Sports Data Analysis and Visualization

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

By Matt Waite

July 29, 2019

Contents

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:

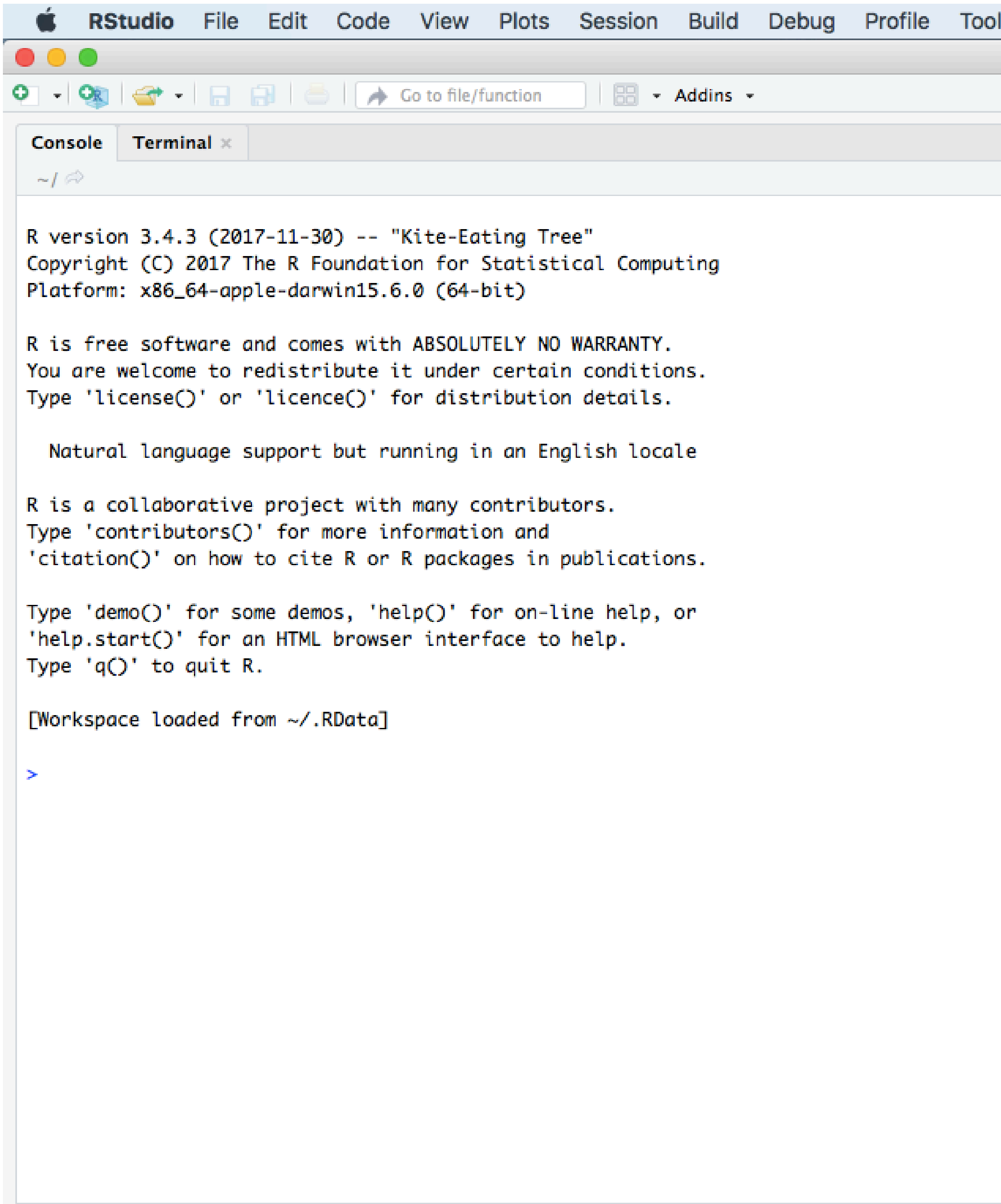
- It is free for students.
- The topics will remain the same but the text is going to be constantly tinkered with.
- What is the work of the author is copyright Matt Waite 2019.
- The text is Attribution-NonCommercial-ShareAlike 4.0 International Creative Commons licensed. That means you can share it 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')`

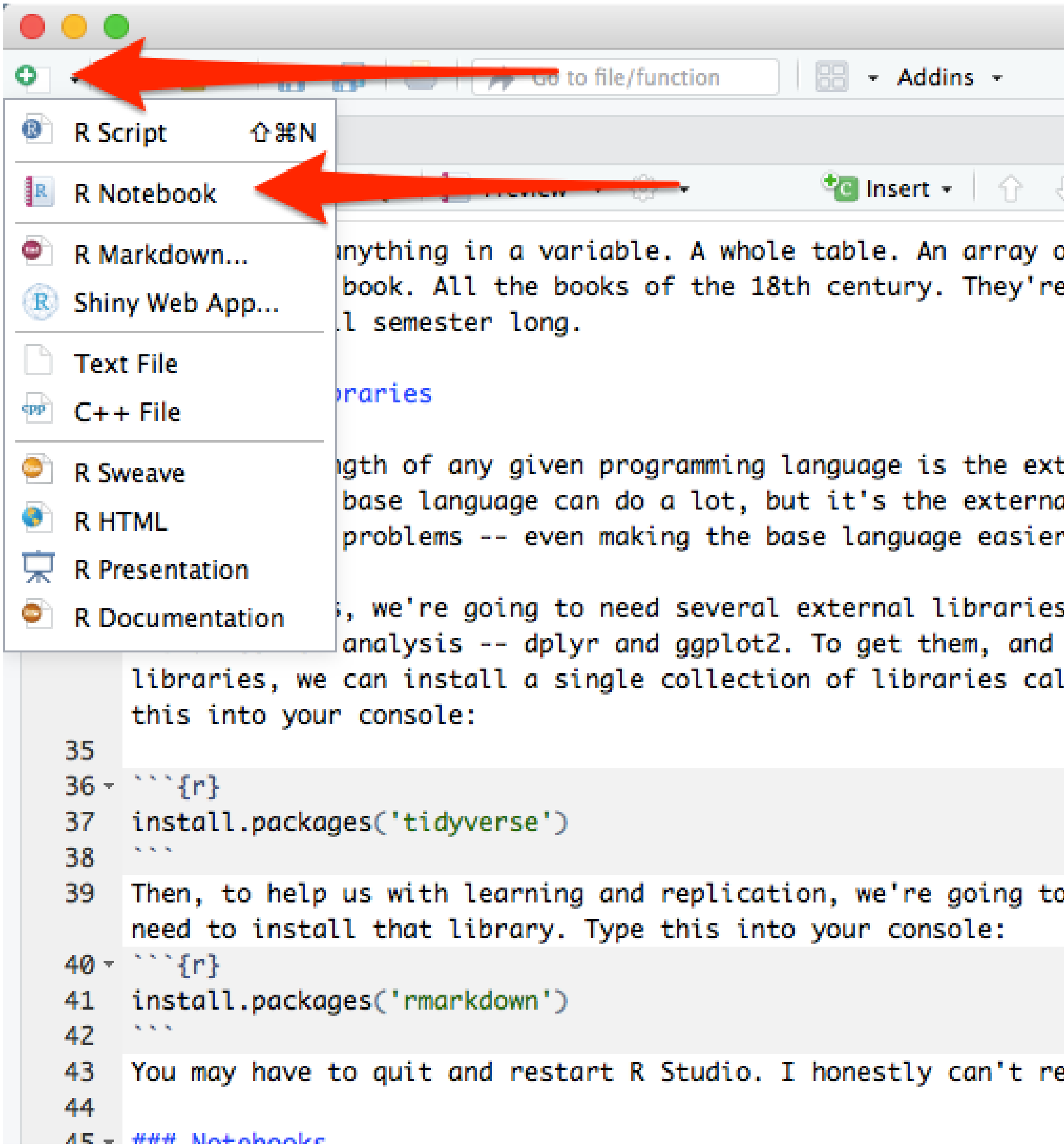
NOTE: This is a pattern. You should always install libraries in the console.

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
this into your console:
```{r}
install.packages('tidyverse')
```
need to install that library. Type this into your console:
```{r}
install.packages('rmarkdown')
```

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

Rows

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.

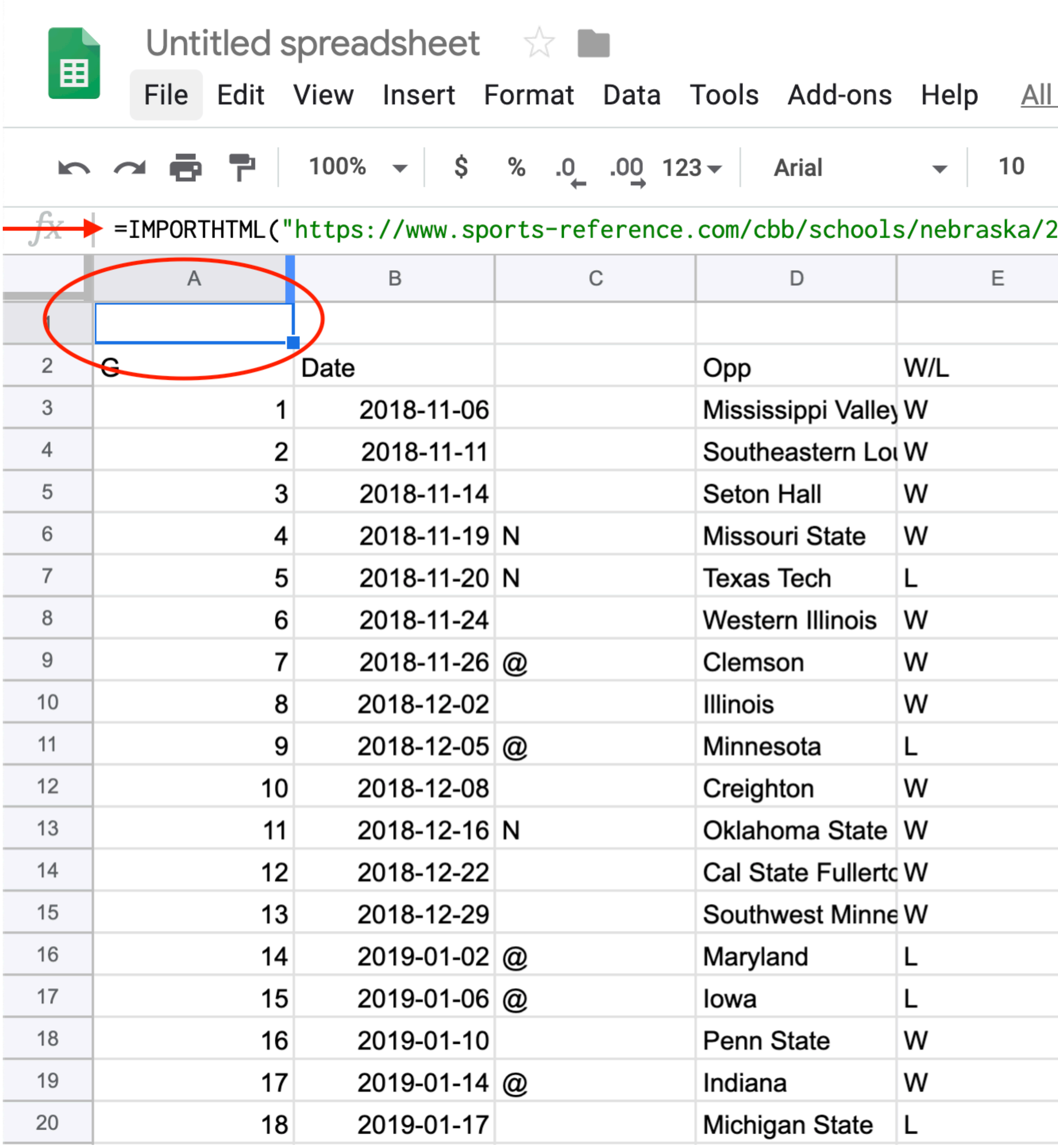
Now, in a new tab, log into Google Docs/Drive and open a new spreadsheet. In the first cell of the first row, copy and paste this formula in:

```
=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:

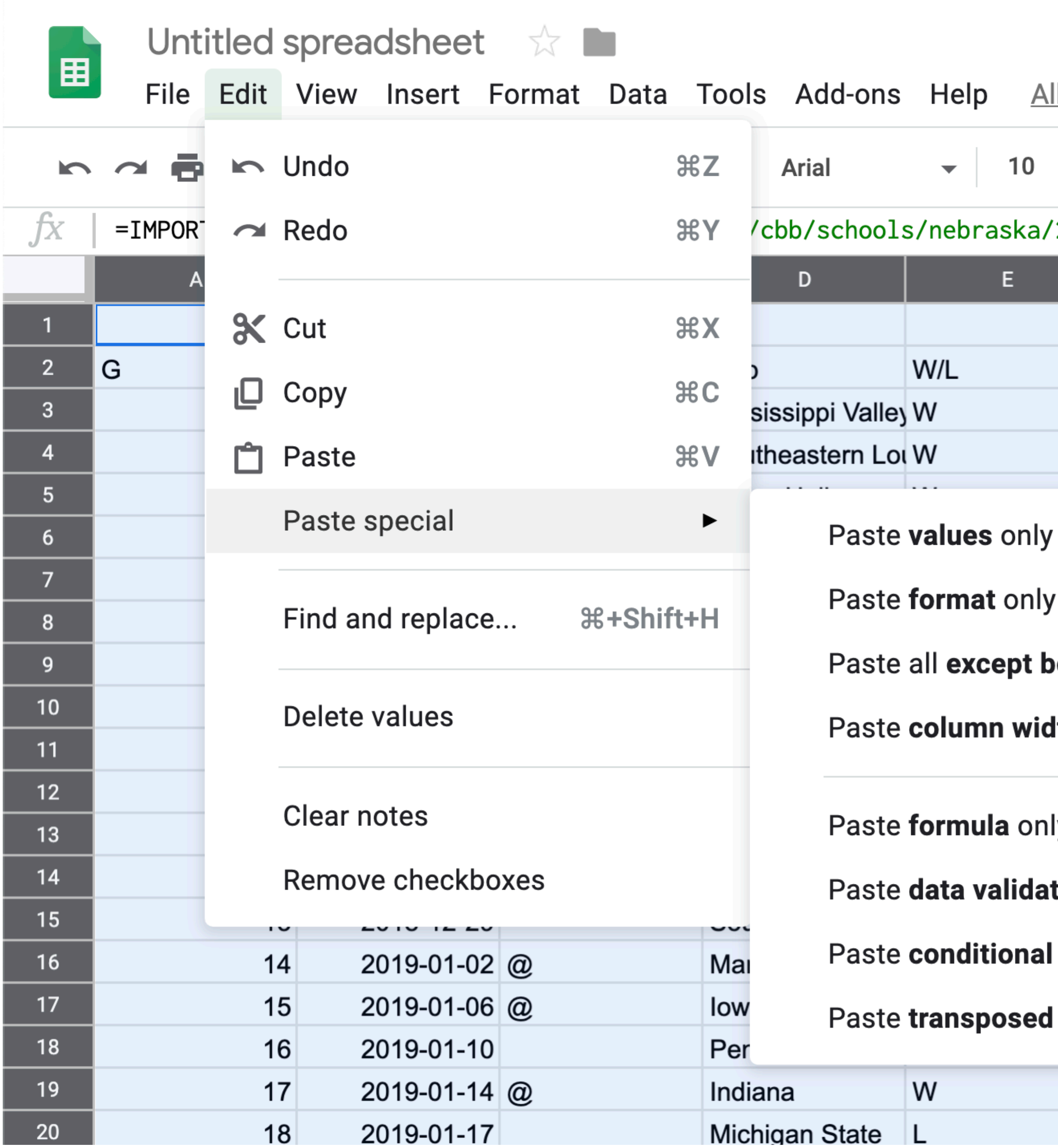


	A	B	C	D	E
1					
2	G	Date		Opp	W/L
3	1	2018-11-06		Mississippi Valley	W
4	2	2018-11-11		Southeastern Lou	W
5	3	2018-11-14		Seton Hall	W
6	4	2018-11-19	N	Missouri State	W
7	5	2018-11-20	N	Texas Tech	L
8	6	2018-11-24		Western Illinois	W
9	7	2018-11-26	@	Clemson	W
10	8	2018-12-02		Illinois	W
11	9	2018-12-05	@	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.

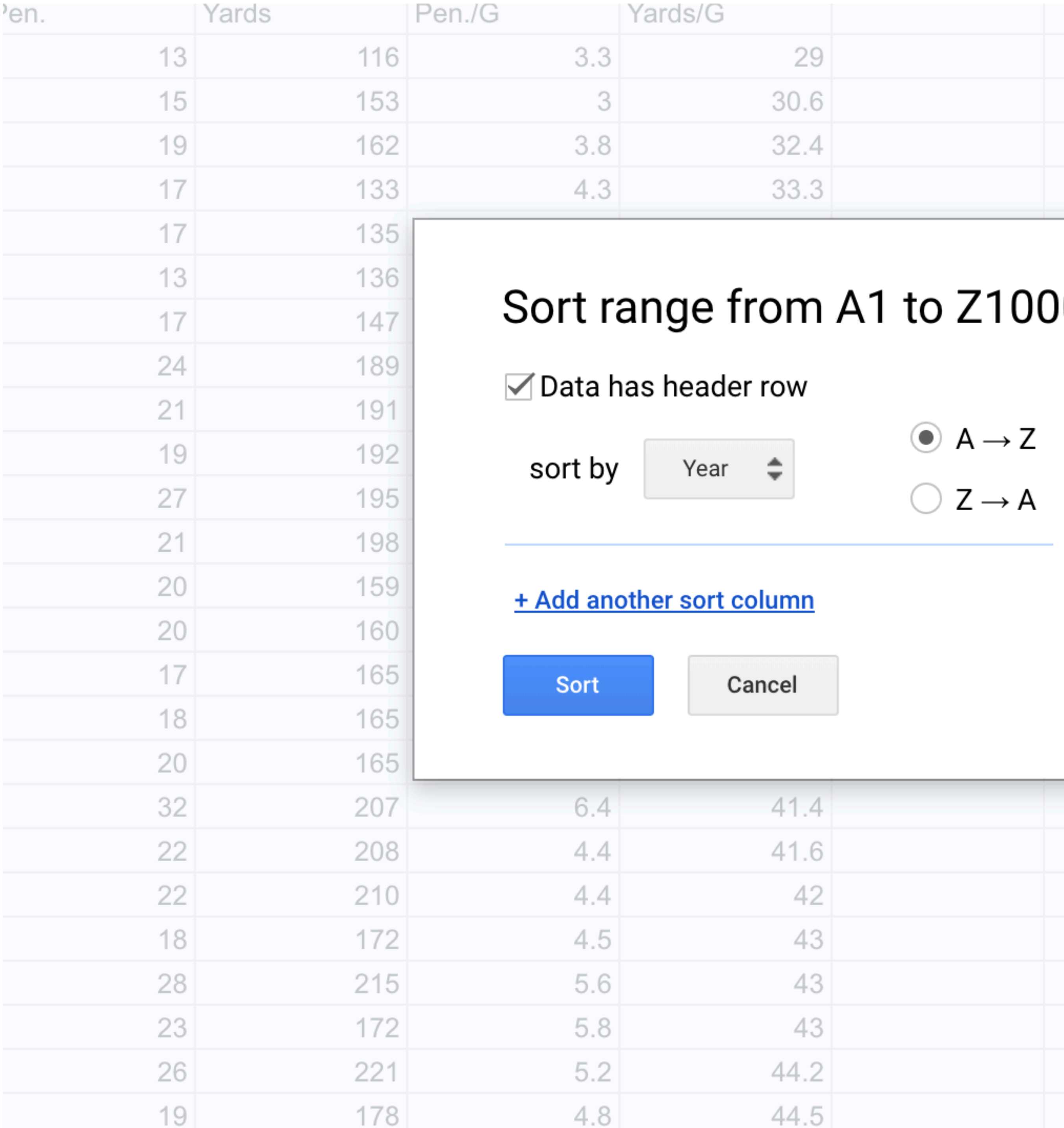

Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help
Undo ⌘Z
Redo ⌘Y
Cut ⌘X
Copy ⌘C
Paste ⌘V
Paste special
Find and replace... ⌘+Shift+H
Delete values
Clear notes
Remove checkboxes
Paste values only
Paste format only
Paste formula only
Paste transposed
Arial
10
=IMPOR
/cbb/schools/nebraska/
W/L
Indiana
Michigan State

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 assignment are `readr` and `dplyr`. The library `readr` reads different types of data in as a dataframe. 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)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
```

```
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

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

But the tidyverse has a neat little trick. We can load most of the libraries we'll need for the whole semester with one line:

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v stringr 1.4.0
## v tidyr   1.1.2     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

From now on, if that's not the first line of your notebook, you're probably doing it wrong.

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 Basketball Reference of advanced metrics for NBA players this season. You can download the CSV here. The Sports Reference sites are a godsend of data, a trove of stuff, and we're going to use it a lot in this class.

So step 2, after setting up our libraries, is most often going to be importing data. In order to analyze data, we need data, so it stands to reason that this would be something we'd do very early.

The code looks *something* like this, but hold off copying it just yet:

```
nbaplayers <- read_csv("~/Box/SportsData/nbaadvancedplayers1920.csv")
```

Let's unpack that.

The first part – nbaplayers – is the name of your variable. A variable is just a name of a thing that stores stuff. In this case, our variable is a data frame, which is R's way of storing data (technically it's a tibble, which is the tidyverse way of storing data, but the differences aren't important and people use them interchangeably). **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 NBA players. 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 `nbaplayers` 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 (nbaadvancedplayers1920.csv). In my case, I've got a folder called Box in my home directory (that's the ~ part), and in there is a folder called SportsData that has the file called nbaadvancedplayers1920.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.

```
nbaplayers <- read_csv("data/nbaadvancedplayers1920.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Player = col_character(),
##   Pos = col_character(),
##   Tm = col_character()
## )
## i Use `spec()` for the full column specifications.
```

Now we can inspect the data we imported. What does it look like? To do that, we use `head(nbaplayers)` 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(nbaplayers)`

```
head(nbaplayers)
```

```
## # A tibble: 6 x 27
##      Rk Player Pos     Age Tm        G    MP   PER `TS%` `3PAr`   FTr `ORB%`
```

```
##   <dbl> <chr>  <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>
## 1     1 Steve~ C        26 OKC      63  1680  20.5 0.604  0.006 0.421   14
## 2     2 Bam A~ PF       22 MIA      72  2417  20.3 0.598  0.018 0.484    8.5
## 3     3 LaMar~ C        34 SAS      53  1754  19.7 0.571  0.198 0.241    6.3
## 4     4 Kyle ~ PF       23 MIA       2    13   4.7 0.5    0      0       17.9
## 5     5 Nicke~ SG       21 NOP      47   591   8.9 0.473  0.5    0.139    1.6
## 6     6 Grays~ SG       24 MEM      38   718  12   0.609  0.562 0.179    1.2
## # ... with 15 more variables: `DRB%` <dbl>, `TRB%` <dbl>, `AST%` <dbl>,
## #   `STL%` <dbl>, `BLK%` <dbl>, `TOV%` <dbl>, `USG%` <dbl>, OWS <dbl>,
## #   DWS <dbl>, WS <dbl>, `WS/48` <dbl>, OBPM <dbl>, DBPM <dbl>, BPM <dbl>,
## #   VORP <dbl>
```

```
nrow(nbaplayers)
```

```
## [1] 651
```

Another way to look at nrow – we have 651 players from this season in our dataset.

What if we wanted to know how many players there were by position? To do that by hand, we'd have to take each of the 651 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."

After we group them together, we need to count them. We do that first by saying we want to summarize our data (a count is a part of a summary). To get a summary, we have to tell it what we want. So in this case, we want a count. To get that, let's create a thing called total and set it equal to n(), which is `dplyr`s way of counting something.

Here's the code:

```
nbaplayers %>%
  group_by(Pos) %>%
  summarise(
    total = n()
  )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 9 x 2
##   Pos   total
##   <chr> <int>
## 1 C       111
```

```
## 2 C-PF      2
## 3 PF      135
## 4 PF-C      2
## 5 PG      111
## 6 SF      113
## 7 SF-PF     4
## 8 SF-SG     3
## 9 SG      170
```

So let's walk through that. We start with our dataset – `nbaplayers` – and then we tell it to group the data by a given field in the data which we get by looking at either the output of `head` or you can look in the environment where you'll see `nbaplayers`.

In this case, we wanted to group together positions, signified by the field name Pos. 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 positions: `total = 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 players 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 positions 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:

```
nbaplayers %>%
  group_by(Pos) %>%
  summarise(
    total = n()
  ) %>% arrange(desc(total))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 9 x 2
##   Pos   total
##   <chr> <int>
## 1 SG      170
## 2 PF      135
## 3 SF      113
## 4 C       111
## 5 PG      111
## 6 SF-PF     4
```

```
## 7 SF-SG     3
## 8 C-PF      2
## 9 PF-C      2
```

So the most common position in the NBA? Shooting guard, followed by power forward.

We can, if we want, group by more than one thing. Which team has the most of a single position? To do that, we can group by the team – called Tm in the data – and position, or Pos in the data:

```
nbaplayers %>%
  group_by(Tm, Pos) %>%
  summarise(
    total = n()
  ) %>% arrange(desc(total))
```

```
## `summarise()` regrouping output by 'Tm' (override with `.groups` argument)
```

```
## # A tibble: 159 x 3
## # Groups:   Tm [31]
##    Tm    Pos   total
##    <chr> <chr> <int>
##  1 TOT   PF       13
##  2 TOT   SG       13
##  3 SAC   PF        9
##  4 TOT   SF        9
##  5 BRK   SG        8
##  6 LAL   SG        8
##  7 TOT   PG        8
##  8 ATL   SG        7
##  9 BRK   SF        7
## 10 DAL   SG        7
## # ... with 149 more rows
```

So wait, what team is TOT?

Valuable lesson: whoever collects the data has opinions on how to solve problems. In this case, Basketball Reference, when a player get's traded, records stats for the player's first team, their second team, and a combined season total for a team called TOT, meaning Total. Is there a team abbreviated TOT? No. So ignore them here.

Sacramento has 9 power forward. Brooklyn has 8 shooting guards, as do the Lakers. You can learn a bit about how a team is assembled by looking at these simple counts.

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.

Sticking with our NBA player data, we can calculate any number of measures inside summarize. Here, we'll use R's built in mean and median functions to calculate … well, you get the idea.

Let's look just a the number of minutes each position gets.

