFG  OpponentFGA
## Differential           -0.180578328      -0.194427271 -0.538515115  0.001768386
## TeamFG                 -0.143210529      -0.014722266  0.144061400  0.302143806
## TeamFGA                -0.223971265       0.107325560  0.256737262  0.301593528
## TeamFGPCT               0.001901048      -0.094653222 -0.020183466  0.126415534
## Team3P                 -0.088374940      -0.024028303  0.123800594  0.148931744
## Team3PA                -0.108839191       0.024995197  0.156380301  0.130628244
## Team3PPCT              -0.020943383      -0.049816585  0.029691341  0.081223790
## TeamFT                  0.051609207       0.217846416  0.057853338  0.193116094
## TeamFTA                 0.072070652       0.250787085  0.043602296  0.193466766
## TeamFTPCT              -0.034614485      -0.025827923  0.036986356  0.040334507
## TeamOffRebounds         0.037172871       0.054233799 -0.046469434  0.024235364
## TeamTotalRebounds       0.109155292      -0.007423332 -0.229331788  0.360268614
## TeamAssists            -0.078920059      -0.105090027 -0.002230876  0.186336827
## TeamSteals              0.078278779       0.005151965 -0.138728115 -0.120696505
## TeamBlocks              0.032775757      -0.054105029 -0.143969401  0.257245080
## TeamTurnovers           1.000000000       0.220285924  0.081879049  0.155947902
## TeamPersonalFouls       0.220285924       1.000000000 -0.015422966 -0.122639976
## OpponentFG              0.081879049      -0.015422966  1.000000000  0.515517123
## OpponentFGA             0.155947902      -0.122639976  0.515517123  1.000000000
## OpponentFGPCT          -0.023017156       0.078411084  0.754791141 -0.161220379
## Opponent3P             -0.018088322      -0.126817358  0.399027442  0.193563166
## Opponent3PA             0.041669476      -0.167647391  0.144074778  0.418730422
## Opponent3PPCT          -0.063187150      -0.015909552  0.395540055 -0.118020866
## OpponentFT              0.123594852       0.793147614 -0.013421944 -0.156152803
## OpponentFTA             0.154110278       0.865844664 -0.027151720 -0.151706668
## OpponentFTPCT          -0.034267574       0.026877590  0.037049836 -0.043324702
## OpponentOffRebounds     0.074131214       0.122282037  0.120715447  0.519792207
## OpponentTotalRebounds  -0.106168146       0.195017438  0.275438081  0.424276325
## OpponentAssists         0.072644677      -0.022619097  0.638304131  0.231851475
```

```
## OpponentSteals          0.709987911       0.064446997  0.140823916  0.165329579
## OpponentBlocks          0.006463872       0.087211248  0.129076992  0.045565883
## OpponentTurnovers       0.188537020       0.101693555 -0.183558009 -0.215633733
## OpponentPersonalFouls   0.131539040       0.322258517  0.015334210  0.136789046
##                       OpponentFGPCT   Opponent3P   Opponent3PA Opponent3PPCT
## Differential           -0.614427717 -0.283754971  0.0139102958 -0.3824278411
## TeamFG                 -0.058571888  0.131517138  0.1911319274  0.0080266219
## TeamFGA                 0.068034775  0.135290090  0.1384457845  0.0572617563
## TeamFGPCT              -0.114791403  0.053105214  0.1187238045 -0.0313705446
## Team3P                  0.029908235  0.079455775  0.0857043764  0.0296662353
## Team3PA                 0.080577258  0.074825900  0.0592729911  0.0463467602
## Team3PPCT              -0.026484376  0.040201241  0.0601150176  0.0005076038
## TeamFT                 -0.075399282  0.024228311  0.0798949051 -0.0354784876
## TeamFTA                -0.091897172  0.009600704  0.0711931792 -0.0471368607
## TeamFTPCT               0.012864509  0.031763685  0.0325547961  0.0139968801
## TeamOffRebounds        -0.068883375 -0.006371032  0.0003753868 -0.0056578317
## TeamTotalRebounds      -0.530432484 -0.053371243  0.2320491861 -0.2735723395
## TeamAssists            -0.139714049  0.035478568  0.1116023406 -0.0502063543
## TeamSteals             -0.068951590 -0.062074442 -0.0391846669 -0.0471147320
## TeamBlocks             -0.353110391 -0.103465578 -0.0422348142 -0.0994401990
## TeamTurnovers          -0.023017156 -0.018088322  0.0416694763 -0.0631871498
## TeamPersonalFouls       0.078411084 -0.126817358 -0.1676473908 -0.0159095518
## OpponentFG              0.754791141  0.399027442  0.1440747785  0.3955400546
## OpponentFGA            -0.161220379  0.193563166  0.4187304220 -0.1180208656
## OpponentFGPCT           1.000000000  0.312295571 -0.1493674362  0.5522792378
## Opponent3P              0.312295571  1.000000000  0.6914518201  0.7094041257
## Opponent3PA            -0.149367436  0.691451820  1.0000000000  0.0303822862
## Opponent3PPCT           0.552279238  0.709404126  0.0303822862  1.0000000000
## OpponentFT              0.106226566 -0.106344743 -0.1743400433  0.0169282910
## OpponentFTA             0.086625216 -0.140194309 -0.1972872368 -0.0080249496
## OpponentFTPCT           0.076650746  0.053774302  0.0101886734  0.0623587723
## OpponentOffRebounds    -0.251623986 -0.085432899  0.0978389488 -0.2013096986
## OpponentTotalRebounds  -0.005789348  0.005903551  0.0810576009 -0.0680836101
## OpponentAssists         0.553535793  0.513869716  0.2641728450  0.4428640799
## OpponentSteals          0.036468797 -0.011661373  0.0214481397 -0.0383569868
## OpponentBlocks          0.111935521 -0.004746412 -0.0495426307  0.0354134646
## OpponentTurnovers      -0.048082678 -0.095218199 -0.0944428800 -0.0421344973
## OpponentPersonalFouls  -0.081776664 -0.011247805  0.0396475169 -0.0466461289
##                         OpponentFT  OpponentFTA OpponentFTPCT
## Differential          -0.269300868 -0.226064714  -0.175223632
## TeamFG                 0.019511923  0.012937366   0.007923359
## TeamFGA                0.157025930  0.159529646   0.023732217
## TeamFGPCT             -0.091558712 -0.101685664  -0.006190565
## Team3P                 0.009796521 -0.002503282   0.022780414
## Team3PA                0.063163000  0.054748838   0.025878762
## Team3PPCT             -0.039087364 -0.048073272   0.008651286
```

```
## TeamFT                 0.161311559  0.183801456  -0.015688533
## TeamFTA                0.180010001  0.213209437  -0.032862991
## TeamFTPCT             -0.009352580 -0.025707797   0.028078614
## TeamOffRebounds        0.043439990  0.058466904  -0.031903278
## TeamTotalRebounds     -0.095266106 -0.022971823  -0.194279344
## TeamAssists           -0.083571639 -0.084160571  -0.027826354
## TeamSteals            -0.034152581 -0.022178476  -0.041125993
## TeamBlocks            -0.070920662 -0.056095076  -0.052504157
## TeamTurnovers          0.123594852  0.154110278  -0.034267574
## TeamPersonalFouls      0.793147614  0.865844664   0.026877590
## OpponentFG            -0.013421944 -0.027151720   0.037049836
## OpponentFGA           -0.156152803 -0.151706668  -0.043324702
## OpponentFGPCT          0.106226566  0.086625216   0.076650746
## Opponent3P            -0.106344743 -0.140194309   0.053774302
## Opponent3PA           -0.174340043 -0.197287237   0.010188673
## Opponent3PPCT          0.016928291 -0.008024950   0.062358772
## OpponentFT             1.000000000  0.928286066   0.393203255
## OpponentFTA            0.928286066  1.000000000   0.063446167
## OpponentFTPCT          0.393203255  0.063446167   1.000000000
## OpponentOffRebounds    0.086671729  0.136423744  -0.082982260
## OpponentTotalRebounds  0.197591588  0.232447345  -0.021281750
## OpponentAssists       -0.012378006 -0.031205800   0.041793598
## OpponentSteals         0.077614062  0.097206119  -0.022196700
## OpponentBlocks         0.101422181  0.110063752   0.008946765
## OpponentTurnovers      0.015778567  0.038679394  -0.052040732
## OpponentPersonalFouls  0.215609923  0.251289640  -0.029978048
##                       OpponentOffRebounds OpponentTotalRebounds OpponentAssists
## Differential                  -0.089347536          -0.420010794   -0.4916760300
## TeamFG                        -0.036316958          -0.225202127    0.0045585394
## TeamFGA                        0.002848058           0.316139528    0.1493200670
## TeamFGPCT                     -0.042399744          -0.512983306   -0.1062526818
## Team3P                        -0.007870292          -0.062384273    0.0294135821
## Team3PA                       -0.018950808           0.202896760    0.0825450568
## Team3PPCT                      0.008677682          -0.263884541   -0.0320289494
## TeamFT                         0.064938518          -0.064969878   -0.0577300621
## TeamFTA                        0.077003661           0.004736343   -0.0638753907
## TeamFTPCT                     -0.016936223          -0.177541483   -0.0074012062
## TeamOffRebounds               -0.014332575          -0.060389134   -0.0386521955
## TeamTotalRebounds             -0.052416263          -0.059965631   -0.2185974327
## TeamAssists                   -0.033384745          -0.222595212    0.0006884142
## TeamSteals                     0.016707012           0.035155522   -0.0533271359
## TeamBlocks                     0.178200671           0.037788375   -0.1511460518
## TeamTurnovers                  0.074131214          -0.106168146    0.0726446766
## TeamPersonalFouls              0.122282037           0.195017438   -0.0226190966
## OpponentFG                     0.120715447           0.275438081    0.6383041307
## OpponentFGA                    0.519792207           0.424276325    0.2318514751
```

```
## OpponentFGPCT               -0.251623986          -0.005789348    0.5535357935
## Opponent3P                  -0.085432899           0.005903551    0.5138697156
## Opponent3PA                  0.097838949           0.081057601    0.2641728450
## Opponent3PPCT               -0.201309699          -0.068083610    0.4428640799
## OpponentFT                   0.086671729           0.197591588   -0.0123780062
## OpponentFTA                  0.136423744           0.232447345   -0.0312058003
## OpponentFTPCT               -0.082982260          -0.021281750    0.0417935976
## OpponentOffRebounds          1.000000000           0.622115242    0.0095497736
## OpponentTotalRebounds        0.622115242           1.000000000    0.1792668711
## OpponentAssists              0.009549774           0.179266871    1.0000000000
## OpponentSteals               0.081573888          -0.038673692    0.1068223463
## OpponentBlocks               0.096186044           0.258597044    0.1337215898
## OpponentTurnovers            0.017562976           0.073936193   -0.1060361856
## OpponentPersonalFouls        0.071468553           0.020500608   -0.0849725350
##                       OpponentSteals OpponentBlocks OpponentTurnovers
## Differential            -0.187754380  -0.2622526274      0.2743269542
## TeamFG                  -0.102436608  -0.1604696630      0.1552932747
## TeamFGA                 -0.131734964   0.2184838647      0.1981279705
## TeamFGPCT               -0.021724636  -0.3562550337      0.0242548332
## Team3P                  -0.053878305  -0.1117820624      0.0092841059
## Team3PA                 -0.052980367  -0.0580421730      0.0638351465
## Team3PPCT               -0.025131672  -0.0965607977     -0.0488449748
## TeamFT                  -0.001883349  -0.0650555225      0.1369220844
## TeamFTA                  0.006758108  -0.0539735876      0.1697047361
## TeamFTPCT               -0.022033431  -0.0411754626     -0.0354639208
## TeamOffRebounds          0.032697776   0.1571812909      0.1154717115
## TeamTotalRebounds        0.066119486   0.0139248895     -0.0343556886
## TeamAssists             -0.028866867  -0.1657235463      0.1314533533
## TeamSteals               0.055697260  -0.0022307839      0.7308851693
## TeamBlocks               0.028453380  -0.0389785933      0.0313757033
## TeamTurnovers            0.709987911   0.0064638717      0.1885370196
## TeamPersonalFouls        0.064446997   0.0872112484      0.1016935547
## OpponentFG               0.140823916   0.1290769921     -0.1835580089
## OpponentFGA              0.165329579   0.0455658832     -0.2156337333
## OpponentFGPCT            0.036468797   0.1119355214     -0.0480826780
## Opponent3P              -0.011661373  -0.0047464115     -0.0952181989
## Opponent3PA              0.021448140  -0.0495426307     -0.0944428800
## Opponent3PPCT           -0.038356987   0.0354134646     -0.0421344973
## OpponentFT               0.077614062   0.1014221807      0.0157785673
## OpponentFTA              0.097206119   0.1100637520      0.0386793945
## OpponentFTPCT           -0.022196700   0.0089467648     -0.0520407316
## OpponentOffRebounds      0.081573888   0.0961860439      0.0175629757
## OpponentTotalRebounds   -0.038673692   0.2585970440      0.0739361927
## OpponentAssists          0.106822346   0.1337215898     -0.1060361856
## OpponentSteals           1.000000000   0.0443672204      0.0740678539
## OpponentBlocks           0.044367220   1.0000000000      0.0001223389
```

```
## OpponentTurnovers        0.074067854   0.0001223389      1.0000000000
## OpponentPersonalFouls    0.030766974  -0.0514541037      0.2252310703
##                       OpponentPersonalFouls
## Differential                       0.16902573
## TeamFG                            -0.02311662
## TeamFGA                           -0.10718930
## TeamFGPCT                          0.06015066
## Team3P                            -0.12719701
## Team3PA                           -0.15536393
## Team3PPCT                         -0.02688769
## TeamFT                             0.79353920
## TeamFTA                            0.86639509
## TeamFTPCT                          0.01875708
## TeamOffRebounds                    0.12406311
## TeamTotalRebounds                  0.18914401
## TeamAssists                       -0.02678208
## TeamSteals                         0.07144201
## TeamBlocks                         0.08058276
## TeamTurnovers                      0.13153904
## TeamPersonalFouls                  0.32225852
## OpponentFG                         0.01533421
## OpponentFGA                        0.13678905
## OpponentFGPCT                     -0.08177666
## Opponent3P                        -0.01124781
## Opponent3PA                        0.03964752
## Opponent3PPCT                     -0.04664613
## OpponentFT                         0.21560992
## OpponentFTA                        0.25128964
## OpponentFTPCT                     -0.02997805
## OpponentOffRebounds                0.07146855
## OpponentTotalRebounds              0.02050061
## OpponentAssists                   -0.08497254
## OpponentSteals                     0.03076697
## OpponentBlocks                    -0.05145410
## OpponentTurnovers                  0.22523107
## OpponentPersonalFouls              1.00000000
```

Notice right away – TeamFG is highly correlated. But it's also highly correlated with TeamFGPCT. And that makes sense. A team that doesn't shoot many shots is not going to have a high score differential. But the number of shots taken and the field goal percentage are also highly related. So including both of these measures would be pointless – they would add error without adding much in the way of predictive power.

> **Your turn**: What else do you see? What other values have predictive power and aren't co-correlated?

We can add more just by simply adding them.

```
model2 <- lm(Differential ~ TeamFGPCT + OpponentFGPCT + TeamTotalRebounds + OpponentTo
summary(model2)
```

```
##
## Call:
## lm(formula = Differential ~ TeamFGPCT + OpponentFGPCT + TeamTotalRebounds +
##     OpponentTotalRebounds, data = logs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -44.813  -5.586  -0.109   5.453  60.831
##
## Coefficients:
##                         Estimate Std. Error  t value Pr(>|t|)
## (Intercept)            -3.655461   0.606119   -6.031 1.64e-09 ***
## TeamFGPCT             100.880013   0.560363  180.026  < 2e-16 ***
## OpponentFGPCT         -97.563291   0.565004 -172.677  < 2e-16 ***
## TeamTotalRebounds       0.516176   0.006239   82.729  < 2e-16 ***
## OpponentTotalRebounds  -0.436402   0.006448  -67.679  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.501 on 57511 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.7291, Adjusted R-squared:  0.7291
## F-statistic: 3.87e+04 on 4 and 57511 DF,  p-value: < 2.2e-16
```

Go down the list:

What is the Adjusted R-squared now? What is the p-value and is it less than .05? What is the Residual standard error?

The final thing we can do with this is predict things. Look at our coefficients table. See the Estimates? We can build a formula from that, same as we did with linear regressions.

```
Differential = (TeamFGPCT*100.880013) + (OpponentFGPCT*-97.563291) + (TeamTotalRebounds
```

How does this apply in the real world? Let's pretend for a minute that you are Fred Hoiberg, and you have just been hired as Nebraska's Mens Basketball Coach. Your job is to win conference titles and go deep into the NCAA tournament. To do that, we need to know what attributes of a team should we emphasize. We can do that by looking at what previous Big Ten conference champions looked like.

So if our goal is to predict a conference champion team, we need to know what those teams did. Here's the regular season conference champions in this dataset.

```
logs %>% filter(Team == "Michigan State Spartans" & season == "2018-2019" | Team == "Michigan Sta
```

```
## # A tibble: 1 x 4
##   avgfgpct avgoppfgpct avgtotrebound avgopptotrebound
##      <dbl>       <dbl>         <dbl>            <dbl>
## 1    0.489       0.409          35.3             27.2
```

Now it's just plug and chug.

```
(0.4886133*100.880013) + (0.4090221*-97.563291) + (35.29834*0.516176) + (27.20994*-0.436402) - 3.
```

```
## [1] 12.076
```

So a team with those numbers is going to average scoring 12 more points per game than their opponent.

How does that compare to Nebraska of this past season? The last of the Tim Miles era?

```
logs %>%
  filter(
    Team == "Nebraska Cornhuskers" & season == "2018-2019"
    ) %>%
  summarise(
    avgfgpct = mean(TeamFGPCT),
    avgoppfgpct = mean(OpponentFGPCT),
    avgtotrebound = mean(TeamTotalRebounds),
    avgopptotrebound = mean(OpponentTotalRebounds)
    )
```

```
## # A tibble: 1 x 4
##   avgfgpct avgoppfgpct avgtotrebound avgopptotrebound
##      <dbl>       <dbl>         <dbl>            <dbl>
## 1    0.431       0.423          32.5             34.9
```

```
(0.4305833*100.880013) + (0.4226667*-97.563291) + (32.5*0.516176) + (34.94444*-0.436402) - 3.6554
```

```
## [1] 0.07093015
```

By this model, it predicted we would outscore our opponent by .07 points over the season. So we'd win slightly more than we'd lose. Nebraska's overall record? 19-17.

Chapter 11

Residuals

When looking at a linear model of your data, there's a measure you need to be aware of called residuals. The residual is the distance between what the model predicted and what the real outcome is. So if your model predicted a team would score 38 points per game given their third down conversion percentage, and they score 45, then your residual is 7. If they had scored 31, then their residual would be -7.

Residuals can tell you severals things, but most importantly is if a linear model the right model for your data. If the residuals appear to be random, then a linear model is appropriate. If they have a pattern, it means something else is going on in your data and a linear model isn't appropriate.

Residuals can also tell you who is underperforming and overperforming the model. Let's take a look at an example we've used regularly this semester – third down conversion percentage and penalties.

Let's first attach libraries and get some data. Note: In the rvest steps, I rename the first column because it's blank on the page and then I merge scoring offense to two different tables – third downs and penalties.

```
library(tidyverse)
```

```
offense <- read_csv("data/correlations.csv")
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   OffPoints = col_double(),
##   OffPointsG = col_double(),
##   DefPoints = col_double(),
##   DefPointsG = col_double(),
##   Pen. = col_double(),
```

```
##   Yards = col_double(),
##   `Pen./G` = col_double(),
##   OffConversions = col_double(),
##   OffConversionPct = col_double(),
##   DefConversions = col_double(),
##   DefConversionPct = col_double()
## )
```

First, let's build a linear model and save it as a new dataframe called `fit`.

```
fit <- lm(`OffPointsG` ~ `OffConversionPct`, data = offense)
summary(fit)
```

```
##
## Call:
## lm(formula = OffPointsG ~ OffConversionPct, data = offense)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.3861  -3.5411  -0.5885   2.9011  13.5188
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -4.74024    3.41041   -1.39    0.167
## OffConversionPct  0.85625    0.08479   10.10   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.111 on 128 degrees of freedom
## Multiple R-squared:  0.4434, Adjusted R-squared:  0.4391
## F-statistic:   102 on 1 and 128 DF,  p-value: < 2.2e-16
```

We've seen this output before, but let's review because if you are using scatterplots to make a point, you should do this. First, note the Min and Max residual at the top. A team has underperformed the model by 11.4 points, and a team has overperformed it by 13.5. The median residual, where half are above and half are below, is just slightly under the fit line. Close here is good.

Next: Look at the Adjusted R-squared value. What that says is that 44 percent of a team's scoring output can be predicted by their third down conversion percentage. This is just one year, so that's a little low. If we did this with more years, that would go up.

Last: Look at the p-value. We are looking for a p-value smaller than .05. At .05, we can say that our correlation didn't happen at random. And, in this case, it REALLY didn't happen at random.

What we want to do now is look at those residuals. We can add them to our dataframe like this:

```
offense$predicted <- predict(fit)
offense$residuals <- residuals(fit)
```

Now we can sort our data by those residuals. Sorting in descending order gives us the teams that are overperforming the model.

```
offense %>% arrange(desc(residuals))
```

```
## # A tibble: 130 x 14
##    Name  OffPoints OffPointsG DefPoints DefPointsG  Pen. Yards `Pen./G`
##    <chr>     <dbl>      <dbl>     <dbl>      <dbl> <dbl> <dbl>    <dbl>
##  1 Tole~       525       40.4       397       30.5    99   931      7.6
##  2 Utah~       618       47.5       289       22.2   101   916      7.8
##  3 Syra~       523       40.2       351       27      94   768      7.2
##  4 Okla~       677       48.4       466       33.3    86   855      6.1
##  5 Clem~       664       44.3       197       13.1    73   674      4.9
##  6 Hous~       571       43.9       483       37.2    87   683      6.7
##  7 Miss~       407       33.9       434       36.2    88   802      7.3
##  8 Neva~       404       31.1       350       26.9    77   738      5.9
##  9 Bost~       384       32         308       25.7    75   590      6.3
## 10 West~       483       40.3       326       27.2    85   800      7.1
## # ... with 120 more rows, and 6 more variables: OffConversions <dbl>,
## #   OffConversionPct <dbl>, DefConversions <dbl>, DefConversionPct <dbl>,
## #   predicted <dbl>, residuals <dbl>
```

So looking at this table, what you see here are the teams who scored more than their third down conversion percentage would indicate. Some of those teams were just lucky. Some of those teams were really good at long touchdown plays that didn't need a lot of third downs to get down the field. But these are your overperformers.

But, before we can bestow any validity on it, we need to see if this linear model is appropriate. We've done that some looking at our p-values and R-squared values. But one more check is to look at the residuals themselves. We do that by plotting the residuals with the predictor. We'll get into plotting soon, but for now just seeing it is enough.

The lack of a shape here – the seemingly random nature – is a good sign that a linear model works for our data. If there was a pattern, that would indicate something else was going on in our data and we needed a different model.

Another way to view your residuals is by connecting the predicted value with the actual value.

The blue line here separates underperformers from overperformers.

11.1 Penalties

Now let's look at it where it doesn't work: Penalties.

```
penalties <- offense
```

```
pfit <- lm(OffPointsG ~ Yards, data = penalties)
summary(pfit)
```

```
##
## Call:
## lm(formula = OffPointsG ~ Yards, data = penalties)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.5381  -4.5779  -0.5204   4.2418  17.0543
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 20.897213   2.819227   7.412 1.49e-11 ***
## Yards        0.012220   0.003966   3.082  0.00252 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 6.61 on 128 degrees of freedom
## Multiple R-squared:  0.06907,    Adjusted R-squared:  0.06179
## F-statistic: 9.496 on 1 and 128 DF,  p-value: 0.002521
```

So from top to bottom:

- Our min and max go from -16.5 to positive 17.1
- Our adjusted R-squared is … .06. Not much at all.
- Our p-value is … .002, which is less than than .05.

So what we can say about this model is that it's statistically significant but utterly meaningless. Normally, we'd stop right here – why bother going forward with a predictive model that isn't predictive? But let's do it anyway.

```
penalties$predicted <- predict(pfit)
penalties$residuals <- residuals(pfit)
```

```
penalties %>% arrange(desc(residuals))
```

```
## # A tibble: 130 x 14
##    Name  OffPoints OffPointsG DefPoints DefPointsG  Pen. Yards `Pen./G`
##    <chr>     <dbl>      <dbl>     <dbl>      <dbl> <dbl> <dbl>    <dbl>
##  1 Okla~       677       48.4       466       33.3    86   855      6.1
##  2 Utah~       618       47.5       289       22.2   101   916      7.8
##  3 Clem~       664       44.3       197       13.1    73   674      4.9
##  4 Alab~       684       45.6       271       18.1    87   796      5.8
##  5 Hous~       571       43.9       483       37.2    87   683      6.7
##  6 UCF         562       43.2       295       22.7    97   848      7.5
##  7 Memp~       601       42.9       447       31.9   101   833      7.2
##  8 Ohio        521       40.1       320       24.6    64   652      4.9
##  9 Syra~       523       40.2       351       27      94   768      7.2
## 10 West~       483       40.3       326       27.2    85   800      7.1
## # ... with 120 more rows, and 6 more variables: OffConversions <dbl>,
## #   OffConversionPct <dbl>, DefConversions <dbl>, DefConversionPct <dbl>,
## #   predicted <dbl>, residuals <dbl>
```

So our model says Oklahoma *should* only be scoring 31.3 points per game given how many penalty yards per game, but they're really scoring 48.4. Oy. What happens if we plot those residuals?

Well … it actually says that a linear model is appropriate. Which an important lesson – just because your residual plot says a linear model works here, that doesn't say your linear model is good. There are other measures for that, and you need to use them.

Here's the segment plot of residuals – you'll see some really long lines. That's a bad sign.

50
40
30
20
OffPointsG
500
700
900
Yards

Chapter 12

Z scores

Z scores are a handy way to standardize scores so you can compare things across groupings. In our case, we may want to compare teams by year, or era. We can use z scores to answer questions like who was the greatest X of all time, because a Z score can put them in context to their era.

We can also use z scores to ask how much better is team A from team B.

So let's use Nebraska basketball, which if you haven't been reading lately is at a bit of a crossroads.

A Z score is a measure of how far a number is from the population mean of that number. An easier way to say that – how different is my grade from the average grade in the class. The formula for calculating a Z score is `(MyScore - AverageScore)/Standard Deviation of Scores`. The standard deviation is a number calculated to show the amount of variation in a set of data. In a normal distribution, 68 percent of all scores will be within 1 standard deviation, 95 percent will be within 2 and 99 within 3.

12.1 Calculating a Z score in R

```
library(tidyverse)
```

Let's look at the current state of Nebraska basketball using the same logs data we've been using, but for this season so far.

```
gamelogs <- read_csv("data/logs20.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
```

```
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )

## See spec(...) for full column specifications.
```

The first thing we need to do is select some fields we think represent team quality:

```
teamquality <- gamelogs %>%
  select(Conference, Team, TeamFGPCT, TeamTotalRebounds, OpponentFGPCT, OpponentTotalRe
```

And since we have individual game data, we need to collapse this into one record for each team. We do that with … group by.

```
teamtotals <- teamquality %>%
  group_by(Conference, Team) %>%
  summarise(
    FGAvg = mean(TeamFGPCT),
    ReboundAvg = mean(TeamTotalRebounds),
    OppFGAvg = mean(OpponentFGPCT),
    OffRebAvg = mean(OpponentTotalRebounds)
    )
```

To calculate a Z score in R, the easiest way is to use the scale function in base R. To use it, you use `scale(FieldName, center=TRUE, scale=TRUE)`. The center and scale indicate if you want to subtract from the mean and if you want to divide by the standard deviation, respectively. We do.

When we have multiple Z Scores, it's pretty standard practice to add them together into a composite score. That's what we're doing at the end here with `TotalZscore`. Note: We have to invert OppZscore and OppRebZScore by multiplying it by a negative 1 because the lower someone's opponent number is, the better.

```
teamzscore <- teamtotals %>%
  mutate(
    FGzscore = as.numeric(scale(FGAvg, center = TRUE, scale = TRUE)),
    RebZscore = as.numeric(scale(ReboundAvg, center = TRUE, scale = TRUE)),
    OppZscore = as.numeric(scale(OppFGAvg, center = TRUE, scale = TRUE)) * -1,
    OppRebZScore = as.numeric(scale(OffRebAvg, center = TRUE, scale = TRUE)) * -1,
    TotalZscore = FGzscore + RebZscore + OppZscore + OppRebZScore
  )
```

So now we have a dataframe called `teamzscore` that has 353 basketball teams with Z scores. What does it look like?

```
head(teamzscore)
```

```
## # A tibble: 6 x 11
## # Groups:   Conference [1]
##   Conference Team  FGAvg ReboundAvg OppFGAvg OffRebAvg FGzscore RebZscore
##   <chr>      <chr> <dbl>      <dbl>    <dbl>     <dbl>    <dbl>     <dbl>
## 1 A-10       Davi~ 0.454       31.1    0.437      30.4    0.507   -0.623
## 2 A-10       Dayt~ 0.525       32.5    0.413      29.0    2.56     0.0380
## 3 A-10       Duqu~ 0.444       32.4    0.427      32.4    0.222   -0.0145
## 4 A-10       Ford~ 0.380       30.0    0.403      34.1   -1.61    -1.14
## 5 A-10       Geor~ 0.422       33.5    0.441      30.7   -0.398    0.480
## 6 A-10       Geor~ 0.424       30.5    0.451      32.7   -0.341   -0.904
## # ... with 3 more variables: OppZscore <dbl>, OppRebZScore <dbl>,
## #   TotalZscore <dbl>
```

A way to read this – a team at zero is precisely average. The larger the positive number, the more exceptional they are. The larger the negative number, the more truly terrible they are.

So who are the best teams in the country?

```
teamzscore %>% arrange(desc(TotalZscore))
```

```
## # A tibble: 353 x 11
## # Groups:   Conference [32]
##    Conference Team  FGAvg ReboundAvg OppFGAvg OffRebAvg FGzscore RebZscore
##    <chr>      <chr> <dbl>      <dbl>    <dbl>     <dbl>    <dbl>     <dbl>
##  1 Big West   UC-I~ 0.473       36.6    0.390      27.1    1.60      2.23
##  2 Big 12     Kans~ 0.482       35.9    0.378      29.0    2.38      1.12
##  3 WCC        Gonz~ 0.517       37.6    0.422      28.4    1.65      1.94
##  4 Big Ten    Mich~ 0.460       37.8    0.379      29.7    1.41      1.59
##  5 Southland  Step~ 0.490       34.2    0.427      26.6    1.71      1.02
##  6 OVC        Murr~ 0.477       35.3    0.401      29.2    1.31      1.36
##  7 Summit     Sout~ 0.492       35.5    0.423      31.3    1.51      1.60
##  8 A-10       Sain~ 0.457       37.4    0.403      30.5    0.598     2.23
##  9 A-10       Dayt~ 0.525       32.5    0.413      29.0    2.56      0.0380
## 10 Horizon    Wrig~ 0.463       37.1    0.416      33.5    1.53      2.28
## # ... with 343 more rows, and 3 more variables: OppZscore <dbl>,
## #   OppRebZScore <dbl>, TotalZscore <dbl>
```

Don't sleep on the Anteaters come tournament time!

But closer to home, how is Nebraska doing.

```
teamzscore %>%
  filter(Conference == "Big Ten") %>%
  arrange(desc(TotalZscore))
```

```
## # A tibble: 14 x 11
## # Groups:   Conference [1]
##    Conference Team  FGAvg ReboundAvg OppFGAvg OffRebAvg FGzscore RebZscore
##    <chr>      <chr> <dbl>      <dbl>    <dbl>     <dbl>    <dbl>     <dbl>
##  1 Big Ten    Mich~ 0.460       37.8    0.379      29.7    1.41     1.59
##  2 Big Ten    Rutg~ 0.449       37      0.385      31.1    0.789    1.20
##  3 Big Ten    Ohio~ 0.446       33.8    0.397      28.2    0.639   -0.307
##  4 Big Ten    Illi~ 0.442       36.4    0.416      29.1    0.402    0.895
##  5 Big Ten    Indi~ 0.442       34.7    0.423      29.3    0.377    0.0934
##  6 Big Ten    Mich~ 0.463       33.4    0.423      32.0    1.60    -0.513
##  7 Big Ten    Mary~ 0.415       36.4    0.399      32.3   -1.20     0.895
##  8 Big Ten    Penn~ 0.432       35.6    0.411      34.2   -0.195    0.522
##  9 Big Ten    Iowa~ 0.451       34.3    0.428      32.6    0.892   -0.0698
## 10 Big Ten    Purd~ 0.418       33.8    0.410      29.3   -0.994   -0.304
## 11 Big Ten    Minn~ 0.422       35.1    0.412      33.4   -0.758    0.312
## 12 Big Ten    Wisc~ 0.426       31.3    0.410      32.0   -0.552   -1.53
## 13 Big Ten    Nort~ 0.417       30.6    0.423      34.6   -1.08    -1.84
## 14 Big Ten    Nebr~ 0.412       32.5    0.446      42     -1.34    -0.939
## # ... with 3 more variables: OppZscore <dbl>, OppRebZScore <dbl>,
## #   TotalZscore <dbl>
```

So, as we can see, with our composite Z Score, Nebraska is … not good. Not good at all.

Chapter 13

Intro to ggplot

With `ggplot2`, we dive into the world of programmatic data visualization. The `ggplot2` library implements something called the grammar of graphics. The main concepts are:

- aesthetics - which in this case means the data which we are going to plot
- geometries - which means the shape the data is going to take
- scales - which means any transformations we might make on the data
- facets - which means how we might graph many elements of the same dataset in the same space
- layers - which means how we might lay multiple geometries over top of each other to reveal new information.

Hadley Wickam, who is behind all of the libraries we have used in this course to date, wrote about his layered grammar of graphics in this 2009 paper that is worth your time to read.

Here are some `ggplot2` resources you'll want to keep handy:

- The ggplot documentation.
- The ggplot cookbook

Let's dive in using data we've already seen before – football attendance. This workflow will represent a clear picture of what your work in this class will be like for much of the rest of the semester. One way to think of this workflow is that your R Notebook is now your digital sketchbook, where you will try different types of visualizations to find ones that work. Then, you will export your work into a program like Illustrator to finish the work.

To begin, we'll import the `ggplot2` and `dplyr` libraries. We'll read in the data, then create a new dataframe that represents our attendance data, similar to what we've done before.

```r
library(tidyverse)
```

```r
attendance <- read_csv('data/attendance.csv')
```

```
## Parsed with column specification:
## cols(
##   Institution = col_character(),
##   Conference = col_character(),
##   `2013` = col_double(),
##   `2014` = col_double(),
##   `2015` = col_double(),
##   `2016` = col_double(),
##   `2017` = col_double(),
##   `2018` = col_double()
## )
```

First, let's get a top 10 list by announced attendance this last season. We'll use the same tricks we used in the filtering assignment.

```r
attendance %>%
  arrange(desc(`2018`)) %>%
  top_n(10) %>%
  select(Institution, `2018`)
```

```
## Selecting by 2018
```

```
## # A tibble: 10 x 2
##    Institution `2018`
##    <chr>        <dbl>
##  1 Michigan    775156
##  2 Penn St.    738396
##  3 Ohio St.    713630
##  4 Alabama     710931
##  5 LSU         705733
##  6 Texas A&M   698908
##  7 Tennessee   650887
##  8 Georgia     649222
##  9 Nebraska    623240
## 10 Oklahoma    607146
```

That looks good, so let's save it to a new data frame and use that data frame instead going forward.

```r
top10 <- attendance %>%
  arrange(desc(`2018`)) %>%
  top_n(10) %>%
  select(Institution, `2018`)
```

```
## Selecting by 2018
```

13.1 The bar chart

The easiest thing we can do is create a simple bar chart of our data. **Bar charts show magnitude. They invite you to compare how much more or less one thing is compared to others.**

We could, for instance, create a bar chart of the total attendance. To do that, we simply tell `ggplot2` what our dataset is, what element of the data we want to make the bar chart out of (which is the aesthetic), and the geometry type (which is the geom). It looks like this:

```
ggplot(top10, aes(x=Institution)) + geom_bar()
```

Note: attendance is our data, `aes` means aesthetics, `x=Institution` explicitly tells `ggplot2` that our x value – our horizontal value – is the Institution field from the data, and then we add on the `geom_bar()` as the geometry. And what do we get when we run that?

```
ggplot(top10, aes(x=Institution)) + geom_bar()
```

We get … weirdness. We expected to see bars of different sizes, but we get all with a count of 1. What gives? Well, this is the default behavior. What we have here is something called a histogram, where `ggplot2` helpfully counted up the number of times the Institution appears and counted them up. Since we only have one record per Institution, the count is always 1. How do we fix this? By adding `weight` to our aesthetic.

```
ggplot(top10, aes(x=Institution, weight=`2018`)) +
  geom_bar()
```

Closer. But … what order is that in? And what happened to our count numbers on the left? Why are they in scientific notation?

Let's deal with the ordering first. `ggplot2`'s default behavior is to sort the data by the x axis variable. So it's in alphabetical order. To change that, we have to `reorder` it. With `reorder`, we first have to tell `ggplot` what we are reordering, and then we have to tell it HOW we are reordering it. So it's reorder(FIELD, SORTFIELD).

```
ggplot(top10, aes(x=reorder(Institution, `2018`), weight=`2018`)) + geom_bar()
```

Better. We can argue about if the right order is smallest to largest or largest to smallest. But this gets us close. By the way, to sort it largest to smallest, put a negative sign in front of the sort field.

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) + geom_bar()
```

13.2 Scales

To fix the axis labels, we need try one of the other main elements of the `ggplot2` library, which is transform a scale. More often that not, that means doing something like putting it on a logarithmic scale or some other kind of transformation. In this case, we're just changing how it's represented. The default in `ggplot2` for large values is to express them as scientific notation. Rarely ever is that useful in our line of work. So we have to transform them into human readable numbers.

The easiest way to do this is to use a library called `scales` and it's already installed.

```
library(scales)
```

To alter the scale, we add a piece to our plot with `+` and we tell it which scale is getting altered and what kind of data it is. In our case, our Y axis is what is needing to be altered, and it's continuous data (meaning it can be any number between x and y, vs discrete data which are categorical). So we need to add `scale_y_continuous` and the information we want to pass it is to alter the labels with a function called `comma`.

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) +
  geom_bar() +
  scale_y_continuous(labels=comma)
```

Better.

13.3 Styling

We are going to spend a lot more time on styling, but let's add some simple labels to this with a new bit called `labs` which is short for labels.

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) +
  geom_bar() +
  scale_y_continuous(labels=comma) +
  labs(title="Top 10 Football Programs By Attendance", x="School", y="Attendance")
```

The library has lots and lots of ways to alter the styling – we can programmatically control nearly every part of the look and feel of the chart. One simple way is to apply themes in the library already. We do that the same way we've done other things – we add them. Here's the light theme.

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) +
  geom_bar() +
  scale_y_continuous(labels=comma) +
  labs(title="Top 10 Football Programs By Attendance", x="School", y="Attendance") +
  theme_light()
```

Or the minimal theme:

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) +
  geom_bar() +
  scale_y_continuous(labels=comma) +
  labs(title="Top 10 Football Programs By Attendance", x="School", y="Attendance") +
  theme_minimal()
```

Later on, we'll write our own themes. For now, the built in ones will get us closer to something that looks good.

