Answering questions with data: Lab Manual

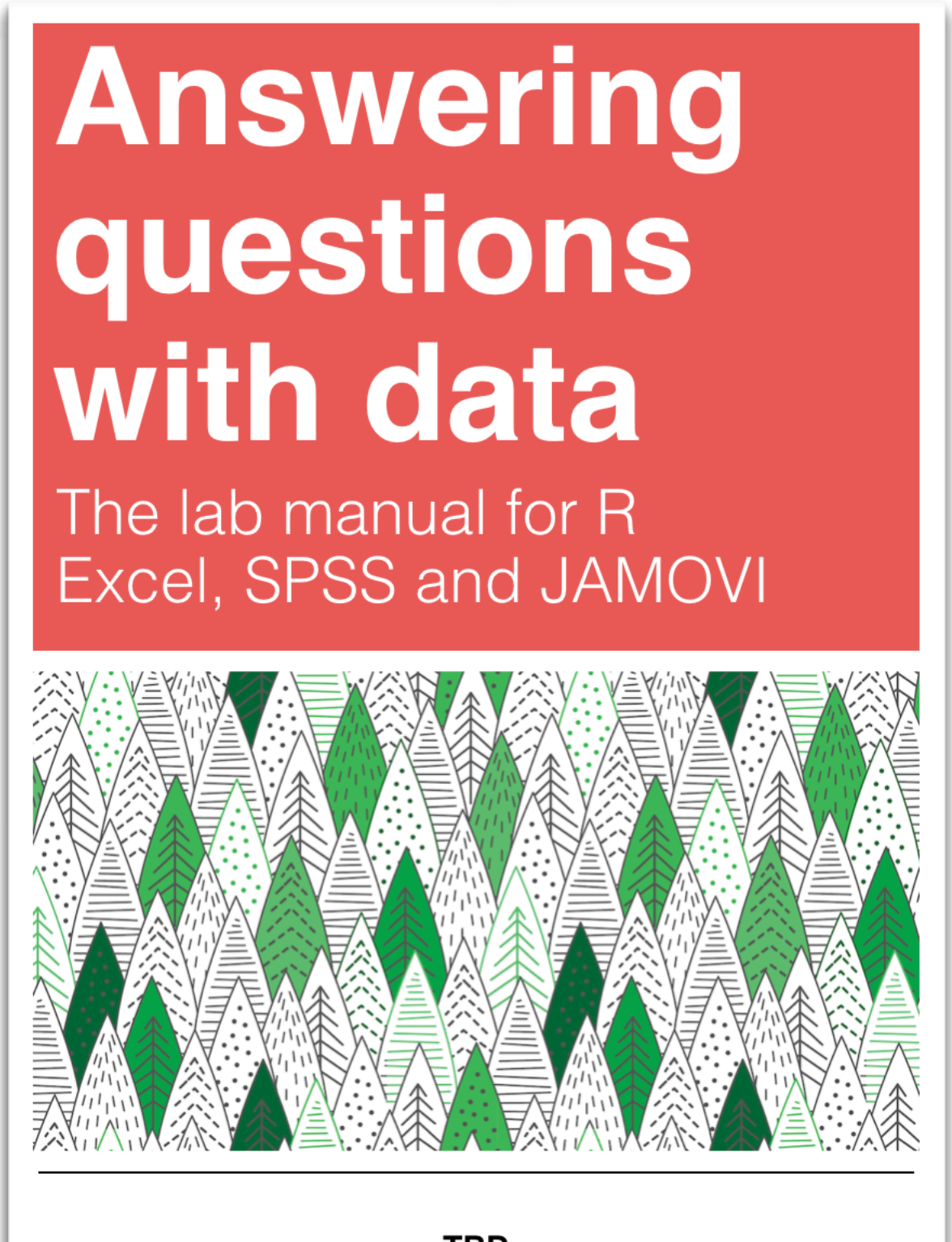
Matthew J. C. Crump
Anajali Krishnan
Stephen Volz
Alla Chavarga

Currently in draft, all attributions and licensed to adapted content will be recognized by the time we are finished.

2018: Last Compiled 2018-07-26

Contents

Preface


Answering questions with data
The lab manual for R Excel, SPSS and JAMOVI

This lab manual involves tutorials and data-analysis problems using the free statistics software R, as well as Excel, and SPSS. The goal is to train students to be able to organize and analyze data common to research in psychology, as well as to understand the ideas behind the analyses so students can take creative approaches to answering questions with data.

0.1 R

R is primarily a computer programming language for statistical analysis. It is *free*, and *open-source* (many people contribute to developing it), and runs on most operating systems. It is a powerful language that can be used for all sorts of mathematical operations, data-processing, analysis, and graphical display of data. I even used R to write this lab manual. And, I use R all the time for my own research, because it makes data-analyis fast, efficient, transparent, reproducible, and exciting.

Statistics Software

- SPSS
- SAS
- JMP
- R
- Julia
- Matlab

0.2 Why R?

There are lots of different options for using computers to analyze data, why use R?. The options all have pros and cons, and can be used in different ways to solve a range of different problems. Some software allows you to load in data, and then analyze the data by clicking different options in a menu. This can sometimes be fast and convenient. For example, once the data is loaded, all you have to do is click a couple buttons to analyse the data! However, many aspects of data-analysis are not so easy. For example, usually particular analyses require that the data be formatted in a particular way so that the program analyze it properly. Often times when a researcher wants to ask a new question of an existing data set, they have to spend time re-formatting the data. If the data is large, then reformattin by hand is very slow, and can lead to errors. Another option, is to use a scripting language to instruct the computer how reformat the data. This is very fast and efficient. R provides the ability to everything all in one place. You can load in data, reformat it any way you like, then anlayze it anyway you like, and create beautiful graphs and tables (publication quality) to display your findings. Once you get the hang of R, it becomes very fast and efficient.

0.3 Installing R and R Studio

Download and install R onto your computer. The R website is: http://www.r-project.org

Find the download R using the link. This will take you to a page with many different mirror links. You can click any of these links to download a version of R that will work on your computer. After you have installed R you can continue.

After you have installed R on your computer, you should want to install another program called R studio. This program provides a user-friendly interface for using R. You must already have installed R before you perform this step. The R-studio website is: http://www.rstudio.com

Find the download link on the front-page, and then download R studio desktop version for your computer. After you have installed R studio you will be ready to start using R.

The website R-fiddle allows you to run R scripts in the cloud, so you can practice R from your web-browser!

0.4 R studio notes and tips

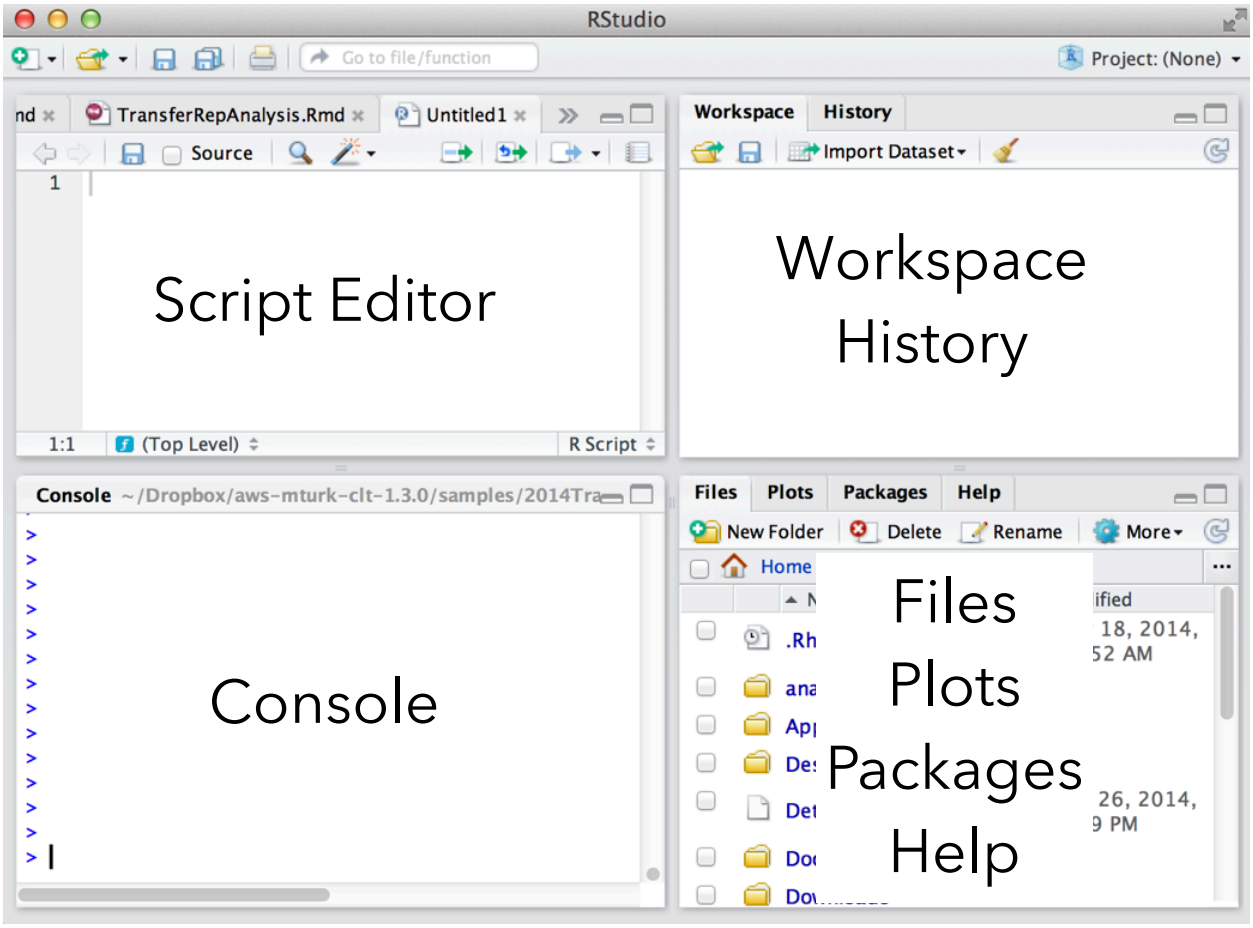


Figure 1: The R-studio workspace

0.4.1 Console

When you open up R studio you will see three or four main windows (the placement of each are configurable). In the above example, the bottom left window is the command line (terminal or console) for R. This is used to directly enter commands into R. Once you have entered a command here, press enter to execute the command. The console is useful for entering single lines of code and running them. Oftentimes this occurs when you are learning how to correctly execute a line of code in R. Your first few attempts may be incorrect resulting in errors, but trying out different variations on your code in the command line can help you produce the correct code. Pressing the up arrow while in the console will scroll through the most recently executed lines of code.

0.4.2 Script Editor

The top left corner contains the script editor. This is a simple text editor for writing and saving R scripts with many lines. Several tabs can be opened at once, with each tab representing a different R script. R scripts can be saved from the editor (resulting in a .r file). Whole scripts can be run by copy and pasting them into the console and pressing enter. Alternatively, you can highlight portions of the script that you want to run (in the script editor) and press command-enter to automatically run that portion in the console (or press the button for running the current line/section: green arrow pointing right).

0.4.3 Workspace and History

The top right panel contains two tabs, one for the workspace and another for history. The workspace lists out all of the variables and functions that are currently loaded in R's memory. You can inspect each of the variables by clicking on them. This is generally only useful for variables that do not contain large amounts of information. The history tab provides a record of the recent commands executed in the console.

0.4.4 File, Plot, Packages, Help

The bottom-right window has four tabs for files, plots, packages, and help. The files tab allows browsing of the computers file directory. An important concept in R is the **current working directory**. This is file folder that R points to by default. Many functions in R will save things directly to this direct, or attempt to read files from this directory. The current working directory can be changed by navigating to the desired folder in the file menu, and then clicking on the more option to set that folder to the current working directory. This is especially important when reading in data to R. The current working directory should be set to the folder containing the data to be inputted into R. The plots tab will show recent plots and figures made in R. The packages tab lists the current R libraries loaded into memory, and provides the ability to download and enable new R packages. The help menu is an invaluable tool. Here, you can search for individual R commands to see examples of how they are used. Sometimes the help files for individual commands are opaque and difficult to understand, so it is necessary to do a google search to find better examples of using these commands.

0.5 Final comments

In this course we will be using R as a tool to analyze data, and as a tool to help us gain a better understanding of what our analyses are doing. Throughout each lab we will show you how to use R to solve specific problems, and then you will use the examples to solve homework and lab assignments. R is a very deep programming language, and in many ways we will only be skimming the surface of what R can do. Along the way, there will be many pointers to more advanced techniques that interested students can follow to become experts in using R for data-analysis, and computer programming in general.

Chapter 1

Lab 1: Graphing Data

The commonality between science and art is in trying to see profoundly - to develop strategies of seeing and showing. —Edward Tufte

As we have found out from the textbook and lecture, when we measure things, we get lots of numbers. Too many. Sometimes so many your head explodes just thinking about them. One of **the most helpful things** you can do to begin to make sense of these numbers, is to look at them in graphical form. Unfortunately, for sight-impaired individuals, graphical summary of data is much more well-developed than other forms of summarizing data for our human senses. Some researchers are developing auditory versions of visual graphs, a process called **sonification**, but we aren't prepared to demonstrate that here. Instead, we will make charts, and plots, and things to look at, rather than the numbers themselves, mainly because these are tools that are easiest to get our hands on, they are the most developed, and they work really well for visual summary. If time permits, at some point I would like to come back here and do the same things with sonification. I think that would be really, really cool!

1.1 General Goals

Our general goals for this first lab are to get your feet wet, so to speak. We'll do these things:

1. Load in some data to a statistical software program
2. Talk a little bit about how the data is structured
3. Make graphs of the data so we can look at it and make sense of it.

1.2 R

Your lab instructor will show you how to open R-studio on the lab computer. Just find it and double-click. Now you have R-studio.

There are numerous resources for learning about R, we put some of them on the course website, under the resouces page. You will find these resources helpful as you learn. We also have a kind of general introduction to R and Rstudio here. This shows you how to download R and R-studio at home (it's free).

When we made this course, we assumed that most students would be unfamiliar with R and R-studio, and might even be frightened of it, because it is a computer programming language (OOOOHHH NOOOOOOO, I NEED TO DROP THIS COURSE NOW)…Don't worry. It's going to be way easier than you think. Let's compare to other statistics courses where you would learn something like SPSS. That is also a limited programming language, but you would mostly learn how to point with a mouse, and click with button. I bet

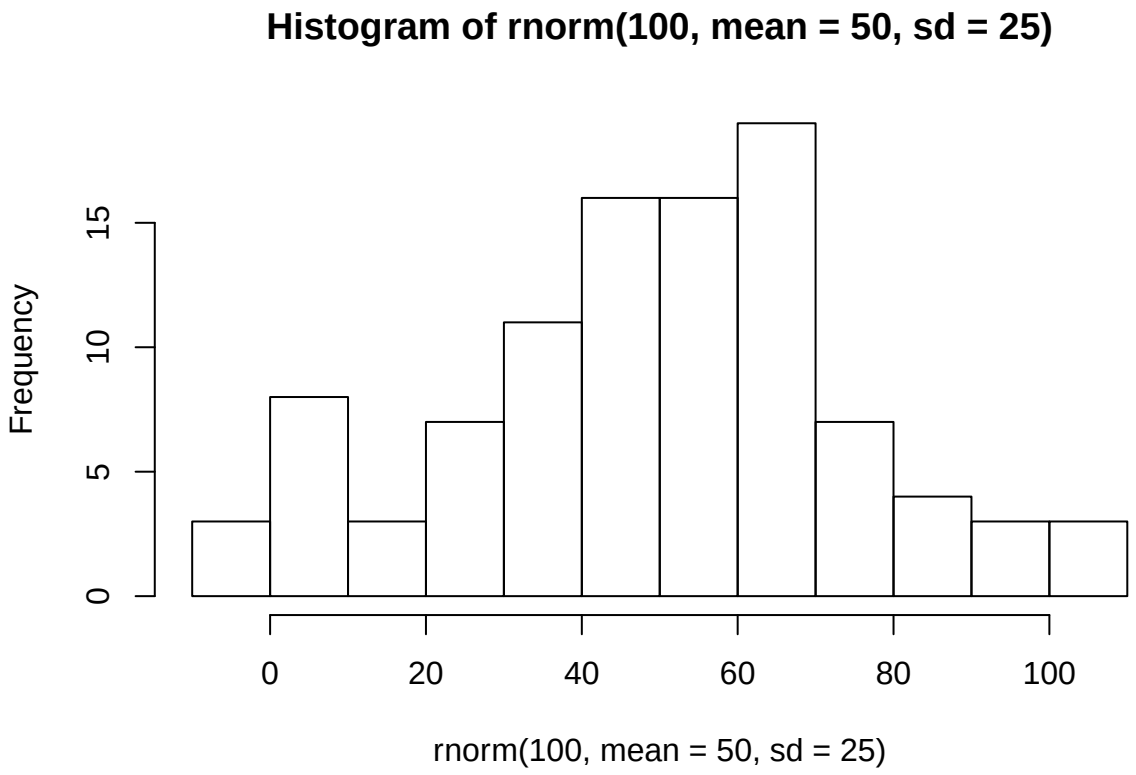
you already know how to do that. I bet you also already know how to copy and paste text, and press enter. That's mostly what we'll be doing to learn R. We will be doing statistics by typing commands, rather than by clicking buttons. However, lucky for you, all of the commands are already written for you. You just have to copy/paste them.

We know that this will seem challenging at first. But, we think that with lots of working examples, you will get the hang of it, and by the end of the course you will be able to do things you might never have dreamed you can do. It's really a fantastic skill to learn, even if you aren't planning on going on to do research in Psychology (in which case, this kind of thing is necessary skill to learn). With that, let's begin.

1.2.1 Get some data

In order to graph data, we need to have some data first…Actually, with R, that's not quite true. Run this bit of code and see what happens:

```
hist(rnorm(100, mean=50, sd=25))
```



You just made R sample 100 numbers, and then plot the results in a histogram. Pretty neat. We'll be doing some of this later in the course, where get R to make fake data for us, and then we learn to think about how data behaves under different kinds of assumptions.

For now, let's do something that might be a little bit more interesting…what movies are going to be filming in NYC? It turns out that NYC makes a lot of data about a lot things open and free for anyone to download and look at. This is the NYC Open Data website: https://opendata.cityofnewyork.us. I searched through the data, and found this data file that lists the locations of film permits for shooting movies all throughout the burroughs. You can download the data to your computer from this link

NOTE TO SELF, COME BACK HERE AND WALKTHROUGH FILE PATHS AND THINGS

```
library(data.table)
nyc_films <-fread("data/Film_Permits.csv")
```

1.2.2 Look at the data

You will be downloading and analyzing all kinds of data files this semester. We will follow the very same steps every time. The steps are to load the data, then look at it. You want to see what you've got.

In R-studio, you will now see a variable called `nyc_films` in the top right-hand corner of the screen, in the environment tab. If you click this thing, it will show you the contents of the data in a new window. The data is stored in something we call a `data frame`. It's R lingo, for the thing that contains the data. Notice is a square, with rows going across, and columns going up and down. It looks kind of like an excel spreadsheet if you are familiar with Excel.

It's useful to know you can look at the data frame this way if you need to. But, this dataframe is really big, it has 50,728 rows of data. That's a lot too much to look at.

1.2.2.1 summarytools

The summarytools packages give a quick way to summarize all of the data in a data frame. Here's how. When you run this code you will see the summary in the viewer on the bottom right hand side. There's a little browser button (arrow on top of little window) that you can click to expand and see the whole thing in a browser.

```
library(summarytools)
view(dfSummary(nyc_films))
```

That is super helpful, but it's still a lot to look at. Because there is so much data here, it's pretty much mind-boggling to start thinking about what to do with it.

1.2.3 Make Plots to answer questions

Let's walk through a couple questions we might have about this data. We can see that there were 50,728 film permits made. We can also see that there are different columns telling us information about each of the film permits. For example, the `Borough` column lists the Borough for each request, whether it was made for: Manhattan, Brooklyn, Bronx, Queen's, or Staten Island. Now we can ask our first question, and learn how to do some plotting in R.

1.2.3.1 Where are the most film permits being requested?

Do you have any guesses? Is it Manhattan, or Brookyln, of the Bronx? Or Queen's or Staten Island? We can find out by plotting the data using a bar plot. We just need to count how many film permits are made in each burough, and then make different bars represent the the counts.

First, we do the counting in R. Run the following code.

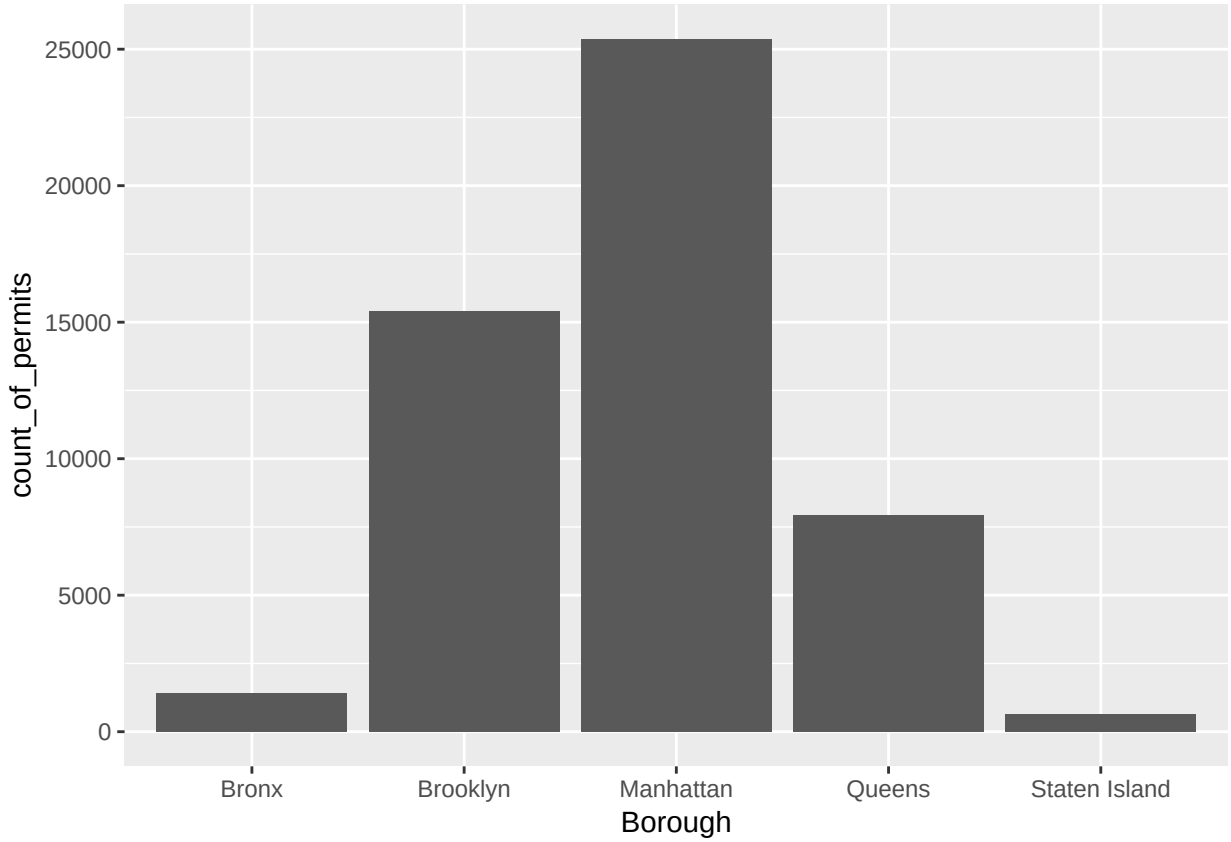
```
library(dplyr)

counts <- nyc_films %>%
          group_by(Borough) %>%
          summarize(count_of_permits = length(Borough))
```

The above grouped the data by each of the five Borough's, and then counted the number of times each Borough occured (using the `length` function). The result is a new variable called `count`. I chose to name this variable `count`. You can see that it is now displayed in the top-right hand corned in the environment tab. If you gave `count` a different name, like `muppets`, then it would be named what you called it.

If you click on the `counts` variable, you will see the five boroughs listed, along with the counts for how many film permits were requested in each Borough. These are the numbers that we want to plot in a graph.

We do the plot using a fantastic package called `ggplot2`. It is very powerful once you get the hand of it, and when you do, you will be able to make all sorts of interesting graphs. Here's the code to make the plot

```
library(ggplot2)

ggplot(counts, aes(x = Borough, y = count_of_permits )) +
  geom_bar(stat="identity")
```



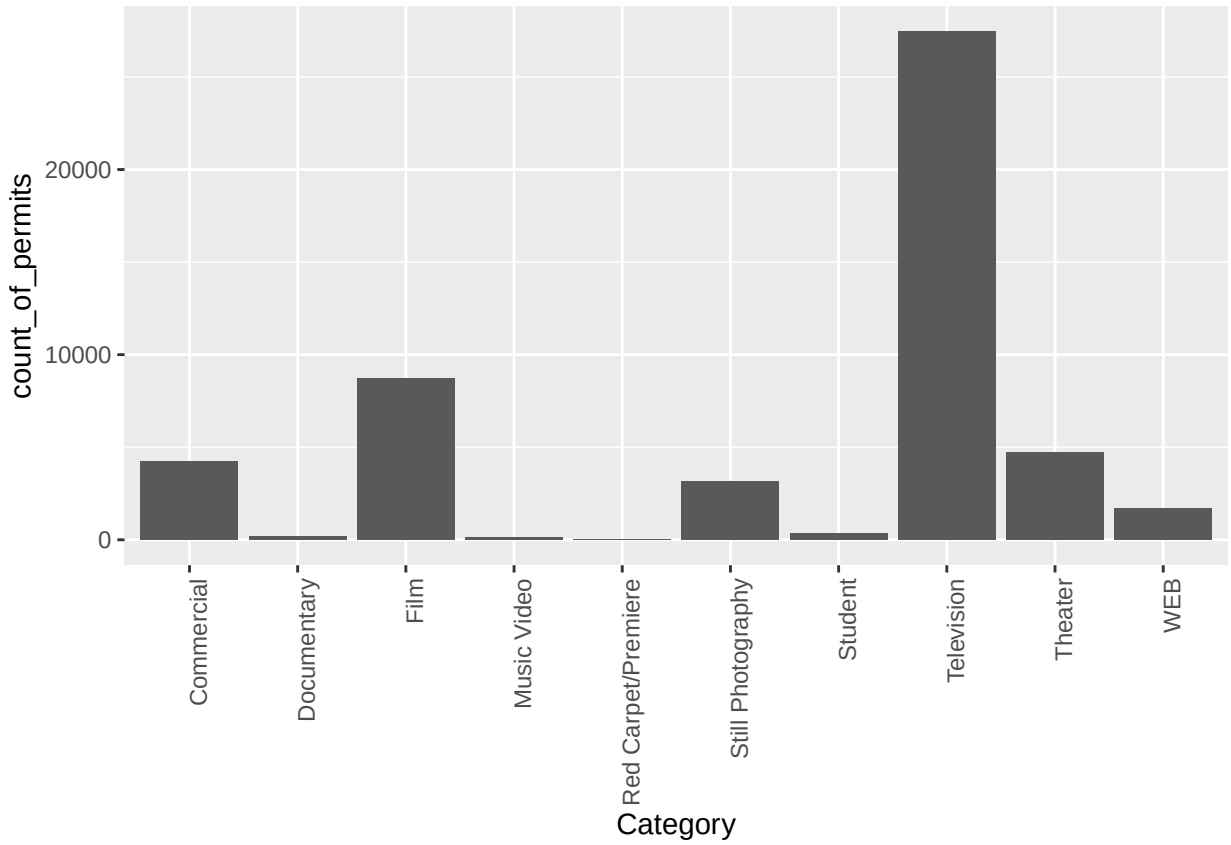
There it is, we're done here! We can easily look at this graph, and answer our question. Most of the film permits were requested in Manhattan, followed by Brooklyn, then Queen's, the Bronx, and finally Staten Island.

1.2.3.2 What kind of "films" are being made, what is the category?

We think you might be skeptical of what you are doing here, copying and pasting things. Soon you'll see just how fast you can do things by copying and pasting, and make a few little changes. Let's quickly ask another question about what kinds of films are being made. The column `Category`, gives us some information about that. Let's just copy paste the code we already made, and see what kinds of categories the films fall into. See if you can tell what I changed in the code to make this work, I'll do it all at once:

```
counts <- nyc_films %>%
          group_by(Category) %>%
          summarize(count_of_permits = length(Category))

ggplot(counts, aes(x = Category, y = count_of_permits )) +
```

```
  geom_bar(stat="identity")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



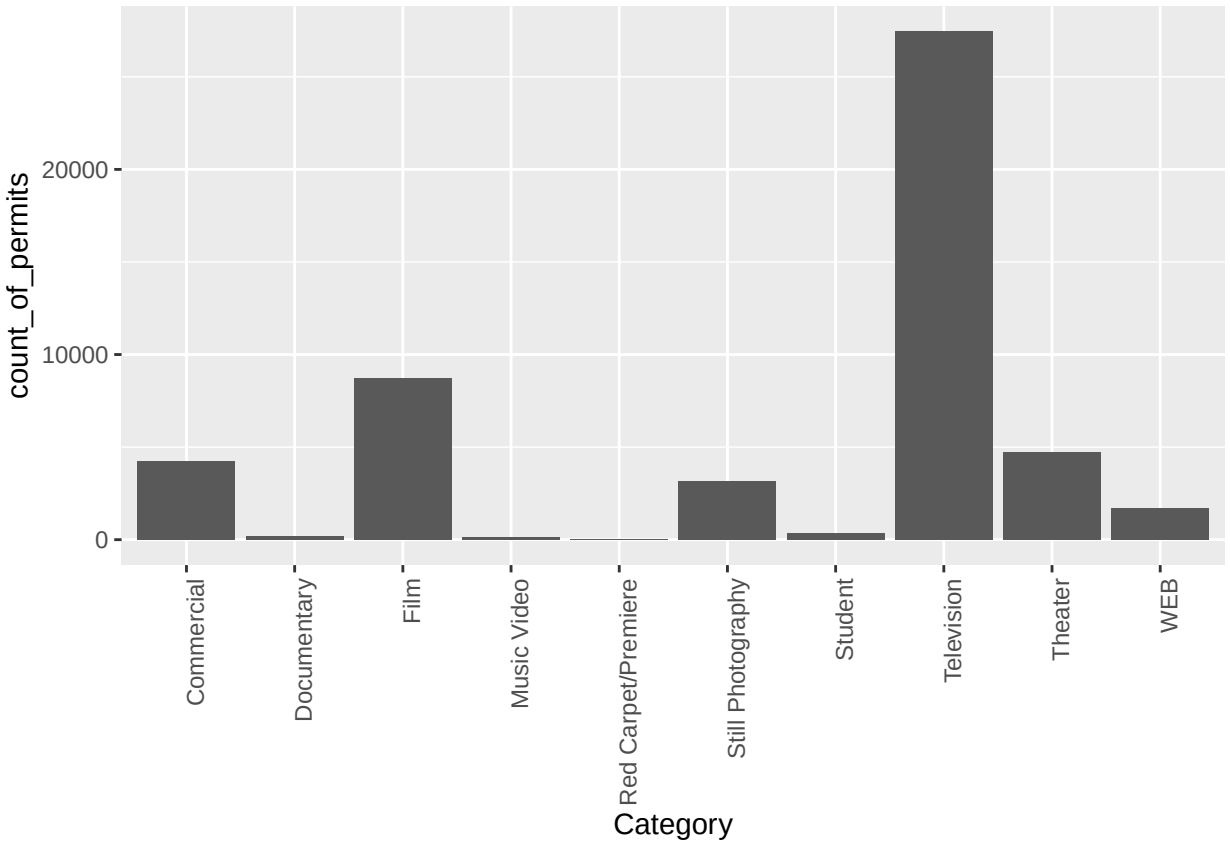
Ok, so this figure might look a bit weird because the labels on the bottom are running into each other. We'll fix that in a bit. First, let's notice the changes.

1. I changed `Borough` to `Category`. That was the main thing
2. I left out a bunch of things from before. None of the `library()` commands are used again, and I didn't re-run the very early code to get the data. R already has those things in it's memory, so we don't need to do that first. If you ever clear the memory of R, then you will need to reload those things. First-things come first.

Fine, so how do we fix the graph? Good question. To be honest, I don't know right now. I totally forgot how. But, I know ggplot2 can do this, and I'm going to google it, right now. Then I'm going to find the answer, and use it here. The googling of your questions is a fine way to learn. It's what everybody does these days....[goes to google...].

Found it, actually found a lot of ways to do this. The trick is to add the last line. I just copy-pasted it from the solution I found on stack overflow (you will become friend's with stack overflow, there are many solutions there to all of your questions)

```
counts <- nyc_films %>%
          group_by(Category) %>%
          summarize(count_of_permits = length(Category))

ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity")+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



1.2.4 ggplot2 basics

Before we go further, I want to point out some basic properties of ggplot2, just to give you a sense of how it is working. This will make more sense in a few weeks, so come back here to remind yourself. We'll do just a bit a basics, and then move on to making more graphs, by copying and pasting.

The ggplot function uses layers. Layers you say? What are these layers? Well, it draws things from the bottom up. It lays down one layer of graphics, then you can keep adding on top, drawing more things. So the idea is something like: Layer 1 + Layer 2 + Layer 3, and so on. If you want Layer 3 to be Layer 2, then you just switch them in the code.

Here is a way of thinking about ggplot code

```
ggplot(name_of_data, aes(x = name_of_x_variable, y = name_of_y_variable)) +
    geom_layer()+
    geom_layer()+
    geom_layer()
```

What I want you to focus on in the above description is the + signs. What we are doing with the plus signs is adding layers to plot. The layers get added in the order that they are written. If you look back to our previous code, you will see we add a `geom_bar` layer, then we added another layer to change the rotation of the words on the x-axis. This is how it works.

BUT WAIT? How am I supposed to know what to add? This is nuts! We know. You're not supposed to know just yet, how could you? We'll give you lots of examples where you can copy and paste, and they will work. That's how you'll learn. If you really want to read the help manual you can do that too. It's on the ggplot2 website. This will become useful after you already know what you are doing, before that, it will

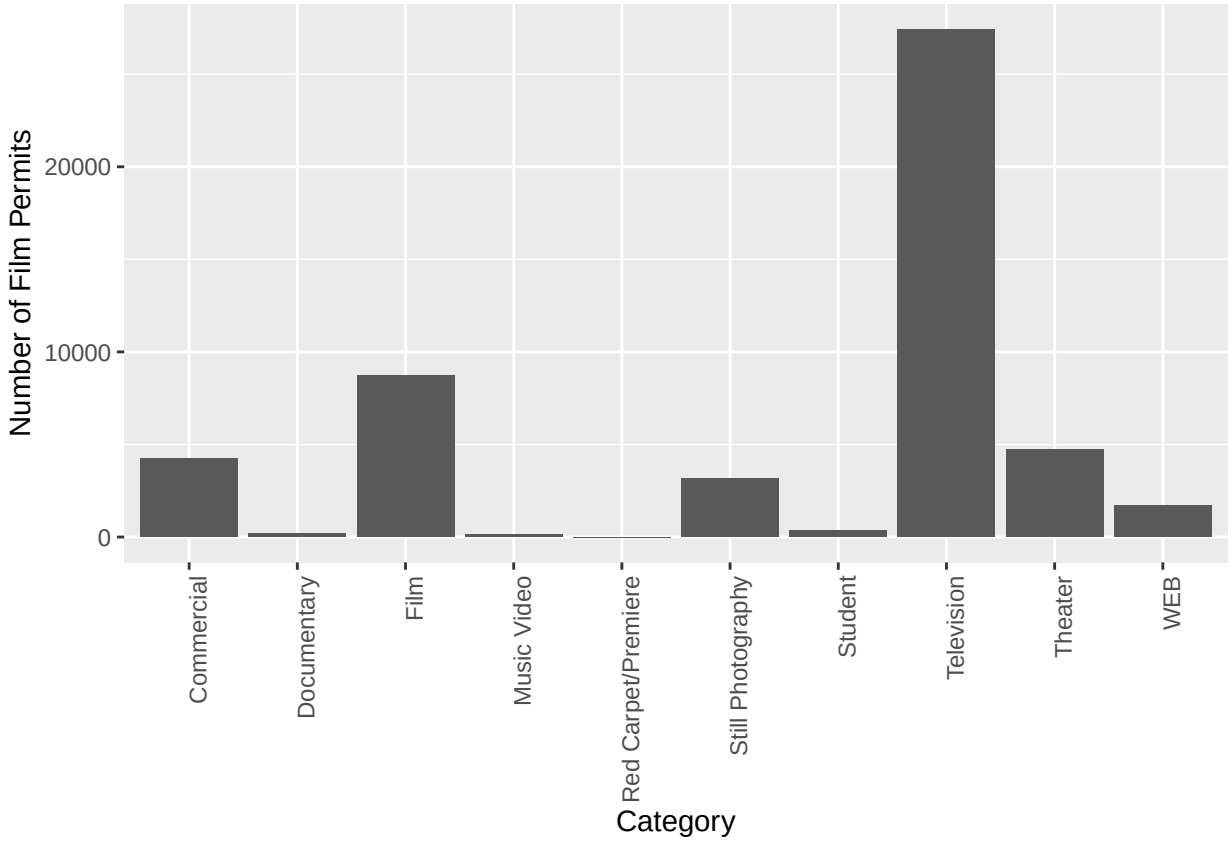
probably just seem very confusing. However, it is pretty neat to look and see all of the different things you can do, it's very powerful.

For now, let's the get the hang of adding things to the graph that let us change some stuff we might want to change. For example, how do you add a title? Or change the labels on the axes? Or add different colors, or change the font-size, or change the background? You can change all of these things by adding different lines to the existing code.

1.2.4.1 ylab() changes y label

The last graph had `count_of_permits` as the label on the y-axis. That doesn't look right. ggplot2 automatically took the label from the column, and made it be the name on the y-axis. We can change that by adding `ylab("what we want")`. We do this by adding a + to the last line, then adding `ylab()`

```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits")
```

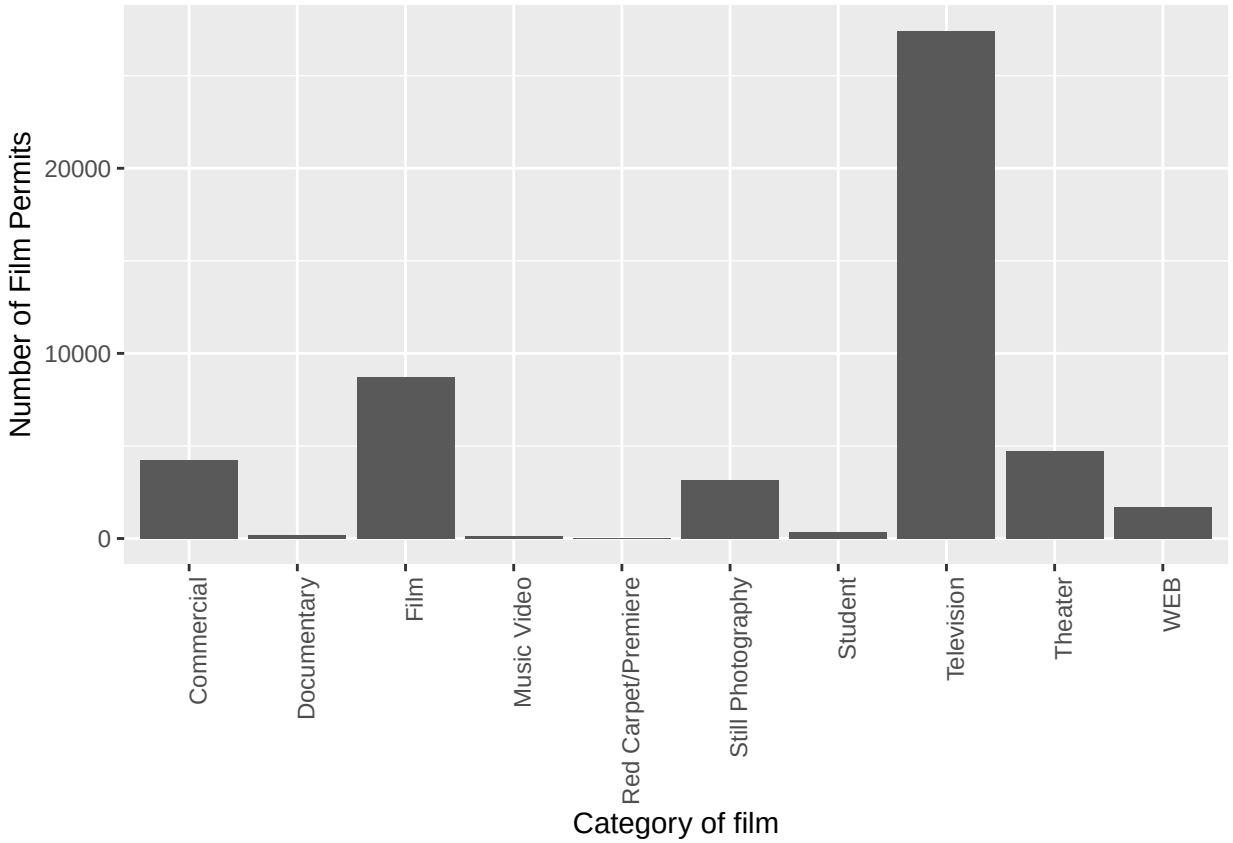


1.2.4.2 xlab() changes x label

Let's slightly modify the x label too:

```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
```

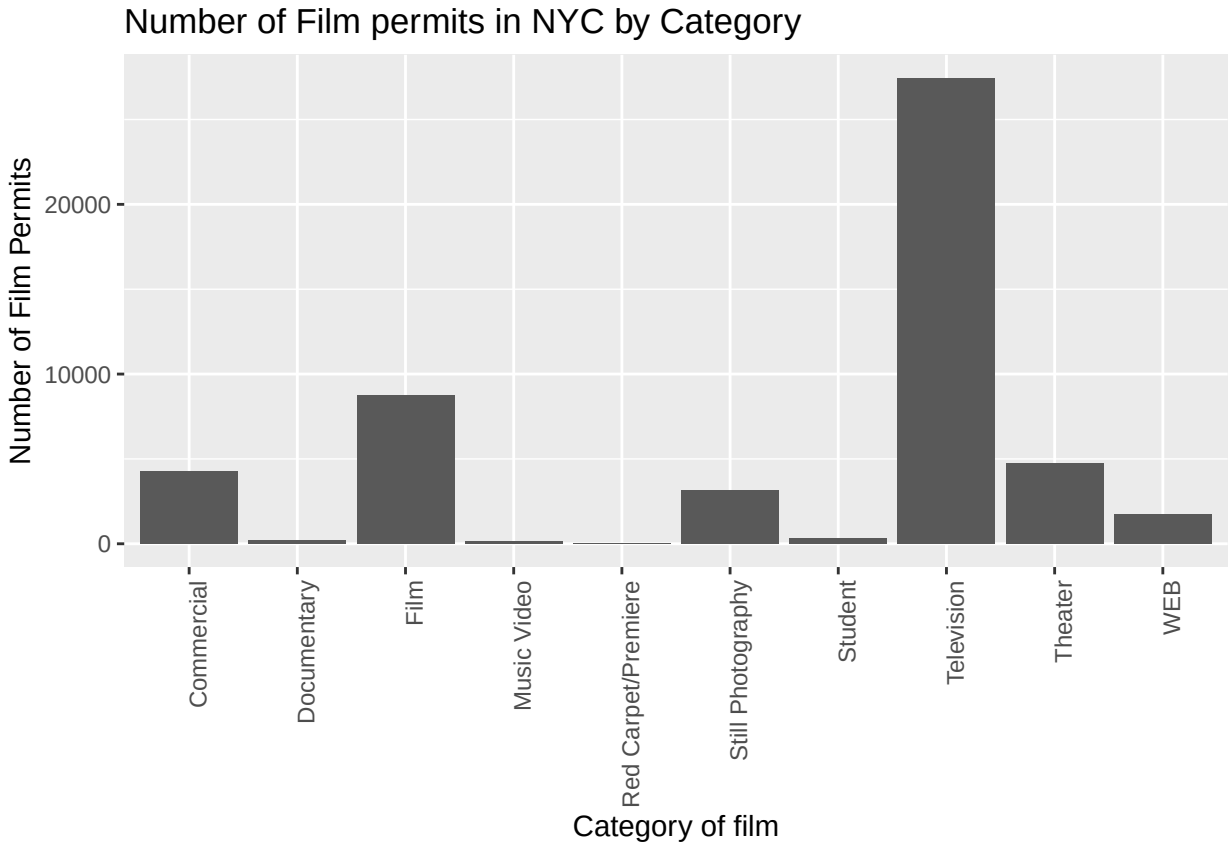
```
  ylab("Number of Film Permits") +
  xlab("Category of film")
```



