Answering questions with data: Lab Manual

Mallory Barnes

2018: Last Compiled 2023-08-23

Contents

Preface

First Draft (version 0.9 = August 15th, 2018) Last Compiled: 2023-08-23

This is the companion lab to our free introductory statistics for undergraduates in psychology textbook, Answering questions with data.

This lab manual involves step by-step tutorials to solve data-analysis problems in software. We use open-data sets that are usually paired with a primary research article.

Each lab is designed to be solved in R & R-studio.

The manual is a free and open resource. See below for more information about copying, making change, or contributing to the lab manual.

0.1 Important notes

This lab manual is based on the OER course package for teaching undergraduate statistics in Psychology, customized for Environmental Science Graduate students by Mallory Barnes. The source code for all material is contained in the GitHub repositories for each, and they are a written in R-markdown, so they are relatively easy to copy and edit. Have Fun!

0.1.1 Attributions

This lab manual was authored by Matt Crump (R exercises), updated by Mallory Barnes.

Labs 6, 7, and 8 were adapted and expanded from Open Stats Labs. Thanks to Open Stats Labs (Dr. Kevin P. McIntyre) for their fantastic work.

0.1.2 CC BY-SA 4.0 license

This license means that you are free to:

- Share: copy and redistribute the material in any medium or format
- Adapt: remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- Attribution: You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- ShareAlike: If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
- No additional restrictions: You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

0.1.3 Copying the lab manual

This lab manual was written in R-Studio, using R Markdown, and compiled into a web-book format using the bookdown package.

All of the source code for compiling the book is available in the GitHub repository for this book:

https://github.com/CrumpLab/statisticsLab

In principle, anybody could fork or otherwise download this repository. Load the Rproj file in R-studio, and then compile the entire book. Then, the individual .rmd files for each chapter could be edited for content and style to better suit your needs.

If you want to contribute to this version of the lab manual, you could make pull requests on GitHub, or discuss issues and request on the issues tab.

0.1.4 Acknowledgments

Thanks to the librarians at Brooklyn College of CUNY, especially Miriam Deutch, and Emily Fairey, for their support throughout the process. Thanks to CUNY for supporting OER development, and for the grant we received to develop this work.

Software

0.2 Data

Data files used for the labs are all taken from open data sources. Links are provided for each lab. For convenience, all of the data files are also available here as single files in the github repository for this lab manual

0.2.1 Data Repository

https://github.com/CrumpLab/statisticsLab/tree/master/data

0.2.2 CSV format

All of the data files in .csv format are also available to download as a .zip file

0.3 R

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.

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

0.3.1 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, particular analyses often require that the data be formatted in a particular way so that the program can 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 reformatting 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.

0.3.2 R studio notes and tips

0.3.2.1 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

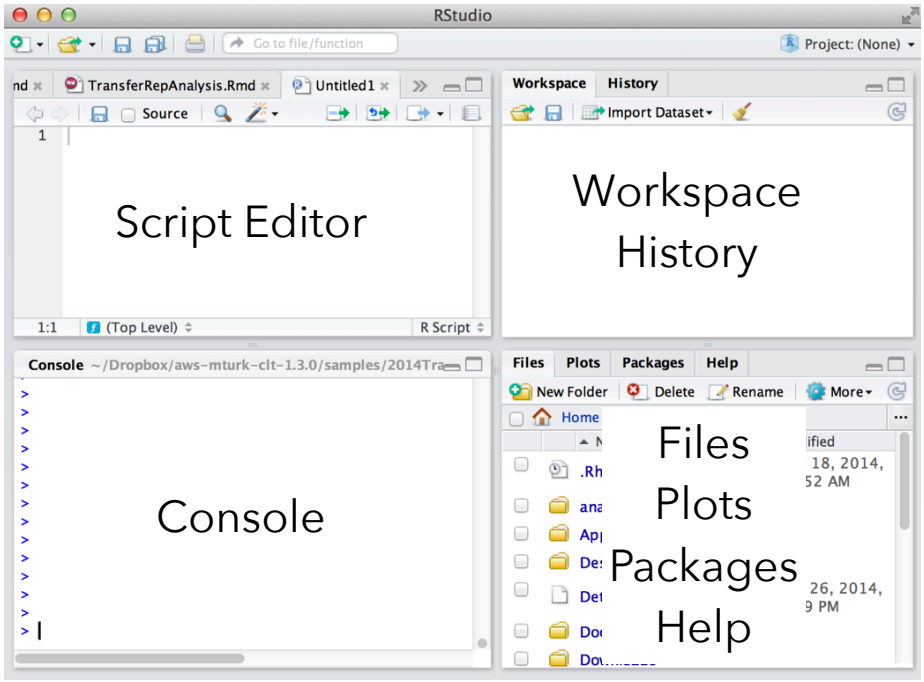


Figure 1: The R-studio workspace

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.

0.3.2.2 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).

0.3.2.3 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.

0.3.2.4 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.

0.3.3 How to complete the R Labs

Each of the labs focuses on particular data-analysis problems, from graphing data, computing descriptive statistics, to running inferential tests in R. All of the labs come in three parts, a training part, a generalization part, and a writing part. The training part includes step-by-step examples of R code that solves particular problems. The R code is always highlighted in grey. The generalization part gives short assignments to change parts of the provided code to solve a new problem. The writing part tasks you with answering questions about statitiscal concepts.

The way to complete each lab is to open a new R Markdown document in R-studio, and then document your progression through each of the parts. By doing this, you will become familiar with how R and R-studio works, and how to create documents that preserve both the code and your notes all in one place. There are a few tricks to getting started that are outline below.

1. Open R-studio

0.3.3.1 R projects

2. Create a new R project
    a. Go to the file menu and select new project, or go to the top right-hand corner of R-studio, you should see a blue cube with an R in it, then select New project from the dropdown menu
3. Save the new R project somewhere that you can find it. If you are working on a lab computer, then save the new R project to the desktop.

What is an R project? When you create a new R project you are creating two things, 1) a new folder on your computer, and 2) a ".Rproj" file. For example, if you gave your R project the name "Lab1", then you will have created a folder title "Lab1", and inside the folder you will find an R project file called "Lab1.Rproj".

As you work inside R-studio you will be creating text documents, and you will be doing things like loading data, and saving the results of your analyses. As your work grows and becomes more complex, you can often find yourself creating many different files. **The R project folder is a very useful way of organizing your files all in one place so you can find them later.** If you double-clik an R project file, R-studio will automatically load and restore your last session. In the labs, you will be using your R project folder to:

1. save data files into this folder
2. save R-markdown files that you will use to write your R-code and lab notes
3. save the results of your analysis

0.3.3.2 Installing libraries

When you install R and R-studio, you get what is called Base R. Base R contains many libraries that allow you to conduct statistical anlayses. Because R is free and open-source, many other developers have created add-on libraries that extend the functionality of R. **We use some of these libraries, and you need to install them before you can do the labs**.

For example, in any of the labs, whenever you see a line code that uses the word library like this `library(libraryname)`, this line of code telling R to load up that library so it can be used. The `libraryname` would be replaced with the actual name of the library. For example, you will see code like this in the labs:

```
library(data.table)
```

This line of code is saying that the `data.table` library needs to be loaded. You can check to see if any library is already loaded by clicking on the "packages" tab in the bottom right hand panel. You will see many packages listed in alphabetical order. Packages that are currently loaded and available have a checkmark. If you scroll down and find that you **do not** have `data.table` installed, then you need to install it. To install any package follow these steps:

1. Click on the packages tab
2. Find the "install" button in the top left hand corner of the packages tab.
3. Click the install button
4. Make sure "install from:" is set to CRAN repository
5. Make sure "dependencies" is clicked on (with a checkmark)
6. type the name of the library into the search bar.
7. As you type, you should see the names of different packages you can install pop-up in a drop-down menu. **You must be connected to the internet to install packages from CRAN**
8. Once you find the package (e.g., `data.table`), click it, or just make sure the full, correctly spelled name, is in the search bar
9. Press the install button

You should see some text appear in the console while R installs the package.

10. After you have installed the package, you should now see that it is listed in the packages tab.
11. You can turn the package on by clicking it in the package tab.
12. OR, you can turn the packge on by running the command `library(data.table)` in the console, to do this type `library(data.table)` into the console, and press enter.

0.3.3.3 Quick install

If you are using R on one of the lab computers, you may find that some of the packages are not installed. The lab computers get wiped everynight, so it may be necessary to install packages each time you come back to the lab. Fortunately, we can tell R to install all of the packages we need in one go. Copy the following lines of code into the console, and press enter. Note you can select all of the lines at once, then copy them, then paste all of them into the console, and press enter to run them all. After each of the packages are installed, you will then be able to load them using `library()`.

```
install.packages(ggplot2)
install.packages(dplyr)
install.packages(data.table)
install.packages(summarytools)
install.packages(gapminder)
install.packages(ggpubr)
```

0.3.3.4 R markdown

Once you have the necessary packages installed you can begin creating R markdown documents for each lab. We admit that at the beginning, R markdown documents might seem a little bit confusing, but you will find they are extremely useful and flexible. Basically, what R markdown allows you to do is combine two kinds of writing, 1) writing R code to conduct analyses, and 2) writing normal text, with headers, sub-headers, and paragraphs. You can think of this like a lab journal, that contains both your writing about what you are doing (e.g., notes to self), and the code that you use for analysis. Additionally, when your code does something like make a graph, or run a statistical test, you can ask R markdown to print the results.

The R markdown website has an excellent tutorial that is well worth your time to check out: https://rmarkdown.rstudio.com/lesson-1.html

0.3.3.5 R markdown lab templates

1. Go to Canvas and download RMarkdownLab1.zip file to your computer.
2. Unzip the file, this will produce a new folder with three important parts
    a. data folder (contains data files for Lab 1)
    b. Blank template (.Rmd file) for completing Lab 1: Lab 01 Graphing_Student Name.Rmd
    c. RMarkdownsLab1.Rproj A file with a little blue cube with an R in it.

3. Double-click the RMarkdownsLab1.Rproj file, this will automatically open R-studio
4. Open the template .rmd, and use it to begin adding your code and notes for lab 1.

One you've downloaded R-studio on your computer, you'll Just need to find it and double-click. Now you have R-studio. Your lab instructor will also walk you through the steps to get started completing the first lab.