```
nbaplayers %>%
  group_by(Pos) %>%
  summarise(
    count = n(),
    mean_minutes = mean(MP),
    median_minutes = median(MP)
  )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 9 x 4
##   Pos   count mean_minutes median_minutes
##   <chr> <int>        <dbl>          <dbl>
## 1 C       111         891.           887
## 2 C-PF      2         316.           316.
## 3 PF      135         790.           567
## 4 PF-C      2        1548.          1548.
## 5 PG      111         944.           850
## 6 SF      113         877.           754
## 7 SF-PF     4         638.           286.
## 8 SF-SG     3        1211           1688
## 9 SG      170         843.           654.
```

So there's 651 players in the data. Let's look at shooting guards. The average shooting guard plays 842 minutes and the median is 653.5 minutes.

Why?

Let's let sort help us.

```
nbaplayers %>% arrange(desc(MP))
```

```
## # A tibble: 651 x 27
##       Rk Player Pos     Age Tm        G    MP   PER `TS%` `3PAr`   FTr `ORB%`
##    <dbl> <chr>  <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>
##  1   323 CJ Mc~ SG       28 POR      70  2556  17   0.541  0.378 0.136    1.9
##  2    55 Devin~ SG       23 PHO      70  2512  20.6 0.618  0.31  0.397    1.3
##  3   198 James~ SG       30 HOU      68  2483  29.1 0.626  0.557 0.528    2.9
```

```
## 4    27 Harri~ PF       27 SAC      72  2482  13.3 0.574  0.338 0.337    3.4
## 5   297 Damia~ PG       29 POR      66  2474  26.9 0.627  0.5   0.384    1.4
## 6   204 Tobia~ PF       27 PHI      72  2469  17.2 0.556  0.304 0.184    3.1
## 7   479 P.J. ~ PF       34 HOU      72  2467   8.3 0.559  0.702 0.113    4.7
## 8   175 Shai ~ SG       21 OKC      70  2428  17.7 0.568  0.247 0.352    2.2
## 9     2 Bam A~ PF       22 MIA      72  2417  20.3 0.598  0.018 0.484    8.5
## 10  343 Donov~ SG       23 UTA      69  2364  18.8 0.558  0.352 0.24     2.6
## # ... with 641 more rows, and 15 more variables: `DRB%` <dbl>, `TRB%` <dbl>,
## #   `AST%` <dbl>, `STL%` <dbl>, `BLK%` <dbl>, `TOV%` <dbl>, `USG%` <dbl>,
## #   OWS <dbl>, DWS <dbl>, WS <dbl>, `WS/48` <dbl>, OBPM <dbl>, DBPM <dbl>,
## #   BPM <dbl>, VORP <dbl>
```

The player with the most minutes on the floor is a shooting guard. Shooting guard is the most common position, so that means there's CJ McCollum rolling up 2,556 minutes in a season, and then there's Cleveland Cavalier's sensation J.P. Macura. Never heard of J.P. Macura? Might be because he logged one minute in one game this season.

That's a huge difference.

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 pine riders and then one ironman superstar, the average will be wildly skewed.

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 position in the NBA plays the most minutes? And what is the highest and lowest minute total for that position? And how wide is the spread between minutes? We can find that with `sum` to add up the minutes to get the total minutes, `min` to find the minimum minutes, `max` to find the maximum minutes and `sd` to find the standard deviation in the numbers.

```
nbaplayers %>%
  group_by(Pos) %>%
  summarise(
    total = sum(MP),
    avgminutes = mean(MP),
    minminutes = min(MP),
    maxminutes = max(MP),
    stdev = sd(MP)) %>% arrange(desc(total))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 9 x 6
##   Pos    total avgminutes minminutes maxminutes stdev
##   <chr>  <dbl>      <dbl>      <dbl>      <dbl> <dbl>
## 1 SG    143229       843.          1       2556 735.
## 2 PF    106654       790.          5       2482 719.
## 3 PG    104745       944.          8       2474 727.
## 4 SF     99109       877.         11       2316 709.
## 5 C      98914       891.          3       2336 619.
## 6 SF-SG   3633      1211          87       1858 977.
## 7 PF-C    3097      1548.        960       2137 832.
## 8 SF-PF   2553       638.         46       1936 873.
## 9 C-PF     633       316.        256        377  85.6
```

So again, no surprise, shooting guards spend the most minutes on the floor in the NBA. They average 842 minutes, but we noted why that's trouble. The minimum is the J.P. Macura Award, max is the Trailblazer's failing at load management, and the standard deviation is a measure of how spread out the data is. In this case, not the highest spread among positions, but pretty high. So you know you've got some huge minutes players and a bunch of bench players.

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')
```

```
##
## -- 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.

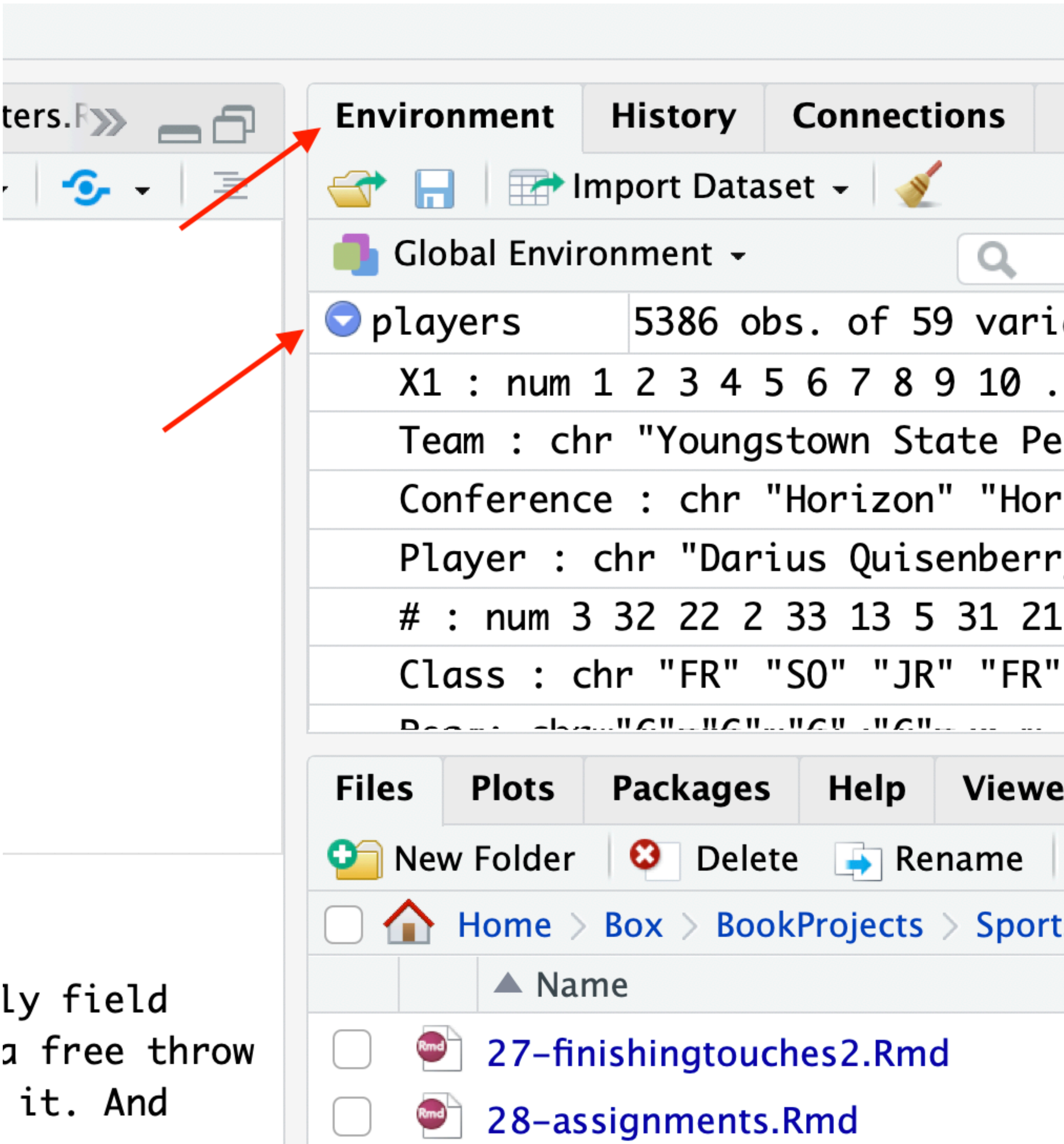
```
players <- read_csv("data/players19.csv")
```

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

```
##
## -- 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()
## )
## i Use `spec()` for the 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.



ters.R
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')
```

```
##
## -- 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]
```

```
##
## -- 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()
## )
## i Use `spec()` for the 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, `
```

```
## 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')
```

```
##
## -- 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.

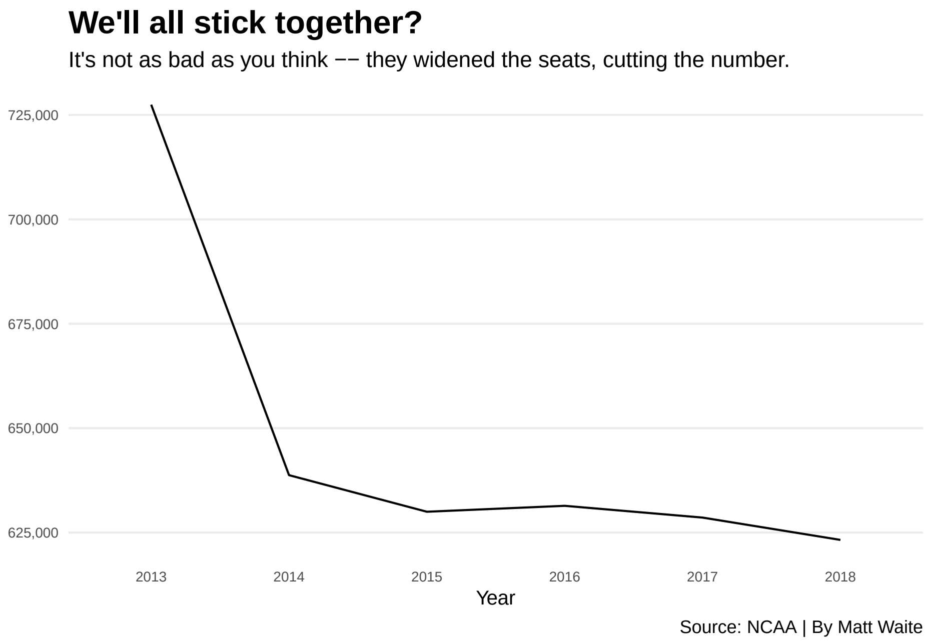
So let me walk you through an example using this data.

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

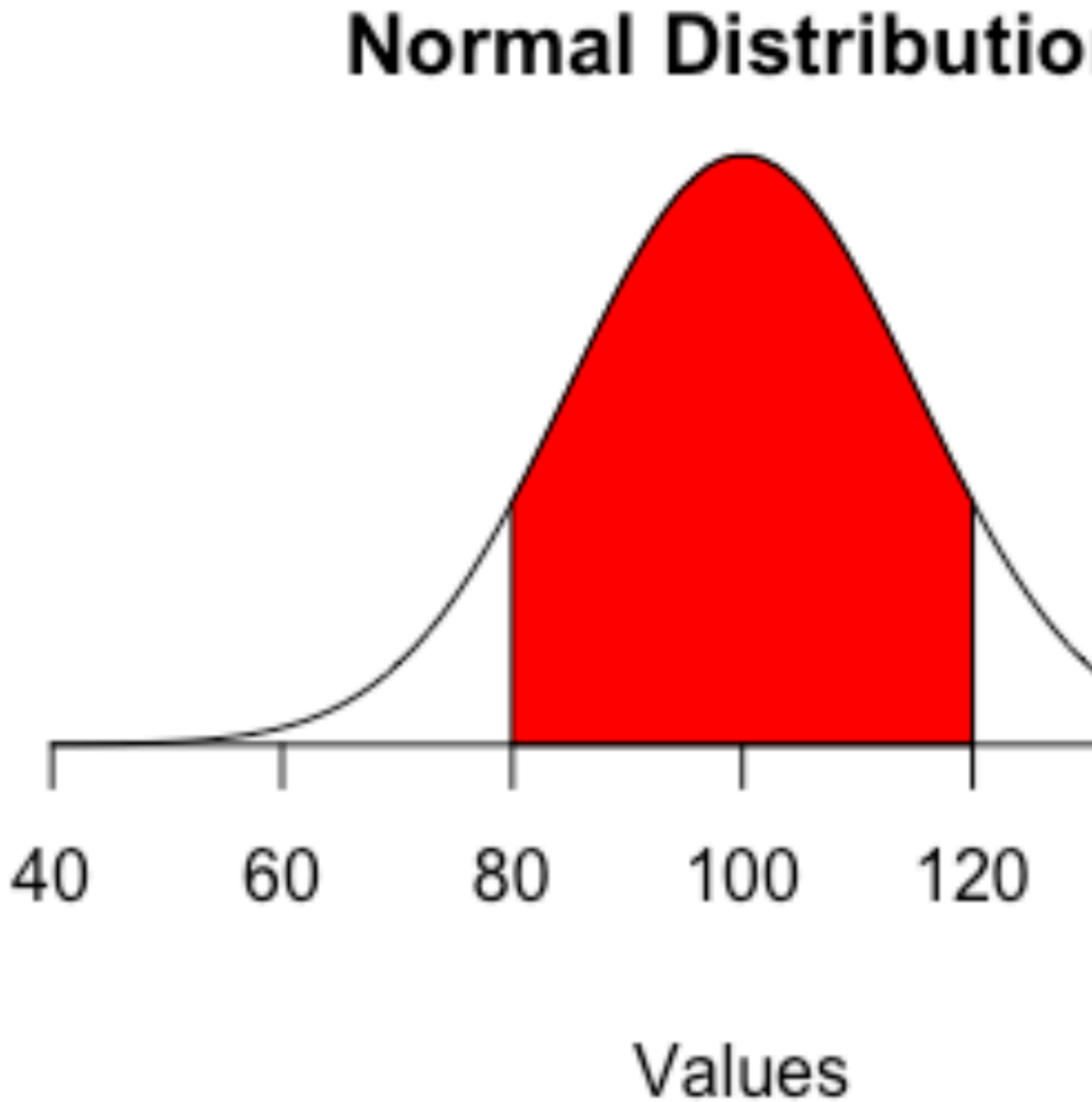
Significance tests

Now that we've worked with data a little, it's time to start asking more probing questions of our data. One of the most probing questions we can ask – one that so few sports journalists ask – is if the difference between this thing and the normal thing is real.

We have a perfect natural experiment going on in sports right now to show how significance tests work. The NBA, to salvage a season and get to the playoffs, put their players in a bubble – more accurately a hotel complex at Disney World in Orlando – and had them play games without fans.

So are the games different from other regular season games that had fans?

To answer this, we need to understand that a significance test is a way to determine if two numbers are *significantly* different from each other. Generally speaking, we're asking if a subset of data – a sample – is different from the total data pool – the population. Typically, this relies on data being in a normal distribution.



If it is, then we know certain things about it. Like the mean – the average – will be a line right at the peak of cases. And that 66 percent of cases will be in that red area – the first standard deviation.

A significance test will determine if a sample taken from that group is different from the total.

Significance testing involves stating a hypothesis. In our case, our hypothesis is that there is a difference between bubble games without people and regular games with people.

In statistics, the **null hypothesis** is the opposite of your hypothesis. In this case, that there is no difference between fans and no fans.

What we're driving toward is a metric called a p-value, which is the probability that you'd get your sample mean *if the null hypothesis is true.* So in our case, it's the probability we'd see the numbers we get if there was no difference between fans and no fans. If that probability is below .05, then we consider the difference significant and we reject the null hypothesis.

So let's see. Here's a log of every game in this NBA season. There's a field called COVID, which labels the game as a regular game or a bubble game.

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

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Season = col_character(),
##   Conference = col_character(),
##   Team = col_character(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   COVID = col_character()
## )
## i Use `spec()` for the full column specifications.
```

First, let's just look at scoring. Here's a theory: fans make players nervous. The screaming makes players tense up, and tension makes for bad shooting. An alternative to this: screaming fans make you defend harder. So my hypothesis is that not only is the scoring different, it's lower.

First things first, let's create a new field, called `totalpoints` and add the two scores together. We'll need this, so we're going to make this a new dataframe called `points`.

```
points <- logs %>% mutate(totalpoints = TeamScore + OpponentScore )
```

Typically speaking, with significance tests, the process involves creating two different means and then running a bunch of formulas on them. R makes this easy by giving you a `t.test` function, which does all the work for you. What we have to tell it is what is the value we are testing, over which groups, and from what data. It looks like this:

```
t.test(totalpoints ~ COVID, data=points)
```

```
##
##  Welch Two Sample t-test
##
## data:  totalpoints by COVID
## t = -5.232, df = 206.88, p-value = 4.099e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -11.64698  -5.27178
## sample estimates:
##    mean in group With Fans mean in group Without Fans
##                   222.8929                  231.3523
```

Now let's talk about the output. I prefer to read these bottom up. So at the bottom, it says that the mean number of points score in an NBA game With Fans is 222.89. The mean scored in games Without Fans is 231.35. That means teams are scoring almost 8.5 points MORE without fans on average.

But, some games are defenseless track meets, some games are defensive slugfests. We learned that averages can be skewed by extremes. So the next thing we need to look at is the p-value. Remember, this is the probability that we'd get this sample mean – the without fans mean – if there was no difference between fans and no fans.

The probability? 4.099e-07 or 4.099 x 10 to the -7 power. Don't remember your scientific notation? That's .00000004099. The decimal, seven zeros and the number.

Remember, if the probability is below .05, then we determine that this number is statistically significant. We'll talk more about statistical significance soon, but in this case, statistical significance means that our hypothesis is correct: points are different without fans than with. And since our hypothesis is correct, we *reject the null hypothesis* and we can confidently say that bubble teams are scoring more than they were when fans packed arenas.

8.1 Accepting the null hypothesis

So what does it look like when your hypothesis is wrong?

Let's test another thing that may have been impacted by bubble games: home court advantage. If you're the home team, but you're not at home, does it affect you? It has to, right? Your fans aren't there. Home and away are just positions on the scoreboard. It can't matter, can it?

My hypothesis is that home court is no longer an advantage, and the home team will score less relative to the away team.

First things first: We need to make a dataframe where Team is the home team. And then we'll create a differential between the home team and away team. If home court is an advantage, the differential should average out to be positive – the home team scores more than the away team.

```
homecourt <- logs %>% filter(is.na(HomeAway) == TRUE) %>% mutate(differential = TeamScore - Oppor
```

Now let's test it.

```
t.test(differential ~ COVID, data=homecourt)
```

```
##
##  Welch Two Sample t-test
##
## data:  differential by COVID
## t = 0.36892, df = 107.84, p-value = 0.7129
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.301628  3.354268
## sample estimates:
##    mean in group With Fans mean in group Without Fans
##                   2.174047                   1.647727
```

So again, start at the bottom. With Fans, the home team averages 2.17 more points than the away team. Without fans, they average 1.64 more.

If you are a bad sportswriter or a hack sports talk radio host, you look at this and scream "the bubble killed home court!"

But two things: first, the home team is STILL, on average, scoring more than the away team on the whole.

And two: Look at the p-value. It's .7129. Is that less than .05? No, no it is not. So that means we have to **accept the null hypothesis** that there is no difference between fans and no fans when it comes to the difference between the home team and the away team's score.

Now, does this mean that the bubble hasn't impacted the magic of home court? Not necessarily. What it's saying is that the variance between one and the other is too large to be able to say that they're different. It could just be random noise that's causing the difference, and so it's not real. More to the point, it's saying that this metric isn't capable of telling you that there's no home court in the bubble.

We're going to be analyzing these bubble games for *years* trying to find the true impact of fans.

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/footballlogs19.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   Result = col_character(),
##   TeamFull = col_character(),
##   TeamURL = col_character(),
##   Outcome = col_character(),
##   Team = col_character(),
##   Conference = col_character()
## )
## i Use `spec()` for the full column specifications.
```

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 = TeamScore - OpponentScore, TotalPenalties = Penalties+DefPenal
```

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 `TotalPenaltyYards`, which is the yards of penalties a team gets in a game. We do that by referencing `differential` and `TotalPenaltyYards` and specifying we want a `pearson` correlation. The number we get back is the correlation coefficient.

```
newcorrelations %>% summarise(correlation = cor(differential, TotalPenaltyYards, metho
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1     -0.0137
```

So on a scale of -1 to 1, where 0 means there's no relationship at all and 1 or -1 means a perfect relationship, penalty yards and whether or not the team scores more points than it give up are at -0.014. You could say they're 1.4 percent related toward the negative – more penalties, the lower your differential. Another way to say it? They're 98.6 percent not related.

What about the number of penalties instead of the yards?

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

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1    -0.00875
```

So wait, what does this all mean?

It means that when you look at every game in college football, the number of penalties and penalty yards does have a negative impact on the score difference between your team and the other team. But the relationship between penalties, penalty yards and the difference between scores is barely anything at all. Like 99 percent not related.

Normally, at this point, you'd quit while you were ahead. A correlation coefficient that shows there's no relationship between two things means stop. It's pointless to go on. But let's beat a dead horse a bit for the sake of talk radio callers who want to complain about undisciplined football teams.

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? So regressions are about prediction, correlations are about description. Correlations describe a relationship. Regressions help us predict what that relationship means and what it might look like in the real world. 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 penalty 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.

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

```
##
## Call:
## lm(formula = differential ~ TotalPenaltyYards, data = newcorrelations)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -73.13 -15.16   0.71  15.61  76.86
##
## Coefficients:
```

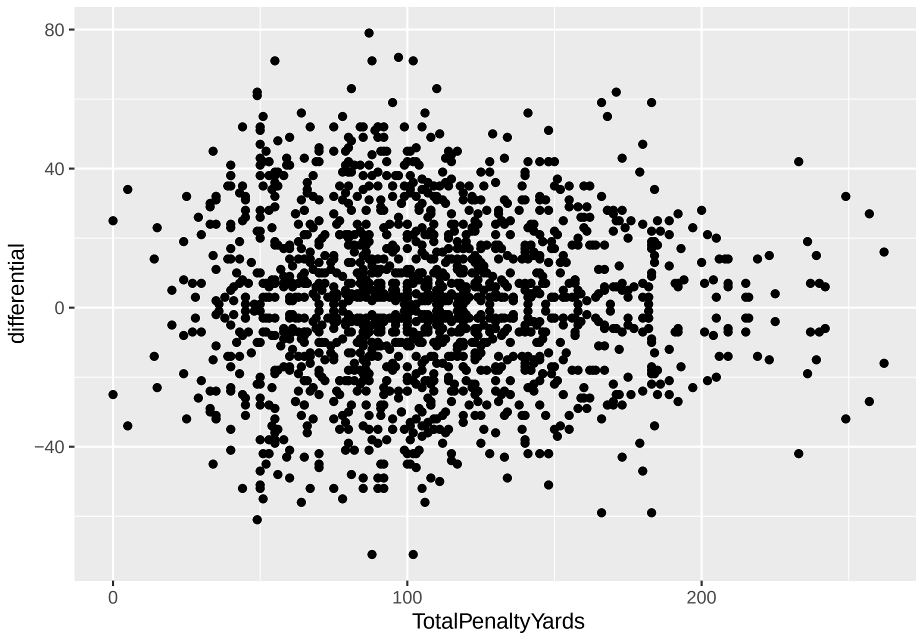
```
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        2.785946   1.525993   1.826   0.0681 .
## TotalPenaltyYards -0.007407   0.013244  -0.559   0.5760
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.3 on 1660 degrees of freedom
## Multiple R-squared:  0.0001884,  Adjusted R-squared:  -0.0004139
## F-statistic: 0.3128 on 1 and 1660 DF,  p-value: 0.576
```

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 Total Penalty Yards, the p-value is .576. 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 total penalty yards and a team's aggregate point differential are **not statistically significant**. The differences in score difference and penalty yards could be completely random. This is another sign we should just stop with this.
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 1.4 percent related to each other, but penalty yard's ability to explain the difference between teams? About .04 percent. That's … not much. It's really nothing. Again, we should quit.
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 2.79 and the TotalPenaltyYards coefficient is -0.007407. 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 score differential in a game – will they score more than they give up – by using `y=mx+b`. In this case, y is the aggregate score, m is -0.007407 and b is 2.79. So we would multiply a teams total penalty yards by -0.007407 and then add 2.79 to it. The result would tell you what the total aggregate score in the game would be, according to our model. Chance that your even close with this? About .04 percent. In other words, you've got a 99.96 percent chance of being completely wrong. Did I say we should quit? Yeah.