1.2.4.3 ggtitle() adds title

Let's give our graph a title

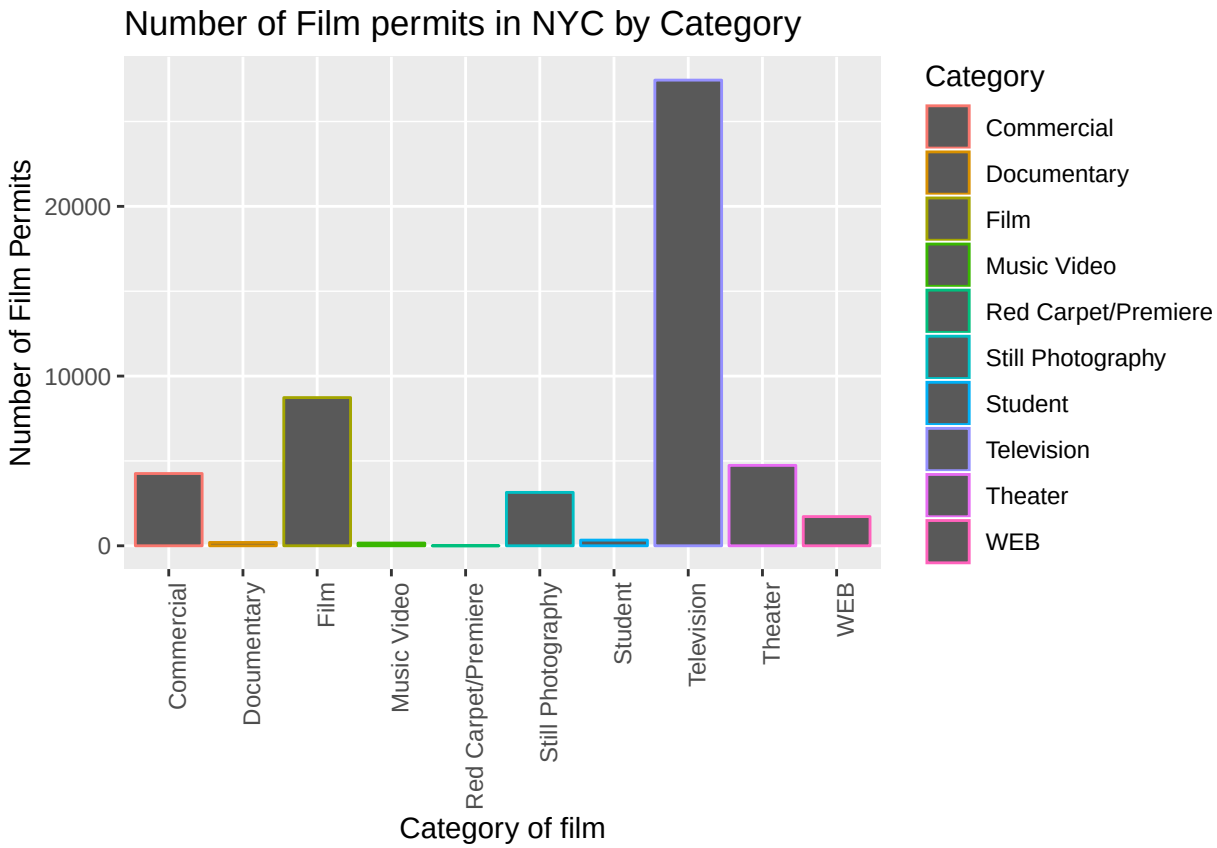
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.4.4 color adds color

Let's make the bars different colors. To do this, we add new code to the inside of the `aes()` part:

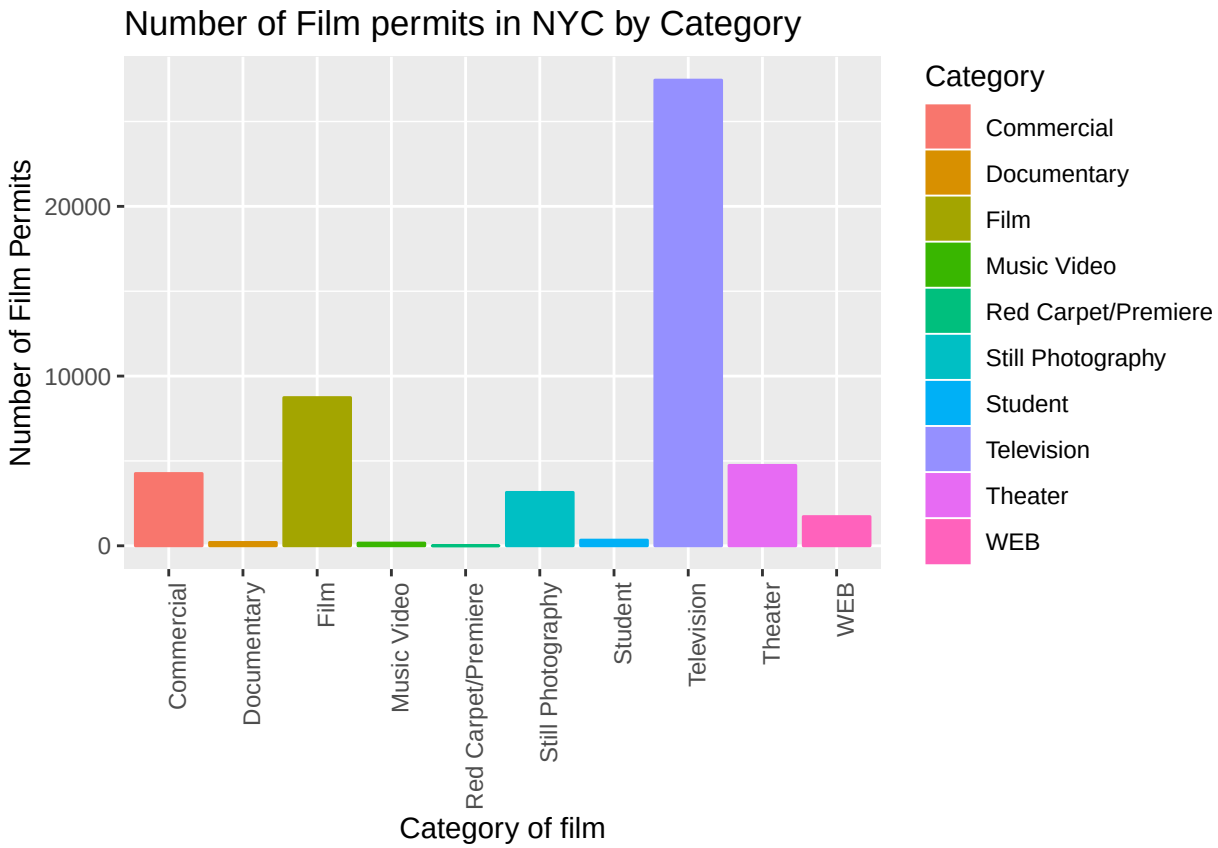
```
ggplot(counts, aes(x = Category, y = count_of_permits, color=Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.4.5 fill fills in color

Let's make the bars different colors. To do this, we add new code to the inside of the `aes()` part...Notice I've started using new lines to make the code more readable.

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```

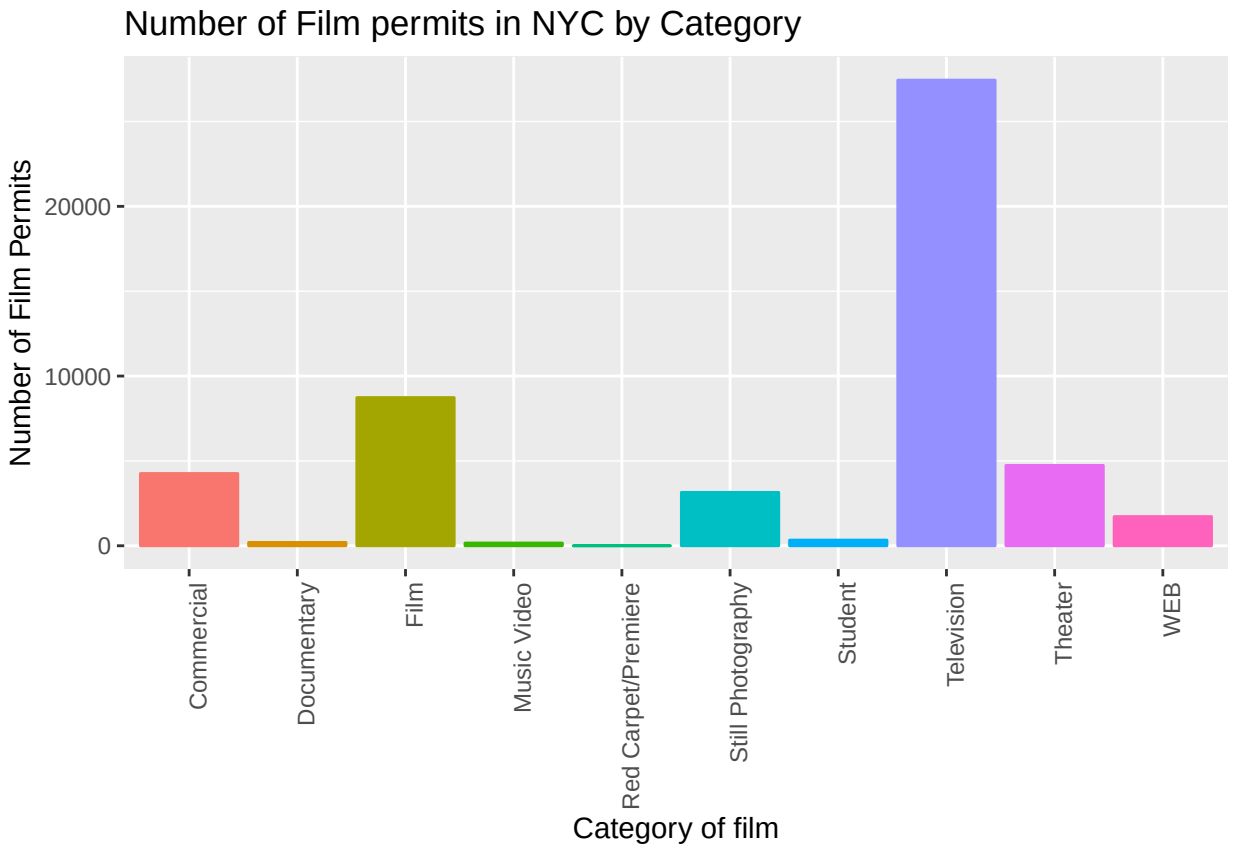


1.2.4.6 get rid of the legend

Sometimes you just don't want the legend on the side, to remove it add

```
theme(legend.position="none")
```

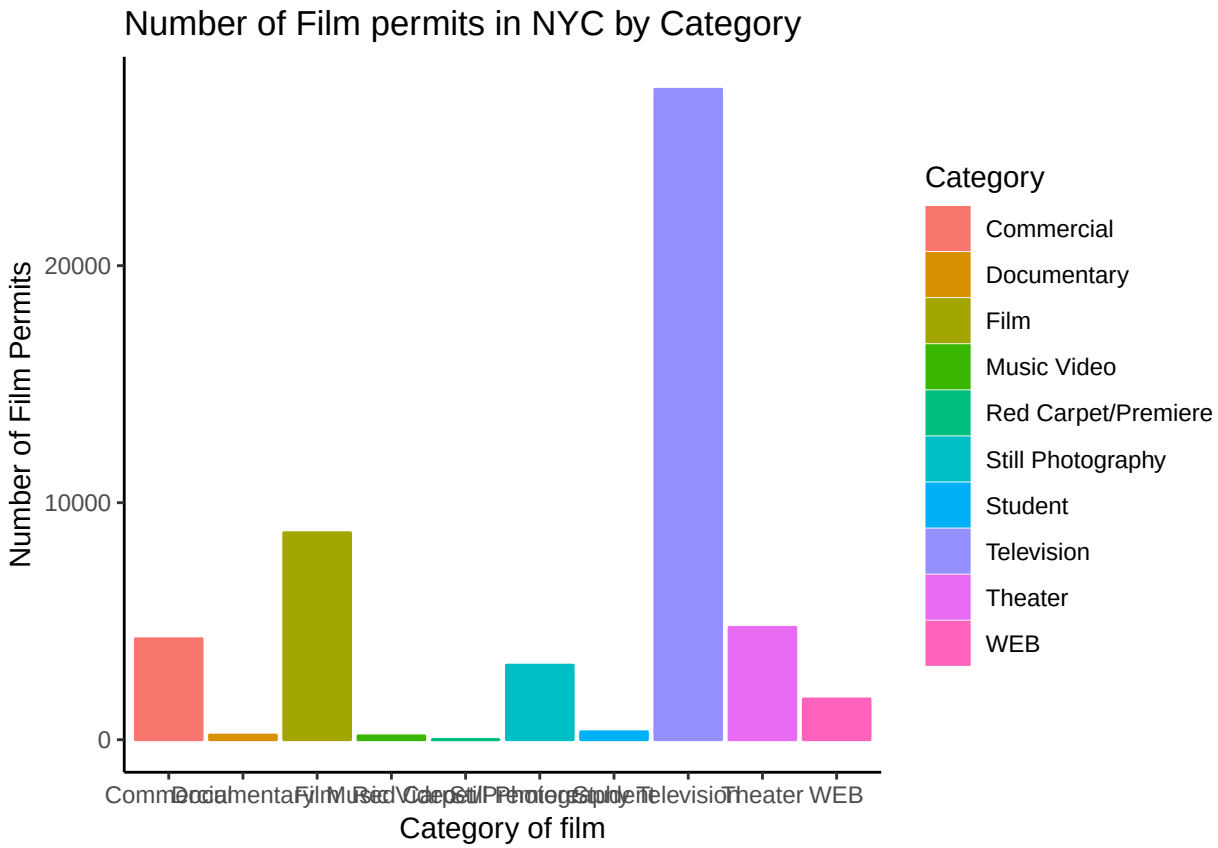
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.4.7 theme_classic() makes white background

The rest is often just visual preference. For example, the graph above has this grey grid behind the bars. For a clean classic no nonsense look, use `theme_classic()` to take away the grid.

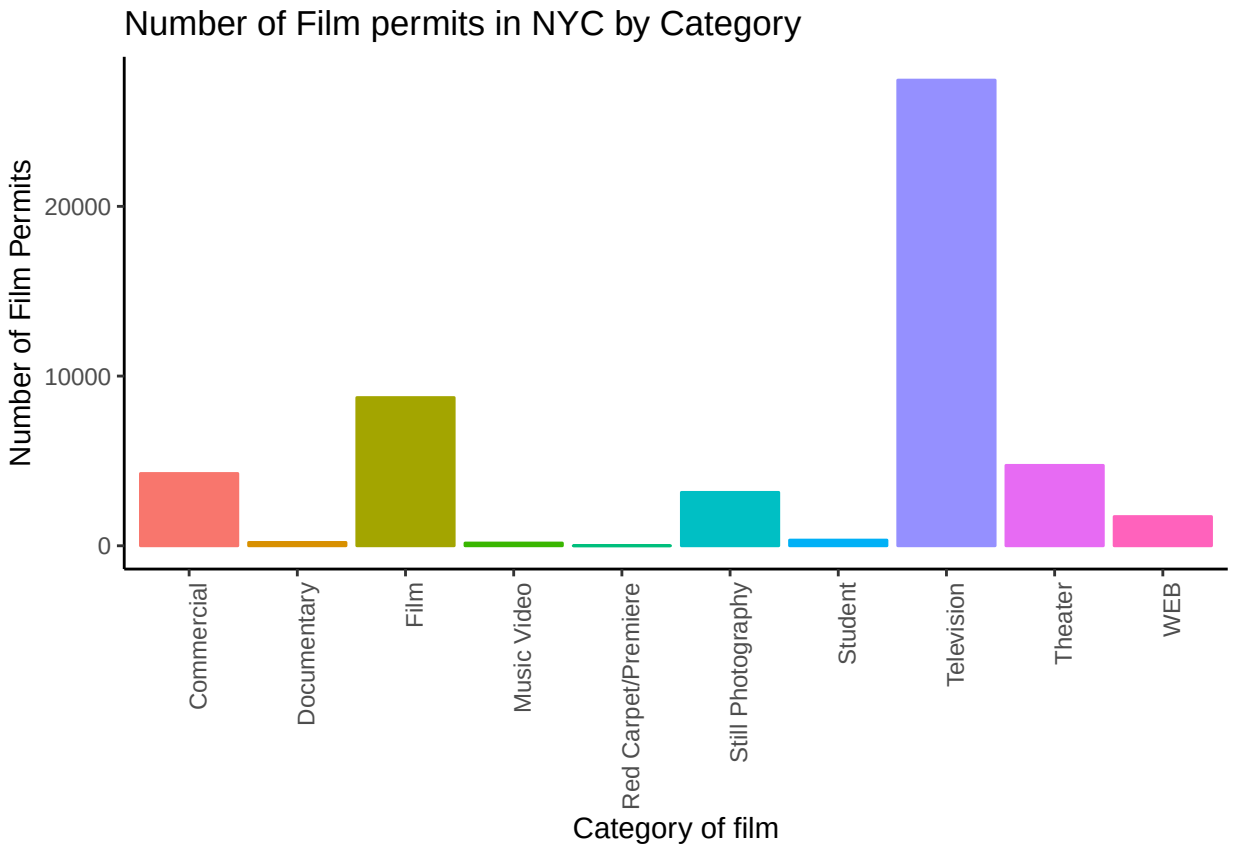
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none") +
  theme_classic()
```



1.2.4.8 Sometimes layer order matters

Interesting, `theme_classic()` is misbehaving a little bit. It looks like we have some of our layer out of order, let's re-order. I just moved `theme_classic()` to just underneath the `geom_bar()` line. Now everything get's drawn properly.

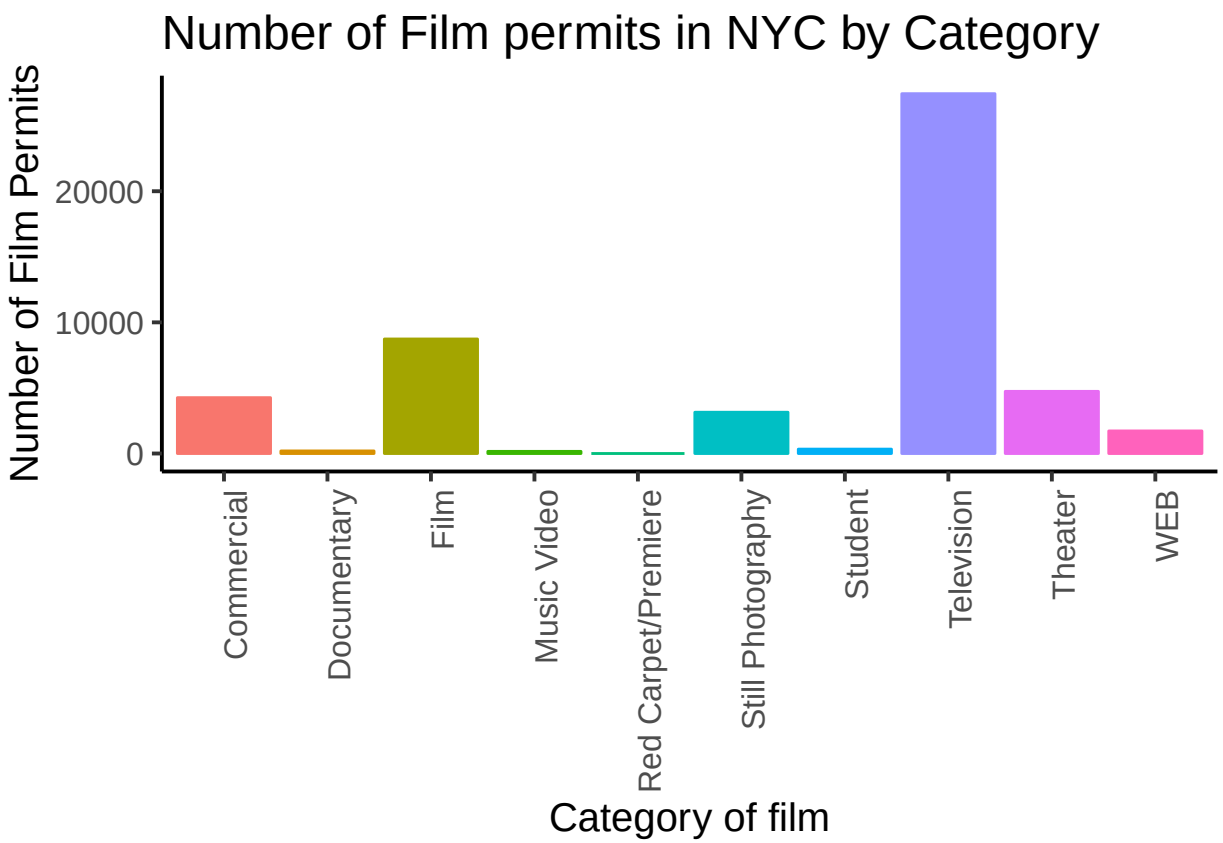
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.4.9 Font-size

Changing font-size is often something you want to do. ggplot2 can do this in different ways. I suggest using the `base_size` option inside `theme_classic()`. You set one number for the largest font size in the graph, and everything else gets scaled to fit with that that first number. It's really convenient. Look for the insisde of `theme_classic()`

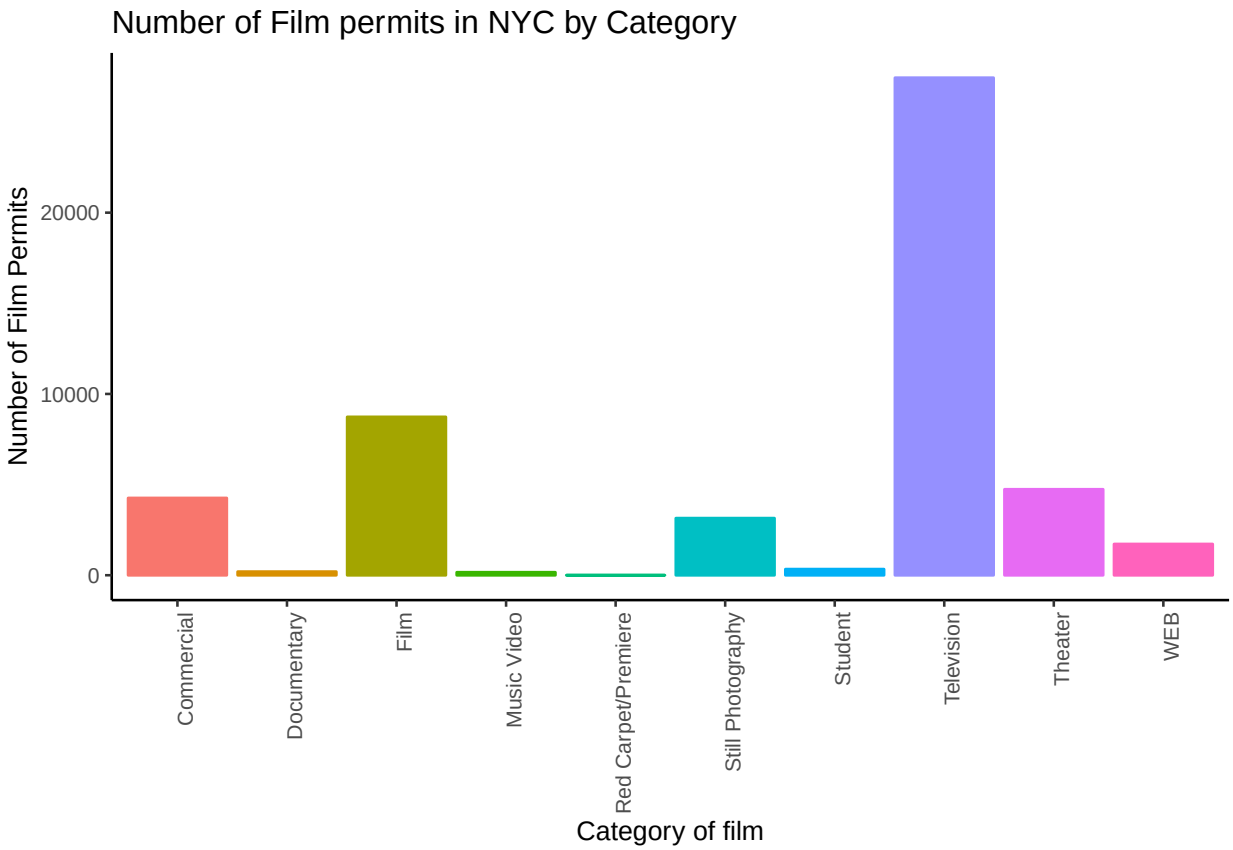
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 15) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



or

make things small... just to see what happens

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category") +
  theme(legend.position="none")
```



1.2.4.10 ggplot2 summary

That's enough of the ggplot2 basics for now. You will discover that many things are possible with ggplot2. It is amazing. We are going to get back to answering some questions about the data with graphs. But, now that we have built the code to make the graphs, all we need to do is copy-paste, and make a few small changes, and boom, we have our graph.

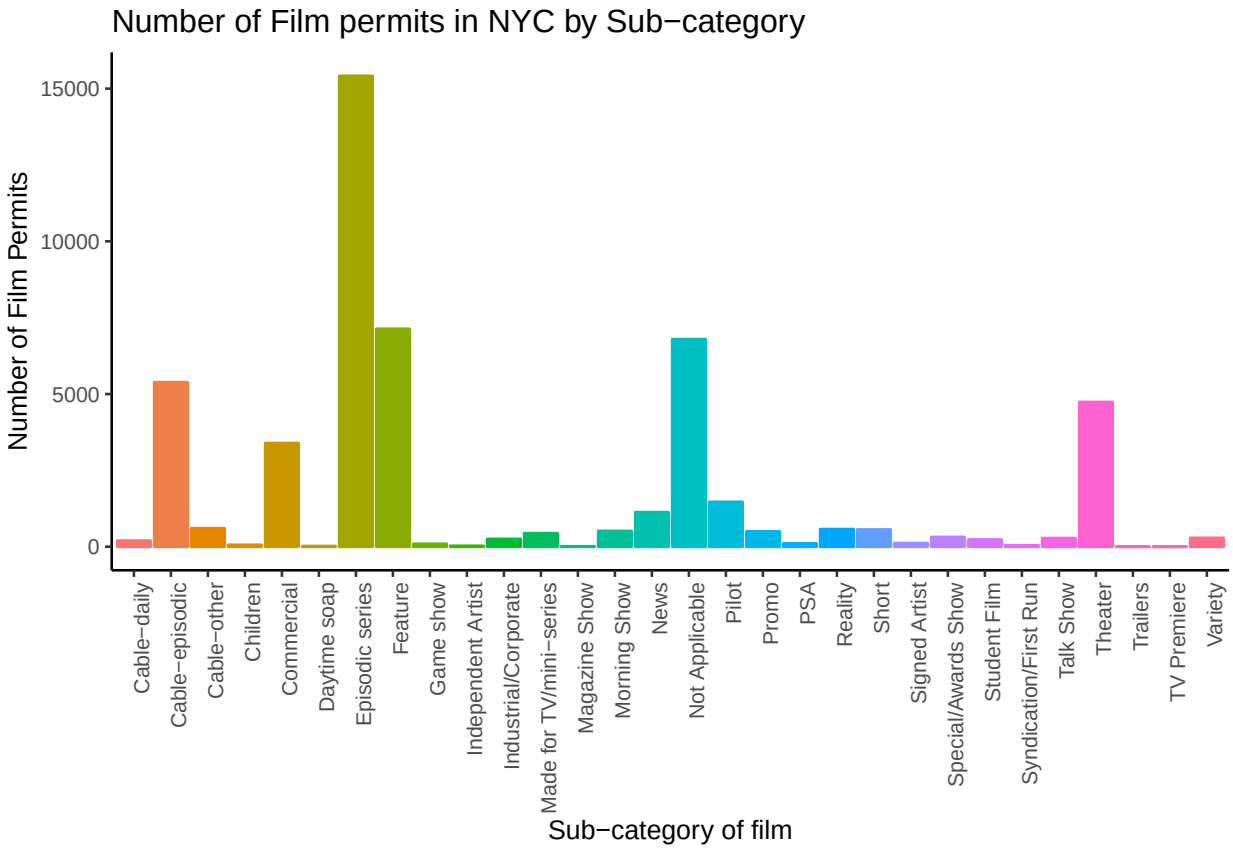
1.2.5 More questions about NYC films

1.2.5.1 What are the sub-categories of films?

Notice the `nyc_films` data frame also has a column for `SubCategoryName`. Let's see what's going on there with a quick plot.

```
# get the counts (this is a comment it's just here for you to read)

counts <- nyc_films %>%
          group_by(SubCategoryName) %>%
          summarize(count_of_permits = length(SubCategoryName))

# make the plot

ggplot(counts, aes(x = SubCategoryName, y = count_of_permits,
                   color=SubCategoryName,
                   fill= SubCategoryName )) +
```

```
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Sub-category of film") +
  ggtitle("Number of Film permits in NYC by Sub-category") +
  theme(legend.position="none")
```



I guess "episodic series" are the most common. Using a graph like this gave us our answer super fast.

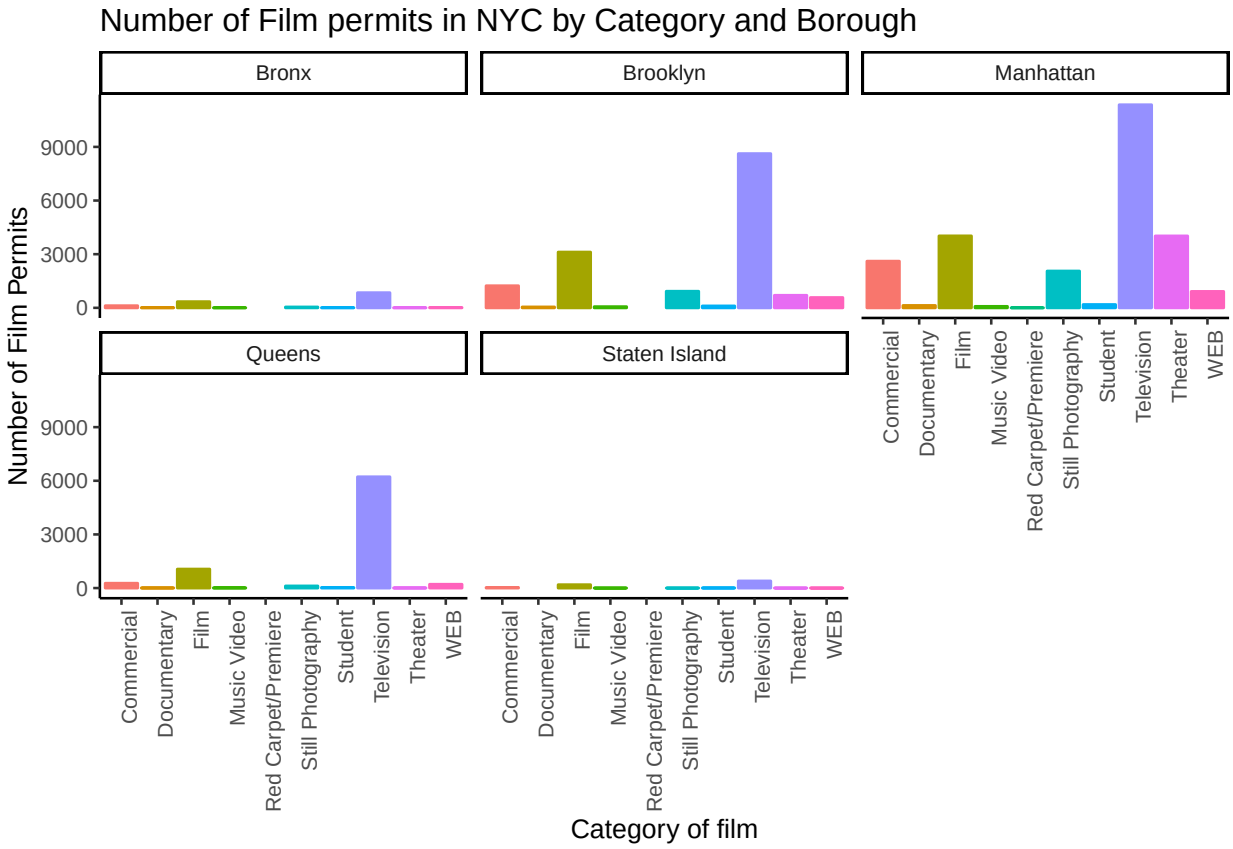
1.2.5.2 Categories by different Boroughs

Let's see one more really useful thing about ggplot2. It's called `facet_wrap()`. It's an ugly word, but you will see that it is very cool, and you can do next-level-super-hero graph styles with `facet_wrap` that other people can't do very easily.

Here's our question. We know that some films are made in different Boroughs, and that sme films are made in different categories, but do different Boroughs have different patterns for the kinds of categories of films they request permits for? Are their more tv shows in Brookyln? How do we find out? Watch, just like this:

```
# get the counts (this is a comment it's just here for you to read)

counts <- nyc_films %>%
          group_by(Borough,Category) %>%
          summarize(count_of_permits = length(Category))

# make the plot
```

```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category and Borough") +
  theme(legend.position="none") +
  facet_wrap(~Borough, ncol=3)
```

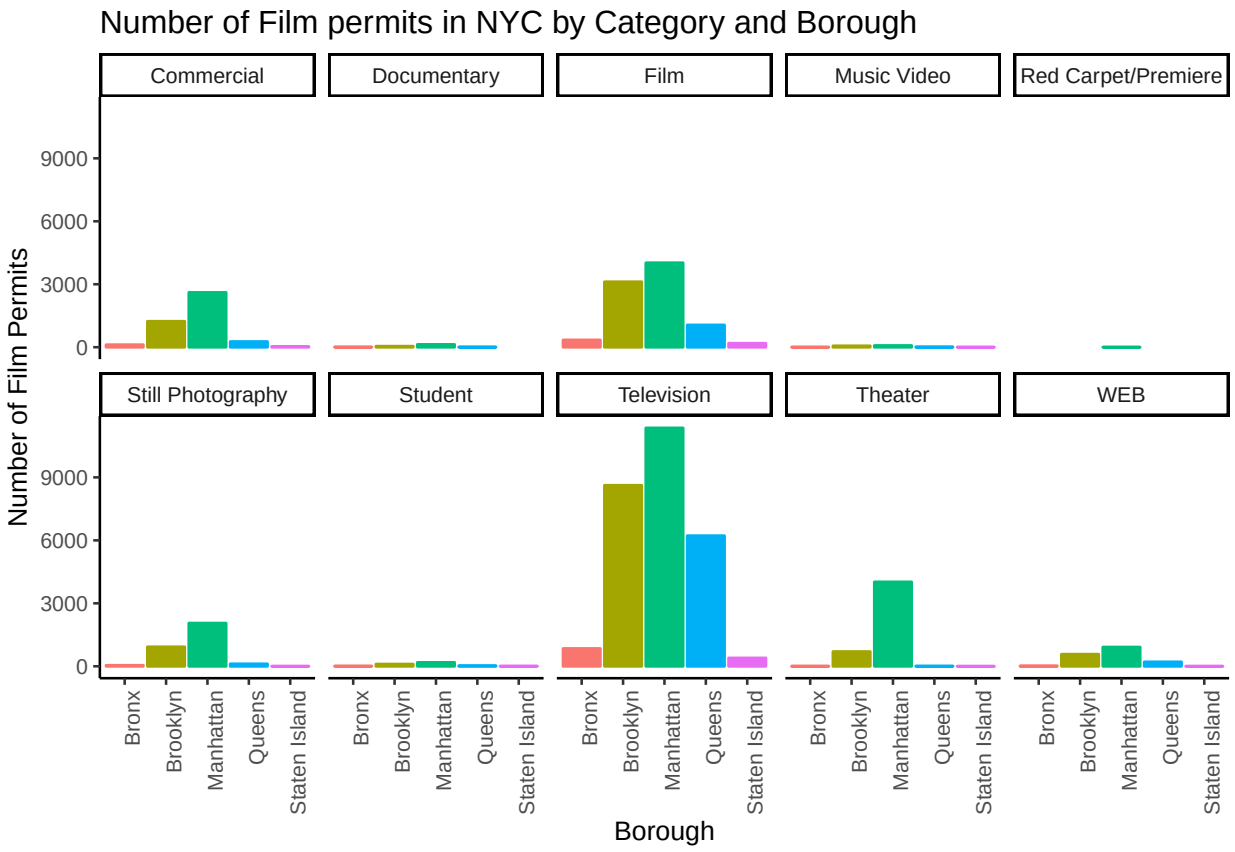


We did two important things. First we added `Borough` and `Category` into the `group_by()` function. This automatically gives separate counts for each category of film, for each Borough. Then we added `facet_wrap(~Borough, ncol=3)` to the end of the plot, and it automatically drew us 5 different bar graphs, one for each Borough! That was fast. Imagine doing that by hand.

The nice thing about this is we can switch things around if we want. For example, we could do it this way by switching the `Category` with `Borough`, and facet-wrapping by Category instead of Borough like we did above. Do what works for you.

```
ggplot(counts, aes(x = Borough, y = count_of_permits,
                   color=Borough,
                   fill= Borough )) +
  geom_bar(stat="identity") +
  theme_classic(base_size = 10) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
```

```
  ylab("Number of Film Permits") +
  xlab("Borough") +
  ggtitle("Number of Film permits in NYC by Category and Borough") +
  theme(legend.position="none") +
  facet_wrap(~Category, ncol=5)
```



1.2.6 Gapminder Data

https://www.gapminder.org is an organization that collects some really interesting worldwide data. They also make cool visualization tools for looking at the data. There are many neat examples, and they have visualization tools built right into their website that you can play around with https://www.gapminder.org/tools/. That's fun check it out.

There is also an R package called `gapminder`. When you install this package, it loads in some of the data from gapminder, so we can play with it in R.

If you don't have the gapminder packe installed, you can install it by running this code

```
install.packages("gapminder")
```

Once the package is installed, you need to load the new library, like this. Then, you can put the `gapminder` data into a data frame, like we do here: `gapminder_df`.

```
library(gapminder)
gapminder_df<-gapminder
```

1.2.6.1 Look at the data frame

You can look at the data frame to see what is in it, and you can use `summarytools` again to view a summary of the data.

```
view(dfSummary(gapminder_df))
```

There are 1704 rows of data, and we see some columns for country, continent, year, life expectancy, population, and GDP per capita.

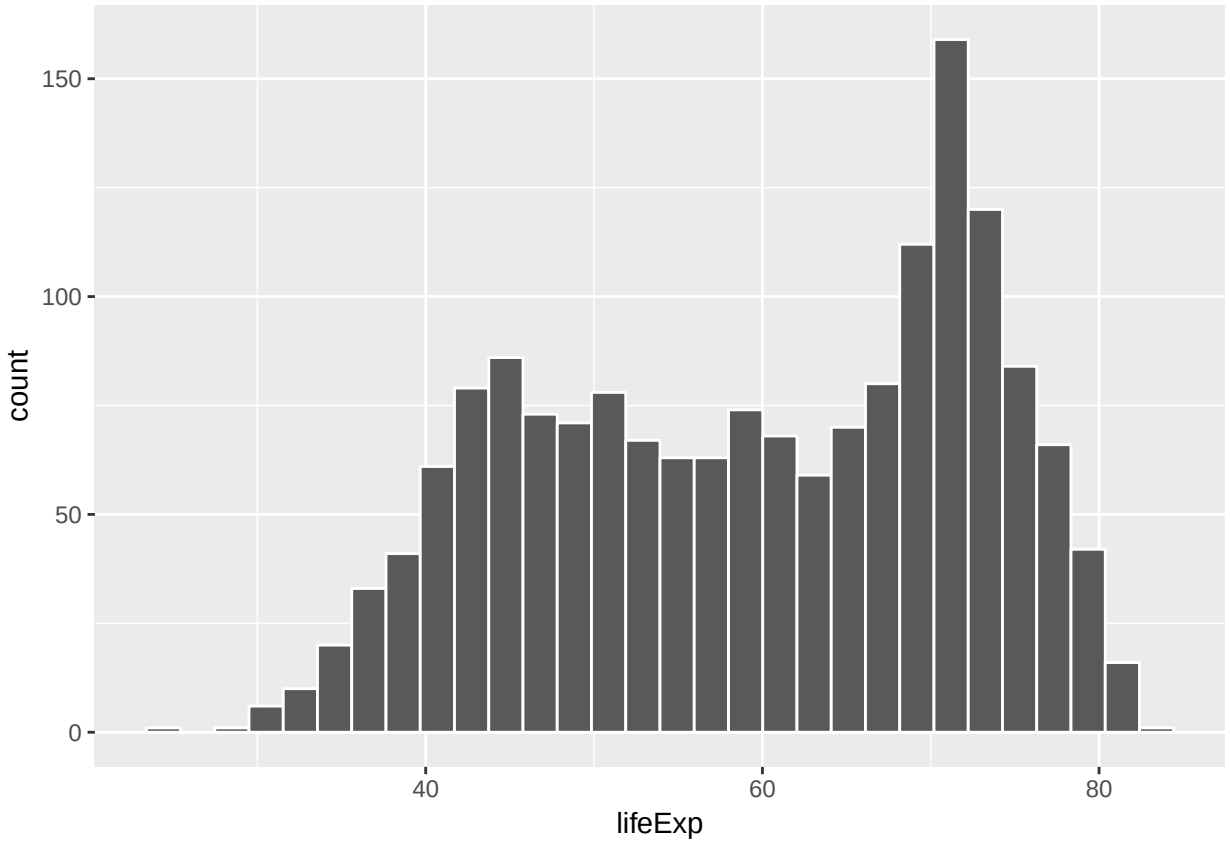
1.2.7 Asking Questions with the gap minder data

We will show you how to graph some the data to answer a few different kinds of questions. Then you will form your own questions, and see if you can answer them with ggplot2 yourself. All you will need to do is copy and paste the following examples, and change them up a little bit

1.2.7.1 Life Expectancy histogram

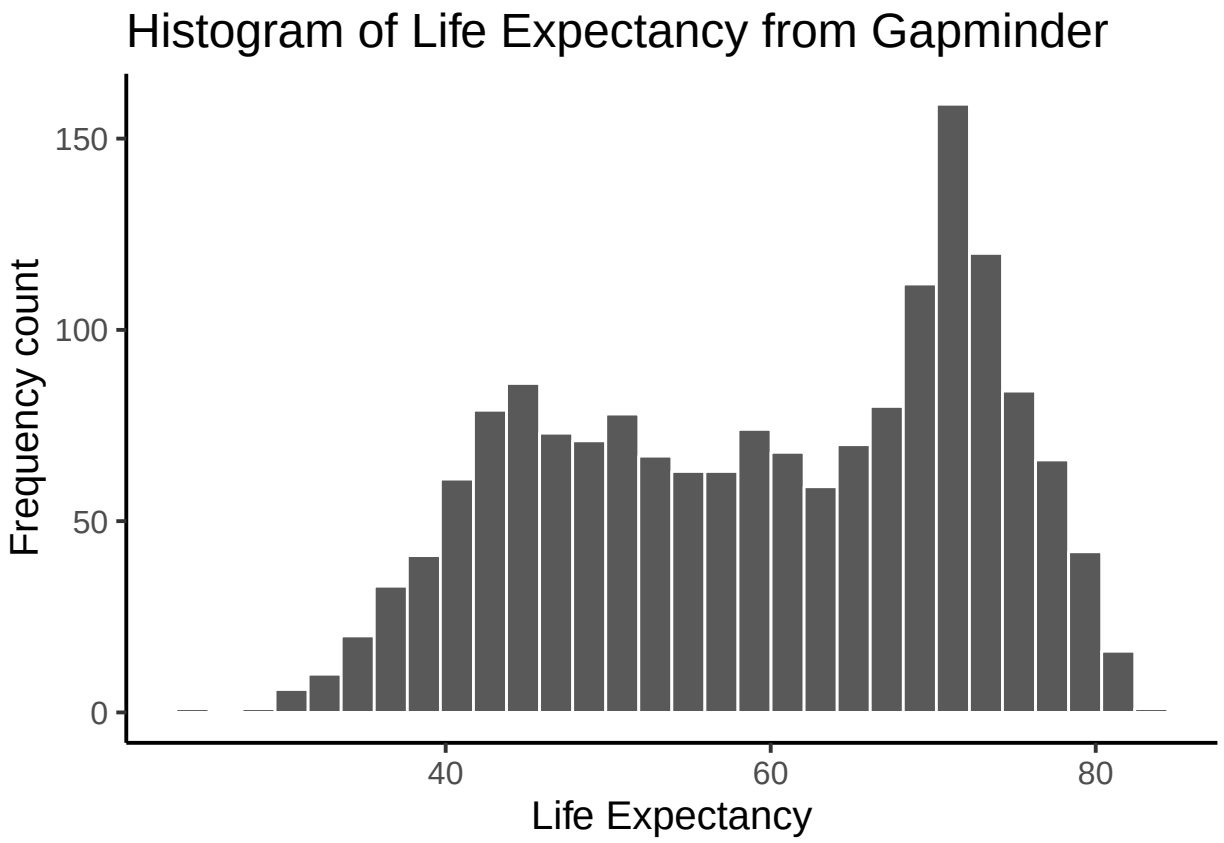
How long are people living all around the world according to this data set? There are many ways we could plot the data to find out. The first way is a histogram. We have many numbers for life expectancy in the column `lifeExp`. This is a big sample, full of numbers for 142 countries across many years. It's easy to make a histrogram in ggplot to view the distribution:

```
ggplot(gapminder_df, aes(x=lifeExp))+
  geom_histogram(color="white")
```



See, that was easy. Next, is a code block that adds more layers and settings if you wanted to modify parts of the graph:

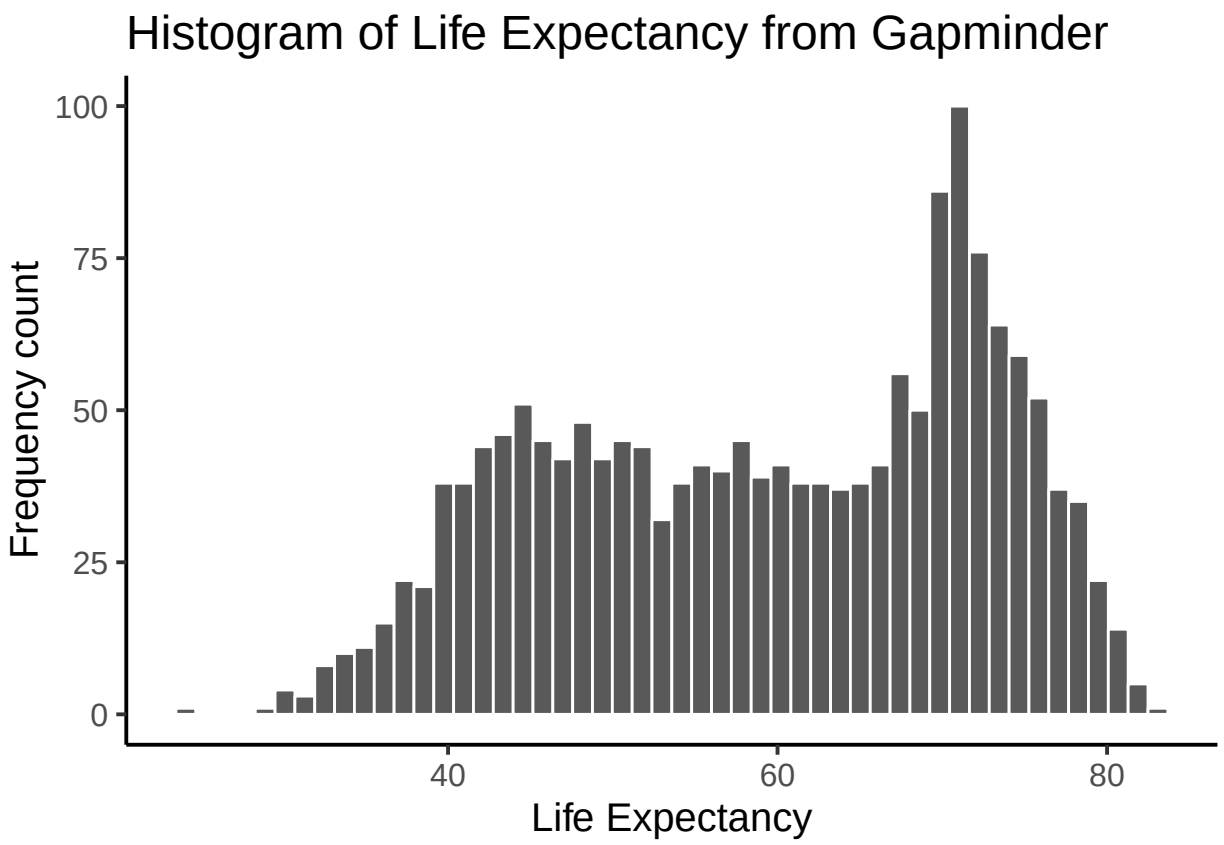
```
ggplot(gapminder_df, aes(x = lifeExp)) +
  geom_histogram(color="white")+
  theme_classic(base_size = 15) +
  ylab("Frequency count") +
  xlab("Life Expectancy") +
  ggtitle("Histogram of Life Expectancy from Gapminder")
```



The histogram shows a wide range of life expectancies, from below 40 to just over 80. Histograms are useful, they can show you what kinds of values happen more often than others.

One final thing about histograms in ggplot. You may want to change the bin size. That controls how wide or narrow, or the number of bars (how they split across the range), in the histogram. You need to set the `bins=` option in `geom_histogram()`.