To get started with Lab 1, follow these steps:

1. copy the template file for lab 1, "Lab 01 Graphing_Student Name.Rmd", and place it into the "RMarkdownsLab" (copy it out of the template folder, and into the RMarkdownsLab folder).
2. Rename the file to add your own name, eg., "Lab1GraphingMattCrump.Rmd"
3. double-click the "RMarkdownsLab.Rproj" file
4. R-studio will now load up.
5. If you click the files tab, you will see all of the files and folders inside the "RMarkdownsLab" folder
6. Click on your .rmd file, it will now load into the editor window.

Each lab template .rmd file contains three main sections, one for each part of the lab. You will write things inside each section to complete the lab.

0.3.4 R-studio Cloud

R-studio is also in the cloud. This means that if you want to use R and R-studio through your web-browser you can do that without even installing R or R-studio on your computer. It's also free!

1. sign up for an R-studio cloud account here: https://rstudio.cloud
2. You can make new R projects, work inside them, and everything is saved in the cloud!
3. To see how everything would work, follow the steps in this video. You will need to download this .zip file to your computer to get started

The link to the video is https://www.youtube.com/watch?v=WsbnV0t7FE4, or you can watch it here:

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. Download the statistical software program R and the RStudio (Integrated Development Environment for R) on your personal computers
2. Load in some data to a statistical software program
3. Talk a little bit about how the data is structured
4. Make graphs of the data so we can look at it and make sense of it.

1.1.1 Important info

1. Data for NYC film permits was obtained from the NYC open data website. The .csv file can be found here: Film_Permits.csv
2. Gapminder data from the gapminder project (copied from the R gapminder library) can be downloaded in .csv format here: gapminder.csv

1.2 R

1.2.1 Install R and R Studio

First, we need to install 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.

1.2.2 Download the lab template

You will be completing each lab by writing your code and notes in an R Markdown document.

1. Go to Canvas and download RMarkdownLab1.zip file to your computer.
2. Unzip the file, this will produce a new folder with three important parts
    a. data folder (contains data files for Lab 1)
    b. Blank template (.Rmd file) for completing Lab 1: Lab 01 Graphing_Student Name.Rmd
    c. RMarkdownsLab1.Rproj A file with a little blue cube with an R in it.
3. Double-click the RMarkdownsLab1.Rproj file, this will automatically open R-studio

4. Open the template .rmd, and use it to begin adding your code and notes for lab 1.

One you've downloaded R-studio on your computer, you'll Just need to find it and double-click. Now you have R-studio. Your lab instructor will also walk you through the steps to get started completing the first lab. The steps are here.

There are numerous resources for learning about R, and I've put some of them on the course Canvas, under the Resources module. You will find these resources helpful as you learn.The Crump lab also has a helpful general introduction to R and Rstudio here. This shows you how to download R and R-studio at home (it's free). Throughout the labs you will be writing things called R Markdown documents. You will learn how to do this throughout the labs, but it can also be worthwhile reading other tutorials, such as the one provided by R Markdown.

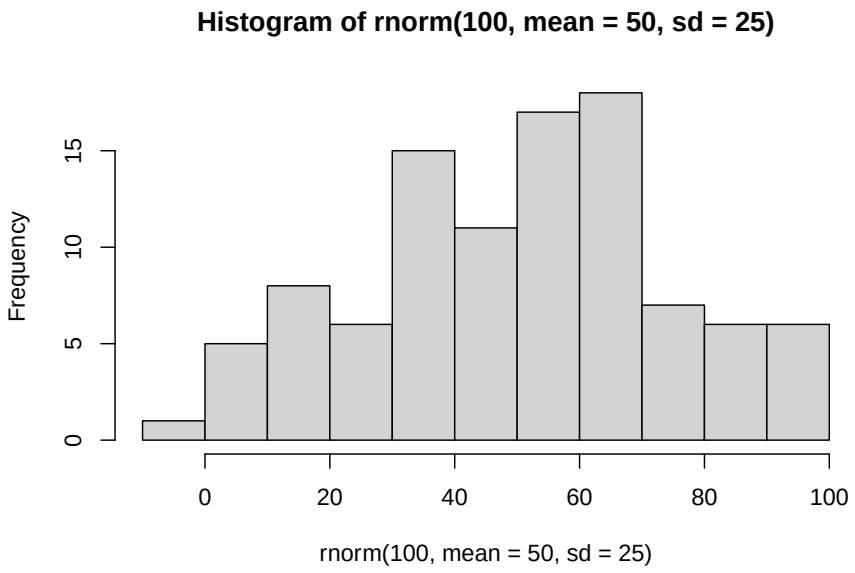
When I designed this course, I considered that many students might be unfamiliar with R and RStudio. The idea of diving into a computer programming language might seem daunting or even intimidating. But I assure you, there's no need to panic or consider dropping the course! In reality, learning R is going to be much more accessible and enjoyable than you might think. Together, we'll demystify the process and make it an engaging experience. Let's compare to other statistics courses where you would learn something like SPSS or SAS. 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.

I know that this will seem challenging at first. But, I 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 which case, this kind of thing is necessary skill to learn). With that, let's begin.

1.2.3 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 a data file that lists the locations of film permits for shooting movies all throughout the Burroughs. There are multiple ways to load this data into R.

1. If you have downloaded the RMarkdownLab1.zip file, then you already have the data file in the data folder. Assuming you are working in your main directory (your .rmd file is saved in the main folder that contains both the data and template folders), then use the following commands to load the data.

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

2. If the above method doesn't work, you can try loading the data from my github:

```
library(data.table)
nyc_films <- fread("https://raw.githubusercontent.com/malloryb/statisticsLabE538/master/data/Film
```

If you are having issues getting the data loaded, then talk to your lab instructor

1.2.4 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 data frame is really big, it has 50,728 rows of data. That's a lot too much to look at.

1.2.4.1 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.

1.2.5 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.

1.2.5.1 Where are the most film permits being requested?

Do you have any guesses? Is it Manhattan, or Brooklyn, 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 borough, and then make different bars represent the the counts.

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

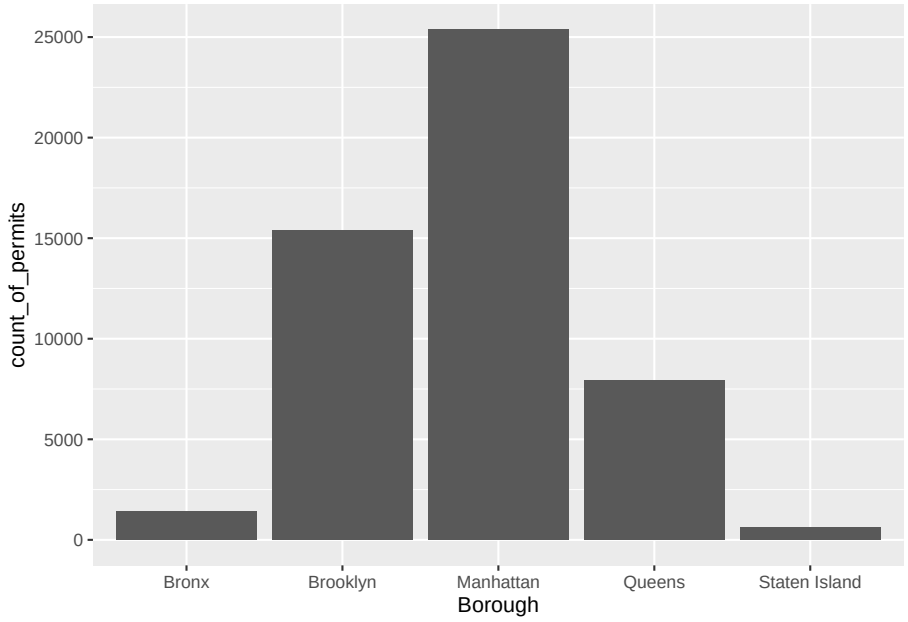
```
library(dplyr)

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 occurred (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")
```

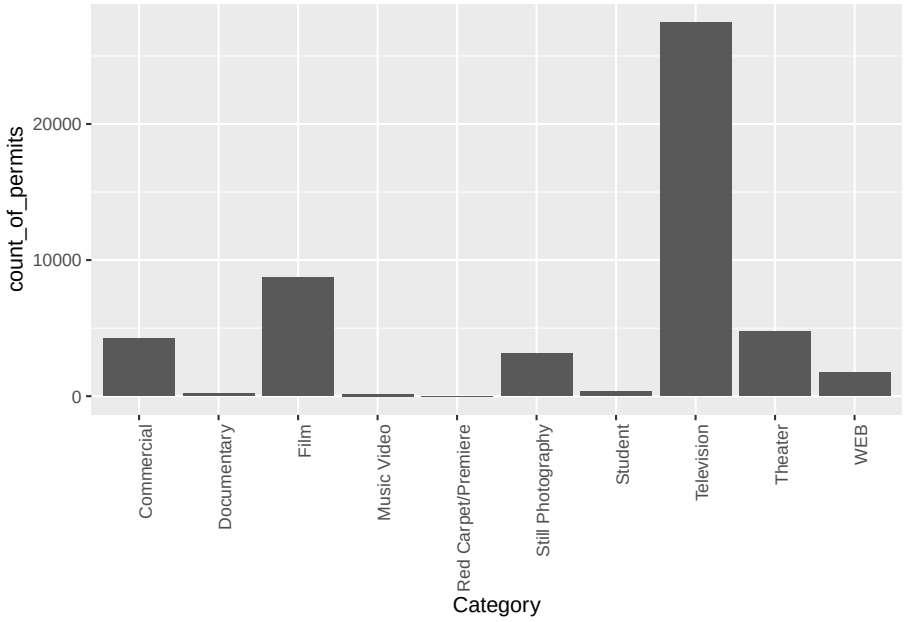


There it is, we're done here! We can easily look at this graph, and answer our question. Most of the film permits were requested in Manhattan, followed by Brooklyn, then Queen's, the Bronx, and finally Staten Island.

1.2.5.2 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.