So penalty yards are totally meaningless to the outcome of a game.

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 few penalties and a positive point differential. On right would be teams with high penalty yards and negative point differentials. Do you see that below?



9.1 A more predictive example

So we've **firmly** established that penalties aren't predictive. But what is?

So instead of looking at penalty yards, let's make a new metric: Net Yards. Can we predict the score differential by looking at the yards a team gained minus the yards they gave up.

```
regressions <- newcorrelations %>% mutate(NetYards = OffensiveYards - DefYards)
```

First, let's look at the correlation coefficent.

```
regressions %>%
  summarise(correlation = cor(differential, NetYards, method="pearson"))
```

```
## # A tibble: 1 x 1
##   correlation
##         <dbl>
## 1       0.850
```

Answer: 85 percent. Not a perfect relationship, but very good. But how meaningful is that relationship and how predictive is it?

```
net <- lm(differential ~ NetYards, data = regressions)
summary(net)
```

```
##
## Call:
## lm(formula = differential ~ NetYards, data = regressions)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.981  -8.566   0.171   8.832  39.361
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.309946   0.302520   1.025    0.306
## NetYards    0.106536   0.001623  65.644   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.29 on 1660 degrees of freedom
## Multiple R-squared:  0.7219, Adjusted R-squared:  0.7217
## F-statistic:  4309 on 1 and 1660 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 -.0000000000000000022. 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: 0.7217. So we can predict a whopping 72 percent of the difference in the score differential by simply looking at the net yards the team has.

Third, the coefficients: In this case, our `y=mx+b` formula looks like `y = .106536x + .309946`. So if we were applying this, let's look at Nebraska's 27-24 loss to Iowa in 2019. Nebraska's net yards that game? -40.

```
(.106536*-40)+.309946
```

```
## [1] -3.951494
```

So by our model, Nebraska should have lost by 3.95 points. That's really close to the 3 point groin kick of a loss that happened.

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]
```

```
##
## -- 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()
## )
## i Use `spec()` for the 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")`

```
library(Hmisc)
```

```
## Loading required package: lattice
```

```
## Loading required package: survival
## Loading required package: Formula
##
## Attaching package: 'Hmisc'
## The following objects are masked from 'package:dplyr':
##
##     src, summarize
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

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*0.516176)
```

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*-0.43640
```

```
## [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. Take our model at the end of the correlation and regression chapter. Our model predicted Nebraska, given a -40 net yardage deficit would lose to Iowa by -3.95 points. They lost by -3. So our residual is -.95. If Iowa fakes that last second field goal and scores a touchdown, our residual would have been -7.

Residuals can tell you several things, but most important 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. And the more robust the model – the better your r-squared value is – the more meaningful that label of under or overperforming is.

Let's go back to our net yards model.

Let's first attach libraries and get some data.

```
library(tidyverse)
```

```
logs <- read_csv("data/footballlogs19.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
```

```
##   Result = col_character(),
##   TeamFull = col_character(),
##   TeamURL = col_character(),
##   Outcome = col_character(),
##   Team = col_character(),
##   Conference = col_character()
## )
## i Use `spec()` for the full column specifications.
```

First, let's make the columns we'll need.

```
residualmodel <- logs %>% mutate(differential = TeamScore - OpponentScore, NetYards = (
```

Now let's create our model.

```
fit <- lm(differential ~ NetYards, data = residualmodel)
summary(fit)
```

```
##
## Call:
## lm(formula = differential ~ NetYards, data = residualmodel)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.981  -8.566   0.171   8.832  39.361
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.309946   0.302520   1.025    0.306
## NetYards    0.106536   0.001623  65.644   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.29 on 1660 degrees of freedom
## Multiple R-squared:  0.7219, Adjusted R-squared:  0.7217
## F-statistic:  4309 on 1 and 1660 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 39 points (!), and a team has overperformed it by 39 points (!!). The median residual, where half are above and half are below, is just slightly above the fit line. Close here is good.

Next: Look at the Adjusted R-squared value. What that says is that 72 percent of a team's scoring output can be predicted by their net yards.

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. But if you know a little bit about football, it doesn't surprise you that the more you outgain your opponent, the more you win by. It's an intuitive result.

What we want to do now is look at those residuals. We want to add them to our individual game records. We can do that by creating two new fields – predicted and residuals – to our dataframe like this:

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

Now we can sort our data by those residuals. Sorting in descending order gives us the games where teams overperformed the model. To make it easier to read, I'm going to use select to give us just the columns we need to see.

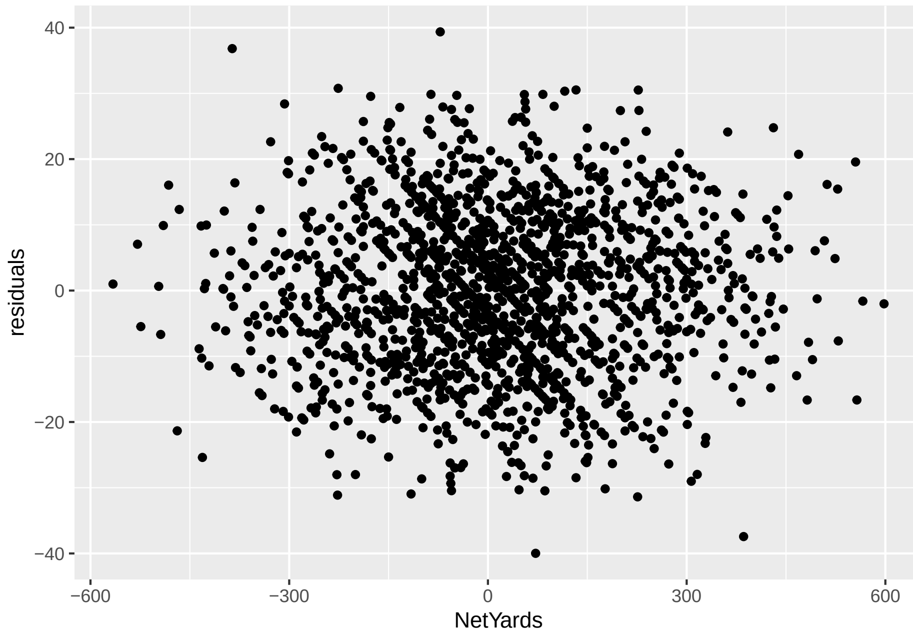
```
residualmodel %>% arrange(desc(residuals)) %>% select(Team, Opponent, Result, NetYards, residuals
```

```
## # A tibble: 1,662 x 5
##    Team           Opponent             Result    NetYards residuals
##    <chr>          <chr>                <chr>        <dbl>     <dbl>
##  1 Penn State     Buffalo              W (45-13)      -72      39.4
##  2 Illinois       Nebraska             L (38-42)     -386      36.8
##  3 Virginia Tech  Miami (FL)           W (42-35)     -226      30.8
##  4 Tennessee      Chattanooga          W (45-0)       133      30.5
##  5 Baylor         Kansas               W (61-6)       227      30.5
##  6 Syracuse       Duke                 W (49-6)       116      30.3
##  7 Houston        North Texas          W (46-25)      -86      29.9
##  8 South Florida  South Carolina State W (55-16)       83      29.8
##  9 Troy           Texas State          W (63-27)       55      29.8
## 10 Miami (FL)     Louisville           W (52-27)      -47      29.7
## # ... with 1,652 more rows
```

So looking at this table, what you see here are the teams who scored more than their net yards would indicate. One of them should jump off the page at you.

Remember Nebraska vs Illinois? We came back to win and everyone was happy and relieved at the same time? We outgained Illinois by **386 yards** in that game and won by 4. Our model predicted Nebraska should have won that game by 36.8 points. Illinois outscored the model by almost as many points as they had. Just goes to show you: you can have all the advantages and you can still screw it up. Just ask Buffalo: They were only outgained by Penn State by 70 yards and still lost by 32.

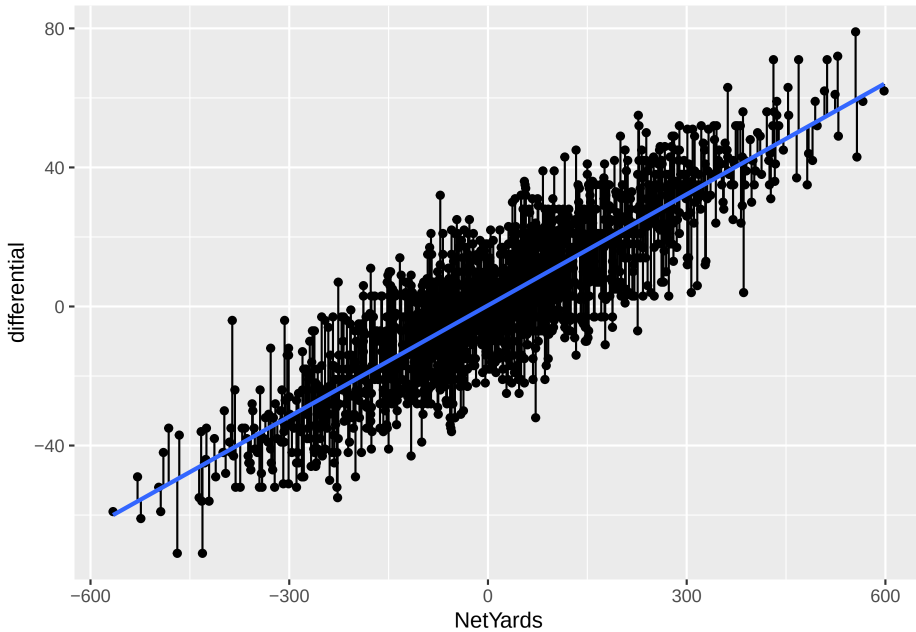
But, before we can bestow any validity on this model, we need to see if this linear model is appropriate. We've done that some looking at our p-values and R-squared values. But one more check is to look at the residuals themselves. We do that by plotting the residuals with the predictor. We'll get into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

```
## `geom_smooth()` using formula 'y ~ x'
```



The blue line here separates underperformers from overperformers.

11.1 Penalties

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

```
penalties <- logs %>%
  mutate(
    differential = TeamScore - OpponentScore,
    TotalPenalties = Penalties+DefPenalties,
    TotalPenaltyYards = PenaltyYds+DefPenaltyYds
  )
```

```
pfit <- lm(differential ~ TotalPenaltyYards, data = penalties)
summary(pfit)
```

```
##
## Call:
## lm(formula = differential ~ TotalPenaltyYards, data = penalties)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -73.13 -15.16   0.71  15.61  76.86
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)        2.785946   1.525993   1.826   0.0681 .
## TotalPenaltyYards -0.007407   0.013244  -0.559   0.5760
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.3 on 1660 degrees of freedom
## Multiple R-squared:  0.0001884,  Adjusted R-squared:  -0.0004139
## F-statistic: 0.3128 on 1 and 1660 DF,  p-value: 0.576
```

So from top to bottom:

- Our min and max go from -73 to positive 77
- Our adjusted R-squared is … -0.0004139. Not much at all.
- Our p-value is … 0.576, which is more than than .05.

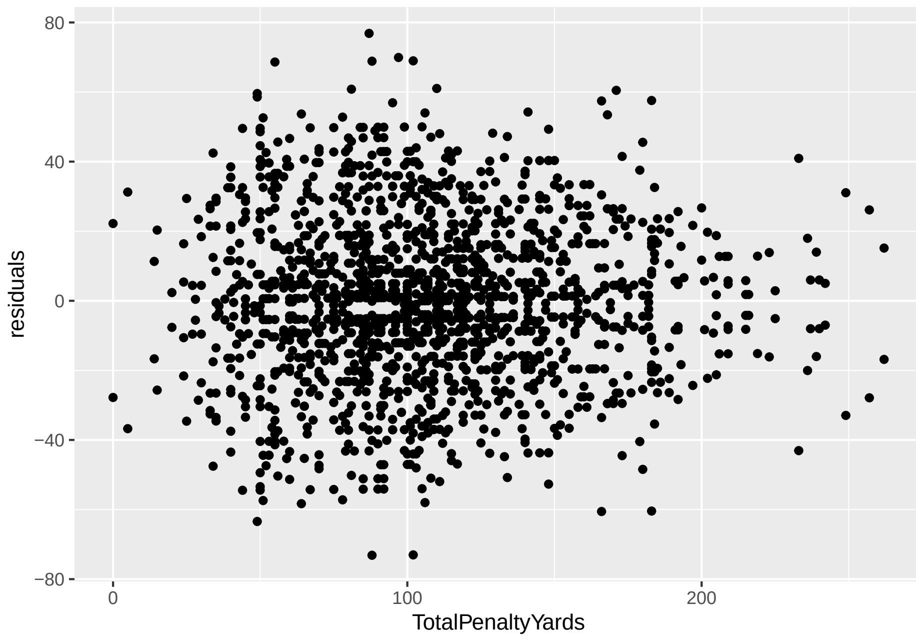
So what we can say about this model is that it's statistically insignificant and utterly meaningless. Normally, we'd stop right here – why bother going forward with a predictive model that isn't predictive? But let's do it anyway.

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

```
penalties %>% arrange(desc(residuals)) %>% select(Team, Opponent, Result, TotalPenaltyY
```

```
## # A tibble: 1,662 x 5
##    Team           Opponent            Result    TotalPenaltyYards residuals
##    <chr>          <chr>               <chr>                 <dbl>     <dbl>
##  1 Maryland       Howard              W (79-0)                 87      76.9
##  2 Penn State     Idaho               W (79-7)                 97      69.9
##  3 Ohio State     Miami (OH)          W (76-5)                102      69.0
##  4 Oregon         Nevada              W (77-6)                 88      68.9
##  5 Louisiana      Texas Southern      W (77-6)                 55      68.6
##  6 Miami (FL)     Bethune-Cookman     W (63-0)                110      61.0
##  7 Alabama        Western Carolina    W (66-3)                 81      60.8
##  8 UCF            Florida A&M         W (62-0)                171      60.5
##  9 South Carolina Charleston Southern W (72-10)                49      59.6
## 10 Wisconsin      Central Michigan    W (61-0)                 49      58.6
## # ... with 1,652 more rows
```

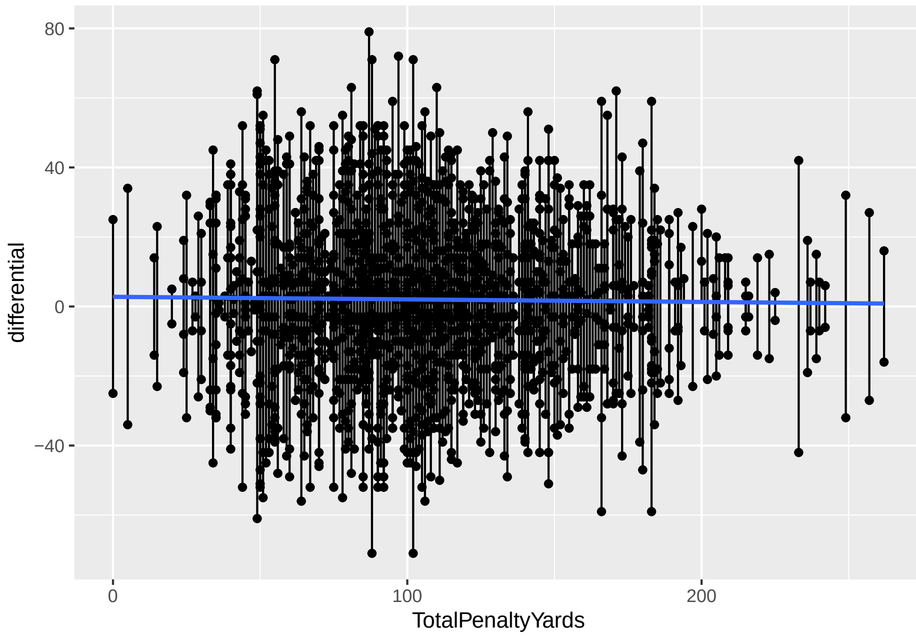
First, note all of the biggest misses here are all blowout games. The worst game of the season last year was, by far, Maryland vs Howard, a 79-0 shellacking that should never have happened. The differential was 79 points. The model missed that differential by … 76.9 points. In other words, this model is terrible. But let's look at it anyway.



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. Another bad sign? A flat fit line. It means there's no relationship between these two things. Which we already know.

```
## `geom_smooth()` using formula 'y ~ x'
```

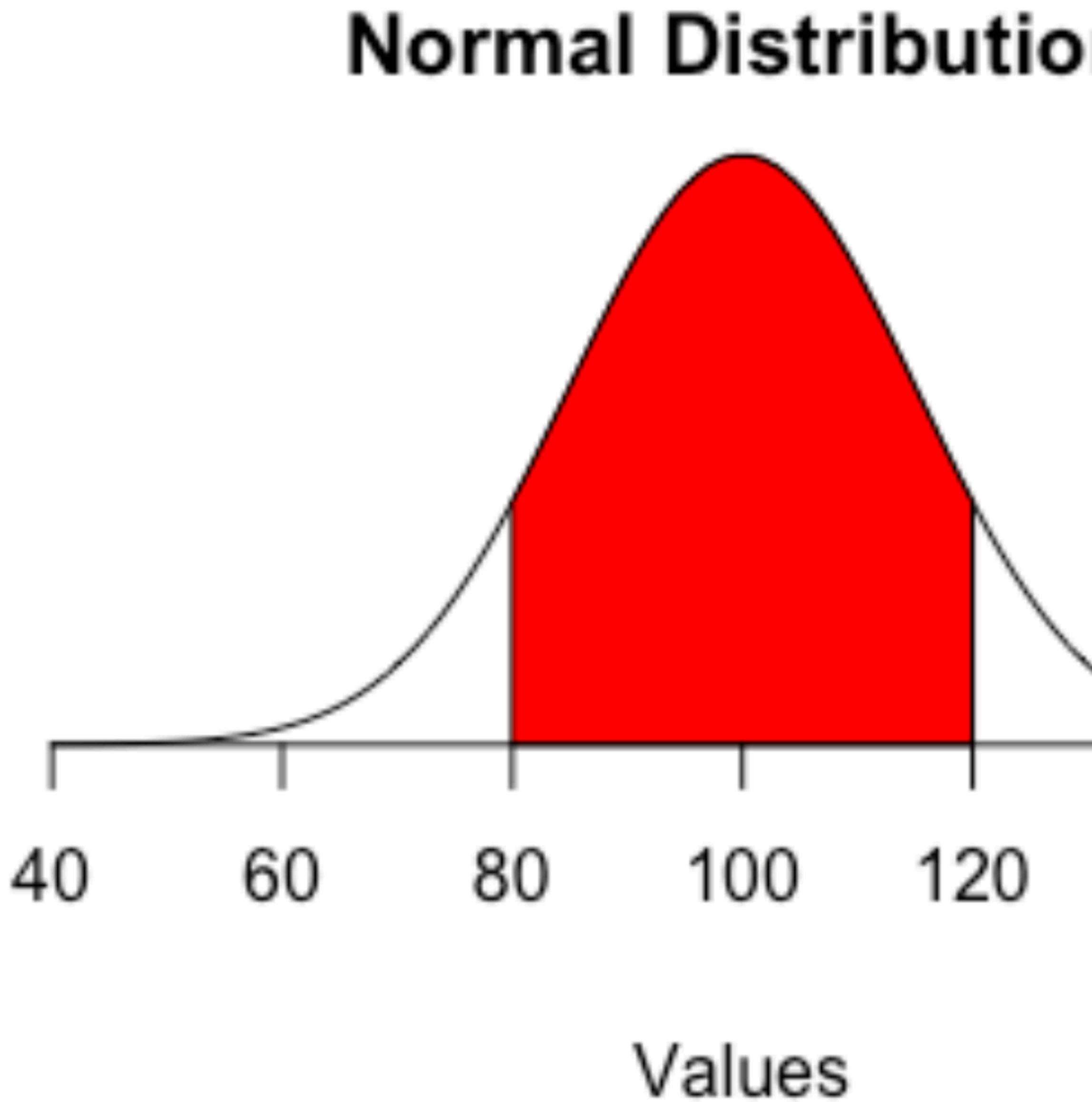

80
40
0
−40
differential
0
100
200
TotalPenaltyYards

Chapter 12

Z-scores

Z-scores are a handy way to standardize numbers so you can compare things across groupings or time. In this class, 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.

A z-score is a measure of how a particular stat is from the mean. It's measured in standard deviations from that mean. A standard deviation is a measure of how much variation – how spread out – numbers are in a data set. What it means here, with regards to z-scores, is that zero is perfectly average. If it's 1, it's one standard deviation above the mean, and 34 percent of all cases are between 0 and 1.



If you think of the normal distribution, it means that 84.3 percent of all case are below that 1. If it were -1, it would mean the number is one standard deviation below the mean, and 84.3 percent of cases would be above that -1. So if you have numbers with z-scores of 3 or even 4, that means that number is waaaaaay above the mean.

So let's use last year's Nebraska basketball team, which if haven't been paying attention to current events, was not good at basketball.

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 for the 2019-2020 season.

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

```
##
## -- 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()
## )
## i Use `spec()` for the full column specifications.
```

The first thing we need to do is select some fields we think represent team quality and a few things to help us keep things straight. So I'm going to pick shooting percentage, rebounding and the opponent version of the same two:

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

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)
    )
```

```
## `summarise()` regrouping output by 'Conference' (override with `.groups` argument)
```

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    0.505   -0.619
## 2 A-10       Dayt~ 0.525       32.5    0.413      29.0    2.59     0.0352
## 3 A-10       Duqu~ 0.444       32.4    0.427      32.4    0.216   -0.0168
## 4 A-10       Ford~ 0.384       30.0    0.402      33.9   -1.53    -1.13
## 5 A-10       Geor~ 0.424       33.8    0.440      30.5   -0.358    0.620
## 6 A-10       Geor~ 0.422       30.5    0.452      32.7   -0.410   -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    2.36      1.13
##  3 WCC        Gonz~ 0.517       37.4     0.424      28.2    1.73      1.90
##  4 Southland  Step~ 0.490       34.2     0.427      26.6    1.76      1.05
##  5 Big Ten    Mich~ 0.460       37.7     0.382      29.6    1.38      1.55
##  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    1.58      1.52
##  8 A-10       Dayt~ 0.525       32.5     0.413      29.0    2.59      0.0352
##  9 A-10       Sain~ 0.457       37.4     0.403      30.5    0.598     2.21
## 10 ACC        Loui~ 0.457       36.6     0.392      29.8    1.11      1.37
## # ... with 343 more rows, and 3 more variables: OppZscore <dbl>,
## #   OppRebZScore <dbl>, TotalZscore <dbl>
```

Don't sleep on the Anteaters! Would have been a tough out at the tournament that never happened.

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       37.7    0.382      29.6    1.38      1.55
##  2 Big Ten    Rutg~ 0.449       37      0.385      31.1    0.727     1.22
##  3 Big Ten    Ohio~ 0.447       33.6    0.400      28.4    0.592    -0.393
##  4 Big Ten    Illi~ 0.444       36.1    0.418      29.1    0.439     0.779
##  5 Big Ten    Indi~ 0.445       35.1    0.419      29.4    0.480     0.306
##  6 Big Ten    Mary~ 0.419       36.1    0.401      31.9   -0.952     0.794
##  7 Big Ten    Mich~ 0.463       33.0    0.428      31.9    1.56     -0.682
##  8 Big Ten    Penn~ 0.432       35.6    0.411      34.2   -0.237     0.550
##  9 Big Ten    Minn~ 0.426       35.5    0.411      33     -0.560     0.520
## 10 Big Ten    Iowa~ 0.452       34.2    0.430      32.4    0.918    -0.104
## 11 Big Ten    Purd~ 0.418       33.8    0.410      29.3   -1.02     -0.271
## 12 Big Ten    Wisc~ 0.426       31.3    0.410      32.0   -0.587    -1.49
## 13 Big Ten    Nort~ 0.417       30.5    0.422      34.8   -1.12     -1.84
## 14 Big Ten    Nebr~ 0.408       32.4    0.453      42.2   -1.62     -0.947
## # ... 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.

12.2 Writing about z-scores

The great thing about z-scores is that they make it very easy for you, the sports analyst, to create your own measures of who is better than who. The downside: Only a small handful of sports fans know what the hell a z-score is.

As such, you should try as hard as you can to avoid writing about them.