```
ggplot(gapminder_df, aes(x = lifeExp)) +
  geom_histogram(color="white", bins=50)+
  theme_classic(base_size = 15) +
  ylab("Frequency count") +
  xlab("Life Expectancy") +
  ggtitle("Histogram of Life Expectancy from Gapminder")
```



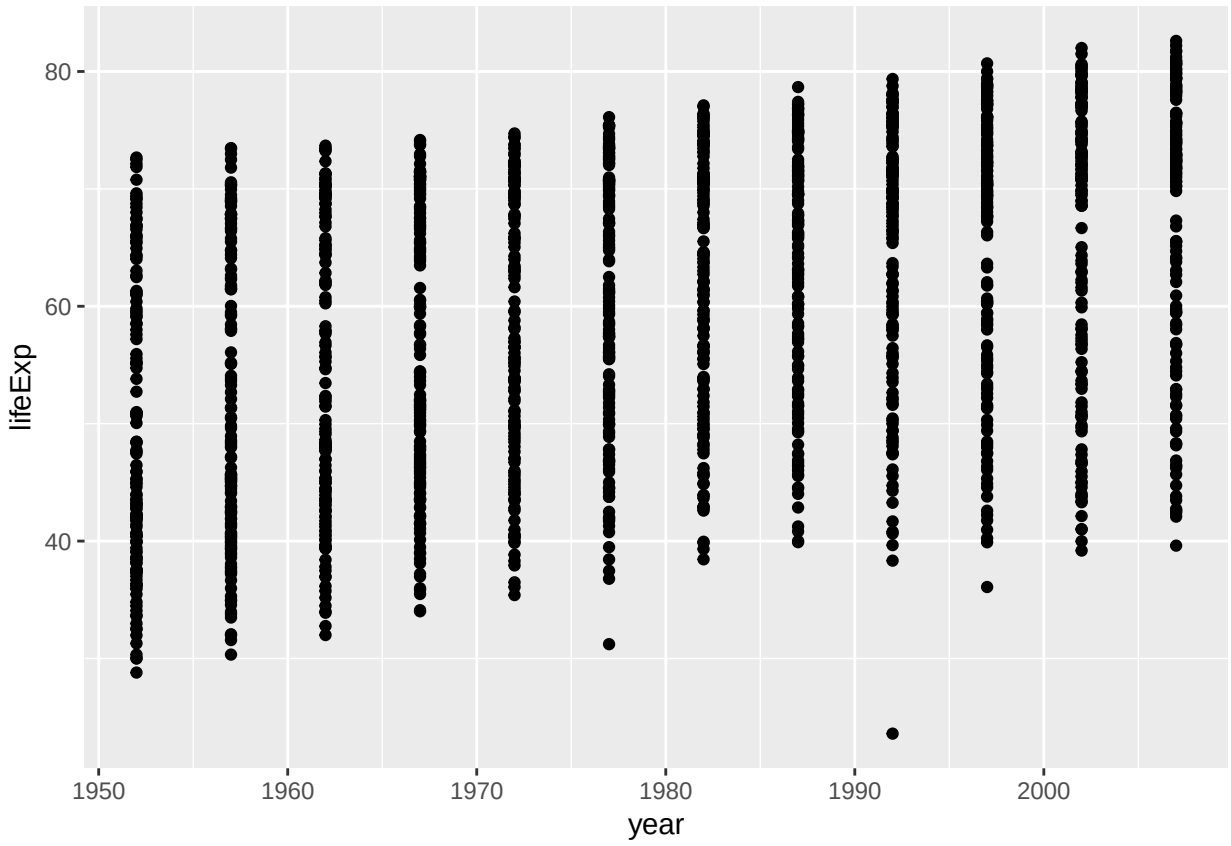
See, same basic patter, but now breaking up the range into 50 little equal sized bins, rather than 30, which is the default. You get to choose what you want to do.

1.2.7.2 Life Expectancy by year Scatterplot

We can see we have data for life expectancy and different years. So, does worldwide life expectancy change across the years in the data set? As we go into the future, are people living longer?

Let's look at this using a scatterplot. We can set the x-axis to be year, and the y-axis to be life expectancy. Then we can use `geom_point()` to display a whole bunch of dots, and then look at them. Here's the simple code:

```
ggplot(gapminder_df, aes(y= lifeExp, x= year))+
  geom_point()
```

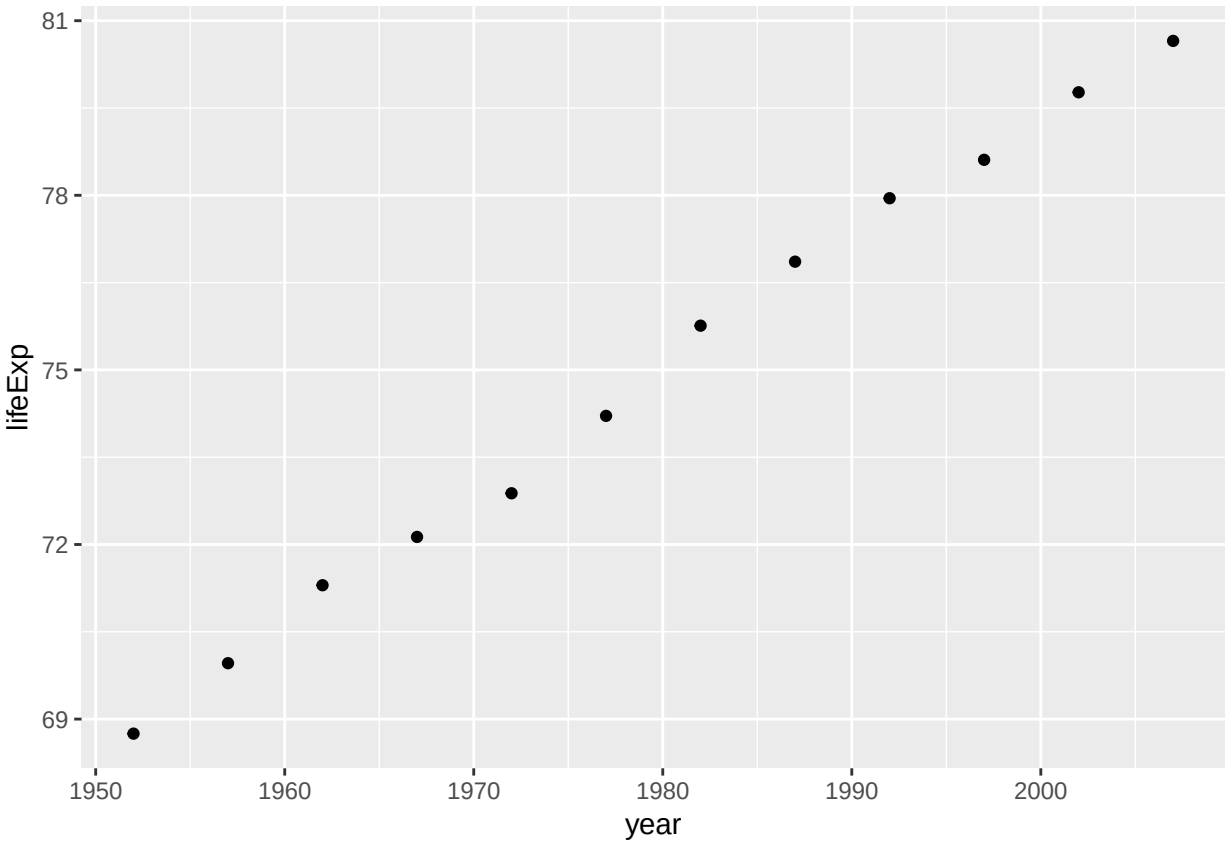


Whoa, that's a lot of dots! Remember that each country is measured each year. So, the bands of dots you see, show the life expectancies for the whole range of countries within each year of the database. There is a big spread inside each year. But, on the whole it looks like groups of dots slowly go up over years.

1.2.7.3 One country, life expectancy by year

I'm (Matt) from Canada, so maybe I want to know if life expectancy for Canadians is going up over the years. To find out the answer for one country, we first need to split the full data set, into another smaller data set that only contains data for Canada. In other words, we want only the rows where the word "Canada" is found in the `country` column. We will use the `filter` function from 'dplyr for this:

```
# filter rows to contain Canada

smaller_df <- gapminder_df %>%
                filter(country == "Canada")

# plot the new data contained in smaller_df

ggplot(smaller_df, aes(y= lifeExp, x= year))+
  geom_point()
```

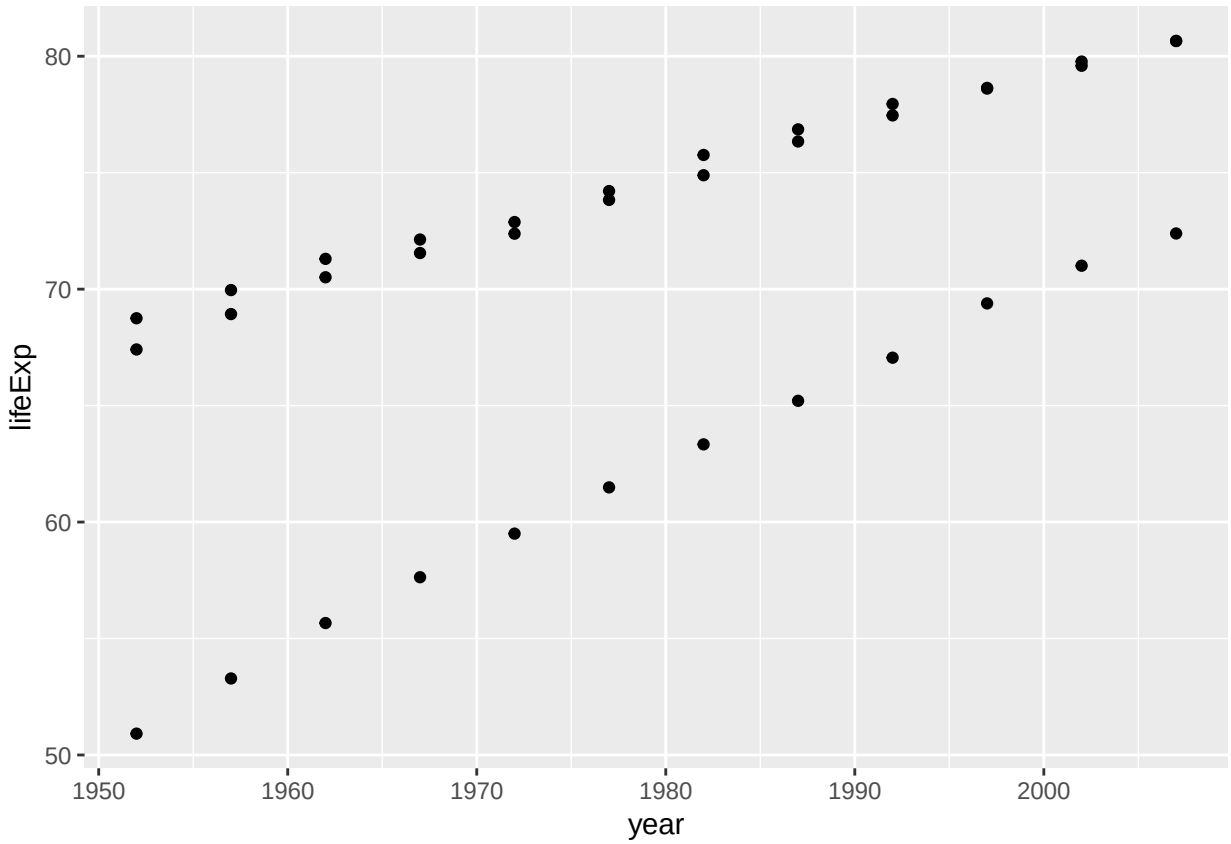


I would say things are looking good for Canadians, their life expectancy is going up over the years!

1.2.7.4 Multiple countries scatterplot

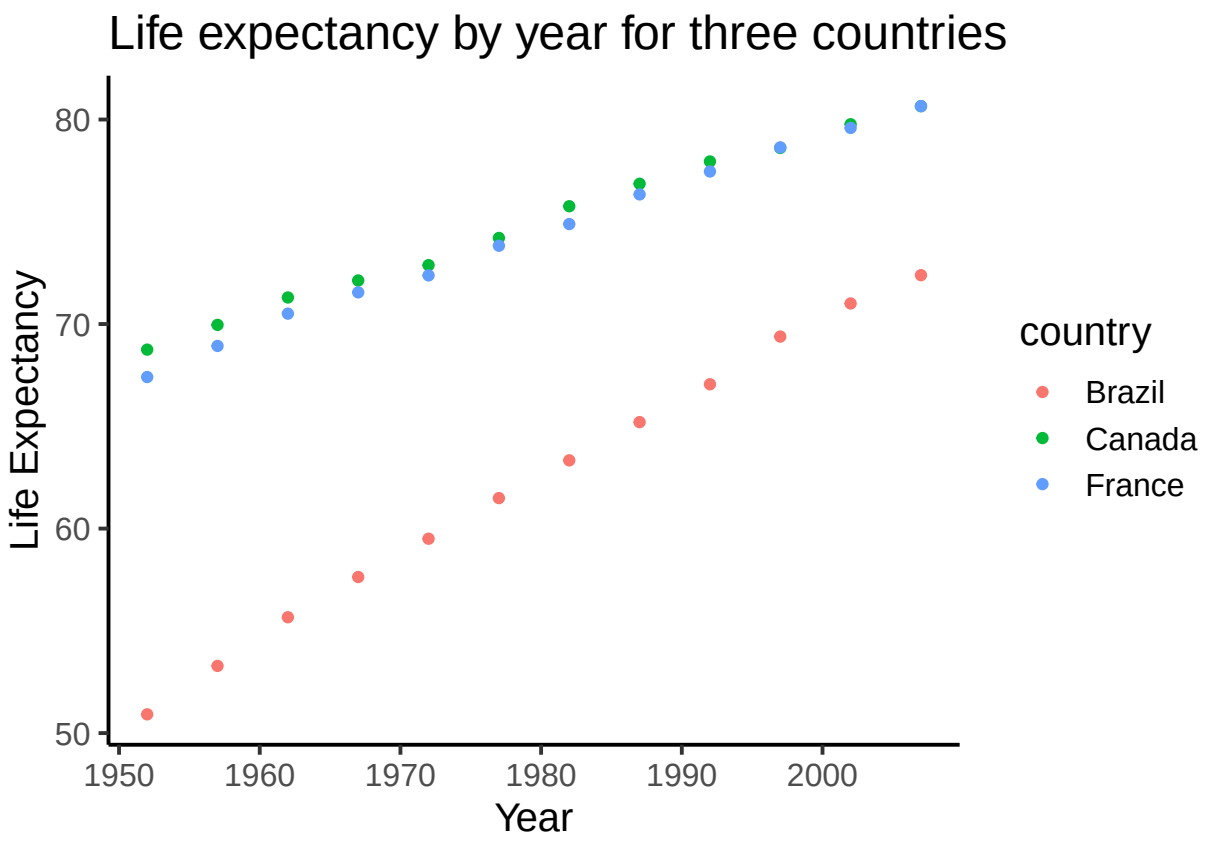
What if we want to look at a few countries altogether. We can do this too. We just change how we filter the data so more than one country is allowed, then we plot the data. We will also add some nicer color options and make the plot look pretty. First, the simple code:

```
# filter rows to contain countries of choice

smaller_df <- gapminder_df %>%
                filter(country %in% c("Canada","France","Brazil") == TRUE)

# plot the new data contained in smaller_df

ggplot(smaller_df, aes(y= lifeExp, x= year, group= country))+
  geom_point()
```



Nice, we can now see three sets of dots, but which are countries do they represent? Let's add a lengend, and make the graph better looking.

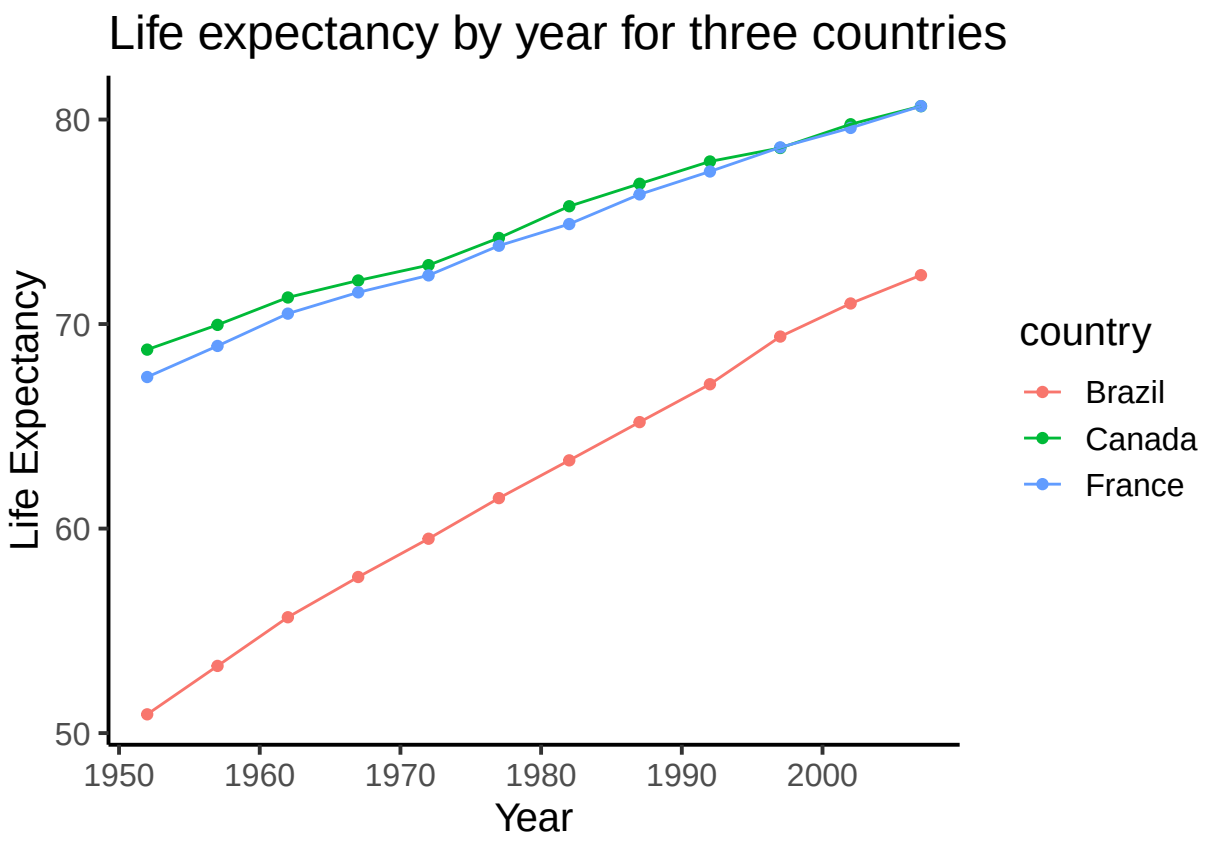
```
ggplot(smaller_df,aes(y= lifeExp, x= year,
                      group= country, color = country)) +
  geom_point()+
  theme_classic(base_size = 15) +
  ylab("Life Expectancy") +
  xlab("Year") +
  ggtitle("Life expectancy by year for three countries")
```



1.2.7.5 geom_line() connecting the dots

We might also want to connect the dots with a line, to make it easier to see the connection! Remember, ggplot2 draws layers on top of layers. So, we add in a new `geom_line()` layer.

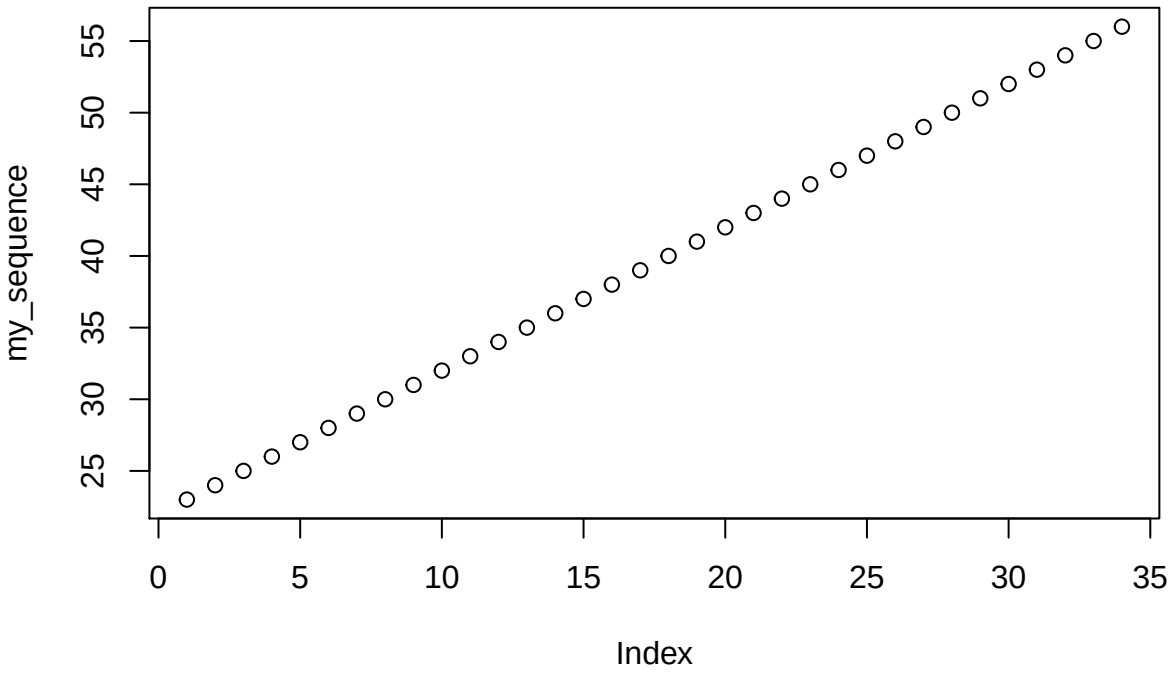
```
ggplot(smaller_df,aes(y= lifeExp, x= year,
                      group= country, color = country)) +
  geom_point()+
  geom_line()+
  theme_classic(base_size = 15) +
  ylab("Life Expectancy") +
  xlab("Year") +
  ggtitle("Life expectancy by year for three countries")
```



Whenever you have a variable with multiple numbers, you can always plot it, just like in the above example. Remember the variable *my_numbers* contains 100 numbers. This means there are 100 slots in the variable. Each slot has an *index* value. The index value for the first slot is 1, the index value for the second slot is 2, and so on. The x-axis (the bottom line in the graph) shows the index value from 1 to 100. Remember also, that each slot contains the number 43. The y-axis (the vertical line in the graph) shows a range of numbers. The dots in the graph represent the value inside each slot of the variable. Because each slot contains the value 43, we see all 100 dots, all in a line, all positioned at 43 with respect to the y-axis.

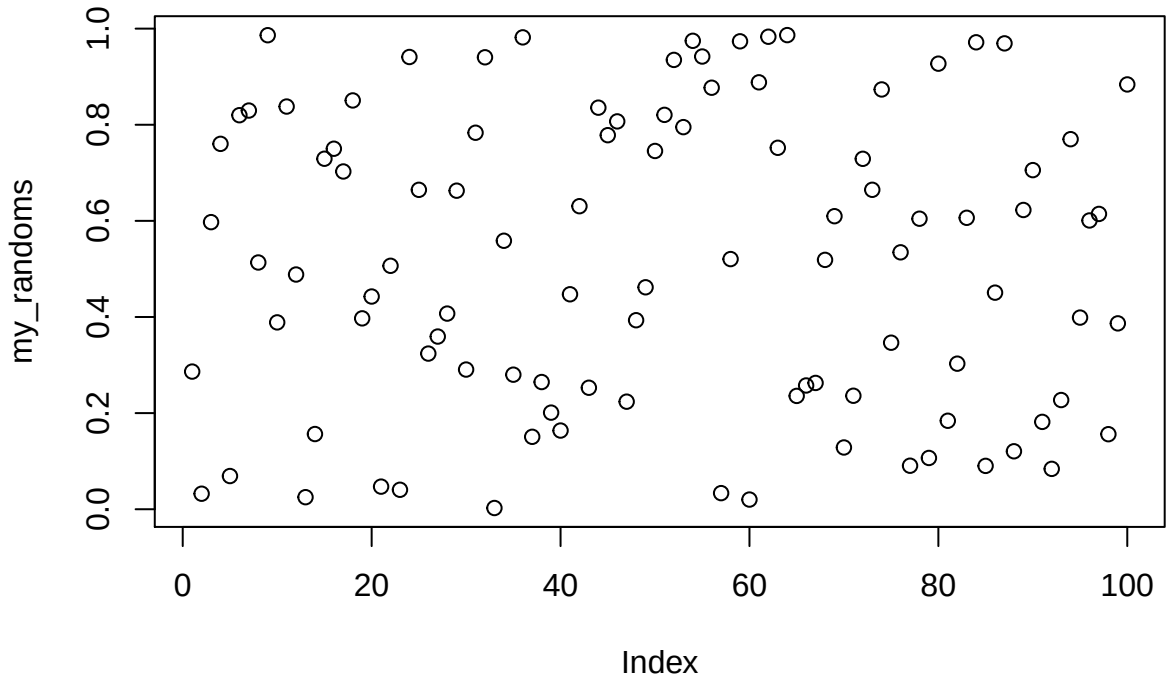
Let's plot some of the other variables we made.

```
plot(my_sequence)
```



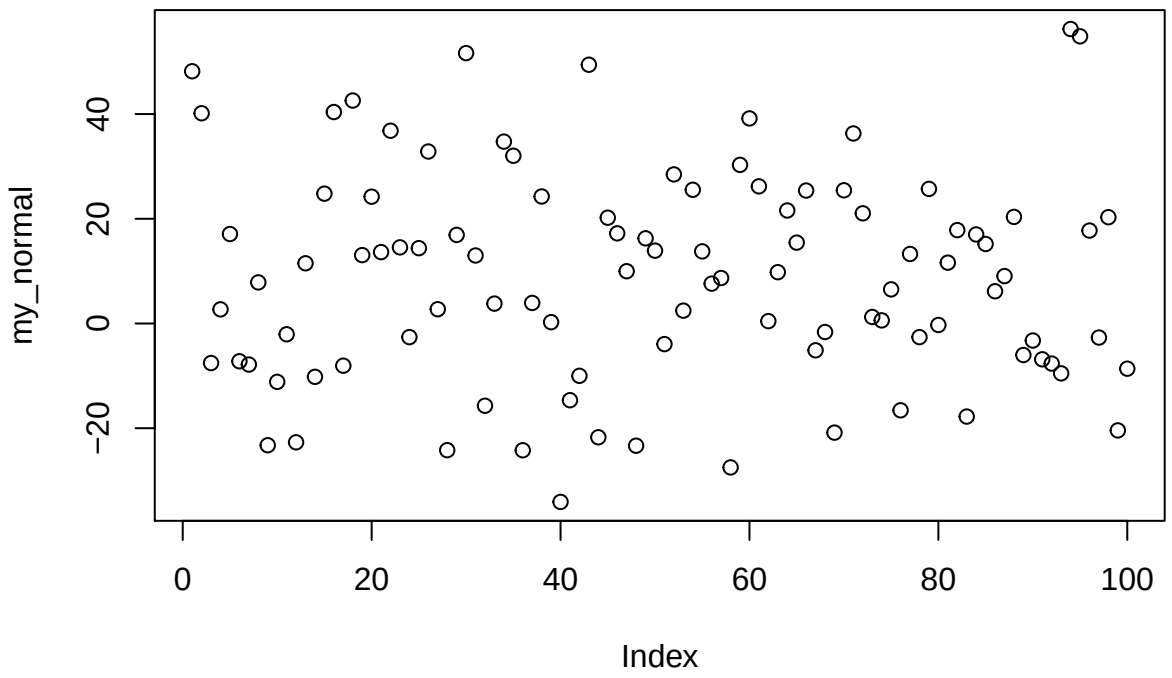
The variable *my_sequence* contains the numbers 23 to 56, going up by one. We see in the plot, the first number (on the x-axis) is a 23 on the y-axis. As we go across the x-axis, the numbers go up by one until we get to 56. We see a straight diagonal line.

```
plot(my_randoms)
```



The variable *my_randoms* contains 100 random numbers between 0 and 1. The plot shows dots all over the place between 0 and 1 on the y-axis.

```
plot(my_normal)
```



The variable *my_normal* contains 100 numbers sample from a normal distribution with a mean of 10, and a standard deviation of 20. Roughly, most of the numbers should be close to 10, some of the numbers will be greater and smaller than 10. But, as the numbers move away from 10 in either direction, really small or really big numbers should occur less and less frequently. We can sort of see this in the plot. For example, you might notice that most the numbers are near the horizontal middle of the graph, near the 10 on the y-axis, and less of the numbers are near the top or bottom of the graph.

1.2.7.6 Histograms

Histograms are used to visually summarize a set of numbers. In particular, histograms split a set of numbers into bins, and then show how many numbers fall within each bin. Each bin represents a pre-defined range.

Let's create a set of numbers made up from 1s, 2s, and 3s. Let's say we have ten 1s, twenty 2s, and 30 3s.

```
my_set <- c(rep(1,10),rep(2,20),rep(3,30))
```

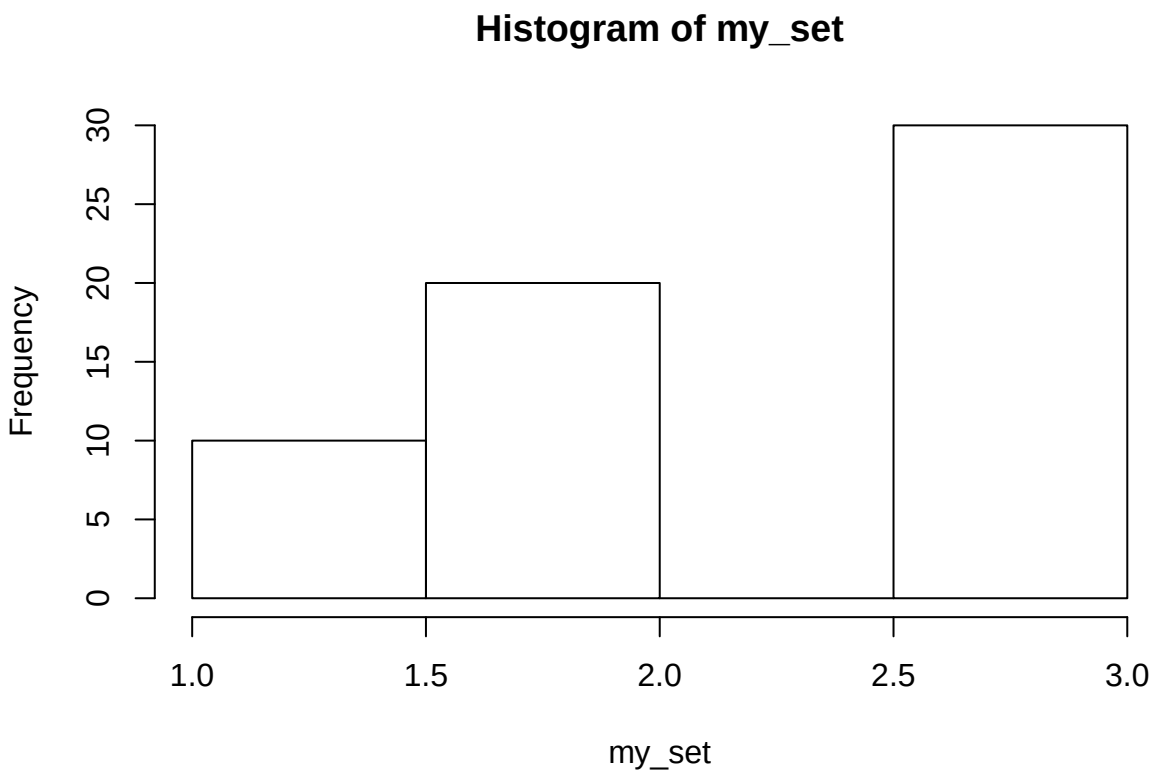
The above line of code uses two R functions, rep(), and c(). We already know how rep works. The c() function is short for combine. So, the above line of code, combines 10 1s, 20 2s and 30 3s, all into one variable. The contents of the variable looks like this:

```
my_set
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
## [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

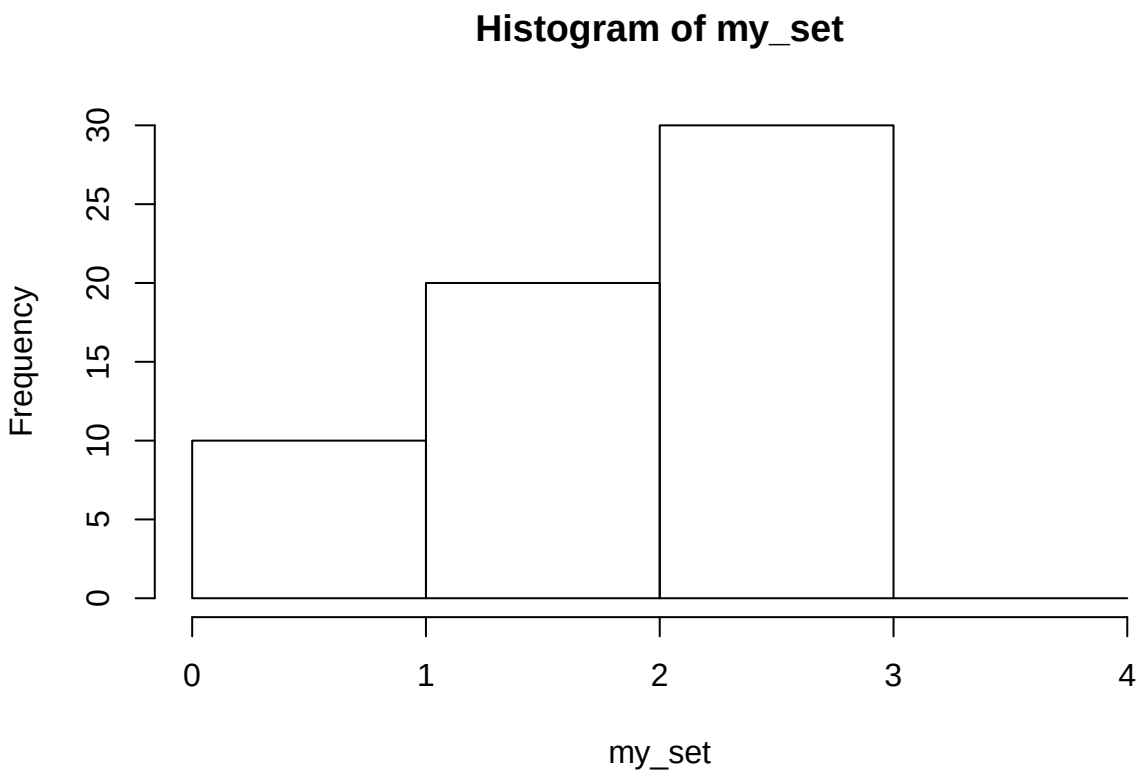
Because, we made this variable, we already know what is inside it. Let's make a histogram of the variable, to see what that looks like:

```
hist(my_set)
```



The histogram is a bar graph. The height of each bar represents a count, or the frequency of how many numbers fall inside each bin. The x-axis shows the bin ranges. If you do not specify the bin ranges, then R will make a reasonable guess for you. In this case, R set the bin ranges in steps of .5. For example, 1-1.5, 1.5-2, 2-2.5, 2.5-3. R uses the word *breaks* to refer to bins. And, when you plot a histogram, you can set your own breaks, or bin ranges.

```
hist(my_set, breaks=c(0,1,2,3,4))
```



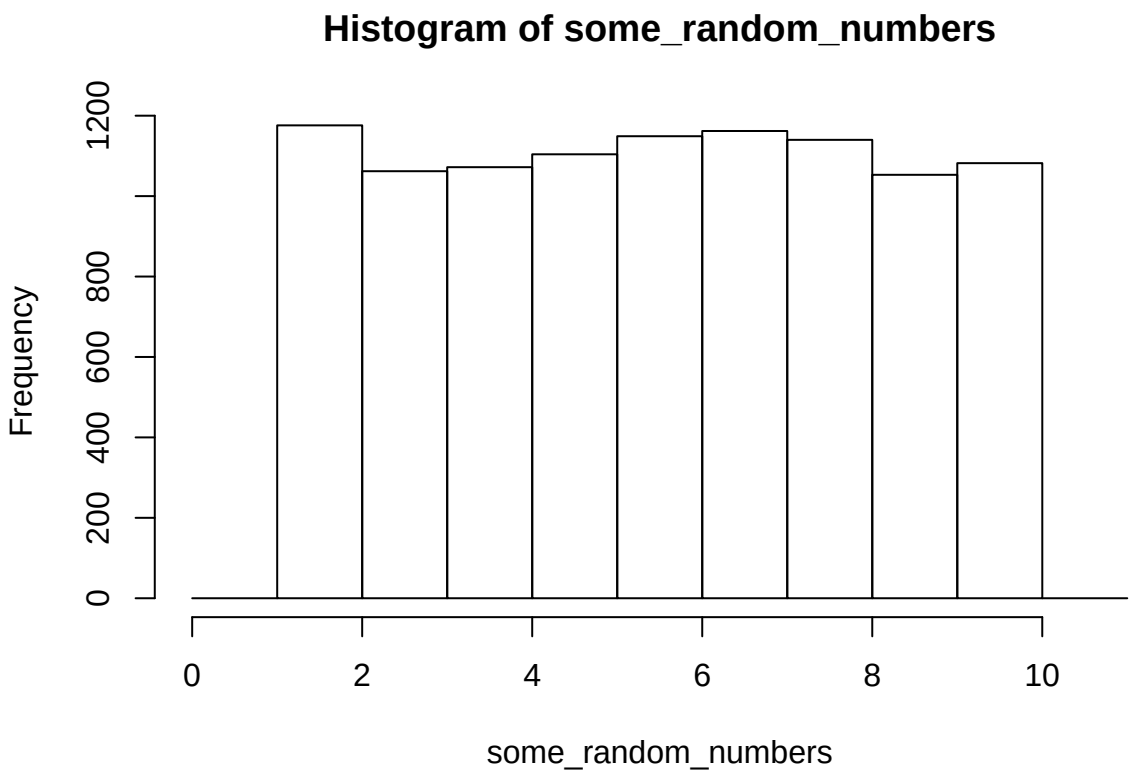
Let's spend a moment interpreting this new histogram. The first bar is between 0 and 1 on the x-axis, and has a value of 10 on the y-axis. This means that there are 10 numbers inside the my_set variable that have a value between 0 and 1; specifically, a value greater than zero up to and equalling 1. The second bar is between 1 and 2 on the x-axis, and has a value of 20 on the y-axis. So, there are 20 numbers in the variable with a value greater than 1 up to and equalling 2. Finally, the third bar shows there are 30 numbers in the range greater than 2, up to equalling 3. We can also see there are no numbers smaller than 0, or greater than 4.

1.2.7.7 What are histograms useful for?

A primary purpose of histograms is to get a quick look at the range and frequency of a set of numbers. In particular, when the bars are of different sizes, we can know that some values occur more than others.