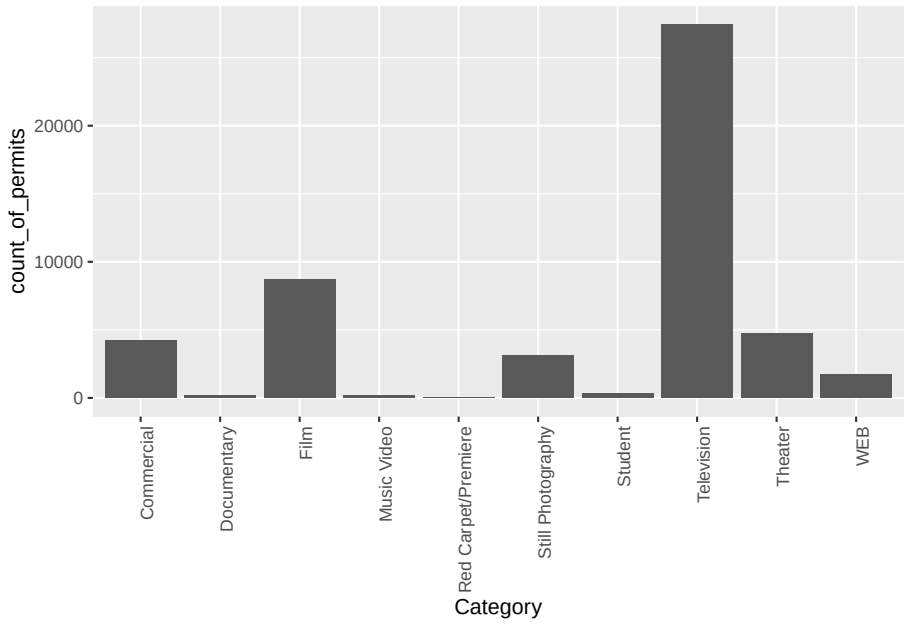
1. I changed `Borough` to `Category`. That was the main thing
2. I left out a bunch of things from before. None of the `library()` commands are used again, and I didn't re-run the very early code to get the data. R already has those things in it's memory, so we don't need to do that first. If you ever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



1.2.6 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.

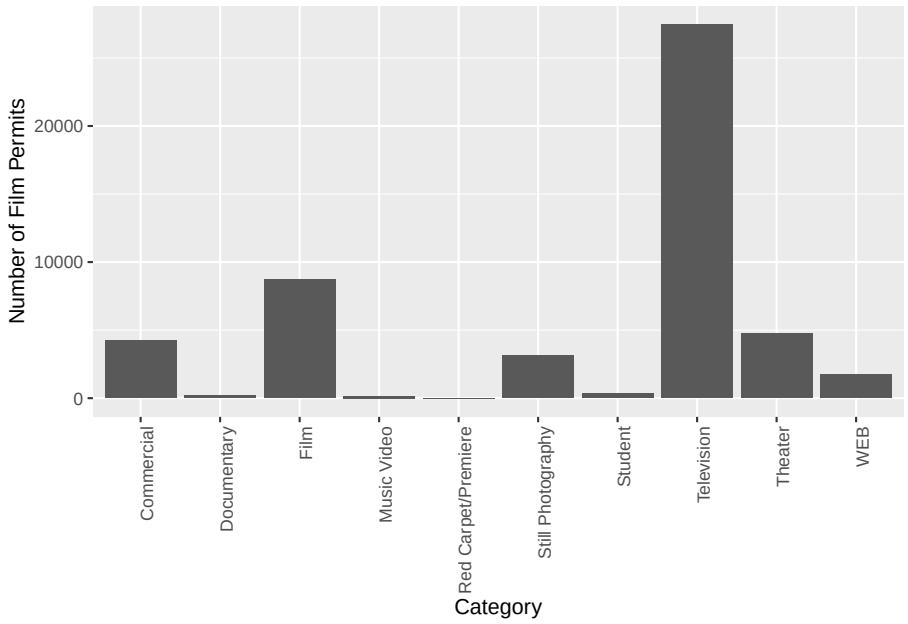
BUT WAIT? How am I supposed to know what to add? This is nuts! We know. You're not supposed to know just yet, how could you? We'll give you lots of examples where you can copy and paste, and they will work. That's how you'll learn. If you really want to read the help manual you can do that too. It's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

1.2.6.1 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()`

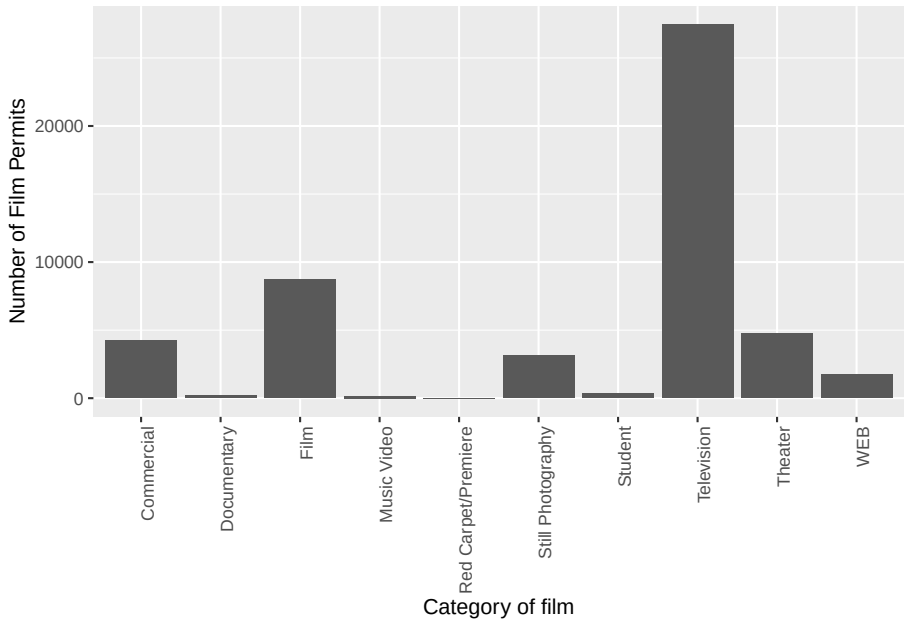
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits")
```



1.2.6.2 xlab() changes x label

Let's slightly modify the x label too:

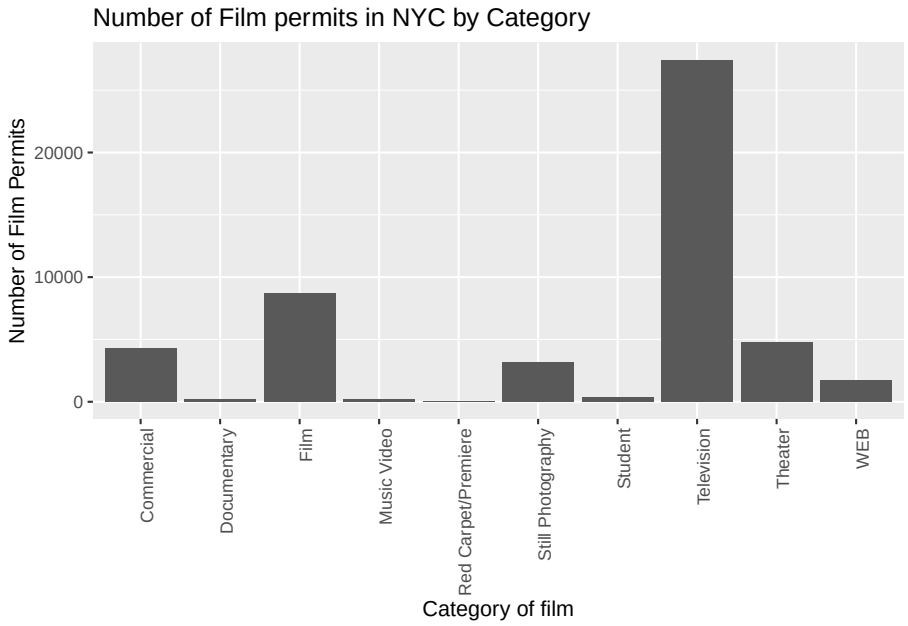
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film")
```



1.2.6.3 ggtitle() adds title

Let's give our graph a title

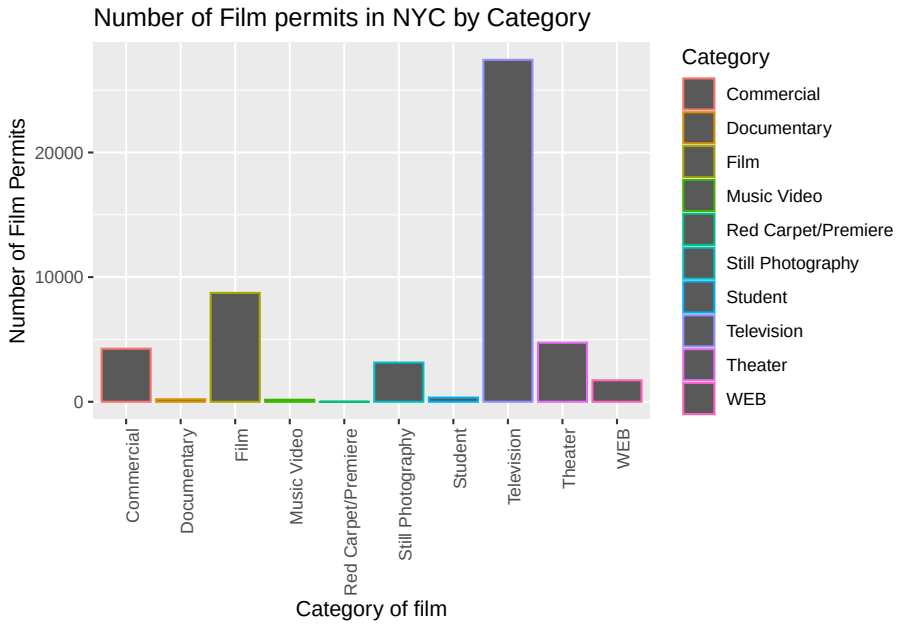
```
ggplot(counts, aes(x = Category, y = count_of_permits )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.6.4 color adds color

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

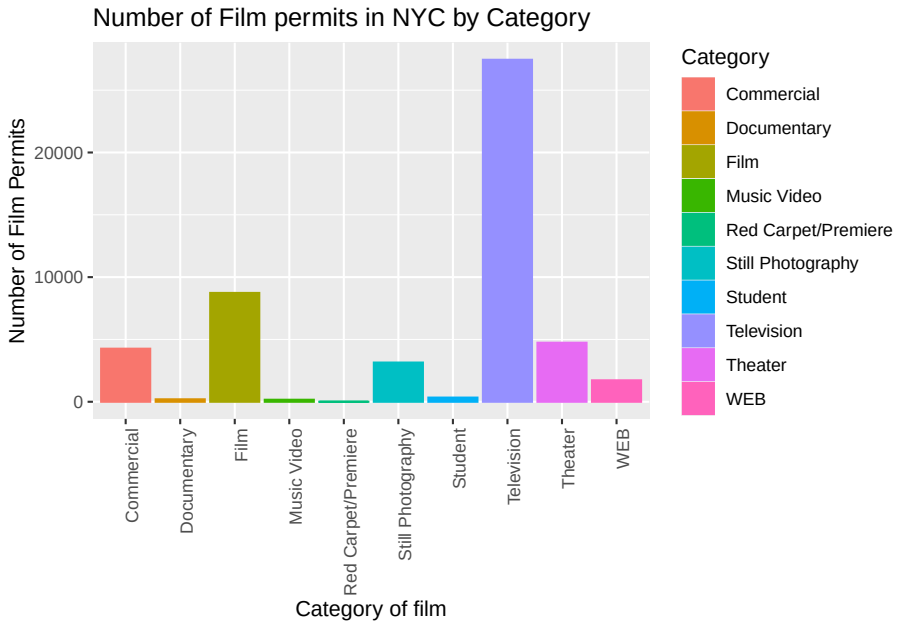
```
ggplot(counts, aes(x = Category, y = count_of_permits, color=Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number of Film permits in NYC by Category")
```



1.2.6.5 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")
```