If the word z-score appears in your story or in a chart, you need to explain what it is. “The ranking uses a statistical measure of the distance from the mean called a z-score” is a good way to go about it. You don’t need a full stats textbook definition, just a quick explanation. And keep it simple.

Never use z-score in a headline. Write around it. Away from it. Z-score in a headline is attention repellent. You won’t get anyone to look at it. So “Tottenham tops in z-score” bad, “Tottenham tops in the Premiere League” good.

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 Wickham, 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 either write the code that adds necessary and required parts to finish it, or you'll export your work into a program like Illustrator to finish the work.

To begin, we'll import libraries as we have all along. We'll read in the data, then create a new dataframe that represents our attendance data, similar to what we've done before.

```
library(tidyverse)
```

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

```
##
## -- 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 in the most recent season we have data. We'll use the same tricks we used in the filtering assignment.

```
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.

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

```
## Selecting by 2018
```

13.1 The bar chart

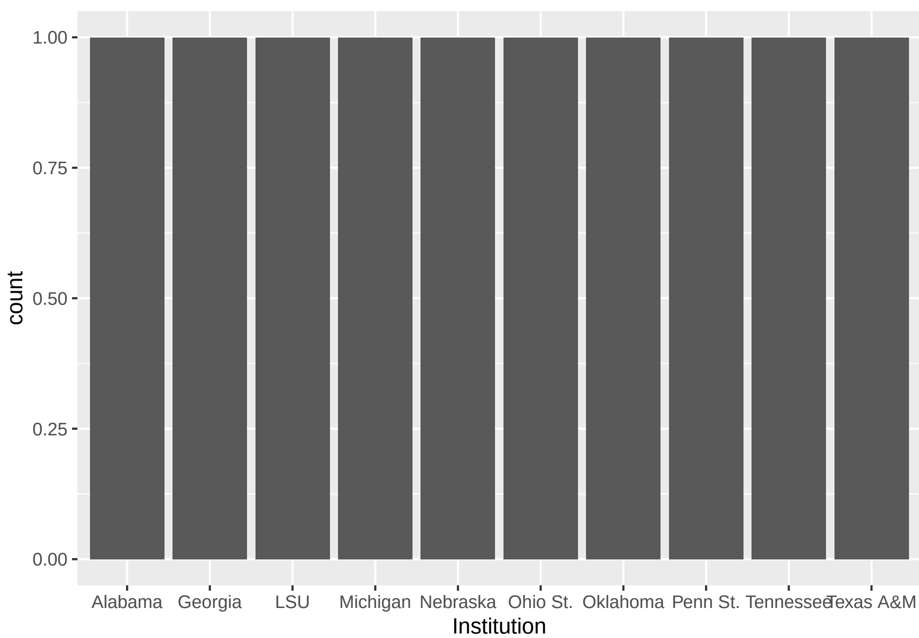
The easiest thing we can do is create a simple bar chart of our data. **Bar charts show magnitude. They invite you to compare how much more or less one thing is compared to others.**

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?

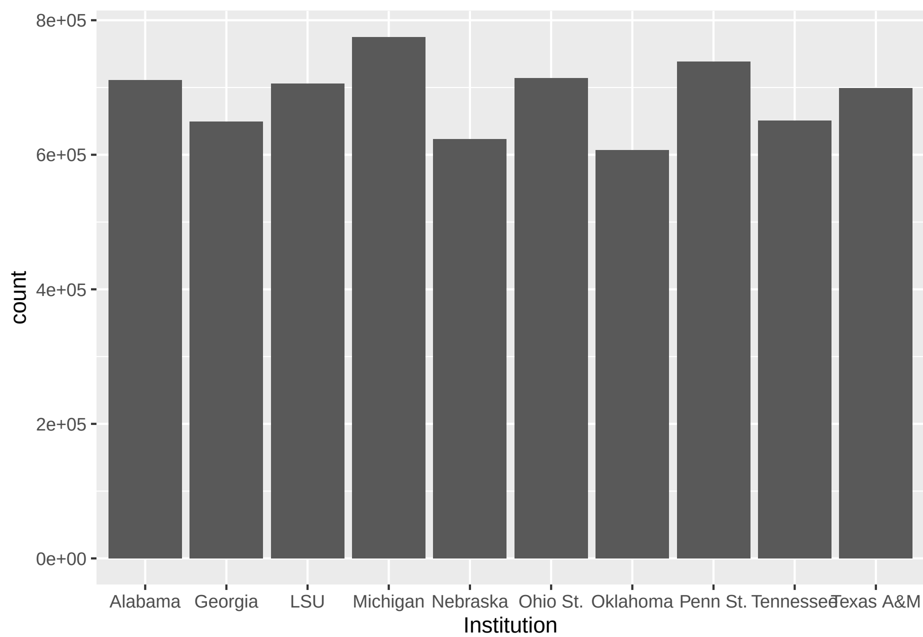
```
ggplot(top10, aes(x=Institution)) + geom_bar()
```



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.

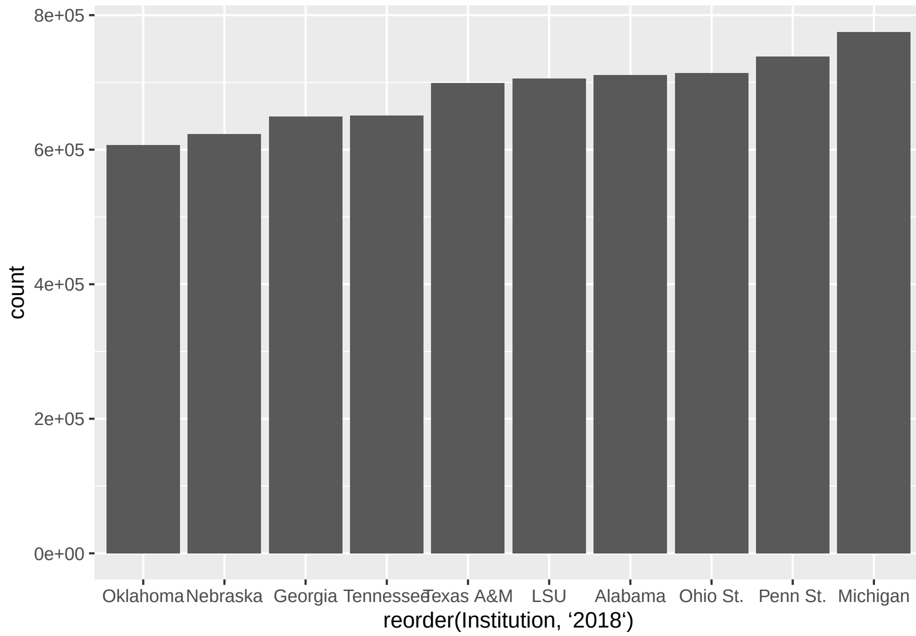
```
ggplot(top10, aes(x=Institution, weight=`2018`)) +
  geom_bar()
```



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

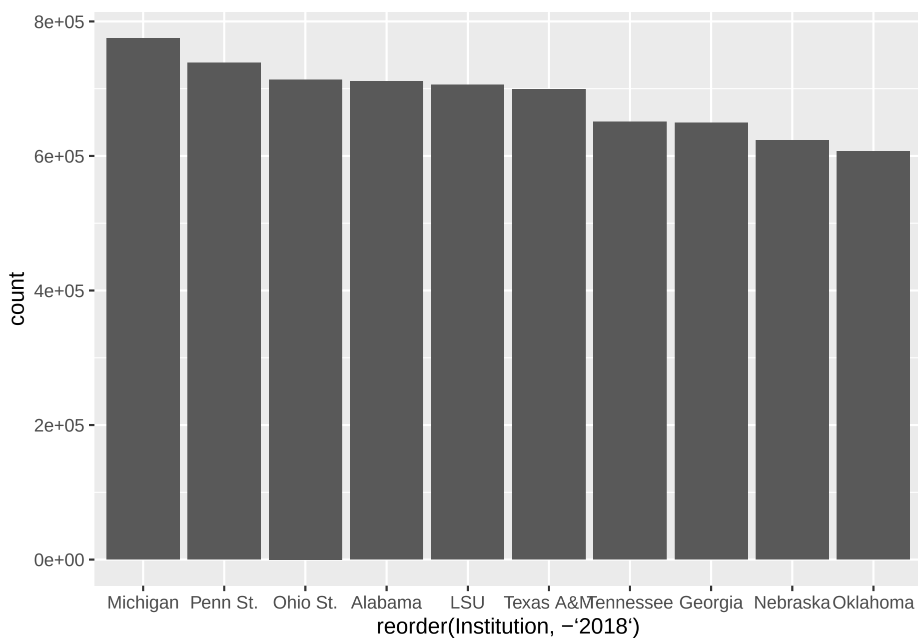
Let's deal with the ordering first. `ggplot2`'s default behavior is to sort the data by the x axis variable. So it's in alphabetical order. To change that, we have to `reorder` it. With `reorder`, we first have to tell `ggplot` what we are reordering, and then we have to tell it HOW we are reordering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

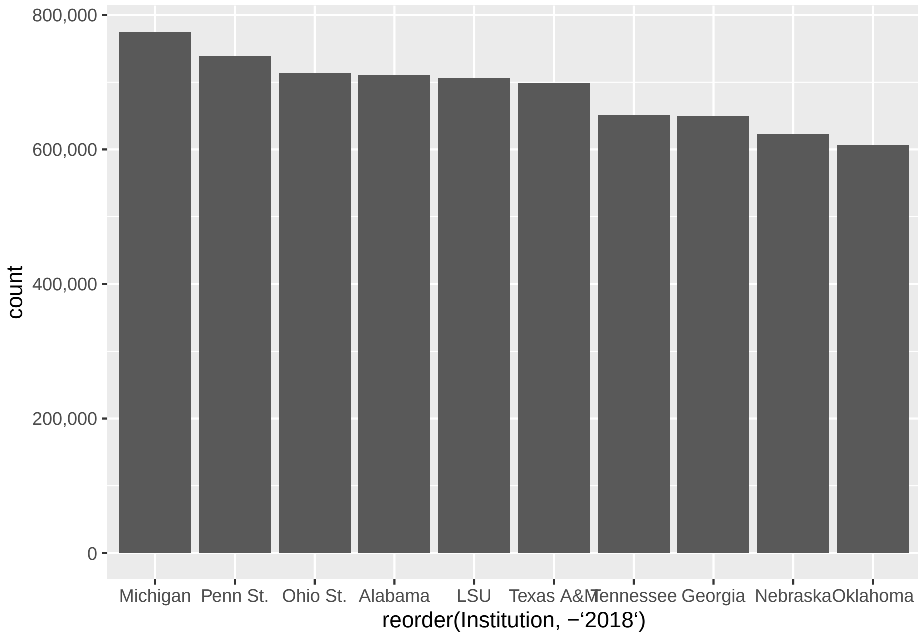
```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

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)
```

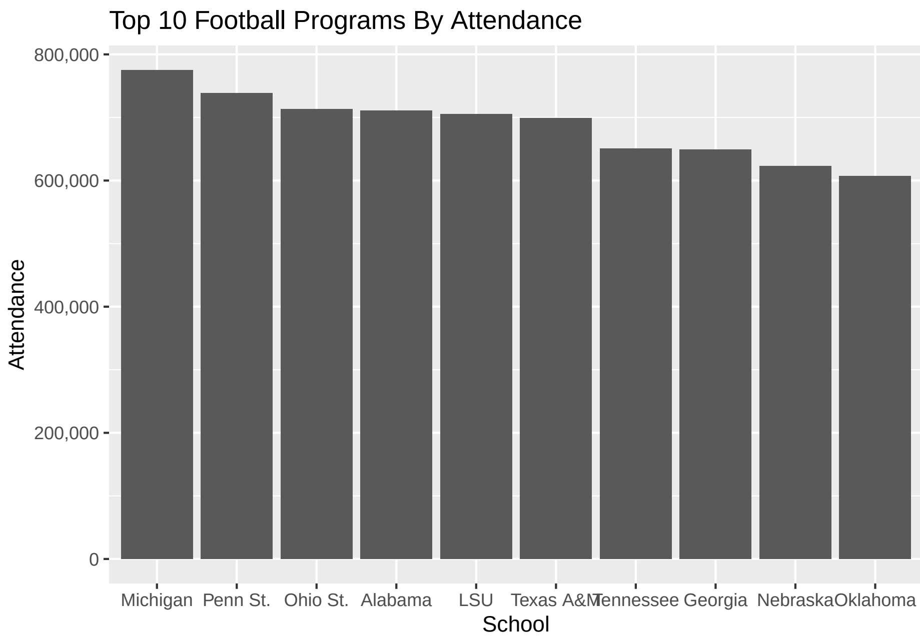


Better.

13.3 Styling

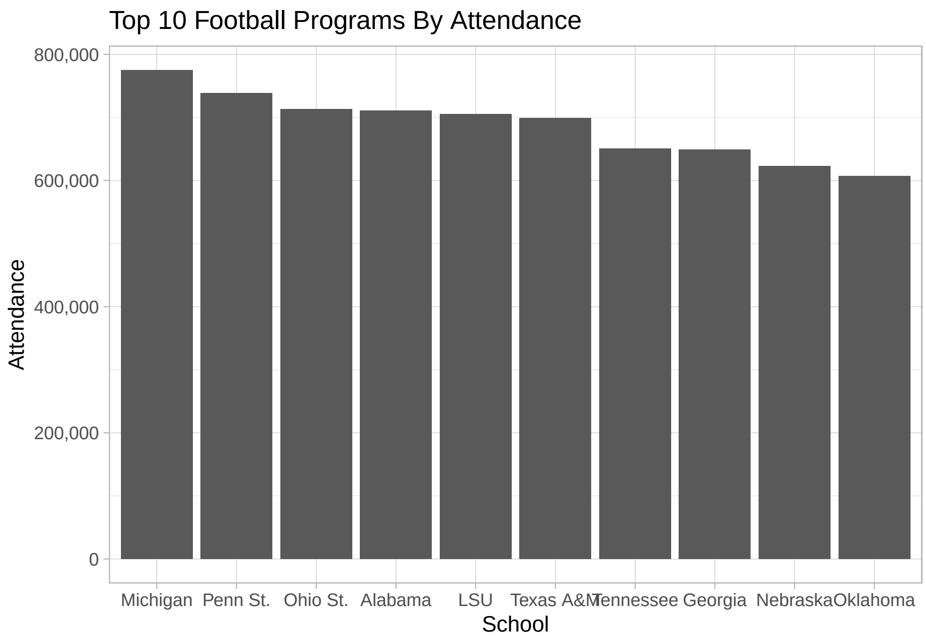
We are going to spend a lot more time on styling, but let's add some simple labels to this with a new bit called `labs` which is short for labels.

```
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"
)
```



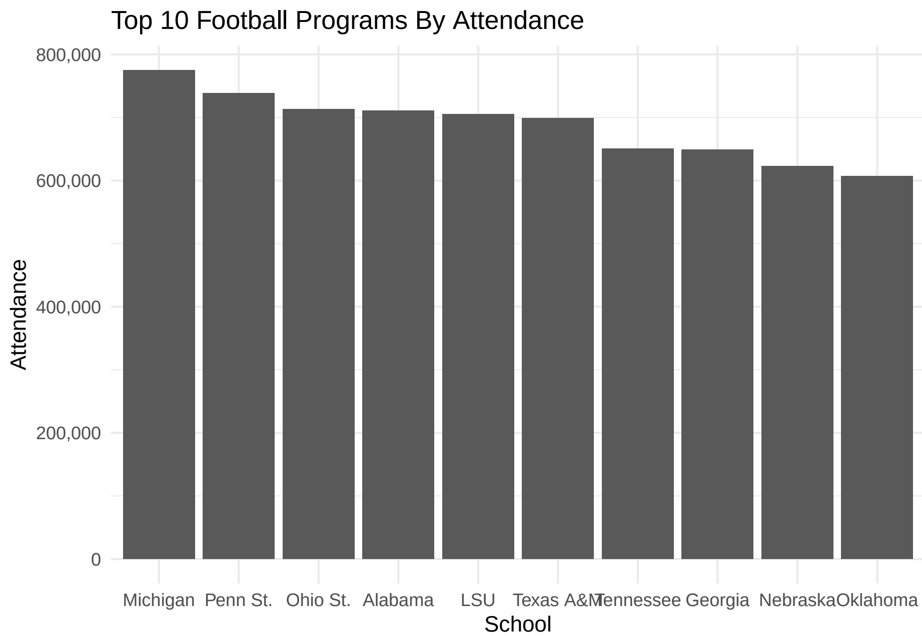
The library has lots and lots of ways to alter the styling – we can programmatically control nearly every part of the look and feel of the chart. One simple way is to apply themes in the library already. We do that the same way we've done other things – we add them. Here's the light theme.

```
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()
```

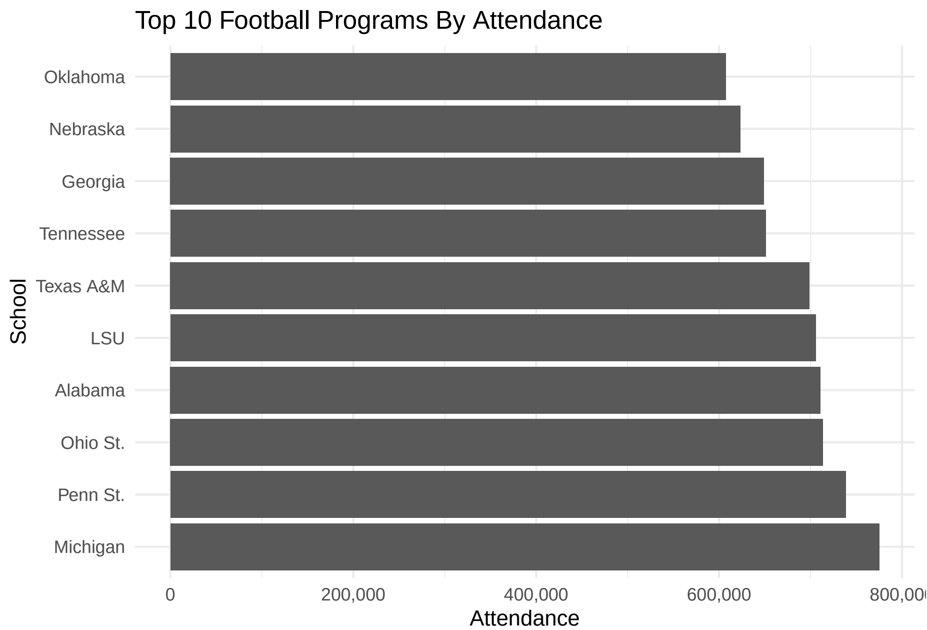


Later on, we'll write our own themes. For now, the built in ones will get us closer to something that looks good.

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
200,000
400,000
600,000
800,00
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 college football games last season. You can get it here.

```
football <- read_csv("data/footballlogs19.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   Result = col_character(),
##   TeamFull = col_character(),
##   TeamURL = col_character(),
##   Outcome = col_character(),
##   Team = col_character(),
##   Conference = col_character()
## )
## i Use `spec()` for the full column specifications.
```

What we have here is every game in college football last season. The question we want to answer is this: Who had the most prolific offenses in the Big Ten last season? And how did they get there?

So to make this chart, we have to just add one thing to a bar chart like we did in the previous chapter. However, it's not that simple.

We have game data, and we need season data. To get that, we need to do some group by and sum work. And since we're only interested in the Big Ten, we have some filtering to do too. For this, we're going to measure offensive production by rushing yards and passing yards. So if we have all the games a team played, and the rushing and passing yards for each of those games, what we need to do to get the season totals is just add them up.

```
football %>%
  group_by(Conference, Team) %>%
  summarise(
    SeasonRushingYards = sum(RushingYds),
    SeasonPassingYards = sum(PassingYds),
  ) %>% filter(Conference == "Big Ten Conference")
```

```
## `summarise()` regrouping output by 'Conference' (override with `.groups` argument)
```

```
## # A tibble: 14 x 4
## # Groups:   Conference [1]
##    Conference         Team           SeasonRushingYards SeasonPassingYards
##    <chr>              <chr>                       <dbl>              <dbl>
##  1 Big Ten Conference Illinois                     1877               2409
##  2 Big Ten Conference Indiana                      1700               3931
##  3 Big Ten Conference Iowa                         1789               2976
##  4 Big Ten Conference Maryland                     2033               2088
##  5 Big Ten Conference Michigan                     1966               3261
##  6 Big Ten Conference Michigan State               1666               3182
##  7 Big Ten Conference Minnesota                    2323               3293
##  8 Big Ten Conference Nebraska                     2454               2551
##  9 Big Ten Conference Northwestern                 2162               1404
## 10 Big Ten Conference Ohio State                   3742               3684
## 11 Big Ten Conference Penn State                   2496               2877
## 12 Big Ten Conference Purdue                       1002               3719
## 13 Big Ten Conference Rutgers                      1612               1672
## 14 Big Ten Conference Wisconsin                    3272               2802
```

By looking at this, we can see we got what we needed. We have 14 teams and numbers that look like season totals for yards. Save that to a new dataframe.

```
football %>%
  group_by(Conference, Team) %>%
  summarise(
    SeasonRushingYards = sum(RushingYds),
    SeasonPassingYards = sum(PassingYds),
```

```
  ) %>% filter(Conference == "Big Ten Conference") -> yards
```

```
## `summarise()` regrouping output by 'Conference' (override with `.groups` argument)
```

Now, the problem we have is that ggplot wants long data and this data is wide. So we need to use `tidyr` to make it long, just like we did in the transforming data chapter.

```
yards %>%
  pivot_longer(
    cols=starts_with("Season"),
    names_to="Type",
    values_to="Yards")
```

```
## # A tibble: 28 x 4
## # Groups:   Conference [1]
##    Conference         Team     Type               Yards
##    <chr>              <chr>    <chr>              <dbl>
##  1 Big Ten Conference Illinois SeasonRushingYards  1877
##  2 Big Ten Conference Illinois SeasonPassingYards  2409
##  3 Big Ten Conference Indiana  SeasonRushingYards  1700
##  4 Big Ten Conference Indiana  SeasonPassingYards  3931
##  5 Big Ten Conference Iowa     SeasonRushingYards  1789
##  6 Big Ten Conference Iowa     SeasonPassingYards  2976
##  7 Big Ten Conference Maryland SeasonRushingYards  2033
##  8 Big Ten Conference Maryland SeasonPassingYards  2088
##  9 Big Ten Conference Michigan SeasonRushingYards  1966
## 10 Big Ten Conference Michigan SeasonPassingYards  3261
## # ... with 18 more rows
```

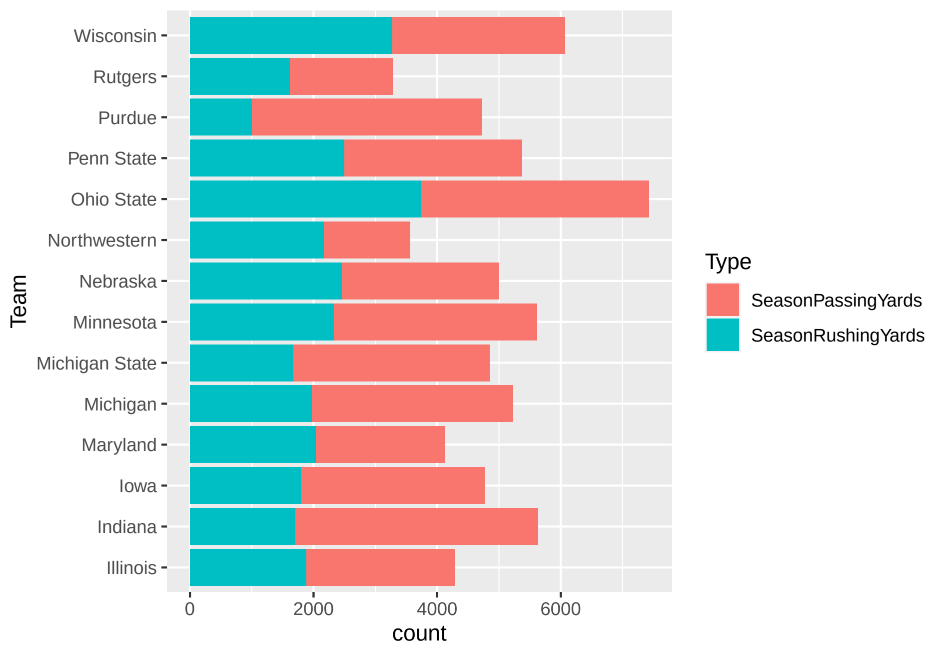
What you can see now is that we have two rows for each team: One for rushing yards, one for passing yards. This is what ggplot needs. Save it to a new dataframe.

```
yards %>%
  pivot_longer(
    cols=starts_with("Season"),
    names_to="Type",
    values_to="Yards") -> yardswide
```

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 we are 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 Type, which means it will stack the number of rushing yards on top of passing yards and we can see how they compare.

```
ggplot(yardswide, aes(x=Team, weight=Yards, fill=Type)) +
  geom_bar() +
```

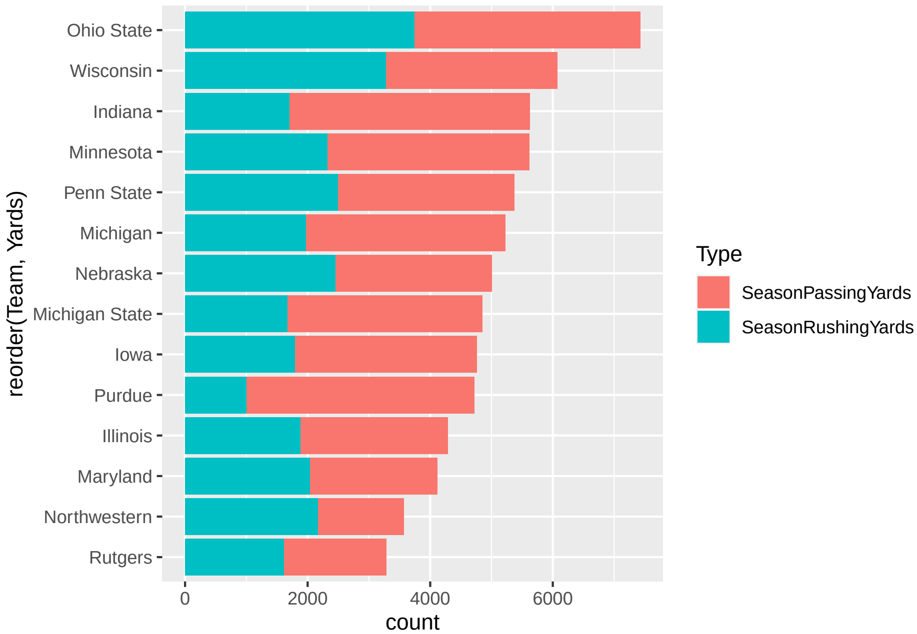
```
  coord_flip()
```



What's the problem with this chart?