13.4 One last trick: coord flip

Sometimes, we don't want vertical bars. Maybe we think this would look better horizontal. How do we do that? By adding `coord_flip()` to our code. It does what it says – it inverts the coordinates of the figures.

```
ggplot(top10, aes(x=reorder(Institution, -`2018`), weight=`2018`)) +
  geom_bar() +
  scale_y_continuous(labels=comma) +
  labs(title="Top 10 Football Programs By Attendance", x="School", y="Attendance") +
  theme_minimal() +
  coord_flip()
```

Top 10 Football Programs By Attendance
Oklahoma
Nebraska
Georgia
Tennessee
Texas A&M
LSU
Alabama
Ohio St.
Penn St.
Michigan
School
0.00
200,000.00
400,000.00
600,000.00
800,000.
Attendance

Chapter 14

Stacked bar charts

One of the elements of data visualization excellence is **inviting comparison**. Often that comes in showing **what proportion a thing is in relation to the whole thing**. With bar charts, we're showing magnitude of the whole thing. If we have information about the parts of the whole, **we can stack them on top of each other to compare them, showing both the whole and the components**. And it's a simple change to what we've already done.

```
library(tidyverse)
```

We're going to use a dataset of graduation rates by gender by school in the NCAA. You can get it here.

```
grads <- read_csv('data/grads.csv')
```

```
## Parsed with column specification:
## cols(
##   `Institution name` = col_character(),
##   `Primary Conference in Actual Year` = col_character(),
##   `Cohort year` = col_double(),
##   Gender = col_character(),
##   `Number of completers` = col_double(),
##   Total = col_double()
## )
```

What we have here is the name of the school, the conference, the cohort of when they started school, the gender, the number of that gender that graduated and the total number of graduates in that cohort.

Let's pretend for a moment we're looking at the graduation rates of men and women in the Big 10 Conference and we want to chart that. First, let's work on our data. We need to filter the "Big Ten Conference" school, and we want

the latest year, which is 2009. So we'll create a dataframe called `BIG09` and populate it.

```
BIG09 <- grads %>% filter(`Primary Conference in Actual Year`=="Big Ten Conference") %>
```

Reminder: `head()` will give you a quick look at what your data looks like, but **it will only show you the first six rows**.

```
head(BIG09)
```

```
## # A tibble: 6 x 6
##   `Institution nam~ `Primary Confer~ `Cohort year` Gender `Number of comp~ Total
##   <chr>             <chr>                    <dbl> <chr>             <dbl> <dbl>
## 1 University of Il~ Big Ten Confere~          2009 Men               2973  5940
## 2 University of Il~ Big Ten Confere~          2009 Women             2967  5940
## 3 Northwestern Uni~ Big Ten Confere~          2009 Men                963  1974
## 4 Northwestern Uni~ Big Ten Confere~          2009 Women             1011  1974
## 5 Indiana Universi~ Big Ten Confere~          2009 Men               2667  5626
## 6 Indiana Universi~ Big Ten Confere~          2009 Women             2959  5626
```

Building on what we learned in the last chapter, we know we can turn this into a bar chart with an x value, a weight and a geom_bar. What're going to add is a `fill`. The `fill` will stack bars on each other based on which element it is. In this case, we can fill the bar by Gender, which means it will stack the number of male graduates on top of the number of female graduates and we can see how they compare.

```
ggplot(BIG09, aes(x=reorder(`Institution name`, -Total), weight=`Number of completers`
```

What's the problem with this chart?

Let me ask a different question – which schools have larger differences in male and female graduation rates? Can you compare Illnois to Northwestern? Not really. We've charted the total numbers. We need the percentage of the whole.

> **YOUR TURN**: Using what you know – hint: mutate – how could you chart this using percents of the whole instead of counts?

Chapter 15

Waffle charts

Pie charts are the devil. They should be an instant F in any data visualization class. I'll give you an example of why.

What's the racial breakdown of journalism majors at UNL?

Here it is in a pie chart:

```
library(tidyverse)

enrollment <- read.csv("~/Box/Courses/JOUR407-Data-Visualization/Data/collegeenrollment.csv")

jour <- filter(enrollment, MajorName == "Journalism")

jdf <- jour %>%
group_by(Race) %>%
summarise(
       total=sum(Count)) %>%
select(Race, total) %>%
filter(total != 0)

ggplot(jdf, aes(x="", y=total, fill=Race)) + geom_bar(width = 1, stat = "identity") + coord_polar
```

You can see, it's pretty white. But … what about beyond that? How carefully can you evaluate angles and area?

Not well.

So let's introduce a better way: The Waffle Chart. Some call it a square pie chart. I personally hate that. Waffles it is.

A waffle chart is designed to show you parts of the whole – proportionality. How many yards on offense come from rushing or passing. How many singles, doubles, triples and home runs make up a teams hits. How many shots a basketball team takes are two pointers versus three pointers.

First, install the library in the console:

```
install.packages('waffle')
```

Now load it:

```
library(waffle)
```

Let's look at the debacle that was Nebraska vs. Ohio State this fall in Football. Here's the box score, which we'll use for this walkthrough.

The easiest way to do waffle charts is to make vectors of your data and plug them in. To make a vector, we use the `c` or concatenate function, something we've done before.

So let's look at offense. Rushing vs passing.

```
nu <- c("Rushing"=184, "Passing"=47)
oh <- c("Rushing"=368, "Passing"=212)
```

So what does the breakdown of the night look like?

The waffle library can break this down in a way that's easier on the eyes than a pie chart. We call the library, add the data, specify the number of rows, give it a title and an x value label, and to clean up a quirk of the library, we've got to specify colors.

```
waffle(nu, rows = 10, title="Nebraska's offense", xlab="1 square = 1 yard", colors = c("black", '
```

Or, we could make this two teams in the same chart.

```
passing <- c("Nebraska"=47, "Ohio State"=212)
```

```
waffle(passing, rows = 10, title="Nebraska vs Ohio State: passing", xlab="1 square = 1 yard", col
```

15.1 Waffle Irons

So what does it look like if we compare the two teams using the two vectors in the same chart? To do that – and I am not making this up – you have to create a waffle iron. Get it? Waffle charts? Iron?

```
iron(
 waffle(nu, rows = 10, title="Nebraska's offense", xlab="1 square = 1 yard", colors = c
 waffle(oh, rows = 10, title="Ohio State's offense", xlab="1 square = 1 yard", colors =
)
```

What do you notice about this chart? Notice how the squares aren't the same size? Well, Ohio State outgained Nebraska by a long way. So the squares aren't the same size because the numbers aren't the same. We can fix that by adding an unnamed padding number so the number of shots add up to the same thing. Let's make the total for everyone be 580, Ohio State's total yards of offense. So to do that, we need to add a padding of 349 to Nebraska. REMEMBER: Don't name it or it'll show up in the legend.

```
nu <- c("Rushing"=184, "Passing"=47, 349)
oh <- c("Rushing"=368, "Passing"=212, 0)
```

Now, in our waffle iron, if we don't give that padding a color, we'll get an error. So we need to make it white. Which, given our white background, means it will disappear.

```
iron(
 waffle(nu, rows = 10, title="Nebraska's offense", xlab="1 square = 1 yard", colors = c
```

```
 waffle(oh, rows = 10, title="Ohio State's offense", xlab="1 square = 1 yard", colors = c("black'
)
```

One last thing we can do is change the 1 square = 1 yard bit – which makes the squares really small in this case – by dividing our vector. Remember what you learned in Swirl about math on vectors?

```
iron(
 waffle(nu/2, rows = 10, title="Nebraska's offense", xlab="1 square = 2 yards", colors = c("black
 waffle(oh/2, rows = 10, title="Ohio State's offense", xlab="1 square = 2 yards", colors = c("bla
)
```

News flash: Ohio State crushed Nebraska.

Chapter 16

Line charts

So far, we've talked about bar charts – stacked or otherwise – are good for showing relative size of a thing compared to another thing. Stacked Bars and Waffle charts are good at showing proportions of a whole.

Line charts are good for showing change over time.

Let's look at how we can answer this question: Why was Nebraska terrible at basketball last season?

Let's start getting all that we need. We can use the tidyverse shortcut.

```
library(tidyverse)
```

Now we'll import the data you need. Mine looks like this:

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

This data has every game from every team in it, so we need to use filtering to limit it, because we just want to look at Nebraska. If you don't remember, flip back to chapter 5.

```
nu <- logs %>% filter(Team == "Nebraska Cornhuskers")
```

Because this data has just Nebraska data in it, the dates are formatted correctly, and the data is long data (instead of wide), we have what we need to make line charts.

Line charts, unlike bar charts, do have a y-axis. So in our ggplot step, we have to define what our x and y axes are. In this case, the x axis is our Date – the most common x axis in line charts is going to be a date of some variety – and y in this case is up to us. We've seen from previous walkthroughs that how well a team shoots the ball has a lot to do with how well a team does in a season, so let's chart that.

```
ggplot(nu, aes(x=Date, y=TeamFGPCT)) + geom_line()
```

See a problem here? Note the Y axis doesn't start with zero. That makes this look worse than it is (and that February swoon is pretty bad). To make the axis what you want, you can use `scale_x_continuous` or `scale_y_continuous` and pass in a list with the bottom and top value you want. You do that like this:

```
ggplot(nu, aes(x=Date, y=TeamFGPCT)) + geom_line() + scale_y_continuous(limits = c(0, .6))
```

Note also that our X axis labels are automated. It knows it's a date and it just labels it by month.

16.1 This is too simple.

With datasets, we want to invite comparison. So let's answer the question visually. Let's put two lines on the same chart. How does Nebraska compare to Michigan State and Purdue, the eventual regular season co-champions?

```
msu <- logs %>% filter(Team == "Michigan State Spartans")
```

In this case, because we have two different datasets, we're going to put everything in the geom instead of the ggplot step. We also have to explicitly state what dataset we're using by saying `data=` in the geom step.

First, let's chart Nebraska. Read carefully. First we set the data. Then we set our aesthetic. Unlike bars, we need an X and a Y variable. In this case, our X is the date of the game, Y is the thing we want the lines to move with. In this case, the Team Field Goal Percentage – TeamFGPCT.

```
ggplot() + geom_line(data=nu, aes(x=Date, y=TeamFGPCT), color="red")
```

Now, by using +, we can add Michigan State to it. REMEMBER COPY AND PASTE IS A THING. Nothing changes except what data you are using.

```
ggplot() + geom_line(data=nu, aes(x=Date, y=TeamFGPCT), color="red") + geom_line(data=m
```

Let's flatten our lines out by zeroing the Y axis.

```
ggplot() + geom_line(data=nu, aes(x=Date, y=TeamFGPCT), color="red") + geom_line(data=msu, aes(x=
```

So visually speaking, the difference between Nebraska and Michigan State's season is that Michigan State stayed mostly on an even keel, and Nebraska went on a two month swoon.

16.2 But what if I wanted to add a lot of lines.

Fine. How about all Power Five Schools? This data for example purposes. You don't have to do it.

```
powerfive <- c("SEC", "Big Ten", "Pac-12", "Big 12", "ACC")

p5conf <- logs %>% filter(Conference %in% powerfive)
```

I can keep layering on layers all day if I want. And if my dataset has more than one team in it, I need to use the `group` command. And, the layering comes in order – so if you're going to layer a bunch of lines with a smaller group of lines, you want the bunch on the bottom. So to do that, your code stacks from the bottom. The first geom in the code gets rendered first. The second gets layered on top of that. The third gets layered on that and so on.

```
ggplot() + geom_line(data=p5conf, aes(x=Date, y=TeamFGPCT, group=Team), color="grey") + geom_line
```

What do we see here? How has Nebraska and Michigan State's season evolved against all the rest of the teams in college basketball?

But how does that compare to the average? We can add that pretty easily by creating a new dataframe with it and add another geom_line.

```
average <- logs %>% group_by(Date) %>% summarise(mean_shooting=mean(TeamFGPCT))
```

```
ggplot() + geom_line(data=p5conf, aes(x=Date, y=TeamFGPCT, group=Team), color="grey") +
```

0.6
0.4
0.2
0.0
TeamFGPCT
Nov
Dec
Jan
Feb
Mar
Date

Chapter 17

Step charts

Step charts are **a method of showing progress** toward something. They combine showing change over time – **cumulative change over time** – with magnitude. They're good at inviting comparison.

There's great examples out there. First is the Washignton Post looking at Lebron passing Jordan's career point total. Another is John Burn-Murdoch's work at the Financial Times (which is paywalled) about soccer stars. Here's an example of his work outside the paywall.

To replicate this, we need cumulative data – data that is the running total of data at a given point. So think of it this way – Nebraska scores 50 points in a basketball game and then 50 more the next, their cumulative total at two games is 100 points.

Step charts can be used for all kinds of things – showing how a player's career has evolved over time, how a team fares over a season, or franchise history. Let's walk through an example.

```
library(tidyverse)
```

And we'll use our basketball log data.

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
```

```
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

Here we're going to look at the scoring differential of teams. If you score more than your opponent, you win. So it stands to reason that if you score a lot more than your opponent over the course of a season, you should be very good, right? Let's see.

The first thing we're going to do is calculate that differential. Then, we'll group it by the team. After that, we're going to summarize using a new function called `cumsum` or cumulative sum – the sum for each game as we go forward. So game 1's cumsum is the differential of that game. Game 2's cumsum is Game 1 + Game 2. Game 3 is Game 1 + 2 + 3 and so on.

```
difflogs <- logs %>%
  mutate(Differential = TeamScore - OpponentScore) %>%
  group_by(Team) %>%
  mutate(CumDiff = cumsum(Differential))
```

Now that we have the cumulative sum for each, let's filter it down to just Big Ten teams.

```
bigdiff <- difflogs %>% filter(Conference == "Big Ten")
```

The step chart is it's own geom, so we can employ it just like we have the others. It works almost exactly the same as a line chart, but it uses the cumulative sum instead of a regular value and, as the name implies, creates a step like shape to the line instead of a curve.

```
ggplot() + geom_step(data=bigdiff, aes(x=Date, y=CumDiff, group=Team))
```

Let's try a different element of the aesthetic: color, but this time inside the aesthetic. Last time, we did the color outside. When you put it inside, you pass it a column name and ggplot will color each line based on what thing that is, and it will create a legend that labels each line that thing.

```
ggplot() + geom_step(data=bigdiff, aes(x=Date, y=CumDiff, group=Team, color=Team))
```

From this, we can see two teams in the Big Ten had negative point differentials last season – Illinois and Rutgers.

Let's look at those top teams plus Nebraska.

```
nu <- bigdiff %>% filter(Team == "Nebraska Cornhuskers")
mi <- bigdiff %>% filter(Team == "Michigan Wolverines")
ms <- bigdiff %>% filter(Team == "Michigan State Spartans")
```

Let's introduce a couple of new things here. First, note when I take the color OUT of the aesthetic, the legend disappears.

The second thing I'm going to add is the annotation layer. In this case, I am adding a text annotation layer, and I can specify where by adding in a x and a y value where I want to put it. This takes some finesse. After that, I'm going to add labels and a theme.

```
ggplot() +
  geom_step(data=bigdiff, aes(x=Date, y=CumDiff, group=Team), color="light grey") +
  geom_step(data=nu, aes(x=Date, y=CumDiff, group=Team), color="red") +
  geom_step(data=mi, aes(x=Date, y=CumDiff, group=Team), color="blue") +
  geom_step(data=ms, aes(x=Date, y=CumDiff, group=Team), color="green") +
  annotate("text", x=(as.Date("2018-12-10")), y=220, label="Nebraska") +
  labs(x="Date", y="Cumulative Point Differential", title="Nebraska was among the leagu
  theme_minimal()
```

Nebraska was among the league's most dominant
Before the misseason skid, Nebraska was at the top of the Big Ten in point differential
Cumulative Point Differential
500
400
300
200
100
0
−100
Nebraska
Nov
Dec
Jan
Feb
Mar
Date
Source: Sports−Reference.com | By Matt Waite

Chapter 18

Ridge charts

Ridgeplots are useful for when you want to show how different groupings compare with a large number of datapoints. So let's look at how we do this, and in the process, we learn about ggplot extensions. The extensions add functionality to ggplot, which doesn't out of the box have ridgeplots (sometimes called joyplots).

In the console, run this: `install.packages("ggridges")`

Now we can add those libraries.

```
library(tidyverse)
library(ggridges)
```

So for this, let's look at every basketball game played since the 2014-15 season. That's more than 28,000 basketball games. Download that data here.

```
logs <- read_csv("data/logs1519.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

So I want to group teams by wins. Wins are the only thing that matter – ask Tim Miles. So our data has a column called W_L that lists if the team won or lost. The problem is it doesn't just say W or L. If the game went to overtime, it lists that. That complicates counting wins. And, with ridgeplots, I want to be be able to separate EVERY GAME by how many wins the team had over a SEASON. So I've got some work to do.

First, here's a trick to find a string of text and make that. It's called `grepl` and the basic syntax is grepl for this string in this field and then do what I tell you. In this case, we're going to create a new field called winloss look for W or L (and ignore any OT notation) and give wins a 1 and losses a 0.

```
winlosslogs <- logs %>% mutate(winloss = case_when(
  grepl("W", W_L) ~ 1,
  grepl("L", W_L) ~ 0)
)
```

Now I'm going to add up all the winlosses for each team, which should give me the number of wins for each team.

```
winlosslogs %>% group_by(Team, Conference, season) %>% summarise(TeamWins = sum(winloss
```

Now that I have season win totals, I can join that data back to my log data so each game has the total number of wins in each season.

```
winlosslogs %>% left_join(teamseasonwins, by=c("Team", "Conference", "season")) -> win
```

Now I can use that same `case_when` logic to create some groupings. So I want to group teams together by how many wins they had over the season. For no good reason, I started with more than 25 wins, then did groups of 5 down to 10 wins. If you had fewer than 10 wins, God help your program.

The way to create a new field based on groupings like that is to use `case_when`, which says, basically, when This Thing Is True, Do This. So in our case, we're going to pass a couple of logical statements that when they are both true, our data gets labeled how we want to label it. So we `mutate` a field called grouping and then use `case_when`.

```
wintotallogs %>% mutate(grouping = case_when(
  TeamWins > 25 ~ "More than 25 wins",
  TeamWins >= 20 & TeamWins <=25 ~ "20-25 wins",
  TeamWins >= 15 & TeamWins <=19 ~ "15-19 wins",
  TeamWins >= 10 & TeamWins <=14 ~ "10-14 wins",
  TeamWins < 10 ~ "Less than 10 wins")
) -> wintotalgroupinglogs
```

So my `wintotalgroupinglogs` table has each game, with a field that gives the total number of wins each team had that season and labeling each game with

one of five groupings. I could use `dplyr` to do group_by on those five groups and find some things out about them, but ridgeplots do that visually.

Let's look at the differences in rebounds by those five groups. Do teams that win more than 25 games rebound better than teams that win fewer games?

The answer might surprise you.

```
ggplot(wintotalgroupinglogs, aes(x = TeamTotalRebounds, y = grouping, fill = grouping)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 0.88
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density_ridges).
```

Answer? Not really. Game to game, maybe. Over five seasons? The differences are indistinguishable.

How about assists?

```
ggplot(wintotalgroupinglogs, aes(x = TeamAssists, y = grouping, fill = grouping)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 0.601
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density_ridges).
```

There's a little better, especially between top and bottom.

```
ggplot(wintotalgroupinglogs, aes(x = Team3PPCT, y = grouping, fill = grouping)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 0.0156
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density_ridges).
```

If you've been paying attention this semester, you know what's coming next.

```
ggplot(wintotalgroupinglogs, aes(x = TeamFGPCT, y = grouping, fill = grouping)) +
  geom_density_ridges() +
  theme_ridges() +
  theme(legend.position = "none")
```

```
## Picking joint bandwidth of 0.0102
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density_ridges).
```

grouping
More than 25 wins
Less than 10 wins
20−25 wins
15−19 wins
10−14 wins
0.2
0.4
0.6
0.8
TeamFGPCT

Chapter 19

Lollipop charts

Second to my love of waffle charts because I'm always hungry, lollipop charts are an excellently named way of showing the difference between two things on a number line – a start and a finish, for instance. Or the difference between two related things. Say, turnovers and assists. They aren't a geom, specifically, but you can assemble them out of points and segments, which are geoms.

```
library(tidyverse)
```

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

For the first example, let's look at the difference between a team's shooting performance and the conference's shooting performance as a whole. To get this, we're going to add up all the shots made by the conference, all the attempts taken by the conference, and then mutate a percentage based on that.

```
conferenceshooting <- logs %>%
  group_by(Conference) %>%
  summarise(totalshots = sum(TeamFG), totalattempts = sum(TeamFGA)) %>%
  mutate(conferenceshootingpct = totalshots/totalattempts)
```

Now I'm going to do the same with teams.

```
teamshooting <- logs %>%
  group_by(Team, Conference) %>%
  summarise(totalshots = sum(TeamFG), totalattempts = sum(TeamFGA)) %>%
  mutate(teamshootingpct = totalshots/totalattempts)
```

The last thing I need to do is join them together. So each team will have the conference shooting percentage as well as their own.

```
shooting <- teamshooting %>% left_join(conferenceshooting, by="Conference")
```

I have every team in college basketball, but that's insane.

```
big10 <- shooting %>% filter(Conference == "Big Ten")
```

So this takes a little doing, but the logic is pretty clear in the end.

A lollipop chart is made up of two things – a line between two points, and two points. So we need a geom_segment and two geom_points. And because they get layered starting at the bottom, our segment is first. A geom segment is made up of two things – an x and a y value, and an x and y end. In this case, our x and xend are the same – the Team – and our y and yend are our two stats. For our points, both x values are the Team and the y is the different stats. What that does is put each point on the same line.

```
ggplot(big10) +
  geom_segment(aes(x=Team, xend=Team, y=teamshootingpct, yend=conferenceshootingpct), c
  geom_point(aes(x=Team, y=teamshootingpct), color="red") +
  geom_point(aes(x=Team, y=conferenceshootingpct), color="blue") +
  coord_flip()
```

We can do better by changing the order of the teams by their shooting performance and giving it some theme love.

```
ggplot(big10) +
  geom_segment(aes(x=reorder(Team, teamshootingpct), xend=Team, y=teamshootingpct, yend=conferenc
  geom_point(aes(x=reorder(Team, teamshootingpct), y=teamshootingpct), color="red") +
  geom_point(aes(x=reorder(Team, teamshootingpct), y=conferenceshootingpct), color="blue") +
  coord_flip() +
   labs(x="", y="Shooting percentage vs league average", title="Except Purdue, shooting predicted
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 1),
    plot.subtitle = element_text(hjust = 1.3),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank(),
    axis.text = element_text(size = 7),
    axis.ticks = element_blank()
  )
```

What if we wanted to order them by wins? Our data has a column called W_L that lists if the team won or lost. The problem is it doens't just say W or L. If the game went to overtime, it lists that. That complicates counting wins. Here's a trick to find a string of text and make that. It's called `grepl` and the basic syntax is grepl for this string in this field and then do what I tell you. In this case, we're going to create a new field called winloss look for W or L (and ignore any OT notation) and give wins a 1 and losses a 0.

```
winlosslogs <- logs %>% mutate(winloss = case_when(
  grepl("W", W_L) ~ 1,
  grepl("L", W_L) ~ 0)
)
```

So let's look at turnovers and assists. We'll call it give and take. Does the difference between those two things indicate something when we sort them by wins?

```
giveandtake <- winlosslogs %>% group_by(Conference, Team) %>% summarise(turnovers = su
```

```
big10gt <- giveandtake %>% filter(Conference == "Big Ten")
```

```
ggplot(big10gt) +
  geom_segment(aes(x=reorder(Team, wins), xend=Team, y=turnovers, yend=assists), color=
  geom_point(aes(x=reorder(Team, wins), y=turnovers), color="red") +
  geom_point(aes(x=reorder(Team, wins), y=assists), color="blue") +
  coord_flip()
```

Short answer: Not really. Something you need to get used to in data visualization – not everything works. Sometimes you find things, sometimes you don't. Don't publish a graphic that doesn't find anything. Let it stay in your notebook as an idea that didn't pan out.

Chapter 20

Scatterplots

In several chapters of this book, we've been fixated on the Nebraska basketball team's shooting percentage, which took a nose dive during the season and ultimately doomed Tim Miles job. The question is … does it matter?

This is what we're going to start to answer today. And we'll do it with scatterplots and correlations.

First, we need libraries and data.

```
library(tidyverse)
```

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

To do this, we need all teams and their season stats. How much, over the course of a season, does a thing matter? That's the question you're going to answer.

In our case, we want to know how much does shooting percentage influence wins? How much difference can we explain in wins with shooting percentage? We're going to total up the number of wins each team has and their season shooting percentage in one swoop.

Let's borrow from our ridgecharts work to get the correct wins and losses totals for each team.

```
winlosslogs <- logs %>% mutate(winloss = case_when(
  grepl("W", W_L) ~ 1,
  grepl("L", W_L) ~ 0)
)
```

Now we can get a dataframe together that gives us the total wins for each team, and the total shots taken and made, which let's us calculate a season shooting percentage.

```
winlosslogs %>%
  group_by(Team) %>%
  summarise(
    wins = sum(winloss),
    totalFGAttempts = sum(TeamFGA),
    totalFG = sum(TeamFG)
  ) %>%
  mutate(fgpct = totalFG/totalFGAttempts) -> fgmodel
```

Now let's look at the scatterplot. With a scatterplot, we put what predicts the thing on the X axis, and the thing being predicted on the Y axis. In this case, X is our shooting percentage, y is our wins.

```
ggplot(fgmodel, aes(x=fgpct, y=wins)) + geom_point()
```

Let's talk about this. It seems that the data slopes up to the right. That would indicate a positive correlation between shooting percentage and wins. And that makes sense, no? You'd expect teams that shoot the ball well to win. But can we get a better sense of this? Yes, by adding another geom – `geom_smooth`.

```
ggplot(fgmodel, aes(x=fgpct, y=wins)) + geom_point() + geom_smooth(method=lm, se=TRUE)
```

But … how strong a relationship is this? How much can shooting percentage explain wins? Can we put some numbers to this?

Of course we can. We can apply a linear model to this – remember Chapter 9? We're going to create an object called fit, and then we're going to put into that object a linear model – `lm` – and the way to read this is "wins are predicted by field goal percentage". Then we just want the summary of that model.

```
fit <- lm(wins ~ fgpct, data = fgmodel)
summary(fit)
```

```
##
## Call:
## lm(formula = wins ~ fgpct, data = fgmodel)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.3536  -3.4523  -0.1125   3.3834  15.4318
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -49.217      4.845  -10.16   <2e-16 ***
## fgpct        149.416     10.915   13.69   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 5.035 on 351 degrees of freedom
## Multiple R-squared:  0.348,  Adjusted R-squared:  0.3462
## F-statistic: 187.4 on 1 and 351 DF,  p-value: < 2.2e-16
```

Remember from Chapter 9: There's just a few things you really need.

The first thing: R-squared. In this case, the Adjusted R-squared value is .3462, which we can interpret as shooting percentage predicts about 35 percent of the variance in wins. Which sounds not great, but in social science, that's huge. That's great. A psychology major would murder for that R-squared.

Second: The P-value. We want anything less than .05. If it's above .05, the change between them is not statistically significant – it's probably explained by random chance. In our case, we have 2.2e-16, which is to say 2.2 with 16 zeros in front of it, or .000000000000000022. Is that less than .05? Yes. Yes it is. So this is not random. Again, we would expect this, so it's a good logic test.

Third: The coefficient. In this case, the coefficient for fgpct is 149.416. Since I didn't convert percentages to decimals, what this says is that for every percentage point improvement in shooting percentage, we can expect the team to win 1.49 more games plus or minus some error.

And we can use this to predict a team's wins: remember your algebra and y = mx + b. In this case, y is the wins, m is the coefficient, x is the shooting percentage and b is the intercept.

So can plug these together: Expected wins = 149.416 * shooting percentage - 49.217

Let's use Nebraska as an example. They shot about .43 on the season (.4294421 to be exact).

y = 149.416 * .4294421 - 49.217 or 14.95 wins. How many wins did Nebraska have? 19.

What does that mean? It means that as disappointing a season as it was, Nebraska actually OVERPERFORMED it's season shooting percentage. They shouldn't have won as many games as they did, according to our model.

Chapter 21

Facet wraps

Sometimes the easiest way to spot a trend is to chart a bunch of small things side by side. Edward Tufte, one of the most well known data visualization thinkers on the planet, calls this "small multiples" where ggplot calls this a facet wrap or a facet grid, depending.

One thing we noticed earlier in the semester – it seems that a lot of teams shoot worse as the season goes on. Do they? We could answer this a number of ways, but the best way to show people would be visually. Let's use Small Multiples.

As always, we start with libraries.

```
library(tidyverse)
```

Now data.

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

Let's narrow our pile and look just at the Big Ten.

```
big10 <- logs %>% filter(Conference == "Big Ten")
```

The first thing we can do is look at a line chart, like we have done in previous chapters.

```
ggplot() + geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) + scale_y_contin
```

And, not surprisingly, we get a hairball. We could color certain lines, but that would limit us to focus on one team. What if we did all of them at once? We do that with a `facet_wrap`. The only thing we MUST pass into a `facet_wrap` is what thing we're going to separate them out by. In this case, we precede that field with a tilde, so in our case we want the Team field. It looks like this:

```
ggplot() + geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) + scale_y_contin
```

Answer: Not immediately clear, but we can look at this and analyze it. We could add a piece of annotation to help us out.

```
big10 %>% summarise(mean(TeamFGPCT))
```

```
## # A tibble: 1 x 1
##   `mean(TeamFGPCT)`
##               <dbl>
## 1             0.442
```

```
ggplot() + geom_hline(yintercept=.4447, color="blue") + geom_line(data=big10, aes(x=Date, y=TeamF
```

What do you see here? How do teams compare? How do they change over time? I'm not asking you these questions because they're an assignment – I'm asking because that's exactly what this chart helps answer. Your brain will immediately start making those connections.

21.1 Facet grid vs facet wraps

Facet grids allow us to put teams on the same plane, versus just repeating them. And we can specify that plane as vertical or horizontal. For example, here's our chart from above, but using facet_grid to stack them.

```
ggplot() + geom_hline(yintercept=.4447, color="blue") + geom_line(data=big10, aes(x=Da
```

Illinois Fighting Illini
Indiana Hoosiers
Iowa Hawkeyes
Maryland Terrapins
Michigan State Spartans
Michigan Wolverines
Minnesota Golden Gophers
Nebraska Cornhuskers
Northwestern Wildcats
Ohio State Buckeyes
Penn State Nittany Lions
Purdue Boilermakers
Rutgers Scarlet Knights
Wisconsin Badgers
TeamFGPCT
0.0
0.2
0.4
0.6
Nov
Dec
Jan
Feb
Mar
Date

And here they are next to each other:

```
ggplot() + geom_hline(yintercept=.4447, color="blue") + geom_line(data=big10, aes(x=Dat
```

Note: We'd have some work to do with the labeling on this – we'll get to that – but you can see where this is valuable comparing a group of things. One warning: Don't go too crazy with this or it loses it's visual power.

21.2 Other types

Line charts aren't the only things we can do. We can do any kind of chart in ggplot. Staying with shooting, where are team's winning and losing performances coming from when we talk about team shooting and opponent shooting?

```
ggplot() + geom_point(data=big10, aes(x=TeamFGPCT, y=OpponentFGPCT, color=W_L)) + scale
```

Illinois Fighting Illini
Indiana Hoosiers
Iowa Hawkeyes
Maryland Terrapins
ichigan State Spartan
Michigan Wolverines
nesota Golden Goph
lebraska Cornhusker
Northwestern Wildcat
Ohio State Buckeyes
enn State Nittany Lion
Purdue Boilermakers
utgers Scarlet Knight
Wisconsin Badgers
OpponentFGPCT
TeamFGPCT
0.0
0.2
0.4
0.6
W_L
L
L (1 OT)
L (2 OT)
W
W (1 OT)
W (2 OT)

Chapter 22

Tables

But not a table. A table with features.

Sometimes, the best way to show your data is with a table – simple rows and columns. It allows a reader to compare whatever they want to compare a little easier than a graph where you've chosen what to highlight. R has a neat package called `formattable` and you'll install it like anything else with `install.packages('formattable')`.

So what does it do? Let's gather our libraries and get some data.

```
library(tidyverse)
library(formattable)
```

```
offense <- read_csv("data/offensechange.csv")
```

```
## Parsed with column specification:
## cols(
##   Year = col_double(),
##   Name = col_character(),
##   G = col_double(),
##   `Rush Yards` = col_double(),
##   `Pass Yards` = col_double(),
##   Plays = col_double(),
##   `Total Yards` = col_double(),
##   `Yards/Play` = col_double(),
##   `Yards/G` = col_double()
## )
```

Let's ask this question: Which college football team saw the greatest improvement in yards per game last regular season? The simplest way to calculate that is by percent change.

```
changeTotalOffense <- offense %>%
  select(Name, Year, `Yards/G`) %>%
  spread(Year, `Yards/G`) %>%
  mutate(Change=(`2019`-`2018`)/`2018`) %>%
  arrange(desc(Change)) %>%
  top_n(20)
```

```
## Selecting by Change
```

We've output tables to the screen a thousand times in this class with `head`, but formattable makes them look good with very little code.

```
formattable(changeTotalOffense)
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

0.7023950

LSU

402.1

564.1

0.4028849

UTSA

247.1

344.9

0.3957912

San Jose State

323.7

427.4

0.3203584

Navy

349.3

455.8

0.3048955

Louisville

352.6

447.3

0.2685763

SMU

387.2

489.8

0.2649793

BYU

364.9

443.8

0.2162236

New Mexico

330.0

400.3

0.2130303

Charlotte

343.1

411.8

0.2002332

Iowa State

371.0

444.3

0.1975741

USC

382.6

454.0

0.1866179

Troy

389.4

456.3

0.1718028

Louisiana-Lafayette

424.3

494.1

0.1645062

Georgia State

378.2

439.8

0.1628768

Louisiana Tech

379.3

436.8

0.1515950

Minnesota

379.6

432.0

0.1380400

Ball State

408.6

463.0

0.1331375

Texas

411.6

465.8

0.1316812

Florida State

361.2

408.3

0.1303987

So there you have it. Central Michigan improved the most (but look at who came in second!). First thing I don't like about formattable tables – the right alignment. Let's fix that.

```
formattable(changeTotalOffense, align="l")
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

0.7023950

LSU

402.1

564.1

0.4028849

UTSA

247.1

344.9

0.3957912

San Jose State

323.7

427.4

0.3203584

Navy

349.3

455.8

0.3048955

Louisville

352.6

447.3

0.2685763

SMU

387.2

489.8

0.2649793

BYU

364.9

443.8

0.2162236

New Mexico

330.0

400.3

0.2130303

Charlotte

343.1

411.8

0.2002332

Iowa State

371.0

444.3

0.1975741

USC

382.6

454.0

0.1866179

Troy

389.4

456.3

0.1718028

Louisiana-Lafayette

424.3

494.1

0.1645062

Georgia State

378.2

439.8

0.1628768

Louisiana Tech

379.3

436.8

0.1515950

Minnesota

379.6

432.0

0.1380400

Ball State

408.6

463.0

0.1331375

Texas

411.6

465.8

0.1316812

Florida State

361.2

408.3

0.1303987

Next? I forgot to multiply by 100. No matter. Formattable can fix that for us.

```
formattable(
  changeTotalOffense,
  align="l",
  list(
```

```
    `Change` = percent)
  )
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

70.24%

LSU

402.1

564.1

40.29%

UTSA

247.1

344.9

39.58%

San Jose State

323.7

427.4

32.04%

Navy

349.3

455.8

30.49%

Louisville

352.6

447.3

26.86%

SMU

387.2

489.8

26.50%

BYU

364.9

443.8

21.62%

New Mexico

330.0

400.3

21.30%

Charlotte

343.1

411.8

20.02%

Iowa State

371.0

444.3

19.76%

USC

382.6

454.0

18.66%

Troy

389.4

456.3

17.18%

Louisiana-Lafayette

424.3

494.1

16.45%

Georgia State

378.2

439.8

16.29%

Louisiana Tech

379.3

436.8

15.16%

Minnesota

379.6

432.0

13.80%

Ball State

408.6

463.0

13.31%

Texas

411.6

465.8

13.17%

Florida State

361.2

408.3

13.04%

Something else not great? I can't really see the magnitude of the 2019 column. A team could improve a lot, but still not gain that many yards. Formattable has embeddable bar charts in the table. They look like this.

```
formattable(
  changeTotalOffense,
  align="l",
  list(
```

```
    `2019` = color_bar("#FA614B"),
    `Change` = percent)
  )
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

70.24%

LSU

402.1

564.1

40.29%

UTSA

247.1

344.9

39.58%

San Jose State

323.7

427.4

32.04%

Navy

349.3

455.8

30.49%

Louisville

352.6

447.3

26.86%

SMU

387.2

489.8

26.50%

BYU

364.9

443.8

21.62%

New Mexico

330.0

400.3

21.30%

Charlotte

343.1

411.8

20.02%

Iowa State

371.0

444.3

19.76%

USC

382.6

454.0

18.66%

Troy

389.4

456.3

17.18%

Louisiana-Lafayette

424.3

494.1

16.45%

Georgia State

378.2

439.8

16.29%

Louisiana Tech

379.3

436.8

15.16%

Minnesota

379.6

432.0

13.80%

Ball State

408.6

463.0

13.31%

Texas

411.6

465.8

13.17%

Florida State

361.2

408.3

13.04%

That gives me some more to mess with.

One thing you can do is set the bar widths to make them relative to each other, using a different function called a `normalize_bar`.

```
formattable(
  changeTotalOffense,
  align="r",
  list(
```

```
    `2019` = normalize_bar("#FA614B"),
    `2018` = normalize_bar("#FA614B"),
    `Change` = percent)
  )
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

70.24%

LSU

402.1

564.1

40.29%

UTSA

247.1

344.9

39.58%

San Jose State

323.7

427.4

32.04%

Navy

349.3

455.8

30.49%

Louisville

352.6

447.3

26.86%

SMU

387.2

489.8

26.50%

BYU

364.9

443.8

21.62%

New Mexico

330.0

400.3

21.30%

Charlotte

343.1

411.8

20.02%

Iowa State

371.0

444.3

19.76%

USC

382.6

454.0

18.66%

Troy

389.4

456.3

17.18%

Louisiana-Lafayette

424.3

494.1

16.45%

Georgia State

378.2

439.8

16.29%

Louisiana Tech

379.3

436.8

15.16%

Minnesota

379.6

432.0

13.80%

Ball State

408.6

463.0

13.31%

Texas

411.6

465.8

13.17%

Florida State

361.2

408.3

13.04%

Note: bookdown is formatting this weird. Your numbers won't look like this.