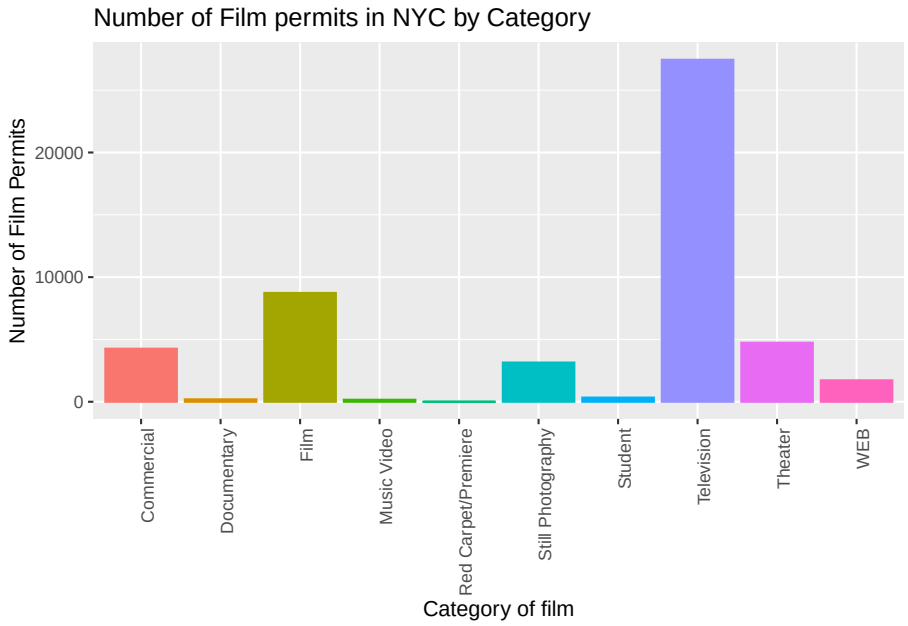


1.2.6.6 get rid of the legend

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

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

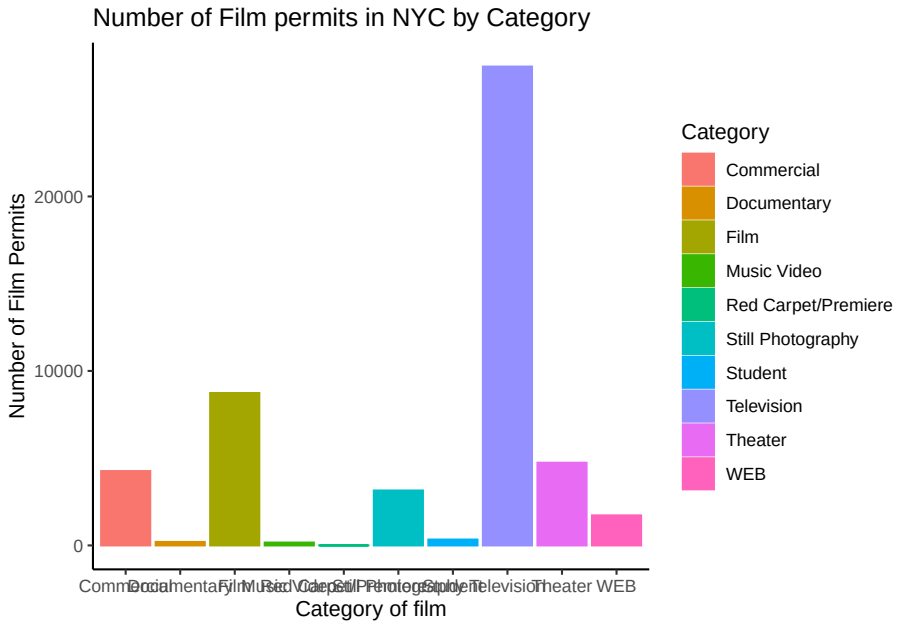
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number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



1.2.6.7 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.

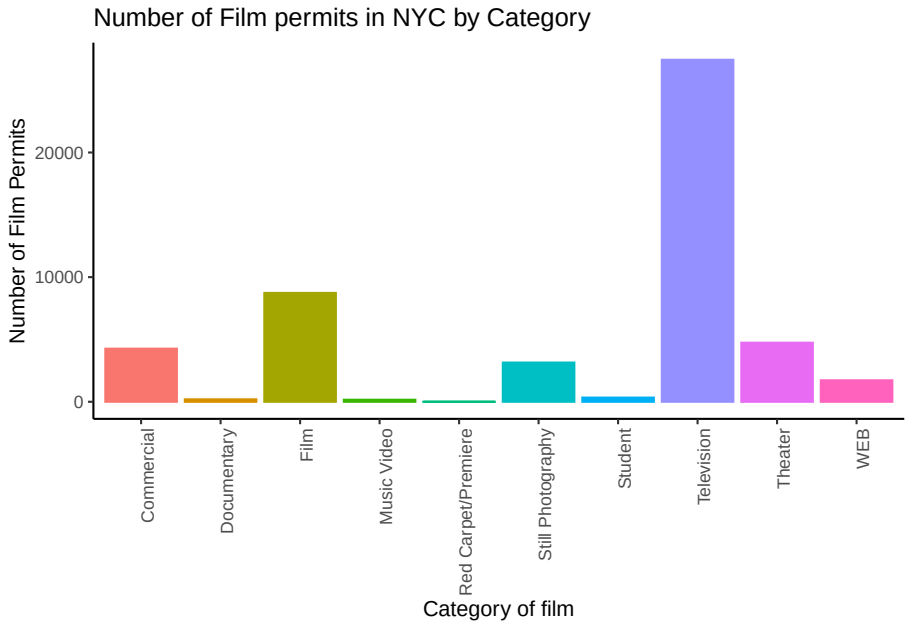
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film") +
  ggtitle("Number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



1.2.6.8 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.