What should a histogram look like for a set of values whose numbers all occur randomly, and equally frequently? By this definition, we are saying that all numbers, within all ranges, occur equally often. For example, imagine we created a set of 10,000 numbers, and chose those numbers from between 1 and 10, such that any number between 1 and 10 occurs with the same frequency as all other numbers. We can use the random number generator and histogram to find out:

```
some_random_numbers <- runif(10000,1,10)
hist(some_random_numbers,breaks=seq(0,11,1))
```



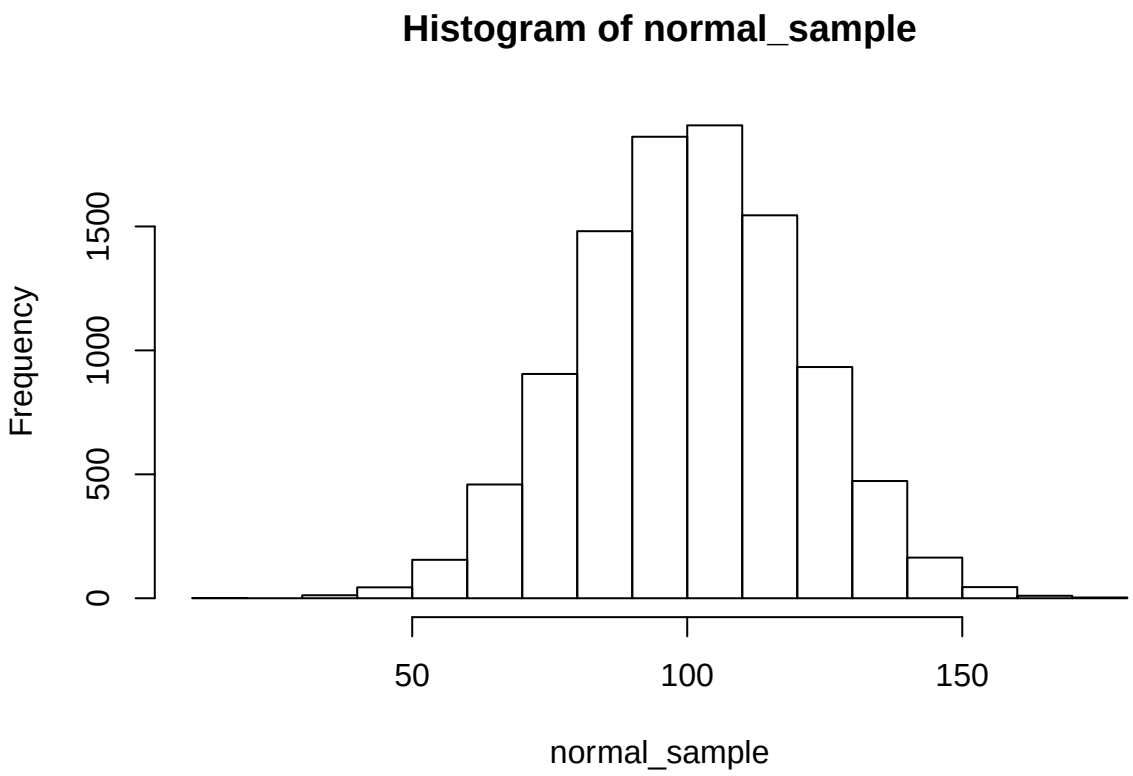

```
#notice I set the breaks using the seq() function
```

We see that heights of the bars are all close to the same number. This is good, because each number between 1 to 10 should have had an equal chance of being selected. However, notice the bars are not exactly the same height. This shows that the random number generator did actually generate each number with equal frequency.

What about sets of numbers where some kinds of numbers occur more than others? Here, we would expect higher bars for ranges containing many values, and smaller numbers for ranges containing fewer values.

Let's plot histogram for a normal distribution and see what it looks like. We will set the mean to 100, and the standard deviation to 20, and we ask R to generate 10000 numbers from this distribution.

```
normal_sample <- rnorm(10000,100,20)
hist(normal_sample)
```



The histogram shows that the highest bars are in the middle, near the 100 mark. So, numbers near 100 occured most frequently in our set. What happens to the heights of the bars on either side of the histogram? The bars are decreasing in height as they move away from the middle in both directions. So, as numbers move away from 100, they occur less and less frequently. For example, we don't see any bars in the range of 500 or 1000. This means that no values that high were in our set of numbers. From the histogram, we can clearly see that our set hardly had any numbers greater than 150, or less than 50. Or, in other words, most of the numbers were between 50 and 150.

–>

1.3 Excel

1.4 SPSS

1.5 JAMOVI

Chapter 2

Lab 2: Descriptive Statistics

Some inspiring quote —Inspiring Person

2.1 Outline of Problem to solve

Stuff we need to say in general

2.1.1 important things

Other things to say

2.2 R

2.2.1 Descriptives basics in R

We learned in lecture and from the textbook that data we want to use ask and answer questions often comes with loads of numbers. Too many numbers to look at all at once. That's one reason we use descriptive statistics. To reduce the big set of numbers to one or two summary numbers that tell use something about all of the numbers. R can produce descriptive statistics for you in many ways. There are base functions for most of the ones that you want. We'll go over some R basics for descriptive statistics, and then use our newfound skills to ask some questions about real data.

2.2.1.1 Making numbers in R

In order to do descriptive statistics we need to put some numbers in a variable. You can also do this using the `c()` command, which stands for combine

```
my_numbers <- c(1,2,3,4)
```

There a few other handy ways to make numbers. We can use `seq()` to make a sequence. Here's making the numbers from 1 to 100

```
one_to_one_hundred <- seq(1,100,1)
```

We can repeat things, using rep. Here's making 10 5s, and 25 1s:

```
rep(10,5)
```

```
## [1] 10 10 10 10 10
```

```
rep(1,25)
```

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

```
all_together_now <- c(rep(10,5),rep(1,25))
```

2.2.1.2 Sum

Let's play with the number 1 to 100. First, let's use the `sum()` function to add them up

```
one_to_one_hundred <- seq(1,100,1)
sum(one_to_one_hundred)
```

```
## [1] 5050
```

2.2.1.3 Length

We put 100 numbers into the variable `one_to_one_hundred`. We know how many numbers there are in there. How can we get R to tell us? We use `length()` for that.

```
length(one_to_one_hundred)
```

```
## [1] 100
```

2.2.2 Central Tendency

2.2.2.1 Mean

Remember the mean of some numbers is their sum, divided by the number of numbers. We can compute the mean like this:

```
sum(one_to_one_hundred)/length(one_to_one_hundred)
```

```
## [1] 50.5
```

Or, we could just use the `mean()` function like this:

```
mean(one_to_one_hundred)
```

```
## [1] 50.5
```

2.2.2.2 Median

The median is the number in the exact middle of the numbers ordered from smallest to largest. If there are an even number of numbers (no number in the middle), then we take the number in between the two (decimal .5). Use the `median` function. There's only 3 numbers here. The middle one is 2, that should be the median

```
median(c(1,2,3))
```

```
## [1] 2
```

2.2.2.3 Mode

R does not a base function for the Mode. You would have to write one for yourself. Here is an example of writing your own mode function, and then using it. Note I searched how to do this on google, and am using the mode defined by this answer on stack overflow

Remember, the mode is the most frequently occuring number in the set. Below 1 occurs the most, so the mode will be one.

```
my_mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

my_mode(c(1,1,1,1,1,1,1,2,3,4))
```

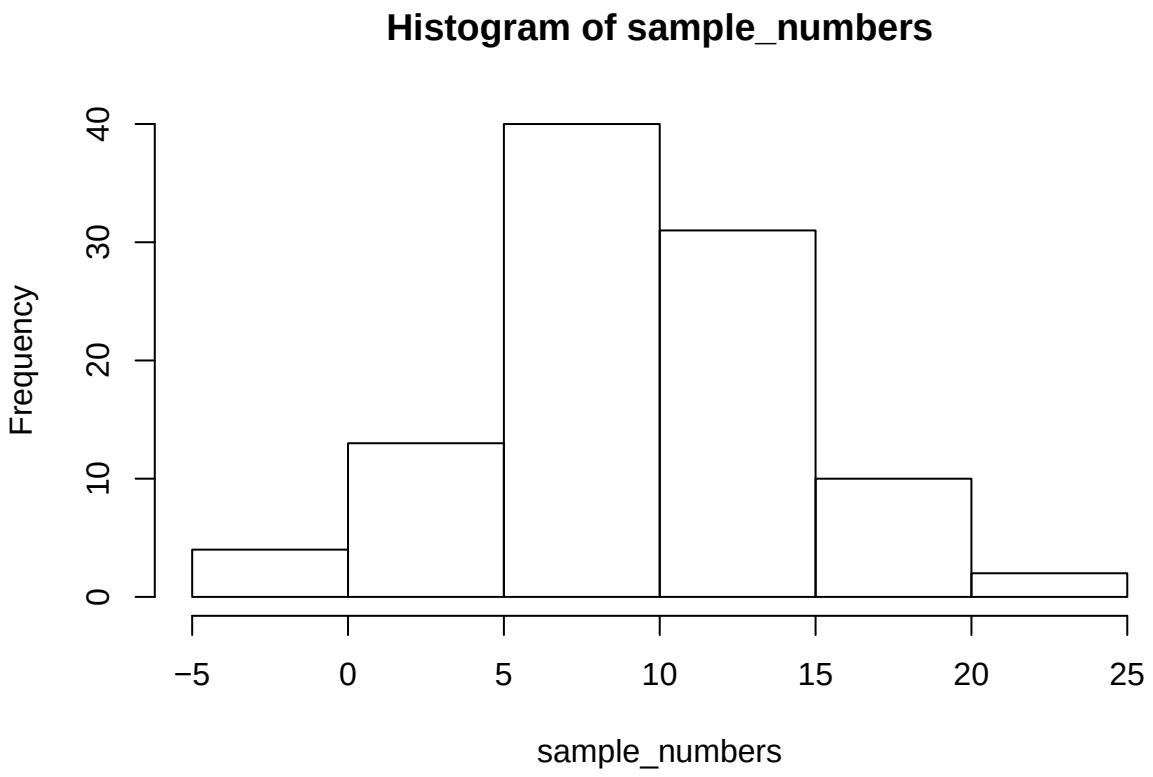
```
## [1] 1
```

2.2.3 Variation

We often want to know how variable the numbers are. We are going to look at descriptive statistics to describe this such as the **range**, **variance**, the **standard deviation**, and a few others.

First, let's remind ourselves what variation looks like (it's when the numbers are different). We will sample 100 numbers from a normal distribution (don't worry about this yet), with a mean of 10, and a standard deviation of 5, and then make a histrogram so we can see the variation around 10..

```
sample_numbers <- rnorm(100,10,5)
hist(sample_numbers)
```



2.2.3.1 range

The range is the minimum and maximum values in the set, we use the `range` function.

```
range(sample_numbers)
```

```
## [1] -3.670097 22.043840
```

2.2.3.2 var = variance

We can find the sample variance using `var`. Note, divides by (n-1)

```
var(sample_numbers)
```

```
## [1] 26.27427
```

2.2.3.3 sd = standard deviation

We find the sample standard deviation us sd. Note, divides by (n-1)

```
sd(sample_numbers)
```

```
## [1] 5.125843
```

Rember that the standard deviation is just the square root of the variance, see:

```
sqrt(var(sample_numbers))
```

```
## [1] 5.125843
```

2.2.3.4 All Descriptives

Let's put all of the descriptives and other functions so far in one place:

```
sample_numbers <- rnorm(100,10,5)

sum(sample_numbers)
```

```
## [1] 988.536
```

```
length(sample_numbers)
```

```
## [1] 100
```

```
mean(sample_numbers)
```

```
## [1] 9.88536
```

```
median(sample_numbers)
```

```
## [1] 10.05067
```

```
my_mode(sample_numbers)
```

```
## [1] 8.865402
```

```
range(sample_numbers)
```

```
## [1] -1.710019 21.689287
```

```
var(sample_numbers)
```

```
## [1] 26.53768
```

```
sd(sample_numbers)
```

```
## [1] 5.151474
```

2.2.4 Descriptives by conditions

Sometimes you will have a single variable with some numbers, and you can use the above functions to find the descriptives for that variable. Other times (most often in this course), you will have a big data frame of numbers, with different numbers in different conditions. You will want to find descriptive statistics for each the sets of numbers inside each of the conditions. Fortunately, this is where R really shines, it does it all for you in one go.

Let's illustrate the problem. Here I make a date frame with 10 numbers in each condition. There are 10 conditions, each labelled, A, B, C, D, E, F, G, H, I, J.

```
scores <- rnorm(100,10,5)
conditions <- rep(c("A","B","C","D","E","F","G","H","I","J"), each =10)
my_df <- data.frame(conditions,scores)
```

If you look at the `my_df` data frame, you will see it has 100 rows, there are 10 rows for each condition with a label in the `conditions` column, and 10 scores for each condition in the `scores` column. What if you wanted to know the mean of the scores in each condition? You would want to find 10 means.

The slow way to do it would be like this:

```
mean(my_df[my_df$conditions=="A",]$scores)
```

```
## [1] 8.284299
```

```
mean(my_df[my_df$conditions=="B",]$scores)
```

```
## [1] 9.071022
```

```
mean(my_df[my_df$conditions=="C",]$scores)
```

```
## [1] 7.426698
```

```
# and then keep going
```

Nobody wants to do that! Not, me I stopped doing it that way, you should to.

2.2.4.1 group_by and summarise

We can easily do everthing all at once using the `group_by` and `summarise` function from the `dplyr` package. Just watch

```
library(dplyr)

my_df %>%
  group_by(conditions) %>%
  summarise(means = mean(scores))
```

```
## # A tibble: 10 x 2
##    conditions means
```

```
##    <fct>      <dbl>
##  1 A           8.28
##  2 B           9.07
##  3 C           7.43
##  4 D           8.63
##  5 E           9.76
##  6 F           5.63
##  7 G           8.31
##  8 H           9.59
##  9 I          12.4
## 10 J          11.9
```

A couple things now. First, the print out of this was ugly. Let's fix that. we put the results of our code into a new variable, then we use `knitr::kable` to print it out nicely when we `knit` the document

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores))

knitr::kable(summary_df)
```

conditions	means
A	8.284299
B	9.071022
C	7.426698
D	8.630906
E	9.759095
F	5.632737
G	8.310846
H	9.590302
I	12.398364
J	11.881236

2.2.4.2 multiple descriptives

The best thing about the `dplyr` method, is that we can add more than one function, and we'll get more than one summary returned, all in a nice format, let's add the standard deviation:

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores),
                         sds = sd(scores))

knitr::kable(summary_df)
```

conditions	means	sds
A	8.284299	5.718872
B	9.071022	7.346908
C	7.426698	5.403125
D	8.630906	4.240390
E	9.759095	3.664083
F	5.632737	5.414865
G	8.310846	5.754972
H	9.590302	4.911016
I	12.398364	3.911585
J	11.881236	4.591293

We'll add the min and the max too:

```
summary_df <- my_df %>%
               group_by(conditions) %>%
               summarise(means = mean(scores),
                         sds = sd(scores),
                         min = min(scores),
                         max = max(scores))

knitr::kable(summary_df)
```

conditions	means	sds	min	max
A	8.284299	5.718872	-0.9643942	18.47722
B	9.071022	7.346908	-2.5467949	19.96221
C	7.426698	5.403125	-1.9385197	16.58471
D	8.630906	4.240390	2.7670265	16.26315
E	9.759095	3.664083	2.6638108	17.19651
F	5.632737	5.414865	-3.2441815	14.16092
G	8.310846	5.754972	-0.6914890	20.09894
H	9.590302	4.911016	1.8352048	18.73463
I	12.398364	3.911585	8.6547700	19.84778
J	11.881236	4.591293	5.9818904	21.35033

2.2.5 Describing gapminder

Now that we know how to get descriptive statistics from R, we cam do this will some real data. Let's quickly ask a few question about the gapminder data. Reinstall the gapminder library if you don't already have it, then laod the data into a dataframe:

```
library(gapminder)
gapminder_df <- gapminder
```

2.2.5.1 What are some descriptive for Life expectancy by continent?

Copy the code from the last part of descriptives using dplyr, then change the names like this:

```
summary_df <- gapminder_df %>%
               group_by(continent) %>%
               summarise(means = mean(lifeExp),
                         sds = sd(lifeExp),
                         min = min(lifeExp),
                         max = max(lifeExp))
```

```
knitr::kable(summary_df)
```

continent	means	sds	min	max
Africa	48.86533	9.150210	23.599	76.442
Americas	64.65874	9.345088	37.579	80.653
Asia	60.06490	11.864532	28.801	82.603
Europe	71.90369	5.433178	43.585	81.757
Oceania	74.32621	3.795611	69.120	81.235

2.2.6 Generalization Exercise

Complete the generalization exercise described in your R Markdown document for this lab.

2.2.7 Writing asignment

Complete the writing assignment described in your R Markdown document for this lab. When you have finished everything. Knit the document and hand in your stuff (you can submit your .RMD file to blackboard if it does not knit.)

2.3 Excel

How to do it in Excel

2.4 SPSS

How to do it in SPSS

2.5 JAMOVI

How to do it in JAMOVI

Chapter 3

Lab 3: Correlation

Some inspiring quote —Inspiring Person

3.1 Outline of Problem to solve

Stuff we need to say in general

3.1.1 important things

Other things to say

3.2 R

In this lab we use explore to explore correlations between any two variables, and alsot show how to do a regression line. There will be three main parts. Getting R to compute the correlation, and looking at the data using scatterplots. We'll look at some correlations from the World Happiness Report. Then you'll look at correlations using data we collect from ourselves. It will be fun.

3.2.1 cor for correlation

R has the `cor` function for computing pearson's r between any two variables. In fact this same function computes other versions of correlation, but we'll skip those here. To use the function you just need two variables with numbers in them like this:

```
x  <- c(1,3,2,5,4,6,5,8,9)
y  <- c(6,5,8,7,9,7,8,10,13)
cor(x,y)
```

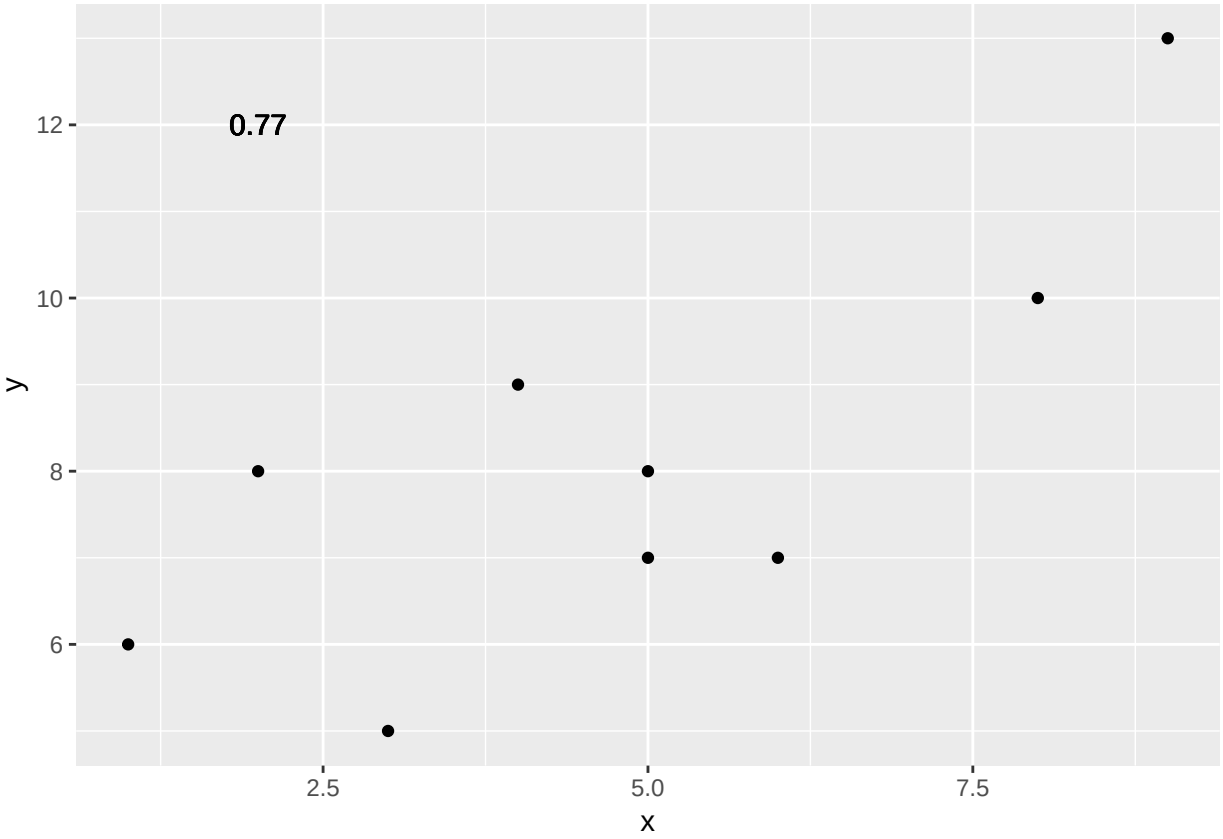
```
## [1] 0.76539
```

Well, that was easy.

3.2.1.1 scatterplots

Let's take our silly example, and plot the data in a scatterplot using ggplot2, and let's also return the correlation and print it on the scatterplot. Remember, ggplot2 wants the data in a data.frame, so we first put our x and y variables in a data frame.

```
library(ggplot2)

# create data frame for plotting
my_df <- data.frame(x,y)

# plot it
ggplot(my_df, aes(x=x,y=y))+
  geom_point()+
  geom_text(aes(label = round(cor(x,y), digits=2), y=12, x=2 ))
```



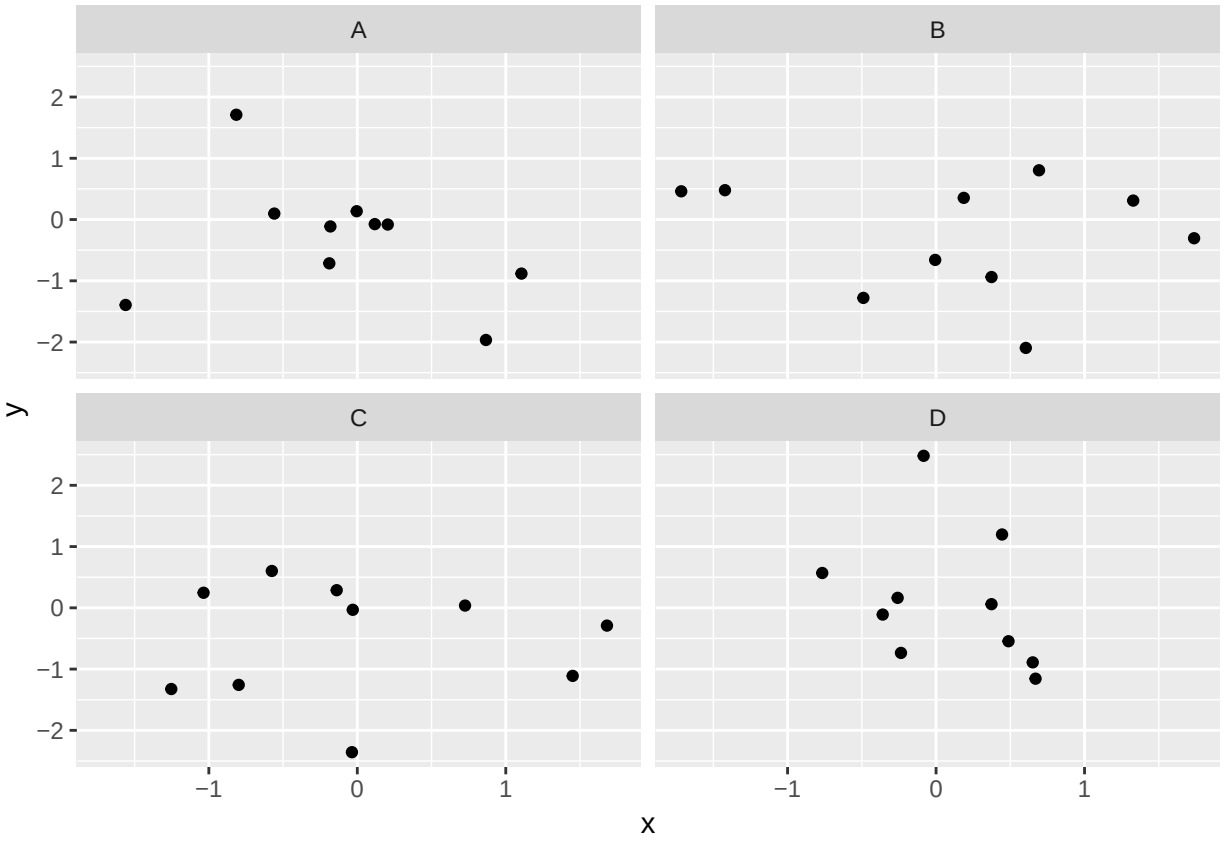
Wow, we're moving fast here.

3.2.1.2 lots of scatterplots

Before we move on to real data, let's look at some fake data first. Often we will have many measures of X and Y, split between a few different conditions, for example, A, B, C, and D. Let's make some fake data for X and Y, for each condition A, B, C, and D, and then use facet_wrapping to look at four scatterplots all at once

```
x<-rnorm(40,0,1)
y<-rnorm(40,0,1)
conditions<-rep(c("A","B","C","D"), each=10)
```

```
all_df <- data.frame(conditions, x, y)

ggplot(all_df, aes(x=x,y=y))+
  geom_point()+
  facet_wrap(~conditions)
```



3.2.1.3 computing the correlations all at once

We've seen how we can make four graphs at once. Facet_wrap will always try to make as many graphs as there are individual conditions in the column variable. In this case there are four, so it makes four.

Notice, the scatterplots don't show the correlation (r) values. Getting these numbers on there is possible, but we have to calculate them first. We'll leave it to you to google how to do this, if it's something you want to do. Instead, what we will do is make a table of the correlations in addition to the scatterplot. We again use `dplyr` to do this:

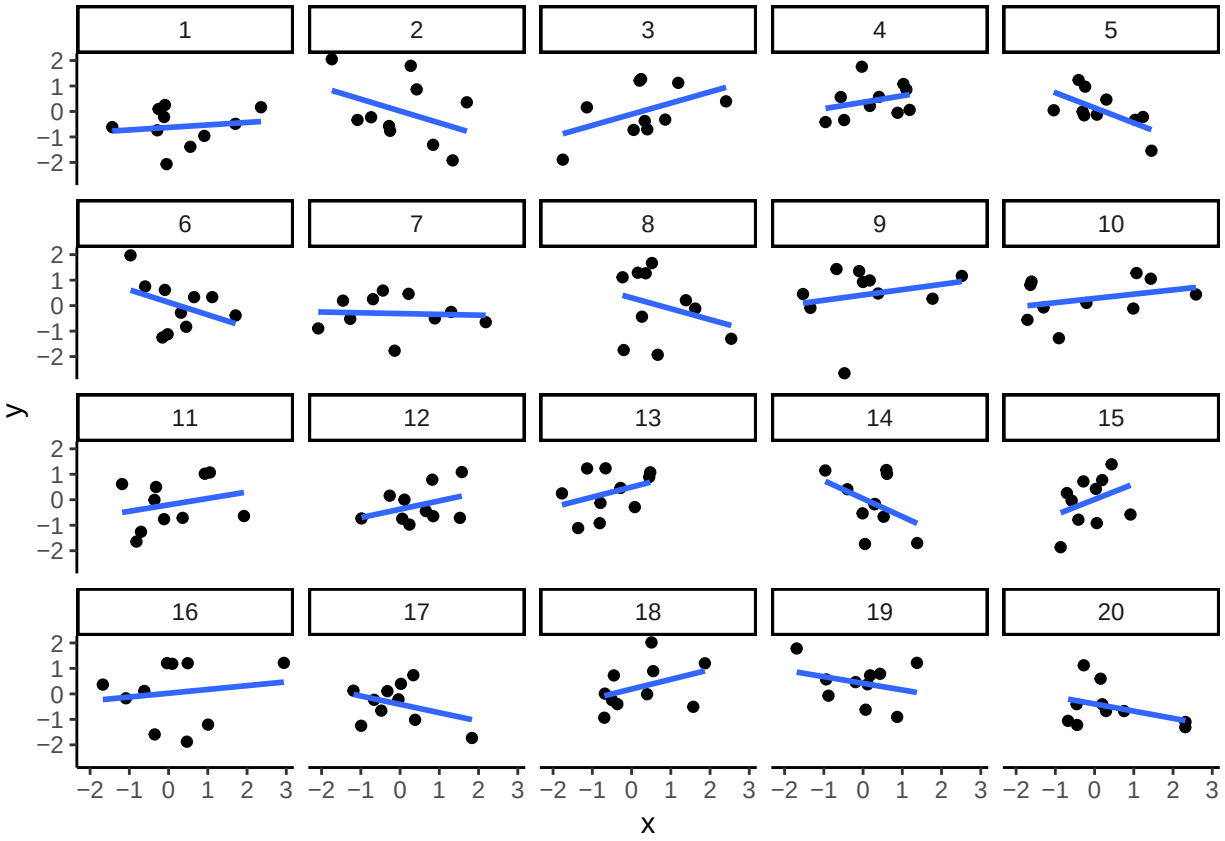
Ok, we are basically ready to turn to some real data and ask if there are correlations between interesting variables…You will find that there are some… But before we do that, we do one more thing. This will help you become a little bit more skeptical of these "correlations".

3.2.1.4 Chance correlations

As you learned from the textbook. We can find correlations by chance alone, even when there is no true correlation between the variables. For example, if we sampled randomly into x, and then sampled some numbers randomly into y. We know they aren't related, because we randomly sampled the numbers. However, doing this creates some correlations some of the time just by chance. You can demonstrate this to yourself

with the following code. It's a repeat of what we already saw, jut with a few more conditions added. Let's look at 20 conditions, with random numbers for x and y in each. For each, sample size will be 10. We'll make the fake data, then make a big graph to look at all. And, even though we get to regression later in the lab, I'll put the best fit line onto each scatterplot, so you can "see the correlations".

```
x<-rnorm(10*20,0,1)
y<-rnorm(10*20,0,1)
conditions<-rep(1:20, each=10)

all_df <- data.frame(conditions, x, y)

ggplot(all_df, aes(x=x,y=y))+
  geom_point()+
  geom_smooth(method=lm, se=FALSE)+
  facet_wrap(~conditions)+
  theme_classic()
```



You can see that the slope of the blue line is not always flat. Sometimes it looks like there is a correlation, when we know there shouldn't be. You can keep re-doing this graph, by reknitting your R Markdown document, or by pressing the little green play button. This is basically you simulating the outcomes as many times as you press the button.

The point is, now you know you can find correlations by chance. So, in the next section, you should always wonder if the correlations you find reflect meaningful assocation between the x and y variable, or could have just occured by chance.

3.2.2 Humor Styles Questionnaire

Let's take a look at some correlations in real data. We are going to look at responses to a questionnaire about happiness that was sent around the world, from the world happiness report

3.2.2.1 Load the data

we load the data into a dataframe.

```
library(data.table)
whr_data <- fread('data/WHR2018.csv')
```

3.2.2.2 Look at the data

```
library(summarytools)
view(dfSummary(whr_data))
```

You should be able to see that there is data for many different countries, across a few different years. There are lots of different kinds of measures, and each are given a name. I'll show you some examples of asking questions about correlations with this data, then you get to ask and answer your own questions.

3.2.2.3 My Question #1

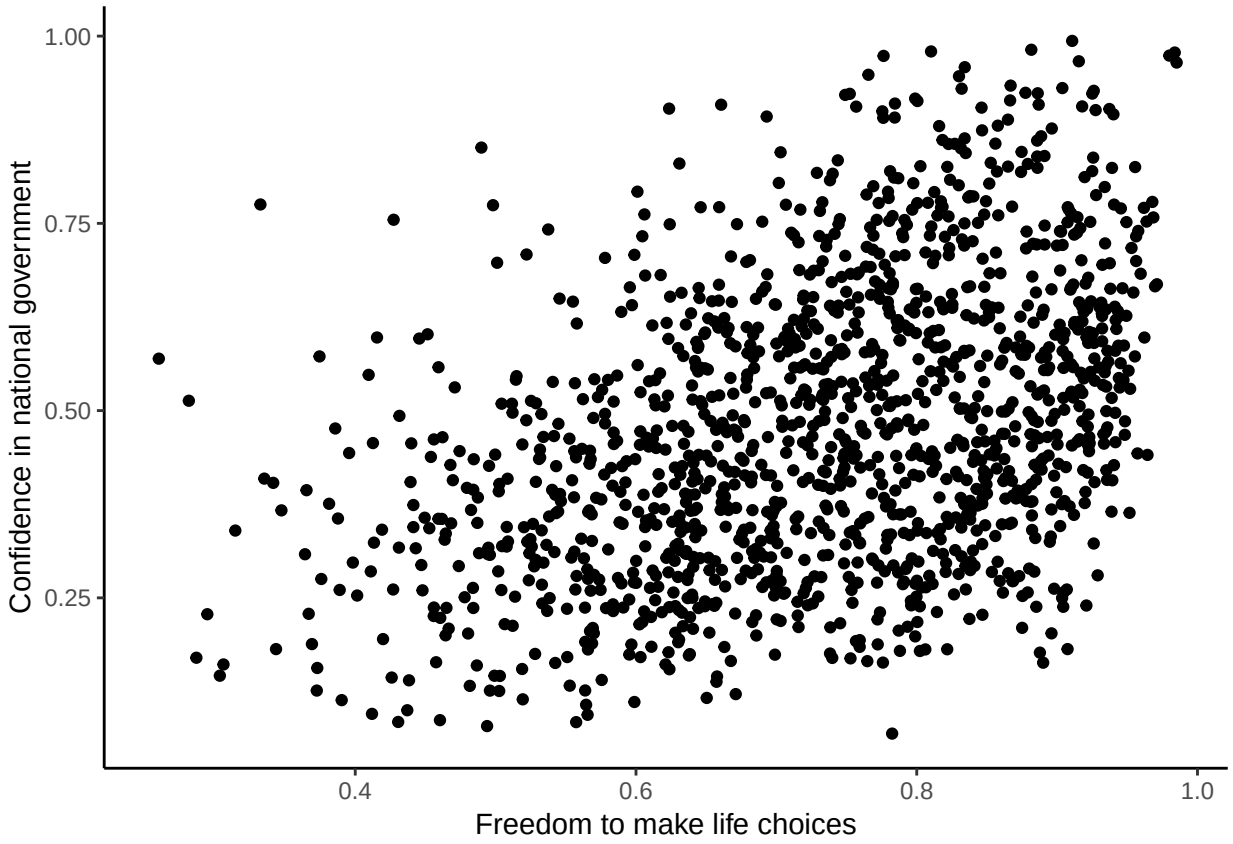
For the year 2017 only, does a countries measure for "freedom to make life choices" correlate with that countries measure for " Confidence in national government"?

Let's find out. We calculate the correlation, and then we make the scatterplot.

```
cor(whr_data$`Freedom to make life choices`,
    whr_data$`Confidence in national government`)
```

```
## [1] NA
```

```
ggplot(whr_data, aes(x=`Freedom to make life choices`,
                     y=`Confidence in national government`))+
  geom_point()+
  theme_classic()
```



Interesting, what happened here? We can see some dots, but the correlation was NA (meaning undefined). This occured because there are some missing data points in the data. We can remove all the rows with missing data first, then do the correlation. We will do this a couple steps, first creating our own data.frame with only the numbers we want to analyse. We can select the columns we want to keep using `select`. Then we use `filter` to remove the rows with NAs.

```
library(dplyr)

smaller_df <- whr_data %>%
               select(country,
                      `Freedom to make life choices`,
                      `Confidence in national government`) %>%
               filter(!is.na(`Freedom to make life choices`),
                      !is.na(`Confidence in national government`))

cor(smaller_df$`Freedom to make life choices`,
    smaller_df$`Confidence in national government`)
```

```
## [1] 0.4080963
```

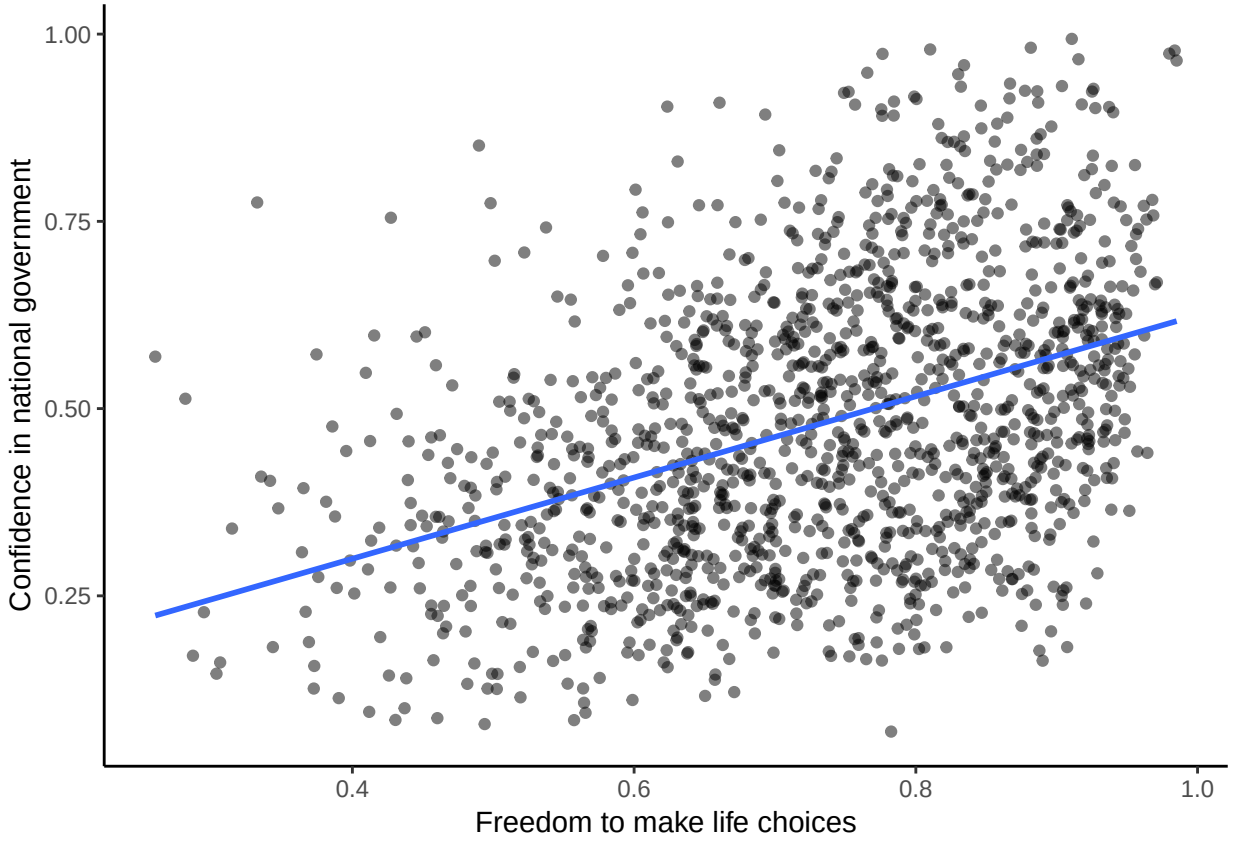
Now we see the correlation is .408.

Although the scatterplot shows the dots are everywhere, it generally shows that as Freedom to make life-choices increases in a country, that countries confidence in their national government also increase. This is a positive correlation. Let's do this again and add the best fit line, so the trend is more clear, we use `geom_smooth(method=lm, se=FALSE)`. I also change the `alpha` value of the dots so they blend it bit, and you can see more of them.

```
# select DVs and filter for NAs

smaller_df <- whr_data %>%
               select(country,
                      `Freedom to make life choices`,
                      `Confidence in national government`) %>%
               filter(!is.na(`Freedom to make life choices`),
                      !is.na(`Confidence in national government`))

# calcualte correlation

cor(smaller_df$`Freedom to make life choices`,
    smaller_df$`Confidence in national government`)
```

```
## [1] 0.4080963
```

```
# plot the data with best fit line

ggplot(smaller_df, aes(x=`Freedom to make life choices`,
                       y=`Confidence in national government`))+
  geom_point(alpha=.5)+
  geom_smooth(method=lm, se=FALSE)+
  theme_classic()
```



3.2.2.4 My Question #2

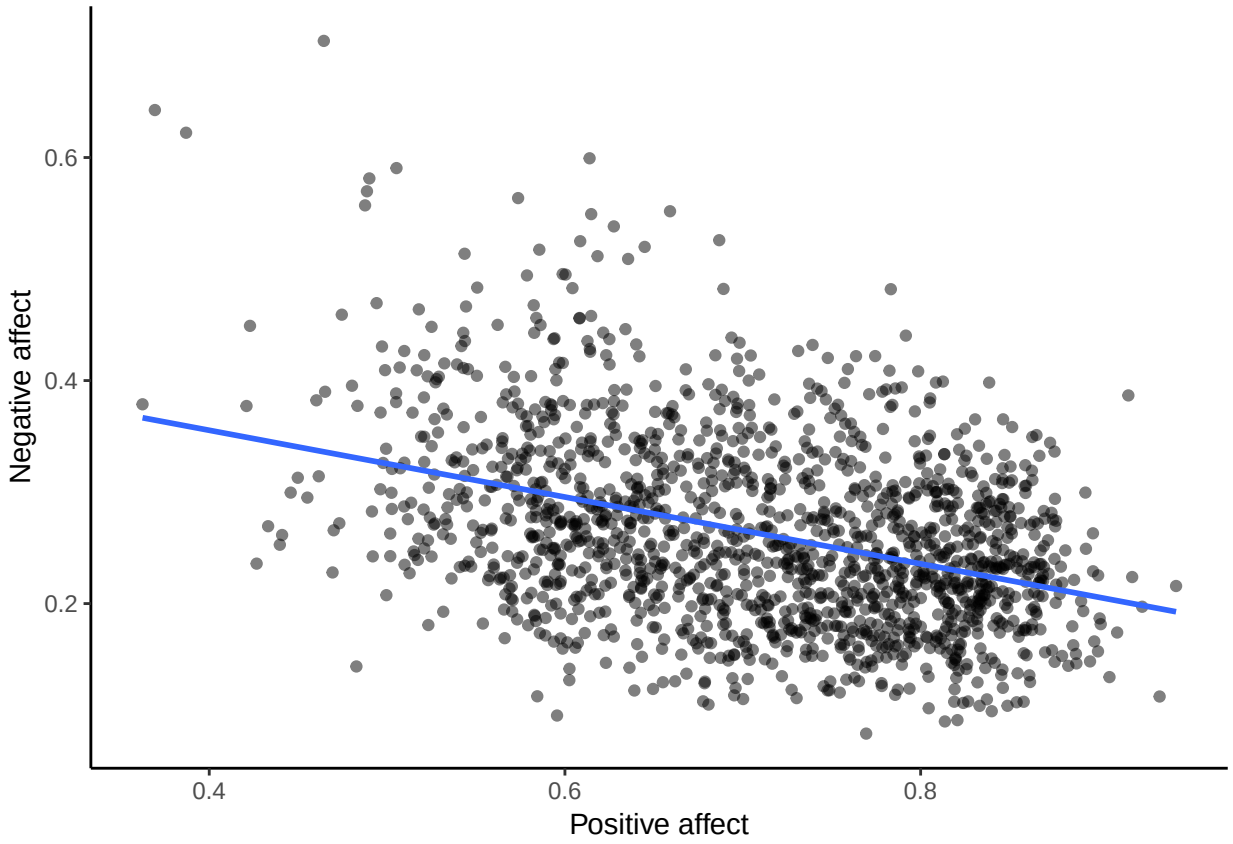
After all that work, we can now speedily answer more questions. For example, what is the relationship between positive affect in a country and negative affect in a country. I wouldn't be surprised if there was a negative correlation here: when positive feelings generally go up, shouldn't negative feelings generally go down?

To answer this question, we just copy paste the last code block, and change the DVs to be `Positive affect`, nad `Negative affect`

```
# select DVs and filter for NAs

smaller_df <- whr_data %>%
               select(country,
                      `Positive affect`,
                      `Negative affect`) %>%
               filter(!is.na(`Positive affect`),
                      !is.na(`Negative affect`))

# calcualte correlation

cor(smaller_df$`Positive affect`,
    smaller_df$`Negative affect`)
```

```
## [1] -0.3841123
```

```
# plot the data with best fit line

ggplot(smaller_df, aes(x=`Positive affect`,
                       y=`Negative affect`))+
  geom_point(alpha=.5)+
  geom_smooth(method=lm, se=FALSE)+
  theme_classic()
```



Bam, there we have it. As positive affect goes up, negative affect goes down. A negative correlation.

3.2.3 Generalization Exercise

Complete the generalization exercise described in your R Markdown document for this lab.

3.2.4 Writing asignment

Complete the writing assignment described in your R Markdown document for this lab. When you have finished everything. Knit the document and hand in your stuff (you can submit your .RMD file to blackboard if it does not knit.)