There's a couple of things, one of which we'll deal with now: The ordering is alphabetical (from the bottom up). So let's `reorder` the teams by Yards.

```
ggplot(yardswide, aes(x=reorder(Team, Yards), weight=Yards, fill=Type)) +
  geom_bar() +
  coord_flip()
```



And just like that, Ohio State comes out on top, with Wisconsin second and … Indiana? … third. Huh.

Chapter 15

Waffle charts

Pie charts are the devil. They should be an instant F in any data visualization class. The problem? How carefully can you evaluate angles and area? Unless they are blindingly obvious and only a few categories, not well. If you've got 25 categories, how can you tell the difference between 7 and 9 percent? You can't.

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. We want a newer version of the `waffle` library than is in CRAN – where you normally get libraries from – so copy and paste this into your console:

```
install.packages("waffle")
```

Now load it:

```
library(waffle)
```

15.1 Waffles two ways: Part 1

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

Maybe the easiest way to do waffle charts, at least at first, is to make vectors of your data and plug them in. To make a vector, we use the `c` or concatenate function.

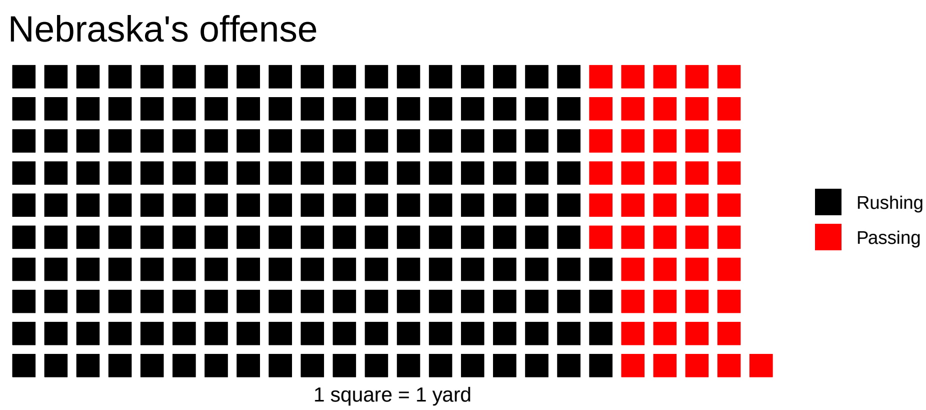
So let's look at offense. Rushing vs passing.

```
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.

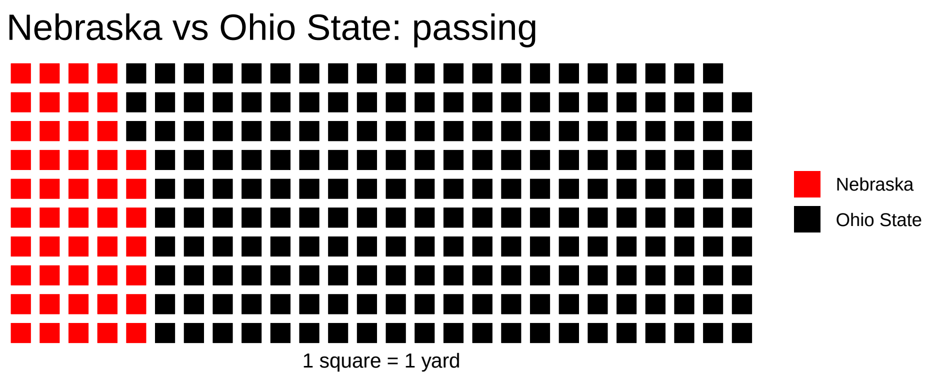
```
waffle(
        nu,
        rows = 10,
        title="Nebraska's offense",
        xlab="1 square = 1 yard",
        colors = c("black", "red")
)
```



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

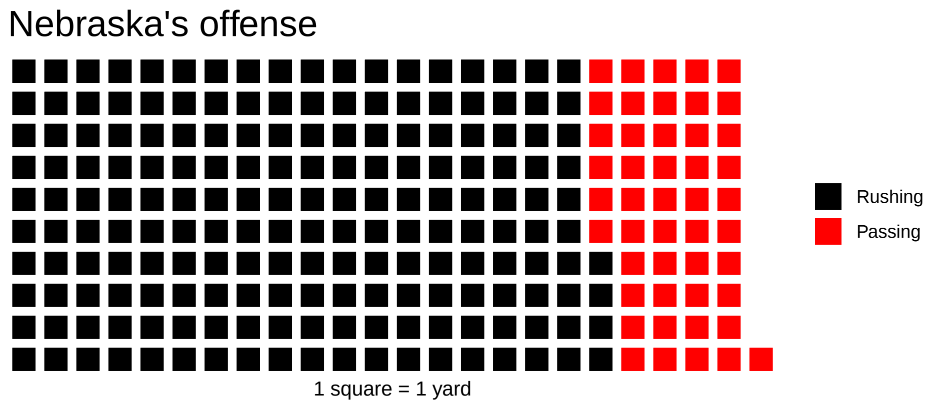
```
passing <- c("Nebraska"=47, "Ohio State"=212)
```

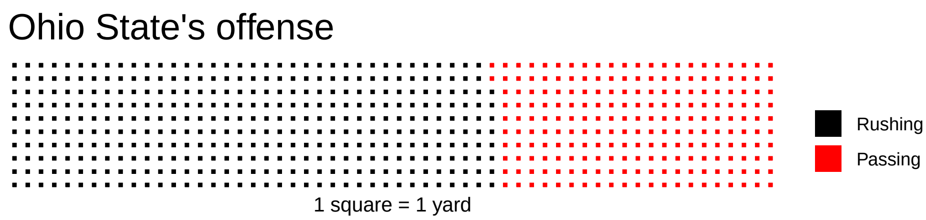
```
waffle(
        passing,
        rows = 10,
        title="Nebraska vs Ohio State: passing",
        xlab="1 square = 1 yard",
        colors = c("red", "black")
)
```



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





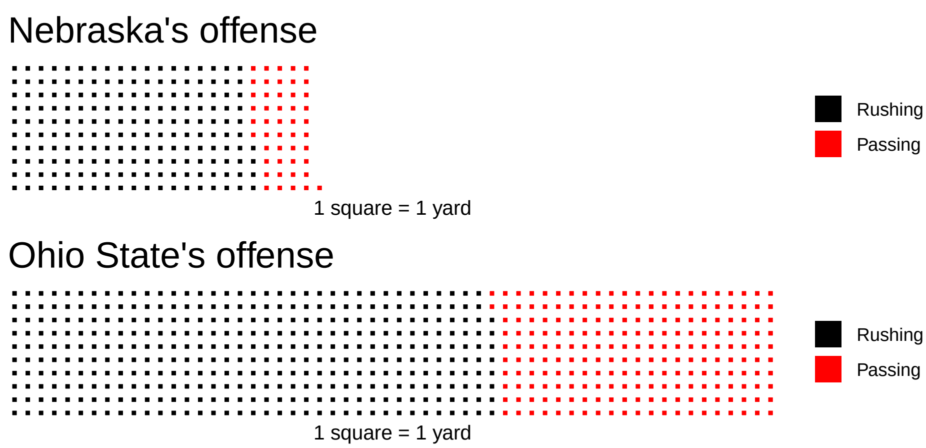
What do you notice about this chart? Notice how the squares aren't the same size? Well, Ohio State out-gained Nebraska by a long way. So the squares aren't the same size because the numbers aren't the same. We can fix that by adding an unnamed padding number so the number of yards add up to the same 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.

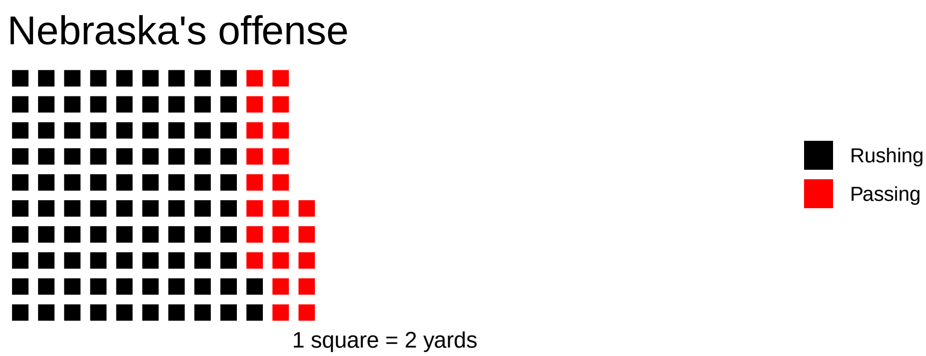
```
iron(
 waffle(nu,
        rows = 10,
        title="Nebraska's offense",
        xlab="1 square = 1 yard",
        colors = c("black", "red", "white")
        ),
 waffle(oh,
        rows = 10,
        title="Ohio State's offense",
        xlab="1 square = 1 yard",
        colors = c("black", "red",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

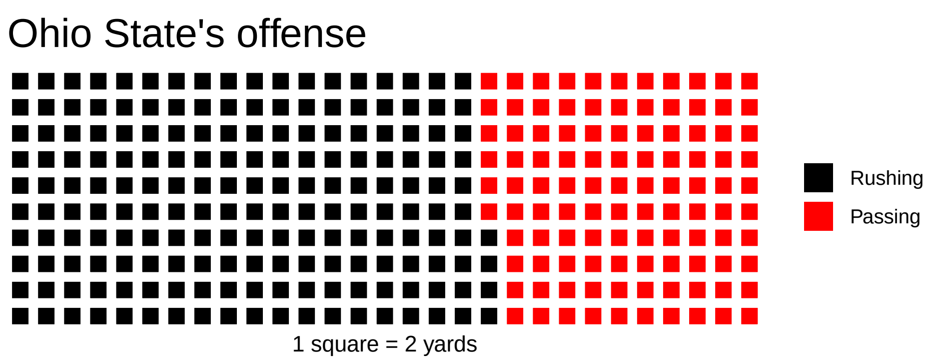
```
        )
)
```



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





News flash: Ohio State crushed Nebraska.

15.2 Waffles two ways: Part 2

For this part, we want a newer version of the `waffle` library than is in CRAN – where you normally get libraries from.

WARNING: This didn't work in a variety of environments, so it may not work on yours.

Copy and paste this into your console:

`install.packages("waffle", repos = "https://cinc.rud.is")`

At first, this way might seem harder than doing it the way we just walked through, but the benefits will come later when it's far, far easier to style this chart, where the previous charts are harder.

We have the log of every game in college football – you can get it here – and we can find this game with some simple filtering.

```
fblogs <- read_csv("data/footballlogs19.csv")
```

```
##
## -- Column specification --------------------------------------------------------
```

```
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   Result = col_character(),
##   TeamFull = col_character(),
##   TeamURL = col_character(),
##   Outcome = col_character(),
##   Team = col_character(),
##   Conference = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
fblogs %>% filter(Team == "Nebraska" & Opponent == "Ohio State")
```

```
## # A tibble: 1 x 54
##    Game Date       HomeAway Opponent Result PassingCmp PassingAtt PassingPct
##   <dbl> <date>     <chr>    <chr>    <chr>       <dbl>      <dbl>      <dbl>
## 1     5 2019-09-28 <NA>     Ohio St~ L (7--          8         17       47.1
## # ... with 46 more variables: PassingYds <dbl>, PassingTD <dbl>,
## #   RushingAtt <dbl>, RushingYds <dbl>, RushingAvg <dbl>, RushingTD <dbl>,
## #   OffensivePlays <dbl>, OffensiveYards <dbl>, OffenseAvg <dbl>,
## #   FirstDownPass <dbl>, FirstDownRush <dbl>, FirstDownPen <dbl>,
## #   FirstDownTotal <dbl>, Penalties <dbl>, PenaltyYds <dbl>, Fumbles <dbl>,
## #   Interceptions <dbl>, TotalTurnovers <dbl>, TeamFull <chr>, TeamURL <chr>,
## #   Outcome <chr>, TeamScore <dbl>, OpponentScore <dbl>, DefPassingCmp <dbl>,
## #   DefPassingAtt <dbl>, DefPassingPct <dbl>, DefPassingYds <dbl>,
## #   DefPassingTD <dbl>, DefRushingAtt <dbl>, DefRushingYds <dbl>,
## #   DefRushingAvg <dbl>, DefRushingTD <dbl>, DefPlays <dbl>, DefYards <dbl>,
## #   DefAvg <dbl>, DefFirstDownPass <dbl>, DefFirstDownRush <dbl>,
## #   DefFirstDownPen <dbl>, DefFirstDownTotal <dbl>, DefPenalties <dbl>,
## #   DefPenaltyYds <dbl>, DefFumbles <dbl>, DefInterceptions <dbl>,
## #   DefTotalTurnovers <dbl>, Team <chr>, Conference <chr>
```

That's the game. So now we need to make this long data – same as we did with the stacked bar charts – and we'll focus on total yards.

```
fblogs %>%
        filter(Team == "Nebraska" & Opponent == "Ohio State") %>%
        select(Team, OffensiveYards, DefYards) %>%
        pivot_longer(
                cols=c("OffensiveYards", "DefYards"),
                names_to="Type",
                values_to="Yards"
                )
```

```
## # A tibble: 2 x 3
```

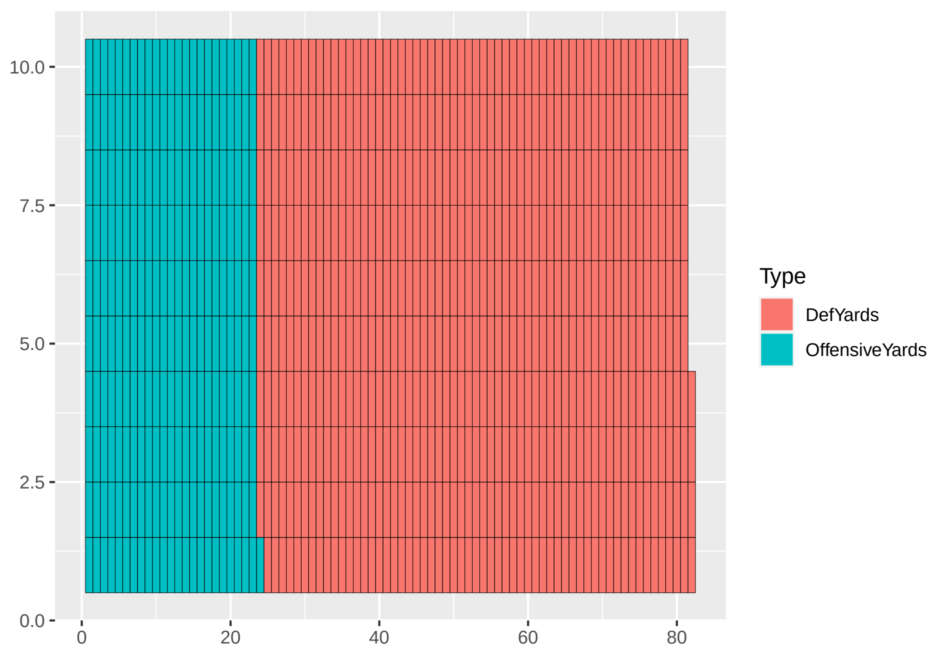
```
##   Team     Type           Yards
##   <chr>    <chr>          <dbl>
## 1 Nebraska OffensiveYards   231
## 2 Nebraska DefYards         583
```

That does what we want, so let's save that to a new dataframe.

```
nuoh <- fblogs %>%
        filter(Team == "Nebraska" & Opponent == "Ohio State") %>%
        select(Team, OffensiveYards, DefYards) %>%
        pivot_longer(
                cols=c("OffensiveYards", "DefYards"),
                names_to="Type",
                values_to="Yards"
                )
```

Now we can use a new geom – `geom_waffle` – that the `waffle` library has added to `ggplot`. The `geom_waffle` takes two required inputs: `fill` and `value`, but otherwise, it looks the same as previous things we've done.

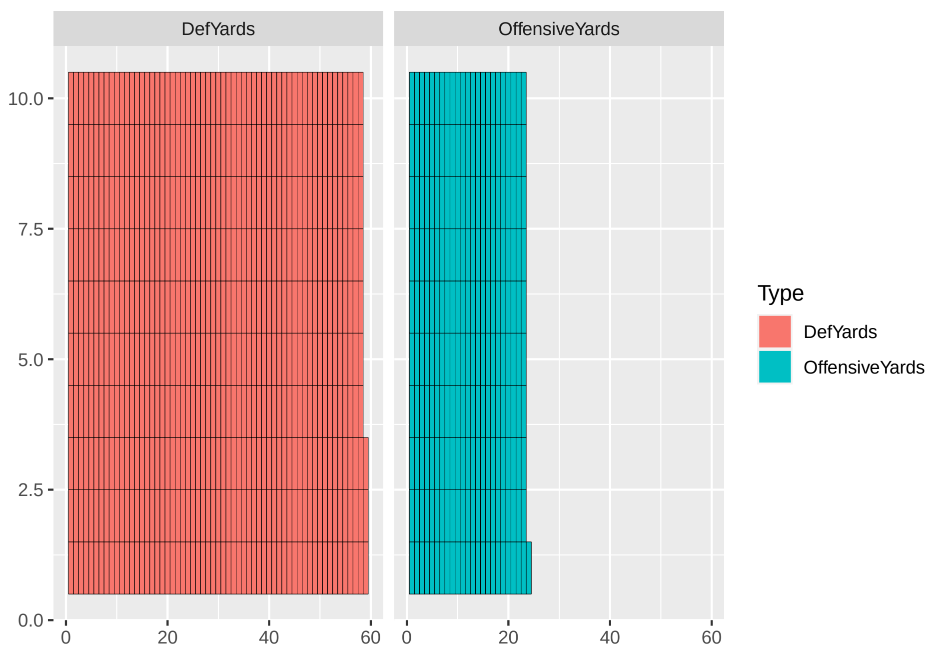
```
ggplot() + geom_waffle(
  data=nuoh,
  aes(fill=Type, values=Yards))
```



First, we can see that going this route changes the boxes to narrow rectangles. That's to fill the space given.

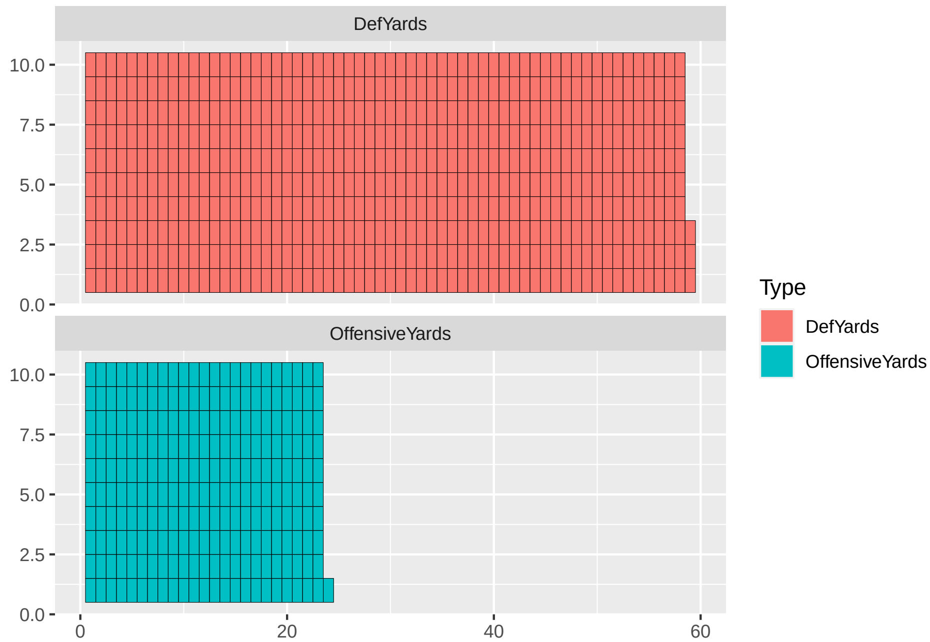
If we want to split them, we can use something called a `facet_wrap` which we will spend a whole class on later, so don't worry about this now. Just know we can split it by `Type` this way.

```
ggplot() + geom_waffle(
  data=nuoh,
  aes(fill=Type, values=Yards)
) + facet_wrap(~Type)
```



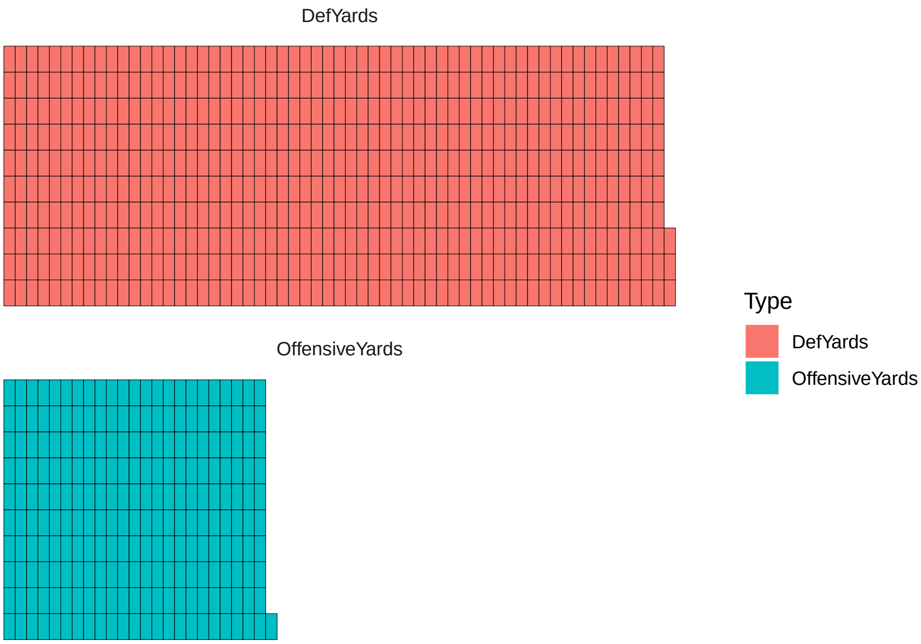
Now we can stack the two charts so they aren't side by side using `ncol` inside the `facet_wrap`.

```
ggplot() + geom_waffle(
  data=nuoh,
  aes(fill=Type, values=Yards)
) + facet_wrap(~Type, ncol=1)
```



Now it's just a matter of formatting, labeling and general cleanup. We'll focus on that later as well, but here's a quick way to get started, which we did in the bar chart chapter.

```
ggplot() + geom_waffle(
  data=nuoh,
  aes(fill=Type, values=Yards)
) +
    facet_wrap(~Type, ncol=1) +
    theme_minimal() +
    theme_enhance_waffle()
```


DefYards
OffensiveYards
Type
DefYards
OffensiveYards

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]
```

```
##
## -- 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()
## )
## i Use `spec()` for the full column specifications.
```

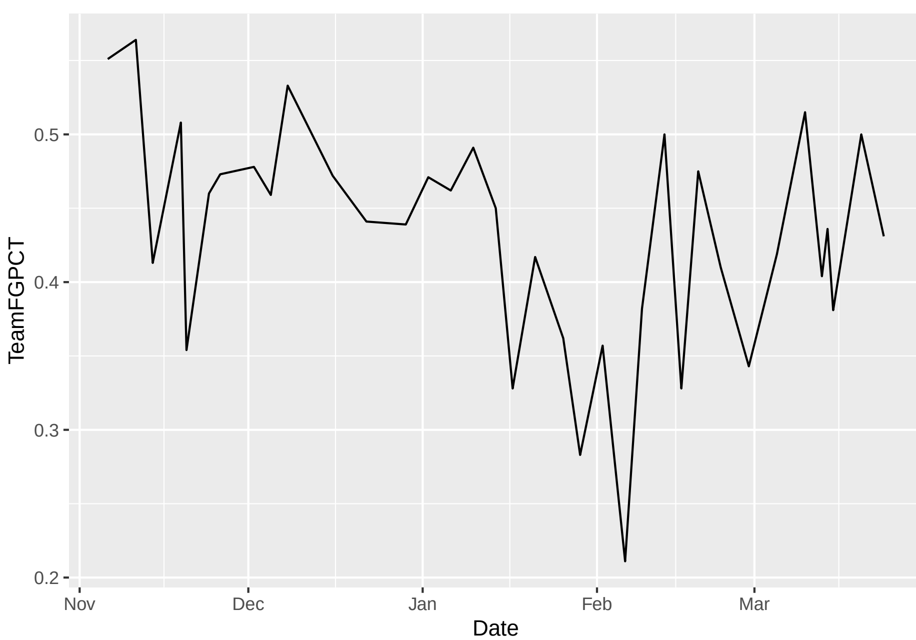
This data has every game from every team in it, so we need to use filtering to limit it, because we just want to look at Nebraska. If you don't remember, flip back to chapter 5.