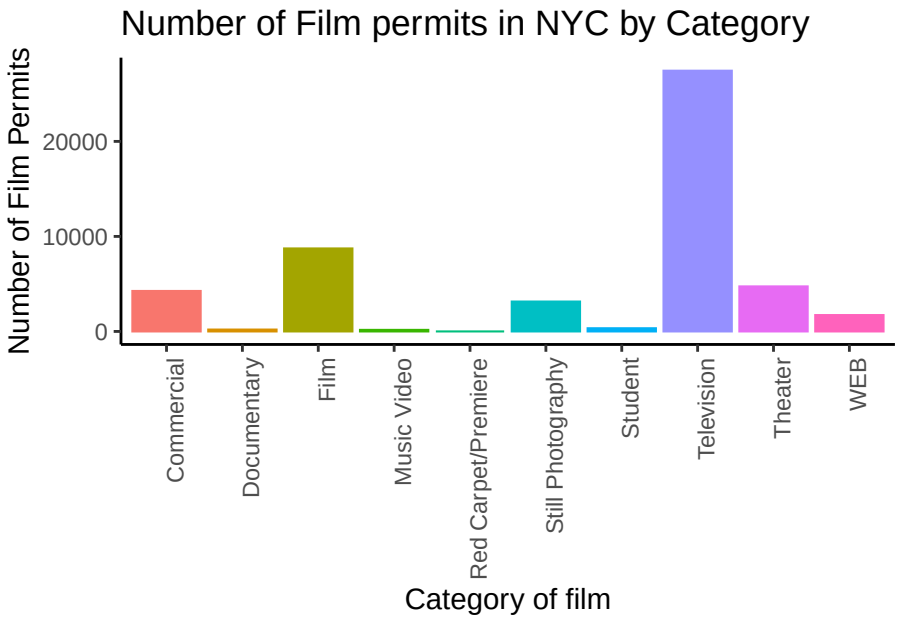
```
ggplot(counts, aes(x = Category, y = count_of_permits,
                   color=Category,
                   fill= Category )) +
  geom_bar(stat="identity") +
  theme_classic() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Number of Film Permits") +
  xlab("Category of film")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



1.2.6.9 Font-size

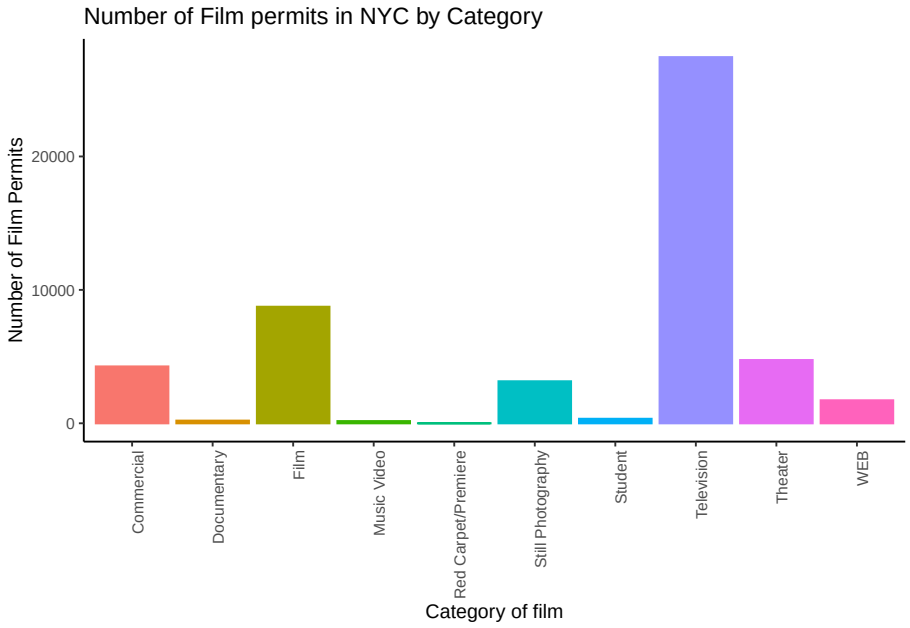
Changing font-size is often something you want to do. ggplot2 can do this in different ways. I suggest using the `base_size` option inside `theme_classic()`. You set one number for the largest font size in the graph, and everything else gets scaled to fit with that that first number. It's really convenient.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



1.2.6.10 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.

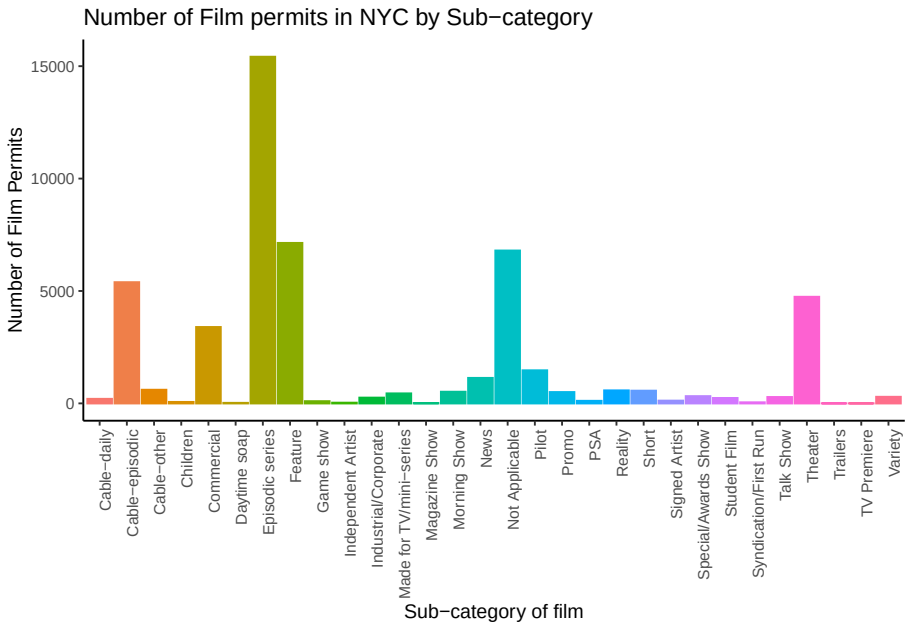
1.2.7 More questions about NYC films

1.2.7.1 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.

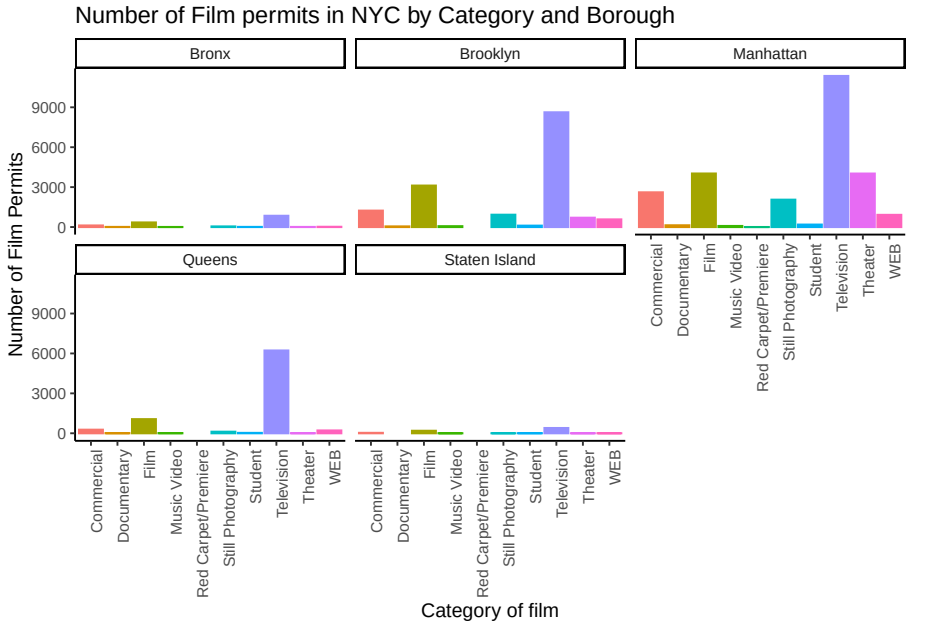
1.2.7.2 Categories by different Boroughs

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 same 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 Brooklyn? 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)
```

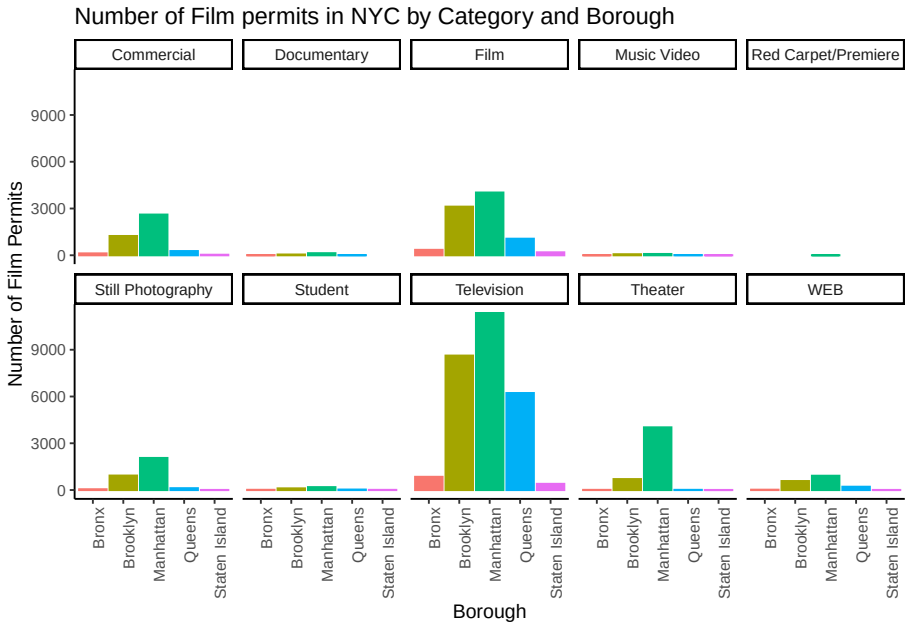


We did two important things. First we added `Borough` and `Category` into the `group_by()` function. This automatically gives separate counts for each

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)
```



1.2.8 Gapminder Data

https://www.gapminder.org is an organization that collects some really interesting worldwide data. They also make cool visualization tools for look-

ing 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 package 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
```

1.2.8.1 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.

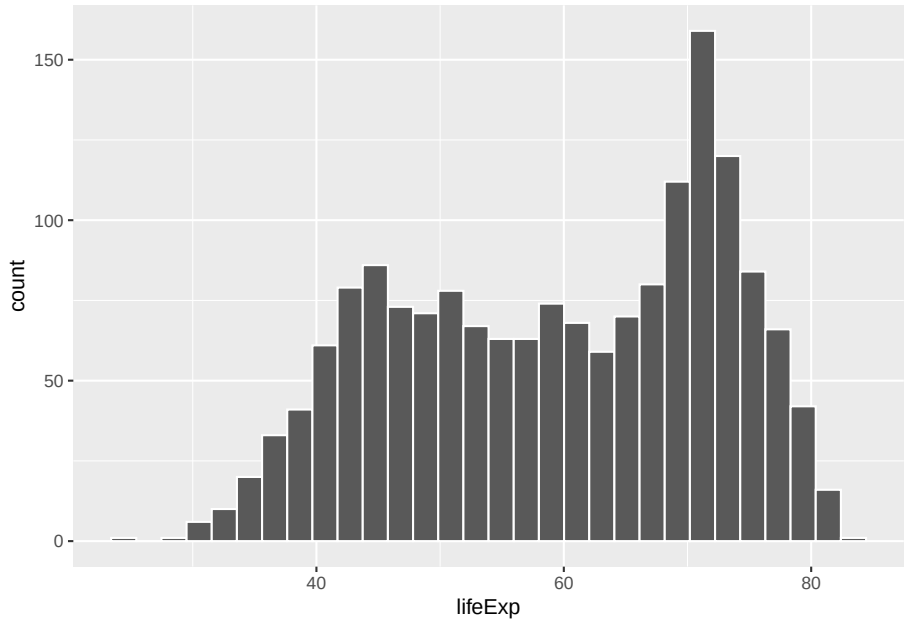
1.2.9 Asking Questions with the gap minder data

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

1.2.9.1 Life Expectancy histogram

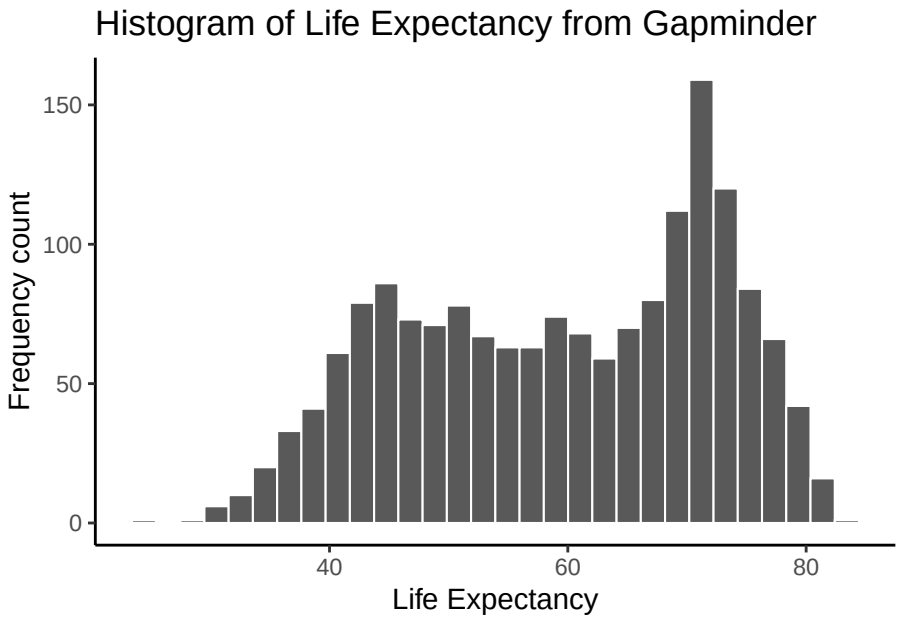
How long are people living all around the world according to this data set? There are many ways we could plot the data to find out. The first way is a histogram. We have many numbers for life expectancy in the column `lifeExp`. This is a big sample, full of numbers for 142 countries across many years.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