Another way to deal with this – color tiles. Change the rectangle that houses the data to a color indicating the intensity of it.

```r
formattable(
  changeTotalOffense,
  align="r",
  list(
     area(col = 2:3) ~ color_tile("#FFF6F4", "#FA614B"),
    `Change` = percent)
  )
```

Name

2018

2019

Change

Central Michigan

254.7

433.6

70.24%

LSU

402.1

564.1

40.29%

UTSA

247.1

344.9

39.58%

San Jose State

323.7

427.4

32.04%

Navy

349.3

455.8

30.49%

Louisville

352.6

447.3

26.86%

SMU

387.2

489.8

26.50%

BYU

364.9

443.8

21.62%

New Mexico

330.0

400.3

21.30%

Charlotte

343.1

411.8

20.02%

Iowa State

371.0

444.3

19.76%

USC

382.6

454.0

18.66%

Troy

389.4

456.3

17.18%

Louisiana-Lafayette

424.3

494.1

16.45%

Georgia State

378.2

439.8

16.29%

Louisiana Tech

379.3

436.8

15.16%

Minnesota

379.6

432.0

13.80%

Ball State

408.6

463.0

13.31%

Texas

411.6

465.8

13.17%

Florida State

361.2

408.3

13.04%

22.0.1 Exporting tables

The first thing you need to do is install some libraries – do this in the console, not in an R Studio code block because htmltools get's a little weird.

```
install.packages("htmltools")
install.packages("webshot")

webshot::install_phantomjs()
```

Now, copy, paste and run this code block entirely. Don't change anything.

```
library("htmltools")
library("webshot")

export_formattable <- function(f, file, width = "100%", height = NULL,
                               background = "white", delay = 0.2)
    {
      w <- as.htmlwidget(f, width = width, height = height)
      path <- html_print(w, background = background, viewer = NULL)
      url <- paste0("file:///", gsub("\\\\", "/", normalizePath(path)))
      webshot(url,
              file = file,
              selector = ".formattable_widget",
              delay = delay)
    }
```

Now, save your formattable table to an object using the `<-` assignment operator.

After you've done that, you can call the function you ran in the previous block to export as a png file. In my case, I created an object called table, which is populated with my formattable table. Then, in export_formattable, I pass in that `table` object and give it a name.

```
table <- formattable(
  changeTotalOffense,
  align="r",
  list(
     area(col = 2:3) ~ color_tile("#FFF6F4", "#FA614B"),
    `Change` = percent)
  )

export_formattable(table,"table.png")
```

Name	2018	2019	Change
Central Michigan	254.7	433.6	70.24%
LSU	402.1	564.1	40.29%
UTSA	247.1	344.9	39.58%
San Jose State	323.7	427.4	32.04%
Navy	349.3	455.8	30.49%
Louisville	352.6	447.3	26.86%
SMU	387.2	489.8	26.50%
BYU	364.9	443.8	21.62%
New Mexico	330.0	400.3	21.30%
Charlotte	343.1	411.8	20.02%
Iowa State	371.0	444.3	19.76%
USC	382.6	454.0	18.66%
Troy	389.4	456.3	17.18%
Louisiana-Lafayette	424.3	494.1	16.45%
Georgia State	378.2	439.8	16.29%
Louisiana Tech	379.3	436.8	15.16%
Minnesota	379.6	432.0	13.80%
Ball State	408.6	463.0	13.31%
Texas	411.6	465.8	13.17%
Florida State	361.2	408.3	13.04%

For now, pngs are what you need to export. There is a way to export PDFs, but they lose all the formatting when you do that, which is kind of pointless.

Chapter 23

Bubble charts

Here is the real talk: Bubble charts are hard. The reason they are hard is not because of the code, or the complexity or anything like that. They're a scatterplot with magnitude added – the size of the dot in the scatterplot has meaning. The hard part is seeing when a bubble chart works and when it doesn't.

If you want to see it work spectacularly well, watch a semi-famous Ted Talk by Hans Rosling from 2006 where bubble charts were the centerpiece. It's worth watching. It'll change your perspective on the world. No seriously. It will.

And since then, people have wanted bubble charts. And we're back to the original problem: They're hard. There's a finite set of circumstances where they work.

First, I'm going to show you an example of them not working to illustrate the point.

I'm going to load up my libraries: tidyverse per usual, rvest to get some data (we'll discuss rvest in greater detail in an upcoming chapter) and ggrepel because I end up using it every time I do a scatterplot.

```
library(tidyverse)
library(rvest)
library(ggrepel)
```

So for this example, I want to look at Nebraska's offense in the 2019 season. It … hasn't gone well. And typical of Nebraska teams for the last decade, they're turning the ball over a lot. So given the number of turnovers, how does Nebraska compare to other teams in the FBS?

I'm going to create a scatterplot yards per game on the X axis and points per game on the Y. They're pretty highly correlated with each other. And then

I'm going to make the dot the size of the turnovers – the bubble in my bubble charts.

Using Rvest, I'm going to grab total offense rankings, scoring offense rankings and turnover rankings and then merge them together with just the fields I need. This will all get explained more thoroughly coming up, but that's what this block of code does. When it's done, I'll have a dataframe called `offense` which I'll use to build my bubble chart.

```
yardsurl <- "http://cfbstats.com/2019/leader/national/team/offense/split01/category10/s

yards19 <- yardsurl %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

yards19 <- yards19[[1]] %>% select(Name, `Yards/G`)

pointsurl <- "http://cfbstats.com/2019/leader/national/team/offense/split01/category09/

points19 <- pointsurl %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

points19 <- points19[[1]] %>% select(Name, `Points/G`)

turnoversurl <- "http://cfbstats.com/2019/leader/national/team/offense/split01/category

turnovers19 <- turnoversurl %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

turnovers19 <- turnovers19[[1]] %>% select(Name, `Total Lost`)


offense <- yards19 %>%
  left_join(points19, by=c("Name")) %>%
  left_join(turnovers19, by=c("Name"))
```

A bubble chart is just a scatterplot with one additional element in the aesthetic – a size. Here's the scatterplot version.

```
ggplot() + geom_point(data=offense, aes(x=`Yards/G`, y=`Points/G`))
```

As expected, yards per game pretty tightly predicts points per game, but you could have guessed that without a chart. So let's add the size element.

```
ggplot() + geom_point(data=offense, aes(x=`Yards/G`, y=`Points/G`, size=`Total Lost`))
```

Eh. What does this chart tell you? Trick question, there's not much new here. The dots are too big. Also, we can't see when they overlap. We can fix that by adding an alpha element outside the aesthetic – alpha in this case is transparency – and we can manually change the size of the dots by adding `scale_size` and a `range`.

```
ggplot() + geom_point(data=offense, aes(x=`Yards/G`, y=`Points/G`, size=`Total Lost`),
```

Before we do any more work, let's return to the earlier question: What story does this tell? Can you discern a story from the bubbles? Are teams with lots of turnovers doing poorly and teams with few turnovers doing well? The problem is, you can't really tell. So this is a dead end for a bubble chart. If you get a big mess, it's a dead giveaway that you probably don't have a bubble chart.

So let's look at something else. Let's look at something that isn't directly correlated – we'll look at offensive points per game vs defensive points per game.

I'm going to edit the same rvest code to grab those points per game stats and merge it all together. When it's done, I'll have a dataframe called `football` and we can look at where good teams fall on the chart with turnover margin as a scaled dot.

```
ourl <- "http://cfbstats.com/2019/leader/national/team/offense/split01/category09/sort01.html"

o19 <- ourl %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

o19 <- o19[[1]] %>% select(Name, `Points/G`)

durl <- "http://cfbstats.com/2019/leader/national/team/defense/split01/category09/sort01.html"

d19 <- durl %>%
```

```
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

d19 <- d19[[1]] %>% select(Name, `Points/G`)

turnoversurl <- "http://cfbstats.com/2019/leader/national/team/offense/split01/category

turnovers19 <- turnoversurl %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

turnovers19 <- turnovers19[[1]] %>% select(Name, Margin)

football <- o19 %>%
  left_join(d19, by=c("Name")) %>%
  left_join(turnovers19, by=c("Name")) %>%
  rename(`Offensive Points Per Game` = `Points/G.x`, `Defensive Points Per Game`=`Poin
```

Now we can do the bubble chart.

```
ggplot() + geom_point(data=football, aes(x=`Offensive Points Per Game`, y=`Defensive Po
```

Better! Teams are spread out a little more. Bottom right quadrant – the good defense, good offense quadrant – have some large dots. The upper left quadrant

– bad defense, bad offense – have some very small dots, meaning they have a really terrible turnover margin.

But what would make this chart better – and what you saw in the Rosling video – is color. What if we colored the dots by if they were above or below zero? Meaning, do they have a positive or negative turnover margin? We can do that with a quick mutate and a case_when statement.

```
football <- football %>% mutate(PositiveNegative = case_when(
  Margin > 0 ~ "Positive",
  Margin < 0 ~ "Negative",
  Margin == 0 ~ "Even"
))
```

Now we can add `color=PositiveNegative` to the aesthetic and our dots will be colored by if they are positive, negative or zero.

```
ggplot() + geom_point(data=football, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per G
```

Now we're getting somewhere. What's the story that this chart tells? Blue dots – positive turnover margins – are all drifting toward that good offense, good defense quadrant. Green dots – negative turnover margins – are drifting toward that bad defense, bad offense quadrant.

Let's add some annotations. Let's look at the top two turnover margin teams and where they come out.

```
topteams <- football %>% filter(Name == "LSU" | Name == "Alabama")
```

```
ggplot() +
  geom_point(data=football, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per
  geom_text_repel(data=topteams, aes(x=`Offensive Points Per Game`, y=`Defensive Points
  scale_size(range = c(0, 6), name="Turnovers")
```

No surprise there. What about Nebraska?

```
nu <- football %>% filter(Name=="Nebraska")
```

```
ggplot() +
  geom_point(data=football, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per
  geom_text_repel(data=topteams, aes(x=`Offensive Points Per Game`, y=`Defensive Points
  geom_point(data=nu, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per Game`
  geom_text_repel(data=nu, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per G
  scale_size(range = c(0, 6), name="Turnovers")
```

Sadly, no surprise there either.

The last things we need to do? Add some labels, apply our finishing touches.

```
ggplot() +
  geom_point(data=football, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per Game`, siz
  geom_text_repel(data=topteams, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per Game`
  geom_point(data=nu, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per Game`, size=Marg
  geom_text_repel(data=nu, aes(x=`Offensive Points Per Game`, y=`Defensive Points Per Game`, labe
  scale_size(range = c(0, 6), name="Turnovers") +
  labs(title="Same song, different year", subtitle="The Husker's turnover margin is zero, putting
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 8),
    plot.subtitle = element_text(size=10),
    panel.grid.minor = element_blank()
    )
```

Same song, different year
The Husker's turnover margin is zero, putting them miles from the top.
Defensive Points Per Game
50
40
30
20
10
20
30
40
50
Offensive Points Per Game
Nebraska
LSU
Alabama
PositiveNegative
Even
Negative
Positive
Turnovers
−10
0
10
20
Source: NCAA | By Matt Waite

Chapter 24

Circular bar plots

At the 27:36 mark in the Half Court Podcast, Omaha World Herald Writer Chris Heady said "November basketball doesn't matter, but it shows you where you are."

It's a tempting phrase to believe, especially a day after Nebraska lost the first game of the Fred Hoiberg era at home to a baseball school, UC Riverside. And it wasn't close. The Huskers, because of a total roster turnover, were a complete mystery before the game. And what happened during it wasn't pretty, so there was a little soul searching going on in Lincoln.

But does November basketball really not matter?

Let's look, using a new form of chart called a circular bar plot. It's a chart type that combines several forms we've used before: bar charts to show magnitude, stacked bar charts to show proportion, but we're going to add bending the chart around a circle to add some visual interstingness to it. We're also going to use time as an x-axis value to make a not subtle circle of time reference – a common technique with circular bar charts.

First we need some libraries.

```
library(tidyverse)
library(lubridate)
```

Let's import every basketball game from last year.

```
logs <- read_csv("data/logs19.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
```

```
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

So let's test the notion of November Basketball Doesn't Matter. What matters in basketball? Let's start simple: Wins.

Sports Reference's win columns are weird, so we need to scan through them and find W and L and we'll give them numbers using `case_when`. I'm also going to filter out tournament basketball.

```
winlosslogs <- logs %>% mutate(winloss = case_when(
  grepl("W", W_L) ~ 1,
  grepl("L", W_L) ~ 0)
) %>% filter(Date < "2019-03-19")
```

Now we can group by date and conference and sum up the wins. How many wins by day does each conference get?

```
dates <- winlosslogs %>% group_by(Date, Conference) %>% summarise(wins = sum(winloss))
```

Earlier, we did stacked bar charts. We have what we need to do that now.

```
ggplot() + geom_bar(data=dates, aes(x=Date, weight=wins, fill=Conference)) + theme_mini
```

Eeek. This is already looking not great. But to make it a circular bar chart, we add `coord_polar()` to our chart.

```
ggplot() + geom_bar(data=dates, aes(x=Date, weight=wins, fill=Conference)) + theme_minimal() + co
```

Based on that, the day is probably too thin a slice, and there's way too many

conferences in college basketball. Let's group this by months and filter out all but the power five conferences.

```
p5 <- c("SEC", "Big Ten", "Pac-12", "Big 12", "ACC")
```

To get months, we're going to use a function in the library `lubridate` called `floor_date`, which combine with mutate will give us a field of just months.

```
wins <- winlosslogs %>% mutate(month = floor_date(Date, unit="months")) %>% group_by(mo
```

Now we can use wins to make our circular bar chart of wins by month in the Power Five.

```
ggplot() + geom_bar(data=wins, aes(x=month, weight=wins, fill=Conference)) + theme_mini
```

Yikes. That looks a lot like a broken pie chart. So months are too thick of a slice. Let's use weeks in our floor date to see what that gives us.

```
wins <- winlosslogs %>% mutate(week = floor_date(Date, unit="weeks")) %>% group_by(weel

ggplot() + geom_bar(data=wins, aes(x=week, weight=wins, fill=Conference)) + theme_minir
```

That looks better. But what does it say? Does November basketball matter? What this is saying is … yeah, it kinda does. The reason? Lots of wins get piled up in November and December, during non-conference play. So if you are a team with NCAA tournament dreams, you need to win games in November to make sure your tournament resume is where it needs to be come March. Does an individual win or loss matter? Probably not. But your record in November does.

Chapter 25

Intro to rvest

All the way back in Chapter 2, we used Google Sheets and importHTML to get our own data out of a website. For me, that's a lot of pointing and clicking and copying and pasting. R has a library that can automate the harvesting of data from HTML on the internet. It's called `rvest`.

Let's grab a simple, basic HTML table from College Football Stats. This is scoring offense for 2019. There's nothing particularly strange about this table – it's simply formatted and easy to scrape.

First we'll need some libraries. We're going to use a library called `rvest`, which you can get by running `install.packages('rvest')` in the console.

```
library(rvest)
library(tidyverse)
```

The rvest package has functions that make fetching, reading and parsing HTML simple. The first thing we need to do is specify a url that we're going to scrape.

```
scoringoffenseurl <- "http://www.cfbstats.com/2019/leader/national/team/offense/split01/category0
```

Now, the most difficult part of scraping data from any website is knowing what exact HTML tag you need to grab. In this case, we want a `<table>` tag that has all of our data table in it. But how do you tell R which one that is? Well, it's easy, once you know what to do. But it's not simple. So I've made a short video to show you how to find it.

When you have simple tables, the code is very simple. You create a variable to receive the data, then pass it the url, read the html that was fetched, find the node you need using your XPath value you just copied and you tell rvest that it's a table.

```
scoringoffense <- scoringoffenseurl %>%
  read_html() %>%
```

```
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()
```

What we get from this is … not a dataframe. It's a list with one element in it, which just so happens to be our dataframe. When you get this, the solution is simple: just overwrite the variable you created with the first list element.

```
scoringoffense <- scoringoffense[[1]]
```

And what do we have?

```
head(scoringoffense)
```

```
##           Name  G TD FG 1XP 2XP Safety Points Points/G
## 1 1        LSU 15 95 21  89   1      1    726     48.4
## 2 2    Alabama 13 83 12  80   0      0    614     47.2
## 3 3 Ohio State 14 88 13  87   0      1    656     46.9
## 4 4    Clemson 15 88 14  85   2      0    659     43.9
## 5 5        UCF 13 74 15  71   1      1    564     43.4
## 6 6   Oklahoma 14 76 19  75   1      0    590     42.1
```

We have data, ready for analysis.

25.1 A slightly more complicated example

What if we want more than one year in our dataframe?

This is a common problem. What if we want to look at every scoring offense going back several years? The website has them going back to 2009. How can we combine them?

First, we should note, that the data does not have anything in it to indicate what year it comes from. So we're going to have to add that. And we're going to have to figure out a way to stack two dataframes on top of each other.

So let's grab 2018.

```
scoringoffenseurl18 <- "http://www.cfbstats.com/2018/leader/national/team/offense/split

scoringoffense18 <- scoringoffenseurl18 %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="content"]/div[2]/table') %>%
  html_table()

scoringoffense18 <- scoringoffense18[[1]]
```

First, how are we going to know, in the data, which year our data is from? We can use mutate.

```
scoringoffense19 <- scoringoffense %>% mutate(YEAR = 2019)
```

```
## Error: Column 1 must be named.
## Use .name_repair to specify repair.
```

Uh oh. Error. What does it say? Column 1 must be named. If you look at our data in the environment tab in the upper right corner, you'll see that indeed, the first column has no name. It's the FBS rank of each team. So we can fix that and mutate in the same step. We'll do that using `rename` and since the field doesn't have a name to rename it, we'll use a position argument. We'll say rename column 1 as Rank.

```
scoringoffense19 <- scoringoffense %>% rename(Rank = 1) %>% mutate(YEAR = 2019)
scoringoffense18 <- scoringoffense18 %>% rename(Rank = 1) %>% mutate(YEAR = 2018)
```

And now, to combine the two tables together length-wise – we need to make long data – we'll use a base R function called `rbind`. The good thing is rbind is simple. The bad part is it can only do two tables at a time, so if you have more than that, you'll need to do it in steps.

```
combined <- rbind(scoringoffense19, scoringoffense18)
```

Note in the environment tab we now have a data frame called combined that has 260 observations – which just so happens to be what 130 from 2019 and 130 from 2018 add up to.

```
head(combined)
```

```
##   Rank       Name  G TD FG 1XP 2XP Safety Points Points/G YEAR
## 1    1        LSU 15 95 21  89   1      1    726     48.4 2019
## 2    2    Alabama 13 83 12  80   0      0    614     47.2 2019
## 3    3 Ohio State 14 88 13  87   0      1    656     46.9 2019
## 4    4    Clemson 15 88 14  85   2      0    659     43.9 2019
## 5    5        UCF 13 74 15  71   1      1    564     43.4 2019
## 6    6   Oklahoma 14 76 19  75   1      0    590     42.1 2019
```

25.2 An even more complicated example

What do you do when the table has non-standard headers?

Unfortunately, non-standard means there's no one way to do it – it's going to depend on the table and the headers. But here's one idea: Don't try to make it work.

I'll explain.

Let's try to get season team stats from Sports Reference. If you look at that page, you'll see the problem right away – the headers span two rows, and they repeat. That's going to be all kinds of no good. You can't import that. Dataframes must

have names all in one row. If you have two-line headers, you have a problem you have to fix before you can do anything else with it.

First we'll grab the page.

```
url <- "https://www.sports-reference.com/cbb/seasons/2019-school-stats.html"
```

Now, similar to our example above, we'll read the html, use XPath to find the table, and then read that table with a directive passed to it setting the header to FALSE. That tells rvest that there isn't a header row. Just import it as data.

```
stats <- url %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="basic_school_stats"]') %>%
  html_table(header=FALSE)
```

What we get back is a list of one element (similar to above). So let's pop it out into a data frame.

```
stats <- stats[[1]]
```

And we'll take a look at what we have.

```
head(stats)
```

```
##   X1                      X2      X3      X4      X5      X6      X7      X8
## 1                        Overall Overall Overall Overall Overall Overall
## 2 Rk                  School       G       W       L    W-L%     SRS     SOS
## 3  1 Abilene Christian NCAA      34      27       7    .794   -1.91   -7.34
## 4  2              Air Force      32      14      18    .438   -4.28    0.24
## 5  3                  Akron      33      17      16    .515    4.86    1.09
## 6  4            Alabama A&M      32       5      27    .156  -19.23   -8.38
##      X9   X10  X11  X12  X13  X14    X15    X16 X17           X18           X19
## 1 Conf. Conf. Home Home Away Away Points Points  NA School Totals School Totals
## 2     W     L    W    L    W    L    Tm.   Opp.  NA            MP            FG
## 3    14     4   13    2   10    4   2502   2161  NA          1370           897
## 4     8    10    9    6    3    9   2179   2294  NA          1300           802
## 5     8    10   14    3    1   10   2271   2107  NA          1325           797
## 6     4    14    4    7    0   18   1938   2285  NA          1295           736
##             X20           X21           X22           X23           X24
## 1 School Totals School Totals School Totals School Totals School Totals
## 2           FGA           FG%            3P           3PA           3P%
## 3          1911          .469           251           660          .380
## 4          1776          .452           234           711          .329
## 5          1948          .409           297           929          .320
## 6          1809          .407           182           578          .315
##             X25           X26           X27           X28           X29
## 1 School Totals School Totals School Totals School Totals School Totals
## 2            FT           FTA           FT%           ORB           TRB
```

```
## 3           457           642          .712           325          1110
## 4           341           503          .678           253          1077
## 5           380           539          .705           312          1204
## 6           284           453          .627           314          1032
##             X30           X31           X32           X33           X34
## 1 School Totals School Totals School Totals School Totals School Totals
## 2           AST           STL           BLK           TOV            PF
## 3           525           297            93           407           635
## 4           434           154            57           423           543
## 5           399           185           106           388           569
## 6           385           234            50           487           587
```

So, that's not ideal. We have headers and data mixed together, and our columns are named X1 to X34. Also note: They're all character fields. Because the headers are interspersed with data, it all gets called character data. So we've got to first rename each field.

```
stats <- stats %>% rename(Rank=X1, School=X2, Games=X3, OverallWins=X4, OverallLosses=X5, WinPct=
```

Now we have to get rid of those headers interspersed in the data. We can do that with filter that say keep all the stuff that isn't this.

```
stats <- stats %>% filter(Rank != "Rk" & Games != "Overall")
```

And finally, we need to change the file type of all the fields that need it. We're going to use a clever little trick, which goes like this: We're going to use `mutate_at`, which means mutate these fields. The pattern for `mutate_at` is `mutate_at` these variables and do this thing to them. But instead of specifying which of 33 variables we're going to mutate, we're going to specify the one we don't want to change, which is the name of the school. And we just want to convert them to numeric. Here's what it looks like:

```
stats %>% mutate_at(vars(-School), as.numeric)
```

```
##     Rank                      School Games OverallWins OverallLosses WinPct
## 1      1      Abilene Christian NCAA    34          27             7  0.794
## 2      2                   Air Force    32          14            18  0.438
## 3      3                       Akron    33          17            16  0.515
## 4      4                 Alabama A&M    32           5            27  0.156
## 5      5          Alabama-Birmingham    35          20            15  0.571
## 6      6               Alabama State    31          12            19  0.387
## 7      7                     Alabama    34          18            16  0.529
## 8      8                 Albany (NY)    32          12            20  0.375
## 9      9                Alcorn State    31          10            21  0.323
## 10    10                    American    30          15            15  0.500
## 11    11           Appalachian State    32          11            21  0.344
## 12    12          Arizona State NCAA    34          23            11  0.676
## 13    13                     Arizona    32          17            15  0.531
```

```
## 14    14                Little Rock    31          10            21  0.323
## 15    15        Arkansas-Pine Bluff    32          13            19  0.406
## 16    16             Arkansas State    32          13            19  0.406
## 17    17                   Arkansas    34          18            16  0.529
## 18    18                       Army    32          13            19  0.406
## 19    19                Auburn NCAA    40          30            10  0.750
## 20    20                Austin Peay    33          22            11  0.667
## 21    21                 Ball State    33          16            17  0.485
## 22    22                Baylor NCAA    34          20            14  0.588
## 23    23               Belmont NCAA    33          27             6  0.818
## 24    24            Bethune-Cookman    31          14            17  0.452
## 25    25                 Binghamton    33          10            23  0.303
## 26    26                Boise State    33          13            20  0.394
## 27    27             Boston College    31          14            17  0.452
## 28    28          Boston University    33          15            18  0.455
## 29    29        Bowling Green State    34          22            12  0.647
## 30    30               Bradley NCAA    35          20            15  0.571
## 31    31              Brigham Young    32          19            13  0.594
## 32    32                      Brown    32          20            12  0.625
## 33    33                     Bryant    30          10            20  0.333
## 34    34                   Bucknell    33          21            12  0.636
## 35    35               Buffalo NCAA    36          32             4  0.889
## 36    36                     Butler    33          16            17  0.485
## 37    37                   Cal Poly    29           6            23  0.207
## 38    38      Cal State Bakersfield    34          18            16  0.529
## 39    39        Cal State Fullerton    34          16            18  0.471
## 40    40       Cal State Northridge    34          13            21  0.382
## 41    41         California Baptist    31          16            15  0.516
## 42    42                   UC-Davis    31          11            20  0.355
## 43    43             UC-Irvine NCAA    37          31             6  0.838
## 44    44               UC-Riverside    33          10            23  0.303
## 45    45           UC-Santa Barbara    32          22            10  0.688
## 46    46   University of California    31           8            23  0.258
## 47    47                   Campbell    33          20            13  0.606
## 48    48                   Canisius    32          15            17  0.469
## 49    49           Central Arkansas    33          14            19  0.424
## 50    50  Central Connecticut State    31          11            20  0.355
## 51    51       Central Florida NCAA    33          24             9  0.727
## 52    52           Central Michigan    35          23            12  0.657
## 53    53        Charleston Southern    34          18            16  0.529
## 54    54                  Charlotte    29           8            21  0.276
## 55    55                Chattanooga    32          12            20  0.375
## 56    56              Chicago State    32           3            29  0.094
## 57    57            Cincinnati NCAA    35          28             7  0.800
## 58    58                    Citadel    30          12            18  0.400
## 59    59                    Clemson    34          20            14  0.588
```

```
## 60    60            Cleveland State    31          10           21  0.323
## 61    61           Coastal Carolina    34          17           17  0.500
## 62    62               Colgate NCAA    35          24           11  0.686
## 63    63       College of Charleston   33          24            9  0.727
## 64    64             Colorado State    32          12           20  0.375
## 65    65                   Colorado    36          23           13  0.639
## 66    66                   Columbia    28          10           18  0.357
## 67    67                Connecticut    33          16           17  0.485
## 68    68               Coppin State    33           8           25  0.242
## 69    69                     Cornell   31          15           16  0.484
## 70    70                  Creighton    35          20           15  0.571
## 71    71                  Dartmouth    30          11           19  0.367
## 72    72                   Davidson    34          24           10  0.706
## 73    73                     Dayton    33          21           12  0.636
## 74    74             Delaware State    31           6           25  0.194
## 75    75                   Delaware    33          17           16  0.515
## 76    76                     Denver    30           8           22  0.267
## 77    77                     DePaul    36          19           17  0.528
## 78    78              Detroit Mercy    31          11           20  0.355
## 79    79                      Drake    34          24           10  0.706
## 80    80                     Drexel    32          13           19  0.406
## 81    81                  Duke NCAA    38          32            6  0.842
## 82    82                   Duquesne    32          19           13  0.594
## 83    83              East Carolina    31          10           21  0.323
## 84    84        East Tennessee State   34          24           10  0.706
## 85    85           Eastern Illinois    32          14           18  0.438
## 86    86           Eastern Kentucky    31          13           18  0.419
## 87    87           Eastern Michigan    32          15           17  0.469
## 88    88         Eastern Washington    34          16           18  0.471
## 89    89                       Elon    32          11           21  0.344
## 90    90                 Evansville    32          11           21  0.344
## 91    91                  Fairfield    31           9           22  0.290
## 92    92    Fairleigh Dickinson NCAA   35          21           14  0.600
## 93    93                Florida A&M    31          12           19  0.387
## 94    94           Florida Atlantic    33          17           16  0.515
## 95    95         Florida Gulf Coast    32          14           18  0.438
## 96    96       Florida International   34          20           14  0.588
## 97    97         Florida State NCAA    37          29            8  0.784
## 98    98               Florida NCAA    36          20           16  0.556
## 99    99                    Fordham    32          12           20  0.375
## 100  100               Fresno State    32          23            9  0.719
## 101  101                     Furman    33          25            8  0.758
## 102  102          Gardner-Webb NCAA    35          23           12  0.657
## 103  103              George Mason    33          18           15  0.545
## 104  104          George Washington   33           9           24  0.273
## 105  105                 Georgetown   33          19           14  0.576
```

```
## 106  106          Georgia Southern   33        21          12  0.636
## 107  107        Georgia State NCAA   34        24          10  0.706
## 108  108              Georgia Tech   32        14          18  0.438
## 109  109                   Georgia   32        11          21  0.344
## 110  110              Gonzaga NCAA   37        33           4  0.892
## 111  111                 Grambling   34        17          17  0.500
## 112  112              Grand Canyon   34        20          14  0.588
## 113  113                 Green Bay   38        21          17  0.553
## 114  114                   Hampton   35        18          17  0.514
## 115  115                  Hartford   33        18          15  0.545
## 116  116                   Harvard   31        19          12  0.613
## 117  117                    Hawaii   31        18          13  0.581
## 118  118                 High Point   31        16          15  0.516
## 119  119                   Hofstra   35        27           8  0.771
## 120  120                 Holy Cross   33        16          17  0.485
## 121  121           Houston Baptist   30        12          18  0.400
## 122  122              Houston NCAA   37        33           4  0.892
## 123  123                    Howard   34        17          17  0.500
## 124  124               Idaho State   30        11          19  0.367
## 125  125                     Idaho   32         5          27  0.156
## 126  126          Illinois-Chicago   32        16          16  0.500
## 127  127            Illinois State   33        17          16  0.515
## 128  128                  Illinois   33        12          21  0.364
## 129  129           Incarnate Word    31         6          25  0.194
## 130  130            Indiana State   31        15          16  0.484
## 131  131                  Indiana   35        19          16  0.543
## 132  132                 Iona NCAA   33        17          16  0.515
## 133  133           Iowa State NCAA   35        23          12  0.657
## 134  134                 Iowa NCAA   35        23          12  0.657
## 135  135         Purdue-Fort Wayne   33        18          15  0.545
## 136  136                     IUPUI   33        16          17  0.485
## 137  137             Jackson State   32        13          19  0.406
## 138  138        Jacksonville State   33        24           9  0.727
## 139  139              Jacksonville   32        12          20  0.375
## 140  140            James Madison   33        14          19  0.424
## 141  141         Kansas State NCAA   34        25           9  0.735
## 142  142               Kansas NCAA   36        26          10  0.722
## 143  143           Kennesaw State    32         6          26  0.188
## 144  144               Kent State   33        22          11  0.667
## 145  145             Kentucky NCAA   37        30           7  0.811
## 146  146                 La Salle   31        10          21  0.323
## 147  147                Lafayette   30        10          20  0.333
## 148  148                    Lamar   33        20          13  0.606
## 149  149                   Lehigh   31        20          11  0.645
## 150  150              Liberty NCAA   36        29           7  0.806
## 151  151                 Lipscomb   37        29           8  0.784
```

```
## 152  152        Cal State Long Beach    34          15           19  0.441
## 153  153      Long Island University    32          16           16  0.500
## 154  154                    Longwood    34          16           18  0.471
## 155  155                   Louisiana    32          19           13  0.594
## 156  156            Louisiana-Monroe    35          19           16  0.543
## 157  157        Louisiana State NCAA    35          28            7  0.800
## 158  158              Louisiana Tech    33          20           13  0.606
## 159  159             Louisville NCAA    34          20           14  0.588
## 160  160                Loyola (IL)    34          20           14  0.588
## 161  161            Loyola Marymount    34          22           12  0.647
## 162  162                Loyola (MD)    32          11           21  0.344
## 163  163                       Maine    32           5           27  0.156
## 164  164                   Manhattan    32          11           21  0.344
## 165  165                      Marist    31          12           19  0.387
## 166  166              Marquette NCAA    34          24           10  0.706
## 167  167                    Marshall    37          23           14  0.622
## 168  168   Maryland-Baltimore County    34          21           13  0.618
## 169  169      Maryland-Eastern Shore    32           7           25  0.219
## 170  170               Maryland NCAA    34          23           11  0.676
## 171  171        Massachusetts-Lowell    32          15           17  0.469
## 172  172               Massachusetts    32          11           21  0.344
## 173  173               McNeese State    31           9           22  0.290
## 174  174                     Memphis    36          22           14  0.611
## 175  175                      Mercer    31          11           20  0.355
## 176  176                  Miami (FL)    32          14           18  0.438
## 177  177                  Miami (OH)    32          15           17  0.469
## 178  178         Michigan State NCAA    39          32            7  0.821
## 179  179               Michigan NCAA    37          30            7  0.811
## 180  180            Middle Tennessee    32          11           21  0.344
## 181  181                   Milwaukee    31           9           22  0.290
## 182  182              Minnesota NCAA    36          22           14  0.611
## 183  183      Mississippi State NCAA    34          23           11  0.676
## 184  184    Mississippi Valley State    32           6           26  0.188
## 185  185            Mississippi NCAA    33          20           13  0.606
## 186  186        Missouri-Kansas City    32          11           21  0.344
## 187  187              Missouri State    32          16           16  0.500
## 188  188                    Missouri    32          15           17  0.469
## 189  189                    Monmouth    35          14           21  0.400
## 190  190               Montana State    32          15           17  0.469
## 191  191                Montana NCAA    35          26            9  0.743
## 192  192              Morehead State    33          13           20  0.394
## 193  193                Morgan State    30           9           21  0.300
## 194  194            Mount St. Mary's    31           9           22  0.290
## 195  195           Murray State NCAA    33          28            5  0.848
## 196  196                        Navy    31          12           19  0.387
## 197  197                       Omaha    32          21           11  0.656
```

```
## 198  198                    Nebraska   36          19            17  0.528
## 199  199            Nevada-Las Vegas   31          17            14  0.548
## 200  200                 Nevada NCAA   34          29             5  0.853
## 201  201               New Hampshire   29           5            24  0.172
## 202  202       New Mexico State NCAA   35          30             5  0.857
## 203  203                  New Mexico   32          14            18  0.438
## 204  204                 New Orleans   33          19            14  0.576
## 205  205                     Niagara   32          13            19  0.406
## 206  206              Nicholls State   31          14            17  0.452
## 207  207                        NJIT   35          22            13  0.629
## 208  208               Norfolk State   36          22            14  0.611
## 209  209               North Alabama   32          10            22  0.313
## 210  210    North Carolina-Asheville   31           4            27  0.129
## 211  211          North Carolina A&T   32          19            13  0.594
## 212  212 North Carolina Central NCAA   34          18            16  0.529
## 213  213   North Carolina-Greensboro   36          29             7  0.806
## 214  214        North Carolina State   36          24            12  0.667
## 215  215   North Carolina-Wilmington   33          10            23  0.303
## 216  216         North Carolina NCAA   36          29             7  0.806
## 217  217     North Dakota State NCAA   35          19            16  0.543
## 218  218                North Dakota   30          12            18  0.400
## 219  219               North Florida   33          16            17  0.485
## 220  220                 North Texas   33          21            12  0.636
## 221  221           Northeastern NCAA   34          23            11  0.676
## 222  222            Northern Arizona   31          10            21  0.323
## 223  223           Northern Colorado   32          21            11  0.656
## 224  224           Northern Illinois   34          17            17  0.500
## 225  225               Northern Iowa   34          16            18  0.471
## 226  226      Northern Kentucky NCAA   35          26             9  0.743
## 227  227          Northwestern State   31          11            20  0.355
## 228  228                Northwestern   32          13            19  0.406
## 229  229                  Notre Dame   33          14            19  0.424
## 230  230                     Oakland   33          16            17  0.485
## 231  231             Ohio State NCAA   35          20            15  0.571
## 232  232                        Ohio   31          14            17  0.452
## 233  233              Oklahoma State   32          12            20  0.375
## 234  234               Oklahoma NCAA   34          20            14  0.588
## 235  235           Old Dominion NCAA   35          26             9  0.743
## 236  236                Oral Roberts   32          11            21  0.344
## 237  237                Oregon State   31          18            13  0.581
## 238  238                 Oregon NCAA   38          25            13  0.658
## 239  239                     Pacific   32          14            18  0.438
## 240  240                  Penn State   32          14            18  0.438
## 241  241                Pennsylvania   31          19            12  0.613
## 242  242                  Pepperdine   34          16            18  0.471
## 243  243                  Pittsburgh   33          14            19  0.424
```

```
## 244  244              Portland State    32          16            16  0.500
## 245  245                    Portland    32           7            25  0.219
## 246  246           Prairie View NCAA    35          22            13  0.629
## 247  247                Presbyterian    36          20            16  0.556
## 248  248                   Princeton    28          16            12  0.571
## 249  249                  Providence    34          18            16  0.529
## 250  250                 Purdue NCAA    36          26            10  0.722
## 251  251                  Quinnipiac    31          16            15  0.516
## 252  252                     Radford    33          22            11  0.667
## 253  253               Rhode Island    33          18            15  0.545
## 254  254                       Rice    32          13            19  0.406
## 255  255                   Richmond    33          13            20  0.394
## 256  256                      Rider    31          16            15  0.516
## 257  257               Robert Morris    35          18            17  0.514
## 258  258                    Rutgers    31          14            17  0.452
## 259  259            Sacramento State    31          15            16  0.484
## 260  260                Sacred Heart    32          15            17  0.469
## 261  261          Saint Francis (PA)    33          18            15  0.545
## 262  262              Saint Joseph's    33          14            19  0.424
## 263  263            Saint Louis NCAA    36          23            13  0.639
## 264  264      Saint Mary's (CA) NCAA    34          22            12  0.647
## 265  265               Saint Peter's    32          10            22  0.313
## 266  266           Sam Houston State    33          21            12  0.636
## 267  267                    Samford    33          17            16  0.515
## 268  268             San Diego State    34          21            13  0.618
## 269  269                   San Diego    36          21            15  0.583
## 270  270               San Francisco    31          21            10  0.677
## 271  271              San Jose State    31           4            27  0.129
## 272  272                  Santa Clara    31          16            15  0.516
## 273  273              Savannah State    31          11            20  0.355
## 274  274                     Seattle    33          18            15  0.545
## 275  275             Seton Hall NCAA    34          20            14  0.588
## 276  276                       Siena    33          17            16  0.515
## 277  277               South Alabama    34          17            17  0.500
## 278  278        South Carolina State    34           8            26  0.235
## 279  279      South Carolina Upstate    32           6            26  0.188
## 280  280              South Carolina    32          16            16  0.500
## 281  281          South Dakota State    33          24             9  0.727
## 282  282                South Dakota    30          13            17  0.433
## 283  283               South Florida    38          24            14  0.632
## 284  284    Southeast Missouri State    31          10            21  0.323
## 285  285      Southeastern Louisiana    33          17            16  0.515
## 286  286         Southern California    33          16            17  0.485
## 287  287            SIU Edwardsville    31          10            21  0.323
## 288  288           Southern Illinois    32          17            15  0.531
## 289  289          Southern Methodist    32          15            17  0.469
```