```
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.

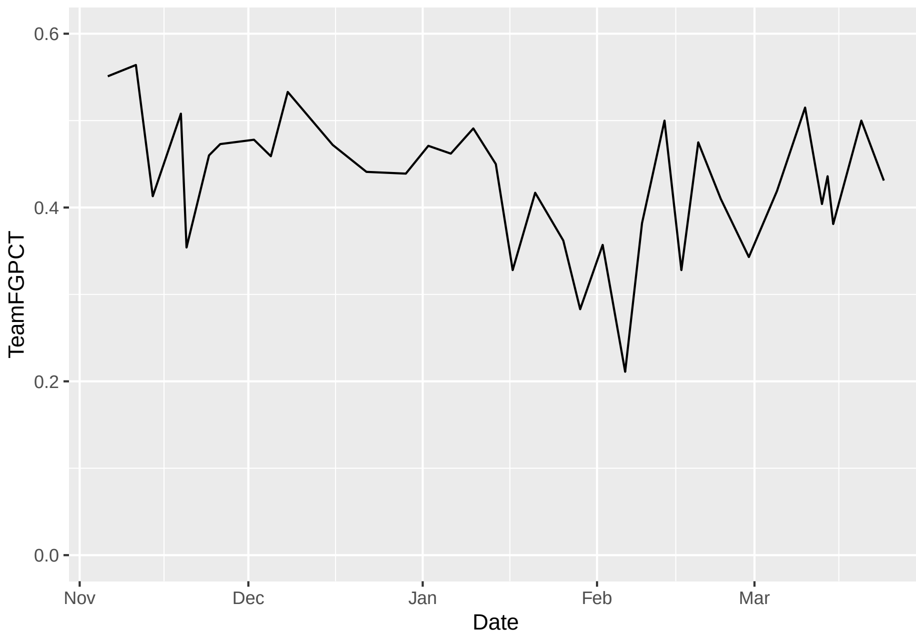
Line charts, unlike bar charts, do have a y-axis. So in our ggplot step, we have to define what our x and y axes are. In this case, the x axis is our Date – the most common x axis in line charts is going to be a date of some variety – and y in this case is up to us. We've seen from previous walkthroughs 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.

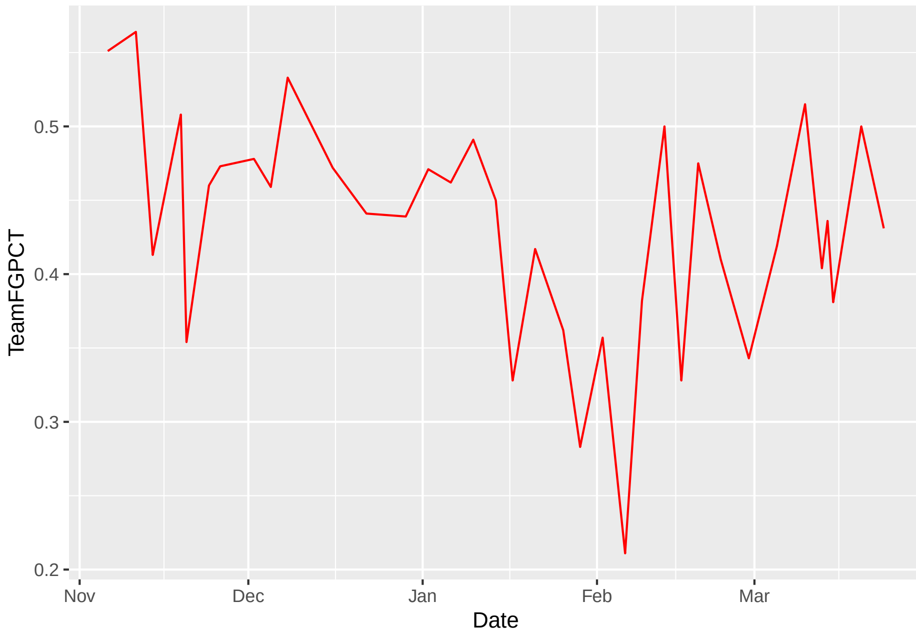
With datasets, we want to invite comparison. So let's answer the question visually. Let's put two lines on the same chart. How does Nebraska compare to Michigan State and Purdue, the eventual regular season co-champions?

```
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.

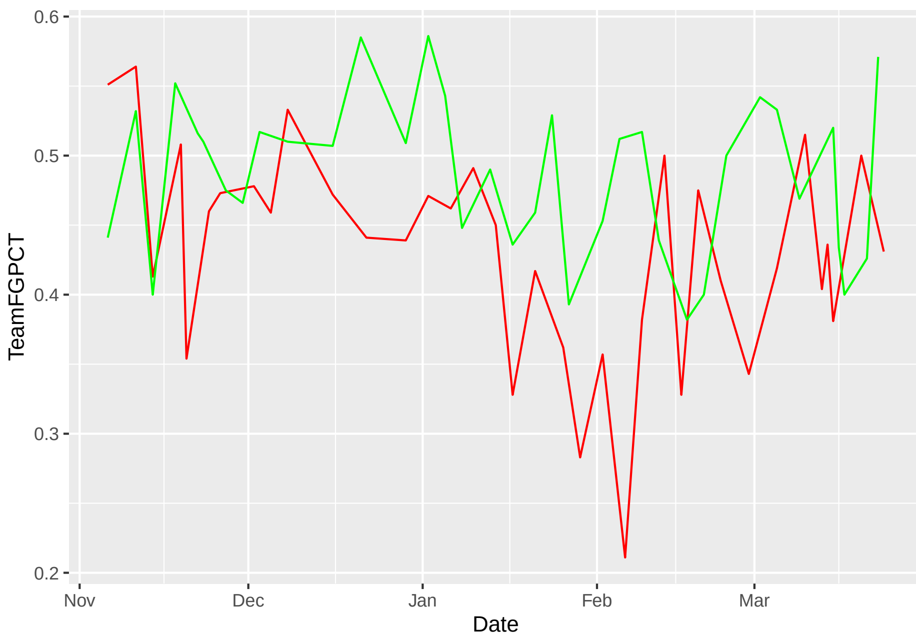
First, let's chart Nebraska. Read carefully. First we set the data. Then we set our aesthetic. Unlike bars, we need an X and a Y variable. In this case, our X is the date of the game, Y is the thing we want the lines to move with. In this case, the Team Field Goal Percentage – TeamFGPCT.

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



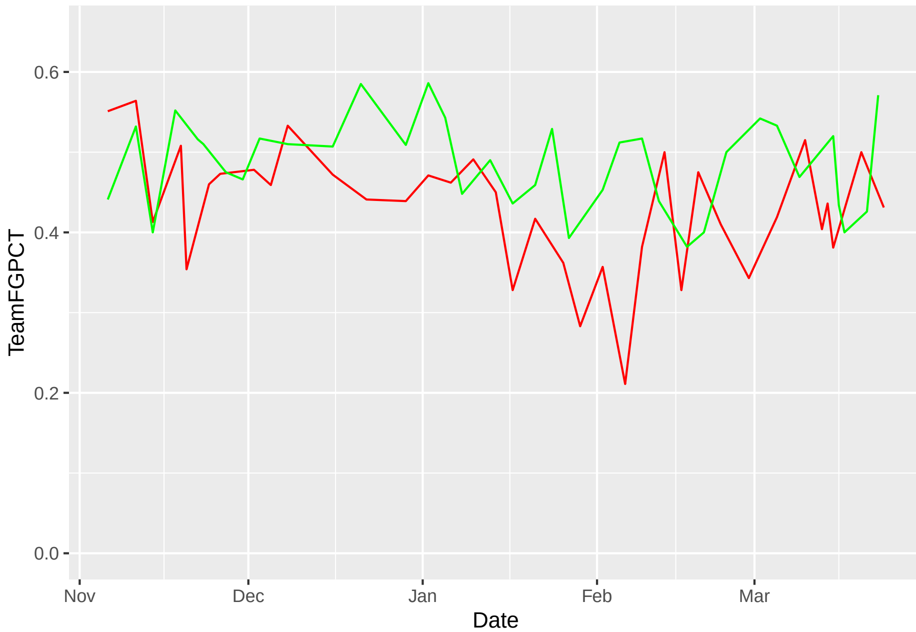
Now, by using +, we can add Michigan State to it. REMEMBER COPY AND PASTE IS A THING. Nothing changes except what data you are using.

```
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.

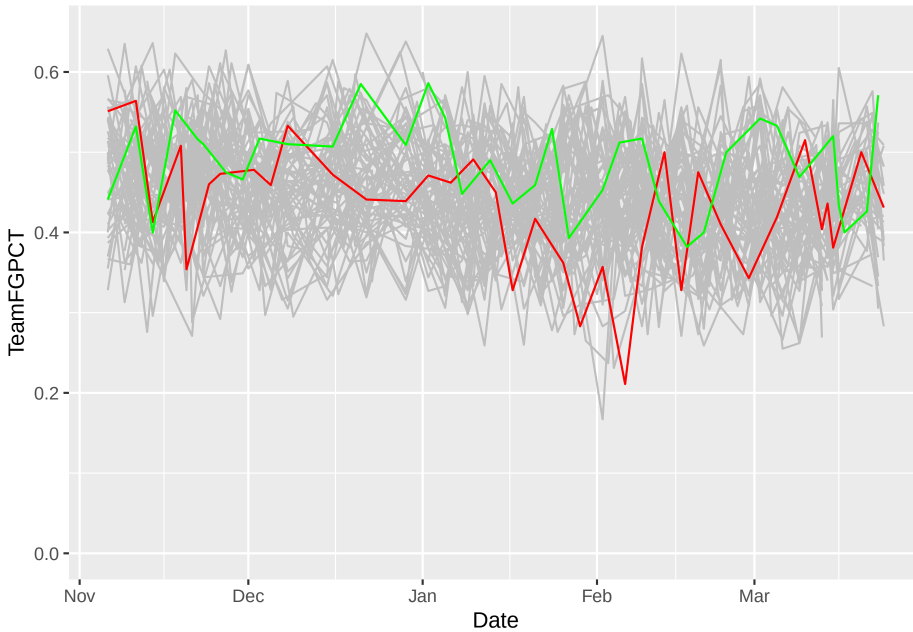
Fine. How about all Power Five Schools? This data for example purposes. You don't have to do it.

```
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
```



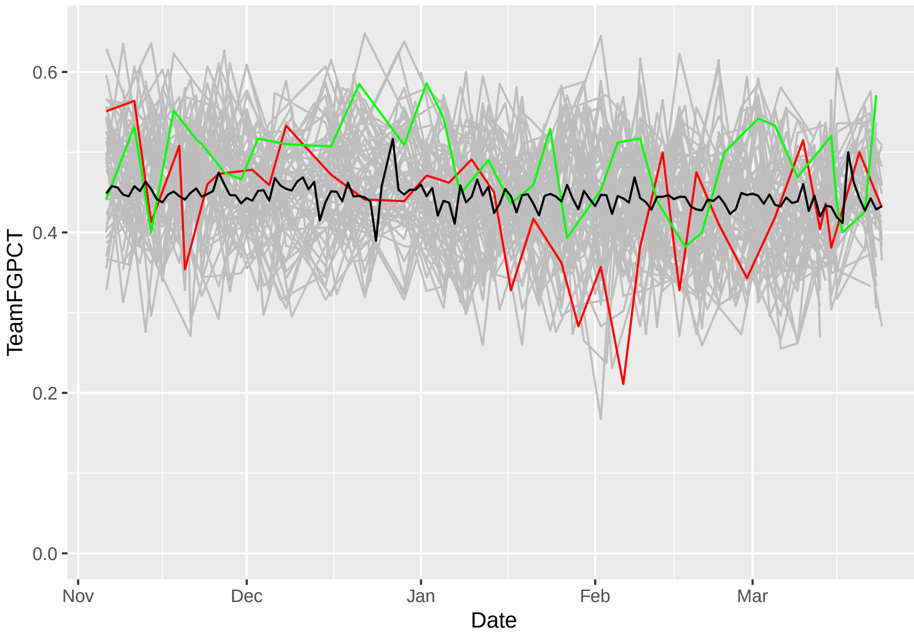
What do we see here? How has Nebraska and Michigan State's season evolved against all the rest of the teams in college basketball?

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))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

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


TeamFGPCT
0.0
0.2
0.4
0.6
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]
```

```
##
## -- 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()
## )
## i Use `spec()` for the 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.

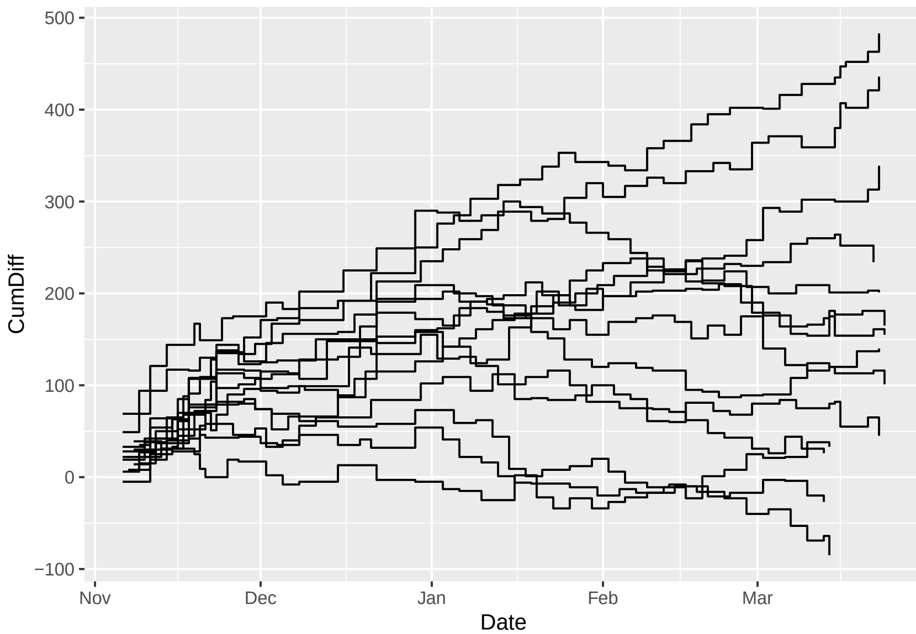
```
difflogs <- logs %>%
  mutate(Differential = TeamScore - OpponentScore) %>%
  group_by(Team) %>%
  mutate(CumDiff = cumsum(Differential))
```

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")
```

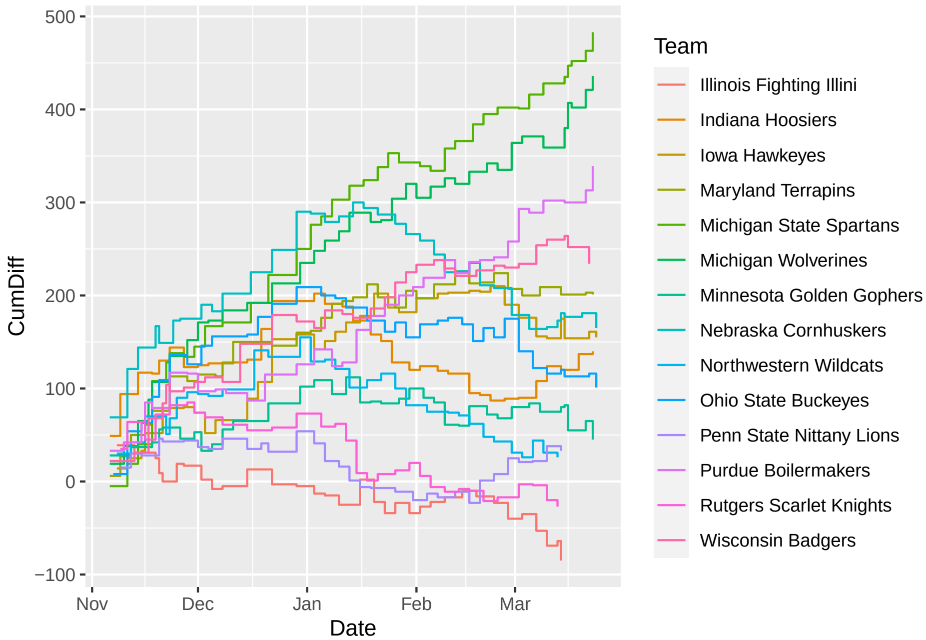
The step chart is it's own geom, so we can employ it just like we have the others. It works almost exactly the same as a line chart, but it uses the cumulative sum instead of a regular value and, as the name implies, creates a step like shape to the line instead of a curve.

```
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.

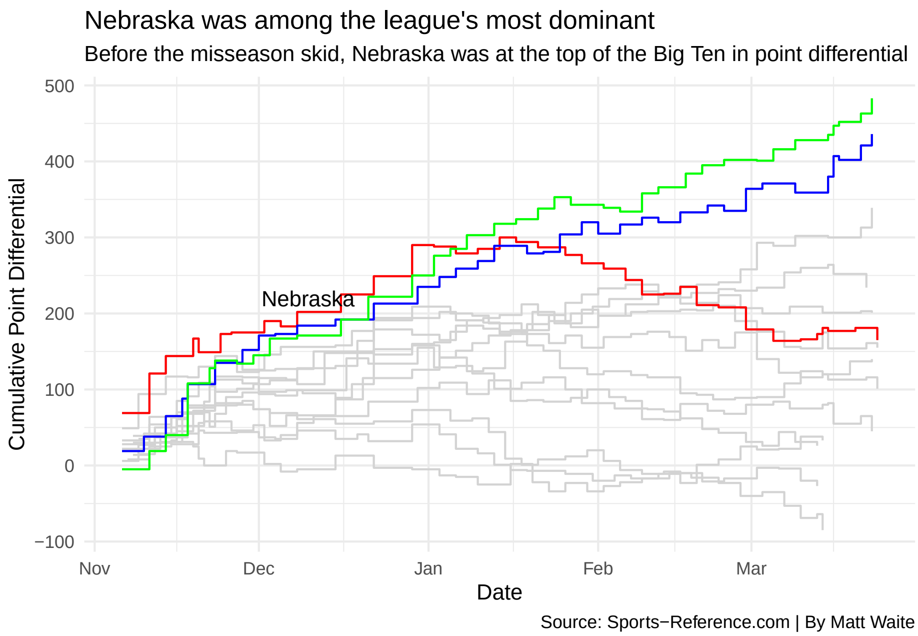
Let's look at those top teams plus Nebraska.

```
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]
```

```
##
## -- 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()
```

```
## )
## i Use `spec()` for the 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
```

```
## `summarise()` regrouping output by 'Team', 'Conference' (override with `.groups` arg
```

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.

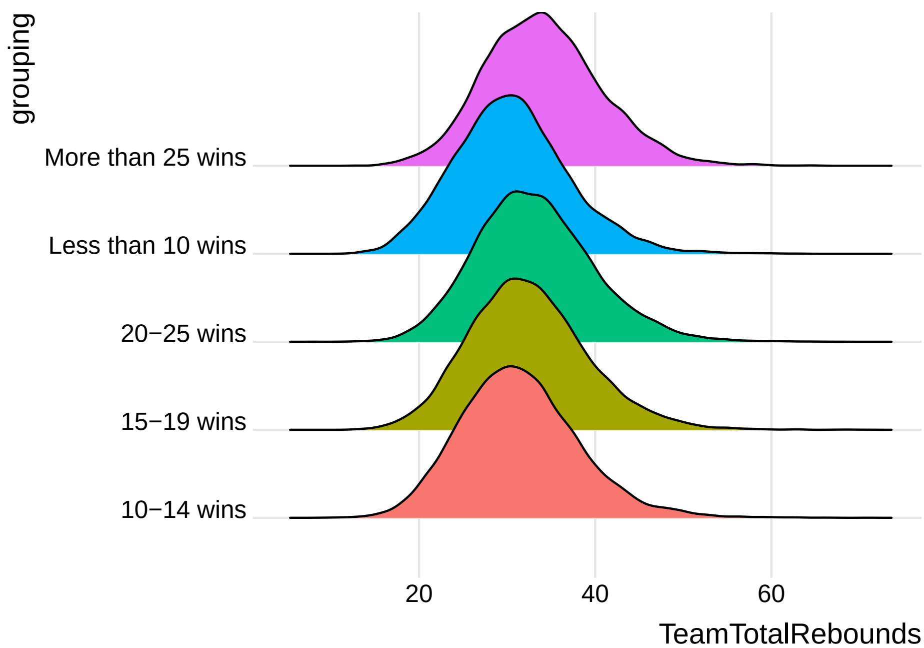
Let's look at the differences in rebounds by those five groups. Do teams that win more than 25 games rebound better than teams that win fewer games?

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).
```



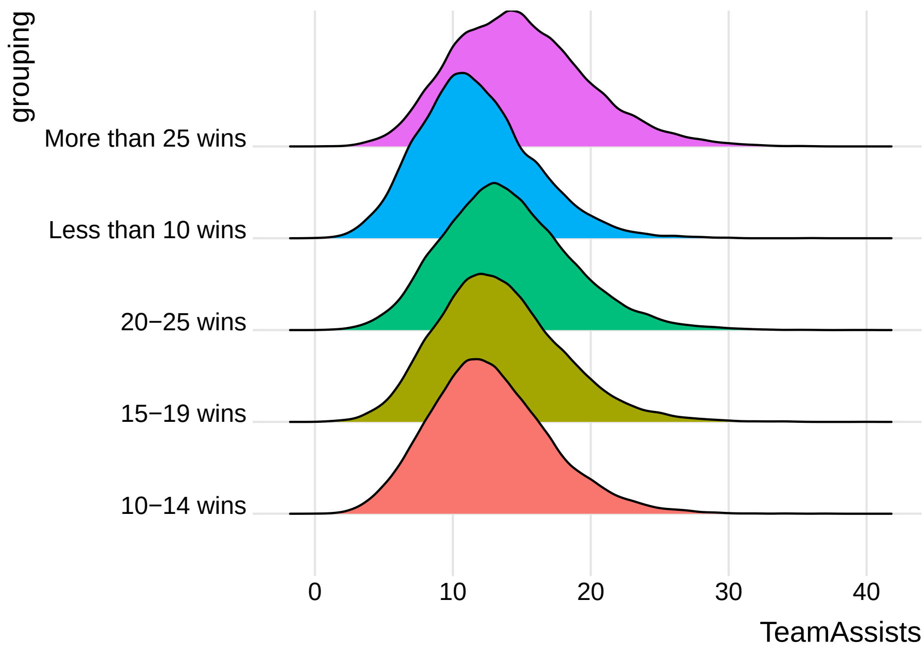
Answer? Not really. Game to game, maybe. Over five seasons? The differences are indistinguishable.

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).
```

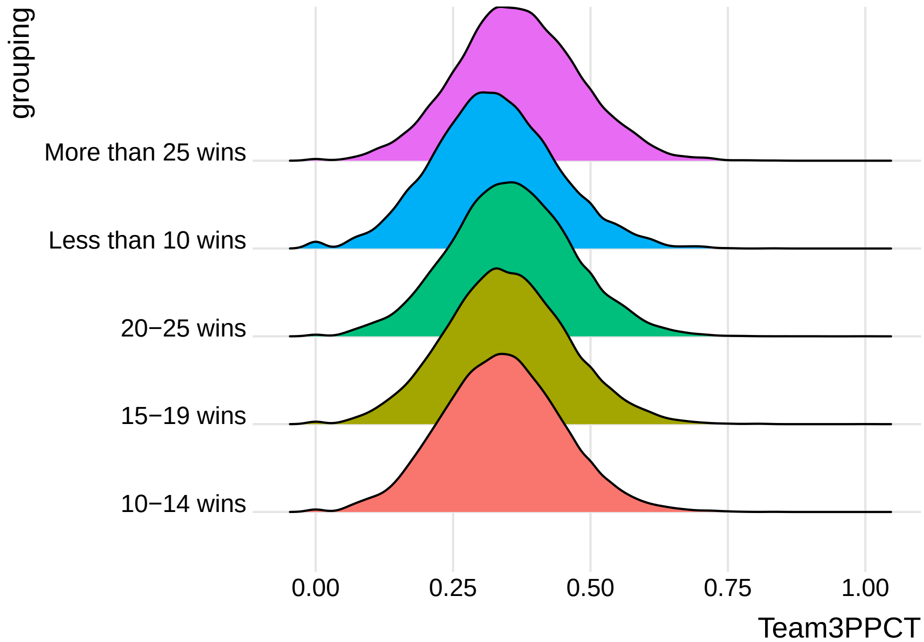


There's a little better, especially between top and bottom.

```
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).
```

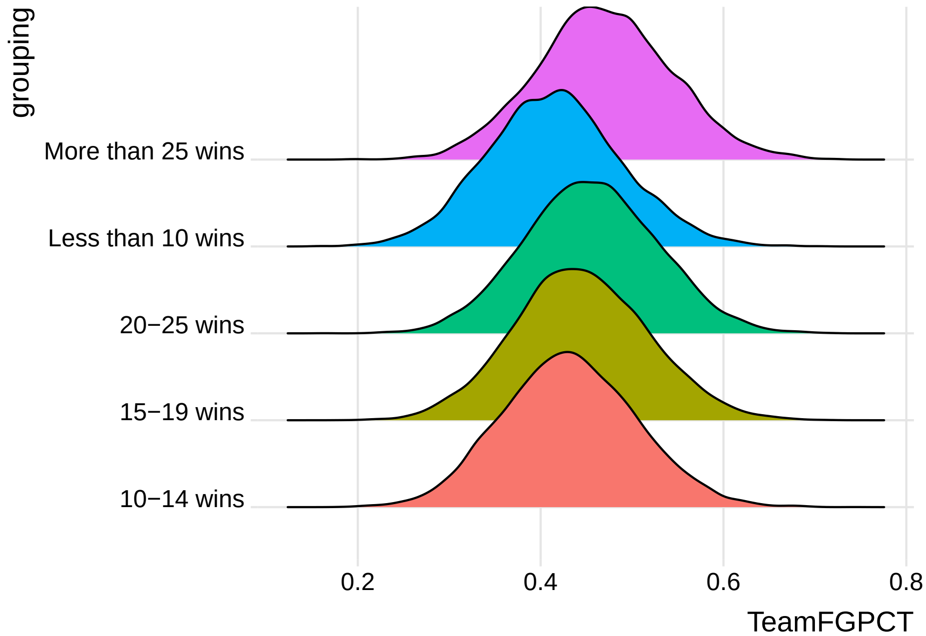


If you've been paying attention this semester, you know what's coming next.

```
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

Dumbbell and lollipop charts

Second to my love of waffle charts because I'm always hungry, dumbbell 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.

Lollipop charts – another excellent name – are a variation on bar charts. They do a good job of showing magnitude and difference between things.

To use both of them, you need to add a new library:

```
install.packages("ggalt")
```

Let's give it a whirl.