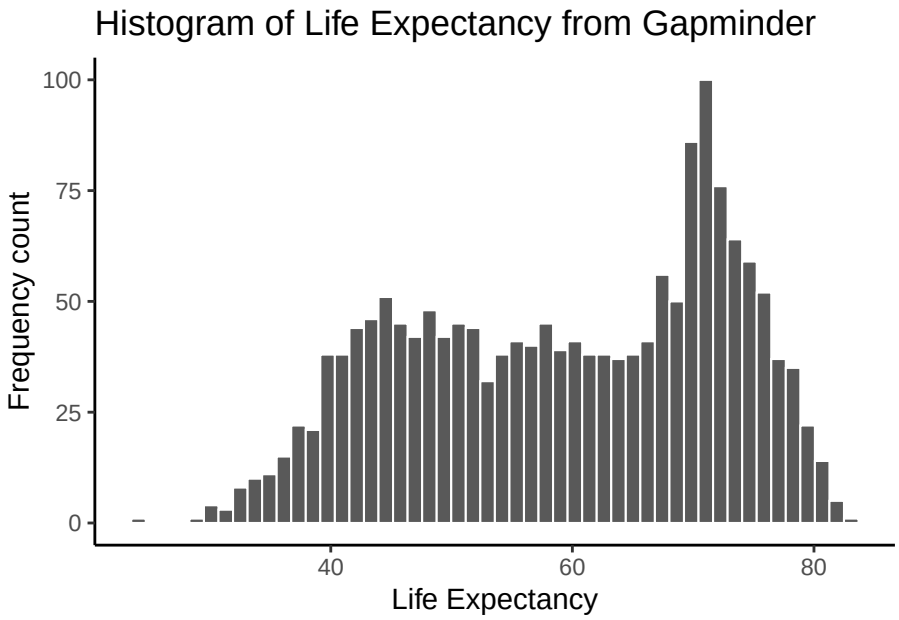
```
ggplot(gapminder_df, aes(x = lifeExp)) +
  geom_histogram(color="white")+
  theme_classic(base_size = 15) +
  ylab("Frequency count") +
  xlab("Life Expectancy") +
  ggtitle("Histogram of Life Expectancy from Gapminder")
```



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")
```



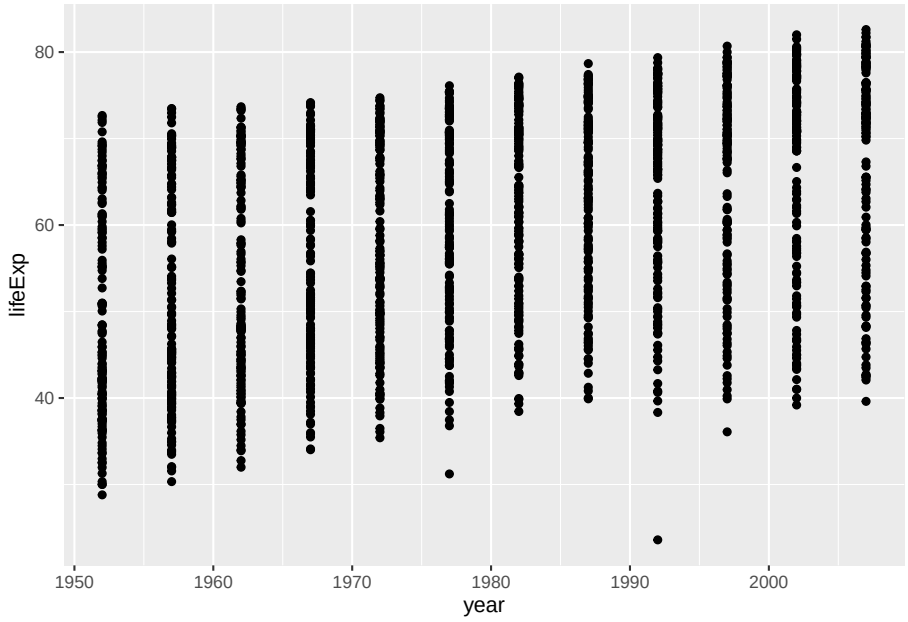
See, same basic patter, but now breaking up the range into 50 little equal sized bins, rather than 30, which is the default. You get to choose what you want to do.

1.2.9.2 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 scatter plot. 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()
```

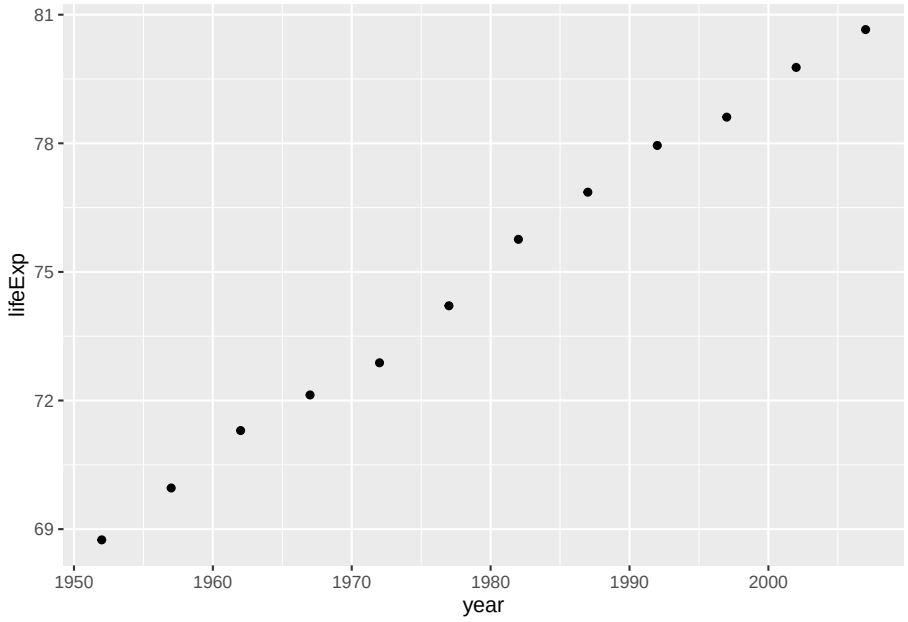


Whoa, that's a lot of dots! Remember that each country is measured each year. So, the bands of dots you see, show the life expectancies for the whole range of countries within each year of the database. There is a big spread inside each year. But, on the whole it looks like groups of dots slowly go up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

1.2.9.3 One country, life expectancy by year

Let's say I'm 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()
```

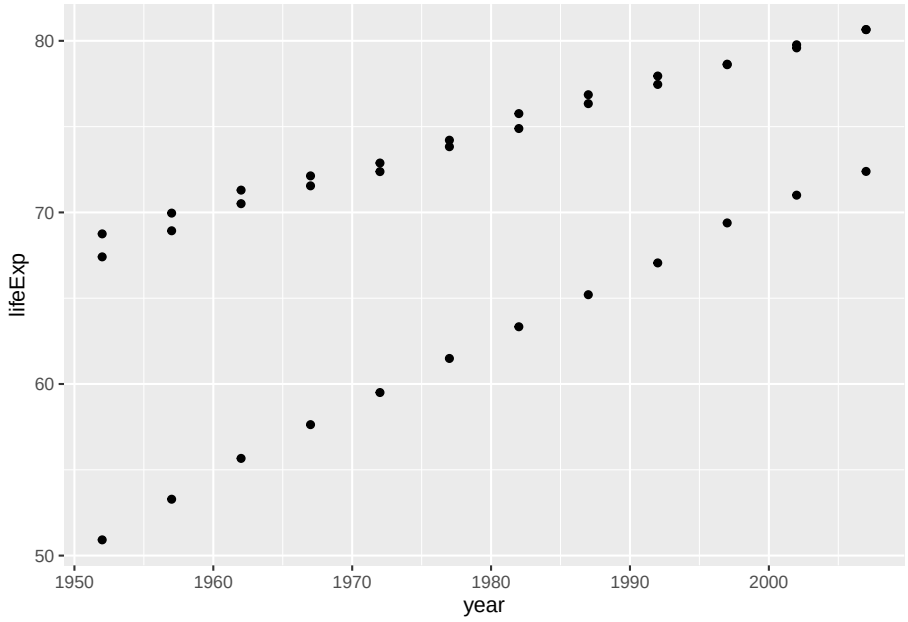


I would say things are looking good for Canadians, their life expectancy is going up over the years!

1.2.9.4 Multiple countries scatterplot

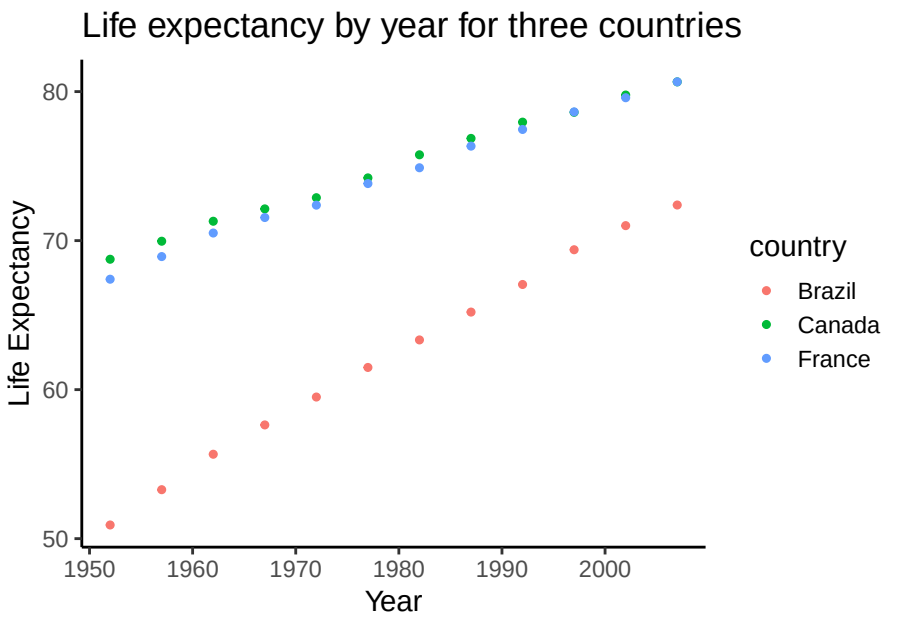
What if we want to look at a few countries altogether. We can do this too. We just change how we filter the data so more than one country is allowed, then we plot the data. We will also add some nicer color options and make the plot look pretty. First, the simple code:

```
# 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 legend, and make the graph better looking.