```
## 290  290        Southern Mississippi   33        20          13  0.606
## 291  291               Southern Utah   34        17          17  0.500
## 292  292                    Southern   32         7          25  0.219
## 293  293             St. Bonaventure   34        18          16  0.529
## 294  294            St. Francis (NY)   33        17          16  0.515
## 295  295        St. John's (NY) NCAA   34        21          13  0.618
## 296  296                    Stanford   31        15          16  0.484
## 297  297           Stephen F. Austin   30        14          16  0.467
## 298  298                     Stetson   31         7          24  0.226
## 299  299                 Stony Brook   33        24           9  0.727
## 300  300               Syracuse NCAA   34        20          14  0.588
## 301  301                 Temple NCAA   33        23          10  0.697
## 302  302            Tennessee-Martin   31        12          19  0.387
## 303  303             Tennessee State   30         9          21  0.300
## 304  304              Tennessee Tech   31         8          23  0.258
## 305  305              Tennessee NCAA   37        31           6  0.838
## 306  306    Texas A&M-Corpus Christi   32        14          18  0.438
## 307  307                   Texas A&M   32        14          18  0.438
## 308  308             Texas-Arlington   33        17          16  0.515
## 309  309             Texas Christian   37        23          14  0.622
## 310  310               Texas-El Paso   29         8          21  0.276
## 311  311     Texas-Rio Grande Valley   37        20          17  0.541
## 312  312           Texas-San Antonio   32        17          15  0.531
## 313  313              Texas Southern   38        24          14  0.632
## 314  314                 Texas State   34        24          10  0.706
## 315  315             Texas Tech NCAA   38        31           7  0.816
## 316  316                       Texas   37        21          16  0.568
## 317  317                      Toledo   33        25           8  0.758
## 318  318                      Towson   32        10          22  0.313
## 319  319                        Troy   30        12          18  0.400
## 320  320                      Tulane   31         4          27  0.129
## 321  321                       Tulsa   32        18          14  0.563
## 322  322                        UCLA   33        17          16  0.515
## 323  323             Utah State NCAA   35        28           7  0.800
## 324  324                 Utah Valley   35        25          10  0.714
## 325  325                        Utah   31        17          14  0.548
## 326  326                  Valparaiso   33        15          18  0.455
## 327  327                  Vanderbilt   32         9          23  0.281
## 328  328                Vermont NCAA   34        27           7  0.794
## 329  329              Villanova NCAA   36        26          10  0.722
## 330  330 Virginia Commonwealth NCAA   33        25           8  0.758
## 331  331                         VMI   32        11          21  0.344
## 332  332          Virginia Tech NCAA   35        26           9  0.743
## 333  333               Virginia NCAA   38        35           3  0.921
## 334  334                      Wagner   30        13          17  0.433
## 335  335                 Wake Forest   31        11          20  0.355
```

```
## 336  336            Washington State    32          11            21  0.344
## 337  337             Washington NCAA    36          27             9  0.750
## 338  338                  Weber State    33          18            15  0.545
## 339  339                West Virginia    36          15            21  0.417
## 340  340            Western Carolina    32           7            25  0.219
## 341  341            Western Illinois    31          10            21  0.323
## 342  342            Western Kentucky    34          20            14  0.588
## 343  343            Western Michigan    32           8            24  0.250
## 344  344               Wichita State    37          22            15  0.595
## 345  345              William & Mary    31          14            17  0.452
## 346  346                    Winthrop    30          18            12  0.600
## 347  347              Wisconsin NCAA    34          23            11  0.676
## 348  348                Wofford NCAA    35          30             5  0.857
## 349  349                Wright State    35          21            14  0.600
## 350  350                     Wyoming    32           8            24  0.250
## 351  351                      Xavier    35          19            16  0.543
## 352  352                    Yale NCAA    30          22             8  0.733
## 353  353            Youngstown State    32          12            20  0.375
##     OverallSRS OverallSOS ConferenceWins ConferenceLosses HomeWins HomeLosses
## 1        -1.91      -7.34             14                4       13          2
## 2        -4.28       0.24              8               10        9          6
## 3         4.86       1.09              8               10       14          3
## 4       -19.23      -8.38              4               14        4          7
## 5         0.36      -1.52             10                8       11          5
## 6       -15.60      -7.84              9                9        8          3
## 7         9.45       9.01              8               10       10          6
## 8        -9.38      -6.70              7                9        6          8
## 9       -22.08      -8.97              6               12       10          3
## 10       -4.19      -7.23              9                9        8          7
## 11       -3.73       0.10              6               12        9          5
## 12       10.28       6.04             12                6       13          3
## 13        8.32       6.32              8               10       12          5
## 14       -4.87      -2.07              5               13        7          9
## 15      -14.43      -8.18             10                8        8          2
## 16       -7.10      -1.23              7               11       10          4
## 17       11.75       8.78              8               10       12          6
## 18       -7.57      -4.73              8               10       10          4
## 19       20.84      10.92             11                7       15          2
## 20        0.59      -4.41             13                5       10          2
## 21        3.39       1.21              6               12        7          7
## 22       13.38       9.26             10                8       13          5
## 23        9.12      -2.60             16                2       13          1
## 24      -11.98      -9.74              9                7       11          4
## 25      -13.92      -4.69              5               11        5         11
## 26        3.61       1.08              7               11        8          7
## 27        5.83       7.76              5               13       10          8
```

```
## 28      -6.61      -5.39            7             11        7          7
## 29       4.24       0.86           12              6       14          2
## 30      -0.08      -0.90            9              9       10          6
## 31       6.15       3.31           11              5       13          3
## 32      -0.62      -3.29            7              7       13          3
## 33     -15.19      -7.66            7             11        8          6
## 34       0.59      -2.93           13              5       12          3
## 35      15.56       2.62           16              2       14          0
## 36       9.22       8.10            7             11       12          4
## 37     -13.95      -3.54            2             14        4          8
## 38      -5.12      -2.63            7              9        9          4
## 39      -3.29      -1.14           10              6        9          4
## 40      -6.54      -3.39            7              9        7          9
## 41      -3.89      -4.12            7              9        8          7
## 42      -6.26      -1.50            7              9        7          6
## 43       5.70      -2.30           15              1       12          2
## 44     -11.12      -3.19            4             12        7          7
## 45      -1.62      -5.55           10              6       12          3
## 46      -3.16       5.42            3             15        7          9
## 47      -3.62      -4.39           12              4       12          4
## 48      -8.79      -4.85           11              7        5          8
## 49     -11.37      -4.81            8             10        8          5
## 50     -14.02      -6.71            5             13        5          7
## 51      13.37       5.58           13              5       15          2
## 52       2.79      -0.34           10              8       13          4
## 53      -2.43      -4.19            9              7       12          4
## 54      -8.34      -0.89            5             13        5          9
## 55      -7.87      -0.76            7             11        8          6
## 56     -24.83       0.67            0             16        3          8
## 57      14.53       5.50           14              4       16          2
## 58      -7.94       0.03            4             14        8          7
## 59      13.85       8.99            9              9       14          5
## 60      -7.96      -1.52            5             13        8          9
## 61      -0.50      -1.94            9              9       10          4
## 62       1.23      -3.83           13              5       15          1
## 63       2.36      -2.95           12              6       13          2
## 64      -0.11       1.41            7             11        8          9
## 65       9.68       3.60           10              8       15          2
## 66      -5.18      -2.14            5              9        5          7
## 67       6.81       4.32            6             12       13          5
## 68     -18.90      -7.11            7              9        3          7
## 69      -6.12      -2.02            7              7        9          4
## 70      12.00       8.59            9              9       13          6
## 71      -5.75      -3.00            2             12        8          6
## 72       6.38       1.96           14              4       14          2
## 73       9.67       2.91           13              5       13          4
```

```
## 74      -26.82     -10.13          2          14        3         9
## 75       -7.91      -4.82          8          10        9         7
## 76      -11.84      -2.97          3          13        6         7
## 77        6.00       4.05          7          11       16         7
## 78       -6.33      -0.36          8          10        6         6
## 79        2.52      -0.66         12           6       13         2
## 80       -7.02      -2.93          7          11        9         7
## 81       26.90      11.98         14           4       15         2
## 82        0.53      -0.63         10           8       14         4
## 83       -5.49       1.31          3          15        8         9
## 84        5.21      -1.79         13           5       13         3
## 85      -11.81      -5.01          7          11        7         6
## 86       -7.40      -2.47          6          12        9         5
## 87        0.40       4.43          9           9       11         7
## 88       -6.77      -4.17         12           8        9         4
## 89      -11.53      -3.56          7          11        5        11
## 90       -3.84       0.13          5          13        9         7
## 91      -10.20      -7.14          6          12        5         7
## 92       -6.09      -7.24         12           6       13         4
## 93      -13.57      -8.09          9           7        6         3
## 94       -1.42      -1.76          8          10        9         5
## 95       -5.11      -1.48          9           7       10         4
## 96       -4.55      -1.45         10           8       12         4
## 97       17.99      10.26         13           5       15         1
## 98       15.42      11.22          9           9        9         6
## 99       -5.02      -2.40          3          15        9        10
## 100       8.99       0.67         13           5       13         4
## 101       7.46      -1.50         13           5       13         3
## 102      -2.61      -4.43         11           6       13         0
## 103       0.84       0.08         11           7       11         6
## 104      -6.65       1.20          4          14        6        11
## 105       6.80       5.31          9           9       13         6
## 106       3.75       0.13         12           6       10         4
## 107       2.44       0.65         13           5       13         1
## 108       6.93       8.15          6          12       11         7
## 109       5.31       8.12          2          16        8         9
## 110      27.79       5.01         16           0       17         0
## 111      -9.73      -9.34         10           8       11         3
## 112       3.85      -1.49         10           6       12         2
## 113      -2.24      -0.36         10           8       15         3
## 114      -3.34      -5.06          9           7       12         3
## 115      -3.95      -4.89         10           6       10         4
## 116       1.86       0.66         10           4        9         2
## 117      -1.30      -3.61          9           7       12         5
## 118      -5.63      -4.39          9           7        9         4
## 119       4.68      -4.53         15           3       15         1
```

```
## 120     -6.43     -4.01            6              12        8          6
## 121     -9.36     -5.53            8              10        9          4
## 122     18.91      4.61           16               2       19          1
## 123    -12.30     -9.52           10               6        6          8
## 124    -13.17     -4.81            7              13        6          7
## 125    -18.74     -6.19            2              18        4         11
## 126     -3.15     -1.99           10               8       12          4
## 127     -2.06      0.25            9               9       12          4
## 128      8.95     11.53            7              13        9          6
## 129    -18.93     -5.11            1              17        5          9
## 130     -2.72      0.62            7              11        9          5
## 131     13.82     10.10            8              12       15          6
## 132     -4.78     -5.50           12               6        8          3
## 133     18.07      9.30            9               9       12          4
## 134     14.27      9.84           10              10       14          4
## 135     -3.47     -3.60            9               7       11          5
## 136     -2.72     -3.04            8              10       11          4
## 137    -14.60     -9.33           10               8        9          4
## 138      1.59     -5.47           15               3       11          1
## 139     -8.27     -4.74            5              11        5          8
## 140     -8.52     -4.07            6              12        8          6
## 141     15.39      9.18           14               4       13          2
## 142     18.35     12.79           12               6       16          0
## 143    -16.17     -1.04            3              13        6          8
## 144      1.26      0.61           11               7       14          3
## 145     21.43     10.29           15               3       17          1
## 146     -3.61      1.29            8              10        5          9
## 147    -11.17     -4.93            7              11        4         11
## 148     -5.94     -7.22           12               6       13          2
## 149     -2.34     -4.46           12               6       11          3
## 150      5.27     -3.88           14               2       16          1
## 151      9.21     -1.20           14               2       14          3
## 152     -4.57     -0.44            8               8        9          5
## 153     -8.99     -9.08            9               9        8          5
## 154     -8.43     -6.62            5              11       10          5
## 155     -2.07     -1.51           10               8       10          4
## 156      0.81     -1.96            9               9       14          3
## 157     16.50      9.16           16               2       15          2
## 158      0.95     -2.31            9               9       15          1
## 159     17.28     11.04           10               8       14          4
## 160      3.99     -0.16           12               6       13          4
## 161      2.93      0.90            8               8       12          4
## 162     -9.13     -4.60            7              11        7          5
## 163    -15.11     -4.21            3              13        3          9
## 164    -12.89     -7.32            8              10        4          9
## 165     -9.49     -7.53            7              11        4          7
```

```
## 166     14.78      6.96          12           6      16        3
## 167     -0.64     -0.61          11           7      16        3
## 168     -5.85     -5.85          11           5      13        4
## 169    -24.21     -7.91           5          11       5        7
## 170     16.01     10.09          13           7      15        3
## 171     -8.28     -6.95           7           9       8        5
## 172     -3.02     -0.18           4          14       9        8
## 173    -14.41     -6.15           5          13       7        8
## 174     10.70      5.09          11           7      17        2
## 175     -2.86     -0.24           6          12       9        6
## 176      9.39      8.70           5          13      11        5
## 177      0.77      2.31           7          11      10        5
## 178     24.93     12.34          16           4      15        1
## 179     21.82     10.55          15           5      17        1
## 180     -6.09      1.44           8          10       9        5
## 181     -8.90     -2.50           4          14       6        8
## 182     12.52     11.27           9          11      14        3
## 183     15.96      9.07          10           8      14        3
## 184    -22.47     -6.99           4          14       6        6
## 185     12.32      8.13          10           8      11        5
## 186     -6.61     -1.55           6          10       8        5
## 187     -1.20     -1.03          10           8      11        4
## 188      8.60      9.16           5          13       9        7
## 189    -10.66     -4.83          10           8       6        6
## 190     -7.91     -5.81          11           9       9        4
## 191      1.50     -5.05          16           4      12        2
## 192     -7.73     -1.76           8          10       7        7
## 193    -15.98    -10.70           4          12       6        7
## 194    -14.57     -6.02           6          12       4        9
## 195      8.96     -3.11          16           2      15        1
## 196    -10.09     -4.00           8          10       8        6
## 197     -2.32     -3.39          13           3      10        2
## 198     14.85     11.31           6          14      13        5
## 199      1.58      0.48          11           7      10        6
## 200     16.00      2.68          15           3      15        0
## 201    -18.66     -6.03           3          13       4       10
## 202     10.05     -2.38          15           1      16        1
## 203     -0.55      1.61           7          11       9        7
## 204     -8.88     -6.47          12           6      12        4
## 205    -11.33     -7.77           6          12       8        8
## 206    -11.87     -6.98           7          11       9        4
## 207     -3.63     -4.74           8           8      11        5
## 208     -8.04     -8.74          14           2      11        2
## 209    -11.00     -1.69           7           9       8        4
## 210    -19.81     -2.35           2          14       3       10
## 211    -11.24     -9.88          13           3      11        2
```

```
## 212    -11.53    -11.24          10              6      10         2
## 213      4.08     -0.90          15              3      15         2
## 214     14.94      6.22           9              9      17         5
## 215     -8.10     -2.04           5             13       5         9
## 216     23.94     11.35          16              2      14         2
## 217     -4.01     -2.07           9              7      10         3
## 218     -8.78     -3.55           6             10       8         6
## 219     -3.47     -0.70           9              7      10         3
## 220     -0.25     -3.48           8             10      12         4
## 221      3.95     -0.70          14              4      11         2
## 222    -10.75     -6.23           8             12       5         7
## 223     -3.92     -6.76          15              5      10         3
## 224      2.58      1.11           8             10      10         6
## 225     -1.54      0.62           9              9       9         5
## 226      4.67     -2.39          13              5      17         1
## 227    -17.06     -5.99           6             12       8         7
## 228      9.97      9.16           4             16      10         8
## 229      7.97      8.28           3             15      11         8
## 230     -2.27     -1.48          11              7      10         6
## 231     13.89     11.00           8             12      12         6
## 232     -1.84      3.16           6             12      11         5
## 233      8.09     11.53           5             13       8         7
## 234     15.30     12.21           7             11      11         4
## 235      3.74     -1.18          13              5      14         2
## 236     -9.34     -1.91           7              9       7         6
## 237      7.84      4.29          10              8      10         5
## 238     13.95      6.13          10              8      13         4
## 239     -2.46      1.97           4             12       9         7
## 240     12.55     11.51           7             13       9         6
## 241      2.04     -0.76           7              7      10         4
## 242      0.90      0.63           6             10       9         5
## 243      7.88      6.69           3             15      11         7
## 244     -8.95     -5.78          11              9      12         4
## 245    -10.80      0.51           0             16       6        11
## 246     -7.29     -9.17          17              1      11         0
## 247     -3.56     -5.35           9              7      12         3
## 248     -3.43     -0.89           8              6       7         5
## 249      8.29      6.71           7             11      11         7
## 250     21.40     11.99          16              4      15         0
## 251     -6.78     -8.03          11              7       7         7
## 252      1.28     -1.95          12              4      11         3
## 253      2.83      1.19           9              9       9         5
## 254     -6.56     -2.06           8             10       9         7
## 255     -1.37     -0.79           6             12       7        10
## 256     -4.96     -6.51          11              7       9         3
## 257     -9.41     -7.62          11              7      13         4
```

```
## 258      8.89      9.76            7            13       10         7
## 259     -8.78     -6.78            8            12        9         5
## 260     -7.88     -8.11           11             7       10         3
## 261     -8.74     -6.16           12             6       12         4
## 262     -0.43      1.66            6            12       10         5
## 263      5.06      2.23           10             8       15         2
## 264     12.58      4.40           11             5       14         3
## 265    -12.15     -6.40            6            12        7         7
## 266     -3.17     -5.84           16             2       12         2
## 267      0.69     -2.03            6            12       11         6
## 268      5.29      2.47           11             7       14         3
## 269      6.12      3.61            7             9       12         4
## 270      8.56      1.13            9             7       13         3
## 271    -16.37      0.59            1            17        4        11
## 272     -1.91      0.92            8             8       11         6
## 273    -20.51     -9.04            8             8        6         5
## 274     -1.47     -3.85            6            10       12         6
## 275     10.25      8.34            9             9       11         4
## 276     -7.90     -6.47           11             7        7         7
## 277     -4.46     -2.71            8            10       13         6
## 278    -16.86     -8.30            5            11        5         6
## 279    -15.30     -4.27            1            16        5         9
## 280      8.34      9.63           11             7       11         6
## 281      5.92     -4.74           14             2       14         1
## 282     -5.52     -3.06            7             9        8         6
## 283      5.96      1.69            8            10       18         5
## 284    -11.32     -4.39            5            13        6         8
## 285     -7.59     -6.30           12             6        9         5
## 286      8.23      4.96            8            10       12         5
## 287    -14.47     -5.06            6            12        7         8
## 288      1.39     -0.17           10             8        9         6
## 289      5.81      4.12            6            12       10         8
## 290      2.58     -0.83           11             7       11         3
## 291     -8.91     -6.22            9            11       11         4
## 292    -16.79     -7.37            6            12        6         5
## 293      3.34      0.25           12             6        9         5
## 294    -10.19     -8.45            9             9       11         3
## 295      7.88      5.50            8            10       11         4
## 296      6.03      5.13            8            10       10         4
## 297    -10.99     -5.92            7            11       10         6
## 298    -15.01     -3.46            3            13        7         8
## 299     -2.10     -6.48           12             4       10         4
## 300     13.73     10.09           10             8       13         6
## 301      8.17      4.93           13             5       13         2
## 302    -10.94     -4.32            6            12       10         4
## 303    -11.62     -3.15            6            12        6         7
```

```
## 304    -14.76     -3.00          4          14       5       10
## 305     21.55     10.16         15           3      18        0
## 306     -9.11     -5.68          9           9       9        7
## 307      7.21      8.59          6          12       9        8
## 308     -0.50      0.84         12           6       9        5
## 309     13.81      9.60          7          11      15        5
## 310     -8.23     -1.71          3          15       8        8
## 311     -2.04     -1.61          9           7      11        9
## 312      0.69      0.05         11           7      11        4
## 313     -5.55     -6.99         14           4      10        2
## 314      1.54     -3.00         12           6      11        4
## 315     22.79      9.53         14           4      17        1
## 316     16.06     11.46          8          10      15        6
## 317      8.21      0.71         13           5      14        2
## 318     -8.44     -3.57          6          12       5        8
## 319     -6.12     -1.01          5          13       8        8
## 320     -6.72      3.63          0          18       3       12
## 321      4.65      3.86          8          10      14        3
## 322      7.17      6.78          9           9      13        5
## 323     11.98      0.89         15           3      14        1
## 324      3.00     -2.30         12           4      14        1
## 325      6.13      5.10         11           7      10        5
## 326     -3.38     -1.00          7          11       8        7
## 327      3.57      7.79          0          18       8       10
## 328      4.88     -4.18         14           2      16        2
## 329     14.33      7.97         13           5      13        2
## 330     11.96      2.87         16           2      16        1
## 331    -10.50      0.09          4          14       8        7
## 332     19.28      7.79         12           6      14        2
## 333     25.46     10.15         16           2      15        1
## 334    -12.99     -7.63          8          10       7        7
## 335      1.21      8.47          4          14       8        8
## 336     -1.67      2.20          4          14       9        7
## 337     12.01      7.01         15           3      15        1
## 338     -4.37     -6.12         11           9      10        5
## 339      6.94     10.57          4          14      11        7
## 340     -9.28      0.34          4          14       4        8
## 341    -10.04     -5.18          4          12       7        6
## 342      3.14      0.96         11           7      10        4
## 343     -6.26      1.48          2          16       5        9
## 344      7.72      5.91         10           8      10        4
## 345     -4.78     -2.08         10           8       9        5
## 346     -2.89     -3.74         10           6      11        3
## 347     17.90     11.01         14           6      12        3
## 348     13.92      0.80         18           0      15        1
## 349      3.29     -0.89         13           5      15        2
```

```
## 350      -9.75       0.19              4               14        6         10
## 351       9.61       8.06              9                9       13          5
## 352       5.52      -1.24             10                4       11          2
## 353      -7.78      -2.05              8               10        6          7
##     AwayWins AwayLosses ForPoints OppPoints Blank Minutes FieldGoalsMade
## 1         10          4      2502      2161    NA    1370            897
## 2          3          9      2179      2294    NA    1300            802
## 3          1         10      2271      2107    NA    1325            797
## 4          0         18      1938      2285    NA    1295            736
## 5          6          6      2470      2370    NA    1410            906
## 6          3         13      2086      2235    NA    1250            712
## 7          4          8      2448      2433    NA    1365            856
## 8          6         10      2150      2216    NA    1300            728
## 9          0         17      1995      2194    NA    1245            709
## 10         6          8      2161      2070    NA    1220            780
## 11         2         12      2557      2539    NA    1290            880
## 12         6          5      2638      2494    NA    1380            899
## 13         4          7      2269      2205    NA    1290            794
## 14         3         12      2301      2353    NA    1250            817
## 15         4         17      2108      2269    NA    1300            719
## 16         2         13      2359      2475    NA    1310            796
## 17         5          8      2559      2458    NA    1370            893
## 18         3         13      2268      2305    NA    1280            837
## 19         4          6      3188      2750    NA    1615           1097
## 20         8          7      2712      2413    NA    1335            973
## 21         8          7      2478      2362    NA    1340            889
## 22         5          6      2442      2302    NA    1365            869
## 23        12          3      2868      2439    NA    1330           1042
## 24         3         12      2290      2245    NA    1250            833
## 25         5         12      2152      2376    NA    1320            793
## 26         3         10      2364      2263    NA    1330            845
## 27         2          8      2196      2256    NA    1260            768
## 28         8         10      2385      2387    NA    1325            922
## 29         5          8      2668      2496    NA    1370            945
## 30         5          8      2329      2285    NA    1405            812
## 31         5          8      2527      2436    NA    1295            878
## 32         7          9      2355      2204    NA    1290            805
## 33         2         13      2088      2314    NA    1200            709
## 34         7          8      2534      2418    NA    1325            867
## 35        12          3      3037      2550    NA    1445           1083
## 36         2         10      2374      2337    NA    1330            861
## 37         1         14      1930      2183    NA    1185            706
## 38         7         10      2412      2412    NA    1375            887
## 39         4         11      2437      2414    NA    1385            868
## 40         4         11      2601      2701    NA    1375            968
## 41         8          6      2403      2241    NA    1260            831
```

```
## 42        3        12      2038      2128     NA     1260          735
## 43       13         2      2675      2353     NA     1500          987
## 44        2        14      2153      2279     NA     1325          792
## 45        7         6      2350      2114     NA     1290          825
## 46        1        10      2121      2387     NA     1245          739
## 47        5         8      2495      2328     NA     1325          867
## 48        9         5      2250      2376     NA     1295          798
## 49        5        13      2383      2498     NA     1330          810
## 50        5        12      2205      2367     NA     1260          754
## 51        5         5      2385      2128     NA     1325          819
## 52        7         6      2896      2690     NA     1425          991
## 53        4        11      2574      2369     NA     1370          926
## 54        3         9      1776      1992     NA     1160          598
## 55        4        11      2268      2376     NA     1285          833
## 56        0        19      1970      2710     NA     1280          718
## 57        7         4      2510      2194     NA     1410          872
## 58        4        10      2553      2584     NA     1210          899
## 59        4         6      2339      2174     NA     1360          833
## 60        2        12      2300      2437     NA     1250          799
## 61        6        11      2641      2520     NA     1370          899
## 62        9         9      2648      2458     NA     1415          955
## 63        8         5      2453      2265     NA     1330          878
## 64        3         8      2393      2406     NA     1285          866
## 65        5         9      2648      2429     NA     1445          924
## 66        4         9      2059      2052     NA     1160          778
## 67        1         8      2436      2354     NA     1325          874
## 68        4        16      2118      2507     NA     1330          710
## 69        6        12      2149      2211     NA     1260          754
## 70        4         8      2727      2561     NA     1420          980
## 71        2        12      2160      2116     NA     1210          791
## 72        7         5      2409      2229     NA     1365          852
## 73        7         4      2406      2183     NA     1335          902
## 74        2        15      1956      2371     NA     1240          695
## 75        7         8      2352      2380     NA     1360          830
## 76        1        14      2086      2319     NA     1205          755
## 77        3         9      2821      2751     NA     1450         1009
## 78        4        14      2279      2464     NA     1240          784
## 79        7         6      2563      2387     NA     1385          908
## 80        3        11      2425      2475     NA     1280          874
## 81        7         2      3143      2576     NA     1525         1157
## 82        4         6      2351      2314     NA     1295          810
## 83        1        11      2088      2299     NA     1255          749
## 84        8         6      2707      2371     NA     1375         1004
## 85        5        10      2299      2457     NA     1310          826
## 86        3        11      2562      2577     NA     1265          902
## 87        4        10      2203      2213     NA     1295          818
```

```
## 88          4          13        2460        2527     NA     1380            876
## 89          6           7        2266        2445     NA     1290            812
## 90          2          13        2234        2317     NA     1295            747
## 91          3          13        2072        2167     NA     1240            761
## 92          7           9        2620        2515     NA     1420            921
## 93          6          14        1927        2057     NA     1245            719
## 94          6           9        2322        2243     NA     1335            809
## 95          4          10        2280        2360     NA     1280            809
## 96          6           9        2798        2734     NA     1360            986
## 97          6           4        2771        2485     NA     1500            960
## 98          5           6        2440        2289     NA     1455            857
## 99          3           8        2108        2141     NA     1295            759
## 100         7           3        2448        2162     NA     1285            847
## 101        11           4        2562        2182     NA     1340            930
## 102         8           9        2718        2468     NA     1430            955
## 103         6           6        2314        2289     NA     1330            825
## 104         2          10        2097        2356     NA     1340            745
## 105         5           6        2625        2576     NA     1355            894
## 106         7           7        2726        2489     NA     1330           1012
## 107         7           7        2598        2491     NA     1360            903
## 108         3           9        2091        2130     NA     1285            755
## 109         2           9        2274        2364     NA     1280            776
## 110         9           1        3243        2400     NA     1480           1177
## 111         6          12        2426        2336     NA     1370            834
## 112         5           8        2560        2353     NA     1370            907
## 113         5          13        3090        3036     NA     1535           1106
## 114         6          12        2847        2669     NA     1430            966
## 115         8           9        2477        2407     NA     1345            848
## 116         8           9        2222        2174     NA     1265            787
## 117         4           5        2241        2131     NA     1250            788
## 118         5           8        2115        2105     NA     1255            763
## 119        10           6        2919        2553     NA     1440            998
## 120         6          11        2216        2296     NA     1335            848
## 121         3          13        2465        2484     NA     1225            865
## 122        10           1        2787        2258     NA     1480            982
## 123         9           8        2684        2661     NA     1370            937
## 124         5          11        2198        2369     NA     1200            779
## 125         1          13        2174        2460     NA     1285            778
## 126         4          12        2391        2385     NA     1295            865
## 127         2          10        2257        2314     NA     1330            813
## 128         1           9        2398        2483     NA     1335            862
## 129         0          14        2044        2355     NA     1255            711
## 130         4           9        2146        2218     NA     1250            757
## 131         3           9        2504        2374     NA     1420            919
## 132         5           9        2532        2508     NA     1320            857
## 133         5           6        2692        2385     NA     1400            974
```

```
## 134        4          6       2740       2585     NA     1415            914
## 135        6          9       2728       2556     NA     1330            991
## 136        3         13       2507       2429     NA     1320            891
## 137        3         14       1974       2107     NA     1285            717
## 138       10          7       2574       2283     NA     1345            953
## 139        6         11       2385       2402     NA     1285            889
## 140        4         11       2322       2409     NA     1350            821
## 141        7          5       2236       2025     NA     1360            806
## 142        3          8       2725       2525     NA     1455            989
## 143        0         15       2030       2455     NA     1285            742
## 144        8          7       2481       2417     NA     1330            878
## 145        8          2       2806       2394     NA     1490            978
## 146        3          8       2091       2243     NA     1240            719
## 147        6          9       2209       2349     NA     1220            802
## 148        6         10       2597       2322     NA     1340            902
## 149        9          8       2479       2413     NA     1250            856
## 150       11          3       2654       2209     NA     1440            963
## 151       14          4       3076       2597     NA     1480           1071
## 152        4         12       2545       2584     NA     1375            877
## 153        7          9       2373       2337     NA     1285            822
## 154        5         12       2407       2408     NA     1375            803
## 155        7          7       2612       2575     NA     1290            901
## 156        4         12       2773       2637     NA     1425            922
## 157        9          1       2815       2558     NA     1435            988
## 158        4         10       2386       2235     NA     1335            840
## 159        5          6       2536       2324     NA     1380            854
## 160        5          7       2231       2066     NA     1360            831
## 161        8          7       2278       2125     NA     1360            817
## 162        3         16       2332       2477     NA     1295            842
## 163        2         18       1988       2271     NA     1310            747
## 164        4         11       1833       2011     NA     1285            651
## 165        6         11       2086       2147     NA     1250            739
## 166        6          4       2629       2363     NA     1375            893
## 167        6         10       2978       2979     NA     1495           1047
## 168        6          8       2296       2177     NA     1400            820
## 169        2         17       1833       2291     NA     1285            666
## 170        6          5       2429       2228     NA     1360            858
## 171        5         12       2456       2420     NA     1290            875
## 172        1         11       2263       2354     NA     1290            808
## 173        2         14       2149       2303     NA     1245            760
## 174        3          8       2882       2680     NA     1450           1013
## 175        2         12       2248       2246     NA     1245            787
## 176        0         10       2297       2275     NA     1285            805
## 177        4         10       2267       2233     NA     1285            777
## 178        8          4       3025       2534     NA     1570           1071
## 179        7          4       2575       2158     NA     1480            941
```

```
## 180        2        11      2141      2313     NA     1280          778
## 181        3        12      2146      2315     NA     1245          765
## 182        2         9      2543      2498     NA     1445          887
## 183        5         5      2628      2394     NA     1370          936
## 184        0        20      2058      2498     NA     1285          730
## 185        6         5      2485      2347     NA     1325          876
## 186        3        13      2220      2341     NA     1290          774
## 187        5         9      2206      2157     NA     1285          774
## 188        2         8      2150      2168     NA     1290          758
## 189        5        11      2286      2490     NA     1420          794
## 190        4        12      2502      2552     NA     1280          858
## 191        9         5      2665      2397     NA     1405          975
## 192        4        11      2431      2543     NA     1335          855
## 193        2        13      2161      2304     NA     1210          758
## 194        5        13      2092      2286     NA     1240          733
## 195       10         3      2727      2255     NA     1320          988
## 196        4        13      2061      2250     NA     1240          732
## 197        9         8      2523      2454     NA     1290          950
## 198        2        10      2586      2421     NA     1445          916
## 199        6         5      2276      2242     NA     1250          791
## 200        9         3      2723      2270     NA     1360          924
## 201        1        14      1742      1990     NA     1160          623
## 202       10         1      2734      2259     NA     1405          962
## 203        4         9      2443      2454     NA     1280          814
## 204        5         9      2384      2325     NA     1345          862
## 205        5        10      2318      2432     NA     1285          795
## 206        3        13      2260      2302     NA     1250          807
## 207       11         8      2472      2401     NA     1410          849
## 208        9         8      2652      2503     NA     1460          900
## 209        2        18      2145      2358     NA     1285          744
## 210        1        15      1857      2313     NA     1245          655
## 211        7        10      2241      2209     NA     1290          825
## 212        4        12      2429      2266     NA     1365          874
## 213       11         4      2738      2443     NA     1445          997
## 214        4         6      2882      2568     NA     1450         1061
## 215        2        12      2515      2670     NA     1335          880
## 216       11         1      3089      2636     NA     1445         1118
## 217        4         9      2556      2541     NA     1410          874
## 218        4        11      2220      2196     NA     1200          822
## 219        5        13      2485      2531     NA     1325          902
## 220        6         7      2302      2077     NA     1325          831
## 221        8         6      2564      2406     NA     1385          880
## 222        5        13      2262      2402     NA     1250          785
## 223       11         5      2450      2229     NA     1295          849
## 224        5         9      2506      2413     NA     1375          940
## 225        3         9      2215      2230     NA     1360          769
```

```
## 226        7          7      2747      2410     NA     1410          979
## 227        3         13      2051      2280     NA     1250          725
## 228        1          9      2108      2082     NA     1290          735
## 229        1          9      2266      2276     NA     1320          776
## 230        6         10      2534      2515     NA     1325          864
## 231        4          7      2419      2318     NA     1405          839
## 232        3         10      2174      2288     NA     1255          803
## 233        2         10      2178      2288     NA     1285          760
## 234        5          7      2423      2318     NA     1365          883
## 235        8          4      2300      2128     NA     1400          823
## 236        3         13      2314      2483     NA     1285          854
## 237        6          5      2275      2165     NA     1250          800
## 238        5          7      2661      2364     NA     1530          958
## 239        5         10      2138      2241     NA     1290          710
## 240        3          9      2229      2196     NA     1295          791
## 241        8          5      2250      2117     NA     1265          821
## 242        2         11      2572      2529     NA     1370          865
## 243        0         11      2306      2267     NA     1330          769
## 244        3         10      2475      2404     NA     1290          856
## 245        1         12      2078      2389     NA     1290          707
## 246        9         12      2627      2547     NA     1400          903
## 247        8         12      2817      2610     NA     1445          966
## 248        8          6      1948      1951     NA     1135          685
## 249        5          6      2428      2374     NA     1385          835
## 250        6          6      2760      2421     NA     1460          967
## 251        9          6      2294      2255     NA     1270          739
## 252        9          7      2440      2247     NA     1330          893
## 253        5          7      2293      2239     NA     1335          821
## 254        4         11      2367      2481     NA     1305          795
## 255        4          7      2314      2333     NA     1320          849
## 256        6         10      2381      2333     NA     1245          847
## 257        5         13      2448      2403     NA     1420          861
## 258        4          9      2105      2132     NA     1250          771
## 259        4         10      2197      2155     NA     1250          806
## 260        4         13      2558      2509     NA     1285          885
## 261        6         11      2517      2510     NA     1320          885
## 262        2         11      2328      2397     NA     1325          793
## 263        4          8      2399      2297     NA     1440          855
## 264        5          5      2463      2185     NA     1365          899
## 265        2         14      2012      2196     NA     1305          707
## 266        8          9      2476      2307     NA     1345          889
## 267        5          9      2583      2487     NA     1355          921
## 268        4          7      2437      2291     NA     1360          845
## 269        5          9      2591      2455     NA     1460          914
## 270        6          5      2358      2101     NA     1250          864
## 271        0         12      2042      2534     NA     1250          713
```

```
## 272        4        6     2115     2172    NA    1255          735
## 273        4       14     2360     2694    NA    1245          825
## 274        4        8     2420     2283    NA    1330          852
## 275        3        8     2508     2443    NA    1375          888
## 276        8        7     2131     2178    NA    1345          772
## 277        3       10     2478     2465    NA    1375          867
## 278        2       19     2388     2593    NA    1375          811
## 279        1       15     2169     2395    NA    1290          775
## 280        4        8     2327     2316    NA    1285          806
## 281        7        6     2789     2425    NA    1320          984
## 282        4        8     2118     2136    NA    1205          729
## 283        5        7     2752     2553    NA    1555          911
## 284        2       13     2216     2378    NA    1265          796
## 285        7        9     2224     2228    NA    1320          765
## 286        2        9     2518     2410    NA    1340          937
## 287        2       11     2265     2520    NA    1280          794
## 288        7        7     2194     2144    NA    1280          802
## 289        3        7     2303     2249    NA    1280          833
## 290        7        7     2392     2160    NA    1330          899
## 291        4       12     2573     2566    NA    1380          881
## 292        1       18     2077     2360    NA    1285          752
## 293        6        7     2250     2145    NA    1385          821
## 294        6       12     2345     2320    NA    1325          841
## 295        4        7     2623     2542    NA    1375          944
## 296        4        9     2255     2227    NA    1245          808
## 297        3        9     2105     2161    NA    1210          745
## 298        0       16     2151     2429    NA    1245          794
## 299       13        4     2364     2188    NA    1335          815
## 300        6        4     2370     2246    NA    1365          808
## 301        8        5     2465     2358    NA    1340          873
## 302        1       14     2356     2486    NA    1250          854
## 303        3       14     2182     2332    NA    1210          751
## 304        3       13     2104     2350    NA    1250          742
## 305        7        3     3035     2580    NA    1505         1106
## 306        5        9     2126     2124    NA    1290          774
## 307        3        7     2253     2297    NA    1280          806
## 308        7       10     2284     2303    NA    1345          781
## 309        3        7     2730     2574    NA    1500          990
## 310        0       13     1859     2001    NA    1170          639
## 311        8        7     2634     2623    NA    1490          895
## 312        5        7     2481     2407    NA    1295          896
## 313       13       11     3102     3008    NA    1550         1083
## 314       10        5     2470     2204    NA    1370          876
## 315        6        3     2765     2261    NA    1530          990
## 316        2        8     2628     2458    NA    1495          929
## 317        8        5     2537     2256    NA    1330          901
```

```
## 318        4          8      2170      2294    NA    1310            810
## 319        4         10      2179      2242    NA    1210            773
## 320        0         11      2082      2403    NA    1240            733
## 321        3          8      2296      2271    NA    1290            776
## 322        3          7      2580      2567    NA    1335            919
## 323        9          4      2753      2349    NA    1410            959
## 324        8          8      2697      2474    NA    1400            930
## 325        6          5      2343      2311    NA    1245            805
## 326        5          8      2205      2187    NA    1335            802
## 327        1         10      2180      2315    NA    1285            736
## 328       11          4      2508      2143    NA    1370            864
## 329        5          6      2654      2425    NA    1455            884
## 330        8          4      2344      2044    NA    1330            820
## 331        2         13      2433      2617    NA    1290            843
## 332        5          5      2574      2172    NA    1410            893
## 333       10          1      2714      2132    NA    1535            974
## 334        6         10      1974      2044    NA    1205            655
## 335        1         10      2119      2344    NA    1250            714
## 336        2          8      2393      2517    NA    1280            833
## 337        7          4      2511      2331    NA    1445            883
## 338        5          7      2617      2454    NA    1330            920
## 339        0         10      2655      2786    NA    1460            901
## 340        3         14      2304      2563    NA    1295            792
## 341        2         14      2198      2272    NA    1245            818
## 342        6          8      2429      2355    NA    1380            853
## 343        2         14      2246      2439    NA    1300            774
## 344        7          7      2609      2542    NA    1485            929
## 345        5         11      2321      2385    NA    1265            841
## 346        7          8      2499      2358    NA    1210            855
## 347        8          5      2333      2099    NA    1385            874
## 348       11          3      2879      2295    NA    1410           1044
## 349        5          8      2561      2365    NA    1405            885
## 350        1         11      2105      2411    NA    1285            690
## 351        4          8      2526      2472    NA    1420            922
## 352        8          5      2427      2202    NA    1215            893
## 353        5         11      2415      2526    NA    1305            879
##     FieldGoalsAttempted FieldGoalPCT ThreePointMade ThreePointAttempts
## 1                  1911        0.469            251                660
## 2                  1776        0.452            234                711
## 3                  1948        0.409            297                929
## 4                  1809        0.407            182                578
## 5                  2003        0.452            234                694
## 6                  1764        0.404            216                673
## 7                  1945        0.440            244                718
## 8                  1750        0.416            274                790
## 9                  1721        0.412            211                646
```