```
library(tidyverse)
library(ggalt)
```

```
## Registered S3 methods overwritten by 'ggalt':
##   method                  from
##   grid.draw.absoluteGrob  ggplot2
##   grobHeight.absoluteGrob ggplot2
##   grobWidth.absoluteGrob  ggplot2
##   grobX.absoluteGrob      ggplot2
##   grobY.absoluteGrob      ggplot2
```

19.1 Dumbbell plots

For this, let's use last season's college football data.

```
logs <- read_csv("data/footballlogs19.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   Result = col_character(),
##   TeamFull = col_character(),
##   TeamURL = col_character(),
##   Outcome = col_character(),
##   Team = col_character(),
##   Conference = col_character()
## )
## i Use `spec()` for the full column specifications.
```

For the first example, let's look at the difference between a team's giveaways – turnovers lost – versus takeaways, turnovers gained. To get this, we're going to add up all offensive turnovers and defensive turnovers for a team in a season and take a look at where they come out. To make this readable, I'm going to focus on the Big Ten.

```
turnovers <- logs %>%
  group_by(Team, Conference) %>%
  summarise(
    Giveaways = sum(TotalTurnovers),
    Takeaways = sum(DefTotalTurnovers)) %>%
  filter(Conference == "Big Ten Conference")
```

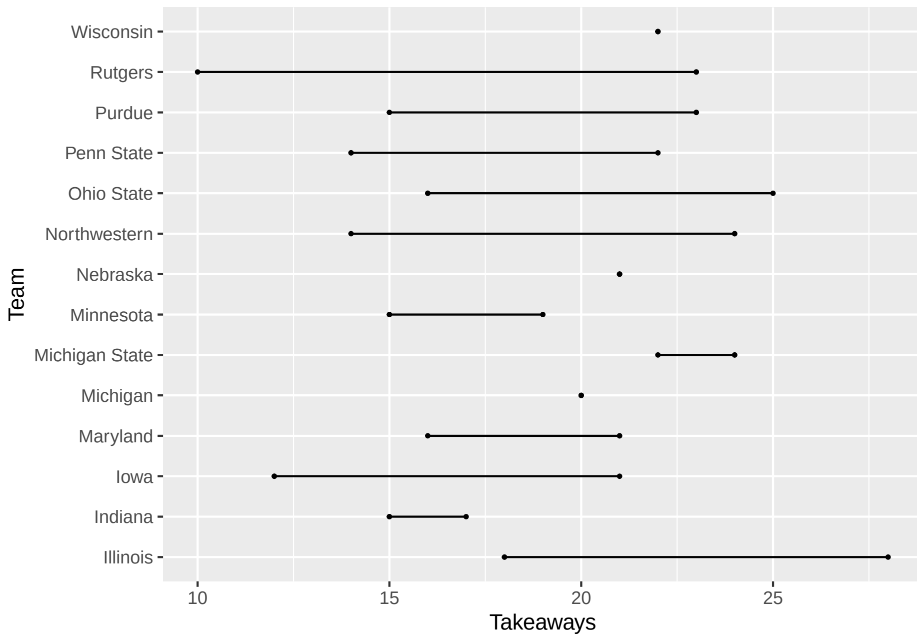
```
## `summarise()` regrouping output by 'Team' (override with `.groups` argument)
```

Now, the way that the `geom_dumbbell` works is pretty simple when viewed through what we've done before. There's just some tweaks.

First: We start with the y axis. The reason is we want our dumbbells going left and right, so the label is going to be on the y axis.

Second: Our x is actually two things: x and xend. What you put in there will decide where on the line the dot appears.

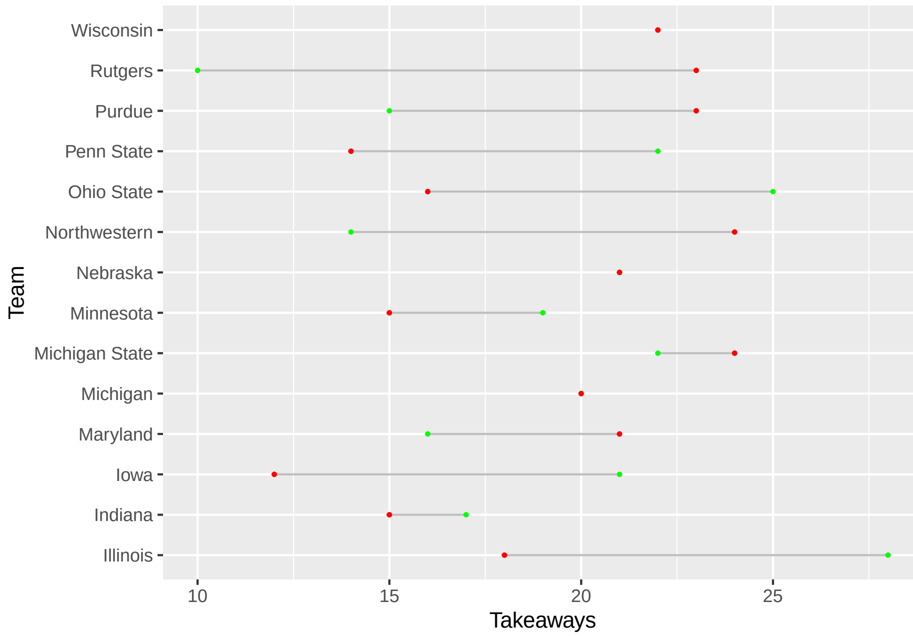
```
ggplot() +
  geom_dumbbell(
    data=turnovers,
    aes(y=Team, x=Takeaways, xend=Giveaways)
  )
```



Well, that's a chart alright, but what dot is the giveaways and what are the takeaways? To fix this, we'll add colors.

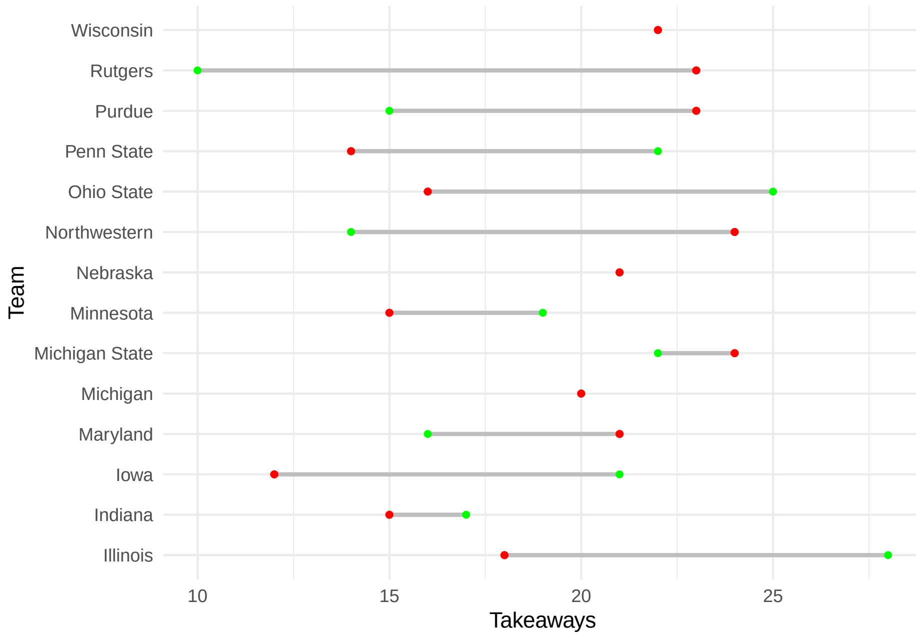
So our choice of colors here is important. We want giveaways to be seen as bad and takeaways to be seen as good. So lets try red for giveaways and green for takeaways. To make this work, we'll need to do three things: first, use the English spelling of color, so `colour`. The, uh, `colour` is the bar between the dots, the `x_colour` is the color of the x value dot and the `xend_colour` is the color of the xend dot. So in our setup, takeaways are x, they're good, so they're green.

```
ggplot() +
  geom_dumbbell(
    data=turnovers,
    aes(y=Team, x=Takeaways, xend=Giveaways),
    colour = "grey",
    colour_x = "green",
    colour_xend = "red")
```



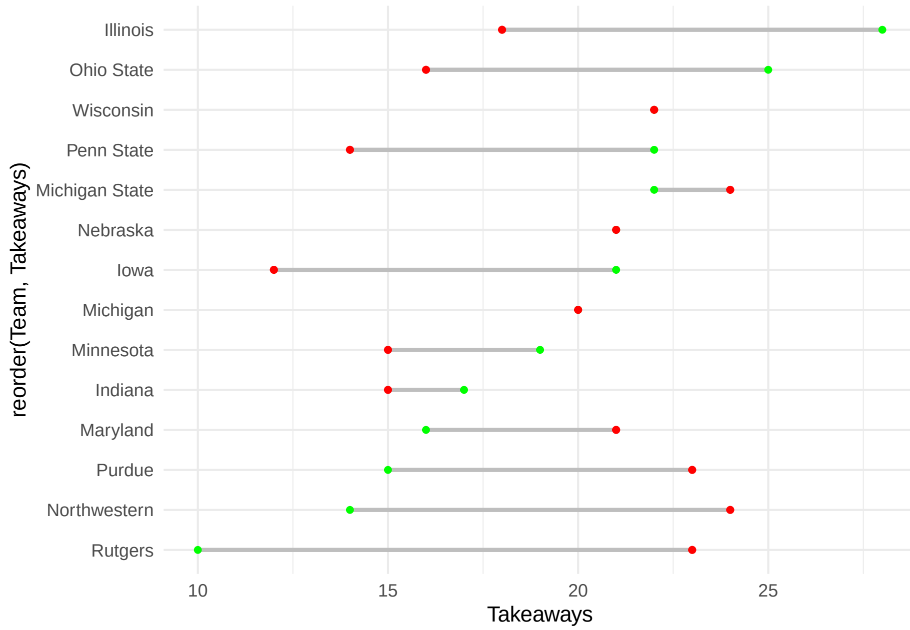
Better. Let's make two more tweaks. First, let's make the whole thing bigger with a `size` element. And let's add `theme_minimal` to clean out some cruft.

```
ggplot() +
  geom_dumbbell(
    data=turnovers,
    aes(y=Team, x=Takeaways, xend=Giveaways),
    size = 1,
    color = "grey",
    colour_x = "green",
    colour_xend = "red") +
  theme_minimal()
```



And now we have a chart that tells a story – got green on the right? That's good. A long distance between green and red? Better. But what if we sort it by good turnovers?

```
ggplot() +
  geom_dumbbell(
    data=turnovers,
    aes(y=reorder(Team, Takeaways), x=Takeaways, xend=Giveaways),
    size = 1,
    color = "grey",
    colour_x = "green",
    colour_xend = "red") +
  theme_minimal()
```

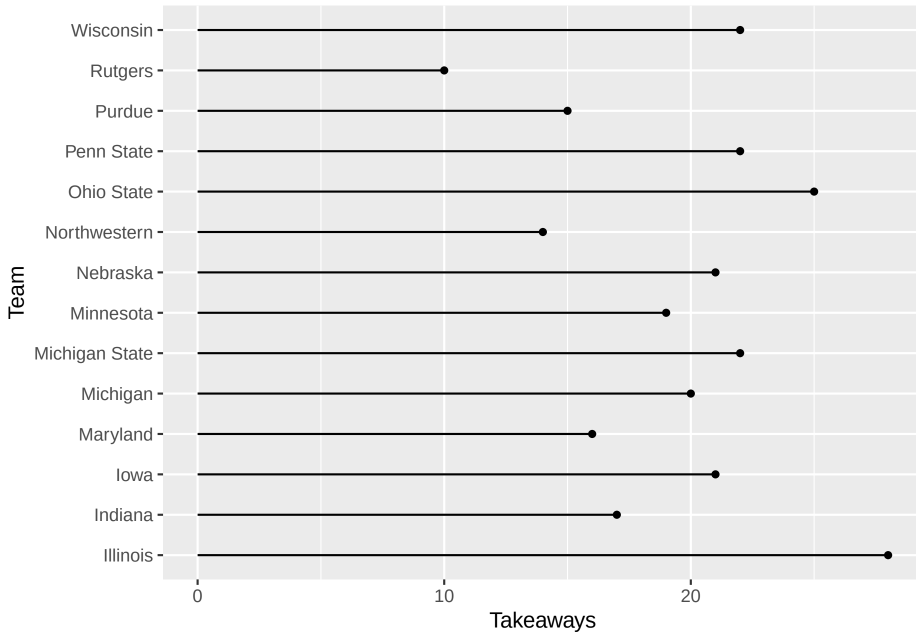


Believe it or not, Illinois had the most takeaways in the Big Ten last season. Don't sleep on the Fighting Lovie Smiths.

19.2 Lollipop charts

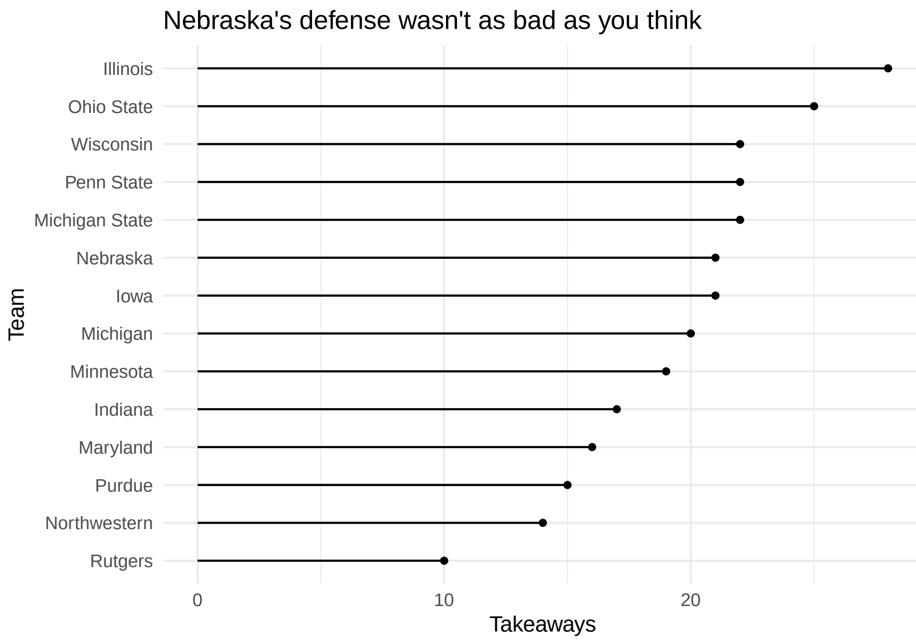
Sticking with takeaways, lollipops are similar to bar charts in that they show magnitude. And like dumbbells, they are similar in that we start with a y – the traditional lollipop chart is on its side – and we only need one x. The only additional thing we need to add is that we need to tell it that it is a horizontal chart.

```
ggplot() +
  geom_lollipop(
    data=turnovers,
    aes(y=Team, x=Takeaways),
    horizontal = TRUE
    )
```



We can do better than this with a simple theme_minimal and some better labels.

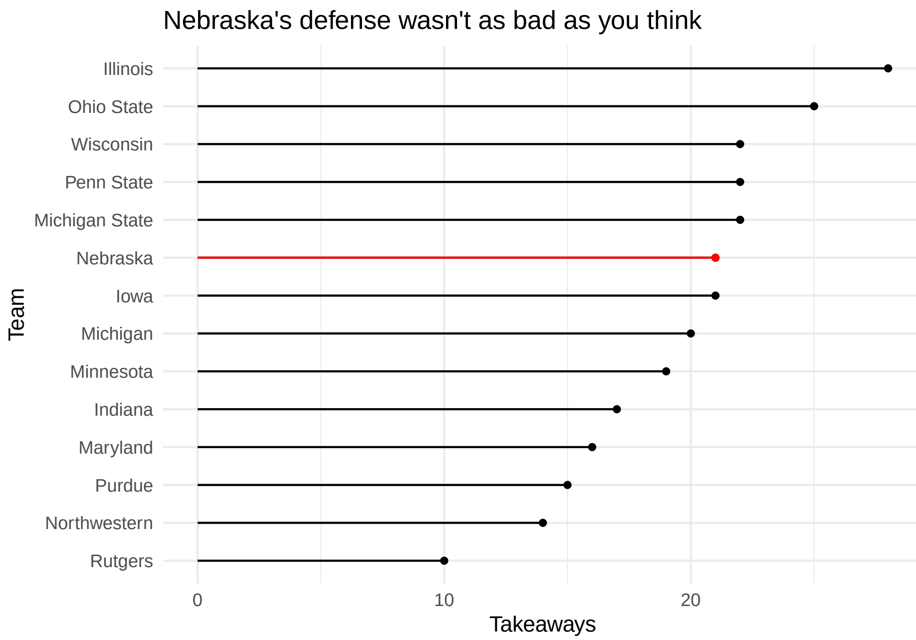
```
ggplot() +
  geom_lollipop(
    data=turnovers,
    aes(y=reorder(Team, Takeaways), x=Takeaways),
    horizontal = TRUE
    ) + theme_minimal() +
  labs(title = "Nebraska's defense wasn't as bad as you think", y="Team")
```



How about some layering?

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

```
ggplot() +
  geom_lollipop(
    data=turnovers,
    aes(y=reorder(Team, Takeaways), x=Takeaways),
    horizontal = TRUE
    ) +
  geom_lollipop(
    data=nu,
    aes(y=Team, x=Takeaways),
    horizontal = TRUE,
    color = "red"
  ) +
  theme_minimal() +
  labs(title = "Nebraska's defense wasn't as bad as you think", y="Team")
```



The headline says it all. Nebraska's defense was middle of the league in takeaways.

Chapter 20

Scatterplots

On the Monday, Sept. 21, 2020 edition of the Pick Six Podcast, Omaha World Herald reporter Sam McKewon talked a little about the Nebraska mens basketball team. Specifically the conversation was about a new roster release, and how the second year of Fred Hoiberg ball was going to look very different, starting with the heights of the players. After a near complete roster turnover, the players on the team now were nearly all taller than 6'4", and one of the shorter ones is penciled in as the starting point guard.

Why is that important? One reason, McKewon posited, is that teams made a lot of three point shots on Nebraska. In fact, Nebraska finished dead last in the conference in three points shots made against them. McKewon chalked this up to bad perimeter defense, and that Nebraska needed to improve it. Being taller – or more specifically having the longer arms that go with being taller – will help with that, McKewon said.

Better perimeter defense, better team.

The question before you is this: is that true? Does keeping a lid on your opponent's ability to score three pointers mean more wins?

This is what we're going to start to answer today. And we'll do it with scatterplots and regressions. Scatterplots are very good at showing **relationships between two numbers**.

First, we need libraries and data.

```
library(tidyverse)
```

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

```
##
## -- 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()
## )
## i Use `spec()` for the full column specifications.
```

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

In our case, we want to know how much do three point shots made influence wins? How much difference can we explain in wins by knowing how many threes the other team made against you? We're going to total up the number of threes each team allowed and their season wins in one swoop.

To do this, we need to use conditional logic – `case_when` in this case – to determine if the team won or lost the game. In this case, we'll create a new column called `winloss`. Case when statements can be read like this: When This is True, Do This. This bit of code – which you can use in a lot of contexts in this class – uses the `grepl` function to look for the letter W in the W_L column and, if it finds it, makes winloss 1. If it finds an L, it makes it 0. Sum your winloss column and you have your season win total. The reason we have to use `grepl` to find W or L is because Sports Reference will record overtime wins differently than regular wins. Same with losses.

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

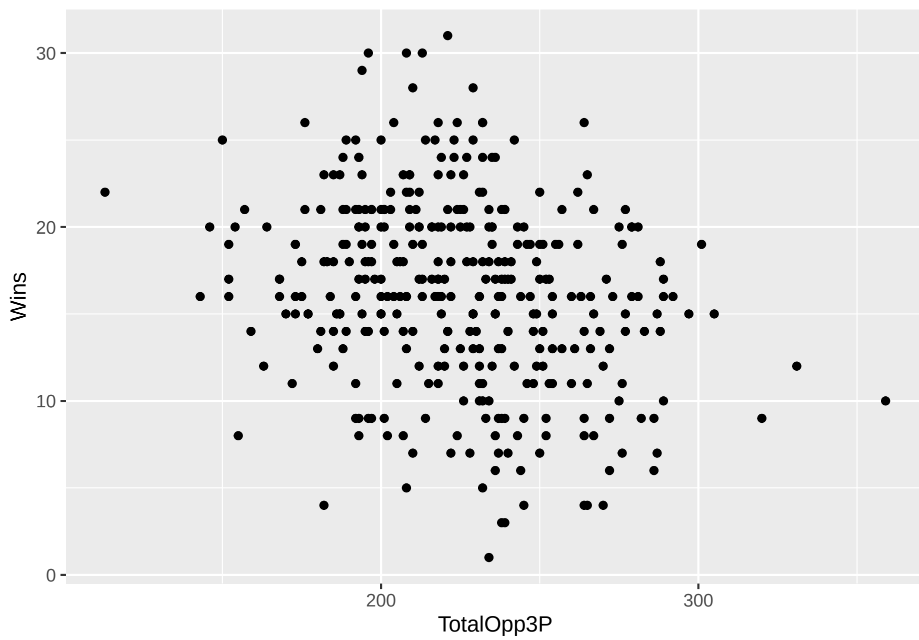
Now we can get a dataframe together that gives us the total wins for each team, and the total three point shots made. We'll call that new dataframe `threedef`.

```
threedef <- winlosslogs %>%
  group_by(Team) %>%
  summarise(
    Wins = sum(winloss),
    TotalOpp3P = sum(Opponent3P)
    )
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

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 three pointers given up, y is our wins.