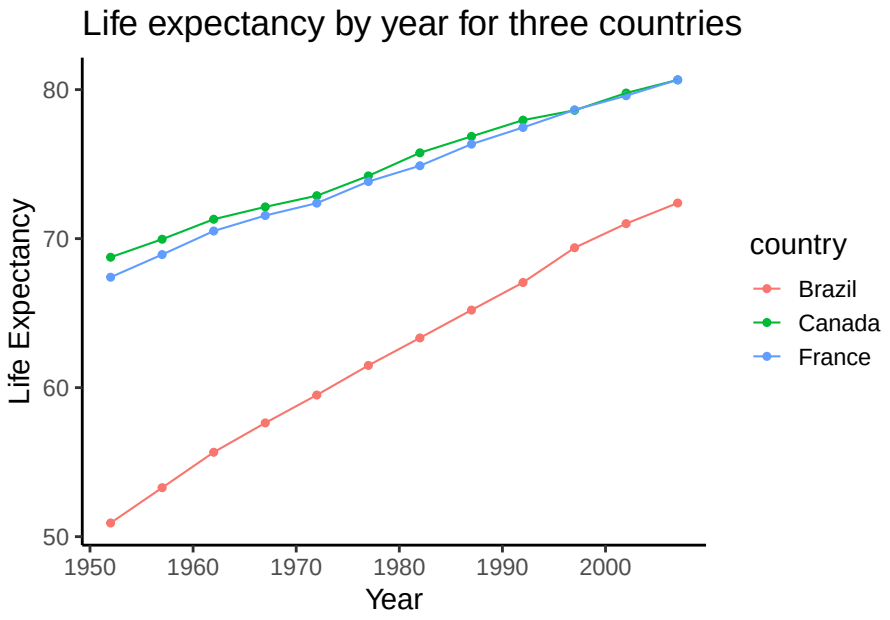
```
ggplot(smaller_df,aes(y= lifeExp, x= year,
                      group= country, color = country)) +
  geom_point()+
  theme_classic(base_size = 15) +
  ylab("Life Expectancy") +
  xlab("Year") +
  ggtitle("Life expectancy by year for three countries")
```



1.2.9.5 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")
```



1.2.10 Generalization Exercise

The following generalization exercise and writing assignment is also in your lab R Markdown document for this lab. Complete your work in that document and hand it in.

(4 points - Pass/Fail)

Use the code from above to attempt to solve the extra things we ask you do for this assignment. You generalization exercises are as follows:

1. Make a graph plotting Life Expectancy by year for the five continents, using the `continent` factor. Make sure you change the title so it reads correctly

2. Make a graph plotting GDP per capita by year for the USA, Canada, and Mexico. Use the `gdpPercap` column for the GDP per capita data

3. Make a new graph plotting anything you are interested in using the gapminder dataset. It just needs to be a plot that we have not given an example for

1.2.11 Writing assignment

Complete the writing assignment described here in your R Markdown document for this lab. When you have finished everything, knit the document and submit to Canvas.

Note: It's vital that the work submitted is entirely your own. Utilizing chatGPT or other AI to complete this portion of the lab assignment is not permissible. These responses must reflect your own words and thoughts. Using external assistance not only constitutes plagiarism but also undermines the educational value of this exercise, hindering your opportunity to learn and grasp the concepts intended.

The question for this lab is a long answer question about histograms. Here is the question:

Describe what histograms are, how to interpret them, and what they are useful for. You should answer each of these questions:

The answers to each of these questions are worth 1 point each, for a total of 8 points

a. What do the bars on a histogram represent?
b. How many bars can a histogram have?
c. What do the heights of the bars tell you
d. What is on the x-axis and y-axis of a histogram
e. What does the tallest bar on a histogram tell you?
f. What does the shortest bar on a histogram tell you?
g. What are some uses for histograms, why would you want to look at a histogram of some numbers that you collected?
h. Imagine you had two histograms, one was very wide and spread out, the other was very narrow with a very tall peak. Which histogram would you expect to contain more consistent numbers (numbers that are close to each other), explain why.

Rubric

General grading.

- You will receive 0 points for missing answers (say, if you do not answer question c, then you will receive 0 points for that question)
- You must write in complete sentences. Point form sentences will be given 0 points.
- Completely incorrect answers will receive 0 points. For example, if you incorrectly describe what the x and y-axes refer to, then you will receive 0 points for that question.
- If your answer is generally correct but very difficult to understand and unclear you may receive half points for the question.

Chapter 2

Lab 2: Descriptive Statistics

Describing comic sensibility is near impossible. It's sort of an abstract silliness, that sometimes the joke isn't the star. —Dana Carvey

The purpose of this lab is to show you how to compute basic descriptive statistics, including measures of central tendency (mean, mode, median) and variation (range, variance, standard deviation).

2.1 General Goals

1. Compute measures of central tendency using software
2. Compute measures of variation using software
3. Ask some questions of a data set using descriptive statistics

2.1.1 Important info

We will be using data from the gapminder project. You can download a small snippet of the data in .csv format from this link (note this dataset was copied from the gapminder library for R) gapminder.csv. If you are using R, then you can install the gapminder package. This method is described later in the R section.

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 new found 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 occurring 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))
```

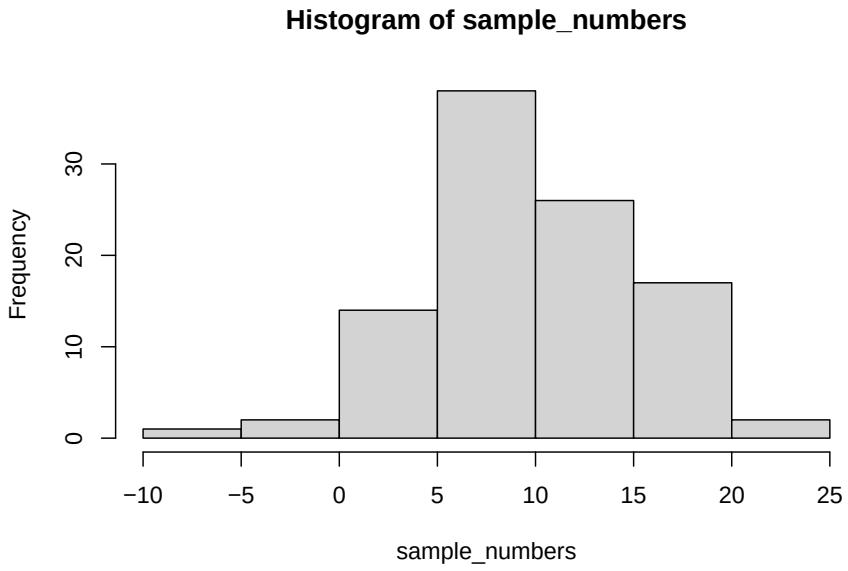
```
## [1] 1
```

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 histogram 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] -6.116085 21.120523
```

2.2.3.2 var = variance

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

```
var(sample_numbers)
```

```
## [1] 27.49566
```

2.2.3.3 sd = standard deviation

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

```
sd(sample_numbers)
```

```
## [1] 5.243631
```

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

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

```
## [1] 5.243631
```

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] 1002.548
```

```
length(sample_numbers)
```

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

```
mean(sample_numbers)
```

```
## [1] 10.02548
```

```
median(sample_numbers)
```

```
## [1] 10.16463
```

```
my_mode(sample_numbers)
```

```
## [1] 10.54972
```

```
range(sample_numbers)
```

```
## [1] -0.3067084 26.5661968
```

```
var(sample_numbers)
```

```
## [1] 25.48026
```

```
sd(sample_numbers)
```

```
## [1] 5.047797
```

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] 9.636791
```

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

```
## [1] 7.364333
```

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

```
## [1] 12.85212
```

```
# 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 everything 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
##    <chr>      <dbl>
##  1 A           9.64
##  2 B           7.36
##  3 C          12.9
##  4 D           8.05
##  5 E           6.94
##  6 F           9.47
##  7 G           7.94
##  8 H           9.49
##  9 I           9.59
## 10 J           9.82
```

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	9.636791
B	7.364333
C	12.852121
D	8.046196
E	6.944765
F	9.468628
G	7.937272
H	9.491867
I	9.585712
J	9.823514

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	9.636791	4.863356
B	7.364333	3.122062
C	12.852121	4.996556
D	8.046196	3.303041
E	6.944765	3.546821
F	9.468628	5.443471
G	7.937272	2.800717
H	9.491867	5.070831
I	9.585712	4.190495
J	9.823514	3.098157

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	9.636791	4.863356	0.1364273	17.64975
B	7.364333	3.122062	3.6478959	12.42980
C	12.852121	4.996556	2.0882922	18.39466
D	8.046196	3.303041	2.4214303	11.46234
E	6.944765	3.546821	-1.1854933	11.79227
F	9.468628	5.443471	1.2916955	16.47118
G	7.937272	2.800717	3.9929287	12.80781
H	9.491867	5.070831	1.2040815	17.31615
I	9.585712	4.190495	3.0639624	16.48314
J	9.823514	3.098157	5.4935171	14.78114

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.

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

Note: The above code will only work if you have installed the gapminder package. Make sure you are connected to the internet, then choose the Packages tab from the bottom right panel, and choose install. Thens search for gapminder, choose it, and install it.

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

(4 points - Pass/Fail)

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

1. What is the mean, standard deviation, minimum and maximum life expectancy for all the gapminder data (across all the years and countries). Hint: do not use `group_by`
2. What is the mean, standard deviation, minimum and maximum life expectancy for all of the continents in 2007, the most recent year in the dataset. Hint: add another pipe using `filter(year==2007) %>%`

2.2.7 Writing assignment

(8 points - Graded)

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.)

Your writing assignment is to answer these questions in full sentences using simple plain langauge:

1. Define the mode.
2. Explain what would need to happen in order for a set of numbers to have two modes
3. Define the median
4. Define the mean
5. Define the range
6. When calculating the standard deviation, explain what the difference scores represent
7. Explain why the difference scores are squared when calculating the standard deviation
8. If one set of numbers had a standard deviation of 5, and another had a standard deviation of 10, which set of numbers would have greater variance, explain why.

Rubric

General grading.

- You will receive 0 points for missing answers (say, if you do not answer question c, then you will receive 0 points for that question)
- You must write in complete sentences. Point form sentences will be given 0 points.
- Completely incorrect answers will receive 0 points.
- If your answer is generally correct but very difficult to understand and unclear you may receive half points for the question

2.2.8 Extra Practice Problems (Optional)

---

1. Using the life expectancy data set, produce a table of output showing the descriptive statistics (measures of central tendency and variability) for both years 1800 and 1934 (during the Great Depression).
2. Plot histograms of life expectancy for both years. How are these distributions different? (Hint: Plot these on the same axes so that they are comparable).

Chapter 3

Lab 3: Correlation

If ... we choose a group of social phenomena with no antecedent knowledge of the causation or absence of causation among them, then the calculation of correlation coefficients, total or partial, will not advance us a step toward evaluating the importance of the causes at work. —Sir Ronald Fisher

In lecture and in the textbook, we have been discussing the idea of correlation. This is the idea that two things that we measure can be somehow related to one another. For example, your personal happiness, which we could try to measure say with a questionnaire, might be related to other things in your life that we could also measure, such as number of close friends, yearly salary, how much chocolate you have in your bedroom, or how many times you have said the word Nintendo in your life. Some of the relationships that we can measure are meaningful, and might reflect a causal relationship where one thing causes a change in another thing. Some of the relationships are spurious, and do not reflect a causal relationship.

In this lab you will learn how to compute correlations between two variables in software, and then ask some questions about the correlations that you observe.

3.1 General Goals

1. Compute Pearson's r between two variables using software
2. Discuss the possible meaning of correlations that you observe

3.1.1 Important Info

We use data from the World Happiness Report. A .csv of the data can be found here: WHR2018.csv

3.2 R

In this lab we use explore to explore correlations between any two variables, and also 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 scatter plots. 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)
```