```
## 10          1645      0.474        190          584
## 11          1938      0.454        292          814
## 12          2012      0.447        240          714
## 13          1861      0.427        235          699
## 14          1687      0.484        206          582
## 15          1685      0.427        194          574
## 16          1904      0.418        237          709
## 17          2004      0.446        259          751
## 18          1956      0.428        272          863
## 19          2439      0.450        454         1204
## 20          2060      0.472        268          702
## 21          1892      0.470        202          621
## 22          1966      0.442        274          803
## 23          2094      0.498        343          922
## 24          1899      0.439        204          635
## 25          1842      0.431        275          788
## 26          1801      0.469        252          715
## 27          1808      0.425        218          687
## 28          1922      0.480        221          639
## 29          2134      0.443        272          762
## 30          1874      0.433        240          653
## 31          1878      0.468        226          684
## 32          1818      0.443        255          745
## 33          1680      0.422        230          708
## 34          1928      0.450        311          885
## 35          2344      0.462        344         1022
## 36          1934      0.445        292          827
## 37          1715      0.412        225          687
## 38          2092      0.424        221          683
## 39          1911      0.454        187          589
## 40          2103      0.460        234          648
## 41          1859      0.447        287          800
## 42          1673      0.439        203          619
## 43          2159      0.457        252          701
## 44          1787      0.443        279          743
## 45          1819      0.454        204          594
## 46          1735      0.426        207          592
## 47          1906      0.455        313          912
## 48          1834      0.435        256          791
## 49          1903      0.426        276          784
## 50          1821      0.414        201          612
## 51          1763      0.465        229          628
## 52          2177      0.455        305          822
## 53          2156      0.429        324          945
## 54          1442      0.415        191          615
## 55          1881      0.443        290          795
```

```
## 56          1752      0.410        141          471
## 57          2018      0.432        232          673
## 58          2011      0.447        370         1076
## 59          1857      0.449        217          661
## 60          1834      0.436        303          820
## 61          1986      0.453        284          766
## 62          1995      0.479        320          815
## 63          1818      0.483        227          667
## 64          1818      0.476        259          722
## 65          2036      0.454        230          711
## 66          1720      0.452        237          655
## 67          1959      0.446        252          732
## 68          1827      0.389        240          841
## 69          1705      0.442        243          731
## 70          2044      0.479        372          961
## 71          1750      0.452        264          721
## 72          1893      0.450        312          882
## 73          1789      0.504        205          617
## 74          1963      0.354        234          789
## 75          1808      0.459        282          746
## 76          1723      0.438        227          622
## 77          2143      0.471        241          703
## 78          1874      0.418        294          825
## 79          1943      0.467        279          773
## 80          1900      0.460        237          669
## 81          2418      0.478        278          903
## 82          1894      0.428        271          841
## 83          1791      0.418        155          545
## 84          2067      0.486        274          758
## 85          1912      0.432        272          735
## 86          2125      0.424        272          832
## 87          1823      0.449        187          634
## 88          2040      0.429        310          886
## 89          1862      0.436        335          943
## 90          1776      0.421        264          776
## 91          1755      0.434        259          740
## 92          1939      0.475        269          672
## 93          1640      0.438        180          489
## 94          1971      0.410        290          895
## 95          1787      0.453        269          724
## 96          2239      0.440        279          923
## 97          2171      0.442        272          819
## 98          2015      0.425        291          872
## 99          1890      0.402        276          831
## 100         1866      0.454        342          897
## 101         1962      0.474        338          935
```

```
## 102          1962      0.487        281          719
## 103          1855      0.445        210          637
## 104          1842      0.404        196          631
## 105          2017      0.443        269          757
## 106          2028      0.499        204          632
## 107          1968      0.459        330          859
## 108          1714      0.440        176          573
## 109          1761      0.441        210          653
## 110          2239      0.526        287          790
## 111          1846      0.452        223          553
## 112          2006      0.452        257          777
## 113          2413      0.458        303          879
## 114          2236      0.432        291          844
## 115          1862      0.455        322          846
## 116          1711      0.460        260          722
## 117          1746      0.451        266          751
## 118          1719      0.444        180          583
## 119          2055      0.486        308          800
## 120          1834      0.462        237          670
## 121          1896      0.456        200          588
## 122          2203      0.446        333          939
## 123          2163      0.433        244          669
## 124          1742      0.447        274          752
## 125          1764      0.441        251          678
## 126          1883      0.459        303          856
## 127          1887      0.431        244          734
## 128          2001      0.431        259          751
## 129          1527      0.466        174          508
## 130          1719      0.440        178          491
## 131          2009      0.457        211          676
## 132          1897      0.452        297          845
## 133          2047      0.476        294          811
## 134          2006      0.456        285          782
## 135          2059      0.481        360          949
## 136          1969      0.453        234          695
## 137          1774      0.404        155          554
## 138          2063      0.462        200          634
## 139          1953      0.455        213          645
## 140          1859      0.442        243          670
## 141          1878      0.429        241          721
## 142          2128      0.465        260          743
## 143          1945      0.381        136          438
## 144          2032      0.432        261          775
## 145          2050      0.477        215          607
## 146          1798      0.400        259          783
## 147          1786      0.449        280          736
```

```
## 148          1976      0.456        228          664
## 149          1775      0.482        287          679
## 150          1977      0.487        322          873
## 151          2230      0.480        327          876
## 152          2031      0.432        217          630
## 153          1880      0.437        277          813
## 154          1875      0.428        333          939
## 155          2026      0.445        283          821
## 156          2029      0.454        353          891
## 157          2162      0.457        236          740
## 158          1928      0.436        253          756
## 159          1967      0.434        294          860
## 160          1687      0.493        206          563
## 161          1800      0.454        163          510
## 162          1836      0.459        210          616
## 163          1728      0.432        201          647
## 164          1608      0.405        219          661
## 165          1647      0.449        245          675
## 166          1969      0.454        319          822
## 167          2367      0.442        362         1058
## 168          1893      0.433        261          807
## 169          1705      0.391        172          610
## 170          1909      0.449        247          707
## 171          1852      0.472        246          721
## 172          1831      0.441        256          732
## 173          1676      0.453        168          548
## 174          2238      0.453        260          807
## 175          1746      0.451        228          663
## 176          1851      0.435        268          805
## 177          1847      0.421        269          832
## 178          2230      0.480        319          844
## 179          2102      0.448        287          839
## 180          1858      0.419        212          650
## 181          1820      0.420        237          692
## 182          2039      0.435        191          603
## 183          1982      0.472        292          774
## 184          1923      0.380        172          554
## 185          1906      0.460        272          760
## 186          1794      0.431        259          728
## 187          1714      0.452        224          649
## 188          1764      0.430        264          727
## 189          1915      0.415        168          565
## 190          1882      0.456        306          818
## 191          1983      0.492        287          763
## 192          1993      0.429        260          762
## 193          1920      0.395        168          521
```

```
## 194          1755      0.418        223          723
## 195          2007      0.492        258          731
## 196          1785      0.410        208          656
## 197          1998      0.475        282          719
## 198          2133      0.429        270          800
## 199          1852      0.427        251          745
## 200          1999      0.462        297          855
## 201          1656      0.376        261          787
## 202          2092      0.460        326          977
## 203          1927      0.422        292          844
## 204          1941      0.444        189          566
## 205          1887      0.421        265          738
## 206          1914      0.422        324          883
## 207          1913      0.444        246          713
## 208          2079      0.433        282          770
## 209          1914      0.389        240          805
## 210          1627      0.403        223          687
## 211          1797      0.459        227          649
## 212          1922      0.455        221          693
## 213          2184      0.457        280          818
## 214          2319      0.458        292          830
## 215          1986      0.443        260          744
## 216          2410      0.464        312          862
## 217          1926      0.454        332          910
## 218          1771      0.464        241          647
## 219          2014      0.448        308          897
## 220          1910      0.435        284          838
## 221          1848      0.476        328          858
## 222          1819      0.432        259          749
## 223          1837      0.462        307          849
## 224          1995      0.471        229          642
## 225          1847      0.416        276          792
## 226          2047      0.478        306          844
## 227          1790      0.405        190          646
## 228          1827      0.402        233          744
## 229          1973      0.393        272          863
## 230          1859      0.465        314          816
## 231          1931      0.434        264          774
## 232          1838      0.437        187          633
## 233          1796      0.423        280          753
## 234          1975      0.447        226          654
## 235          2026      0.406        264          756
## 236          1874      0.456        266          734
## 237          1728      0.463        203          633
## 238          2126      0.451        295          840
## 239          1681      0.422        198          566
```

```
## 240          1898      0.417        222          693
## 241          1810      0.454        282          803
## 242          1949      0.444        313          819
## 243          1841      0.418        225          680
## 244          2017      0.424        238          769
## 245          1708      0.414        216          664
## 246          2061      0.438        225          703
## 247          2150      0.449        394         1034
## 248          1666      0.411        207          679
## 249          1973      0.423        225          690
## 250          2145      0.451        365          977
## 251          1730      0.427        348          923
## 252          1938      0.461        285          751
## 253          1914      0.429        172          615
## 254          1867      0.426        277          797
## 255          1800      0.472        251          723
## 256          1893      0.447        222          651
## 257          1991      0.432        280          789
## 258          1842      0.419        191          612
## 259          1800      0.448        175          527
## 260          1926      0.460        243          681
## 261          2035      0.435        254          724
## 262          1943      0.408        280          869
## 263          2052      0.417        205          675
## 264          1905      0.472        253          670
## 265          1639      0.431        184          567
## 266          2011      0.442        308          840
## 267          1951      0.472        254          687
## 268          1940      0.436        261          721
## 269          2002      0.457        284          808
## 270          1851      0.467        258          727
## 271          1765      0.404        210          641
## 272          1653      0.445        239          663
## 273          2080      0.397        351         1199
## 274          1945      0.438        238          642
## 275          2021      0.439        240          741
## 276          1779      0.434        288          849
## 277          1883      0.460        285          761
## 278          1891      0.429        193          582
## 279          1872      0.414        269          817
## 280          1907      0.423        242          662
## 281          1965      0.501        327          802
## 282          1664      0.438        227          666
## 283          2120      0.430        272          819
## 284          1809      0.440        266          719
## 285          1735      0.441        235          696
```

```
## 286          2039      0.460      281          735
## 287          1898      0.418      201          665
## 288          1720      0.466      213          570
## 289          1884      0.442      263          772
## 290          1929      0.466      275          712
## 291          1997      0.441      272          799
## 292          1710      0.440      180          544
## 293          1914      0.429      208          635
## 294          1989      0.423      270          806
## 295          2100      0.450      289          809
## 296          1783      0.453      207          653
## 297          1696      0.439      201          588
## 298          1962      0.405      215          684
## 299          1937      0.421      233          715
## 300          1907      0.424      274          824
## 301          1998      0.437      247          748
## 302          1915      0.446      250          726
## 303          1739      0.432      240          716
## 304          1784      0.416      213          620
## 305          2231      0.496      262          714
## 306          1803      0.429      202          621
## 307          1842      0.438      204          663
## 308          1908      0.409      234          773
## 309          2168      0.457      281          813
## 310          1592      0.401      199          620
## 311          2139      0.418      203          636
## 312          2068      0.433      294          855
## 313          2379      0.455      276          852
## 314          1971      0.444      244          744
## 315          2110      0.469      277          759
## 316          2146      0.433      325          934
## 317          1990      0.453      325          859
## 318          1860      0.435      180          550
## 319          1719      0.450      236          704
## 320          1783      0.411      213          670
## 321          1732      0.448      222          652
## 322          2012      0.457      256          721
## 323          2038      0.471      274          772
## 324          1929      0.482      283          740
## 325          1735      0.464      298          806
## 326          1783      0.450      185          585
## 327          1750      0.421      230          739
## 328          1887      0.458      273          761
## 329          2019      0.438      380         1081
## 330          1872      0.438      235          771
## 331          1974      0.427      323          899
```

```
## 332                1900        0.470            327                831
## 333                2056        0.474            321                813
## 334                1696        0.386            236                773
## 335                1812        0.394            197                640
## 336                1861        0.448            301                828
## 337                1956        0.451            273                780
## 338                1934        0.476            249                684
## 339                2181        0.413            269                852
## 340                1744        0.454            296                800
## 341                1860        0.440            241                626
## 342                1910        0.447            199                608
## 343                1822        0.425            228                723
## 344                2277        0.408            276                890
## 345                1773        0.474            254                735
## 346                1878        0.455            372                997
## 347                1945        0.449            241                672
## 348                2132        0.490            385                929
## 349                2028        0.436            281                818
## 350                1656        0.417            248                720
## 351                1977        0.466            245                740
## 352                1812        0.493            230                634
## 353                2057        0.427            303                891
##     ThreePointPct FreeThrowsMade FreeThrowsAttempted FreeThrowPCT
## 1           0.380            457                 642        0.712
## 2           0.329            341                 503        0.678
## 3           0.320            380                 539        0.705
## 4           0.315            284                 453        0.627
## 5           0.337            424                 630        0.673
## 6           0.321            446                 684        0.652
## 7           0.340            492                 739        0.666
## 8           0.347            420                 564        0.745
## 9           0.327            366                 543        0.674
## 10          0.325            411                 589        0.698
## 11          0.359            505                 706        0.715
## 12          0.336            600                 882        0.680
## 13          0.336            446                 620        0.719
## 14          0.354            461                 703        0.656
## 15          0.338            476                 702        0.678
## 16          0.334            530                 721        0.735
## 17          0.345            514                 769        0.668
## 18          0.315            322                 471        0.684
## 19          0.377            540                 760        0.711
## 20          0.382            498                 705        0.706
## 21          0.325            498                 709        0.702
## 22          0.341            430                 635        0.677
## 23          0.372            441                 598        0.737
```

```
## 24          0.321          420          689          0.610
## 25          0.349          291          452          0.644
## 26          0.352          422          584          0.723
## 27          0.317          442          633          0.698
## 28          0.346          320          468          0.684
## 29          0.357          506          755          0.670
## 30          0.368          465          672          0.692
## 31          0.330          545          750          0.727
## 32          0.342          490          685          0.715
## 33          0.325          440          618          0.712
## 34          0.351          489          653          0.749
## 35          0.337          527          767          0.687
## 36          0.353          360          488          0.738
## 37          0.328          293          452          0.648
## 38          0.324          417          620          0.673
## 39          0.317          514          726          0.708
## 40          0.361          431          676          0.638
## 41          0.359          454          576          0.788
## 42          0.328          365          515          0.709
## 43          0.359          449          640          0.702
## 44          0.376          290          424          0.684
## 45          0.343          496          693          0.716
## 46          0.350          436          603          0.723
## 47          0.343          448          590          0.759
## 48          0.324          398          575          0.692
## 49          0.352          487          667          0.730
## 50          0.328          496          634          0.782
## 51          0.365          518          798          0.649
## 52          0.371          609          908          0.671
## 53          0.343          398          588          0.677
## 54          0.311          389          524          0.742
## 55          0.365          312          457          0.683
## 56          0.299          393          581          0.676
## 57          0.345          534          758          0.704
## 58          0.344          385          523          0.736
## 59          0.328          456          623          0.732
## 60          0.370          399          585          0.682
## 61          0.371          559          778          0.719
## 62          0.393          418          563          0.742
## 63          0.340          470          615          0.764
## 64          0.359          402          579          0.694
## 65          0.323          570          757          0.753
## 66          0.362          266          377          0.706
## 67          0.344          436          641          0.680
## 68          0.285          458          689          0.665
## 69          0.332          398          551          0.722
```

```
## 70          0.387          395          582          0.679
## 71          0.366          314          436          0.720
## 72          0.354          393          525          0.749
## 73          0.332          397          575          0.690
## 74          0.297          332          489          0.679
## 75          0.378          410          564          0.727
## 76          0.365          349          460          0.759
## 77          0.343          562          773          0.727
## 78          0.356          417          581          0.718
## 79          0.361          468          618          0.757
## 80          0.354          440          584          0.753
## 81          0.308          551          803          0.686
## 82          0.322          460          663          0.694
## 83          0.284          435          638          0.682
## 84          0.361          425          634          0.670
## 85          0.370          375          545          0.688
## 86          0.327          486          683          0.712
## 87          0.295          380          604          0.629
## 88          0.350          398          546          0.729
## 89          0.355          307          443          0.693
## 90          0.340          476          660          0.721
## 91          0.350          291          420          0.693
## 92          0.400          509          702          0.725
## 93          0.368          309          503          0.614
## 94          0.324          414          551          0.751
## 95          0.372          393          588          0.668
## 96          0.302          547          855          0.640
## 97          0.332          579          778          0.744
## 98          0.334          435          603          0.721
## 99          0.332          314          462          0.680
## 100         0.381          412          572          0.720
## 101         0.361          364          508          0.717
## 102         0.391          527          741          0.711
## 103         0.330          454          634          0.716
## 104         0.311          411          600          0.685
## 105         0.355          568          771          0.737
## 106         0.323          498          720          0.692
## 107         0.384          462          694          0.666
## 108         0.307          405          588          0.689
## 109         0.322          512          726          0.705
## 110         0.363          602          791          0.761
## 111         0.403          535          781          0.685
## 112         0.331          487          666          0.731
## 113         0.345          575          818          0.703
## 114         0.345          622          790          0.787
## 115         0.381          459          619          0.742
```

```
## 116        0.360        388        538        0.721
## 117        0.354        399        576        0.693
## 118        0.309        409        605        0.676
## 119        0.385        615        767        0.802
## 120        0.354        283        424        0.667
## 121        0.340        535        793        0.675
## 122        0.355        490        697        0.703
## 123        0.365        566        780        0.726
## 124        0.364        366        498        0.735
## 125        0.370        367        507        0.724
## 126        0.354        358        525        0.682
## 127        0.332        387        540        0.717
## 128        0.345        415        591        0.702
## 129        0.343        448        553        0.810
## 130        0.363        454        621        0.731
## 131        0.312        455        695        0.655
## 132        0.351        521        699        0.745
## 133        0.363        450        615        0.732
## 134        0.364        627        849        0.739
## 135        0.379        386        557        0.693
## 136        0.337        491        699        0.702
## 137        0.280        385        607        0.634
## 138        0.315        468        630        0.743
## 139        0.330        394        582        0.677
## 140        0.363        437        607        0.720
## 141        0.334        383        574        0.667
## 142        0.350        487        691        0.705
## 143        0.311        410        590        0.695
## 144        0.337        464        641        0.724
## 145        0.354        635        859        0.739
## 146        0.331        394        512        0.770
## 147        0.380        325        437        0.744
## 148        0.343        565        823        0.687
## 149        0.423        480        620        0.774
## 150        0.369        406        524        0.775
## 151        0.373        607        801        0.758
## 152        0.344        574        831        0.691
## 153        0.341        452        643        0.703
## 154        0.355        468        639        0.732
## 155        0.345        527        710        0.742
## 156        0.396        576        737        0.782
## 157        0.319        603        802        0.752
## 158        0.335        453        690        0.657
## 159        0.342        534        687        0.777
## 160        0.366        363        545        0.666
## 161        0.320        481        645        0.746
```

```
## 162        0.341          438              617        0.710
## 163        0.311          293              452        0.648
## 164        0.331          312              534        0.584
## 165        0.363          363              530        0.685
## 166        0.388          524              692        0.757
## 167        0.342          522              723        0.722
## 168        0.323          395              573        0.689
## 169        0.282          329              489        0.673
## 170        0.349          466              627        0.743
## 171        0.341          460              639        0.720
## 172        0.350          391              574        0.681
## 173        0.307          461              647        0.713
## 174        0.322          596              835        0.714
## 175        0.344          446              648        0.688
## 176        0.333          419              570        0.735
## 177        0.323          444              626        0.709
## 178        0.378          564              749        0.753
## 179        0.342          406              579        0.701
## 180        0.326          373              551        0.677
## 181        0.342          379              508        0.746
## 182        0.317          578              848        0.682
## 183        0.377          464              647        0.717
## 184        0.310          426              647        0.658
## 185        0.358          461              589        0.783
## 186        0.356          413              590        0.700
## 187        0.345          434              627        0.692
## 188        0.363          370              526        0.703
## 189        0.297          530              773        0.686
## 190        0.374          480              651        0.737
## 191        0.376          428              621        0.689
## 192        0.341          461              654        0.705
## 193        0.322          477              669        0.713
## 194        0.308          403              594        0.678
## 195        0.353          493              673        0.733
## 196        0.317          389              544        0.715
## 197        0.392          341              473        0.721
## 198        0.338          484              694        0.697
## 199        0.337          443              647        0.685
## 200        0.347          578              816        0.708
## 201        0.332          235              385        0.610
## 202        0.334          484              714        0.678
## 203        0.346          523              745        0.702
## 204        0.334          471              681        0.692
## 205        0.359          463              629        0.736
## 206        0.367          322              418        0.770
## 207        0.345          528              732        0.721
```

```
## 208          0.366          570          803          0.710
## 209          0.298          417          619          0.674
## 210          0.325          324          437          0.741
## 211          0.350          364          548          0.664
## 212          0.319          460          641          0.718
## 213          0.342          464          666          0.697
## 214          0.352          468          661          0.708
## 215          0.349          493          673          0.733
## 216          0.362          541          728          0.743
## 217          0.365          476          613          0.777
## 218          0.372          335          482          0.695
## 219          0.343          373          539          0.692
## 220          0.339          356          532          0.669
## 221          0.382          476          634          0.751
## 222          0.346          433          614          0.705
## 223          0.362          445          633          0.703
## 224          0.357          397          543          0.731
## 225          0.348          401          537          0.747
## 226          0.363          483          725          0.666
## 227          0.294          411          604          0.680
## 228          0.313          405          551          0.735
## 229          0.315          442          594          0.744
## 230          0.385          492          663          0.742
## 231          0.341          477          650          0.734
## 232          0.295          381          602          0.633
## 233          0.372          378          549          0.689
## 234          0.346          431          618          0.697
## 235          0.349          388          585          0.663
## 236          0.362          340          492          0.691
## 237          0.321          472          636          0.742
## 238          0.351          450          624          0.721
## 239          0.350          520          698          0.745
## 240          0.320          425          613          0.693
## 241          0.351          326          511          0.638
## 242          0.382          529          696          0.760
## 243          0.331          543          779          0.697
## 244          0.309          525          748          0.702
## 245          0.325          448          668          0.671
## 246          0.320          596          875          0.681
## 247          0.381          491          674          0.728
## 248          0.305          371          502          0.739
## 249          0.326          533          771          0.691
## 250          0.374          461          641          0.719
## 251          0.377          468          646          0.724
## 252          0.379          369          515          0.717
## 253          0.280          479          704          0.680
```

```
## 254          0.348            500                729          0.686
## 255          0.347            365                552          0.661
## 256          0.341            465                755          0.616
## 257          0.355            446                638          0.699
## 258          0.312            372                584          0.637
## 259          0.332            410                592          0.693
## 260          0.357            545                722          0.755
## 261          0.351            493                698          0.706
## 262          0.322            462                615          0.751
## 263          0.304            484                809          0.598
## 264          0.378            412                555          0.742
## 265          0.325            414                570          0.726
## 266          0.367            390                518          0.753
## 267          0.370            487                666          0.731
## 268          0.362            486                679          0.716
## 269          0.351            479                641          0.747
## 270          0.355            372                568          0.655
## 271          0.328            406                620          0.655
## 272          0.360            406                567          0.716
## 273          0.293            359                531          0.676
## 274          0.371            478                668          0.716
## 275          0.324            492                697          0.706
## 276          0.339            299                440          0.680
## 277          0.375            459                669          0.686
## 278          0.332            573                772          0.742
## 279          0.329            350                492          0.711
## 280          0.366            473                694          0.682
## 281          0.408            494                635          0.778
## 282          0.341            433                585          0.740
## 283          0.332            658               1018          0.646
## 284          0.370            358                519          0.690
## 285          0.338            459                640          0.717
## 286          0.382            363                563          0.645
## 287          0.302            476                677          0.703
## 288          0.374            377                550          0.685
## 289          0.341            374                523          0.715
## 290          0.386            321                489          0.656
## 291          0.340            539                757          0.712
## 292          0.331            393                588          0.668
## 293          0.328            400                539          0.742
## 294          0.335            393                580          0.678
## 295          0.357            446                620          0.719
## 296          0.317            432                643          0.672
## 297          0.342            414                607          0.682
## 298          0.314            348                503          0.692
## 299          0.326            501                695          0.721
```

```
## 300        0.333        480        701        0.685
## 301        0.330        472        649        0.727
## 302        0.344        398        528        0.754
## 303        0.335        440        628        0.701
## 304        0.344        407        645        0.631
## 305        0.367        561        744        0.754
## 306        0.325        376        541        0.695
## 307        0.308        437        631        0.693
## 308        0.303        488        659        0.741
## 309        0.346        469        688        0.682
## 310        0.321        382        578        0.661
## 311        0.319        641        923        0.694
## 312        0.344        395        536        0.737
## 313        0.324        660        988        0.668
## 314        0.328        474        694        0.683
## 315        0.365        508        694        0.732
## 316        0.348        445        637        0.699
## 317        0.378        410        531        0.772
## 318        0.327        370        532        0.695
## 319        0.335        397        550        0.722
## 320        0.318        403        585        0.689
## 321        0.340        522        751        0.695
## 322        0.355        486        768        0.633
## 323        0.355        559        747        0.748
## 324        0.382        554        764        0.725
## 325        0.370        435        616        0.706
## 326        0.316        416        595        0.699
## 327        0.311        478        710        0.673
## 328        0.359        507        678        0.748
## 329        0.352        506        695        0.728
## 330        0.305        469        669        0.701
## 331        0.359        424        581        0.730
## 332        0.394        461        606        0.761
## 333        0.395        445        598        0.744
## 334        0.305        428        575        0.744
## 335        0.308        494        681        0.725
## 336        0.364        426        564        0.755
## 337        0.350        472        679        0.695
## 338        0.364        528        708        0.746
## 339        0.316        584        849        0.688
## 340        0.370        424        628        0.675
## 341        0.385        321        440        0.730
## 342        0.327        524        724        0.724
## 343        0.315        470        680        0.691
## 344        0.310        475        655        0.725
## 345        0.346        385        576        0.668
```

```
## 346          0.373            417                565        0.738
## 347          0.359            344                531        0.648
## 348          0.414            406                577        0.704
## 349          0.344            510                692        0.737
## 350          0.344            477                660        0.723
## 351          0.331            437                644        0.679
## 352          0.363            411                557        0.738
## 353          0.340            354                505        0.701
##     OffensiveRebounds TotalRebounds Assists Steals Blocks Turnovers
## 1                 325          1110     525    297     93       407
## 2                 253          1077     434    154     57       423
## 3                 312          1204     399    185    106       388
## 4                 314          1032     385    234     50       487
## 5                 367          1279     401    218     82       399
## 6                 365          1094     313    203    102       451
## 7                 384          1285     418    157    160       465
## 8                 294          1081     402    195     78       454
## 9                 334          1079     391    229    107       492
## 10                251          1014     402    221    131       386
## 11                310          1127     406    171     90       418
## 12                399          1351     459    213    109       466
## 13                324          1115     398    165     73       370
## 14                244          1060     443    169     87       503
## 15                317          1063     365    216    109       478
## 16                403          1211     363    183    139       448
## 17                345          1152     543    276    172       450
## 18                259          1125     499    198     50       376
## 19                457          1369     572    369    190       466
## 20                397          1215     454    252    110       392
## 21                308          1243     426    217    118       480
## 22                450          1281     473    209    159       446
## 23                286          1275     645    220    125       376
## 24                415          1274     444    234    108       501
## 25                244          1060     324    183    152       409
## 26                242          1016     398    200     68       379
## 27                331          1139     382    159    103       378
## 28                311          1091     447    199     73       425
## 29                408          1396     427    227     90       412
## 30                313          1195     423    186    132       440
## 31                263          1118     482    203    107       354
## 32                261          1157     418    241    123       458
## 33                285           969     319    158     79       383
## 34                286          1154     519    173    116       409
## 35                452          1470     599    263    140       433
## 36                288          1058     434    193     62       359
## 37                284           975     301    147     99       334
```

```
## 38          486        1272    379    218    105      439
## 39          277        1224    407    200    118      450
## 40          352        1214    512    225    154      410
## 41          306        1171    345    148     85      379
## 42          218         929    375    196     71      412
## 43          429        1471    479    210    153      441
## 44          286        1051    425    164     60      425
## 45          379        1204    431    159    106      355
## 46          249         900    338    227     99      352
## 47          266        1050    482    218    104      337
## 48          300        1026    466    204    108      412
## 49          299        1164    463    202    120      478
## 50          332        1107    353    172    110      427
## 51          315        1215    439    189    150      393
## 52          419        1321    467    267    107      425
## 53          407        1277    487    263    110      441
## 54          199         892    288    138    101      418
## 55          278        1095    418    139     79      409
## 56          252         995    376    193     72      512
## 57          442        1264    466    214    155      365
## 58          322        1107    462    201    110      386
## 59          309        1204    410    223    154      450
## 60          303        1093    460    174     97      412
## 61          376        1256    451    214     78      472
## 62          322        1240    541    213    121      456
## 63          260        1060    387    216     76      347
## 64          256        1080    472    199     76      372
## 65          363        1352    478    181    107      479
## 66          255         985    443    218     78      366
## 67          380        1167    422    200    145      427
## 68          314        1218    326    169    115      555
## 69          191        1015    453    181    107      414
## 70          288        1196    561    237     92      464
## 71          246        1020    426    158    100      361
## 72          264        1185    499    175     82      378
## 73          261        1145    541    164     81      414
## 74          378        1144    333    214     66      435
## 75          254        1083    425    149     78      378
## 76          275        1041    354    142     78      408
## 77          396        1347    513    227    117      488
## 78          336        1019    319    238     77      355
## 79          279        1214    518    180    130      449
## 80          312        1126    464    144     62      377
## 81          495        1567    606    346    257      488
## 82          379        1133    442    250    151      451
## 83          317        1061    382    224    110      416
```

```
## 84          442        1351    534    252    135     483
## 85          347        1147    434    200    120     427
## 86          362        1167    458    317    101     447
## 87          400        1155    369    237    143     442
## 88          299        1192    467    200    101     401
## 89          265        1082    468    135     62     413
## 90          215        1095    394    185     89     416
## 91          273         968    401    202     63     436
## 92          323        1155    486    261    132     477
## 93          288         991    355    226    111     499
## 94          366        1296    419    184     93     485
## 95          272        1064    449    219    133     479
## 96          367        1217    491    359    153     477
## 97          418        1391    475    266    161     492
## 98          375        1201    437    257    131     420
## 99          305        1082    360    188    108     345
## 100         300        1137    457    228    111     400
## 101         326        1171    524    288    137     407
## 102         271        1190    502    238    101     407
## 103         323        1166    399    204     97     428
## 104         274        1116    375    170     96     413
## 105         350        1299    542    198    131     443
## 106         320        1190    403    255    132     451
## 107         273        1110    415    269    158     391
## 108         262        1074    433    221    166     459
## 109         357        1215    423    185    163     506
## 110         354        1443    668    278    204     394
## 111         307        1287    420    193    147     536
## 112         350        1238    478    207     75     411
## 113         376        1406    578    291    154     500
## 114         363        1370    487    215    121     379
## 115         278        1050    462    228     69     401
## 116         288        1119    369    174    123     491
## 117         278        1054    478    156     66     368
## 118         363        1183    404    170     91     406
## 119         305        1170    492    228    122     342
## 120         220         916    513    241    127     350
## 121         409        1142    449    234     54     442
## 122         448        1502    548    240    161     410
## 123         396        1253    463    198    144     427
## 124         275         975    383    154     88     360
## 125         258        1061    399    130     53     433
## 126         271        1106    478    210    124     457
## 127         298        1139    421    162    106     417
## 128         351        1091    446    245     86     435
## 129         204         845    354    184     53     504
```

```
## 130          250          1007     342     207     73      402
## 131          341          1272     469     226     154     433
## 132          283          1152     432     223     90      406
## 133          327          1230     527     247     160     379
## 134          357          1246     542     211     112     427
## 135          271          1140     525     247     158     395
## 136          403          1233     471     228     103     462
## 137          331          1125     359     209     124     436
## 138          400          1310     451     245     138     440
## 139          330          1163     457     213     166     463
## 140          345          1123     403     211     101     420
## 141          325          1137     464     256     76      385
## 142          374          1376     478     243     139     472
## 143          372          1163     362     185     106     437
## 144          412          1185     430     226     119     378
## 145          427          1428     501     220     176     466
## 146          328          1060     391     230     98      435
## 147          233          1004     419     158     126     402
## 148          446          1282     481     283     97      473
## 149          232          1105     471     171     71      418
## 150          270          1174     532     223     96      395
## 151          329          1378     654     268     108     489
## 152          377          1263     399     219     105     479
## 153          321          1196     430     218     157     471
## 154          284          1194     401     210     71      477
## 155          364          1142     487     226     125     423
## 156          317          1176     498     206     78      398
## 157          460          1355     452     308     148     452
## 158          368          1204     440     190     115     386
## 159          345          1297     458     149     99      412
## 160          190          1013     468     220     78      407
## 161          314          1098     460     250     102     425
## 162          292          997      448     278     73      444
## 163          280          1015     471     239     94      460
## 164          326          1026     360     221     78      487
## 165          241          967      377     172     53      387
## 166          331          1292     456     168     145     467
## 167          319          1260     527     334     170     453
## 168          296          1145     453     268     83      428
## 169          264          989      361     186     70      431
## 170          378          1336     447     148     163     437
## 171          293          1115     500     181     85      447
## 172          342          1123     468     154     93      445
## 173          310          1054     429     147     97      450
## 174          436          1358     553     288     144     538
## 175          307          1057     393     207     80      407
```

```
## 176          291        1061    413    212    100      372
## 177          353        1190    349    194     87      413
## 178          414        1579    715    201    205      493
## 179          299        1307    512    225    149      334
## 180          382        1228    352    190     85      512
## 181          288        1094    356    145     94      382
## 182          402        1314    523    175    140      418
## 183          393        1215    481    274    172      451
## 184          378        1165    361    176     70      415
## 185          322        1128    475    235    118      416
## 186          261         966    351    233     90      391
## 187          270        1000    354    221    127      385
## 188          345        1130    354    149     53      448
## 189          360        1224    379    237    101      502
## 190          292        1083    491    169     46      394
## 191          293        1176    510    229    106      416
## 192          357        1177    440    186    137      402
## 193          397        1078    362    230     48      401
## 194          293        1036    354    146    107      410
## 195          351        1247    591    249    149      401
## 196          359        1160    399    167     58      437
## 197          273        1047    409    180     61      304
## 198          379        1278    466    260    151      345
## 199          408        1204    401    184    115      401
## 200          325        1274    498    211    134      352
## 201          260         986    325    116     53      356
## 202          445        1343    514    197     91      419
## 203          351        1199    446    220    107      455
## 204          393        1166    497    285    134      517
## 205          315        1143    343    128    127      390
## 206          280        1041    428    208    128      386
## 207          250        1178    376    223     94      421
## 208          366        1318    457    219    154      523
## 209          351        1170    334    209     74      435
## 210          298         931    331    196     86      456
## 211          291        1059    452    215    108      417
## 212          399        1265    526    207     93      515
## 213          399        1251    453    315    140      423
## 214          486        1390    539    257    138      449
## 215          387        1221    495    150     68      436
## 216          477        1589    678    252    120      473
## 217          229        1105    402    160     87      368
## 218          233        1009    398    171     87      380
## 219          363        1252    520    212    165      536
## 220          368        1244    428    210     88      443
## 221          237        1075    480    211     74      381
```

```
## 222          290          1055    381    196     65        372
## 223          309          1128    422    176    101        398
## 224          304          1179    374    188     75        404
## 225          262          1115    370    165     75        393
## 226          364          1298    592    218    139        441
## 227          352          1103    340    192    138        496
## 228          278          1086    451    176    134        349
## 229          370          1208    423    179    148        306
## 230          275          1063    580    191     99        426
## 231          328          1215    494    206     85        438
## 232          332          1152    451    200    115        448
## 233          295          1093    422    183    139        386
## 234          305          1263    430    203    102        401
## 235          418          1365    439    197    160        408
## 236          335          1147    434    152    107        447
## 237          310          1091    465    185    164        384
## 238          361          1307    501    290    162        442
## 239          285          1039    325    199     46        423
## 240          382          1174    389    245    122        399
## 241          270          1101    465    210     76        395
## 242          290          1085    479    229     95        432
## 243          383          1211    387    227    126        448
## 244          491          1282    434    264    130        449
## 245          233          1032    369    175    123        425
## 246          392          1151    445    309     55        445
## 247          324          1248    565    214     69        358
## 248          272          1016    295    168     72        354
## 249          399          1241    491    253    136        445
## 250          421          1341    518    220    143        381
## 251          311          1111    393    159     62        415
## 252          335          1170    467    196    101        358
## 253          375          1194    401    235    129        405
## 254          316          1221    425    171    104        438
## 255          196           957    519    249    101        336
## 256          363          1144    450    282    101        415
## 257          373          1178    487    257     92        507
## 258          391          1210    390    201    124        408
## 259          308          1050    431    239    104        404
## 260          354          1231    471    170    133        465
## 261          430          1246    413    221     94        402
## 262          258          1145    397    186    110        305
## 263          492          1428    460    254    145        464
## 264          329          1180    341    201     87        355
## 265          293          1032    301    172    177        488
## 266          369          1191    555    242     94        413
## 267          316          1225    461    238    127        463
```

```
## 268          346          1228    487    197    113    424
## 269          290          1238    516    204    112    466
## 270          340          1138    439    188    105    321
## 271          323          1130    423    131     52    484
## 272          242          1026    416    146    111    433
## 273          383          1188    462    234     96    478
## 274          336          1221    386    141    118    403
## 275          361          1213    459    235    132    423
## 276          248           973    415    190     93    321
## 277          288          1094    471    201    141    456
## 278          376          1137    386    167     60    493
## 279          305          1051    434    207     94    440
## 280          383          1160    413    202    140    433
## 281          251          1259    489    202     65    388
## 282          219           942    382    165     68    324
## 283          496          1483    521    306    124    604
## 284          284          1081    402    182     75    447
## 285          346          1117    452    215     80    536
## 286          324          1188    536    202    117    392
## 287          351          1079    382    198     70    438
## 288          289          1081    438    212    129    447
## 289          365          1135    441    189    123    350
## 290          274          1178    553    253    110    345
## 291          282          1250    426    199    114    466
## 292          313           980    372    216     86    511
## 293          333          1183    417    199    164    421
## 294          346          1191    387    194    136    413
## 295          264          1120    472    302    111    357
## 296          322          1147    392    186    147    467
## 297          313          1008    348    192    113    442
## 298          359          1110    418    210     94    394
## 299          410          1348    372    209    142    478
## 300          363          1174    408    278    162    423
## 301          316          1136    476    284     73    368
## 302          351          1062    441    223     88    418
## 303          317          1032    342    219     78    497
## 304          330          1122    389    226    134    517
## 305          379          1391    660    221    199    412
## 306          349          1158    423    197     90    456
## 307          372          1195    376    215    142    444
## 308          372          1235    421    211     97    467
## 309          395          1347    598    255    162    499
## 310          285          1070    307    135     84    466
## 311          358          1215    503    345    105    471
## 312          323          1216    420    213     82    402
## 313          467          1503    591    309    114    589
```