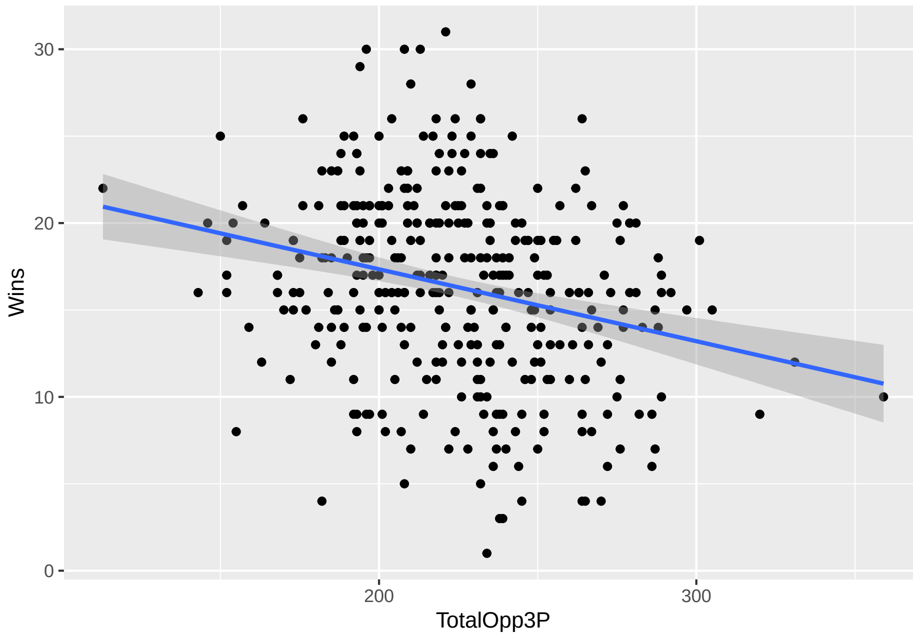
```
ggplot() + geom_point(data=threedef, aes(x=TotalOpp3P, y=Wins))
```



Let's talk about this. This seems kind of random, but clustered around the middle and maybe sloping down to the right. That would mean the more threes you give up, the less you win. And that makes intuitive sense. But can we get a better sense of this? Yes, by adding another geom – `geom_smooth`. It's identical to our `geom_point`, but we add a method to the end, which in this case we're using the linear method or `lm`.

```
ggplot() +
  geom_point(data=threedef, aes(x=TotalOpp3P, y=Wins)) +
  geom_smooth(data=threedef, aes(x=TotalOpp3P, y=Wins), method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



So it does slope down to the right like we expect, but this still doesn't look good to me. It's very spread out. Which is a clue that you should be asking a question here: how strong of a relationship is this? How much can threes given up 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 opponent threes". Then we just want the summary of that model.

```
fit <- lm(Wins ~ TotalOpp3P, data = threedef)
summary(fit)
```

```
##
## Call:
## lm(formula = Wins ~ TotalOpp3P, data = threedef)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.9345  -3.4593   0.3006   3.6036  14.5274
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.619712   1.859947  13.774  < 2e-16 ***
## TotalOpp3P  -0.041390   0.008184  -5.057 6.87e-07 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.313 on 351 degrees of freedom
## Multiple R-squared:  0.06792,    Adjusted R-squared:  0.06526
## F-statistic: 25.58 on 1 and 351 DF,  p-value: 6.869e-07
```

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 0.06526, which we can interpret as shooting percentage predicts about 6.5 percent of the variance in wins. Which sounds not great.

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 6.869e-07, which is to say 6.869 with 7 zeros in front of it, or .00000006869. 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 TeamOpp3P is -0.041390. What this model predicts, given that and the intercept of 25.619712, is this: Every team starts with about 26 wins. For every 100 three pointers the other team makes, you lose 4.139 games off that total. So if you give up 100 threes in a season, you'll be a 20 win team. Give up 200, you're a 17 win team, and so on. How am I doing that? Remember your algebra and y = mx + b. In this case, y is the wins, m is the coefficient, x is the number of threes given up and b is the intercept.

Let's use Nebraska as an example. They had 276 threes scored on them in the last season.

y = -0.041390 * 276 + 25.619712 or 14.19 wins.

How many wins did Nebraska have? 7.

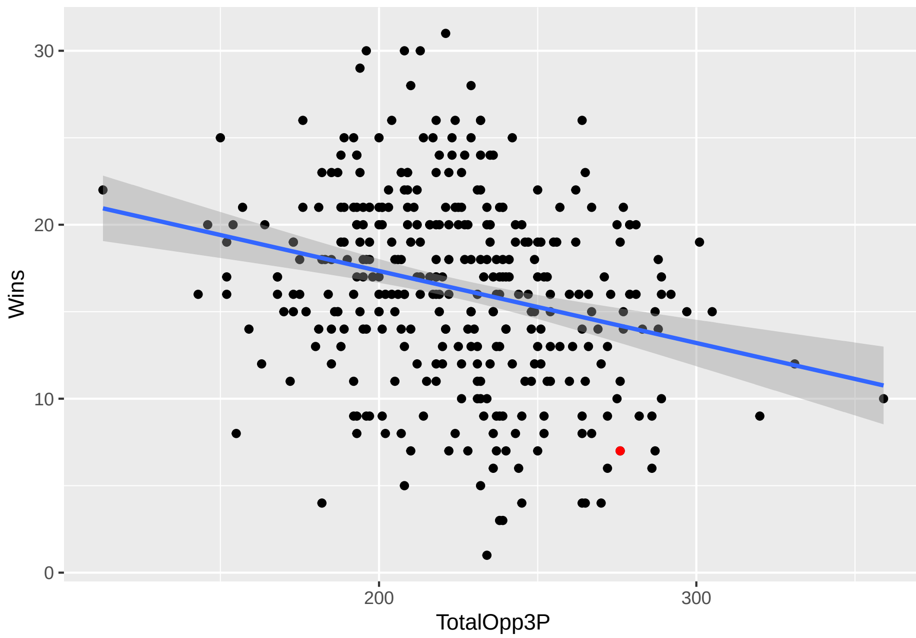
What does that mean? It means that as disappointing a season as it was, Nebraska UNDERPERFORMED according to this model. But our R-squared is only 6.5 percent. Put another way: 93.5 percent of the difference in wins between teams is predicted by something else.

Where is Nebraska on the plot? We know we can use layering for that.

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

```r
ggplot() +
  geom_point(data=threedef, aes(x=TotalOpp3P, y=Wins)) +
  geom_smooth(data=threedef, aes(x=TotalOpp3P, y=Wins), method="lm") +
  geom_point(data=nu, aes(x=TotalOpp3P, y=Wins), color="red")
```

```
## `geom_smooth()` using formula 'y ~ x'
```


30
20
10
0
Wins
200
300
TotalOpp3P

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/logs20.csv")
```

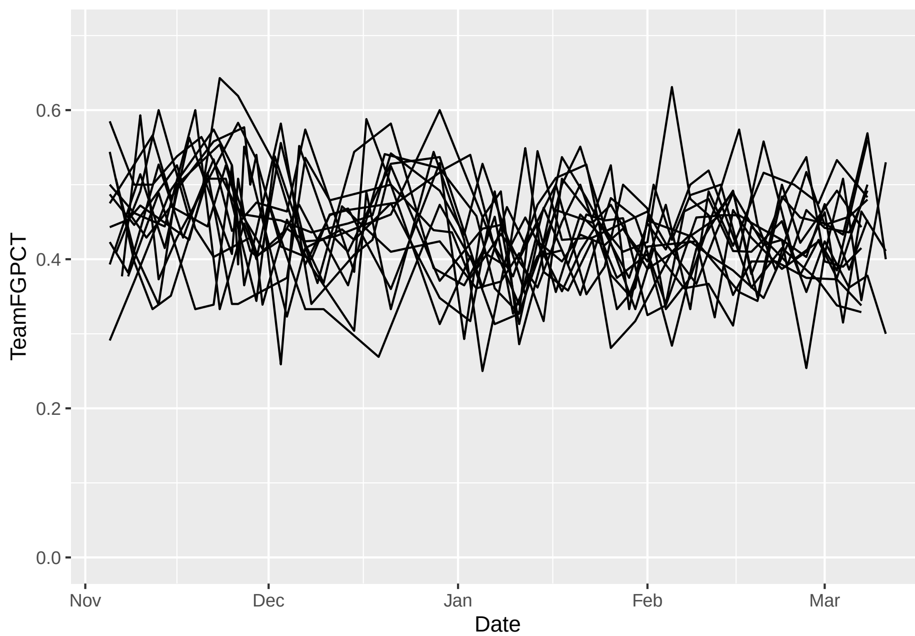
```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   Date = col_date(format = ""),
##   HomeAway = col_character(),
##   Opponent = col_character(),
##   W_L = col_character(),
##   Blank = col_logical(),
##   Team = col_character(),
##   Conference = col_character(),
##   season = col_character()
## )
## i Use `spec()` for the full column specifications.
```

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

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

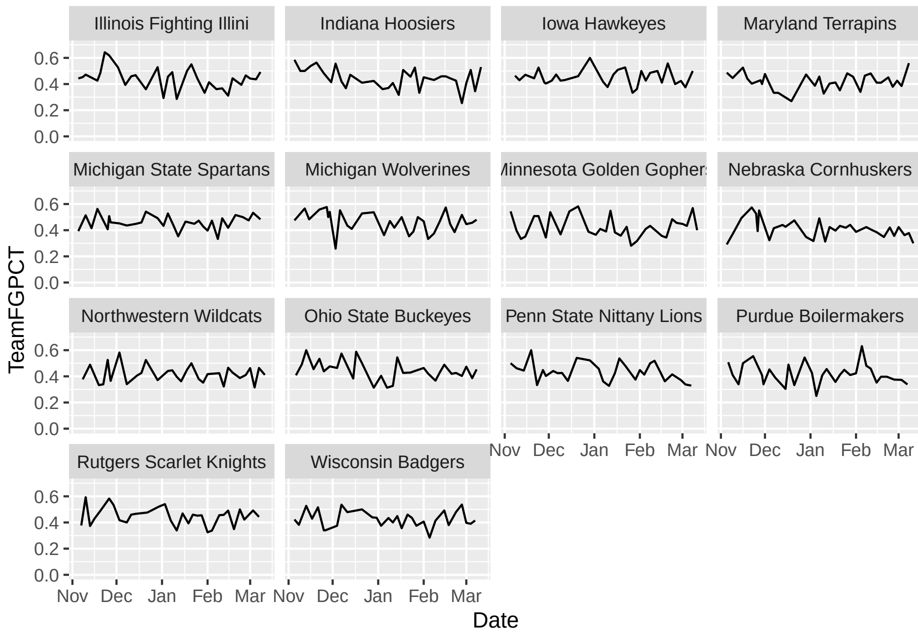
The first thing we can do is look at a line chart, like we have done in previous chapters.

```
ggplot() +
  geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) +
  scale_y_continuous(limits = c(0, .7))
```



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_continuous(limits = c(0, .7)) +
  facet_wrap(~Team)
```

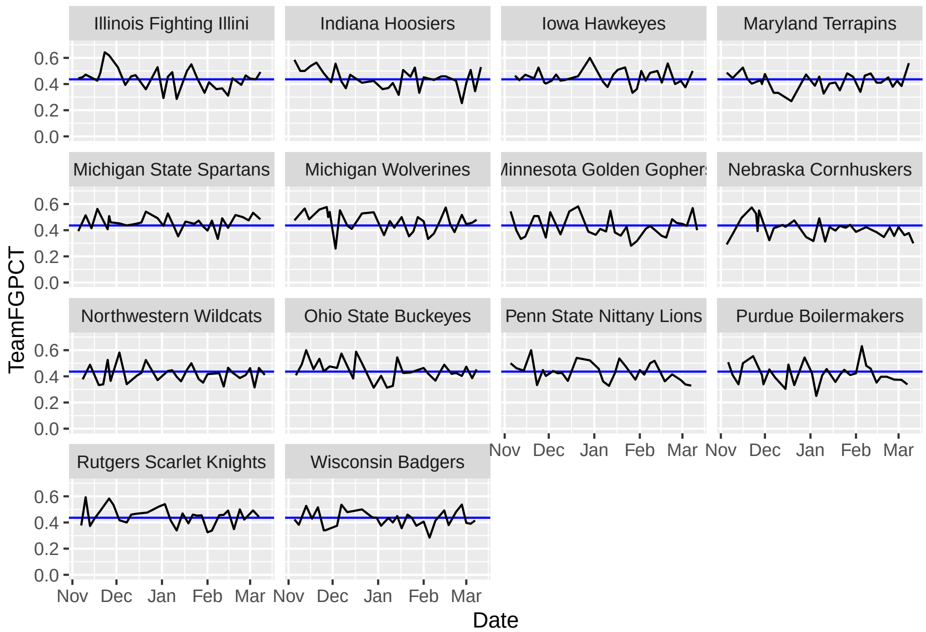


Answer: Not immediately clear, but we can look at this and analyze it. We could add a piece of annotation to help us out.

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

```
## # A tibble: 1 x 1
##   `mean(TeamFGPCT)`
##               <dbl>
## 1             0.436
```

```
ggplot() +
  geom_hline(yintercept=.4361078, color="blue") +
  geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) +
  scale_y_continuous(limits = c(0, .7)) +
  facet_wrap(~Team)
```

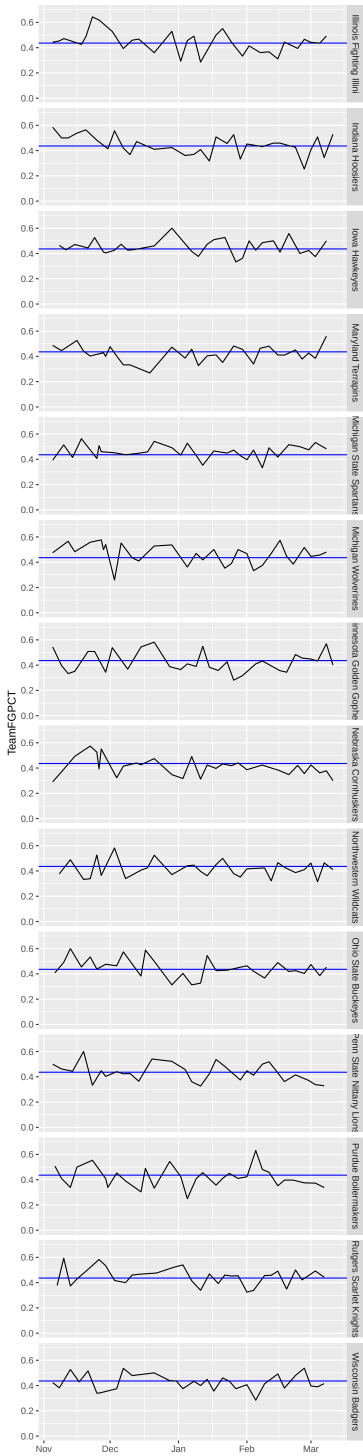


What do you see here? How do teams compare? How do they change over time? I'm not asking you these questions because they're an assignment – I'm asking because that's exactly what this chart helps answer. Your brain will immediately start making those connections.

21.1 Facet grid vs facet wraps

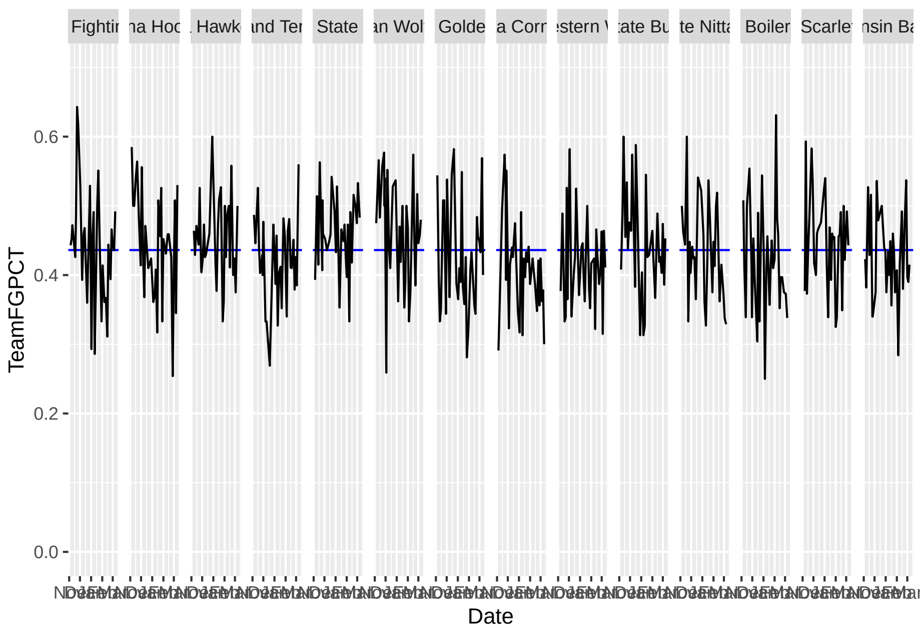
Facet grids allow us to put teams on the same plane, versus just repeating them. And we can specify that plane as vertical or horizontal. For example, here's our chart from above, but using facet_grid to stack them.

```
ggplot() +
  geom_hline(yintercept=.4361078, color="blue") +
  geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) +
  scale_y_continuous(limits = c(0, .7)) +
  facet_grid(Team ~ .)
```


Illinois Fighting Illini
Indiana Hoosiers
Iowa Hawkeyes
Maryland Terrapins
Michigan State Spartans
Michigan Wolverines
innesota Golden Gophe
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=.4361078, color="blue") +
  geom_line(data=big10, aes(x=Date, y=TeamFGPCT, group=Team)) +
  scale_y_continuous(limits = c(0, .7)) +
  facet_grid(. ~ Team)
```

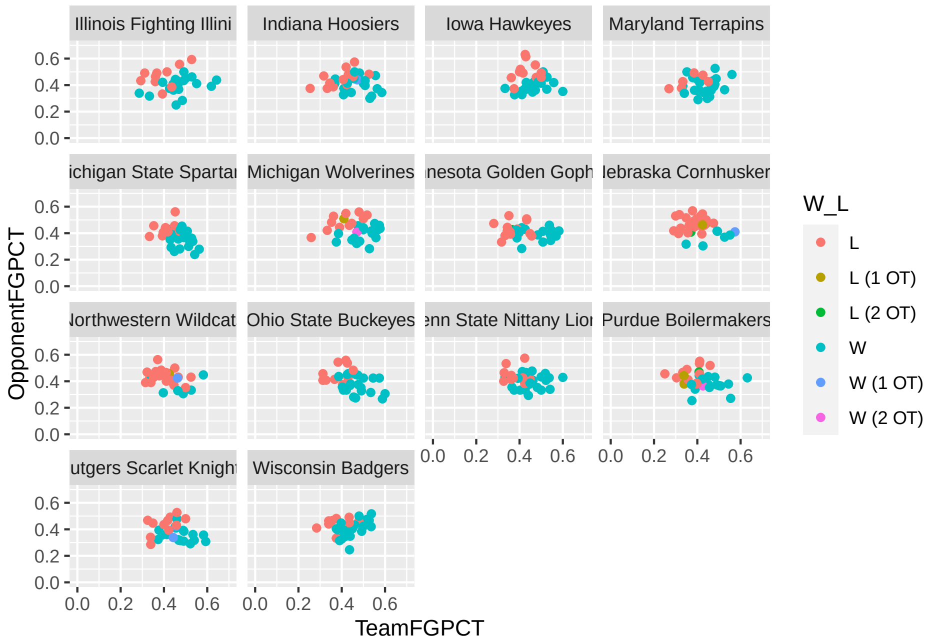


Note: We'd have some work to do with the labeling on this – we'll get to that – but you can see where this is valuable comparing a group of things. One warning: Don't go too crazy with this or it loses it's visual power.

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_y_continuous(limits = c(0, .7)) +
  scale_x_continuous(limits = c(0, .7)) +
  facet_wrap(~Team)
```


Illinois Fighting Illini
Indiana Hoosiers
Iowa Hawkeyes
Maryland Terrapins
ichigan State Spartai
Michigan Wolverines
nesota Golden Goph
lebraska Cornhusker
Northwestern Wildcat
Ohio State Buckeyes
enn State Nittany Lior
Purdue Boilermakers
utgers Scarlet Knigh
Wisconsin Badgers
0.0
0.2
0.4
0.6
OpponentFGPCT
TeamFGPCT
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 `kableExtra`.

For this assignment, we're going to need a bunch of new libraries. Go over to the console and run these:

```
install.packages("kableExtra")
install.packages("formattable")
install.packages("htmltools")
install.packages("webshot")
webshot::install_phantomjs()
```

So what does all of these libraries do? Let's gather a few and get some data of every game in the last 5 years.

```
library(tidyverse)
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```
logs <- read_csv("data/logs1520.csv")
```

```
##
## -- 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()
## )
## i Use `spec()` for the full column specifications.
```

Let's ask this question: Which college football team saw the greatest increase in three point attempts last season as a percentage of shots? The simplest way to calculate that is by percent change.

We've got a little work to do, putting together ideas we've used before. What we need to end up with is some data that looks like this:

`Team | 2018-2019 season threes | 2019-2020 season threes | pct change`

To get that, we'll need to do some filtering to get the right seasons, some grouping and summarizing to get the right number, some pivoting to get it organized correctly so we can mutate the percent change.

```
threechange <- logs %>%
  filter(season == "2018-2019" | season == "2019-2020") %>%
  group_by(Team, Conference, season) %>%
  summarise(Total3PA = sum(Team3PA)) %>%
  pivot_wider(names_from=season, values_from = Total3PA) %>%
  mutate(PercentChange = (`2019-2020`-`2018-2019`)/`2018-2019`) %>%
  arrange(desc(PercentChange)) %>%
  ungroup() %>%
  top_n(10) # just want a top 10 list
```

```
## `summarise()` regrouping output by 'Team', 'Conference' (override with `.groups` arg
```

```
## Selecting by PercentChange
```

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

```
threechange %>% kable()
```

Team	Conference	2018-2019	2019-2020	PercentChange
Mississippi Valley State Delta Devils	SWAC	554	837	0.5108303
Valparaiso Crusaders	MVC	585	843	0.4410256
Ball State Cardinals	MAC	621	842	0.3558776
San Jose State Spartans	MWC	641	861	0.3432137
Alabama Crimson Tide	SEC	718	957	0.3328691
Minnesota Golden Gophers	Big Ten	603	762	0.2636816
Georgia Southern Eagles	Sun Belt	631	792	0.2551506
Tennessee Tech Golden Eagles	OVC	620	776	0.2516129
San Francisco Dons	WCC	728	899	0.2348901
McNeese State Cowboys	Southland	547	675	0.2340037

So there you have it. Mississippi Valley State changed their team so much they took 51 percent more threes last season from the season before. Where did Nebraska come out? Isn't Fred Ball supposed to be a lot of threes? We ranked 111th in college basketball in terms of change from the season before. Believe it or not, Nebraska took four fewer threes last season under Fred Ball than the last season of Tim Miles.

Kable has a mountain of customization options. The good news is that it works in a very familiar pattern. We'll start with default styling.

```
threechange %>%
  kable() %>%
  kable_styling()
```

Team	Conference	2018-2019	2019-2020	PercentChange
Mississippi Valley State Delta Devils	SWAC	554	837	0.5108303
Valparaiso Crusaders	MVC	585	843	0.4410256
Ball State Cardinals	MAC	621	842	0.3558776
San Jose State Spartans	MWC	641	861	0.3432137
Alabama Crimson Tide	SEC	718	957	0.3328691
Minnesota Golden Gophers	Big Ten	603	762	0.2636816
Georgia Southern Eagles	Sun Belt	631	792	0.2551506
Tennessee Tech Golden Eagles	OVC	620	776	0.2516129
San Francisco Dons	WCC	728	899	0.2348901
McNeese State Cowboys	Southland	547	675	0.2340037

Let's do more than the defaults, which you can see are pretty decent. Let's stripe every other row with a little bit of grey, and let's smush the width of the rows.

```
threechange %>%
  kable() %>%
```

```
  kable_styling(bootstrap_options = c("striped", "condensed"))
```

Team	Conference	2018-2019	2019-2020	PercentChange
Mississippi Valley State Delta Devils	SWAC	554	837	0.5108303
Valparaiso Crusaders	MVC	585	843	0.4410256
Ball State Cardinals	MAC	621	842	0.3558776
San Jose State Spartans	MWC	641	861	0.3432137
Alabama Crimson Tide	SEC	718	957	0.3328691
Minnesota Golden Gophers	Big Ten	603	762	0.2636816
Georgia Southern Eagles	Sun Belt	631	792	0.2551506
Tennessee Tech Golden Eagles	OVC	620	776	0.2516129
San Francisco Dons	WCC	728	899	0.2348901
McNeese State Cowboys	Southland	547	675	0.2340037

Throughout the semester, we've been using color and other signals to highlight things. Let's pretend we're doing a project on Minnesota. We can use row_spec to highlight things.

What row_spec is doing here is we're specifying which row – 6 – and making all the text on that row bold. We're making the color of the text white, because we're going to set the background to a color – in this case, the hex color for Minnesota gold.

```
threechange %>%
  kable() %>%
  kable_styling(bootstrap_options = c("striped", "condensed")) %>%
  row_spec(6, bold = T, color = "white", background = "#FBB93C")
```

Team	Conference	2018-2019	2019-2020	PercentChange
Mississippi Valley State Delta Devils	SWAC	554	837	0.5108303
Valparaiso Crusaders	MVC	585	843	0.4410256
Ball State Cardinals	MAC	621	842	0.3558776
San Jose State Spartans	MWC	641	861	0.3432137
Alabama Crimson Tide	SEC	718	957	0.3328691
Minnesota Golden Gophers	**Big Ten**	**603**	**762**	**0.2636816**
Georgia Southern Eagles	Sun Belt	631	792	0.2551506
Tennessee Tech Golden Eagles	OVC	620	776	0.2516129
San Francisco Dons	WCC	728	899	0.2348901
McNeese State Cowboys	Southland	547	675	0.2340037

There's also something called column_spec where we can change the styling on individual columns. What if we wanted to make all the team names bold?