3.3 Excel

How to do it in Excel

3.4 SPSS

How to do it in SPSS

3.5 JAMOVI

How to do it in JAMOVI

Chapter 4

Lab 4: Normal Distribution & Central Limit Theorem

Some inspiring quote —Inspiring Person

4.1 Outline of Problem to solve

1. Distributions
2. Sampling from distributions
3. Sampling distribution of the mean
4. Sampling statistics (statistics of many samples)
5. Central limit theorem
6. Normal Distribution
7. z-scores

4.1.1 important things

Other things to say

4.2 R

This is one of two special labs where we don't use too much real data. We will mostly fake everything. Yes, you will learn how to fake data in this course. Be a superhero, and only use these skills for good and not for evil.

As we progress through the course, you will learn how generating simulated data can be very useful to help you understand real data. In fact, I will say this right now. If you can't simulate the data you expect to find, then you probably can't understand the data that you do find very well. That's a bold statement. It's probably partly true.

4.2.1 Generating Numbers in R

There are many ways to make R generate numbers for you. In all of the case you get define how the numbers are generated. We'll go through a few of the many ways.

4.2.1.1 sample

The sample function is like an endless gumball machine. You put the gumballs inside with different properties, say As and Bs, and then you let sample endlessly take gumballs out. Check it out:

```
gumballs <- c("A","B")
sample_of_gumballs <-sample(gumballs, 10, replace=TRUE)
sample_of_gumballs
```

```
##  [1] "B" "A" "A" "A" "B" "A" "B" "A" "B" "B"
```

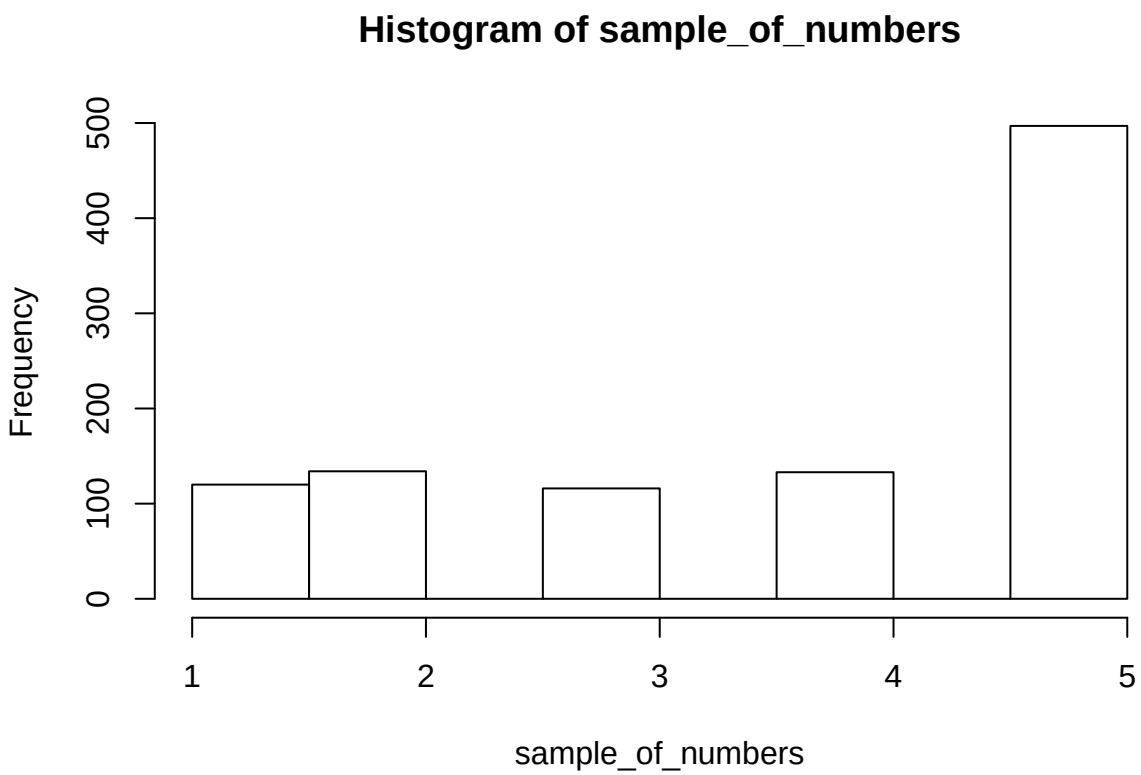
Here the sample function randomly picks A or B each time. We set it do this 10 times, so our sample has 10 things in it. We set `replace=TRUE` so that after each sample, we put the item back into the gumball machine and start again. Here's another example with numbers

```
some_numbers <- c(1,2,3,4,5,5,5,5)
sample_of_numbers <-sample(some_numbers, 20, replace=TRUE)
sample_of_numbers
```

```
##  [1] 5 5 2 5 4 5 3 5 5 5 2 2 1 5 5 5 5 5 4 5
```

Let's do one more thing with sample. Let's sample 1000 times from our `some_numbers` variable, and then look at the histogram

```
library(ggplot2)
some_numbers <- c(1,2,3,4,5,5,5,5)
sample_of_numbers <-sample(some_numbers, 1000, replace=TRUE)
hist(sample_of_numbers)
```

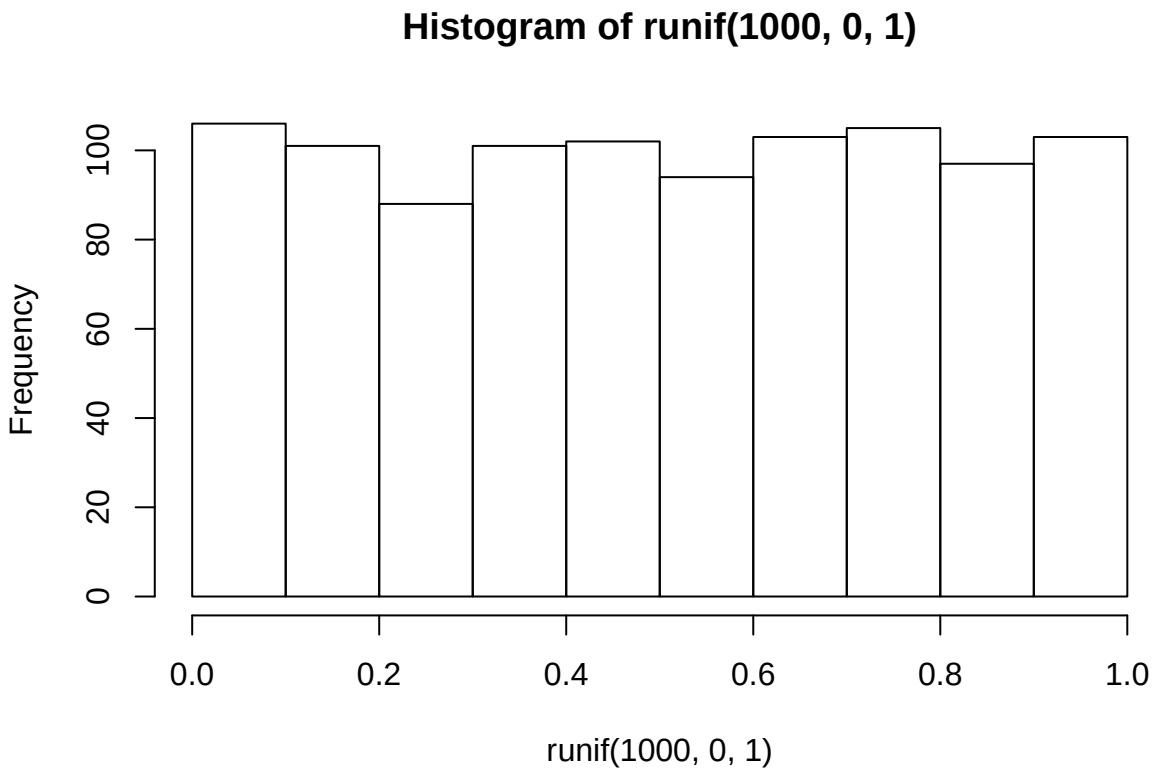


We are looking at lots of samples from our little gumball machine of numbers. We put more 5s in, and voila, more 5s come out of in our big sample of 1000.

4.2.1.2 runif uniform distribution

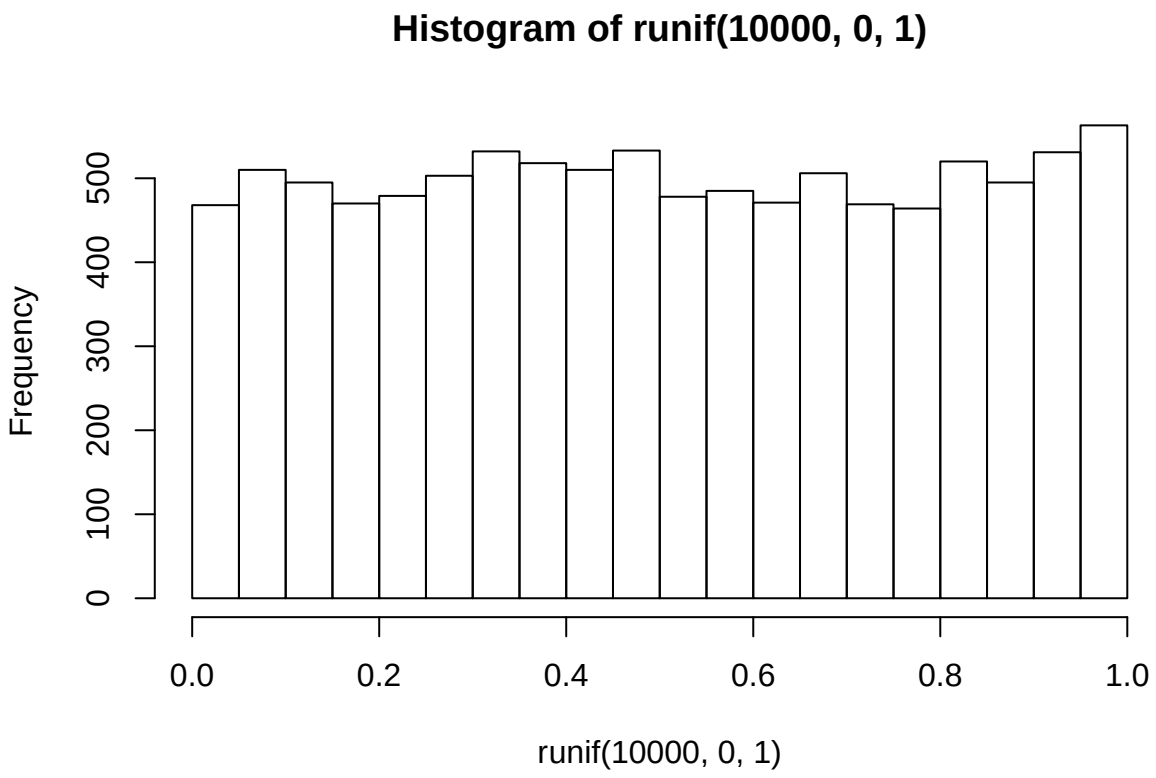
We can sample random numbers between any range using the `runif(n, min=0, max = 1)` function for the uniform distribution. We discussed this in the textbook. A uniform distribution is flat, and all the numbers between the min and max should occur roughly equally frequently. Let's take 1000 random numbers between 0 and 1 and plot the histgram. We'll just do it all in one line for speed.

```
hist(runif(1000,0,1))
```



This is histogram is flattish. Not perfectly flat, after all we only took 1000 samples. What if we took many more, say 10,000 total samples? Now it looks more flat, each bin is occuring about 500 times each, which is pretty close to the same amount.

```
hist(runif(10000,0,1))
```



4.2.1.3 rbinom the binomial distribution

The binomial distribution sounds like a scary word... binomial (AAGGGGHHHHH, stay away!). The binomial can be a coin flipping distribution. You use `rbinom(n, size, prob)`. `n` gives the number of samples you want to take. We'll keep `size = 1` for now, it's the number of trials (forget this for now, it's more useful for more complicated things than what we are doing, if you want to know what it does, try it out, and see if you figure it out). `prob` is a little list you make of probabilities, that define how often certain things happen.

For example, consider flipping a coin. It will be heads or tails, and the coin, if it is fair, should have a 50% chance of being heads or tails. Here's how we flip a coin 10 times using rbinom

```
coin_flips <- rbinom(10,1,.5)
coin_flips
```

```
##  [1] 0 0 0 1 0 0 1 1 1 0
```

We get a bunch of 0s, and 1s. We can pretend 0 = tails, and 1 = heads. Great, now we can do coin flipping if we want. For example, if you flip 10 coins, how many heads do you get? We can can do the above again, and then `sum(coin_flips)`. All the 1s are heads, so it will work out.

```
coin_flips <- rbinom(10,1,.5)
sum(coin_flips)
```

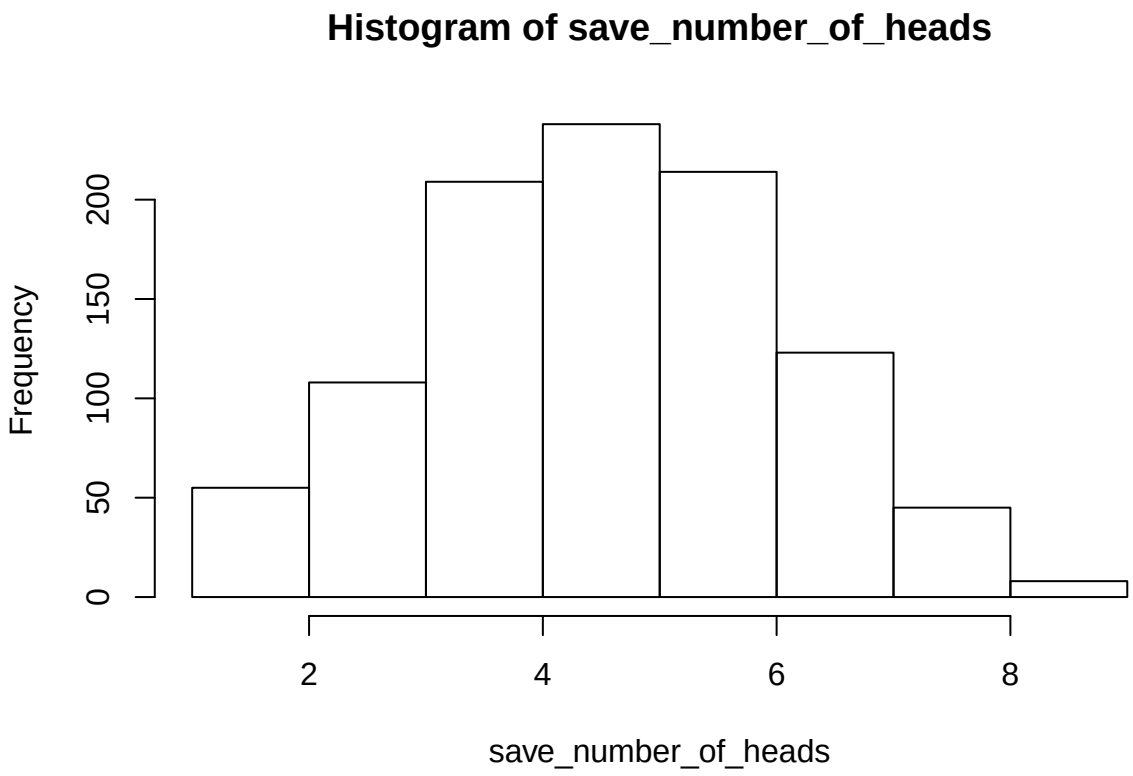
```
## [1] 6
```

Alright, so we get the sum, which tells us the number of heads. But, should we alwyas get that number of heads if we flipped a coin 10 times? If you keep redoing the above, you'll get different answers. 5 heads will be the most frequent answer, but you will get lots of other answers too.

Hold on to your seats for this next one. With R, we can simulate the flipping of a coin 10 times (you already know that, you just did it), and we can do that over and over as many times as we want. For example, we could do it 100 times over, saving the number of heads for each set of 10 flips. Then we could look at the

distribution of those sums. That would tell us about the range of things that can happen when we flip a coin 10 times. We can do that in loop like this:

```
save_number_of_heads<-length(1000) # make an empty variable to save things in

for(i in 1:1000){
  save_number_of_heads[i] <- sum(rbinom(10,1,.5))
}

hist(save_number_of_heads)
```

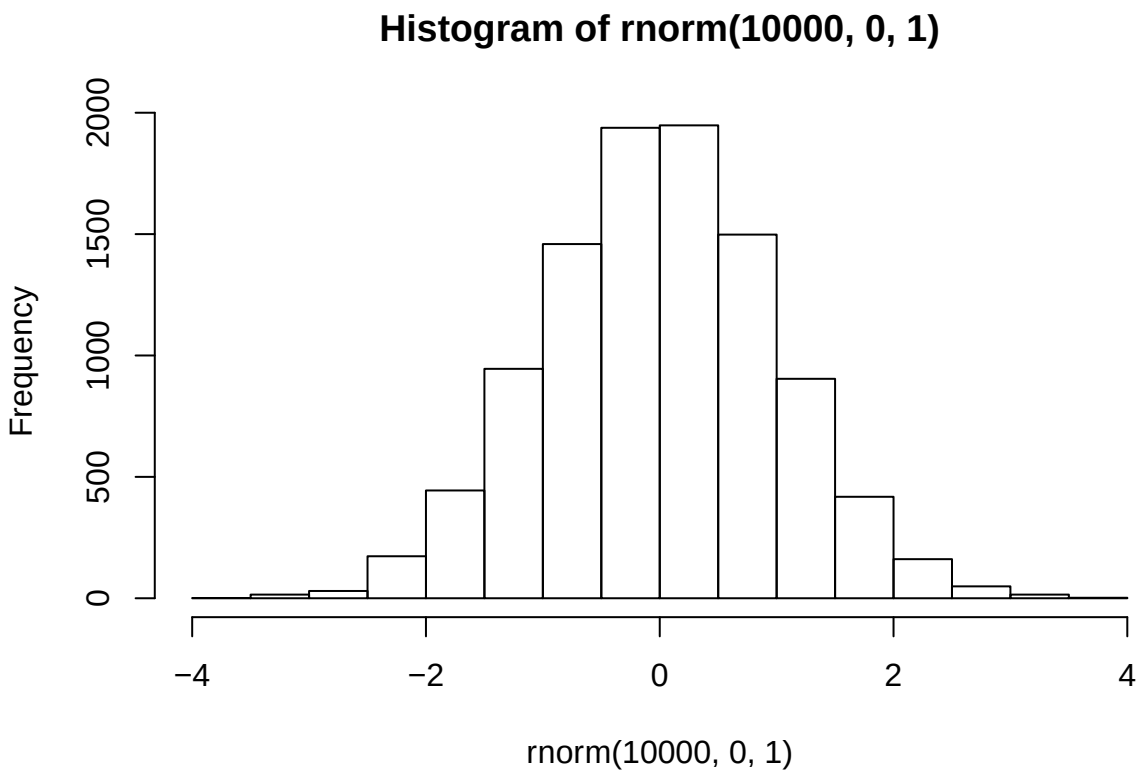


See, that wasn't too painful. Now we see another histrogram. The histogram shows us the frequency observing different numbers of heads (for 10 flips) across the 1000 simulations. 5 happens the most, but 2 happens sometimes, and so does 8. All of the possibilities seem to happen sometimes, some more than others.

4.2.1.4 rnorm the normal distribution

We'll quickly show how to use `rnorm(n, mean=0, sd=1)` to sample numbers from a normal distribution. And, then we'll come back to the normal distribution later, because it is so important.

```
hist(rnorm(10000,0,1))
```

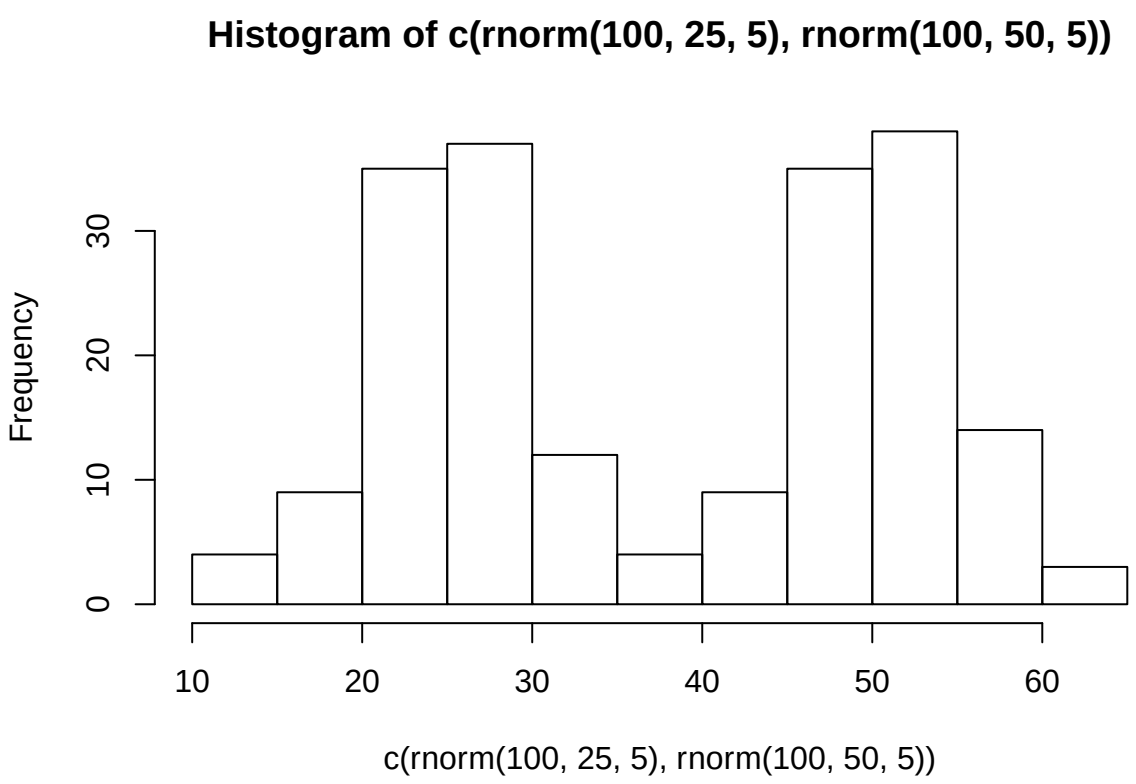


There it is, a bell-shaped normal distribution with a mean of 0, and a standard deviation of 1. You've probably seen things like this before. Now you can sample numbers from normal distributions with any mean or standard deviation, just by changing those parts of the `rnorm` function.

4.2.1.5 mixing it up

The r functions are like Legos, you can put them together and come up with different things. What if wanted to sample from a distribution that looked like a two-humped camel's back? Just sample from rnorm twice like this… mix away.

```
hist( c( rnorm(100,25,5), rnorm(100,50,5)) )
```



4.2.1.6 summary

You can generate as many numbers as your computer can handle with R. PSA: Don't ask R to generate a bajillion numbers or it will explode (or more likely just crash, probably won't explode, that's a metaphor).

4.2.2 sampling distribution of the mean.

Remember the sampling distribution of the sample means from the textbook? Now, you will see the R code that made the graphs from before. As we've seen, we can take samples from distributions in R. We can take as many as we want. We can set our sample-size to be anything we want. And, we can take multiple samples of the same size as many times as we want.

4.2.2.1 Taking multiple samples of the same size

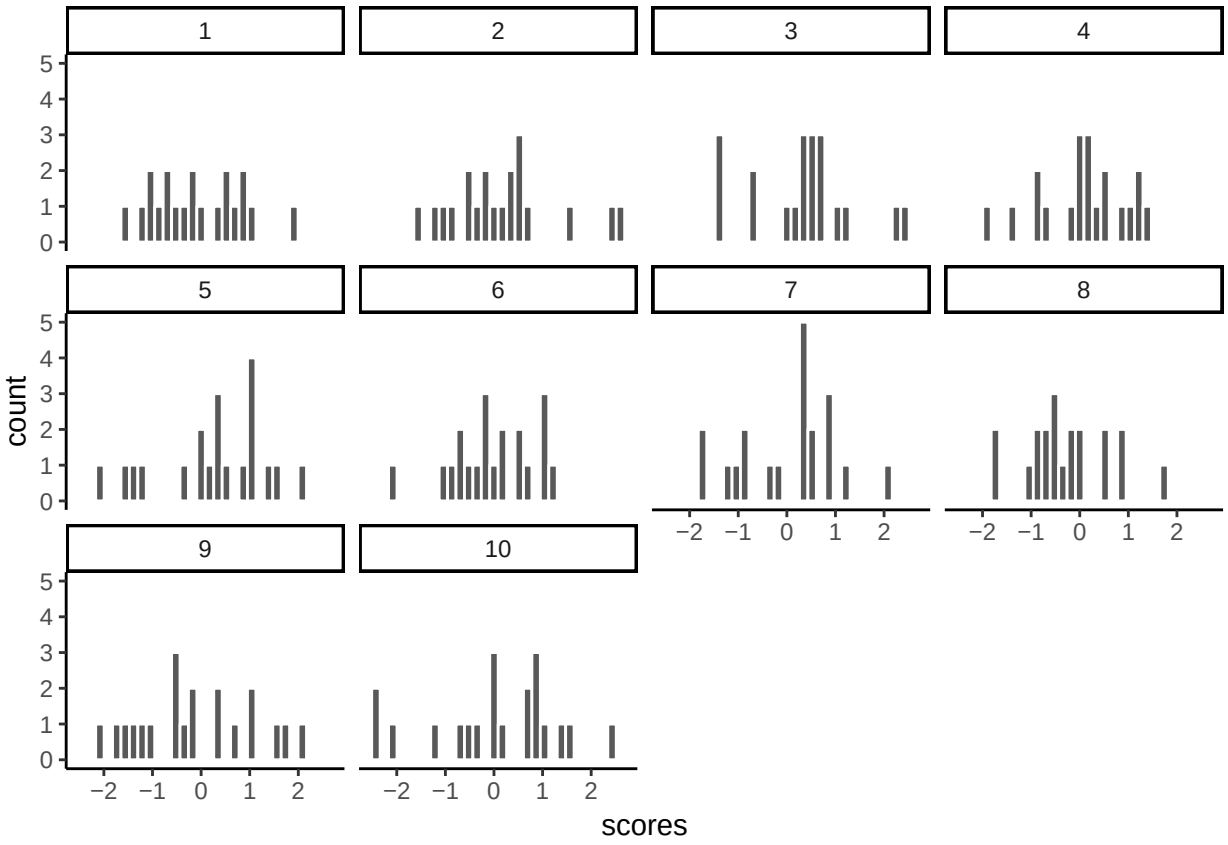
Let's take 10 samples from a normal distribution (mean = 0, and sd = 1). Let's set the sample-size for each to be 20. Then, we'll put them all in a data frame and look at 10 different histrograms, one for each sample.

```
scores <- rnorm(10*20,0,1)
samples <- rep(1:10,each=20)
my_df <- data.frame(samples,scores)
```

First, look at the new my_df dataframe. You can see there is a column with numbers 1 to 10, these are the sample names. There are also 20 scores for each inhe scores column. Let's make histograms for each sample, so we can see what all of the samples look like:

```
ggplot(my_df, aes(x=scores))+
  geom_histogram(color="white")+
```

```
  facet_wrap(~samples)+
  theme_classic()
```



Notice, all of the samples do not have the same looking histogram. This is because of random sampling error. All of the samples are coming from the same normal distributions, but random chance makes each sample a little bit different (e.g., you don't always get 5 heads and 5 tails when you flip a coin right)

4.2.2.2 Getting the means of the samples

Now, let's look at the means of the samples, we will use dplyr to get the means for each sample, and put them in a table:

```
library(dplyr)

sample_means <- my_df %>%
                group_by(samples) %>%
                summarise(means=mean(scores))

knitr::kable(sample_means)
```

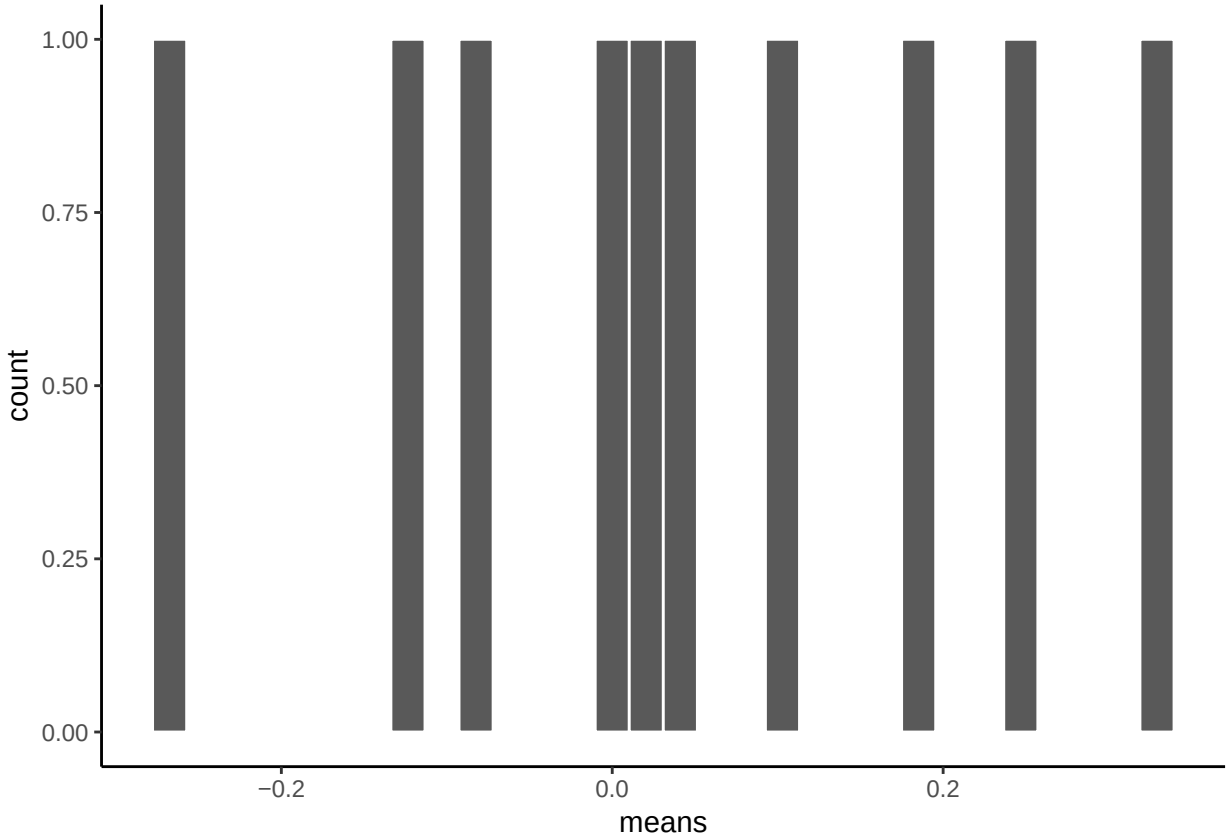
samples	means
1	-0.0736481
2	0.1759426
3	0.3238048
4	0.0977153
5	0.2449510
6	-0.0084961
7	0.0160611
8	-0.2729770
9	-0.1153894
10	0.0382659

So, those are the means of our samples. What should the means be? Well, we would hope they are estimating the mean of the distribution they came from, which was 0. Notice, the numbers are all not 0, but they are kind of close to 0.

histogram for the means of the samples

What if we now plot these 10 means (of each of the samples) in their own distribution?

```
ggplot(sample_means, aes(x=means))+
 geom_histogram(color="white")+
 theme_classic()
```



That is the distribution of the sample means. It doesn't look like much eh? That's because we only took 10 samples right.

Notice one more thing…What is the mean of our 10 sample means? This is a mean of means. Remember that.

```
mean(sample_means$means)
```

```
## [1] 0.04262301
```

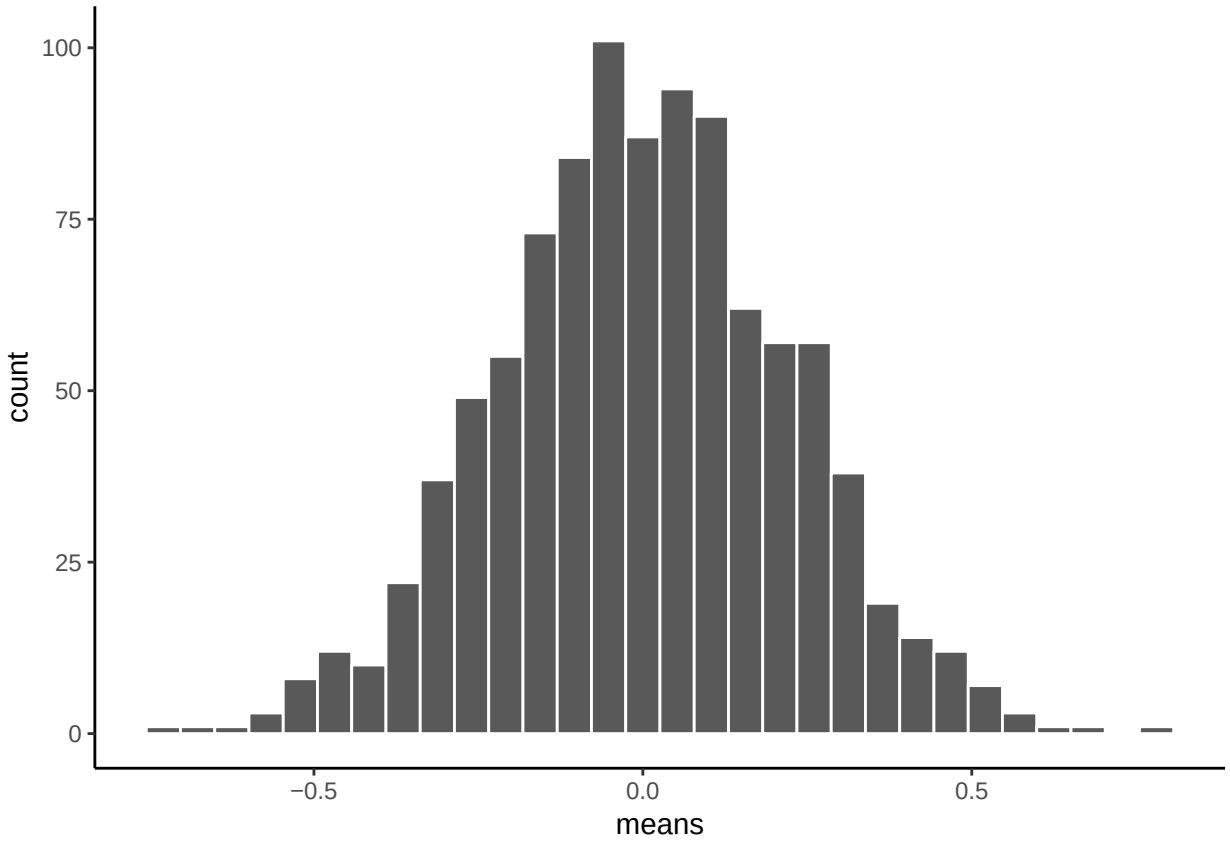
Well, that's pretty close to zero. Which is good. When we average over our samples, they better estimate the mean of the distribution they came from.

4.2.2.3 simulating the distribution of sample means

Our histogram with 10 sample means looked kind of sad. Let's give it some more friends. How about we repeat our little sampling experiment 1000 times.

Explain...We take 1000 samples. Each sample takes 20 scores from a normal distribution (mean=0, sd=1). Then we find the means of each sample (giving us 1000 sample means). Then, we plot that distribution.

```
# get 1000 samples with 20 scores each

scores <- rnorm(1000*20,0,1)
samples <- rep(1:1000,each=20)
my_df <- data.frame(samples,scores)

# get the means of the samples

sample_means <- my_df %>%
                group_by(samples) %>%
                summarise(means=mean(scores))

# make a histogram

 ggplot(sample_means, aes(x=means))+
  geom_histogram(color="white")+
  theme_classic()
```



There, that looks more like a sampling distribution of the sample means. Notice it's properties. It is centered on 0, which tells us that sample means are mostly around zero. It is also bell-shaped, like the normal distribution it came from. It is also quite narrow. The numbers on the x-axis don't go much past -.5 to +.5.

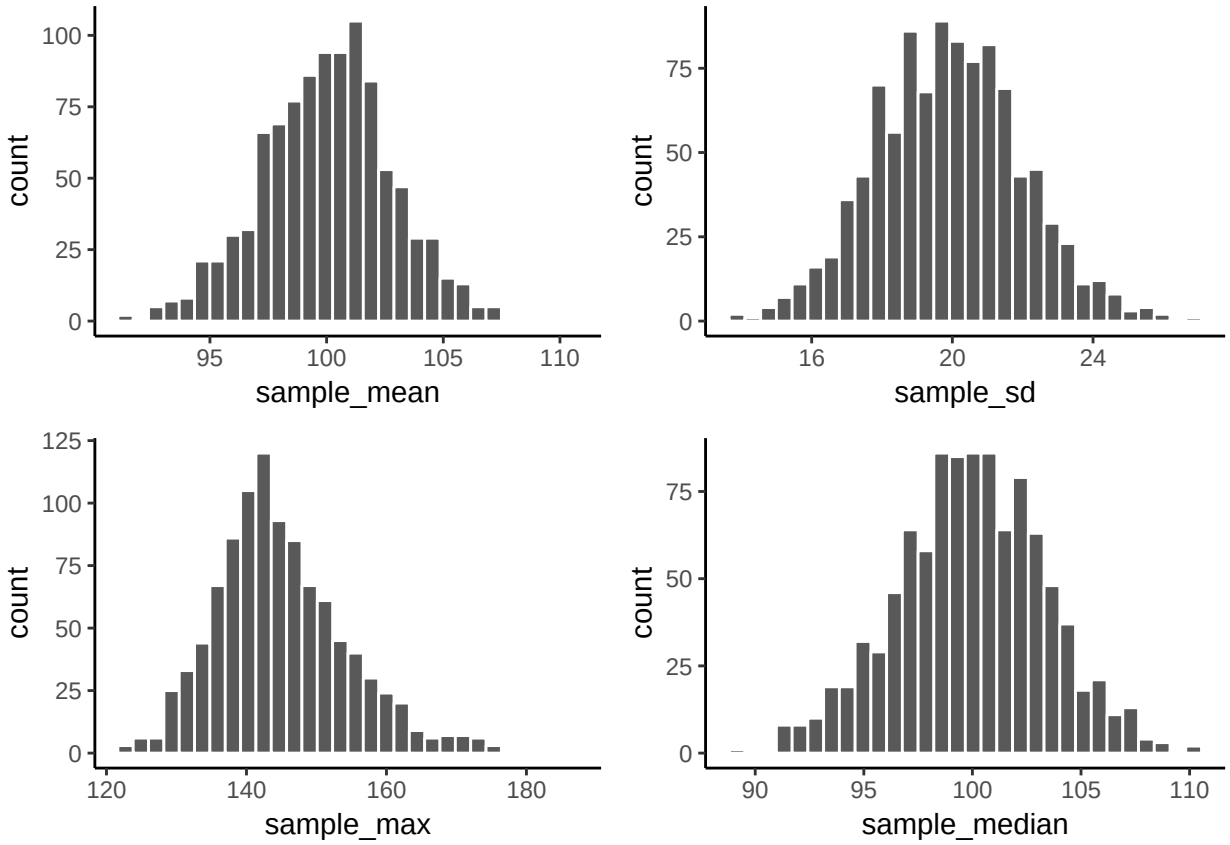
We will use things like the sampling distribution of the mean to make inferences about what chance can do in your data later on in this course.

4.2.3 Sampling distributions for any statistic

Just for fun here are some different sampling distributions for different statistics. We will take a normal distribution with mean = 100, and standard deviation =20. Then, we'll take lots of samples with n = 50 (50 observations per sample). We'll save all of the sample statistics, then plot their histograms. We do the sample means, standard deviations, maximum values, and medians. Let's do it.

```
all_df<-data.frame()
for(i in 1:1000){
  sample<-rnorm(50,100,20)
  sample_mean<-mean(sample)
  sample_sd<-sd(sample)
  sample_max<-max(sample)
  sample_median<-median(sample)
  t_df<-data.frame(i,sample_mean,sample_sd,sample_max,sample_median)
  all_df<-rbind(all_df,t_df)
}

library(ggpubr)
```

```
a<-ggplot(all_df,aes(x=sample_mean))+
  geom_histogram(color="white")+
  theme_classic()
b<-ggplot(all_df,aes(x=sample_sd))+
  geom_histogram(color="white")+
  theme_classic()
c<-ggplot(all_df,aes(x=sample_max))+
  geom_histogram(color="white")+
  theme_classic()
d<-ggplot(all_df,aes(x=sample_median))+
  geom_histogram(color="white")+
  theme_classic()

ggarrange(a,b,c,d,
          ncol = 2, nrow = 2)
```



From reading the textbook and attending lecture, you should be able to start thinking about why these sampling statitistic distributions might be useful…For now, just know that you can make a sampling statistic for pretty much anythin in R, just by simulating the process of sampling, measuring the statistic, doing it over a bunch, and then plotting the histogram. This gives you a pretty good estimate of the distribution for that sampling statistic.

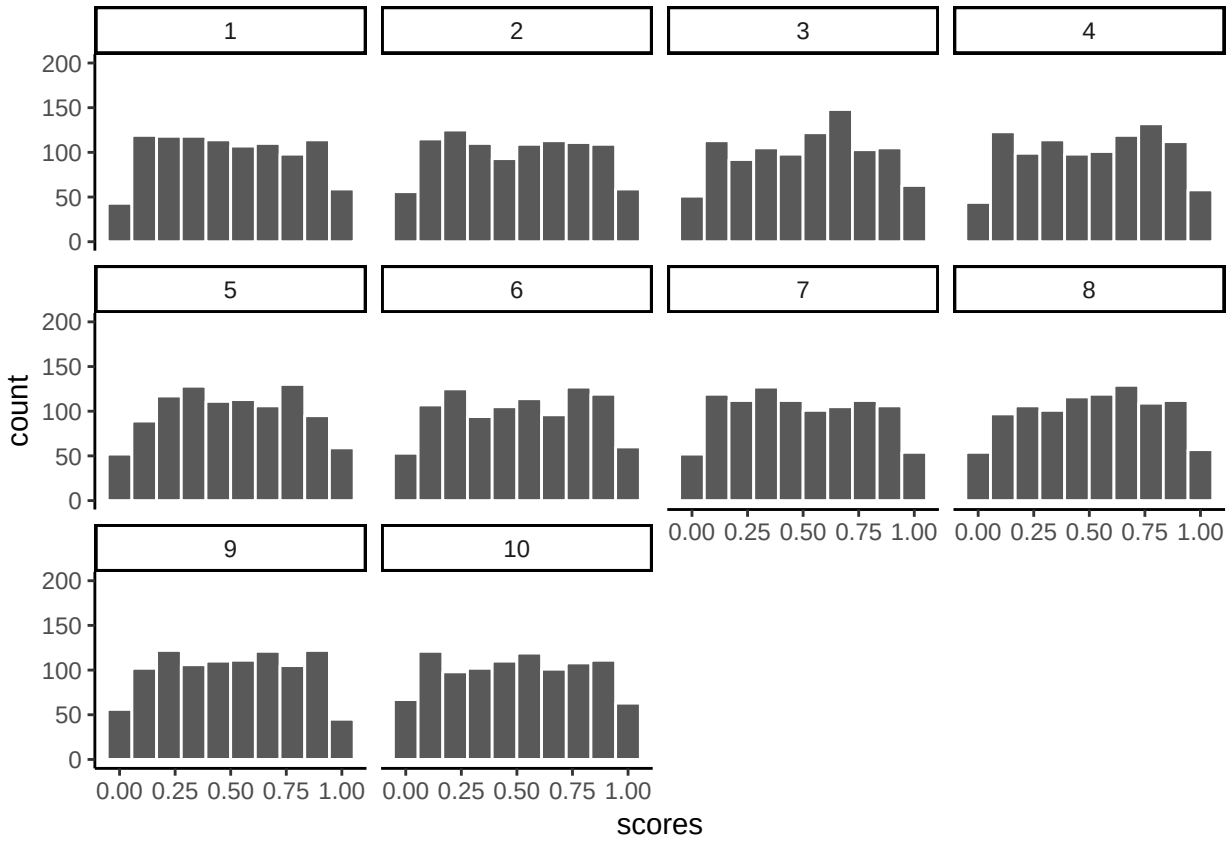
4.2.4 Central limit theorem

We have been building you up for the central limit theorem, described in the textbook and in class. The central limit theorem is basically that the distribution of sample means will be a normal curve. We already

saw that before. But, the interesting thing about it, is that the distribution of your sample means will be normal, even if the distribution the samples came from is not normal. Huh what?