```
## 314                409          1291     493     234      84        455
## 315                315          1297     518     278     186        457
## 316                358          1264     486     227     152        401
## 317                327          1281     538     170     152        402
## 318                389          1184     319     154      94        390
## 319                277          1021     392     171     135        408
## 320                298          1131     444     147     109        467
## 321                239          1119     430     176      76        404
## 322                380          1361     472     194     136        467
## 323                377          1405     598     215     147        449
## 324                318          1271     484     201      89        487
## 325                317          1096     440     144      80        409
## 326                294          1115     431     202     101        439
## 327                325          1139     371     154     126        454
## 328                304          1194     399     188     136        377
## 329                368          1253     503     194     106        388
## 330                358          1208     450     263     148        458
## 331                280          1110     442     217      61        390
## 332                310          1142     531     236      78        392
## 333                342          1326     544     211     149        342
## 334                378          1095     351     184     108        414
## 335                399          1174     329     150      77        417
## 336                276          1076     478     169      89        440
## 337                334          1129     419     323     206        479
## 338                220          1177     383     179     140        382
## 339                519          1425     488     223     126        557
## 340                303          1135     389     135      53        577
## 341                254          1100     319     165     132        353
## 342                338          1256     417     235     174        466
## 343                345          1197     374     147      78        490
## 344                452          1417     499     174     135        446
## 345                251          1057     529     158     146        390
## 346                330          1232     460     134      83        456
## 347                283          1197     430     177     140        327
## 348                365          1232     528     233     105        377
## 349                382          1229     484     214      72        402
## 350                167           983     331     176      88        450
## 351                371          1281     519     190     128        450
## 352                259          1157     503     177     131        392
## 353                418          1207     449     210     115        423
##     PersonalFouls
## 1             635
## 2             543
## 3             569
## 4             587
## 5             578
```

```
## 6              565
## 7              572
## 8              566
## 9              548
## 10             520
## 11             585
## 12             675
## 13             594
## 14             622
## 15             629
## 16             684
## 17             693
## 18             544
## 19             731
## 20             596
## 21             579
## 22             636
## 23             509
## 24             610
## 25             474
## 26             570
## 27             509
## 28             578
## 29             596
## 30             617
## 31             625
## 32             600
## 33             575
## 34             627
## 35             659
## 36             593
## 37             544
## 38             747
## 39             639
## 40             641
## 41             623
## 42             596
## 43             673
## 44             562
## 45             558
## 46             574
## 47             522
## 48             627
## 49             611
## 50             597
## 51             554
```

```
## 52              593
## 53              571
## 54              490
## 55              553
## 56              625
## 57              563
## 58              528
## 59              552
## 60              551
## 61              691
## 62              547
## 63              486
## 64              522
## 65              616
## 66              539
## 67              643
## 68              620
## 69              492
## 70              580
## 71              468
## 72              539
## 73              463
## 74              589
## 75              523
## 76              506
## 77              630
## 78              528
## 79              568
## 80              542
## 81              595
## 82              580
## 83              622
## 84              603
## 85              569
## 86              661
## 87              529
## 88              632
## 89              539
## 90              602
## 91              612
## 92              578
## 93              580
## 94              575
## 95              514
## 96              655
## 97              705
```

```
## 98              614
## 99              559
## 100             592
## 101             481
## 102             561
## 103             563
## 104             526
## 105             581
## 106             590
## 107             583
## 108             560
## 109             565
## 110             603
## 111             699
## 112             598
## 113             730
## 114             649
## 115             556
## 116             554
## 117             556
## 118             502
## 119             526
## 120             482
## 121             593
## 122             705
## 123             652
## 124             497
## 125             582
## 126             684
## 127             577
## 128             700
## 129             592
## 130             586
## 131             596
## 132             564
## 133             517
## 134             565
## 135             495
## 136             578
## 137             643
## 138             553
## 139             523
## 140             612
## 141             550
## 142             610
## 143             525
```

```
## 144            601
## 145            600
## 146            630
## 147            515
## 148            622
## 149            571
## 150            543
## 151            633
## 152            620
## 153            610
## 154            624
## 155            612
## 156            545
## 157            634
## 158            535
## 159            591
## 160            471
## 161            602
## 162            573
## 163            566
## 164            672
## 165            583
## 166            635
## 167            664
## 168            578
## 169            575
## 170            526
## 171            589
## 172            643
## 173            467
## 174            739
## 175            557
## 176            465
## 177            581
## 178            647
## 179            514
## 180            583
## 181            579
## 182            578
## 183            586
## 184            570
## 185            603
## 186            647
## 187            540
## 188            625
## 189            698
```

```
## 190          567
## 191          655
## 192          561
## 193          614
## 194          576
## 195          527
## 196          534
## 197          486
## 198          580
## 199          556
## 200          584
## 201          533
## 202          632
## 203          627
## 204          631
## 205          511
## 206          587
## 207          582
## 208          763
## 209          614
## 210          533
## 211          514
## 212          610
## 213          631
## 214          691
## 215          585
## 216          611
## 217          523
## 218          485
## 219          573
## 220          589
## 221          539
## 222          537
## 223          538
## 224          653
## 225          554
## 226          664
## 227          578
## 228          583
## 229          455
## 230          534
## 231          623
## 232          504
## 233          516
## 234          527
## 235          580
```

```
## 236          575
## 237          536
## 238          665
## 239          630
## 240          597
## 241          500
## 242          639
## 243          611
## 244          681
## 245          588
## 246          758
## 247          617
## 248          459
## 249          600
## 250          625
## 251          524
## 252          528
## 253          582
## 254          568
## 255          488
## 256          591
## 257          683
## 258          566
## 259          566
## 260          572
## 261          598
## 262          497
## 263          634
## 264          572
## 265          614
## 266          659
## 267          560
## 268          638
## 269          667
## 270          543
## 271          539
## 272          597
## 273          564
## 274          657
## 275          638
## 276          462
## 277          649
## 278          670
## 279          621
## 280          623
## 281          557
```

```
## 282          527
## 283          790
## 284          615
## 285          602
## 286          590
## 287          612
## 288          474
## 289          515
## 290          507
## 291          696
## 292          716
## 293          553
## 294          635
## 295          650
## 296          586
## 297          579
## 298          536
## 299          631
## 300          588
## 301          581
## 302          490
## 303          735
## 304          554
## 305          654
## 306          610
## 307          521
## 308          630
## 309          586
## 310          551
## 311          763
## 312          581
## 313          679
## 314          667
## 315          663
## 316          627
## 317          517
## 318          561
## 319          523
## 320          535
## 321          529
## 322          563
## 323          649
## 324          612
## 325          548
## 326          546
## 327          568
```

```
## 328            568
## 329            580
## 330            650
## 331            541
## 332            532
## 333            542
## 334            639
## 335            555
## 336            554
## 337            653
## 338            609
## 339            707
## 340            671
## 341            487
## 342            528
## 343            564
## 344            705
## 345            516
## 346            535
## 347            511
## 348            589
## 349            545
## 350            588
## 351            550
## 352            510
## 353            656
```

And just like that, we have a method for getting up to the minute season stats for every team in Division I.

Chapter 26

Advanced rvest

With the chapter, we learned how to grab one table from one page. But what if you needed more than that? What if you needed hundreds of tables from hundreds of pages? What if you needed to combine one table on one page into a bigger table, but hundreds of times. There's a way to do this, it just takes patience, a lot of logic, a lot of debugging and, for me, a fair bit of swearing.

So what we are after are game by game stats for each college basketball team in America.

We can see from this page that each team is linked. If we follow each link, we get a ton of tables. But they aren't what we need. There's a link to gamelogs underneath the team names.

So we can see from this that we've got some problems.

1. The team name isn't in the table. Nor is the conference.
2. There's a date we'll have to deal with.
3. Non-standard headers and a truly huge number of fields.
4. And how do we get each one of those urls without having to copy them all into some horrible list?

So let's start with that last question first and grab libraries we need.

```
library(tidyverse)
library(rvest)
library(lubridate)
```

First things first, we need to grab the url to each team from that first link.

```
url <- "https://www.sports-reference.com/cbb/seasons/2019-school-stats.html"
```

But notice first, we don't want to grab the table. The table doesn't help us. We need to grab the only *link* in the table. So we can do that by using the table

xpath node, then grabbing the anchor tags in the table, then get only the link out of them (instead of the linked text).

```
schools <- url %>%
  read_html() %>%
  html_nodes(xpath = '//*[@id="basic_school_stats"]') %>%
  html_nodes("a") %>%
  html_attr('href')
```

Notice we now have a list called schools with … 353 elements. That's the number of teams in college basketball, so we're off to a good start. Here's what the fourth element is.

```
schools[4]
```

```
## [1] "/cbb/schools/alabama-am/2019.html"
```

So note, that's the relative path to Alabama A&M's team page. By relative path, I mean it doesn't have the root domain. So we need to add that to each request or we'll get no where.

So that's a problem to note.

Before we solve that, let's just make sure we can get one page and get what we need.

We'll scrape Abilene Christian.

To merge all this into one big table, we need to grab the team name and their conference and merge it into the table. But those values come from somewhere else. The scraping works just about the same, but instead of html_table you use html_text.

So the first part of this is reading the html of the page so we don't do that for each element – we just do it once so as to not overwhelm their servers.

The second part is we're grabbing the team name based on it's location in the page.

Third: The conference.

Fourth is the table itself, noting to ignore the headers. The last bit fixes the headers, removes the garbage header data from the table, converts the data to numbers, fixes the date and mutates a team and conference value. It looks like a lot, and it took a bit of twiddling to get it done, but it's no different from what you did for your last homework.

```
page <- read_html("https://www.sports-reference.com/cbb/schools/abilene-christian/2019-

team <- page %>%
  html_nodes(xpath = '//*[@id="meta"]/div[2]/h1/span[2]') %>%
  html_text()
```

```
conference <- page %>%
    html_nodes(xpath = '//*[@id="meta"]/div[2]/p[1]/a') %>%
    html_text()

table <- page %>%
  html_nodes(xpath = '//*[@id="sgl-basic"]') %>%
  html_table(header=FALSE)

table <- table[[1]] %>% rename(Game=X1, Date=X2, HomeAway=X3, Opponent=X4, W_L=X5, TeamScore=X6,
```

Now what we're left with is how do we do this for ALL the teams. We need to send 353 requests to their servers to get each page. And each url is not the one we have – we need to alter it.

First we have to add the root domain to each request. And, each request needs to go to /2019-gamelogs.html instead of /2019.html. If you look at the urls two the page we have and the page we need, that's all that changes.

What we're going to use is what is known in programming as a loop. We can loop through a list and have it do something to each element in the loop. And once it's done, we can move on to the next thing.

Think of it like a program that will go though a list of your classmates and ask each one of them for their year in school. It will start at one end of the list and move through asking each one "What year in school are you?" and will move on after getting an answer.

Except we want to take a url, add something to it, alter it, then request it and grab a bunch of data from it. Once we're done doing all that, we'll take all that info and cram it into a bigger dataset and then move on to the next one. Here's what that looks like:

```
uri <- "https://www.sports-reference.com"

logs <- tibble()

for (i in schools){
  log_url <- gsub("/2019.html","/2019-gamelogs.html", i)
  school_url <- paste(uri, log_url, sep="")  # creating the url to fetch

  page <- read_html(school_url)

  team <- page %>%
    html_nodes(xpath = '//*[@id="meta"]/div[2]/h1/span[2]') %>%
    html_text()

  conference <- page %>%
```

```
    html_nodes(xpath = '//*[@id="meta"]/div[2]/p[1]/a') %>%
    html_text()

  table <- page %>%
    html_nodes(xpath = '//*[@id="sgl-basic"]') %>%
    html_table(header=FALSE)

table <- table[[1]] %>% rename(Game=X1, Date=X2, HomeAway=X3, Opponent=X4, W_L=X5, Tea

  logs <- rbind(logs, table)  # binding them all together
  Sys.sleep(5)  # Sys.sleep(3) pauses the loop for 3s so as not to overwhelm website's
}
```

The magic here is in `for (i in schools){`. What that says is for each iterator in schools – for each school in schools – do what follows each time. So we take the code we wrote for one school and use it for every school.

This part:

```
  log_url <- gsub("/2019.html","/2019-gamelogs.html", i)
  school_url <- paste(uri, log_url, sep="")  # creating the url to fetch

  page <- read_html(school_url)
```

`log_url` is what changes our school page url to our logs url, and `school_url` is taking that log url and the root domain and merging them together to create the complete url. Then, page just reads that url we created.

What follows that is taken straight from our example of just doing one.

The last bits are using rbind to take our data and mash it into a bigger table, over and over and over again until we have them all in a single table. Then, we tell our scraper to wait a few seconds because we don't want our script to machine gun requests at their server as fast as it can go. That's a guaranteed way to get them to block scrapers, and could knock them off the internet. Aggressive scrapers aren't cool. Don't do it.

Lastly, we write it out to a csv file.

```
write.csv(logs, "logs.csv")
```

So with a little programming knowhow, a little bit of problem solving and the stubbornness not to quit on it, you can get a whole lot of data scattered all over the place with not a lot of code.

26.1 One last bit

Most tables that Sports Reference sites have are in plain vanilla HTML. But some of them – particularly player based stuff – are hidden with a little trick.

The site puts the data in a comment – where a browser will ignore it – and then uses javascript to interpret the commented data. To a human on the page, it looks the same. To a browswer or a scraper, it's invisible. You'll get errors. How do you get around it?

1. Scrape the comments.
2. Turn the comment into text.
3. Then read that text as html.
4. Proceed as normal.

```
h <- read_html('https://www.baseball-reference.com/leagues/MLB/2017-standard-pitching.shtml')

df <- h %>% html_nodes(xpath = '//comment()') %>%    # select comment nodes
    html_text() %>%    # extract comment text
    paste(collapse = '') %>%    # collapse to a single string
    read_html() %>%    # reparse to HTML
    html_node('table') %>%    # select the desired table
    html_table()
```

Chapter 27

Annotations

Some of the best sports data visualizations start with a provocative question. At a college just under three hours from Kansas City, my classes are lousy with Chiefs fans. So the first day of classes in the spring of 2019, I asked them: Are the Chief's Screwed in the Playoffs? The answer ultimately was yes, and how I was able to make that argument before a playoff game had even been played is a good example of how labeling and annotations can make a chart much better.

Going to add a new library to the mix called `ggrepel`. You'll need to install it in the console with `install.packages("ggrepel")`.

```
library(tidyverse)
library(ggrepel)
```

Now we'll grab the data and join that data together using the Team name as the common element.

```
offense <- read_csv("data/nfloffense.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
defense <- read_csv("data/nfldefense.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

```
total <- offense %>% left_join(defense, by="Team")

head(total)
```

```
## # A tibble: 6 x 52
##   Team      G PointsFor OffYards OffPlays OffYardsPerPlay OffensiveTurnov~
##   <chr> <dbl>     <dbl>    <dbl>    <dbl>           <dbl>            <dbl>
## 1 Kans~    16       565     6810      996             6.8               18
## 2 Los ~    16       527     6738     1060             6.4               19
## 3 New ~    16       504     6067     1010             6                 16
## 4 New ~    16       436     6295     1073             5.9               18
## 5 Indi~    16       433     6179     1070             5.8               24
## 6 Pitt~    16       428     6453     1058             6.1               26
## # ... with 45 more variables: FumblesLost <dbl>, OffFirstDowns <dbl>,
## #   OffPassingComp <dbl>, OffPassingAtt <dbl>, OffPassingYards <dbl>,
## #   OffensivePassingTD <dbl>, OffPassingINT <dbl>,
## #   OffensivePassingYardsPerAtt <dbl>, OffensivePassingFirstDowns <dbl>,
## #   OffRushingAtt <dbl>, OffRushingYards <dbl>, OffRushingTD <dbl>,
## #   RushingYardsPerAtt <dbl>, RushingFirstDowns <dbl>,
## #   OffensivePenalties <dbl>, OffPenaltyYards <dbl>,
## #   OffFirstFromPenalties <dbl>, OffScoringPct <dbl>,
## #   OffensiveTurnoverPct <dbl>, OffensiveExpectedPoints <dbl>,
## #   PointsAllowed <dbl>, YdsAllowed <dbl>, PlaysFaced <dbl>,
## #   DefYardPerPlay <dbl>, Takeaways <dbl>, DefFumblesLost <dbl>,
## #   FirstDownsAllowed <dbl>, PassingCompsAllowed <dbl>, PassingAttFaced <dbl>,
## #   PassingYdsAllowed <dbl>, PassingTDAllowed <dbl>, DefPassingINT <dbl>,
## #   PassingYardsPerPlayAllowed <dbl>, PassingFirstDownsAllowed <dbl>,
## #   RushingAttFaced <dbl>, RushingYdsAllowed <dbl>, RushingTDAllowed <dbl>,
## #   RushingYardsPerAttAllowed <dbl>, RushingFirstDownsAllowed <dbl>,
## #   DefPenalties <dbl>, DefPenaltyYards <dbl>, DefFirstDownByPenalties <dbl>,
## #   OffensiveScoringPctAllowed <dbl>, DefTurnoverPercentage <dbl>,
## #   DefExpectedPoints <dbl>
```

I'm going to set up a point chart that places team on two-axes – yards per play on offense on the x axis, and yards per play on defense.

To build the annotations, I want the league average for offensive yards per play and defensive yards per play. We're going to use those as a proxy for quality. If your team averages more yards per play on offense, that's good. If they average fewer yards per play on defense, that too is good. So that sets up a situation where we have four corners, anchored by good at both and bad at both. The averages will create lines to divide those four corners up.

```
league_averages <- total %>% summarise(AvgOffYardsPer = mean(OffYardsPerPlay), AvgDefYa

league_averages
```

```
## # A tibble: 1 x 2
##   AvgOffYardsPer AvgDefYardsPer
##            <dbl>          <dbl>
## 1           5.59           5.59
```

I also want to highlight playoff teams and, of course, the Chiefs, since that was my question. Are they screwed. First, we filter them from our total list.

```
playoff_teams <- c("Kansas City Chiefs", "New England Patriots", "Los Angeles Chargers", "Indiana

playoffs <- total %>% filter(Team %in% playoff_teams)

chiefs <- total %>% filter(Team == "Kansas City Chiefs")
```

Now we create the plot. We have three geom_points, starting with everyone, then playoff teams, then the Chiefs. I alter the colors on each to separate them. Next, I add a geom_hline to add the horizontal line of my defensive average and a geom_vline for my offensive average. Next, I want to add some text annotations, labeling two corners of my chart (the other two, in my opinion, become obvious). Then, I want to label all the playoff teams. I use `geom_text_repel` to do that – it's using the ggrepel library to push the text away from the dots, respective of other labels and other dots. It means you don't have to move them around so you can read them, or so they don't cover up the dots.

The rest is just adding labels and messing with the theme.

```
ggplot() +
  geom_point(data=total, aes(x=OffYardsPerPlay, y=DefYardPerPlay), color="light grey") +
  geom_point(data=playoffs, aes(x=OffYardsPerPlay, y=DefYardPerPlay)) +
  geom_point(data=chiefs, aes(x=OffYardsPerPlay, y=DefYardPerPlay), color="red") +
  geom_hline(yintercept=5.59375, color="dark grey") +
  geom_vline(xintercept=5.590625, color="dark grey") +
  geom_text(aes(x=6.2, y=5, label="Good Offense, Good Defense"), color="light blue") +
  geom_text(aes(x=5, y=6, label="Bad Defense, Bad Offense"), color="light blue") +
  geom_text_repel(data=playoffs, aes(x=OffYardsPerPlay, y=DefYardPerPlay, label=Team)) +
  labs(x="Offensive Yards Per Play", y="Defensive Points Per Play", title="Are the Chiefs screwed
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.text = element_text(size = 7),
    axis.ticks = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major.x = element_blank()
  )
```

Are the Chiefs screwed in the playoffs?
Their offense is great. Their defense? Not so much
Los Angeles Rams
Kansas City Chiefs
Philadelphia Eagles
New Orleans Saints
New England Patriots
Indianapolis Colts
Dallas Cowboys
Los Angeles Chargers
Chicago Bears
Defensive Points Per Play
6.0
5.6
5.2
4.8
5
6
Offensive Yards Per Play
Source: Sports−Reference.com | By Matt Waite

Chapter 28

Finishing touches, part 1

The output from ggplot is good, but not great. We need to add some pieces to it. The elements of a good graphic are:

- Headline
- Chatter
- The main body
- Annotations
- Labels
- Source line
- Credit line

That looks like:

Headline
Very Bullish on Growth
Chatter
Both private-sector and other government forecasters have
Trump administration.
Gross domestic product in 2017 dollars
La
$26 trillion
24
22
20
18
16
2010
2012
2014
2016
2018
2020
2022
Source
2010 to 2017 data as reported by the Bureau of Economic A
forward based on Times calculations of projections as summ
forecasters = Blue Chip Economic Indicators; Federal Reserv
policymakers
Credit
By The New York Times

28.1 Graphics vs visual stories

While the elements above are nearly required in every chart, they aren't when you are making visual stories.

- When you have a visual story, things like credit lines can become a byline.
- In visual stories, source lines are often a note at the end of the story.
- Graphics don't always get headlines – sometimes just labels, letting the visual story headline carry the load.

An example from The Upshot. Note how the charts don't have headlines, source or credit lines.

28.2 Getting ggplot closer to output

Let's explore fixing up ggplot's output before we send it to a finishing program like Adobe Illustrator. We'll need a graphic to work with first.

```
library(tidyverse)
library(ggrepel)
```

```
scoring <- read_csv("data/scoringoffense.csv")
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   G = col_double(),
##   TD = col_double(),
##   FG = col_double(),
##   `1XP` = col_double(),
##   `2XP` = col_double(),
##   Safety = col_double(),
##   Points = col_double(),
##   `Points/G` = col_double(),
##   Year = col_double()
## )
```

```
total <- read_csv("data/totaloffense.csv")
```

```
## Parsed with column specification:
## cols(
##   Name = col_character(),
##   G = col_double(),
##   `Rush Yards` = col_double(),
##   `Pass Yards` = col_double(),
##   Plays = col_double(),
##   `Total Yards` = col_double(),
##   `Yards/Play` = col_double(),
```

```
##   `Yards/G` = col_double(),
##   Year = col_double()
## )
```

```
offense <- total %>% left_join(scoring, by=c("Name", "Year"))
```

We're going to need this later, so let's grab Nebraska's 2018 stats from this dataframe.

```
nu <- offense %>% filter(Name == "Nebraska") %>% filter(Year == 2018)
```

We'll start with the basics.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey")
```

Let's take changing things one by one. The first thing we can do is change the figure size. Sometimes you don't want a square. We can use the `knitr` output settings in our chunk to do this easily in our notebooks.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey")
```

Now let's add a fit line.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE)
```

And now some labels.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE) +
  labs(x="Total yards per game", y="Points per game", title="Nebraska's underperforming offense",
```

Let's get rid of the default plot look and drop the grey background.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE) +
  labs(x="Total yards per game", y="Points per game", title="Nebraska's underperforming
  theme_minimal()
```

Off to a good start, but our text has no real heirarchy. We'd want our headline to stand out more. So let's change that. When it comes to changing text, the place to do that is in the theme element. There are a lot of ways to modify the theme. We'll start easy. Let's make the headline bigger and bold.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE) +
  labs(x="Total yards per game", y="Points per game", title="Nebraska's underperforming
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold")
    )
```

Now let's fix a few other things – like the axis labels being too big, the subtitle could be bigger and lets drop some grid lines.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE) +
  labs(x="Total yards per game", y="Points per game", title="Nebraska's underperforming
```

```
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 8),
    plot.subtitle = element_text(size=10),
    panel.grid.minor = element_blank()
    )
```

Missing from this graph is the context that the headline promises. Where is Nebraska? We haven't added it yet. So let's add a point and a label for it.

```
ggplot(offense, aes(x=`Yards/G`, y=`Points/G`)) +
  geom_point(color="grey") + geom_smooth(method=lm, se=FALSE) +
  labs(x="Total yards per game", y="Points per game", title="Nebraska's underperforming offense",
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 8),
    plot.subtitle = element_text(size=10),
    panel.grid.minor = element_blank()
    ) +
  geom_point(data=nu, aes(x=`Yards/G`, y=`Points/G`), color="red") +
  geom_text_repel(data=nu, aes(x=`Yards/G`, y=`Points/G`, label="Nebraska 2018"))
```

If we're happy with this output – if it meets all of our needs for publication – then we can simply export it as a png file. We do that by adding `+ ggsave("plot.png", width=5, height=2)` to the end of our code. Note the width and the height are from our knitr parameters at the top – you have to repeat them or the graph will export at the default 7x7.

If there's more work you want to do with this graph that isn't easy or possible in R but is in Illustrator, simply change the file extension to `pdf` instead of `png`. The pdf will open as a vector file in Illustrator with everything being fully editable.

Chapter 29

Finishing Touches 2

Frequently in my classes, students use the waffle charts library quite extensively to make graphics. This is a quick walkthough on how to get a waffle chart into a publication ready state.

```
library(waffle)
```

Let's look at the offensive numbers from Nebraska v. Wisconsin football game. Nebraska lost 41-24, but Wisconsin gained only 15 yards more than Nebraska did. You can find the official stats on the NCAA's website.

I'm going to make two vectors for each team and record rushing yards and passing yards.

```
nu <- c("Rushing"=111, "Passing"=407, 15)
wi <- c("Rushing"=370, "Passing"=163, 0)
```

So what does the breakdown of Nebraska's night look like? How balanced was the offense?

The waffle library can break this down in a way that's easier on the eyes than a pie chart. We call the library, add the data, specify the number of rows, give it a title and an x value label, and to clean up a quirk of the library, we've got to specify colors.

ADDITIONALLY

We can add labels and themes, but you have to be careful. The waffle library is applying it's own theme, but if we override something they are using in their theme, some things that are hidden come back and make it worse. So here is an example of how to use ggplot's `labs` and the theme to make a fully publication ready graphic.

```
waffle(nu/10, rows = 5, xlab="1 square = 10 yards", colors = c("black", "red", "white")
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank()
  )
```

Note: The alignment of text sucks.

How to fix that? We can use ggsave to a pdf and fix it in Illustrator.

```
waffle(nu/10, rows = 5, xlab="1 square = 10 yards", colors = c("black", "red")) + labs
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank()
  ) + ggsave("waffle.pdf")
```

But what if we're using a waffle iron? And what if we want to change the output size? It gets tougher.

Truth is, I'm not sure what is going on with the sizing. You can try it and you'll find that the outputs are … unpredictable.

Things you need to know about waffle irons:

- They're a convenience method, but all they're really doing is executing two waffle charts together. If you don't apply the theme to both waffle charts, it breaks.
- You will have to get creative about applying headline and subtitle to the top waffle chart and the caption to the bottom.
- Using ggsave doesn't work either. So you'll have to use R's pdf output.

Here is a full example. I start with my waffle iron code, but note that each waffle is pretty much a self contained thing. That's because a waffle iron isn't really a thing. It's just a way to group waffles together, so you have to make each waffle individually. My first waffle has the title and subtitle but no x axis labels and the bottom one has not title or subtitle but the axis labels and the caption.

```
iron(
 waffle(
   nu/10,
   rows = 2,
   colors = c("black", "red", "white")) +
   labs(title="Nebraska vs Wisconsin: By the numbers", subtitle="The Huskers couldn't run, Wiscon
   theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank()
  ),
 waffle(
   wi/10,
   rows = 2,
   xlab="1 square = 10 yards",
   colors = c("black", "red", "white")) + labs(caption="Source: NCAA | Graphic by Matt Waite")
)
```

If you try to use ggsave on that, you'll only get the last waffle chart. Like I said, irons aren't really anything, so ggplot ignores them. So to do this, we have to use R's pdf capability.

Here's the same code, but wrapped in the R pdf functions. The first line says we're going to output this as a pdf with this name. Then my code, then `dev.off` to tell R that's what I want as a PDF. Don't forget that.

```
pdf("waffleiron.pdf")
iron(
 waffle(
   nu/10,
   rows = 2,
```

```
   colors = c("black", "red", "white")) +
   labs(title="Nebraska vs Wisconsin: By the numbers", subtitle="The Huskers couldn't r
   theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.title.y = element_blank()
  ),
 waffle(
   wi/10,
   rows = 2,
   xlab="1 square = 10 yards",
   colors = c("black", "red", "white")) + labs(caption="Source: NCAA | Graphic by Matt
)
dev.off()
```

It probably still needs work in Illustrator, but less than before.

Chapter 30

Plotly

By John Strasheim

We've been working on making charts and graphs and outputting them as static images – png files. Why? Because images will embed into any website, work just fine on mobile, and require no special coding. But the wonder of the internet is that it's interactive. The trouble with interactive graphics, though, is the code can be exceedingly complicated and difficult for beginners to grasp, as is the case with the javascript visualization library D3. Or the tools are ultra simple and allow for minimal customization, such as tools like Tableau. The third problem is that the accessible interactive tools – the ones that don't require a ton of code knowledge – cost money to publish.

The library we're going to look at in this chapter is from a company called Plotly, which sits in between all these problems. You can create interactive graphs with point and click, but you can also do it in code. You can publish graphs for free, but if you're going to do it for a company or with a large audience, you need to pay. For our purposes, you'll see the power without needing to pony up.

You will need to install plotly. Go to the console – not in the notebook – and run `install.packages("plotly")`.

Then we'll load it:

```
library(tidyverse)
library(plotly)
```

To start out, we'll make a graph similar to something we've already done. We're going to use a dataset of batting stats from the 2019 season. This dataset has players who had more than 190 plate appearances and includes their basic stats and some advanced metrics.

```
batting <- read_csv("data/batting.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Name = col_character(),
##   Tm = col_character(),
##   Division = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

Let's look at the relationship between a player's batting average and wRC+ (weighted runs created plus).

```
ggplot() +
  geom_point(data=batting, aes(x=`wRC+`, y=AVG)) +
  geom_smooth(data=batting, aes(x=`wRC+`, y=AVG), method='lm', se=FALSE) +
  labs(x='wRC+', y='Batting Average', title= "2019 MLB Season Batting", subtitle="190 P
```

You can obviously add more aesthetics to make it look better, but you get the picture. If I'm a fan of sports though, obviously I want to see who the outlier points are or just scroll through and see each player at an individual location. We can always annotate data, but that process can be tedious.

Here is where plotly comes in.

Plotly will make your visualizatons interactive. Additionally, you can zoom in

on certain parts of the viz too. For example, you can drag a box around players with a .300 average, and see all the guys in that specific range.

Let's start simple with just the minimum needed for a scatterplot. For that, we need to specify our data source – `batting` – and set an X and a Y value, just like above. One difference? We prepend a ~ before the field names. We'll add a color to separate out players by division too.

```
plot_ly(data=batting, x= ~`wRC+`, y= ~AVG, color= ~Division)
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```

We get a chart, but hover over a point. Recognize those players? You can't unless you know each player's specific stats to divine who they are. That isn't very friendly, so let's add a hover element. Then we want to specify what we want our users to see when they hover over a data point, hence hoverinfo = "text". The next step will be to define what our text is. How that gets done is a little bit of HTML and a little bit of R. What is in quotes is what the users are going to see directly, what's after the quotes is what data is going to appear. So "Player:", Name translates to something like Player: Christian Yelich when the user hovers above Yelich's data point.

Then we add the HTML, `<br>`, or break. All this means is we're having a line break so all of our data is not on one line. Simple. Do this process for whatever

variables you want to have your users see. So for mine, I wanted my users to see the Player's name, wRC+, Batting Avg, and what division they are in.

```
plot_ly(data=batting, x= ~`wRC+`, y= ~`AVG`, color= ~`Division`,
        hoverinfo = "text",
        text = ~paste("Player:", Name,
                      '<br>wRC+:', `wRC+`,
                      '<br>AVG:', AVG,
                      '<br>Team:', Tm,
                      '<br>Division:', Division
                      ))
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```

Now we can see each player a little better. If you look at the players on the farthest right, you'll find Mike Trout (shocker), Yordan Alvarez and Christian Yelich.

To finish, we're going to fix the layout a bit, very similar to how we've been doing it in ggplot. We're just telling plotly what we want the layout to be of our viz, starting with the title, and then doing the x and y axis names after that.

```
plot_ly(data=batting, x= ~`wRC+`, y= ~`AVG`, color= ~`Division`,
```

```
        hoverinfo = "text",
        text = ~paste("Player:", Name,
                      '<br>wRC+:', `wRC+`,
                      '<br>AVG:', AVG,
                      '<br>Team:', Tm,
                      '<br>Division:', Division
                      )) %>%
  layout(
    title = "2019 MLB Season Batting",
    xaxis = list(title = "wRC+"),
    yaxis = list(title = "Batting Average")
  )
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```

30.1 Publishing using Plotly

We want to now export our plotly visualization. First, you'll need to sign up for a free plotly account. Then you'll need to register your plotly username and your API key.

More info about how to do that can be found on Plotly's website.

For our purposes, we need to register our username and API key this way, where you put your username and API key where prompted:

```
Sys.setenv("plotly_username"="Enter your plotly username here")
Sys.setenv("plotly_api_key"="Enter your API key here")
```

Now run this line of code specifiying what variable you are exporting, and what you want the file to be named on plotly's servers. From plotly's website you can then do several different things like editing it on there, embedding it on websites, or create a shareable link.

To publish our chart, we need to save it to an object similar to how we've been creating dataframes. So something like this:

```
p <- plot_ly(data=batting, x= ~`wRC+`, y= ~`AVG`, color= ~`Division`,
        hoverinfo = "text",
        text = ~paste("Player:", Name,
                      '<br>wRC+:', `wRC+`,
                      '<br>AVG:', AVG,
                      '<br>Team:', Tm,
                      '<br>Division:', Division
                      )) %>%
  layout(
    title = "2019 MLB Season Batting",
    xaxis = list(title = "wRC+"),
    yaxis = list(title = "Batting Average")
  )
```

To publish it, we simply run the following, passing in our chart value `p` for plotly and we give it a filename.

```
api_create(p, filename="MLBOffense19")
```

If all goes well, a browser will pop up with your chart in it.

Chapter 31

Clustering

One common effort in sports is to classify teams and players – who are this players peers? What teams are like this one? Who should we compare a player to? Truth is, most sports commentators use nothing more sophisticated that looking at a couple of stats or use the "eye test" to say a player is like this or that.

There's better ways.

In this chapter, we're going to use a method that sounds advanced but it really quite simple called k-means clustering. It's based on the concept of the k-nearest neighbor algorithm. You're probably already scared. Don't be.

Imagine two dots on a scatterplot. If you took a ruler out and measured the distance between those dots, you'd know how far apart they are. In math, that's called the Euclidean distance. It's just the space between them in numbers. Where k-nearest neighbor comes in, you have lots of dots and you want measure the distance between all of them. What does k-means clustering do? It lumps them into groups based on the average distance between them. Players who are good on offense but bad on defense are over here, good offense good defense are over here. And using the Eucliean distance between them, we can decide who is in and who is out of those groups.

For this exercise, I want to look at Cam Mack, Nebraska's point guard and probably the most interesting player on Fred Hoiberg's first team. This is Mack's first year in major college basketball – he played a year at a community college – so we don't have much to go on. But with three games in the books, who does Cam Mack compare to?

To answer this, we'll use k-means clustering.

First thing we do is load some libraries and set a seed, so if we run this repeatedly, our random numbers are generated from the same base. If you don't have the

cluster library, just add it on the console with `install.packages("cluster")`

```
library(tidyverse)
library(cluster)

set.seed(1234)
```

I've gone and scraped stats for every player in this current season so let's load that up.

```
players <- read_csv("data/players20.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   Team = col_character(),
##   Player = col_character(),
##   Class = col_character(),
##   Pos = col_character(),
##   Height = col_character(),
##   Hometown = col_character(),
##   `High School` = col_character(),
##   Summary = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

To cluster this data properly, we have some work to do.

First, it won't do to have players who haven't played, so we can use filter to find anyone with greater than 0 minutes played. Next, Cam Mack is a guard, so let's just look at guards. Third, we want to limit the data to things that make sense to look at for Cam Mack – things like shooting, three point shooting, assists, turnovers and points.

```
playersselected <- players %>%
  filter(MP>0) %>% filter(Pos == "G") %>%
  select(Player, Team, Pos, MP, `FG%`, `3P%`, AST, TOV, PTS) %>%
  na.omit()
```

Now, k-means clustering doesn't work as well with data that can be on different scales. So comparing a percentage to a count metric – shooting percentage to points – would create chaos because shooting percentages are a fraction of 1 and points, depending on when they are in the season, could be quite large. So we have to scale each metric – put them on a similar basis using the distance from the max value as our guide. Also, k-means clustering won't work with text data, so we need to create a dataframe that's just the numbers, but scaled. We can do that with another select, and using mutate_all with the scale function. The `na.omit()` means get rid of any blanks, because they too will cause errors.

```
playersscaled <- playersselected %>%
  select(MP, `FG%`, `3P%`, AST, TOV, PTS) %>%
  mutate_all(scale) %>%
  na.omit()
```

With k-means clustering, we decide how many clusters we want. Most often, researchers will try a handful of different cluster numbers and see what works. But there are methods for finding the optimal number. One method is called the Elbow method. One implementation of this, borrowed from the University of Cincinnati's Business Analytics program, does this quite nicely with a graph that will help you decide for yourself.

All you need to do in this code is change out the data frame – `playersscaled` in this case – and run it.

```
# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(playersscaled, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15

# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```
plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```

The Elbow method – so named because you're looking for the "elbow" where the line flattens out. In this case, it looks like a K of 5 is ideal. So let's try that. We're going to use the kmeans function, saving it to an object called k5. We just need to tell it our dataframe name, how many centers (k) we want, and we'll use a sensible default for how many different configurations to try.

```
k5 <- kmeans(playersscaled, centers = 5, nstart = 25)
```

Let's look at what we get.