To demonstrat this the next bit of code is modified from what we did ealier. We create 100 samples. Each sample has 1000 observations. All of them come from a uniform distribution between 0 to 1. This means all of the numbers between 0 and 1 should occur equally frequently. Below I plot histrograms for the first 10 samples (out of the 100 total, 100 is too many to look at). Notice the histograms are not normal, they are roughly flat.

```
scores <- runif(100*1000,0,1)
samples <- rep(1:100,each=1000)
my_df <- data.frame(samples,scores)

ggplot(my_df[1:(10*1000),], aes(x=scores))+
  geom_histogram(color="white", bins=10)+
  facet_wrap(~samples)+
  theme_classic()+
  ylim(0,200)
```

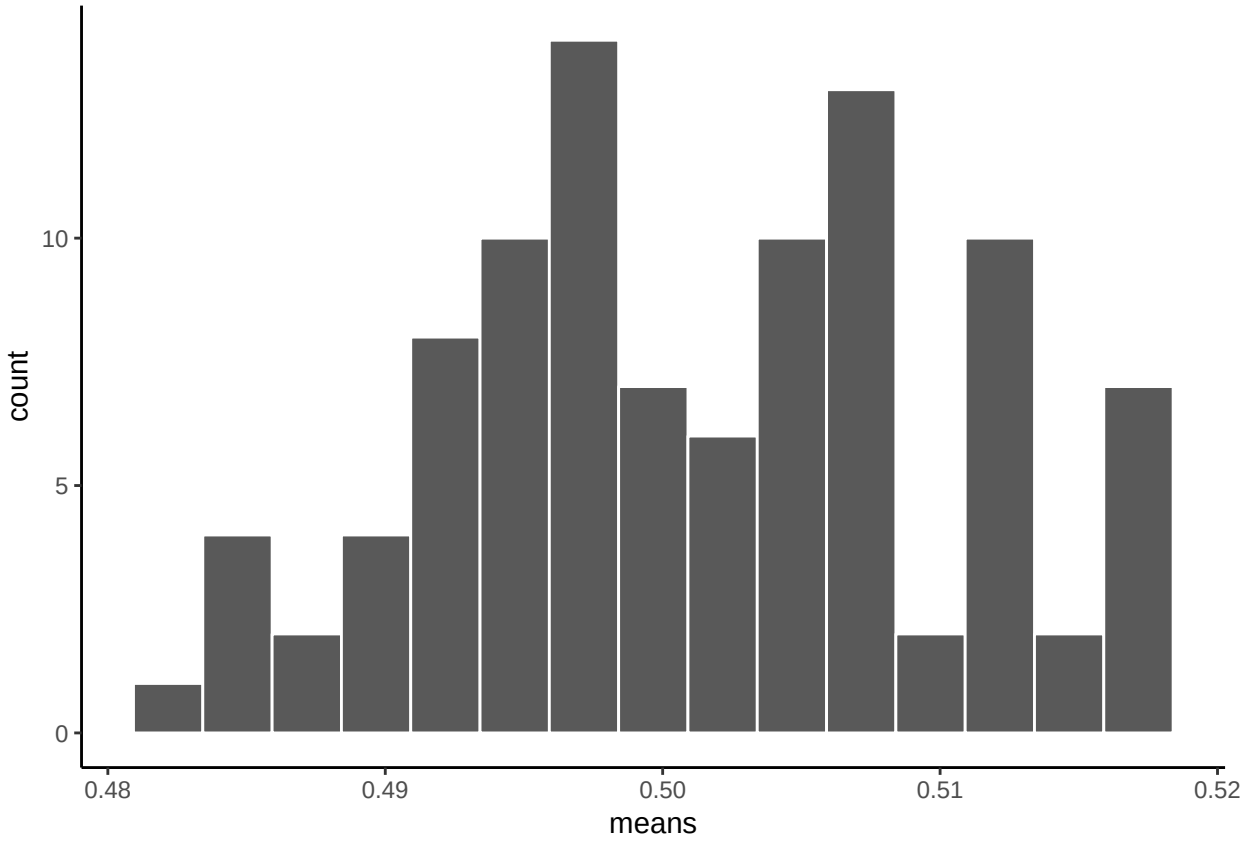


We took samples from a flat uniform distribution, and the samples themselves look like that same flat distribution.

HOWEVER, if we now do the next step, and compute the means of each of our 100 samples, we could then look at the sampling distribution of the sample means. Let's do that:

```
sample_means <- my_df %>%
                group_by(samples) %>%
                summarise(means=mean(scores))
```

```
# make a histogram

ggplot(sample_means, aes(x=means))+
 geom_histogram(color="white", bins=15)+
 theme_classic()
```



As you can see, the sampling distribution of the sample means is not flat. It's shaped kind of normalish. If we had taken many more samples, found their means, and then looked at a histogram, it would become even more normal looking. Because that's what happens according to the central limit theorem.

4.2.5 The normal distribution

"Why does any of this matter, why are we doing this, can we stop now!!!!!! PLEEEEAASSEE, somebody HELP".

We are basically just repeating what was said in the textbook, and the lecture, so that you get the concept explained in a bunch of different ways. It will sink in.

The reason the central limit theorem is important, is because researchers often take many samples, then analyse the means of their samples. That's what they do.

An experiment might have 20 people. You might take 20 measurements from each person. That's taking 20 samples. Then, because we know that samples are noisy. We take the means of the samples.

So, what researchers are often looking at, (and you too, very soon) are means of samples. Not just the samples. And, now we know that means of samples (if we had a lot of samples), look like they are distributed normally (the central limit theorem says the should be).

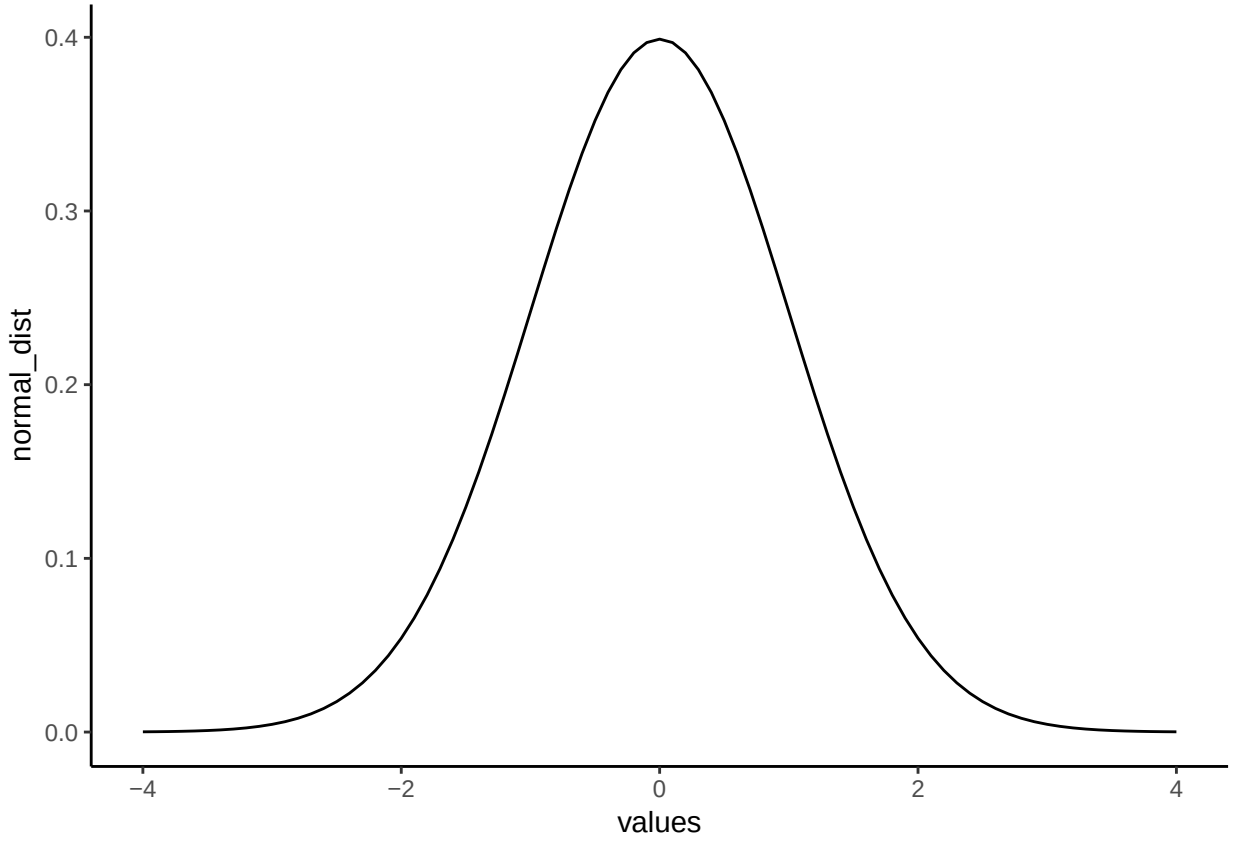
We can use this knowledge. If we learn a little bit more about normal distributions, and how they behave and work, we can take that and use it to understand our sample means better. This will become more clear as head into the topic of statistical inference next week. This is all a build up for that.

To continue the build-up we now look at some more properties of the normal distribution.

4.2.5.1 Graphing the normal distribution

"Wait, I thought we alread did that". We sort of did. We sampled numbers and made histograms that looked like normal distributions. But, a "normal distribution" is more of an abstract idea. It looks like this in the abstract:

```
normal_dist <- dnorm(seq(-4,4,.1), 0, 1)
values <-seq(-4,4,.1)
normal_df <-data.frame(values,normal_dist)

ggplot(normal_df, aes(x=values,y=normal_dist))+
  geom_line()+
  theme_classic()
```



A really nice shaped bell-like thing. This normal distribution has a mean of 0, and standard deviation of 1. The heights of the lines tell you roughly how likely each value is. Notice, it is centered on 0 (most likely that numbers from this distributino will be near 0), and it goes down as numbers get bigger or smaller (so bigger or smaller numbers get less likely). There is a range to it. Notice the values don't go much beyond -4 and +4. This because those values don't happen very often. Theoretically any value could happen, but really big or small values have really low probabilites.

4.2.5.2 calculating the probability of specific ranges.

We can use R to tell us about the probability of getting numbers in a certain range. For example, when you think about. It should be obvious that you have a 50% probability of getting the number 0 or greater. Half of the distribution is 0 or greater, so you have a 50% probability.

We can use the `pnorm` function to confirm this:

```
pnorm(0, mean = 0, sd= 1, lower.tail=FALSE)
```

```
## [1] 0.5
```

Agreed, pnorm tells us the probability of getting 0 or greater is .5.

Well, what is the probability of getting a 2 or greater? That's a bit harder to judge, obviously less than 50%. Use R like this to find out:

```
pnorm(2, mean = 0, sd= 1, lower.tail=FALSE)
```

```
## [1] 0.02275013
```

The probability of getting a 2 or greater is .0227 (not very probable)

What is the probability of getting a score between -1 and 1?

```
ps<-pnorm(c(-1,1), mean = 0, sd= 1, lower.tail=FALSE)
ps[1]-ps[2]
```

```
## [1] 0.6826895
```

About 68%. About 68% of all the numbers would be between -1 and 1. So naturally, about 34% of the numbers would be between 0 and 1. Notice, we are just getting a feeling for this, you'll see why in a bit when we do z-scores (some of you may realize we are already doing that…)

What about the numbers between 1 and 2?

```
ps<-pnorm(c(1,2), mean = 0, sd= 1, lower.tail=FALSE)
ps[1]-ps[2]
```

```
## [1] 0.1359051
```

About 13.5% of numbers fall in that range, not much.

How about between 2 and 3?

```
ps<-pnorm(c(2,3), mean = 0, sd= 1, lower.tail=FALSE)
ps[1]-ps[2]
```

```
## [1] 0.02140023
```

Again a very small amount, only 2.1 % of the numbers, not a alot.

4.2.5.3 summary pnorm

You can always use pnorm to figure how the probabilities of getting certain values from any normal distribution. That's great.

4.2.6 z-scores

We just spent a bunch of time looking at a very special normal distribution, the one where the mean = 0, and the standard deviation = 1. Then we got a little bit comfortable with what those numbers mean. 0

happens a lot. Numbers between -1 and 1 happen alot. Numbers bigger or smaller than 1 also happen fairly often, but less often. Number bigger than 2 don't happen alot, numbers bigger than 3 don't happen hardly at all.

We can use this knowledge for our convenience. Often, we are not dealing with numbers exactly like these. For example, someone might say, I got a number, it's 550. It came from a distribution with mean = 600, and standard deviation = 25. So, does 545 happen alot or not? The numbers dont' tell you right away.

If we were talking about our handy distribution with mean = 0 and standard deviation = 1, and I told I got a number 4.5 from that distribution. You would automatically know that 4.5 doesn't happen a lot. Right? Right!

z-scores are a way of transforming one set of numbers into our neato normal distribution, with mean = 0 and standard deviation = 1.

Here's a simple example, like what we said in the textbook. If you have a normal distribution with mean = 550, and standard deviation 25, then how far from the mean is the number 575? It's a whole 25 away (550+25 = 575). How many standard deviations is that? It's 1 whole standard deviation. So does a number like 575 happen a lot? Well, based on what you know about normal distributions, 1 standard deviation of the mean isn't that far, and it does happen fairly often. This is what we are doing here.

4.2.6.1 Calculating z-scores

1. get some numbers

```
some_numbers <- rnorm(20,50,25)
```

2. Calculate the mean and standard deviation

```
my_mean <- mean(some_numbers)
my_sd <-sd(some_numbers)

print(my_mean)
```

```
## [1] 50.02857
```

```
print(my_sd)
```

```
## [1] 27.23455
```

3. subtract the mean from your numbers

```
differences<-some_numbers-my_mean
print(my_sd)
```

```
## [1] 27.23455
```

4. divide by the standard deviation

```
z_scores<-differences/my_sd
print(z_scores)
```

```
##  [1] -0.20011359  1.21876252  0.77224612  0.49107624 -0.59732747
##  [6] -0.39124337  0.14905347 -0.82439512  0.06832726 -1.07170308
## [11] -1.62565408 -0.13210288 -0.80832484 -0.57890065 -0.35931783
## [16]  2.46523045  0.47588190  0.54268971  1.58377947 -1.17796423
```

Done. Now you have converted your original numbers into what we call standardized scores. They are standardized to have the same properties (assumed properties) as a normal distribution with mean = 0, and sd = 1.

You could look at each of your original scores, and try to figure out if they are likely or unlikely numbers. But, if you make them into z-scores, then you can tell right away. Numbers close to 0 happen alot, bigger numbers closer to 1 happen less often, but still fairly often, and number bigger than 2 or 3 hardly happen at all.

If someone said they did 3 standard deviations above the mean on a test, this means they did really good, and hardly nobody also did that good.

Z-scores are a useful language once you understand them. We show you this language in case you find them useful.

4.2.7 Generalization Exercise

Complete the generalization exercise described in your R Markdown document for this lab.

4.2.8 Writing asignment

Complete the writing assignment described in your R Markdown document for this lab. When you have finished everything. Knit the document and hand in your stuff (you can submit your .RMD file to blackboard if it does not knit.)

4.3 Excel

How to do it in Excel

4.4 SPSS

How to do it in SPSS

4.5 JAMOVI

How to do it in JAMOVI

Chapter 5

Lab 5: Fundamentals of Hypothesis Testing

Some inspiring quote —Inspiring Person

5.1 Outline of Problem to solve

Stuff we need to say in general

5.1.1 important things

Other things to say

5.2 R

How to do it in R

5.3 Excel

How to do it in Excel

5.4 SPSS

How to do it in SPSS

5.5 JAMOVI

How to do it in JAMOVI

Chapter 6

Lab 6: t-Test (one-sample, paired sample)

This lab is modified and extended from Open Stats Labs. Thanks to Open Stats Labs (Dr. Kevin P. McIntyre) for their fantastic work.

6.1 Does Music Convey Social Information to Infants?

This lab activity uses the open data from Experiment 1 of Mehr, Song, and Spelke (2016) to teach one-sample and paired samplest -tests. Results of the activity provided below should exactly reproduce the results described in the paper.

6.1.1 STUDY DESCRIPTION

Parents often sing to their children and, even as infants, children listen to and look at their parents while they are singing.Research by Mehr, Song, and Spelke (2016) sought to explore the psychological function that music has for parents and infants, by examining the hypothesis that particular melodies convey important social information to infants.Specifically, melodies convey information about social affiliation.

The authors argue that melodies are shared within social groups. Whereas children growing up in one culture may be exposed to certain songs as infants (e.g., "Rock-a-bye Baby"), children growing up in other cultures (or even other groups within a culture) may be exposed to different songs.Thus, when a novel person (someone who the infant has never seen before) sings a familiar song, it may signal to the infant that this new person is a member of their social group.

To test this hypothesis, the researchers recruited 32 infants and their parents to complete an experiment.During their first visit to the lab, the parents were taught a new lullaby (one that neither they nor their infants had heard before).The experimenters asked the parents to sing the new lullaby to their child every day for the next 1-2 weeks.

Following this 1-2 week exposure period, the parents and their infant returned to the lab to complete the experimental portion of the study.Infants were first shown a screen with side-by-side videos of two unfamiliar people, each of whom were silently smiling and looking at the infant.The researchers recorded the looking behavior (or gaze) of the infants during this 'baseline' phase. Next, one by one, the two unfamiliar people on the screen sang either the lullaby that the parents learned or a different lullaby (that had the same lyrics and rhythm, but a different melody).Finally, the infants saw the same silent video used at baseline, and the

researchers again recorded the looking behavior of the infants during this 'test' phase.For more details on the experiment's methods, please refer to Mehr et al. (2016) Experiment 1.

6.2 Lab skills learned

1. Conducting a one-sample t-test
2. Conducting a two-sample t-test
3. Plotting the data
4. Discussing inferences and limitations

6.3 Important Stuff

- citation: Mehr, S. A., Song. L. A., & Spelke, E. S. (2016). For 5-month-old infants, melodies are social. Psychological Science, 27, 486-501.
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format

6.4 R

6.4.1 Loading the data

The first thing to do is download the .csv formatted data file, using the link above, or just click here. It turns out there are lots of ways to load .csv files into R.

1. Load the data.table library. Then use the fread function and supply the web address to the file. Just like this. No downloading required.

```
library(data.table)
all_data <- fread("https://raw.githubusercontent.com/CrumpLab/statisticsLab/master/data/MehrSongSpelke2
```

2. Or, if you downloaded the .csv file. Then you can use fread, but you need to point it to the correct file location. The file location in this next example will not work for you, because the file is on my computer.

```
library(data.table)
all_data <- fread("Data/MehrSongSpelke2016.csv")
```

6.4.2 Inspect the data frame

When you have loaded data it's always a good idea to check out what it looks like. You can look at all of the data in the environment tab on the top right hand corner. The data should be in a variable called `all_data`. Clicking on `all_data` will load it into a viewer, and you can scroll around. This can be helpful to see things. But, there is so much data, can be hard to know what you are looking for.

6.4.2.1 summarytools

The summarytools packages give a quick way to summarize all of the data in a data frame. Here's how. When you run this code you will see the summary in the viewer on the bottom right hand side. There's a

little browser button (arrow on top of little window) that you can click to expand and see the whole thing in a browser.

```
library(summarytools)
view(dfSummary(all_data))
```

6.4.3 Get the data for Experiment one

The data contains all of the measurements from all five experiments in the paper. By searching through the `all_data` dataframe, you should look for the variables that code for each experiment. For example, the third column is called `exp1`, which stands for experiment 1. Notice that it contains a series of 1s. If you keep scrolling down, the 1s stop. These 1s identify the rows associated with the data for Experiment 1. We only want to analyse that data. So, we need to filter our data, and put only those rows into a new variable. We do this with the `dplyr` library, using the `filter` function.

```
library(dplyr)
experiment_one <- all_data %>% filter(exp1==1)
```

Now if you look at the new variable `experiment_one`, there are only 32 rows of data. Much less than before. Now we have the data from experiment 1.

6.4.4 Baseline phase: Conduct a one sample t-test

You first want to show that infants' looking behavior did not differ from chance during the baseline trial. The baseline trial was 16 seconds long. During the baseline, infants watched a video of two unfamilar people, one of the left and one on the right. There was no sound during the basline. Both of the actors in the video smiled directly at the infant.

The important question was to determine whether the infant looked more or less to either person. If they showed no preference, the infant should look at both people about 50% of the time. How could we determine whether the infant looked at both people about 50% of the time?

The `experiment_one` dataframe has a column called `Baseline_Proportion_Gaze_to_Singer`. All of these values show how the proportion of time that the infant looked to the person who would later sing the familiar song to them. If the average of these proportion is .5 across the infants, then we would have some evidence that the infants were not biased at the beginning of the experiment. However, if the infants on average had a bias toward the singer, then the average proportion of the looking time should be different than .5.

Using a one-sample t-test, we can test the hypothesis that our sample mean for the `Baseline_Proportion_Gaze_to_Singer` was not different from .5.

To do this in R, we just need to isolate the column of data called `Baseline_Proportion_Gaze_to_Singer`. We will do this using the `$` operator. The `$` operator is placed after any data frame variable, and allows you to select a column of the data. The next bit of code will select the column of data we want, and put it into a new variable called `Baseline`. Note, if you type `exp1$` then Rstudio should automatically bring up all the columns you can choose from.

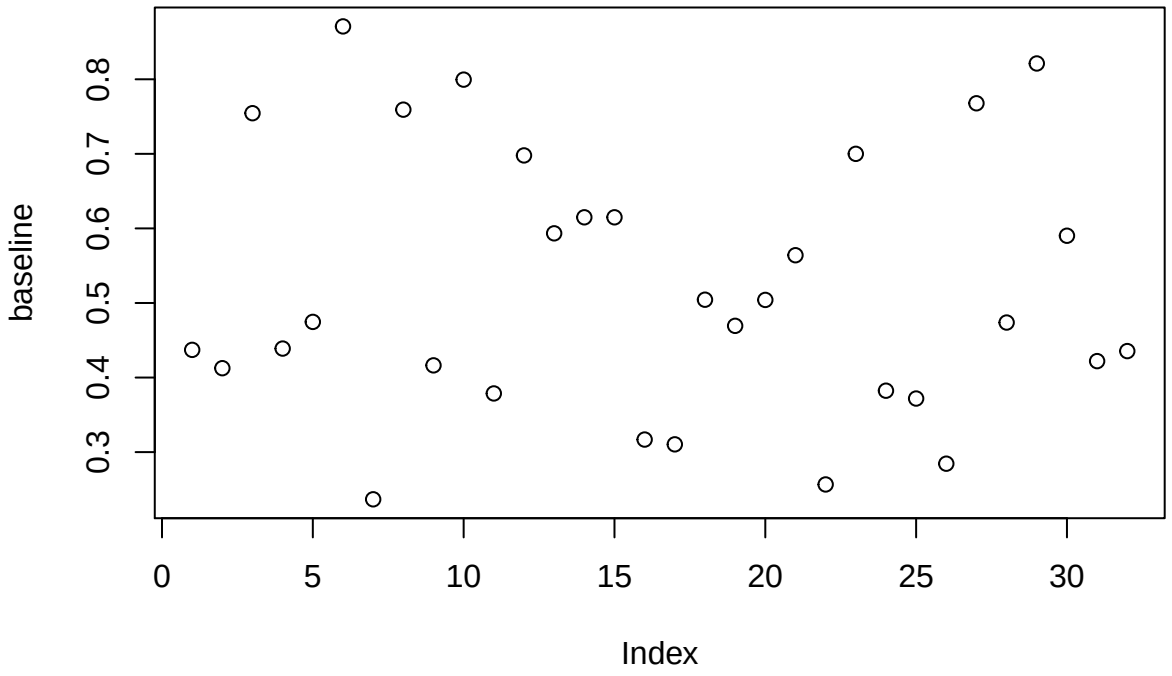
```
baseline <- experiment_one$Baseline_Proportion_Gaze_to_Singer
```

6.4.4.1 Look at the numbers

Question: Why is it important to look at your numbers? What could happen if you didn't?

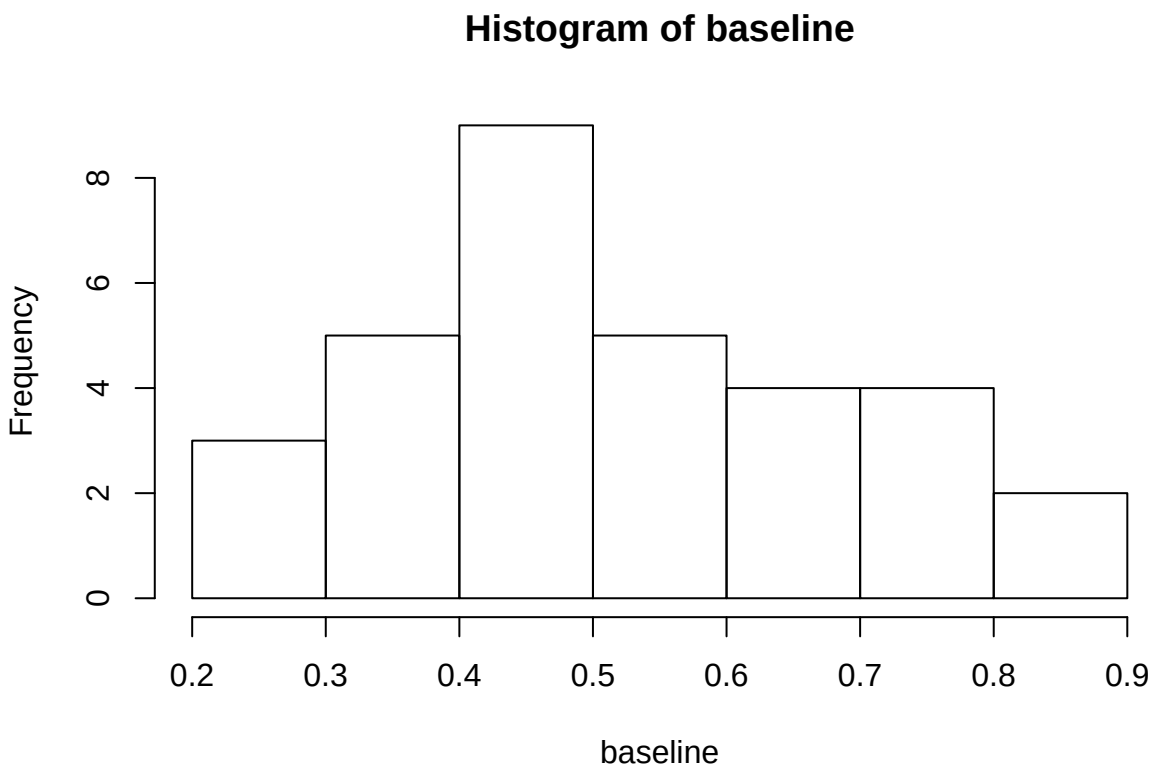
Ok, we could just do the t-test right away, it's really easy, only one line of code. But, we haven't even looked at the numbers yet. Let's at least do that. First, we'll just use plot. It will show every data point for each infant as a dot.

```
plot(baseline)
```



That's helpful, we see that the dots are all over the place. Let's do a histrogram, so we can get a better sense of the frequency of different proportions.

```
hist(baseline)
```



6.4.4.2 Look at the descriptives

Let's get the mean and standard deviation of the sample

```
mean(baseline)
```

```
## [1] 0.5210967
```

```
sd(baseline)
```

```
## [1] 0.1769651
```

Ok, so just looking at the mean, we see the proportion is close to .5 (it's .521). And, we see there is a healthy amount of variance (the dots were all over the place), as the standard deviation was about .176.

Question: Based on the means and standard deviations can you make an educated guess about what the t and p values might be? Learn how to do this and you will be improving your data-sense.

Now, before we run the t-test, what do you think is going to happen? We are going to get a t-value, and an associated p-value. If you can make a guess at what those numbers would be right now in your head, and those end up being pretty close to the ones we will see in a moment, then you should pat yourself on the back. You have learned how to have intuitions about the data. As I am writing this I will tell you that 1) I can't remember what the t and p was from the paper, and I haven't done the test yet, so I don't know the answer. So, I am allowed to guess what the answer will be. Here are my guesses t(31) = 0.2, p = .95. The numbers in the brackets are degrees of freedom, which we know are 31 (df= n-1 = 32-1= 31). More important than the specific numbers I am guessing (which will probably be wrong), I am guessing that the p-value will be pretty large, it will not be less than .05, which the author's set as their alpha rate. So, I am guessing we will not reject the hypothesis that .52 is different from .5.

Let's do the t-test and see what happens.

6.4.4.3 Conduct t.test

```
t.test(baseline, mu=.5)
```

```
##
##  One Sample t-test
##
## data:  baseline
## t = 0.67438, df = 31, p-value = 0.5051
## alternative hypothesis: true mean is not equal to 0.5
## 95 percent confidence interval:
##  0.4572940 0.5848994
## sample estimates:
## mean of x
## 0.5210967
```

Question: Why was the baseline condition important for the experiment? What does performance in this condition tell us?

So, there we have it. We did a one-sample t-test. Here's how you would report it, t(31) = .67, p = .505. Or, we might say something like:

> During the baseline condition, the mean proportion looking time toward the singer was .52, and was not significantly different from .5, according to a one-sample test, t(31) = .67, p = .505.

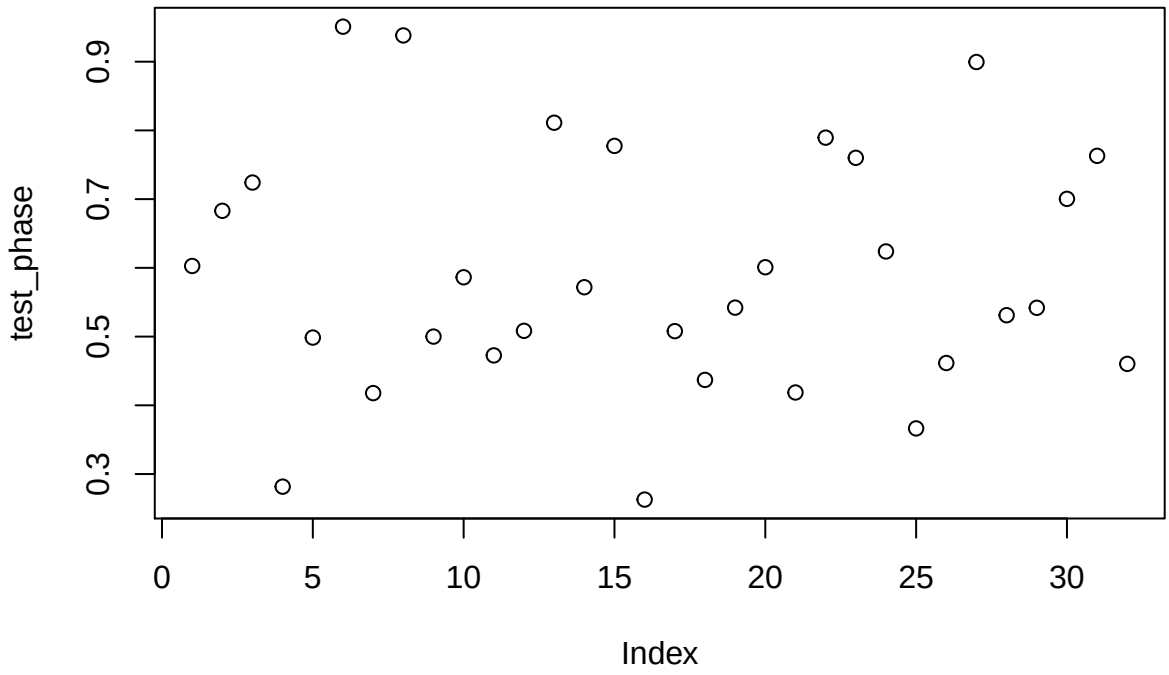
You should take the time to check this result, and see if it is the same one that was reported in the paper.

6.4.5 Test phase

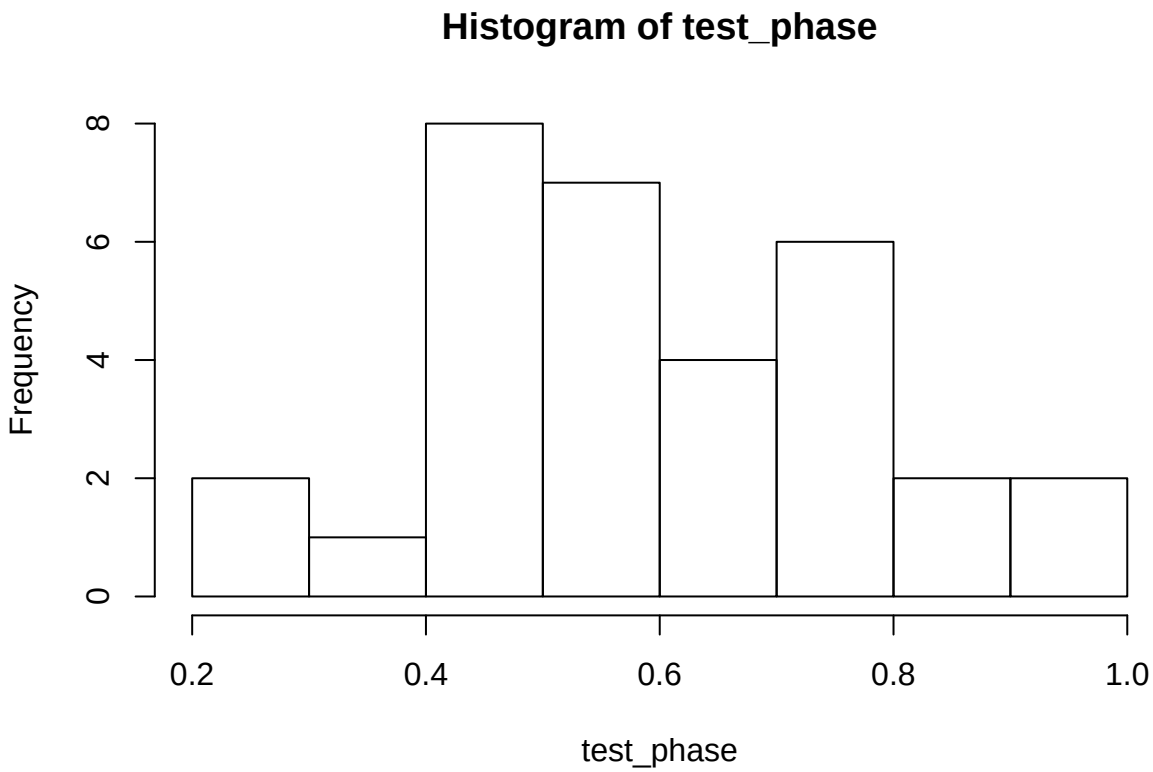
Remember how the experiment went. Infants watched silent video recordings of two women (Baseline). Then each person sung a song, one was familiar to the infant (their parents sung the song to them many times), and one was unfamiliar (singing phase). After the singing phase, the infants watched the silent video of the two singers argain (test phase). The critical question was whether the infants would look more to the person who sung the familiar song compared to the person who sun the unfamiliar song. If the infants did this, they should look more than 50% of the time to the singer who sang the familiar song. We have the data, we can do another one sample t-test to find out. We can re-use all the code we already wrote to do this. I'll put it all in one place. If we run all of this code, we will see all of the things we want to see.

We only need to make two changes. We will change `experiment_one$Baseline_Proportion_Gaze_to_Singer` to `experiment_one$Test_Proportion_Gaze_to_Singer`, because that column has the test phase data. And, instead of putting the data into the variable `baseline`. We will make a new variable called `test_phase` to store the data.

```
test_phase <- experiment_one$Test_Proportion_Gaze_to_Singer
plot(test_phase)
```




```
hist(test_phase)
```




```
mean(test_phase)
```

```
## [1] 0.5934913
```

```
sd(test_phase)
```

```
## [1] 0.1786884
```

```
t.test(test_phase, mu = .5)
```

```
##
##  One Sample t-test
##
## data:  test_phase
## t = 2.9597, df = 31, p-value = 0.005856
## alternative hypothesis: true mean is not equal to 0.5
## 95 percent confidence interval:
##  0.5290672 0.6579153
## sample estimates:
## mean of x
## 0.5934913
```

Question: Why was the test condition important for the experiment? What does performance in this condition tell us?

Alright. What did we find? You should take a stab at writing down what we found. You can use the same kind of language that I used from the first one sample-test. You should state the mean proportion, the t-value, the dfs, and the p-value. You should be able to answer the question, did the infants look longer at the singer who sang the familiar song? And, did they look longer than would be consist with chance at 50%.

6.4.6 Paired-samples t-test

The paired samples t-test is easy to do. We've already made two variables called `baseline`, and `test_phase`. These contain each of the infants looking time proportions to the singer for both parts of the experiment. We can see if the difference between them was likely or unlikely due to chance by running a paired samples t-test. We do it like this in one line:

```
t.test(test_phase, baseline, paired=TRUE, var.equal=TRUE)
```

```
##
##  Paired t-test
##
## data:  test_phase and baseline
## t = 2.4164, df = 31, p-value = 0.02175
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.01129217 0.13349698
## sample estimates:
## mean of the differences
##              0.07239458
```

Question: Why was the paired samples t-test necessary if we already did two one sample t-test? What new question is the paired samples t-test asking?

I'l leave it to you to interpret these values, and to see if they are the same as the ones in the paper. Based on these values what do you conclude? Is there a difference between the mean proportion looking times for the baseline and testing phase?

6.4.6.1 Relationship between one-sample and paired sample t-test

Question: Why is it that a paired samples t-test can be the same as the one sample t-test? What do you have to do the data in the paired samples t-test in order to conduct a one-sample t-test that would give you the same result?

We've discussed in the textbook that the one-sample and paired sample t-test are related, they can be the same test. The one-sample test whether a sample mean is different from some particular mean. The paired sample t-test, is to determine whether one sample mean is different from another sample mean. If you take the scores for each variable in a paired samples t-test, and subtract them from one another, then you have one list of difference scores. Then, you could use a one sample t-test to test whether these difference scores are different from 0. It turns out you get the same answer from a paired sample t-test testing the difference between two sample means, and the one sample t-test testing whether the mean difference of the difference scores between the samples are different from 0. We can show this in r easily like this:

```
t.test(test_phase, baseline, paired=TRUE, var.equal=TRUE)
```

```
##
##  Paired t-test
##
## data:  test_phase and baseline
## t = 2.4164, df = 31, p-value = 0.02175
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.01129217 0.13349698
## sample estimates:
## mean of the differences
##              0.07239458
```

```
difference_scores<-test_phase-baseline
t.test(difference_scores, mu=0)
```

```
##
##  One Sample t-test
##
## data:  difference_scores
## t = 2.4164, df = 31, p-value = 0.02175
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.01129217 0.13349698
## sample estimates:
##  mean of x
## 0.07239458
```

6.4.6.2 Usefulness of difference scores

Ok fine, the paired samples t-test can be a one sample t-test if you use the difference scores. This might just seem like a mildly useful factoid you can use for stats trivia games (which no one plays). The point of drawing your attention to the relationship, is to get you to focus on the difference scores. These are what we are actually interested in.

Let's use the difference scores to one more useful thing. Sometime the results of a t-test aren't intuitively obvious. By the t-test we found out that a small difference between the test phase and baseline was not likely produced by chance. How does this relate to the research question about infants using familiar songs as cues for being social? Let's ask a very simple question. How many infants actually showed the bias? How many infants out of 32 looked longer at the singer who sang the familiar song during test, compared to during baseline.

We can determine this by calculating the difference scores. Then, asking how many of them were greater than zero:

```
difference_scores <- test_phase-baseline
length(difference_scores[difference_scores>0])
```

```
## [1] 22
```

So, 22 out of 32 infants showed the effect. To put that in terms of probability, 68.75% of infants showed the effect. These odds and percentages give us another way to appreciate how strong the effect is. It wasn't strong enough for all infants to show it.

6.4.7 Graphing the findings

It is often useful to graph the results of our analysis. We have already looked at dot plots and histograms of the individual samples. But, we also conducted some t-tests on the means of the baseline and test_phase samples. One of the major questions was whether these means are different. Now, we will make a graph that shows the means for each condition. Actually, we will make a few different graphs, so that you can think about what kinds of graphs are most helpful. There are two major important things to show: 1) the sample means, and 2) a visual aid for statistical inference showing whether the results were likely due to chance.

We will use the ggplot2 package to make our graphs. Remember, there are two steps to take when using ggplot2. 1) put your data in a long form dataframe, where each measure has one row, and 2) define the layers of the ggplot.

6.4.7.1 Make the dataframe for plotting

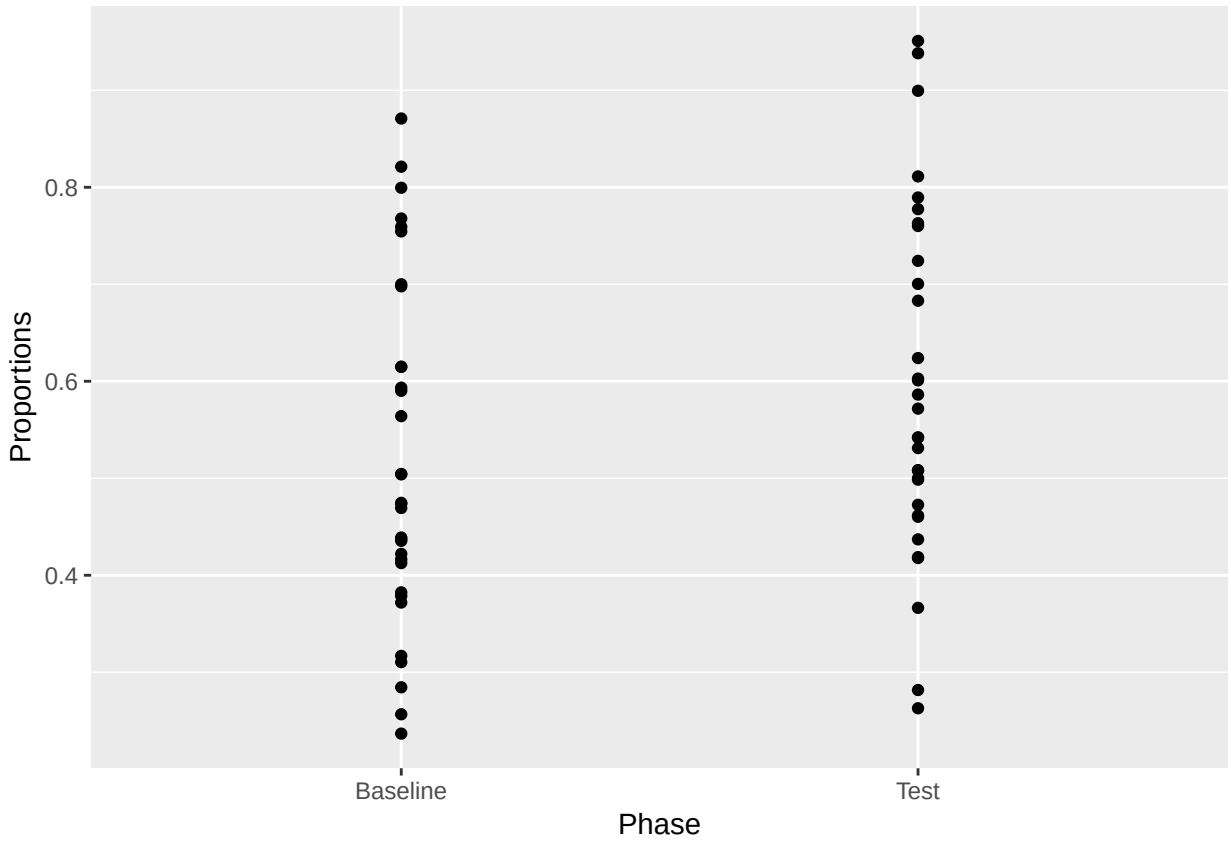
To start we will need 2 columns. One column will code the experimental phase, Baseline or Test. There are 32 observations in each phase, so we want the word `Baseline` to appear 32 times, followed by the word `Test` 32 times. Then we want a single column with each of the proportions for each infant.

```
Phase <- rep(c("Baseline","Test"), each = 32)
Proportions <- c(baseline,test_phase)
plot_df <- data.frame(Phase,Proportions)
```

6.4.7.2 Dot plot of raw scores

This shows every scores value on the y-axis, split by the baseline and test groups. If you just looked at this, you might not think the test phase was different from the basline phase. Still very useful to see the spread of individual scores. Supports the intuition that the scores are still kind of in a similar ballpark.

```
library(ggplot2)
ggplot(plot_df, aes(x=Phase, y=Proportions))+
  geom_point()
```



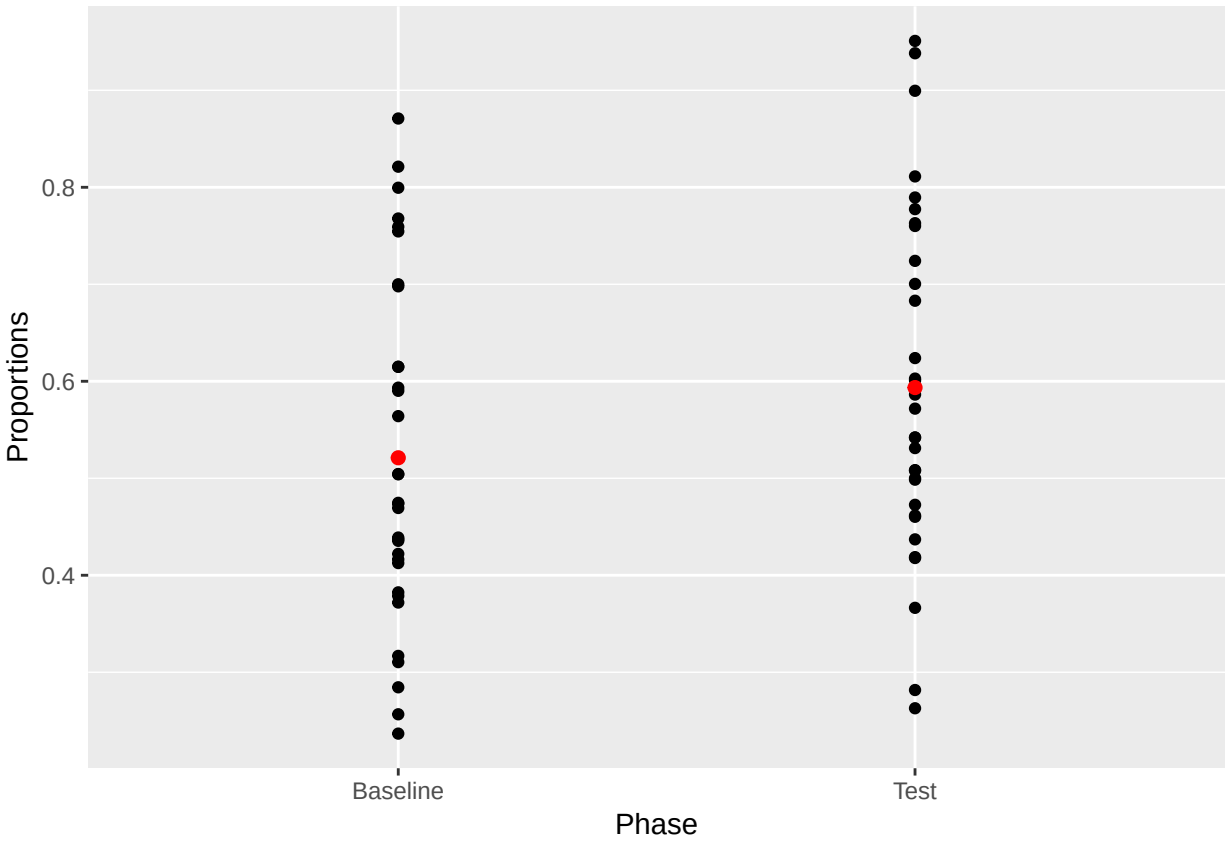
6.4.7.3 Dot plot with means and raw scores

Question: What kinds of inferences about the role of chance in producing the difference between the means can we make from this graph? What is missing?