```
k5
```

```
## K-means clustering with 5 clusters of sizes 863, 989, 242, 58, 495
##
## Cluster means:
##           MP         FG%         3P%        AST        TOV        PTS
## 1 -0.8549890  0.02411493 -0.04833668 -0.7090093 -0.7700257 -0.7982928
## 2  0.5451701  0.14083415  0.12854961  0.1392254  0.2539075  0.3248641
## 3 -1.3602270 -1.98012908 -1.65325745 -0.9392522 -1.1172744 -1.1417360
## 4 -1.3785814  3.22934178  3.92207534 -0.9251265 -1.1485250 -1.1077564
## 5  1.2279089  0.26624900  0.17613521  1.5255303  1.5159849  1.4306790
##
## Clustering vector:
##    [1] 5 2 2 2 2 1 1 1 1 2 2 5 2 1 1 1 5 2 1 1 3 3 5 2 2 2 2 2 1 4 2 2 5 2 2 1 3
##   [38] 5 5 2 2 1 2 1 1 5 2 2 2 1 1 2 5 2 2 2 2 1 1 3 2 5 1 2 2 1 1 1 1 1 2 2 5 2
##   [75] 2 1 3 1 3 5 2 2 2 1 1 1 1 5 2 2 2 2 1 3 5 2 2 1 1 1 5 2 1 2 1 1 1 1 4 2 2
##  [112] 1 1 1 2 1 3 5 2 2 2 1 1 1 5 2 2 1 3 3 5 2 2 1 1 1 5 2 2 2 2 1 1 5 5 2 2 2
##  [149] 2 3 5 2 2 2 1 1 1 3 5 2 2 2 2 5 1 3 5 5 2 2 2 2 1 1 2 2 2 2 1 1 1 1 1 1 5
##  [186] 2 3 3 5 2 2 2 1 1 3 1 5 5 2 2 1 1 1 4 3 5 2 2 5 2 2 1 1 1 3 5 2 1 1 1 3 3
##  [223] 5 5 2 2 2 2 2 3 5 2 2 5 1 5 2 5 2 2 1 1 1 2 2 2 2 2 1 3 5 5 5 2 2 2 1 1 3
```

```
##  [260] 3 5 5 2 2 2 1 1 1 3 5 5 2 2 2 1 1 1 5 2 2 2 3 5 5 2 2 1 3 2 5 5 2 5 2 1 1
##  [297] 5 2 2 2 1 1 1 1 1 5 2 2 1 3 1 3 3 5 5 2 2 1 1 5 5 2 2 2 1 1 2 2 2 2 1 1 1
##  [334] 5 5 2 2 2 2 1 5 5 2 1 1 1 3 5 2 2 2 2 1 3 5 5 2 1 2 1 1 3 3 5 2 2 2 2 1 1
##  [371] 1 5 5 2 2 1 1 1 1 1 1 1 5 5 2 5 2 1 1 1 3 5 5 2 1 1 5 5 2 2 1 3 5 5 1 1 1
##  [408] 5 5 2 1 1 3 5 2 2 2 2 1 1 1 4 5 2 2 2 2 1 1 1 5 5 2 1 1 2 5 2 2 2 2 1 3 5
##  [445] 5 2 2 1 1 1 1 5 5 2 2 1 3 5 2 2 2 1 1 1 1 2 5 2 2 2 1 1 1 5 2 2 1 1 1 1 3
##  [482] 5 5 2 2 1 1 3 3 5 5 2 1 1 2 1 5 5 2 2 2 1 3 3 2 2 2 5 1 1 5 2 2 2 2 1 3 5
##  [519] 2 2 2 1 1 4 1 5 5 2 2 2 1 3 4 5 5 2 5 1 1 1 3 5 2 2 2 2 1 1 1 1 5 5 2 1 1
##  [556] 1 5 5 2 2 2 1 1 3 2 2 2 2 1 3 3 3 2 2 2 1 1 1 3 5 5 5 1 1 1 1 3 3 5 5 2 2
##  [593] 1 2 5 5 2 2 2 2 1 1 4 5 2 2 1 3 3 3 5 2 2 2 2 1 1 3 3 5 5 2 2 1 5 2 2 2 1
##  [630] 1 3 5 2 5 2 1 1 3 5 2 2 2 1 4 3 5 2 2 2 2 1 2 2 2 2 2 2 1 1 1 5 2 5 1 1 1
##  [667] 5 2 2 5 1 1 1 3 4 5 2 2 1 1 1 5 5 2 2 2 1 1 1 3 5 2 2 2 5 1 1 2 2 2 1 3 3
##  [704] 3 2 5 2 2 2 2 4 1 3 2 5 5 2 5 1 3 5 5 2 2 1 1 3 1 2 5 2 1 1 2 2 5 2 2 1 2
##  [741] 1 1 5 5 2 1 5 5 2 1 1 1 1 4 3 5 2 2 1 1 5 2 2 2 2 1 1 5 5 5 2 2 1 4 2 2 2
##  [778] 1 1 1 1 1 5 2 2 2 2 1 1 3 3 2 2 2 2 2 4 1 4 1 1 2 2 5 2 2 2 1 5 2 2 2 1 1
##  [815] 1 2 5 5 2 2 3 1 3 3 5 2 2 2 1 3 5 2 2 1 1 1 1 1 1 2 2 2 2 2 1 3 3 5 2 2 2
##  [852] 1 1 2 1 5 2 2 2 1 5 2 1 1 1 5 2 2 2 1 1 1 5 5 2 2 1 1 1 4 5 2 2 2 3 2 5 2
##  [889] 1 2 1 1 1 4 5 2 2 3 1 1 3 2 5 2 2 1 2 2 5 2 1 1 3 3 3 5 5 2 2 1 3 3 5 2 1
##  [926] 1 1 2 2 2 2 1 2 1 1 1 1 5 5 2 5 1 1 1 1 1 5 2 2 2 1 1 5 2 2 2 1 1 1 5 2 2
##  [963] 2 2 1 1 5 2 2 1 1 1 3 3 5 5 2 2 2 1 1 1 3 5 5 2 1 1 1 1 1 1 5 5 2 1 2 3 5
## [1000] 2 2 1 1 1 3 5 5 2 2 1 1 1 1 1 5 2 2 2 2 2 1 1 3 2 5 2 2 2 2 2 1 4 3 5 5 2
## [1037] 2 3 1 5 2 2 2 1 3 1 2 5 2 1 2 1 2 3 2 5 2 2 2 1 3 5 2 5 5 2 1 1 3 5 2 5 1
## [1074] 1 3 3 5 2 5 2 1 1 1 1 1 1 1 5 2 2 1 1 1 1 5 5 5 2 1 1 1 1 3 5 2 2 2 1 2 1
## [1111] 1 4 5 5 1 2 1 1 1 1 5 5 2 2 2 1 2 1 1 1 3 2 2 2 1 2 2 1 1 4 1 3 5 2 2 2 1
## [1148] 1 5 5 5 2 1 3 5 5 2 2 1 3 2 5 2 2 2 1 1 1 5 5 2 2 2 1 3 1 5 2 2 1 1 4 3 1
## [1185] 3 5 2 5 1 3 1 3 5 2 2 1 3 1 3 1 3 5 2 2 2 1 1 1 3 1 5 2 5 2 2 1 5 5 2 2 1
## [1222] 5 5 2 1 1 2 5 2 2 2 1 1 5 5 2 2 2 3 1 4 5 2 2 2 1 1 3 2 2 2 2 1 2 1 1 5 5
## [1259] 2 1 1 3 3 1 5 5 5 2 1 1 3 5 2 2 2 2 3 5 2 2 1 3 3 5 5 2 2 2 1 4 1 2 5 5 2
## [1296] 2 1 3 5 2 2 2 1 4 3 5 2 1 1 1 1 3 3 5 2 2 2 2 1 1 2 2 2 2 2 1 5 2 2 2 2 1
## [1333] 1 3 1 5 5 2 2 2 1 1 1 2 2 2 5 2 1 1 2 5 2 1 2 2 1 1 2 5 1 2 1 1 4 3 5 5 2
## [1370] 2 2 1 3 3 5 2 2 2 1 3 4 4 2 2 2 2 1 2 1 3 5 2 2 1 1 1 1 3 5 5 2 1 1 1 5 2
## [1407] 5 1 1 4 5 2 2 2 1 1 3 3 2 2 2 2 2 1 3 4 3 5 1 1 1 1 1 3 5 5 2 2 2 1 1 4 3
## [1444] 3 3 2 1 1 1 1 5 2 2 2 1 1 1 3 2 2 2 2 2 1 5 2 2 2 2 1 1 3 1 5 5 1 1 1 1 1
## [1481] 5 2 2 1 1 1 3 5 5 2 1 1 1 1 1 5 5 2 2 1 1 5 5 5 2 2 1 4 3 2 5 2 2 2 1 1 2
## [1518] 2 2 2 1 1 1 1 1 2 2 2 2 1 1 1 5 5 2 2 1 3 3 5 5 5 2 2 1 5 2 2 1 1 1 1 2 2
## [1555] 2 2 2 5 2 3 2 5 5 1 5 2 2 2 1 1 1 5 2 1 1 1 1 1 3 5 2 5 2 1 1 1 1 4 5 2 2
## [1592] 2 1 5 2 5 1 3 1 1 2 2 2 2 2 2 1 3 3 2 2 2 2 2 2 1 1 3 5 2 5 2 1 1 2 1 1 4
## [1629] 3 3 3 5 5 5 2 2 1 5 2 2 2 2 1 3 2 2 2 2 2 1 1 3 5 5 2 1 2 1 1 4 3 5 2 2 2
## [1666] 1 3 3 2 2 2 1 4 5 5 2 2 2 1 1 1 2 5 2 2 2 1 1 1 5 2 2 2 1 3 1 5 2 2 2 1 2
## [1703] 1 1 1 5 5 2 2 1 5 2 2 2 2 1 1 1 5 2 2 1 1 1 1 4 5 5 2 2 2 1 1 1 2 5 2 5 2
## [1740] 5 2 2 2 2 1 3 5 2 2 1 1 3 4 5 5 2 2 1 1 3 5 2 2 2 2 3 2 5 2 2 1 1 5 2 2 1
## [1777] 1 1 1 3 2 5 5 2 1 1 1 5 2 2 2 4 1 1 1 3 5 5 5 2 2 2 1 4 5 5 2 2 1 5 2 2 2
## [1814] 1 2 1 3 1 4 3 2 5 5 1 1 1 5 5 2 2 1 1 3 3 5 5 2 2 1 1 1 5 5 5 5 2 3 5 2 5
## [1851] 2 2 1 1 1 1 2 2 2 5 5 1 4 5 2 5 1 1 1 1 3 5 2 2 2 1 3 2 5 5 1 2 1 4 3 5 2
## [1888] 5 1 1 1 3 2 2 2 2 2 2 1 3 1 2 2 2 1 2 1 1 5 2 2 1 1 2 5 2 2 1 1 1 4 5 2 2
## [1925] 2 2 1 1 1 5 2 5 2 1 1 3 3 5 2 2 1 1 1 2 5 2 1 2 1 5 2 2 1 1 1 3 2 2 2 2 1
```

```
## [1962] 5 2 2 2 2 2 1 4 4 5 5 5 2 2 1 1 3 3 3 5 2 2 5 2 1 1 1 1 1 2 2 1 2 1 1 3 5
## [1999] 5 2 2 1 1 1 5 2 2 5 1 2 1 1 1 5 5 5 2 2 3 5 2 2 2 2 1 1 5 2 2 2 3 3 4 4 5
## [2036] 2 1 2 3 5 5 2 1 3 5 2 2 2 2 2 1 1 4 3 5 2 2 1 1 3 5 2 1 1 1 3 3 1 5 5 2 2
## [2073] 1 1 1 1 5 5 5 1 1 1 3 5 5 2 2 2 1 1 1 3 5 2 2 5 1 1 1 3 5 5 2 2 2 1 1 4 3
## [2110] 2 2 2 2 2 1 1 1 5 5 5 2 1 1 1 1 2 2 1 1 1 3 2 5 2 2 2 2 1 1 5 2 5 2 2 1 3
## [2147] 5 2 2 2 1 1 3 1 5 5 2 2 1 1 1 5 5 2 2 2 1 1 4 5 2 2 1 1 4 3 4 5 5 2 2 1 1
## [2184] 5 2 2 2 1 1 1 3 2 5 2 2 2 2 1 1 4 1 2 2 5 2 2 1 1 1 1 1 2 2 2 5 2 1 1 3 5
## [2221] 2 2 2 1 1 1 1 5 2 2 1 1 5 5 2 2 1 1 1 1 1 5 5 2 2 2 1 1 3 1 5 2 2 2 1 1 1
## [2258] 3 2 2 5 2 2 1 1 5 2 5 2 2 1 1 1 2 2 2 2 1 2 1 3 1 5 5 5 2 1 1 1 2 5 2 2 1
## [2295] 1 2 2 2 2 2 1 1 1 4 5 2 2 2 1 1 3 5 2 2 2 1 1 2 2 2 2 1 2 1 1 5 2 2 1 1 1
## [2332] 2 5 2 2 2 2 1 1 1 1 5 2 5 2 1 1 1 4 5 5 5 1 3 5 5 2 1 1 4 3 2 5 2 2 2 2 2
## [2369] 2 2 2 3 3 5 5 1 1 5 5 2 2 1 3 5 2 2 1 1 1 5 5 2 2 2 3 1 5 2 1 1 1 1 3 3 5
## [2406] 5 2 2 2 1 1 1 4 5 2 2 2 3 1 1 2 2 5 2 2 2 1 1 1 2 5 2 2 2 1 1 1 3 5 5 2 1
## [2443] 1 5 5 2 2 2 1 1 4 5 2 2 2 1 1 1 5 2 2 2 2 2 1 4 5 5 2 2 1 1 1 1 5 2 5 2 2
## [2480] 1 3 3 5 5 2 2 1 1 3 5 5 2 5 2 2 1 1 1 5 5 1 1 1 3 1 5 5 2 2 1 1 3 1 3 5 2
## [2517] 2 2 2 5 5 2 2 2 2 3 3 1 1 2 1 1 1 1 1 3 5 2 2 2 1 1 1 3 5 2 2 2 1 1 5 2 2
## [2554] 1 1 1 1 5 2 2 1 1 1 3 3 3 5 2 2 2 1 1 3 3 5 5 2 2 1 1 1 3 5 5 2 2 2 1 3 1
## [2591] 5 5 2 2 1 1 5 2 2 2 2 1 1 1 1 1 1 5 5 5 1 1 1 1 3 2 2 5 2 3 1 3 3 5 5 2 1
## [2628] 3 3 5 2 2 2 1 1 1 1 1 1 5 2 2 2 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 1430.7781 1341.0495  319.4278  250.5515 1094.8283
##  (between_SS / total_SS =  72.1 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Interpreting this output, the very first thing you need to know is that **the cluster numbers are meaningless**. They aren't ranks. They aren't anything. After you have taken that on board, look at the cluster sizes at the top. Clusters 3 and 5 are pretty large compared to others. That's notable. Then we can look at the cluster means. For reference, 0 is going to be average. So group 1 are well above average on minutes played. Group 3 is slightly above, group 5 is slightly below. In fact, group 5 is below average on every metric. Group 3 is slightly above average on all metrics.

So which group is Cam Mack in? Well, first we have to put our data back together again. In K5, there is a list of cluster assignments in the same order we put them in, but recall we have no names. So we need to re-combine them with our original data. We can do that with the following:

```
playercluster <- data.frame(playersselected, k5$cluster)
```

Now we have a dataframe called playercluster that has our player names and what cluster they are in. The fastest way to find Cam Mack is to double click

on the playercluster table in the environment and use the search in the top right of the table. Because this is based on some random selections of points to start the groupings, these may change from person to person, but Mack is in Group 1 in my data.

We now have a dataset and can plot it like anything else. Let's get Cam Mack and then plot him against the rest of college basketball on assists versus minutes played.

```
cm <- playercluster %>% filter(Player == "Cameron Mack")
```

```
ggplot() +
  geom_point(data=playercluster, aes(x=MP, y=AST, color=k5.cluster)) +
  geom_point(data=cm, aes(x=MP, y=AST), color="red")
```

Not bad, not bad. But who are Cam Mack's peers? If we look at the numbers in Group 1, there's 335 of them. So let's limit them to just Big Ten guards. Unfortunately, my scraper didn't quite work and in the place of Conference is the coach's name. So I'm going to have to do this the hard way and make a list of Big Ten teams and filter on that. Then I'll sort by minutes played.

```
big10 <- c("Nebraska Cornhuskers", "Iowa Hawkeyes", "Minnesota Golden Gophers", "Illinois Fightin

playercluster %>% filter(k5.cluster == 4) %>% filter(Team %in% big10) %>% arrange(desc(MP))
```

```
##          Player                    Team Pos MP   FG.  X3P. AST TOV PTS
## 1   Cole Bajema     Michigan Wolverines   G 35 0.750 0.571   0   2  24
```

```
## 2    Reese Mona      Maryland Terrapins   G 30 1.000 1.000   2   1   9
## 3   Joey Downes Rutgers Scarlet Knights   G 10 0.667 1.000   0   1   8
## 4 Travis Valmon      Maryland Terrapins   G  8 0.500 1.000   0   0   4
## 5  Cooper Bybee        Indiana Hoosiers   G  4 1.000 1.000   0   0   3
##   k5.cluster
## 1          4
## 2          4
## 3          4
## 4          4
## 5          4
```

So there are the 8 guards most like Cam Mack in the Big Ten. It'll be interesting to watch this evolve over the season. Fred Hoiberg and others think he might be one of the best guards in the league. We'll see, using cluster analysis.

31.1 Advanced metrics

How much does this change if we change the metrics? I used pretty standard box score metrics above. What if we did it using Player Efficiency Rating, True Shooting Percentage, Point Production, Assist Percentage, Win Shares Per 40 Minutes and Box Plus Minus (you can get definitions of all of them by hovering over the stats on Nebraksa's stats page).

We'll repeat the process. Filter out players who don't play, players with stats missing, and just focus on those stats listed above.

```
playersadvanced <- players %>%
  filter(MP>0) %>%
  filter(Pos == "G") %>%
  select(Player, Team, Pos, PER, `TS%`, PProd, `AST%`, `WS/40`, BPM) %>%
  na.omit()
```

Now to scale them.

```
playersadvscaled <- playersadvanced %>%
  select(PER, `TS%`, PProd, `AST%`, `WS/40`, BPM) %>%
  mutate_all(scale) %>%
  na.omit()
```

Let's find the optimal number of clusters.

```
# function to compute total within-cluster sum of square
wss <- function(k) {
  kmeans(playersadvscaled, k, nstart = 10 )$tot.withinss
}

# Compute and plot wss for k = 1 to k = 15
k.values <- 1:15
```

```r
# extract wss for 2-15 clusters
wss_values <- map_dbl(k.values, wss)
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```r
plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Number of clusters K",
       ylab="Total within-clusters sum of squares")
```

Looks like 5 again.

```r
advk5 <- kmeans(playersadvscaled, centers = 5, nstart = 25)
```

What do we have here?

```r
advk5
```

```
## K-means clustering with 5 clusters of sizes 104, 632, 766, 1253, 9
##
## Cluster means:
##          PER        TS%      PProd       AST%      WS/40        BPM
## 1 -2.6922417 -2.6802561 -1.1497380 -1.0109009 -2.8398288 -2.8455990
## 2 -0.6089002 -0.6736304 -0.8322766 -0.2324539 -0.5651578 -0.6493960
```

```
## 3  0.5022214  0.2566711  1.2288970  0.7362004  0.4083263  0.5004325
## 4  0.1594366  0.3741799 -0.2279088 -0.2650562  0.2130658  0.2118878
## 5  8.9269330  4.3359455 -1.1326482  2.2478231  8.0858448  6.3926472
##
## Clustering vector:
##    [1] 3 4 4 4 4 2 4 2 4 3 4 3 4 4 2 4 3 4 4 4 2 2 4 1 3 4 4 4 4 2 4 4 3 3 3 4 4
##   [38] 4 2 3 3 4 4 4 4 4 2 2 3 4 4 4 4 4 4 1 3 3 4 4 3 4 4 4 4 3 3 3 4 4 4 2 2 2
##   [75] 2 2 3 4 3 4 3 2 1 4 2 3 4 4 4 4 2 4 2 3 3 4 4 4 2 2 3 3 4 2 2 2 2 3 4 4 4
##  [112] 4 4 2 2 4 3 3 4 4 4 4 2 2 3 3 4 2 4 2 2 3 3 4 4 2 4 5 1 3 4 4 4 2 4 3 4 4
##  [149] 4 4 2 4 1 3 3 4 4 3 4 2 3 3 4 2 4 2 5 2 1 3 3 4 4 4 3 4 2 3 3 4 3 4 4 4 2
##  [186] 3 3 3 4 4 4 4 4 4 2 3 3 2 2 3 4 4 4 2 4 2 4 3 3 4 4 2 4 4 5 2 3 4 4 3 4 4
##  [223] 4 2 4 2 3 4 4 4 4 2 2 3 3 4 4 4 4 4 2 3 4 3 3 4 3 4 3 4 4 4 4 4 2 1 3 3 3
##  [260] 4 2 4 1 3 3 3 4 4 4 4 4 2 3 1 1 3 3 4 4 4 4 4 2 2 3 3 4 4 4 4 4 4 4 3 4 4
##  [297] 4 3 2 3 3 3 4 2 1 3 3 3 4 3 4 4 4 3 3 4 3 4 4 4 4 4 3 4 2 2 2 2 2 2 3 3 4
##  [334] 4 2 2 3 3 4 4 2 4 4 3 4 2 4 2 4 2 3 3 4 4 2 4 4 3 3 4 2 2 2 2 3 4 3 2 4 2
##  [371] 2 3 3 4 4 2 2 2 2 2 3 4 3 4 3 4 4 3 3 3 4 4 4 4 2 2 2 2 4 2 3 3 4 3 2 4 2
##  [408] 4 1 3 3 4 4 2 2 3 3 3 4 4 4 1 3 3 2 2 1 4 1 3 3 4 4 2 2 2 3 4 4 4 2 2 2 2
##  [445] 4 3 3 4 3 4 4 4 2 3 3 4 4 4 2 3 3 4 4 4 4 4 1 1 3 3 4 4 2 2 2 2 3 3 4 4 2
##  [482] 2 1 3 4 4 4 2 2 4 2 3 3 4 4 4 4 2 2 3 3 2 2 2 2 2 2 1 3 3 4 4 2 4 2 1 3 3
##  [519] 4 4 3 2 2 3 3 3 4 2 2 2 2 2 3 4 4 3 2 2 2 1 1 3 4 4 2 2 2 2 3 4 4 4 2 2 4
##  [556] 4 3 3 4 4 4 4 2 4 3 3 3 3 4 4 2 1 3 4 4 4 3 4 4 2 4 3 3 4 4 2 4 3 3 4 4 2
##  [593] 4 4 2 3 4 4 2 4 2 2 1 3 3 4 4 4 2 4 1 3 3 3 4 4 2 4 2 1 3 3 4 4 4 4 2 3 3
##  [630] 4 4 4 4 2 2 4 3 4 4 4 2 1 1 3 3 3 4 4 4 2 2 2 3 3 3 4 4 3 3 4 4 4 2 4 1 3
##  [667] 4 3 4 4 4 2 3 4 2 2 2 5 1 2 3 3 4 4 3 4 4 4 4 3 4 3 2 4 2 1 3 4 3 4 4 4 3
##  [704] 3 4 3 4 4 4 2 5 3 4 4 4 2 2 3 3 4 4 4 2 4 2 4 4 1 3 4 3 4 3 2 4 4 4 3 3 4
##  [741] 2 2 2 3 3 4 4 4 4 4 2 1 3 3 3 4 3 4 2 1 3 3 4 4 4 2 2 2 3 3 3 4 4 4 2 3 3
##  [778] 3 3 2 4 4 2 4 3 3 3 4 3 3 4 4 4 4 2 4 2 3 3 4 2 2 1 3 3 4 4 4 3 4 3 3 3 4
##  [815] 4 4 3 3 3 4 4 4 2 4 2 3 4 4 3 2 4 2 4 2 3 4 4 4 4 4 2 4 2 2 3 1 3 3 3 4 4
##  [852] 2 2 3 3 4 4 4 4 2 3 3 3 4 4 2 4 2 2 2 3 4 2 4 4 2 1 3 4 4 4 4 4 2 4 4 3 4
##  [889] 3 4 4 4 2 4 3 4 3 4 4 4 3 4 3 3 3 4 4 3 4 4 4 2 2 3 3 4 4 2 4 4 3 3 3 4 2
##  [926] 2 2 4 3 4 4 4 4 1 3 3 4 4 2 2 4 2 4 3 4 4 2 2 2 2 3 3 3 4 4 3 4 2 4 2 2 2
##  [963] 2 2 3 3 4 3 4 2 2 3 3 4 2 3 4 4 4 3 4 2 2 4 4 2 3 3 3 3 4 2 2 4 4 3 3 3 4
## [1000] 4 4 3 4 4 4 4 4 2 3 3 4 4 4 4 2 3 4 4 4 2 4 2 1 3 3 3 4 4 4 4 4 2 3 3 4 4
## [1037] 4 4 4 4 4 3 3 4 4 4 1 3 3 4 4 2 4 2 3 3 4 4 4 4 4 2 4 3 3 4 4 4 4 2 4 2 3
## [1074] 3 3 4 4 4 2 4 4 2 3 3 3 4 2 2 3 4 3 4 4 2 4 4 3 2 4 4 4 2 1 2 3 4 4 4 2 2
## [1111] 3 3 3 3 2 4 2 2 2 3 3 3 4 2 2 1 3 3 3 3 4 4 2 4 2 4 4 3 4 2 4 2 2 2 3 3 3
## [1148] 4 4 4 4 4 2 3 3 4 4 4 2 4 2 4 3 3 4 4 4 2 4 2 3 3 4 4 4 4 3 4 4 4 4 3 3 4
## [1185] 4 4 4 4 3 4 4 1 3 4 3 4 4 4 3 3 3 3 2 1 3 3 3 4 4 4 1 3 3 4 4 4 4 2 4 3 3
## [1222] 4 4 4 2 2 2 3 4 3 2 4 5 2 4 2 3 4 3 4 2 4 2 3 4 3 2 2 2 2 2 1 3 4 4 4 4 4
## [1259] 2 2 4 3 4 3 3 4 2 3 3 4 4 2 3 3 4 2 2 3 3 3 4 4 2 2 3 3 4 4 2 2 4 4 3 4 4
## [1296] 4 2 2 4 4 3 4 4 4 4 4 4 3 3 4 4 4 4 2 2 3 3 3 2 2 2 1 3 3 4 3 4 2 3 4 4 2
## [1333] 2 1 3 3 4 4 4 4 4 2 3 3 3 3 4 4 2 3 3 4 4 2 4 2 3 3 4 4 4 3 4 2 2 3 3 3 4
## [1370] 4 2 4 3 3 3 4 3 4 2 3 4 4 4 4 2 2 2 3 3 3 4 4 4 4 2 4 3 4 4 3 3 4 2 3 3 3
## [1407] 4 4 4 4 4 3 3 4 4 4 2 2 4 2 3 3 4 4 2 4 2 1 3 3 3 3 2 2 4 4 4 2 2 2 2 2 2
## [1444] 2 2 3 4 3 4 3 2 4 2 3 3 4 4 2 4 3 3 3 4 4 4 3 4 4 4 2 2 1 2 1 4 3 3 4 4 4
## [1481] 2 5 4 1 3 4 2 2 2 2 2 3 3 4 3 4 2 2 2 2 1 1 4 4 4 2 2 3 3 4 4 4 4 4 2 1 4
```

```
## [1518] 3 3 4 4 4 4 3 4 3 3 4 4 4 2 4 3 3 4 4 2 2 2 3 3 4 4 2 4 2 3 3 4 4 2 2 2 2
## [1555] 3 3 4 4 4 2 3 3 4 4 3 2 4 2 3 3 4 2 4 4 4 3 4 4 4 2 4 2 2 2 3 3 4 4 4 4 2
## [1592] 4 3 3 4 4 2 2 1 3 3 3 4 4 2 3 3 4 4 4 4 4 4 4 4 4 4 3 4 1 3 3 3 4 4 3 3 3
## [1629] 4 4 4 4 3 2 2 2 2 2 2 2 3 3 3 4 4 4 4 4 4 3 4 4 3 2 3 3 3 2 2 2 2 4 4 4 4
## [1666] 4 2 2 2 2 3 4 4 4 4 4 4 4 2 3 3 3 4 2 4 2 2 2 5 2 2 1 3 3 3 2 2 2 2 3 4 4
## [1703] 4 4 2 1 3 3 3 3 4 4 4 1 3 3 4 4 4 2 4 4 4 2 3 4 3 4 2 2 1 1 3 3 3 4 4 1 3
## [1740] 3 3 4 4 2 4 4 2 1 4 3 3 4 4 2 2 2 4 3 3 4 4 4 2 2 3 4 4 4 4 4 4 2 4 3 3 4
## [1777] 4 4 3 4 4 2 4 4 2 4 2 3 4 4 2 3 2 4 4 3 3 4 4 3 2 2 2 3 3 3 3 4 3 4 2 2 4
## [1814] 4 2 3 2 3 2 2 2 2 3 3 3 4 2 4 2 4 1 3 4 4 4 2 2 4 3 3 4 4 4 2 3 4 4 4 4 4
## [1851] 4 1 3 4 3 4 4 2 4 3 3 4 2 4 4 2 2 2 4 1 3 3 3 3 4 4 4 4 3 3 4 4 2 3 4 4 4
## [1888] 4 2 4 2 4 4 2 3 3 3 4 4 4 3 3 4 4 4 4 2 1 3 3 4 4 2 2 3 3 3 3 3 3 4 2 1 3
## [1925] 3 3 3 4 2 4 4 2 4 3 3 4 3 3 4 2 2 3 4 3 4 2 2 4 2 3 3 4 4 4 2 2 1 3 3 3 2
## [1962] 4 2 4 1 3 3 3 4 4 2 4 1 3 4 4 3 3 4 4 2 2 3 4 4 4 4 2 2 3 4 4 4 2 4 3 3 4
## [1999] 4 4 4 4 3 2 2 3 4 4 4 4 4 4 2 2 3 4 3 4 4 4 2 1 3 4 3 2 4 2 3 3 4 4 2 4 4
## [2036] 3 3 4 4 4 2 2 4 3 4 4 4 3 4 4 2 4 4 4 3 4 2 3 3 3 4 4 2 2 2 1 1 3 3 4 3 4
## [2073] 4 4 4 4 2 4 4 4 4 4 4 4 1 3 3 4 4 4 2 4 3 3 3 3 4 4 4 2 2 3 3 3 3 4 1 3 4
## [2110] 4 4 2 2 2 4 1 3 3 4 4 2 2 4 4 3 3 4 4 2 3 3 4 4 2 3 4 4 4 4 4 4 4 4 2 3 4
## [2147] 4 4 2 1 3 4 4 4 4 2 2 2 2 3 3 2 2 2 4 2 4 3 3 3 4 4 4 1 3 2 4 4 4 4 2 2 2
## [2184] 3 3 4 3 4 4 4 4 4 1 3 3 4 4 4 4 4 4 2 3 4 4 4 4 4 4 2 3 3 3 3 4 4 4 4 4 3
## [2221] 4 4 4 4 2 2 3 3 4 4 2 2 4 2 4 3 3 3 4 4 4 1 1 3 4 4 4 2 2 2 2 2 3 3 3 4 2
## [2258] 4 4 3 3 4 4 4 4 4 4 3 3 4 2 2 2 2 5 3 3 4 4 4 2 4 4 3 4 4 4 4 4 2 2 4 3 4
## [2295] 2 4 2 2 2 4 4 3 3 4 4 4 4 4 4 4 4 3 4 4 3 4 4 2 1 3 3 4 4 2 4 2 2 3 4 2 2
## [2332] 2 3 3 4 4 4 3 2 2 2 3 3 4 4 4 4 2 2 2 3 3 3 4 4 2 4 2 4 3 3 4 4 4 2 3 4 3
## [2369] 4 4 4 4 2 4 4 2 4 2 2 2 2 2 2 3 3 3 4 2 2 2 3 3 4 2 2 4 3 3 4 4 4 4 4 2 4 3
## [2406] 4 3 2 4 4 1 3 3 4 4 4 4 4 4 4 3 4 2 4 2 3 3 4 2 4 2 3 3 4 3 4 4 4 2 2 4 3
## [2443] 4 3 4 4 4 4 4 3 3 3 2 2 3 3 4 2 4 4 2 4 3 4 4 4 3 4 4 4 4 2 4 3 3 2 4 3 3
## [2480] 4 4 4 2 1 1 3 4 3 4 2 2 3 3 4 4 3 2 3 3 4 2 2 4 2 2 2 3 3 4 4 4 2 4 4 4 2
## [2517] 3 3 4 4 2 4 4 3 3 3 4 4 3 2 4 4 4 3 3 4 4 4 4 2 2 1 1 3 3 4 2 2 1 1 3 3 4
## [2554] 4 4 4 3 4 3 4 4 2 4 4 4 3 4 4 3 4 4 4 4 3 3 4 4 4 4 4 4 2 3 3 3 4 4 4 2 1
## [2591] 3 3 4 4 4 4 2 3 3 4 3 3 4 4 4 4 3 3 4 4 2 2 4 1 3 4 4 4 2 2 2 4 1 3 3 3 4
## [2628] 4 1 3 3 3 4 4 4 2 1 2 2 2 2 4 2 4 2 4 2 1 3 4 3 4 4 2 4 1 3 3 4 3 4 4 3 3
## [2665] 4 4 4 4 2 4 3 3 4 2 4 2 2 2 1 3 4 4 4 4 4 1 2 3 3 4 4 4 4 2 4 1 3 3 4 4 4
## [2702] 2 2 2 3 3 3 4 4 2 3 4 2 4 4 4 2 2 4 2 2 3 3 3 4 4 2 4 1 1 3 4 3 2 2 2 2 1
## [2739] 3 3 4 2 2 4 2 3 4 4 4 4 4 4 4 2 4 4 3 3 3 2 4 4 4 4
##
## Within cluster sum of squares by cluster:
## [1]  737.6141 1248.0870 1849.5824 2624.4456  722.9391
##  (between_SS / total_SS =  56.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Looks like this time, cluster 1 is all below average and cluster 5 is all above.
Which cluster is Cam Mack in?

```
playeradvcluster <- data.frame(playersadvanced, advk5$cluster)
```

Cluster 5 on my dataset. So three games in, we can say he's in a big group of players who are all above average on these advanced metrics.

Now who are his Big Ten peers?

```
playeradvcluster %>%
  filter(advk5.cluster == 5) %>%
  filter(Team %in% big10) %>%
  arrange(desc(PProd))
```

```
##  [1] Player        Team          Pos           PER           TS.
##  [6] PProd         AST.          WS.40         BPM           advk5.cluster
## <0 rows> (or 0-length row.names)
```

Sorting on Points Produced, Cam Mack is eighth out of the 31 guards in the Big Ten who land in Cluster 5.

It's early goings, but watch this player. He's fun to watch and the stats back it up.

Chapter 32

Rtweet and Text Analysis

By Collin K. Berke, Ph.D.

One of the best rivalries in college volleyball was played on Saturday, November 2, 2019. The seventh-ranked Penn State Nittany Lions (16-3) took on the eigth-ranked Nebraska Cornhuskers (16-3). This match featured two of the best middle blockers in the country, Nebraska's Lauren Stivrins and Penn State's Kaitlyn Hord. Stivrins was ranked No. 1 in Big Ten hitting percentage, a .466 before the match up. Hord, close behind, had a .423 hitting percentage and was ranked towards the top as one of the league's top blockers.

Alongside being a competition between premier players, this match was set to be a battle between two of the winningest coaches in NCAA Women's Volleyball history. Russ Rose, head coach of the Nittany Lions, came into the match with a 1289-209 (.860) record, 17 Big Ten Conference Championships, and 7 NCAA National Championships. For the Nebraska Cornhuskers' head coach, John Cook came into the match with a 721-148 (.830) record, 9 Big 12 Conference Championships, 4 Big Ten Conference Championships, and 5 NCAA National Championships. Check out this article here to get a better understanding of the significance of this game and rivalry.

Being a contest between two storied programs, premier players, and two of the most winningest coaches in NCAA volleyball history, this match was poised to be one of the premier Big Ten matches of the 2019 season. If history was to serve as a guide, this match would easily go into five exciting, nail-biting sets.

To no surprise–it did. Nebraska came out victorious, 3 sets to 2, winning the fifth set 15 - 13. Although we have commentators, analysts, and reporters to tell us the story of the game, wouldn't it be interesting to tell the story from the fan's perspective? Can what they say allow us to take a pulse of how the fan base feels during the game? We can answer this question using Twitter tweet data, which we will access with the `rtweet` package.

Question - How do people feel during a game? Positive? Negative? Neutral?

This chapter will teach you how to extract, analyze, and visualize Twitter text data to tell a story about peoples sentiments toward any sport team, player, or event. Although Twitter is conventionally thought of as a social media platform, at a general level, it can be thought of as a corpus of textual data, which is generated by millions of users, talking about a wide array of topics over time.

During this chapter, we will access text data held within the body of tweets, which we will extract and import into R through the use of an API (application programming interface). This can seem like a pretty technical term, but all it really is is a portal to which data can be shared between computers and humans. NPR has an API 101 post on their site, which you can read to get a rough idea of what an API is and how they are used.

In fact, many news organizations provide APIs for people to access and use their data. For example, many news services like The New York Times, NPR, The Associated Press and social media platforms like Facebook have APIs that can be used to access content or varying types of data. Many of these just require you to: a). have a developer account; b) have the proper API keys; and c). use their API in accordance with their terms of service. Every API you come across should have documentation outlining its use.

Above was a pretty hand-wavy explanation of APIs. Indeed, APIs have many different uses beyond just extracting data, but such a discussion is beyond the scope of this chapter. Nevertheless, APIs can be a powerful, useful tool to access data not normally available on web pages or other statistical reporting services.

32.1 Prerequisites

You will need to have a Twitter account to access and extract data. If needed, you can sign up for an account here.

32.1.1 Tools for text analysis

This chapter will also require you to load and acquaint yourself with functions in four packages, `rtweet`, `lubridate`, `stringr`, and `tidytext`. You may have used some of functions in other portions of this class. Others may be new to you.

- `rtweet` is a R package used to access Twitter data via the Twitter API.
- `lubridate` is a package that makes working with dates and times a bit easier.
- `stringr` is a package that provides several functions to make working with string data a little easier.

- `tidytext` is a package used to tidy, analyze, and visualize textual analyses. We will use this package to calculate tweet sentiments (e.g., positive and negative feelings).

This chapter will also use other packages you have gained familiarity with throughout the class: `dplyr` and `ggplot2`. To install these packages and load them for use in our analysis session, run the following code:

```
install.packages("rtweet") # installs the rtweet package

install.packages("tidytext") # installs the tidytext package

install.packages("tidyverse") # A collection of packages, includes the stringr packages

install.packages("lubridate") # Provides functions to make working with dates/times easier
```

```
# Load the packages to be used in your analysis session

library(rtweet)

library(tidytext)

library(tidyverse)

library(lubridate)

library(ggrepel)
```

32.1.2 Working with string data

String data is just basically letters, words, symbols, and even emojis. Take for example the following tweet:

```
knitr::include_graphics(rep("images/volleyballTweet.png"))
```

Everything contained in the message portion of the tweet is string data, even the emojis. When it comes to emojis, most have a special textual code that is

rendered by a browser or device that gets displayed as an image. For example, the ear of corn emoji is actually written as `:corn:`, but it gets rendered as an image when we view the tweet on our computers/devices. We can extract, analyze, and visualize this string data to tell a wide range of stories from users' tweets. Our goal being to show sentiment over the length of a Husker volleyball Match and football game.

32.2 Verifying your account to access Twitter data

Before you can access Twitter data, you will need to verify your account. The `rtweet` package makes this really easy to do. You will first need to run one of the package's functions for it to walk you through the authentication process. To do this, let's just search for the most recent 8000 (non-retweeted) tweets containing the **#huskers** and **#GBR** hashtags.

Before you run the following code chunk, though, be aware a few things will take place. First, a browser window will open up asking you to verify that `rtweet` is allowed to access Twitter data via the API on behalf of your account. Accept this request and enter your credentials if you are asked to. Once you do this, you should get a message in your browser stating you have successfully authenticated the `rtweet` package. The data will then begin to download. The amount of time needed to import this data will depend on how many tweets the hashtag(s) are associated with. More tweets generally means longer import times.