`ggplot2` is great because it let's us add different layers on top of an existing plot. It would be good to see where the mean values for each sample lie on top of the sample scores. We can do this. But, to do it, we need to supply ggplot with another data frame, one that contains the means for each phase in long form. There are are only two phases, and two means, so we will make a rather small data.frame. It will have two columns, a Phase column, and Mean_value column. There will only be two rows in the dataframe, one for the mean for each phase.

To make the smaller data frame for the means we will use the `aggregate` function. This allows us to find the means for each phase from the plot_df dataframe. It also automatically returns the data frame we are looking for.

```
mean_df <- aggregate(Proportions ~ Phase, plot_df, mean)

ggplot(plot_df, aes(x=Phase, y=Proportions))+
  geom_point()+
  geom_point(data=mean_df, color="Red", size=2)
```



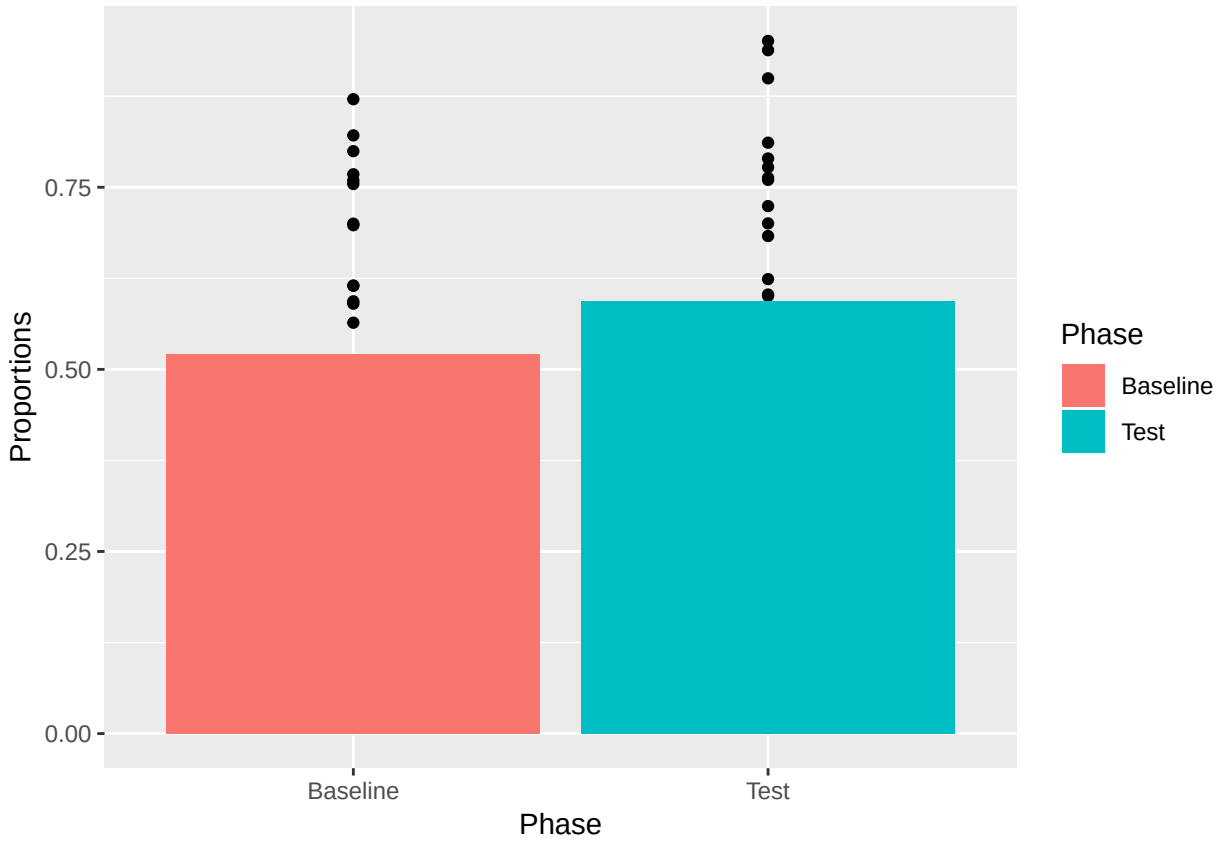
6.4.7.4 Bar plot

It's very common to use bars in graphs. We can easily do this by using `geom_bar`, rather than `geom_point`. Also, we can plot bars for the means, and keep showing the dots like this…(note this will be messed up, but I want to show you why).

Also look for these changes.

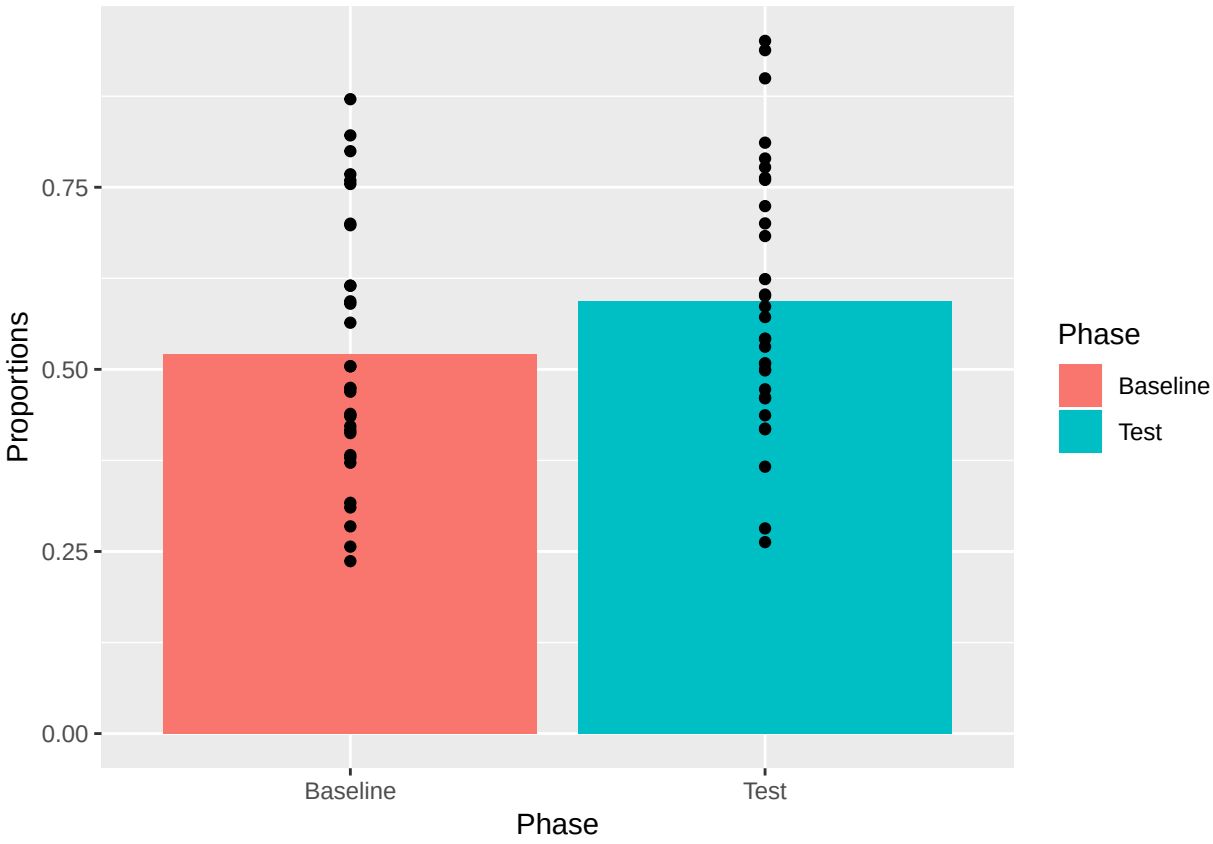
1. added `stat="identity"` Necessary for bar plot to show specific numbers
2. added `aes(fill=Phase)` Makes each bar a different color, depending on which phase it comes from

```
ggplot(plot_df, aes(x=Phase, y=Proportions))+
  geom_point()+
  geom_bar(data=mean_df, stat="identity",aes(fill=Phase))
```



Ok, we see the bars and some of the dots, but not all of them. What is going on? Remember, ggplot2 works in layers. Whatever layer you add first will be printed first in the graph, whatever layer you add second will be printed on top of the first. We put the bars on top of the dots. Let's change the order of the layers so the dot's go on top of the bars.

```
ggplot(plot_df, aes(x=Phase, y=Proportions))+
  geom_bar(data=mean_df, stat="identity",aes(fill=Phase))+
  geom_point()
```



6.4.7.5 Bar plot with error bars

So far we have only plotted the means and individual sample scores. These are useful, but neither of them give us clear visual information about our statistical test. Our paired sample t-test suggested that the mean difference between Baseline and Test was not very likely by chance. It could have happened, but wouldn't happen very often.

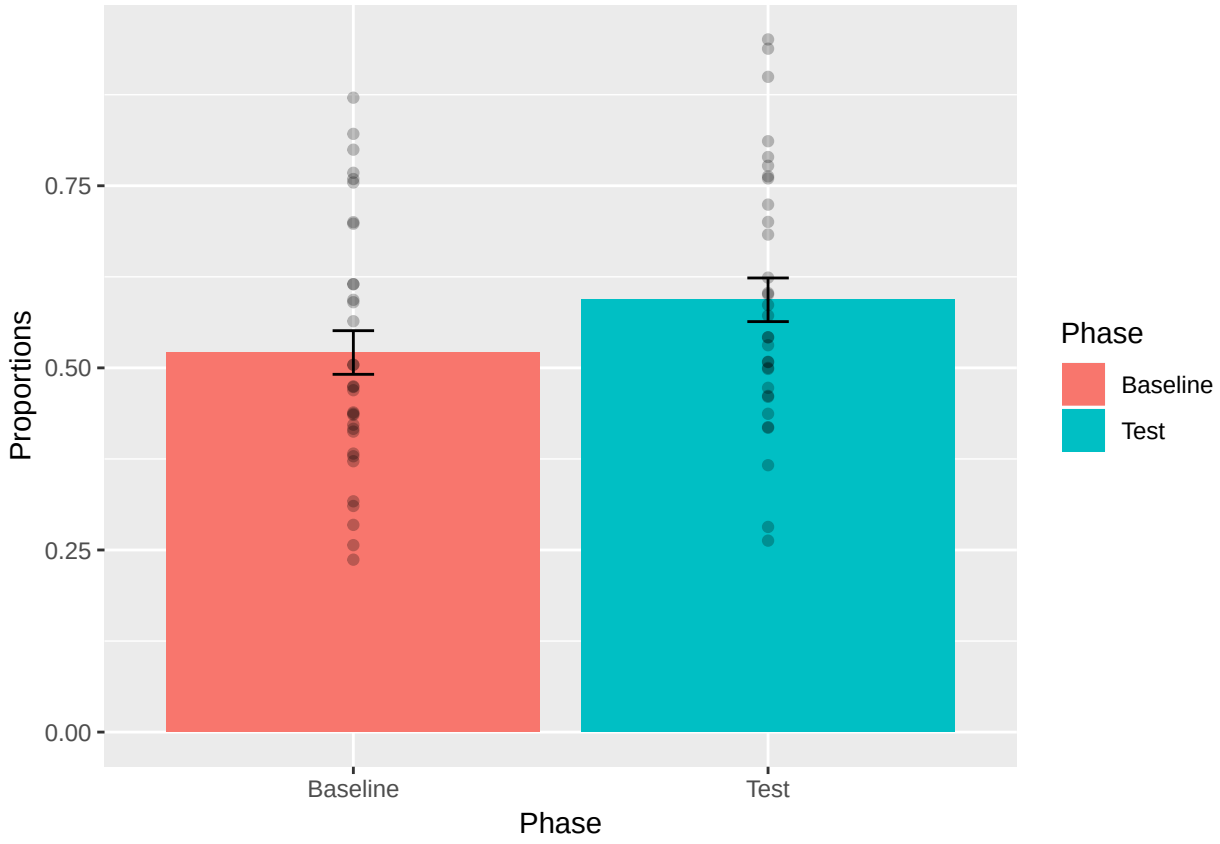
Question: Why would the standard deviation of each mean, or the standard error of each mean be inappropriate to use in this case?

Question: How would error bars based on the standard error of the mean differences aid in visual inference about the role of chance in producing the difference?

Error bars are commonly used as an aid for visual inference. The use of error bars can be a subtle nuanced issue. This is because there a different kinds of error bars that can be plotted, and each one supports different kinds of inference. In general, the error bar is supposed to represent some aspect of the variability associated with each mean. We could plot little bars that are +1 or -1 standard deviations of each mean, or we would do +1 or -1 standard errors of each mean. In the case of paired samples, neither or these error bars would be appropriate, they wouldn't reflect the variability associated with mean we are interested in. In a paired samples t-test, we are interested in the variability of the mean of the difference scores. Let's calculate the standard error of the mean (SEM) for the difference scores between Baseline and Test, and then add error bars to the plot.

```
difference_scores <- baseline-test_phase #calculate difference scores
standard_error <- sd(difference_scores)/sqrt(length(difference_scores)) #calculate SEM


ggplot(plot_df, aes(x=Phase, y=Proportions))+
```

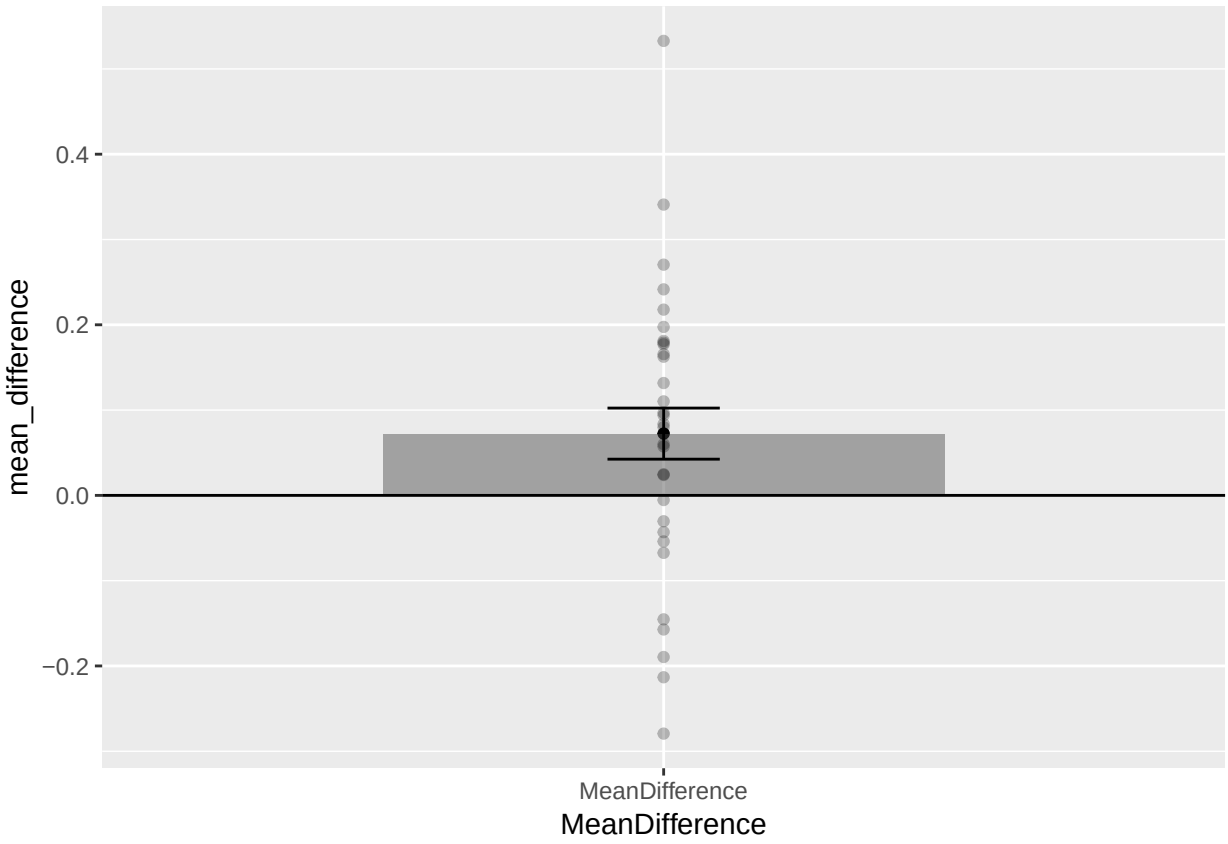
```
  geom_bar(data=mean_df, stat="identity",aes(fill=Phase))+
  geom_errorbar(data=mean_df, aes(ymin=Proportions-standard_error,
                                  ymax=Proportions+standard_error), width=.1) +
  geom_point(alpha=.25)
```



Question: What is one reason why these error bars (standard error of the mean difference between Baseline and Test) are appropriate to use. What is one reason they are not appropriate to use?

We have done something that is useful for visual inference. From the textbook, we learned that differences of about 2 standard errors of the mean are near the point where we would claim that chance is unlikely to have produced the difference. This is a rough estimate. But, we can see that the top of the error bar for Baseline is lower than the bottom of the error bar for Test, resulting in a difference greater than 2 standard error bars. So, based on this graph, we might expect the difference between conditions to be significant. We can also complain about what we have done here, we are placing the same error bars from the meand difference scores onto the means for each condition. In some sense this is misleading. The error bars are not for the depicted sample means, they are for the hidden single set of difference scores. To make this more clear, we will make a bar plot with a single bar only for the differences scores.

```
difference_scores <- test_phase-baseline #calculate difference scores
standard_error <- sd(difference_scores)/sqrt(length(difference_scores)) #calculate SEM
mean_difference <- mean(difference_scores)

qplot(x="MeanDifference", y=mean_difference)+
  geom_bar(stat="identity", width=.5, alpha=.5)+
  geom_hline(yintercept=0)+
  geom_point(aes(y=difference_scores), alpha=.25)+
  geom_errorbar(aes(ymin=mean_difference-standard_error,
                                  ymax=mean_difference+standard_error), width=.1)
```



Question: Why is is it more appropriate to put the standard error of the difference on this bar graph? What important aspects of the original results shown in the previous graph are missing from this graph?

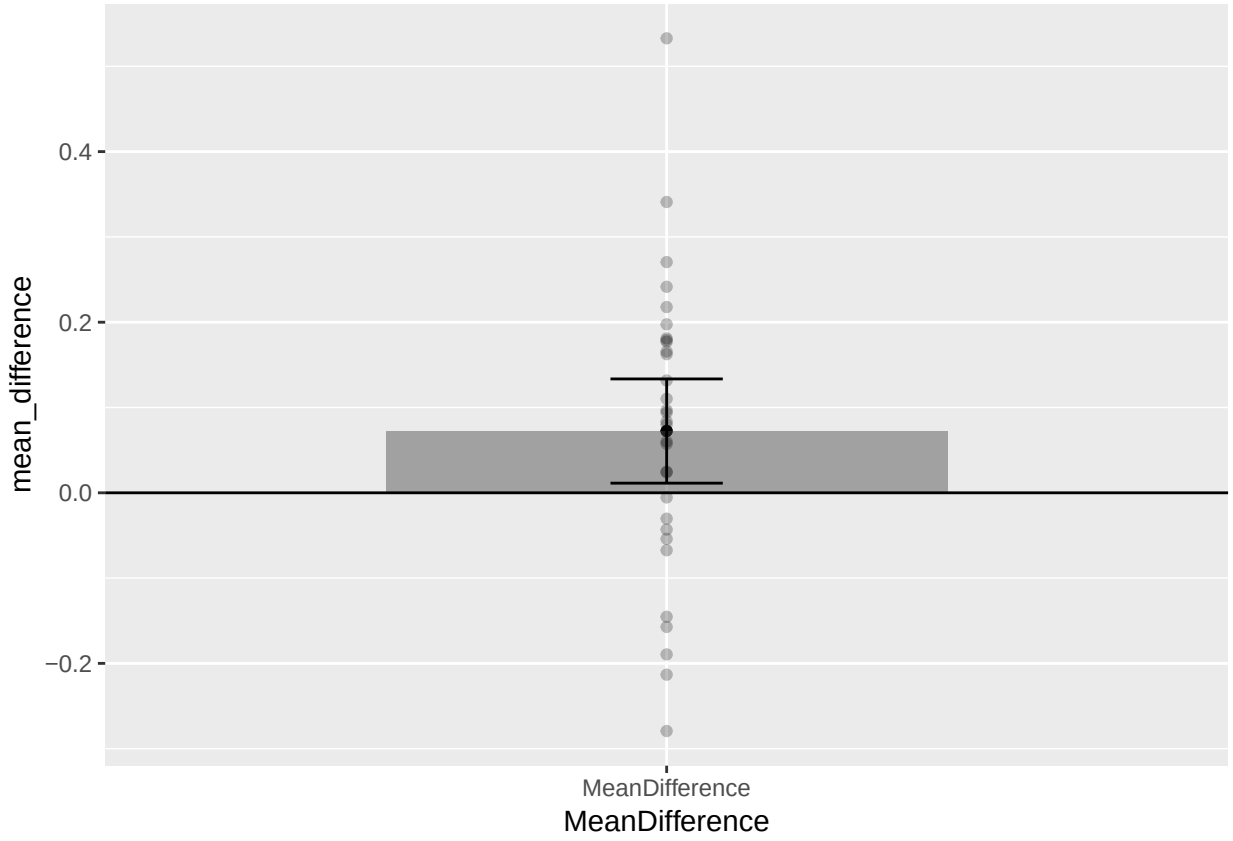
This plot is very useful too, it gives us some new information. We can see that the mean difference (test - baseline) was greater than 0. And, we are plotting the standard error of the mean differences, which are the error bars that more formally belong to this graph. Still, we are in a position to make some guesses for visual inference. The lower error bar represents only 1 SEM. It does not cross 0, but the fact that 1 SEM does not cross zero isn't important. For SEMs it's usually closer to 2. We can sort of visually guesstimate that that 2 SEMs would not cross zero, which would suggest we would obtain a small p-value. We also see each of the mean difference scores. It's very clear that they are all over the place. This means that not every infant showed a looking time preference toward the singer of the familiar song. Finally, this single bar plot misses something. It doesn't tell us what the values of the original means were.

6.4.7.6 Bar plot with confidence intervals

Confidence intervals are also often used for error bars, rather than the standard error (or other measure of variance). If we use 95% confidence intervals, then our error bars can be even more helpful for visual inference. Running the `t.test` function produces confidence interval estimates, and we can pull them out and use them for error bars.

```
t_test_results <- t.test(difference_scores)
lower_interval<- t_test_results$conf.int[1]
upper_interval<- t_test_results$conf.int[2]

qplot(x="MeanDifference", y=mean_difference)+
  geom_bar(stat="identity", width=.5, alpha=.5)+
  geom_hline(yintercept=0)+
```

```
  geom_point(aes(y=difference_scores), alpha=.25)+
  geom_errorbar(aes(ymin=lower_interval,
                                  ymax=upper_interval), width=.1)
```



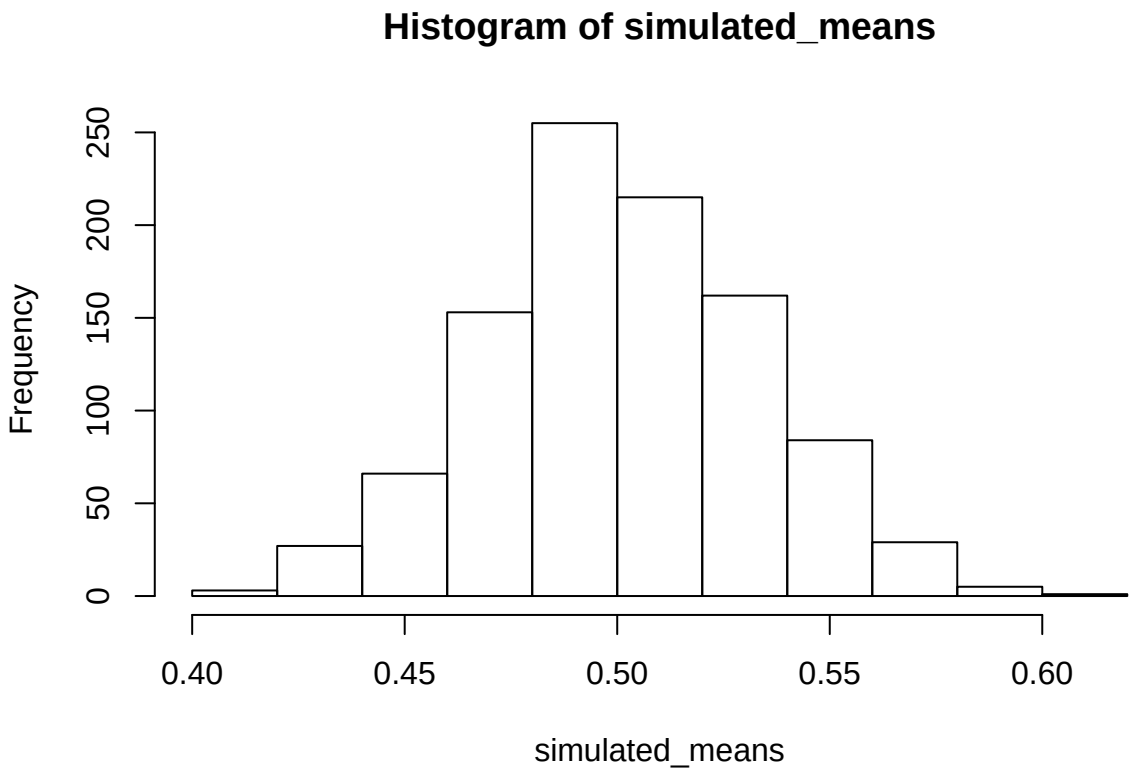
Notice that the 95% confidence intervals around the mean are wider than the SEM error bars from the previous graph. These new confidence intervals tell us that 95% of the time our sample mean will fall between the two lines. The bottom line is slightly above 0, so we can now visually see support for our statistical inference that chance was unlikely to produce the result. If chance was likely to produce the result, the horizontal line indicating 0, would be well inside the confidence interval. We can also notice that the mean difference was just barely different from chance, that lower bar is almost touching 0.

6.4.8 Data-simulation

We can do a little bit of data simulation to get a better feel for this kind of data. For example, we saw that the dots in our plots were quite variable. We might wonder about what chance can do in the current experiment. One way to do this is to estimate the standard deviation of the looking time proportions. Perhaps the best way to do that would be to an average of the standard deviation in the baseline and test_phase conditions. Then, we could simulate 32 scores from a normal distribution with mean = .5, and standard deviation equalling our mean standard deviation. We could calculate the mean of our simulated sample. And, we could do this many times, say 1000 times. Then, we could look at a histogram of our means. This will show the range of sample means we would expect just by chance. This is another way to tell whether the observed difference in this experiment in the testing phase was close or not close from being produced by chance. Take a look at the histogram. What do you think?

```
sample_sd   <- (sd(baseline)+sd(test_phase))/2

simulated_means <- length(1000)
```

```
for(i in 1:1000){
 simulated_means[i] <- mean(rnorm(32,.5, sample_sd))
}

hist(simulated_means)
```

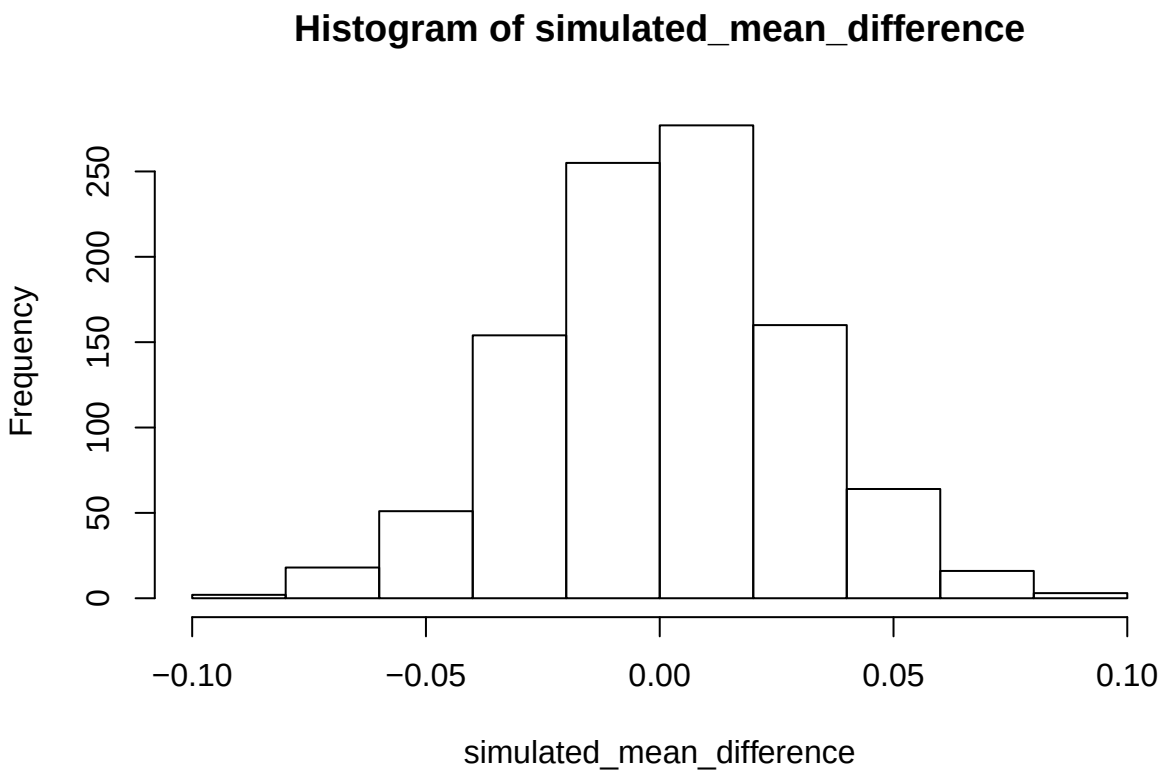


####

Simulating the mean differences

We can do the simulation slightly differently to give us a different look at chance. Above we simulated the sample means from a normal distibrution centered on .5. The experimental question of interest was whether the mean difference between the baseline and test_phase condition was different. So, we should do a simulation of the difference scores. First, we estimate the standard deviation of the difference scores, and then run the simulation with a normal distribution centered on 0 (an expected mean difference of 0). This shows roughly how often we might expect mean differences of various sizes to occur. One major limitation is that we likely had only a rough estimate of the true standard deviation of these mean differences, after all there were only 32 of them, so we should take this with a grain of salt. Nevertheless, the pattern in the simulation fits well with the observations in that we already made from the data.

```
sample_sd   <- sd(baseline-test_phase)

simulated_mean_difference <- length(1000)
for(i in 1:1000){
 simulated_mean_difference[i] <- mean(rnorm(32,0, sample_sd))
}

hist(simulated_mean_difference)
```



6.5 Excel

How to do it in Excel

6.6 SPSS

How to do it in SPSS

6.7 JAMOVI

How to do it in JAMOVI

Chapter 7

Lab 7: t-test (Independent Sample)

This lab is modified and extended from Open Stats Labs. Thanks to Open Stats Labs (Dr. Kevin P. McIntyre) for their fantastic work.

7.1 Do you come across as smarter when people read what you say or hear what you say?

7.1.1 STUDY DESCRIPTION

Imagine you were a job candidate trying to pitch your skills to a potential employer. Would you be more likely to get the job after giving a short speech describing your skills, or after writing a short speech and having a potential employer read those words? That was the question raised by Schroeder and Epley (2015).The authors predicted that a person's speech (i.e., vocal tone, cadence, and pitch) communicates information about their intellect better than their written words (even if they are the same words as in the speech).

To examine this possibility, the authors randomly assigned 39 professional recruiters for Fortune 500 companies to one of two conditions. In the audio condition, participants listened to audio recordings of a job candidate's spoken job pitch. In the transcript condition, participants read a transcription of the job candidate's pitch. After hearing or reading the pitch, the participants rated the job candidates on three dimensions: intelligence, competence, and thoughtfulness. These ratings were then averaged to create a single measure of the job candidate's intellect, with higher scores indicating the recruiters rated the candidates as higher in intellect. The participants also rated their overall impression of the job candidate (a composite of two items measuring positive and negative impressions). Finally, the participants indicated how likely they would be to recommend hiring the job candidate (0 - not at all likely, 10 - extremely likely).

What happened? Did the recruiters think job applicants were smarter when they read the transcripts, or when the heard the applicants speak? We have the data, we can find out.

7.2 Lab skills learned

1. Conduct independent samples t-tests
2. Generate figures
3. Discuss the results and implications

7.3 Important Stuff

- citation: Schroeder, J., & Epley, N. (2015). The sound of intellect: Speech reveals a thoughtful mind, increasing a job candidate's appeal. Psychological Science, 26, 877-891.
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format

7.4 R

7.4.1 Load the data

Remember that any line with a # makes a comment and the code does not run. Below is how to load the .csv data from the online repository, or from a local file (you need to change the file path to where the local file is, if you downloaded it). The data contains all of the measures and conditions from Experiment 4.

```
library(data.table)
# load from github repo
#all_data <- fread("https://raw.githubusercontent.com/CrumpLab/statisticsLab/master/data/SchroederEpley

all_data <- fread("Data/SchroederEpley2015data.csv") # load from file on computer
```

7.4.2 Inspect data frame

This will give you a big picture of the data frame. Click the button to view it in your browser, then take a look to see what is in it.

```
library(summarytools)
view(dfSummary(all_data))
```

7.4.3 Find the data you need

This time the data comes prefiltered for us. The authors ran lots of experiments, but we only have the data from Experiment 4. This is great, we don't need to subset the data frame to find all of the data that we need. But, we do still need to understand what data we want to analyze. Let's start with identify the column that codes the experimental conditions for whether or not the evaluator read a transcript or heard the interview.

7.4.3.1 Condition variable

Lucky for us, the condition variable is called `CONDITION`! Let's take a look. We printed it out just by writing down `all_data$CONDITION`. There 0s and 1s for each condition (audio vs. transcript). But which one is which? This isn't clear from the data, and it isn't clear from the paper, or from the repository. We have to do some guess work. I went ahead and computed the means for the Intellect_rating between each condition, and then compared those to the graph in the paper for E4. It looks like 1 = audio condition, and 0 = transcript condition.

```
all_data$CONDITION
```

```
##  [1] 1 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 1 1 0
## [36] 1 0 1 1
```

```
aggregate(Intellect_Rating~CONDITION,all_data,mean)
```

```
##   CONDITION Intellect_Rating
## 1         0         3.648148
## 2         1         5.634921
```

Let's use words instead of 0s and 1s to refer to our experimental conditions. To do this, we will chance the values of 0 and 1, to the words `transcript` and `audio`. We can do this in two steps. First we convert the CONDITION column to a factor. This will automatically turn the 0s and 1s into strings (not numbers, text). Factors have an internal variable for the names of the levels, which will be 0 and 1. We can simply change the level names to transcript and audio.

```
all_data$CONDITION <- as.factor(all_data$CONDITION)
levels(all_data$CONDITION) <- c("transcript","audio")
```

Now if you look at the `all_data` variable, you will see the words transcript and audio, where 0s and 1s used to be.

7.4.3.2 Dependent Measures

Next it's time to find the depende measure columns. The graph from the paper shows three different measures in each condition. These included `Intellect`, `General Impression`, and `Hiring Likelihood`. Every evaluator (either given a transcript or audio recording of the interview) gave ratings on a scale of 1 to 10 for each of those concepts. It's not immediately clear which columns in `all_data` correspond to those three measures. There are lots of different measures that could be the ones they reported. It turns out the relevant ones are called

1. `Intellect_Rating`
2. `Impression_Rating`
3. `Hire_Rating`

In this tutorial we are going to walk through doing an independent samples t-test for the first measure, `Intellect_Rating`. You can follow these same steps to complete the same kind of t-test for the other two variables.

7.4.4 Look at the dependent variable.

Question: Why do we always want to look at the data?

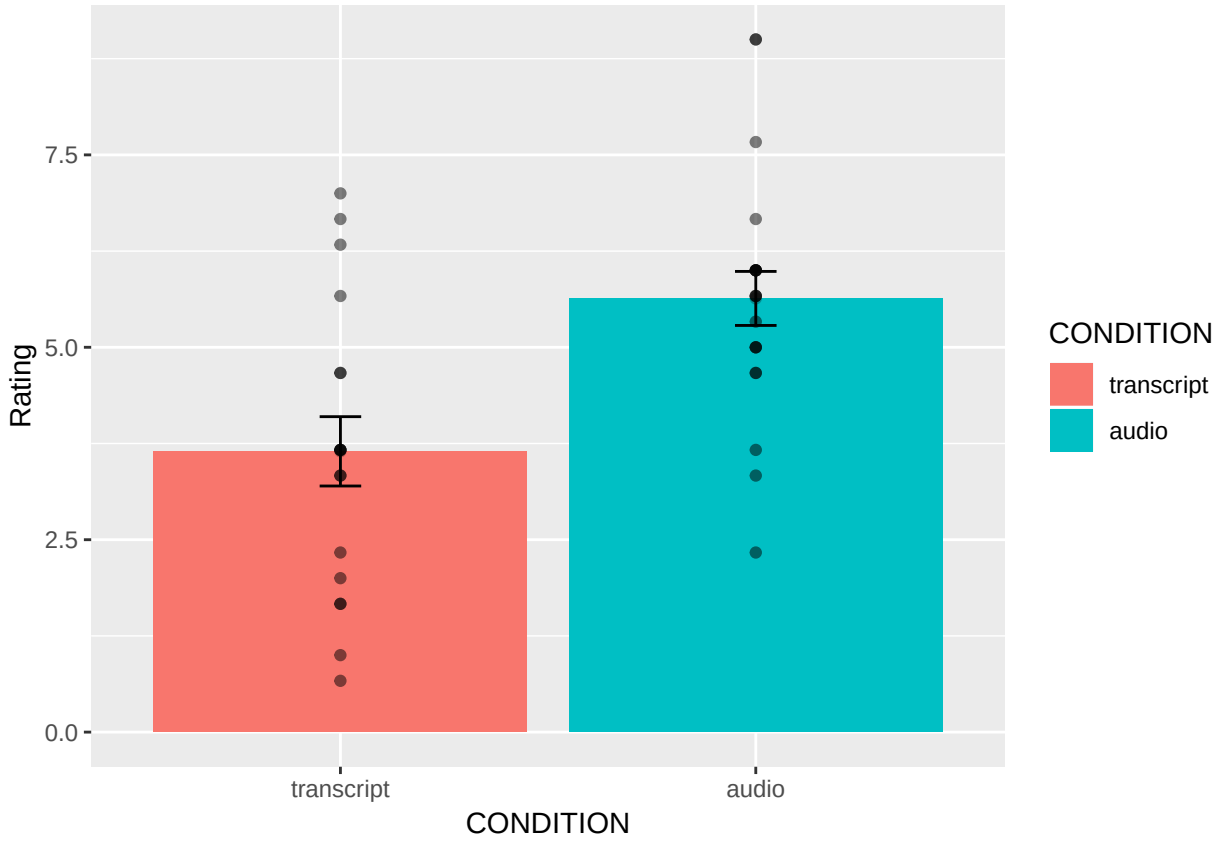
What is the first thing we do before even considering an inferential test? Look at the data. Always look at the data. We could make a dot plot or histogram of the data from the `Intellect_ratings`. But, from our last lab we already learned how to make graphs showing most of the information we would want to look at. For example, we could make a bar graph that has the means for each condition (transcript vs. audio), standard errors of the mean and the actual scores as little dots. This would be great to look at it. Not only will it tell us if there are really weird numbers in the data (who knows maybe the data file is corrupted, you need to look), it will also give us strong intuitions about what to expect for the t-test.

We can plot each score as a dot using the `all_data` dataframe. If we want to add on a layer for the sample means, and for the sample standard errors, we have to compute those and put them in a new dataframe first. Then we use both dataframes with ggplot to plot all of the information.

We will use `dplyr` to quickly get the means and the standard errors and put them in a new data frame called `decriptive_df`.

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

```
# get means and SEs
descriptive_df <- all_data %>%
                    group_by(CONDITION) %>%
                    summarise(means= mean(Intellect_Rating),
                              SEs = sd(Intellect_Rating)/sqrt(length(Intellect_Rating)))

# Make the plot
ggplot(descriptive_df, aes(x=CONDITION, y=means))+
  geom_bar(stat="identity", aes(fill=CONDITION))+ # add means
  geom_errorbar(aes(ymin=means-SEs,               # add error bars
                    ymax=means+SEs), width=.1) +
  geom_point(data=all_data, aes(x=CONDITION, y=Intellect_Rating), alpha=.5)+
  geom_point(alpha=.25)+
  ylab("Rating")
```



This plot is very useful. First, we can see the numbers in our dependent measure are behaving sensibly. We know that the numbers have to be between 1-10, because those were the only options in the scale. If we found numbers bigger or smaller, we would know something was wrong. Checking for things that are obvsiouly wrong in the data is one reason why we always look at first. We are checking for obvious errors. There are other ways to check to, but looking is fast and easy.

Question: Why are the standard errors of each sample an appropriate thing to use for error bars?

Now that you can see the patterns in the data, you should form an intuition about how the independent samples t-test will turn out. You can see how big the error bars (+1/-1 standard error of each sample man). The t-test will tell us whether the observed difference (or greater) is likely due to chance. Should we find a big t-value or a small t-value? Should we find a big p-value or a small t-value. If you understand how

t-values and p-values work, the answer should be very clear from the graph. You should already know how the t-test will turn out before you run it. Running it will confirm what you already suspect to be true.

7.4.5 Conduct Independent samples t-test

Question: Why are we conducting an independent samples t-test, and not a one-sample or paired samples t-test?

We use the very same `t.test` function that we used last time to conduct a t-test. The only difference is that we don't tell the R to use a paired sample t-test. We leave the `paired=TRUE` statment out, and R automatically knows we want to do an independent samples t-test. Remember to set the `var.equal=TRUE`, otherwise R will compute a different version of the t-test.

You can use different syntax to run the t-test. Because our data is already in a data frame we can use this syntax.

```
t.test(Intellect_Rating~CONDITION, data=all_data, var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Intellect_Rating by CONDITION
## t = -3.5259, df = 37, p-value = 0.001144
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.1284798 -0.8450652
## sample estimates:
## mean in group transcript      mean in group audio
##                 3.648148                 5.634921
```

The `t.test` function also will work on two variables, not in a data frame. For example, the following does the same thing. But, it's harder to read, and the means are described in terms of X and Y, not terms of `transcript` and `audio`, like the report above.

```
t.test(all_data[all_data$CONDITION=='transcript',]$Intellect_Rating,
       all_data[all_data$CONDITION=='audio',]$Intellect_Rating,
       var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  all_data[all_data$CONDITION == "transcript", ]$Intellect_Rating and all_data[all_data$CONDITI
## t = -3.5259, df = 37, p-value = 0.001144
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.1284798 -0.8450652
## sample estimates:
## mean of x mean of y
##  3.648148  5.634921
```

Question: What conclusions do we draw from the t-test? Based on these results, if you were being evaluated for a job interview, would you rather have the evaluator read a transcript of your interview or listen to an audio recording?

So, now we have the t-test. It shows the t-value, the p-value, and the means for each group. You can double-check with the paper to see if we found the same results as reported by the authors.

7.4.6 Remaining ratings

Now, you should use what you have learned to anlayse the last two ratings for the dependent variables `Impression_Rating`, and `Hire_Rating`. Remember to plot the data for each, and conduct a t-test for each. Then compare what you found to the original article. What did you find, and what do the results mean?

7.4.7 Reconstructing the graph from the paper

The results from Experimen4 4 in the paper plot the means and error bars (+1 / -1 SEM) for all three dependent measures, for both experimental conditions. We can do this in ggplot using the data. We will have to make a couple changes to the dataframe. But, it won't be too hard. What we need to do is make a fully long form data frame. Remember a long form data frame has one row per dependent measure.

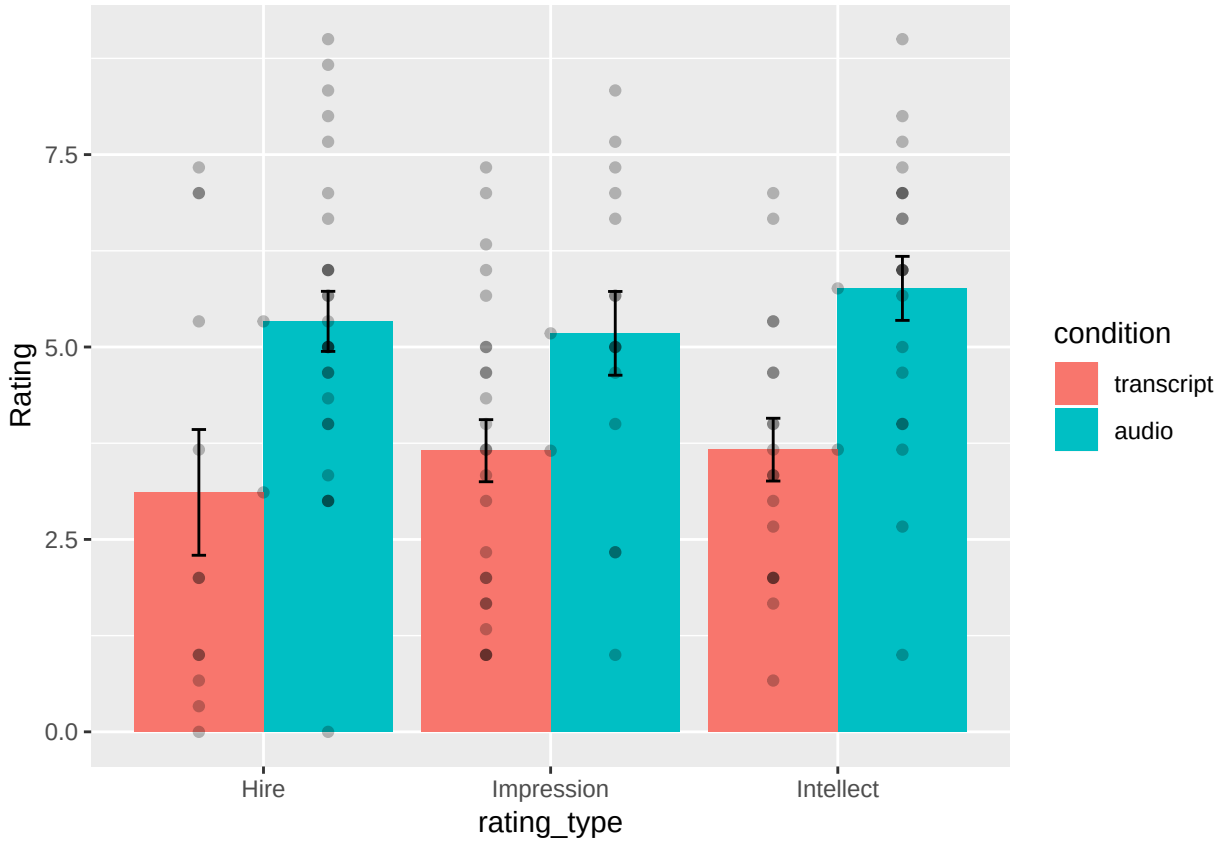
The `all_data` frame is partly long and partly wide. If we are only interested in one dependent measure, then it is a long data frame for that measure. For example, example if we are only interested in plotting `Intellect_Rating`, then we already have one observation of that dependent measure for each row. But, in the other columns, the dependent measures for `Impression_Rating` and `Hire_Rating` are in the same rows.

Before continuing, it is very much worth mentioning that this part of data analysis happens alot, and it is kind of annoying. I call it the rubix cube problem, because we need to "rotate" and transform the format of the data to accomplish different kinds of analysis goals. It's good to be able to know how to do this. This problem occurs all of the time, can occur for any software package. It's a good thing you are learning R, because we can do these things easily in R. They are not often so easy to do without a computer programming language like R. The worst thing to do is transform the data by hand. That really sucks. Believe me you don't want to do it. Why? Because you will make mistakes, and you will mess up the data, then you will mess up your analysis. And, you won't be able to find your mistakes, and it will take you ages to correct them. That sucks.

There's more than one way to transform data in R. For example the `cast` and `melt` functions do this kind of thing. You can look those up. In this example we will not use those functions. Instead we will show some steps to build the required dataframe one step at a time.

```
# repeat CONDITION column three times

condition <- rep(all_data$CONDITION,3)

# make a ratings variable with all three ratings in one variable

ratings <- c(all_data$Intellect_Rating,
             all_data$Impression_Rating,
             all_data$Hire_Rating)

# make a new factor variable with the names of the ratings
# need to repeat each level name the appropriate number of times

num_to_repeat <- length(all_data$CONDITION)

rating_type <- rep(c("Intellect","Impression","Hire"),num_to_repeat)

# put the new variables into a data frame

plot_all <- data.frame(condition,rating_type,ratings)

# Get the means and standard errors for each rating by condition
```

```
descriptive_all <- plot_all %>%
                    group_by(condition,rating_type) %>%
                    summarise(means= mean(ratings),
                              SEs = sd(ratings)/sqrt(length(ratings)))

# Make the plot

ggplot(descriptive_all, aes(x=rating_type, y=means, group=condition))+
  geom_bar(stat="identity", aes(fill=condition), position='dodge')+
  geom_errorbar(aes(ymin=means-SEs,
                    ymax=means+SEs),
                width=.1,
                position = position_dodge(width=.9)) +
  geom_point(data=plot_all, aes(x=rating_type,
                                y=ratings,
                                group=condition),
             alpha=.25,
             position = position_dodge(width=.9))+
  geom_point(alpha=.25)+
  ylab("Rating")
```



Well, we didn't make the exact graph. We have the bars, the error bars, and we added the individual scores because they are useful to look at. Otherwise, it's the same graph (except the the ordering of bars is determined alphabetically here. We change that in ggplot, but we won't do that today.)

7.5 Excel

How to do it in Excel

7.6 SPSS

How to do it in SPSS

7.7 JAMOVI

How to do it in JAMOVI

Chapter 8

Lab 8: One-way ANOVA

This lab is modified and extended from Open Stats Labs. Thanks to Open Stats Labs (Dr. Kevin P. McIntyre) for their fantastic work.

8.1 How to not think about bad memories by playing Tetris

This lab activity uses the open data from Experiment 2 of James et al. (2015) to teach one-way ANOVA with planned comparisons. Results of the activity provided below should exactly reproduce the results described in the paper.

8.1.1 STUDY DESCRIPTION

Following traumatic experiences, some people have flashbacks, which are also called "intrusive memories"" and are characterized by involuntary images of aspects of the traumatic event. Although people often try to simply forget traumatic memories, this approach is not very effective. Instead, previous research suggests that a better approach may be to try to change aspects of the memory after it is formed. For example, some research shows that traumatic memories can be altered and weakened to the point that they are no longer intrusive.

Because intrusive memories of trauma are often visual in nature, James and colleagues (2015) sought to explore whether completing a visuospatial task (e.g., Tetris) after a memory was formed would interfere with the storage of that memory, and thereby reduce the frequency of subsequent intrusions. They hypothesized that only participants who complete a visuo-spatial task after reactivation of the traumatic memories would experience a reduction in intrusive memories. In comparison, simply completing a visuo-spatial task (without reactivation) or reactivation (without a visuo-spatial task), would not reduce the occurrence intrusive memories.

In other words, if you play Tetris shortly after you were remembering bad memories, playing Tetris might weaken those memories, which could cause you experience those kinds of intrusive memories less often in the future.

8.1.2 Study Methods

To test their hypothesis, the authors conducted an experiment (N = 72, n = 18 per condition). The procedure is summarized as follows:

Trauma Film: All participants viewed a series of video clips of graphic violence (e.g., a person getting hit by a van while using his phone as he crosses the road) as a way to create memories that should become intrusive memories. Participants then went home and recorded the number of intrusive memories they experienced over the next 24 hours. Because this is before the experimental manipulations, all groups were predicted to have an equal occurrence of intrusive memories during the first 24-hours (called Day 0).

Experimental Task: After this 24-hour period, the participants returned to the lab and completed the experimental task. The experimenters randomly assigned participants to ONE of the following conditions:

1. No-task control: These participants completed a 10-minute music filler task.
2. Reactivation + Tetris: These participants were shown a series of images from the trauma film to reactivate the traumatic memories (i.e., reactivation task). After a 10-minute music filler task, participants played the video game Tetris for 12 minutes.
3. Tetris Only: These participants played Tetris for 12 minutes, but did not complete the reactivation task.
4. Reactivation Only: These participants completed the reactivation task, but did not play Tetris.

Intrusive Memories: All participants were asked to record the number of intrusive memories that they experienced over the next seven days (Days 1 to 7).

After the seven days had passed, participants completed an Intrusion-Provocation Task, in which they were shown blurred images from the trauma film and asked to indicate whether the blurred image triggered an intrusive memory.

8.2 Lab Skills Learned

8.3 Important Stuff

- citation: James, E. L., Bonsall, M. B., Hoppitt, L., Tunbridge, E. M., Geddes, J. R., Milton, A. L., & Holmes, E. A. (2015). Computer game play reduces intrusive memories of experimental trauma via re-consolidation-update mechanisms. Psychological Science, 26, 1201-1215.
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format

8.4 R

8.4.1 Load the data

Remember that any line with a # makes a comment and the code does not run. Below is how to load the .csv data from the online repository, or from a local file (you need to change the file path to where the local file is, if you downloaded it). The data contains all of the measures and conditions from Experiment 2 in the paper.

```
library(data.table)
#fread("https://raw.githubusercontent.com/CrumpLab/statisticsLab/master/data/Jamesetal2015Experiment2.c
all_data <- fread("data/Jamesetal2015Experiment2.csv")
```

8.4.2 Inspect the dataframe

This will give you a big picture of the data frame. Click the button to view it in your browser, then take a look to see what is in it.

```
library(summarytools)
view(dfSummary(all_data))
```

8.4.3 Get the data you need

Again we have only the data from Experiment 2, so we don't need to get rid of any rows of the data. But we do need look at the columns to see how the independent variable and dependent variables were coded.

8.4.3.1 The independent variable

There was one important independent variable, and it had four levels. The first column in the data frame is called condition, and it has four levels. The levels are 1, 2, 3, and 4 in the data-frame. These will correspond to the four levels in shown in figure:

- No-task control
- Reactivation Plus Tetris
- Tetris only
- Reactivation only

Each of these refer to what subjects did after they watched the traumatic film. But, which of these correspond to the numbers 1 to 4 in the data-frame? It turns out the are in the right order, and 1 to 4 refer to:

1. No-task control
2. Reactivation Plus Tetris
3. Tetris only
4. Reactivation only

Let's do ourselves a favor and rename the levels of the `Condition` column with words so we know what they refer to. First, convert the `Condition` column to a factor, then rename the levels. Then.

```
all_data$Condition <- as.factor(all_data$Condition)
levels(all_data$Condition) <- c("Control",
                                "Reactivation+Tetris",
                                "Tetris_only",
                                "Reactivation_only")
```

8.4.3.2 The dependent variable

The authors showed two figures, one where they analysed intrusive memory frequency as the mean for the week, and the other where the used intrusive memory frequency on Day 7. In this tutorial, we will do the steps to run an ANOVA on the mean for the week data, and you will do follow these steps to run another ANOVA on the Day 7 data.

The mean for the week data for each subject is apparently coded in the column `Days_One_to_Seven_Number_of_Intrusions`.

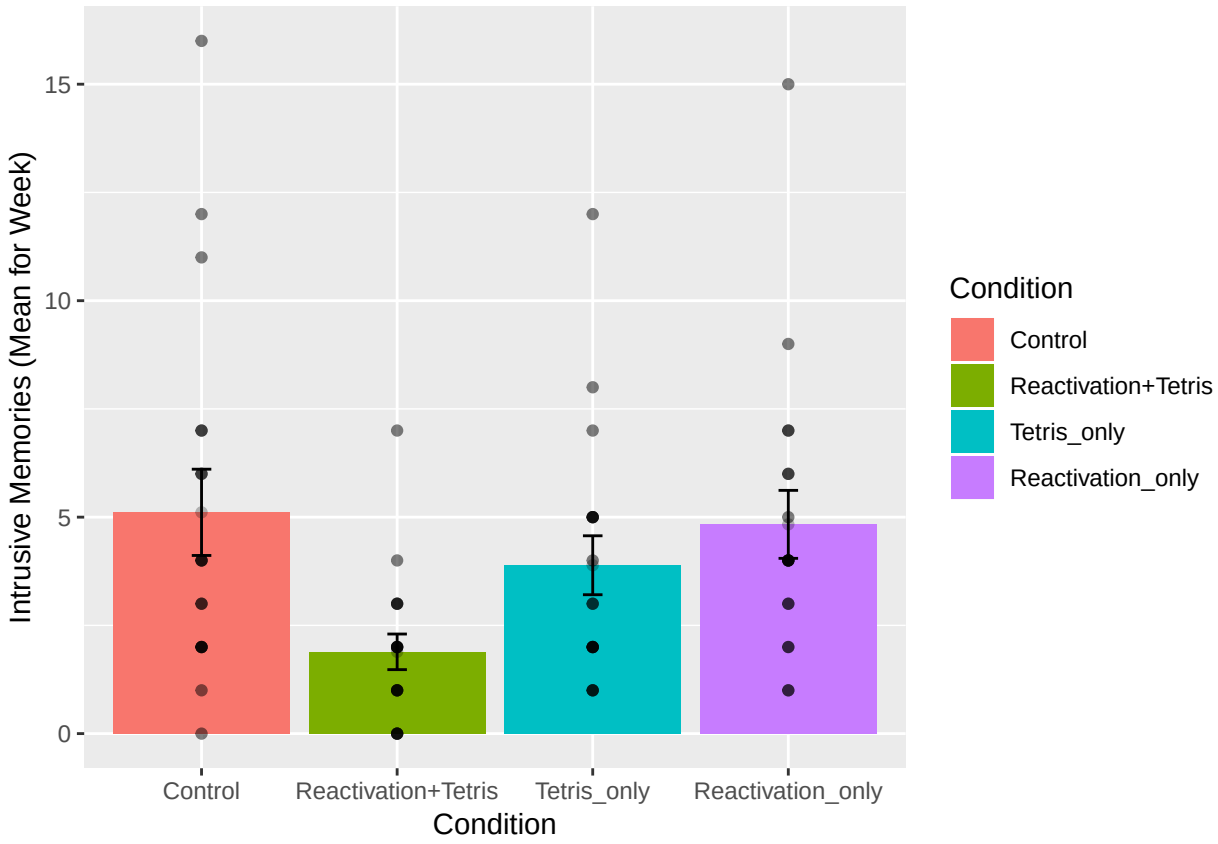
8.4.4 Look at the data

Remember before we do any analysis, we always want to "look" at the data. This first pass let's us know if the data "look right". For example, the data file could be messed up and maybe there aren't any numbers

there, or maybe the numbers are just too weird.

In the last two labs, we saw how to show all the data we are interested in looking at in one go. We can do the same things here. For example, in one little piece of code, we can display the means in each condition, the standard errors of the mean, and the individual scores for each subject in each condition. This is a lot to look at, but it's everything we want to look at for starters all in the same place. Looking at the data this way will also give us intuitions about what our ANOVA will tell us before we run the ANOVA. This is helpful for determining whether the results of your ANOVA are accurate or not. Let's do it...also notice that the code is very similar to what we did for the independent t-test. In fact, all I did was copy and paste that code right here, and edited it a little bit. This is really fast, and shows the efficiency of doing things this way.

```
library(dplyr)

library(ggplot2)

# get means and SEs
descriptive_df <- all_data %>%
                    group_by(Condition) %>%
                    summarise(means= mean(Days_One_to_Seven_Number_of_Intrusions),
                              SEs = sd(Days_One_to_Seven_Number_of_Intrusions)/sqrt(length(Days_One_to_S

# Make the plot
ggplot(descriptive_df, aes(x=Condition, y=means))+
  geom_bar(stat="identity", aes(fill=Condition))+ # add means
  geom_errorbar(aes(ymin=means-SEs,               # add error bars
                    ymax=means+SEs), width=.1) +
  geom_point(data=all_data, aes(x=Condition, y=Days_One_to_Seven_Number_of_Intrusions), alpha=.5)+
  geom_point(alpha=.25)+
  ylab("Intrusive Memories (Mean for Week)")
```



There we have it. A nice graph showing us everything. We can see there was a lot of differences at the level of individual subjects. Some subjects had a mean of 15 intrusive memories for the week. Some had 0. Lots of differences.

And, to be completely honest, this data seems a bit weird to me. It might not be weird, but it might be. The wording in the manuscript is that this data is the mean frequency of intrusive memories over the week. For the people who had 15, this means that every day they had on average 15 intrusive memories. Some people had on average 0 per day. This could be the data. But, it also seems like the data might just be frequency counts and not means at all. For example, the data could be that some people had a total of 15 intrusive over the week. The data might make more sense if these were frequency counts. Otherwise the differences between people are truly very large. For example the person who had an average of 15 intrusive memories per week, must have had 15*7 = 105 intrusive memories, which is a lot more than zero. In any case, this kind of wondering about the data is what happens when you start to notice how numbers work. It's useful develop your data sense.

Let's move on to the ANOVA. By looking at the above graph do you have an intuition about what the ANOVA will tell us? I do, it should easily tell us that we are going to get an F-value that is bigger than one, and a p-value that is probably smaller than .05? How do I know this? I looked at the error bars, which represent the standard errors of the mean. If we look across conditions we can see that that the error bars don't always overlap. This suggests there are differences in the data that don't happen very often by chance. So, we expect a smallish p-value. Why do you think I expect the F-value to be greater than 1? If you can answer this question with a justification and explanation of how F works, pat yourself on the back!

8.4.5 Conduct the ANOVA

Conducting an ANOVA in R is pretty easy. It's one line of code, just like the t-test.

It's worth knowing that there are a few different packages out there to do an ANOVA, for example Mike Lawrence's `ezANOVA` package is pretty popular.

For this tutorial we'll show you how to run the ANOVA using the `aov` function from base R (comes pre-installed). This is pretty "easy" too. I put easy in quotes because the first time I saw this, it was not easy for me. Here's a brief digression for me to tell you that I feel your pain. I was first introduced to R by Patrick Bennett (a professor at McMaster University, where I attended graduate school). Pat, like I am doing know, forced us to use R in his statistics class. I remember having zero clue about so many things, and was often very frustrated. So, I imagine some of you are quite frustrated too. I was luck, like some of you, to have had some previous experience with other programming languages, so I was familiar with what R might be doing. What I was most frustrated with, was learning how to tell R what to do. In other words, I didn't know how to write the commands properly. I didn't understand what we call the **syntax** of R.

This was back many years ago now, well before there was so many helpful examples on Google with working code, showing you how the syntax works. All we had was the R help files, which were a total mystery to me. If you want to see the help file for `aov`, just type `?aov()` into the console, and press enter. You will see an "explanation" of how the `aov` function is supposed to work. You can use the same trick for any R function, like this `?name_of_function()`. To be clear, you have to replace the letters `name_of_function`, with the name of the function. Some of you think that might be super obvious, but that is the kind of thing I did not think was obvious. So, when I read the help file for how to use the `aov` function, that is to learn what to put where, I didn't feel like it was showing me what I needed to do. Usually, at the bottom of the help file, there are some examples, and these are helpful, but sometimes they are missing the example you need, and you are expected to generalize your knowledge of how `aov` works, to make it work for your problem. This is a catch-22 because if you don't know how it works, you can't generalize. IMO, you need a lot of examples of things that work.

So, with that digression, I'm going to explain the syntax for the aov function. It looks like this:

```
aov(DV ~ IV, dataframe)
```

That probably looks really foreign. Let me explain. First you need write `aov()`. `aov` is the name of the function, followed by the brackets. The brackets are a sandwich. Sandwiches have a top and a bottom, and they enclose the things you put in the sandwich. We then put things inside the `()`, in specific ways to make the `aov` sandwich. The `DV` stands for the name of the dependent variable in the data frame. For us, this will be `Days_One_to_Seven_Number_of_Intrusions`. So, when we add that, our function will look like:

```
aov(Days_One_to_Seven_Number_of_Intrusions ~ IV, dataframe)
```

Next, the `~` (tilda) stands for the word 'by'. We are saying we want to analyse the Dependent variable **by** the conditions of the independent variable.

The `IV` stands for the name of the column that is your independent variable. Our's is called `Condition`. Adding that in, our formula looks like:

```
aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, dataframe)
```

Finally, the last part is the name of the data-frame we are using. The `aov` function only works on long-form data, where each score on the dependent variable has it's own row. Our's is already set up this way! The name of our data-frame is `all_data`. We add that in, and it looks like:

```
aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, all_data)
```

In English, this means, do an ANOVA on the dependent variable as a function of the independent variable, and use the data in my data frame.

This is it for now. The `aov` function is very flexible because you can define different kinds of formulas (the `DV ~ IV` part). We'll see other examples in later chapters. For now, this is all we need. Also, what's cool, is that this will work for any single IV with any number of levels (conditions) 2 or greater. Fun. Let's see it in action.

```
aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, all_data)
```

```
## Call:
##    aov(formula = Days_One_to_Seven_Number_of_Intrusions ~ Condition,
##     data = all_data)
##
## Terms:
##                 Condition Residuals
## Sum of Squares   114.8194  685.8333
## Deg. of Freedom         3        68
##
## Residual standard error: 3.175812
## Estimated effects may be unbalanced
```

What is this garbage? I don't see an ANOVA table, what is this? You are seeing the raw print out of the aov function. Clearly, this is not helpful, it's not what we want to see.

Fortunately, R comes with another function called `summary`. What it does is summarize the results of functions like `aov`, and prints them out in a nicer way. Let's see the summary function do it's thing:

```
summary(aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, all_data))
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## Condition    3  114.8   38.27   3.795 0.0141 *
## Residuals   68  685.8   10.09
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Alright, now we have an ANOVA table we can look at. However, it still looks ugly, at least to me. When you are working inside an R Markdown document, you have some more options to make it look nice. We can use the `kable` and `xtable` function together, like this.

```
library(xtable)
aov_out<-aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, all_data)
summary_out<-summary(aov_out)

knitr::kable(xtable(summary_out))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Condition	3	114.8194	38.27315	3.794762	0.0140858
Residuals	68	685.8333	10.08578	NA	NA

Now we see a nicer print out. Especially if we `knit` the document into a webpage.

8.4.6 Reporting the ANOVA results

Refer tot the textbook on ANOVAs for a deeper discussion of all the things in the ANOVA table. We'll remind about some of those things here.

First, let's look at how we might report the results. There are three very important parts to this.

1. Saying what test you did to what data
2. Reporting the inferential statistics
3. Reporting the pattern in the means

Here is part 1, we need to say what data we used, and what kind of test we used on that data:

> We submitted the mean intrusive memories for the week from each subject in each condition to a one-factor betwee-subjects ANOVA, with Intervention type (No-task control, Reactivation Plus tetris, Tetris only, Reactivation only) as the sole independent variable.

Part 2 is saying what the results of the test were. Specifically, we report the values from the inferential test (see the textbook for why these values). Also, you should be able to answer this question: why do we report the values that we do?

> We found a main effect of Intervention type, F(3, 68) = 3.79, MSE = 10.09, p = 0.014.

Part 3 is saying what the pattern was in the means. Remember, that in the ANOVA, a significant effect refers to the global variation among the means. In other words, we can say that there are some differences between the means, but we can't specifically say which pairs of means are different, or which groups of means are different from one another. How can we report this, where are the means? In fact, we already found them when we plotted the data earlier. So, we can copy paste that code, and print out the means, rather than the figure:

```
# get means and SEs
descriptive_df <- all_data %>%
                    group_by(Condition) %>%
                    summarise(means= mean(Days_One_to_Seven_Number_of_Intrusions),
                              SEs = sd(Days_One_to_Seven_Number_of_Intrusions)/sqrt(length(Days_One_to_S

knitr::kable(descriptive_df)
```

Condition	means	SEs
Control	5.111111	0.9963623
Reactivation+Tetris	1.888889	0.4113495
Tetris_only	3.888889	0.6806806
Reactivation_only	4.833333	0.7848650

No we have to use a sentence to describe these means.

> Refer to table 1 for the means and standard errors of the mean in each condition

or,

> Mean intrusive memories per week were 5.11 (SE = .99); 1.89 (SE = .41); 3.89 (SE = .68); and 4.83 (SE= .78), in the Control, Reaction plus Tetris, Tetris Only, and Reactivation only conditions, respectively

Ugh, what a mouthful. Be mindful of how you write results. The above is not helpful because you see a list of numbers, and then a list of conditions, and the reader has to do a lot of work to keep everything straight. I like the table option.

I also like this other kind of option:

> Mean intrusive memories were significantly different between the Control (M = 5.11, SE = .99), Reactivation plus Tetris (M = 3.89, SE = .68), Tetris only (M= 3.89, SE = .68), and Reactivation only (M = 4.83, .78) conditions.

That's a little bit better. Let's put it all in one place to see what it look like:

> We submitted the mean intrusive memories for the week from each subject in each condition to a one-factor betwee-subjects ANOVA, with Intervention type (No-task control, Reactivation Plus tetris, Tetris only, Reactivation only) as the sole independent variable. We found a main effect of Intervention type, F(3, 68) = 3.79, MSE = 10.09, p = 0.014. Mean intrusive memories were significantly different between the Control (M = 5.11, SE = .99), Reactivation plus Tetris (M = 3.89, SE = .68), Tetris only (M= 3.89, SE = .68), and Reactivation only (M = 4.83, .78) conditions.

8.4.7 Individual comparisons

This next part is complicated, so we intentionally simplify it. There are these things (remember from the textbook), called comparisons. We use them to compare differences between specific conditions of interest. That part isn't very complicated. You just pick the things you want to compare, and compare them.

What is complicated is exactly what you "should" do to make the comparison. It turns out there are lots of recommendations and disagreements about what you should do. There are also lots of tests that you can do, so you have a lot of options. We are not going to show you here all of the tests. This is beyond the scope of what we are trying to teach you. Instead, we will use tests that you already know, the t-test for independent samples from the last lab. It will do the job for this data.

We think that before you can make good decisions about the kind of comparison test you want to use, you have to have a solid understanding of **what you are comparing** and **what you are not comparing** when you use different tests. We think this understanding is more important than what test you use. More important, is that you know what means you want to compare. In this case, we will talk about what means we want to compare, and then just do a t-test.

8.4.7.1 What did the ANOVA tell us

Remember, the ANOVA we conducted is termed the **omnibus** test in the textbook. What means was it comparing? It wasn't comparing specific means. It was asking a kind of blind and very general question: Are any of these means different. Our answer was yes. Our next question is: What means were different from what other means? The ANOVA doesn't know the answer to this question. It just says I don't know…

8.4.7.2 Comparing specific means and the experimental question

Notice there are 4 means to compare. So, there are (4-1)! = 3! = 3x2x1 = 6 total different comparisons. The ! stands for factorial. What's more important is recognizing that when you have more than two conditions (where you can only make one comparison, A vs. B), you get increasingly more comparisons. For four conditions, A, B, C, D, you get six comparisons, they are:

`AB, AC, AD, BC, BD, and CD` where each letter pair refers to A compared to B (AB), A compared to C (AC), and so on.

Do we actually care about all of these comparisons? Perhaps. What was the point of the experiment? Remember, the experiment was asking a questions, that's why they set-up the condition necessary to produce these means. What question were they asking? What did they want to know?

They wanted to find out if various interventions after watching the scary movie, would change how many bad intrusive memories people would experience in the week following the movie. They discuss the idea that a memory can become malleable when it is re-activated. They want to "Re-activate" the bad memories, and then while they were changeable, do something to them to make them less likely to be intrusive later on. The big idea was that doing a bit of "re-activation" AND then playing Tetris (which takes up visual cognitive bandwidth) could cause changes to the re-activated memories, that would decrease the number of intrusive memories later on. With that reminder, let's go through some different comparisons that we can make, and talk about what they might mean.

8.4.7.3 Control vs. Reactivation_only

There was a control group that did not do anything special after watching the traumatic movie. The mean number of intrusive memories for the control group, gives some idea of the amount of intrusive memories we would expect to measure, when you do nothing to change that number.

Comparing to the control group is a very sensible thing do to for each of the other groups. If one of the manipulations worked, it should show a different mean (something changed) than the control group.

So, did the Reaction_only group have less intrusive memories over the control group? First we use the `filter` function from `dplyr` to select only the rows from the data frame that have the Control and Reactivation_only conditions. Then we run a t-test

```
comparison_df <- all_data %>%
                  filter(Condition %in% c('Control','Reactivation_only')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = 0.219, df = 34, p-value = 0.828
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.299851  2.855406
## sample estimates:
##           mean in group Control mean in group Reactivation_only
##                        5.111111                        4.833333
```

The means are both basically 5, not a big difference!. The p-value is large, suggesting that change could easily have produced the tiny differences between the means. In other words, it doesn't look like the "re-activation" phase did anything to suppress the amount of intrusive memories that people would have over one week, compared to control.

Notice, this was a comparison we could make. But, was it an informative one about the goal of the study? Not really.

8.4.7.4 Control vs. Reactivation+Tetris

What we really want to know is if Reactivation+Tetris cause fewer intrusive memories...but compared to what? Well, if it did something, it should have a smaller mean than the Control group. So, let's do that comparison:

Note: we just change the one level name to the level we want `Reactivation+Tetris`.

```
comparison_df <- all_data %>%
                  filter(Condition %in% c('Control','Reactivation+Tetris')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = 2.9893, df = 34, p-value = 0.005167
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.031592 5.412852
```

```
## sample estimates:
##             mean in group Control mean in group Reactivation+Tetris
##                          5.111111                          1.888889
```

There is a bigger difference now, roughly 5.1 intrusive memories for control, and 1.9 for Reactivation+Tetris. The p-value is quite small, indicating this difference was not likely produced by chance. Now, we have some evidence that Reactivation+Tetris caused something to change, that condition produced fewer intrusive memories than control.

8.4.7.5 Control vs. Tetris_only

Now we can really start wondering what caused the difference. Was it just playing Tetris? It wasn't just doing the reactivation, we already found out that didn't do anything. Does just playing Tetris reduce the number of intrusive memories during the week? Let's compare that to control:

```
comparison_df <- all_data %>%
                 filter(Condition %in% c('Control','Tetris_only')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = 1.0129, df = 34, p-value = 0.3183
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.230036  3.674480
## sample estimates:
##     mean in group Control mean in group Tetris_only
##                  5.111111                 3.888889
```

There's mean difference of about 1, but the p-value isn't very small. This suggests chance produces a difference of this size fairly often. If we claimed that just playing Tetris caused a difference based on this data, we could easily be making a type I error (claiming the result is real when it is not, a false-positive). Still, the difference was in the right direction wasn't it.

8.4.7.6 Tetris_only vs. Reactivation + Tetris

Finally, we might ask if the Reactivation+Tetris group had fewer unwanted memories than the Tetris_only group. Did putting the two things together (reactivation AND Tetris) really do something special here, beyond just playing Tetris.

```
comparison_df <- all_data %>%
                 filter(Condition %in% c('Tetris_only','Reactivation+Tetris')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
```

```
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = -2.5147, df = 34, p-value = 0.01681
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.6162855 -0.3837145
## sample estimates:
## mean in group Reactivation+Tetris         mean in group Tetris_only
##                          1.888889                          3.888889
```

Well, according to the t-test, the p-value is again fairly small. Suggesting that the difference between Reactivation+Tetris (M=1.89) and Tetris_only (3.89), was not likely to be produced by chance. So, on the basis of this, there is some evidence that Reactivation+Tetris, really does cause fewer intrusive memories.

8.4.8 Writing it all up.

Because we have spent so much time on individual comparisons, we won't do a full write up of the results. A full write-up would include telling the reader what data was used, what test was conducted, the results of the test, and the pattern of the means, AND then, the results of specific comparisons of interest. You can read the paper to see how the authors did it.

8.4.9 Now it's your turn.

You get to repeat the above, but instead of analyzing the first Dependent variable (mean intrusive memories per week), you get to analyze the other one (see if you can find it from what is described in the paper)

8.4.10 Food for thought

Think about what we did here. We almost blindly just downloaded the data and ran the same analysis as the authors did. Sure, we looked at the data first, and then did the analysis. But, did we really look? Did you notice anything about what you looked at? What did we not look closely at that might make you skeptical of the conclusions from the research…

Here's a hint. Sample-size. We know that is important. Let's ask the question, was there an equal number of participants in each of the 4 conditions. We can use `dplyr` again to do this. We'll add the `length` function, which counts the number of subjects in each condition:

```
descriptive_df <- all_data %>%
                    group_by(Condition) %>%
                    summarise(means= mean(Days_One_to_Seven_Number_of_Intrusions),
                              SEs = sd(Days_One_to_Seven_Number_of_Intrusions)/sqrt(length(Days_One_to_S
                              count = length(Days_One_to_Seven_Number_of_Intrusions))

knitr::kable(descriptive_df)
```

Condition	means	SEs	count
Control	5.111111	0.9963623	18
Reactivation+Tetris	1.888889	0.4113495	18
Tetris_only	3.888889	0.6806806	18
Reactivation_only	4.833333	0.7848650	18

The answer is YES, there were an equal number of subjects. That's good. We should have checked that before. Lesson for next time. For example, if there were only 9 subjects in the Reactivation+Tetris group, we might be suspicious that they got lucky, and accidentally (by chance) assigned people to that group who

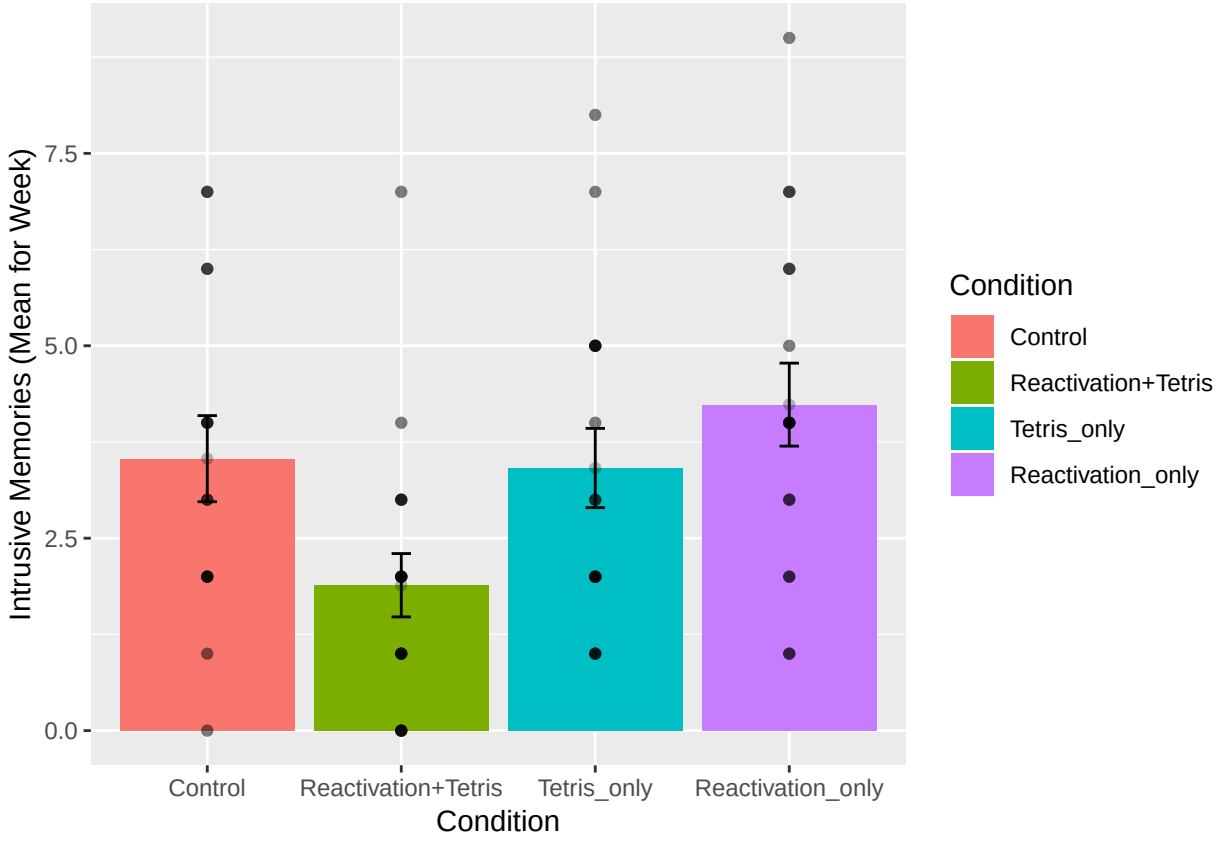
are unlikely to report having intrusive memories. After all, different people are different, and not everybody is as susceptible to intrusive memories.

Let's do one more thing for fun, and to see everything in action all in one place. Let's consider the role of outliers. Looking at the first graph we can see that most people in all the groups had fewer than 10 intrusive memories (mean we assume) per week. It looks like 5 people had more than that, and they just happened to be in the other groups. Maybe Reactivation+Tetris made those people have way less intrusive memories (which is why no one is above 10), or maybe the researchers got a little bit unlucky, and accidentally didn't get any "outliers" (people with extreme values on the measure) in that group.

Let's re-run the analysis, but remove anybody with a mean higher than 10. This will only remove 5 subjects, so we will still have a lot left. What happens?

Before we find out, let me point out again the beauty of R. All we need to do is copy and paste our previous code. Then, just filter the data once to remove the outliers, then voila, we redo everything all in one go. It's much more complicated and time consuming to do this in many other software programs. You are lucky to be learning R.

```
# get rid out of outliers

all_data  <- all_data %>%
             filter(Days_One_to_Seven_Number_of_Intrusions < 10)

# get means and SEs

descriptive_df <- all_data %>%
                    group_by(Condition) %>%
                    summarise(means= mean(Days_One_to_Seven_Number_of_Intrusions),
                              SEs = sd(Days_One_to_Seven_Number_of_Intrusions)/sqrt(length(Days_One_to_S

# Make the plot

ggplot(descriptive_df, aes(x=Condition, y=means))+
  geom_bar(stat="identity", aes(fill=Condition))+ # add means
  geom_errorbar(aes(ymin=means-SEs,               # add error bars
                    ymax=means+SEs), width=.1) +
  geom_point(data=all_data, aes(x=Condition, y=Days_One_to_Seven_Number_of_Intrusions), alpha=.5)+
  geom_point(alpha=.25)+
  ylab("Intrusive Memories (Mean for Week)")
```




```
# run and report the ANOVA

aov_out<-aov(Days_One_to_Seven_Number_of_Intrusions ~ Condition, all_data)
summary_out<-summary(aov_out)

knitr::kable(xtable(summary_out))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Condition	3	51.49152	17.163841	4.024458	0.0110347
Residuals	63	268.68758	4.264882	NA	NA

```
# conduct critical comparisons

## control vs reactivation+Tetris

comparison_df <- all_data %>%
                 filter(Condition %in% c('Control','Reactivation+Tetris')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = 2.4159, df = 31, p-value = 0.02178
## alternative hypothesis: true difference in means is not equal to 0
```

```
## 95 percent confidence interval:
##  0.2562181 3.0326708
## sample estimates:
##             mean in group Control mean in group Reactivation+Tetris
##                          3.533333                           1.888889
```

```
## Tetris_only vs reactivation+Tetris

comparison_df <- all_data %>%
                 filter(Condition %in% c('Tetris_only','Reactivation+Tetris')==TRUE)

t.test(Days_One_to_Seven_Number_of_Intrusions ~ Condition,
       comparison_df,
       var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  Days_One_to_Seven_Number_of_Intrusions by Condition
## t = -2.3239, df = 33, p-value = 0.02643
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.8561054 -0.1896463
## sample estimates:
## mean in group Reactivation+Tetris         mean in group Tetris_only
##                          1.888889                           3.411765
```

The take home is that yes, even after removing outliers, the same basic pattern in the data is observed. Overall, this is a small n study, and ideally the basic findings should be replicated in another lab before we really have full confidence in them. But, I'd say the trends here look promising.

That's ANOVA. Come back next week for another ANOVA tutorial, this time using within-subject data. It's called a repeated measures ANOVA, and it's what's happening next week.

8.5 Excel

How to do it in Excel

8.6 SPSS

How to do it in SPSS

8.7 JAMOVI

How to do it in JAMOVI

Chapter 9

Lab 9: One-way ANOVA

9.1 Lab Skills Learned

9.2 Important Stuff

- citation: Rosenbaum, D., Mama, Y., & Algom, D. (2017). Stand by Your Stroop: Standing Up Enhances Selective Attention and Cognitive Control. Psychological science, 28(12), 1864-1867.
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format

9.3 Outline of Problem to solve

Stuff we need to say in general

9.3.1 important things

Other things to say

9.4 R

How to do it in R

9.5 Excel

How to do it in Excel

9.6 SPSS

How to do it in SPSS

9.7 JAMOVI

How to do it in JAMOVI

Chapter 10

Lab 10: Factorial ANOVA

10.1 Lab Skills Learned

10.2 Important Stuff

- citation: Barasch, A., Diehl, K., Silverman, J., & Zauberman, G. (2017). Photographic memory: The effects of volitional photo taking on memory for visual and auditory aspects of an experience. Psychological science, 28(8), 1056-1066.
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format

10.3 Outline of Problem to solve

Stuff we need to say in general

10.3.1 important things

Other things to say

10.4 R

How to do it in R

10.5 Excel

How to do it in Excel

10.6 SPSS

How to do it in SPSS

10.7 JAMOVI

How to do it in JAMOVI

Chapter 11

Lab 11: Mixed Factorial ANOVA

11.1 Lab Skills Learned

11.2 Important Stuff

- citation:
- Link to .pdf of article
- Data in .csv format
- Data in SPSS format ## Outline of Problem to solve

Stuff we need to say in general

11.2.1 important things

Other things to say

11.3 R

How to do it in R

11.4 Excel

How to do it in Excel

11.5 SPSS

How to do it in SPSS

11.6 JAMOVI

How to do it in JAMOVI