```
huskers <- search_tweets(
  "#huskers", n = 8000, include_rts = FALSE
)

gbr <- search_tweets(
  "#GBR", n = 8000, include_rts = FALSE
)
```

Important Note: Depending on when you run the above code chunk, the API will return different data then the data used for the examples later in this chapter. This is due to the query rate cap Twitter places on it's API. Twitter's API caps queries to 18,000 of the most recent tweets during the past couple of days. This cap resets every 15 minutes. The `rtweet` package does has functionality to pull data once your query limit resets. However, if you're looking to pull tweets for a very popular event (e.g., The Super Bowl), you may want to consider other options to extract this type of data. This is also important to understand because if you are looking to pull tweets for a specific event, you will need to make sure you are pulling this data within a reasonable time during or after the event. If you don't, these rate limits might not allow you access the data you need to do your analysis.

The data we will use later for the examples in the chapter can be found here and here. The first data set are tweets that use the #huskers hashtag. The second has data of tweets that use the #gbr hashtag. You will need to download both data sets, put them in the right directory, and import both for the below examples to work correctly. The code to import this data will look something like this:

```
huskerTweets <- read_csv("data/huskerTweets.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   created_at = col_datetime(format = ""),
##   display_text_width = col_double(),
##   is_quote = col_logical(),
##   is_retweet = col_logical(),
##   favorite_count = col_double(),
##   retweet_count = col_double(),
##   quote_count = col_logical(),
##   reply_count = col_logical(),
##   symbols = col_logical(),
##   ext_media_type = col_logical(),
##   quoted_created_at = col_datetime(format = ""),
##   quoted_favorite_count = col_double(),
##   quoted_retweet_count = col_double(),
##   quoted_followers_count = col_double(),
##   quoted_friends_count = col_double(),
##   quoted_statuses_count = col_double(),
##   quoted_verified = col_logical(),
##   retweet_status_id = col_logical(),
##   retweet_text = col_logical(),
##   retweet_created_at = col_logical()
##   # ... with 21 more columns
## )
```

```
## See spec(...) for full column specifications.
```

```
gbrTweets <- read_csv("data/gbrTweets.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   created_at = col_datetime(format = ""),
##   display_text_width = col_double(),
##   is_quote = col_logical(),
##   is_retweet = col_logical(),
##   favorite_count = col_double(),
##   retweet_count = col_double(),
```

```
##   quote_count = col_logical(),
##   reply_count = col_logical(),
##   symbols = col_logical(),
##   ext_media_type = col_logical(),
##   quoted_created_at = col_datetime(format = ""),
##   quoted_favorite_count = col_double(),
##   quoted_retweet_count = col_double(),
##   quoted_followers_count = col_double(),
##   quoted_friends_count = col_double(),
##   quoted_statuses_count = col_double(),
##   quoted_verified = col_logical(),
##   retweet_status_id = col_logical(),
##   retweet_text = col_logical(),
##   retweet_created_at = col_logical()
##   # ... with 21 more columns
## )
## See spec(...) for full column specifications.
```

```
## Warning: 1 parsing failure.
##  row     col           expected actual                 file
## 4365 symbols 1/0/T/F/TRUE/FALSE    GBR 'data/gbrTweets.csv'
```

This brings up a good point about saving any data you import from Twitter's API. **Always save your data.** Remember those rate limits? If you don't save your data and too many days pass, you will not be able to access that data again. To do this, you can use the `write_as_csv()` function from the `rtweet` package to save a `.csv` file of your data. The code to do this will look something like this:

```
write_as_csv(huskerTweets, "data/huskerTweets.csv")
```

Be aware that this function will overwrite data. If you make changes to your `huskerTweets` object and then run the `write_as_csv()` function again, it will overwrite your saved file with the modifications you made to your object. The lesson then is to always save an extra copy of your data in a separate directory, just in case you do accidentally make a mistake in overwriting your data.

32.3 The data used here

To provide a little context, I pulled the data in this chapter on Sunday, November 3, 2019. This was the day after Nebraska Football lost to Purdue, and Nebraska Volleyball won against Penn State. You can follow the steps above to download this data for the following examples.

To make it easier to work with, I am going to combine these two data sets into one using the `bind_rows()` function from `dplyr`. There is a slight problem

though, some people may have had a tweet that contained both the **#huskers** and **#GBR** hashtags in their tweet. So if we combine these two data sets, there might be duplicate data. To dedupe the data, we can apply a `distinct(text, .keep_all = TRUE)` to remove any duplicates. The `.keep_all = TRUE` argument just tells R to keep all columns in the data frame after our data has been deduped.

```
tweet_data <- bind_rows(huskerTweets, gbrTweets) %>%
  distinct(text, .keep_all = TRUE)
```

32.4 Data Exploration

Let's explore the data a bit. Run a `glimpse(tweet_data)` to get a view of what data was returned from twitter. My query on November 3rd, 2019 returned 11,523 non-retweeted tweets from 3,917 accounts using the `#huskers` and/or the `#GBR` hashtag within the tweet's body (again if you ran the code above, your data will be different) .

It's important to remember these tweets can come from accounts that are people, organizations, and even bots. So when drawing conclusions from this data, make sure to keep in mind that these tweets may not represent the sentiment of just one person. Additionally, it is important to remember that not all fans of a sports team are on or use Twitter, so it surely is not a valid representation of all fan sentiment. Indeed, you could also have fans of other teams using your hashtags.

```
glimpse(tweet_data)
```

```
## Observations: 11,523
## Variables: 90
## $ user_id                 <chr> "x17636179", "x17636179", "x17636179", "x15...
## $ status_id               <chr> "x1191100304940396544", "x11908526509059440...
## $ created_at              <dttm> 2019-11-03 21:10:16, 2019-11-03 04:46:11, ...
## $ screen_name             <chr> "SeanKeeler", "SeanKeeler", "SeanKeeler", "...
## $ text                    <chr> "ICYMI, #CSURams fans, a recap of @denverpo...
## $ source                  <chr> "Twitter Web App", "Twitter for iPhone", "T...
## $ display_text_width      <dbl> 269, 188, 182, 122, 135, 189, 172, 135, 94,...
## $ reply_to_status_id      <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_to_user_id        <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_to_screen_name    <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ is_quote                <lgl> FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FA...
## $ is_retweet              <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F...
## $ favorite_count          <dbl> 0, 0, 1, 116, 19, 2, 40, 23, 149, 5, 31, 15...
## $ retweet_count           <dbl> 0, 0, 0, 6, 1, 0, 1, 2, 5, 1, 3, 1, 0, 0, 0...
## $ quote_count             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ reply_count             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
```

```
## $ hashtags                <chr> "CSURams Huskers ProudToBe AtThePeak CSU", ...
## $ symbols                 <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ urls_url                <chr> "tinyurl.com/y2b3kog9 tinyurl.com/yy7ntps4"...
## $ urls_t.co               <chr> "https://t.co/1FlAnRRgEq https://t.co/t8j1L...
## $ urls_expanded_url       <chr> "https://tinyurl.com/y2b3kog9 https://tinyu...
## $ media_url               <chr> NA, NA, NA, "http://pbs.twimg.com/ext_tw_vi...
## $ media_t.co              <chr> NA, NA, NA, "https://t.co/FSCP827hg0", "htt...
## $ media_expanded_url      <chr> NA, NA, NA, "https://twitter.com/HuskerSpor...
## $ media_type              <chr> NA, NA, NA, "photo", "photo", "photo", "pho...
## $ ext_media_url           <chr> NA, NA, NA, "http://pbs.twimg.com/ext_tw_vi...
## $ ext_media_t.co          <chr> NA, NA, NA, "https://t.co/FSCP827hg0", "htt...
## $ ext_media_expanded_url  <chr> NA, NA, NA, "https://twitter.com/HuskerSpor...
## $ ext_media_type          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ mentions_user_id        <chr> "x8216772", "x8216772", "x24725032", "x1210...
## $ mentions_screen_name    <chr> "denverpost", "denverpost", "DPostSports", ...
## $ lang                    <chr> "en", "en", "en", "en", "en", "en", "en", "...
## $ quoted_status_id        <chr> NA, "x1190803840213209088", NA, NA, NA, NA,...
## $ quoted_text             <chr> NA, "From Nebraska to Fort Collins, how CSU...
## $ quoted_created_at       <dttm> NA, 2019-11-03 01:32:14, NA, NA, NA, NA, N...
## $ quoted_source           <chr> NA, "TweetDeck", NA, NA, NA, NA, NA, NA, NA...
## $ quoted_favorite_count   <dbl> NA, 5, NA, NA, NA, NA, NA, NA, NA, NA, 194,...
## $ quoted_retweet_count    <dbl> NA, 1, NA, NA, NA, NA, NA, NA, NA, NA, 16, ...
## $ quoted_user_id          <chr> NA, "x24725032", NA, NA, NA, NA, NA, NA, NA...
## $ quoted_screen_name      <chr> NA, "DPostSports", NA, NA, NA, NA, NA, NA, ...
## $ quoted_name             <chr> NA, "Denver Post Sports", NA, NA, NA, NA, N...
## $ quoted_followers_count  <dbl> NA, 34841, NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ quoted_friends_count    <dbl> NA, 395, NA, NA, NA, NA, NA, NA, NA, NA, 60...
## $ quoted_statuses_count   <dbl> NA, 113697, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ quoted_location         <chr> NA, "Denver, Colorado", NA, NA, NA, NA, NA,...
## $ quoted_description      <chr> NA, "Sports news & analysis from @denverpos...
## $ quoted_verified         <lgl> NA, TRUE, NA, NA, NA, NA, NA, NA, NA, NA, F...
## $ retweet_status_id       <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_text            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_created_at      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_source          <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_favorite_count  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_retweet_count   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_user_id         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_screen_name     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_name            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_followers_count <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_friends_count   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_statuses_count  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_location        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_description     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ retweet_verified        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
```

```
## $ place_url               <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_name              <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_full_name         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ place_type              <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ country                 <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ country_code            <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ geo_coords              <chr> "NA NA", "NA NA", "NA NA", "NA NA", "NA NA"...
## $ coords_coords           <chr> "NA NA", "NA NA", "NA NA", "NA NA", "NA NA"...
## $ bbox_coords             <chr> "NA NA NA NA NA NA NA NA", "NA NA NA NA NA ...
## $ status_url              <chr> "https://twitter.com/SeanKeeler/status/1191...
## $ name                    <chr> "Sean Keeler", "Sean Keeler", "Sean Keeler"...
## $ location                <chr> "Denver, CO", "Denver, CO", "Denver, CO", "...
## $ description             <chr> "@DenverPost staffer, dad, husband, drummer...
## $ url                     <chr> "https://t.co/z0eFbv9eaz", "https://t.co/z0...
## $ protected               <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F...
## $ followers_count         <dbl> 5245, 5245, 5245, 30451, 30451, 30451, 3045...
## $ friends_count           <dbl> 1619, 1619, 1619, 701, 701, 701, 701, 701, ...
## $ listed_count            <dbl> 296, 296, 296, 247, 247, 247, 247, 247, 247...
## $ statuses_count          <dbl> 28124, 28124, 28124, 12801, 12801, 12801, 1...
## $ favourites_count        <dbl> 4498, 4498, 4498, 6033, 6033, 6033, 6033, 6...
## $ account_created_at      <dttm> 2008-11-25 23:53:13, 2008-11-25 23:53:13, ...
## $ verified                <lgl> FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE...
## $ profile_url             <chr> "https://t.co/z0eFbv9eaz", "https://t.co/z0...
## $ profile_expanded_url    <chr> "http://www.seankeeler.tumblr.com", "http:/...
## $ account_lang            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,...
## $ profile_banner_url      <chr> "https://pbs.twimg.com/profile_banners/1763...
## $ profile_background_url  <chr> "http://abs.twimg.com/images/themes/theme10...
## $ profile_image_url       <chr> "http://pbs.twimg.com/profile_images/118130...
```

As you can see, `glimpse()` returns a lot of columns that are not really relevant to our analysis. Let's apply a `select()` function to only retain the data relevant to our analysis, .

```
tweet_data <- tweet_data %>%
  select(user_id, status_id, created_at, screen_name, text, display_text_width,
         favorite_count, retweet_count, hashtags, description, followers_count)
```

32.5 Cleaning data for analysis

If you examine the data set, you will see this data needs some wrangling. First, we need to fix the `created_at` variable. Right now it is represented in Greenwich Mean Time (GMT), but we need it to be in Central Standard Time (CST). We do this so we can make sense of when during the game things happened. Second, the data is outside of the time frame we are interested in examining, so we need to filter the data to be windowed during the time of the game. We will filter the data by date and time, examining tweets a little before and after the game.

32.5.1 Fixing the date and focusing only on game tweets

We will use the `with_tz()` on our `created_at` variable within our `mutate()` function to transform the `created_at` column into Central Standard Time (CST). We do this by setting the `tzone` argument to `"America/Chicago"`. Once out time is adjusted, we need group tweets within a specific bin of time. For this example I have decided to bin tweets to the nearest 5 minute mark. We can do this by using the `round_time()` function provided to us by the `lubridate` package.

Then, since we are only interested in tweets during the game, we can apply a `dplyr` `filter()` function to window our data set to tweets being posted around the start and end of the game.

```
volleyball_tweets <- tweet_data %>%
  mutate(created_at = with_tz(created_at, tzone = "America/Chicago"),
         created_at = round_time(created_at, "5 mins", tz = "America/Chicago")) %>%
  filter(created_at >= "2019-11-02 18:30:00" & created_at <= "2019-11-02 23:30:00")
```

32.6 Number of tweets throughout the game

One question we might have pertains to the number of tweets that occur during the course of the match. To do this, all we need to do is `group_by()` our tweets by our `created_at` variable, and then use the `count()` function to count the number of tweets within each five minute bin. We then use `ggplot` to plot a line chart where `created_at` is placed on the x-axis and `n`, number of tweets, is placed on the y-axis.

```
start_time <- tibble(time = as_datetime("2019-11-02 19:35:00", tz = "America/Chicago")

volleyball_time <- volleyball_tweets %>%
  group_by(created_at) %>%
  count()

ggplot() +
  geom_line(data = volleyball_time, aes(x = created_at, y = n)) +
  geom_point(data = start_time, aes(x = time, y = 5), color = "red") +
  geom_text_repel(data = start_time, aes(x = time, y = 3, label = label), nudge_x = -2
  labs(y = "Number of Tweets",
       x = "Central Standard Time (CST)") +
  theme_minimal() +
  theme(axis.title.x = element_blank())
```

There you have it. A trend line plotting tweet volume throughout the course of the event. Do you see any areas where the match might have had a significant number of tweets?

32.7 Tidying the text data for analysis, applying the sentiment scores

Okay, that's cool–but what we really want to know is what are peoples' sentiments throughout the game? Did they feel positive or negative throughout the event? Were there times that were more positive or negative? To achieve this, we are going to use the `tidytext` package to tidy up our text data and apply a sentiment score to each word held within each tweet. Let's break this down step-by-step.

First, we need to get the dictionary that contains the sentiment scoring for thousands of words used in the English language. `afinn <- get_sentiments("afinn")` does just that for us. The development of these sentiment dictionaries is beyond this chapter. However, most of these dictionaries are crowd sourced by having people provide self-responses on how positive or negative a word is to them. For now, just understand the `affin` variable contains many words that have been rated for how positive or negative a word is on a scale that ranges from -5 to 5. -5 being the most negative, and 5 being the most positive. If you want to learn more about this dictionary or others, you can read more about them here.

Second, now that we have our dictionary imported, we need to clean up our tweets data set so we can apply sentiment scores to each word used within each tweet. There's one problem, though. Each row in our data set is a complete tweet. For us to apply a sentiment score for each word, each word needs to get its own row. This is where the `unnest_tokens()` function from the `tidytext` package comes into play. We use this function to create a data set that will create a new column called word, which will place every word from every tweet in our data set on its own row, which it knows which text data to this because we set the the second argument to the column name that holds our text data. In this case, we give it the `text` column. Once you run this code, if you look at the `volleyball_tweets_tidy` object, you should now have a data set where every row has its own word which was done for every tweet. This data frame should now be a super long data frame.

Lastly, the English language has many words that really don't mean anything in regards to sentiment. Take for example the word 'the'. This article really doesn't represent a positive or negative sentiment. Thus, these types of words need to be taken out of our data set to enhance improve the accuracy of our analysis. To do this, we will apply the `anti_join(stop_words)` to our `dplyr` chain. All this does is get rid of the stop words in our data set that really don't contribute to the sentiment scores we are eventually going to calculate.

If you get an error on the next bit of code, you'll likely need to install the textdata package on the console with `install.packages("textdata")`. Next, if this next block of code hangs, it's because in the console you're being asked if you want to download some data. You do indeed want to do that, so type 1 and hit enter.

```
afinn <- get_sentiments("afinn")

volleyball_tweets_tidy <- volleyball_tweets %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

32.7.0.1 Fan sentiment over the game

Now that we have a tidied textual data set, all we need to do is apply our sentiment scores to these words using the `inner_join()`, then `group_by()` our `created_at` variable, and calculate the mean sentiment for each five minute interval. At this point, we will use `ggplot` to plot sentiment of the tweets over time. We will do this by plotting the `created_at` variable on the x-axis and the newly calculated `sentiment` variable on the y-axis. The rest is just adding annotations and styling, which we are already familiar with.

```
volleyball_tweets_sentiment <- volleyball_tweets_tidy %>%
  inner_join(afinn) %>%
```

```
  group_by(created_at) %>%
  summarise(sentiment = mean(value))
```

```
## Joining, by = "word"
```

```
ggplot() +
  geom_line(data = volleyball_tweets_sentiment, aes(x = created_at, y = sentiment)) +
  geom_text(aes(x = as_datetime("2019-11-02 19:00:00", tz = "America/Chicago"), y = 3), color = '
  geom_text(aes(x = as_datetime("2019-11-02 19:00:00", tz = "America/Chicago"), y = -3), color =
  labs(title = "People's Sentiment on Twitter Positive Towards\nHusker Volleyball's Win Against P
       subtitle = "Sentiment mostly positive throughout the game",
       caption = "Source: #huskers and #GBR Tweets, 2019-11-02 | By Collin K. Berke",
       y = "Sentiment",
       x = "Central Standard Time (CST)") +
  scale_y_continuous(limits = c(-4, 4)) +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.text = element_text(size = 7)
        )
```

When looking at this trend line, you can see that during the volleyball game, tweets using the #husker and #gbr hashtags had some wide variation in sentiment. Overall it seems that tweets during the volleyball match were mostly positive, where at times it dipped negative. Why might this be the case? Well,

unfortunately, even though this was a big game for Husker volleyball, not many people were tweeting during the match (take a look at the number of tweets chart above). So if there was one word used in a tweet that was ranked as very negative in sentiment, it would have easily drove our average sentiment into the negative region quickly.

Also, we need to consider that this match took place after the Huskers loss to Purdue, which we will examine in the next example. This is important to know because people during the volleyball match may have also been tweeting about how poorly the football game went earlier in the day. Thus, low tweet volume mixed with the potential for tweets referencing something other than the match at hand may have had some influence on the sentiment scores.

There's also one last thing to keep in mind when you draw conclusions from this type of text data. Language is complex–it can have multiple meanings, which is highly influenced by context. Take for example the word 'destroy', like its use in the following statement: "This team is going to destroy the defense today." Although we clearly can see this is a positive statement, when a computer applies sentiment scores, the context of the statement is stripped away, and destroy will be scored as negative sentiment. In short, computers are not smart enough to include context when they calculate sentiment, yet. So, keep this limitation in mind when you draw conclusions from your sentiment analyses using text data.

32.7.1 Example 2 - Nebraska's loss to Purdue, what were fan's sentiments towards this loss?

The Nebraska Cornhuskers–a 3-point favorite going into West Lafayette, IN–squared off with the Purdue Boilermakers on November 2, 2019. Purdue was 2-6 on the season. Nebraska, with a 4-4 record coming off of a 38-31 home loss to Indiana, had many fans hoping Scott Frost could lead his team to a must needed win. Especially given the expectation was the Cornhuskers would go 6-6 on the season, and the team still had to play Wisconsin (6-2), Maryland (3-6), and Iowa (6-2) to get to those needed six wins to become bowl eligible. So, how did people particularly take this loss? Let's use our Twitter data to get an answer.

Again, we need to fix the time zone with the `with_tz()` function so the data is represented in Central Standard Time (CST). Then we apply our `filter()` command to window our data to when the game was taking place.

```
football_tweets <- tweet_data %>%
  mutate(created_at = with_tz(created_at, tzone = "America/Chicago"),
         created_at = round_time(created_at, "5 mins", tz = "America/Chicago")) %>%
  filter(created_at >= "2019-11-02 11:00:00" & created_at <= "2019-11-02 15:30:00")
```

Now, let's just get a sense of the number of tweets that occurred at certain points in the game. We again need to do some data wrangling with `group_by()`, and then we use the `count()` function to add up all the tweets during each five

minute interval. Once the data is wrangled, we can use our `ggplot` code to visualize tweet volume throughout the game.

```
football_time <- football_tweets %>%
  group_by(created_at) %>%
  count()

ggplot() +
  geom_line(data = football_time, aes(x = created_at, y = n)) +
  theme_minimal() +
  labs(y = "Number of Tweets")
```

Looking at this plot, we can see the tweet volume is a lot higher than that of the volleyball match. In fact, it looks like towards the end of the game there was a five minute interval where ~80 or so tweets occurred. Given the outcome of the game, I assume people were not real happy during this spike in activity. Well we have the tools to answer this question.

As before, let's pull in our sentiment library with the `get_sentiments()` function. Then lets tidy up our tweet data using the `unnest_tokens()` and `anti_join(stop_words)`. Remember this step just places every word within a tweet on its own row and filters out any words that don't have any real meaning to the calculation of sentiment (i.e., and, the, a, etc.).

```
afinn <- get_sentiments("afinn")

football_tweets_tidy <- football_tweets %>%
```

```
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

We now have our clean textual data, let's apply sentiment scores for each word using the `inner_join(affin)` function, `group_by(created_at)` to create a group for each five minute interval, and then use `summarise()` to calculate then mean sentiment for each time period.

You can then use the `ggplot` code to plot sentiment over time, introduce annotations to highlight specific aspects within our plot, and then apply styling to the plot to move it closer to publication readiness. Now for the big reveal, how did people take the loss to a 2-6 Purdue? Run the code and find out.

```
football_tweets_sentiment <- football_tweets_tidy %>%
  inner_join(afinn) %>%
  group_by(created_at) %>%
  summarise(sentiment = mean(value)) %>%
  arrange(sentiment)
```

```
## Joining, by = "word"
```

```
ggplot() +
  geom_line(data = football_tweets_sentiment, aes(x = created_at, y = sentiment)) +
  geom_text(aes(x = as_datetime("2019-11-02 14:30:00", tz = "America/Chicago"), y = 2)
  geom_text(aes(x = as_datetime("2019-11-02 14:30:00", tz = "America/Chicago"), y = -2
  scale_y_continuous(limits = c(-2.5, 2.5)) +
  labs(title = "People's Sentiment on Twitter Negative\nTowards Husker Football's Loss
       subtitle = "People were positive at the start and part of the first half,\nthen
       caption = "Source: #huskers and #GBR Tweets, 2019-11-02 | By Collin K. Berke",
       y = "Sentiment",
       x = "Central Standard Time (CST)") +
  theme_minimal() +
  theme(axis.title.x = element_blank(),
        plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 10),
    axis.text = element_text(size = 7)
  )
```

As you can see, it started out pretty positive. Then, it started to go negative throughout the first half. However, there was a bump, which was around the time D-Lineman, Darrion Daniels almost scored a pick six. Around halftime, we can see a little bit of a bump towards positive sentiment. This was probably most likely due to people cheering on the Huskers to come out strong after the half. As the second half progressed, you can see things turned for the worst again, and sentiment became negative up until the end of the game, most likely because people realized they were going to get another L on the schedule. You can relive all this excitement again by catching the game recap here.

Chapter 33

Arranging multiple plots together

Sometimes you have two or three (or more) charts that are really just one chart that you need to merge them together. It would be nice to be able to arrange them programmatically and not have to mess with it in illustrator.

Good news.

There is.

It's called `cowplot`, and it's pretty easy to use. First install cowplot with install.packages("cowplot"). Then let's load tidyverse and cowplot.

```
library(tidyverse)
library(cowplot)
```

What follows is just stuff for me to set up a couple of bar charts. You can run it – it'll work on your machine without changing a thing – but what I'm doing here isn't important. The stuff you need to do is below.

```
attendance <- read_csv("https://raw.githubusercontent.com/mattwaite/sportsdatabook/Master/data/at
```

```
## Parsed with column specification:
## cols(
##   Institution = col_character(),
##   Conference = col_character(),
##   `2013` = col_double(),
##   `2014` = col_double(),
##   `2015` = col_double(),
##   `2016` = col_double(),
##   `2017` = col_double(),
##   `2018` = col_double()
```

```
## )
```

Making a quick percent change.

```
attendance <- attendance %>% mutate(change = ((`2018`-`2017`)/`2017`)*100)
```

Let's chart the top 10 and bottom 10 of college football ticket growth … and shrinkage.

```
top10 <- attendance %>% top_n(10, wt=change)
bottom10 <- attendance %>% top_n(10, wt=-change)
```

Okay, now to do this I need to save my plots to an object. We do this the same way we save things to a dataframe – with the arrow. We'll make two identical bar charts, one with the top 10 and one with the bottom 10.

```
bar1 <- ggplot() + geom_bar(data=top10, aes(x=reorder(Institution, change), weight=char
```

```
bar2 <- ggplot() + geom_bar(data=bottom10, aes(x=reorder(Institution, change), weight=c
```

With cowplot, we can use a function called `plot_grid` to arrange the charts:

```
plot_grid(bar1, bar2)
```

We can also stack them on top of each other:

```
plot_grid(bar1, bar2, ncol=1)
```

To make these publishable, we should add headlines, chatter, decent labels, credit lines, etc. But to do this, we'll have to figure out which labels go on which charts, so we can make it look decent. For example – both charts don't need x or y labels. If you don't have a title and subtitle on both, the spacing is off, so you need to leave one blank or the other blank. You'll just have to fiddle with it until you get it looking right.

```
bar1 <- ggplot() + geom_bar(data=top10, aes(x=reorder(Institution, change), weight=change)) + coo
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 8),
    plot.subtitle = element_text(size=10),
    panel.grid.minor = element_blank()
    )
```

```
bar2 <- ggplot() + geom_bar(data=bottom10, aes(x=reorder(Institution, change), weight=change)) +
  theme(
    plot.title = element_text(size = 16, face = "bold"),
    axis.title = element_text(size = 8),
    plot.subtitle = element_text(size=10),
    panel.grid.minor = element_blank()
    )
```

Saving a cowplot plot_grid is the same as anything else we did in the class:

```
plot_grid(bar1, bar2) + ggsave("test.png")
```

```
## Saving 6.5 x 4.5 in image
```

Chapter 34

Assignments

This is a collection of assignments I've used in my Sports Data Analysis and Visualization course at the University of Nebraska-Lincoln. The overriding philosophy is to have students do lots of small assignments that directly apply what they learned, and often pulling from other assignments. Each small assignment is just a few points each – I make them 5 points each and make the grading a yes/no decision on 5 different questions – so a bad grade on one doesn't matter. Then, twice during the semester, I have them create blog posts with visualizations on a topic of their choosing. The topic must have a point of view – Nebraska's woes on third down are why the team is struggling, for example – and must be backed up with data. They have to write a completely documented R Notebook explaining what they did and why; they have to write a publicly facing blog post for a general audience and that post has to have at least three graphs backing up their point; and they have to give a lightning talk (no more than five minutes) in class about what they have found. Those two assignments are typically worth 50 percent of the course grade.

I think rubrics are crap, but I give students these questions as a guide to what I'm expecting:

1. Did you read the data into a dataframe?
2. Did you use the skill discussed in the chapter correctly?
3. Did you answer all the questions posed by the assignment?
4. Did you use Markdown comments to explain your steps, what you did and why?

Chapter 1: Intro

- Install Slack on your computer and your phone.
- If on a Mac, install the Command Line Tools.
- Install R for your computer.

- Install R Studio Desktop for your computer ONLY AFTER YOU HAVE INSTALLED R

Chapter 2: Basics

Part 1:

In the console, type `install.packages("swirl")`

Then `library("swirl")`

Then `swirl()`

Follow instructions on the screen. Each time you are asked if which one you want, you want the first one. The basics, the beginning, the first parts. All the first ones. Then just follow the instructions on the screen.

Part 2:

Create an R notebook (which you should have done if you were following along). In it, delete all the generated text from it, so you have a blank document. Then, write a sentence in the document telling me what the last thing you did in Swirl was. Then add a code block (see about inserting code in the chapter) and add two numbers together. Any two numbers. Run that code block. Save the file and submit the .Rmd file created when you save it to Canvas. That's it. Simple.

Chapter 3: Data, structures and types

Using what you learned in the chapter, fetch the list of the Big Ten's leading tacklers. Submit the CSV file to Canvas. In the comments, label each field type. What are they? Dates? Characters? Numeric?

Chapter 4: Aggregates

Import this dataset of every college basketball game in the 2018-19 season. Using what you learned in the chapter, answer the following questions:

1. What team shot the most shots?
2. What team averaged the most shots?
3. What team had the highest median number of shots?
4. How much difference is there between the top average shots team and the top median shots team? Why do you think that is?

Chapter 5: Mutating Data

Import this dataset of every college basketball game in the 2018-19 season. Using what you learned in the chapter, mutate a new variable: differential. Differential is the difference between the team score and the opponent score. A positive number means the team in question won. A negative number means the team in question lost. After creating the differential, average them together and sort them in descending order. Which team had the highest average point differential in college basketball? In other words, which team consistently won by the largest margins?

Chapter 6: Filters and selections

Import the data of every college basketball player's season stats in 2018-19 season. Using this data, let's get closer to a real answer to where the cutoff for true shooting season should be from the chapter. First, find the median number of shots attempted in the season, then set the cutoff filter for who had the best true shooting percentage using that number.

Chapter 7: Transforming data

Import this dataset of college football attendance data from 2013-2018. This data is long data – one team, one year, one row. We need it to be wide data. Hint: it'll be much easier if you select only the columns you need to make it wide instead of using them all. Submit your notebook.

Chapter 8: Simulations

On Feb. 6, Nebraska's basketball team had a nightmare night shooting the ball. They attempted 57 shots … and made only 12. The team shot .429 on the season. Simulate 1000 games of them taking 57 shots using their season long .429 as the probability that they'll make a shot. How many times do they make just 12?

Chapter 9: Correlations and regressions

Do the same thing described in the chapter, but for defense. Report your R-squared number, your p-value, what those mean and from that, how close does it come to predicting the Iowa Nebraska game?

Chapter 10: Multiple regression

You have been hired by Fred Hoiberg to build a team. He's interested in the model started in the chapter, but wants more.

There are more predictors to be added to our model. You are to find two. Two that contribute to the predictive quality of the model without largely overlapping another predictor.

In your notebook, report the adjusted r-squared you achieved.

You are to generate a new set of coefficients, a new formula and a new set of numbers of what a conference champion would expect in terms of differential. I've done a lot of work for you. Continue it. Add two more predictors and complete the prediction. And compare that to Nebraska of this season.

Turn in your notebook with these answers and comments to the code you added, making sure to add WHY you are doing things. Why did you select those two variables.

Chapter 11: Residuals

Using the same data from the chapter, model defensive third down percentage and defensive points allowed. Which teams are overperforming that model given the residual analysis?

Chapter 12: Z scores

Refine the composite Z Score I started in the chapter. Add two more elements to it. What else do you think is important to the success of a basketball team? I've got shooting, rebounds and the opponents shooting. What else would you add?

In an R Notebook, make your case for the two elements you are adding – what is your logic? Why these two?. Then, follow my steps here until you get to the `teamquality` dataframe step, where you'll need to add the fields you are going to add to the composite. Then you'll need to add your fields to the `teamtotals` dataframe. Then you'll need to adjust `teamzscore`.

Finally, look at your ranking of Big Ten teams and compare it to mine. Did adding more elements to the composite change anything? Explain the differences in your notebook. Which one do you think is more accurate? I won't be offended if you say yours, but why do you feel that way?

Chapter 13: Intro to ggplot

Take this same attendance data. I want you to produce a bar chart of the top 10 schools by percent change in attendance between 2018 and 2013. I want you to change the title and the labels and I want you to apply a theme different from the ones I used above. You can find more themes in the ggplot documentation.

Chapter 14: Stacked bar charts

I want you to make this same chart, except I want you to make the weight the percentage of the total number of graduates that gender represents. You'll be mutating a new field to create that percentage. You'll then chart it with the fill. The end result should be a stacked bar chart allowing you to compare genders between universities. Answer the following question: Which schools have the largest gender imbalances?

Chapter 15: Waffle charts

Compare Nebraska and Michgan's night on the basketball court using a Waffle chart and another metric than what I've done above for the game.

Here's the library's documentation. Here's the stats from the game.

Turn in your notebook with your waffle chart. It must contain these two things:

- Your waffle chart
- A written narrative of what it says. What does your waffle chart say about how that game turned out?

Chapter 16: Line Charts

Import this dataset of every college basketball game in the 2018-19 season.

- How does Nebraska's shooting percentage compare to the Big Ten over the season? Put the Big Ten on the same chart as Nebraska, you'll need

two dataframes, two geoms and with your Big Ten dataframe, you need to use `group` in the aesthetic.

- After working on this chart, your boss comes in and says they don't care about field goal percentage anymore. They just care about three-point shooting because they read on some blog that three-point shooting was all the rage. Change what you need to change to make your line chart now about how the season has gone behind the three-point line. How does Nebraska compare to the rest of the Big Ten?

Chapter 17: Step charts

Re-make the chart in the chapter, but with rebounding. I want you to visualize the differential between our rebounds and their rebounds, and then plot the step chart showing over the course of the season. Highlight Nebraska. Highlight the top team. Add annotation layers to label both of them.

Chapter 18: Ridge charts

You've been hired by Fred Hoiberg to tell him how to win the Big Ten. He's not impressed with that I came up with. So what you need to do is look for a *composite* measure that produces a meaningful ridgeplot. What that means is you're going to mutate `wintotalgroupinglogs` one more time. Is the differential between rebounding meaningful instead of just the total? Or assists? Or something else? Your call. Your goal is to produce a ridgeplot that tells The Mayor he needs to focus on doing X better than the opponent to win a Big Ten title.

Chapter 19: Lollipop charts

You've been hired by Fred Hoiberg to tell him how to win the Big Ten. He's not impressed with that I came up with. So what else could you look at with lollipop charts? Your call. Your goal is to produce a lollipop chart that tells The Mayor he needs to focus on the gap between X and Y if he wants to win a Big Ten title.

Chapter 20: Scatterplot

Using the data from the walkthrough, model and graph two other elements of Nebraska's season versus wins. How much does your choices of metrics predict the season? What do the scatterplots of what you chose look like? What do the linear models say (r-squared, p-values)? How predictive are they, i.e. using y=mx+b, how close to Nebraska's win total do your models get to?

Chapter 21: Facet Wraps

Which Big Ten teams were good a shooting three point shots? Which teams weren't? Using a facet grid, chart each teams three point shooting season against the league average.

Chapter 22

Import this dataset of every college basketball game in the 2018-19 season.

Create a dataframe that shows the 10 best or 10 worst at something. Or rank the Big Ten. Your choice. Then use formattable to show in both a table and visually how the best were different from the worst.

Export it to a PNG using the example above. Then, in Illustrator, add a headline, chatter, source and credit lines. Turn in the PNG file you export from Illustrator.

New Chapter

Import this dataset of every college basketball game in the 2018-19 season. Create a bubble chart looking at two stats to make your scatterplot and a third making the size of your bubble. Make the color the conference name.

I want to see your bubble chart, but more importantly, I want you to discuss if what you came up with makes an effective bubble chart. Does it tell a story? Does the size of the bubble enhance understanding? No is an acceptable answer. But explain why it did or didn't work.

New Chapter

Your turn: Let's evaluate the second part of the quote from the chapter: November basketball tells you where you are.

We've looked at wins. What else could you look at over the course of the season that tells you where you are? Pick a metric. Explain your choice. Make a circular bar chart. Evaluate the result. What does it say?

Chapter 23: Rvest

I am a huge Premiere League fan, so I want data on the league. For now, I just want teams. Scrape the team data at the top, but before you do, look at the header. Is it one row? Does that make it standard? Nope. So what now?

Chapter 24: Advanced Rvest

I don't usually assign an advanced rvest assignment because I don't want to turn 30 students loose on some poor provider's servers.

Note: There are no assignments for annotations and finishing touches. In my classes, I have students present two major visual stories where they have to incorporate the elements of those assignments as part of their grade.

Chapter 30: Plotly

First, create a simple ggplot like we did above exploring WRAA – weighted runs above average – as your x value and and plate appearances (PA) as your y variable.

Next, create a plotly visualization using the same two variables. Alter the hover elements to show relevant data. If you leave it the same from the chapter, you lose points.

Export your plotly visualization to plotly's website. Include a link of your viz in your notebook. In your notebook, discuss the relative advantages and disadvantages of this interactive plot versus the static plots we've been doing.

Chapter 31: Clustering

We looked at who Cam Mack's peers are, but what about the team? Use k-means clustering on a dataset of every college basketball team's season stats and determine who Nebraska's peers are.

To complete this assignment, you'll need to pick the metrics you want to measure teams by. One note – teams haven't played the same number of games, so it would be wise to either focus on the per game or percentage metrics, either using them or creating them yourself. You'll then need to scale them. You'll need to decide the optimal number of clusters (k) and then run them. Combine the data back, determine which cluster Nebraska is in in your clustering and then show Nebraska's peers.

In your notebook, write a few sentences and answering this question: Is the peer group Nebraska is in fair?

Chapter 32: Rtweet

Using the Rtweet library, gather tweets about the Maryland game Saturday night. WARNING: The API only lets you go back 18000 tweets – we settled on 8000 in the walkthrough. If you wait too long to run the scrape after the game, you won't get the tweets you need to complete it. If you don't intend to do the assignment Saturday night, at least scrape the data and save it as a CSV as detailed in the chapter. You can then analyze it any time you want.

Analyze sentiment and chart it as in the chapter. Compare the Maryland game to the Purdue game. Are they different? Is sentiment better or worse for one or the other? Describe the differences in your notebook.

Chapter 35

Appendix

These are some additional materials I use in my classes.

35.1 How to get help in this class

> This is the contents of a document I send out every semester. I use Slack to help students with code problems outside of class. It's much easier than other options, such as email. A suggestion: set ground rules on when you will and won't answer Slack messages.

1. **Use Slack**. Email is a miserable way to handle technological questions. I'm not answering code questions via email. On Slack we can have a back and forth where we solve this quickly instead of waiting on each other to respond to an email.
2. **Don't use screenshots**. Tell me what you're trying to do and then copy and paste your code into the Slack message.
3. **Always copy and paste the error message you are getting**. There is a near infinite number of things you could have done and a nearly limitless number of errors you could be getting. Both help.

Slack tips

- Slack uses Markdown in messages. Did you know your code blocks in R Notebook are Markdown themselves? If you copy the whole block – with the "‘ and everything, Slack will format it like this:

```
simulations <- rbinom(n = 1000, size = 39, prob = .309)

hist(simulations)

table(simulations)
```

- Using the Slack app will also mean getting alerted to messages right away. If you are logging in through a web browser, you won't know when I've responded. If we're having a Slack conversation and I see you log in, send me a message, then disappear right away, I know you're using a browser and when that conversation drags because you aren't getting messages, **I'm going to get frustrated**. You likely aren't the only one asking for help at that moment.