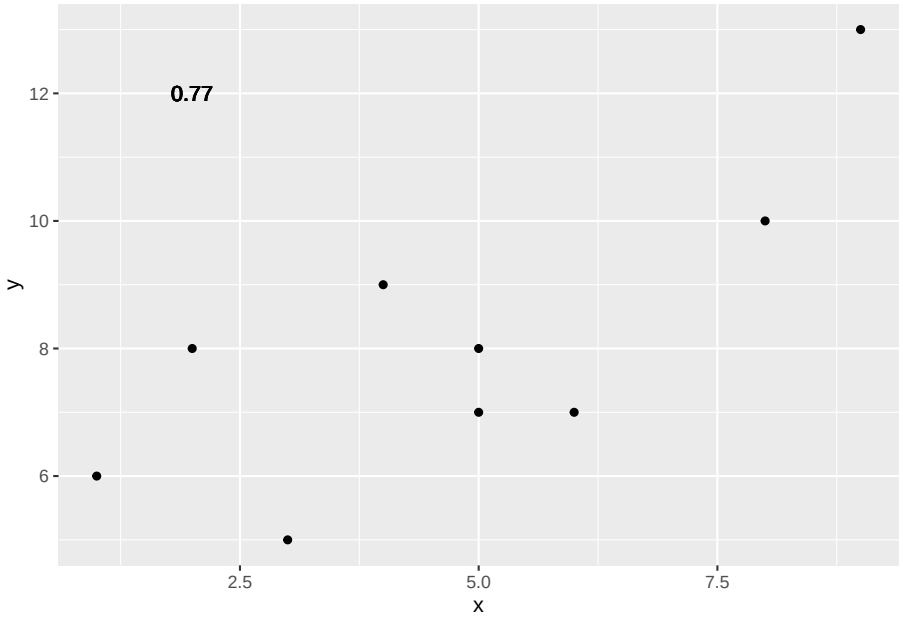
```
## [1] 0.76539
```

Well, that was easy.

3.2.1.1 scatterplots

Let's take our silly example, and plot the data in a scatter plot using ggplot2, and let's also return the correlation and print it on the scatter plot. 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 ))
```

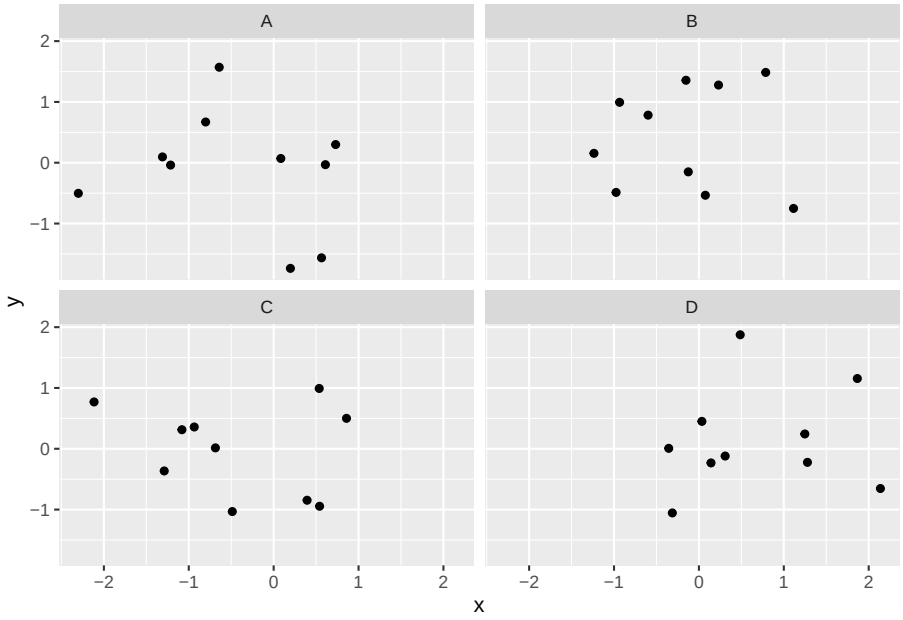


Wow, we're moving fast here.

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 scatter plots 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 scatter plots 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 scatter plot. We again use `dplyr` to do this:

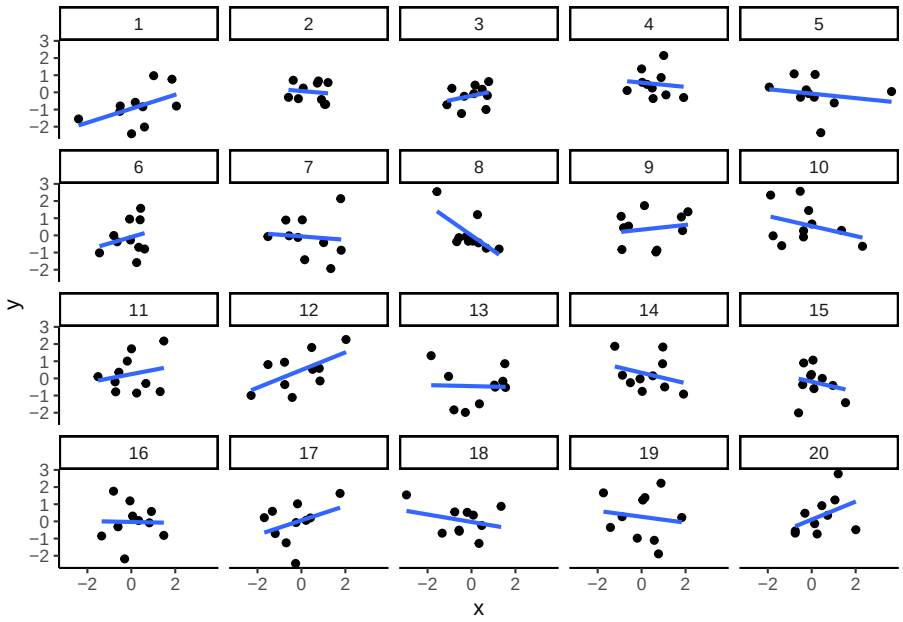
OK, we are basically ready to turn to some real data and ask if there are correlations between interesting variables…You will find that there are some… But before we do that, we do one more thing. This will help you become a little bit more skeptical of these "correlations".

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 scatter plot, 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 re-knitting 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 association between the x and y variable, or could have just occurred by chance.

3.2.2 World Happiness Report

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 data frame. Reminder, the following assumes that you have downloaded the RMarkdownsLab.zip file which contains the data file in the data folder.

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

You can also load the data using the following URL

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

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

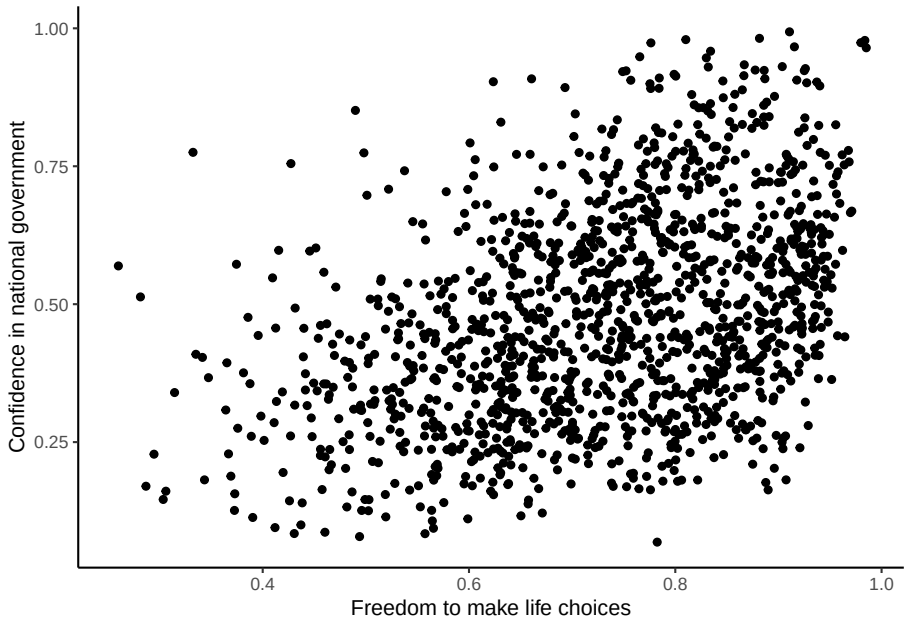
For the year 2017 only, does a countries measure for "freedom to make life choices" correlate with that countries measure for " Confidence in national government"?

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

```
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 occurred 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
```

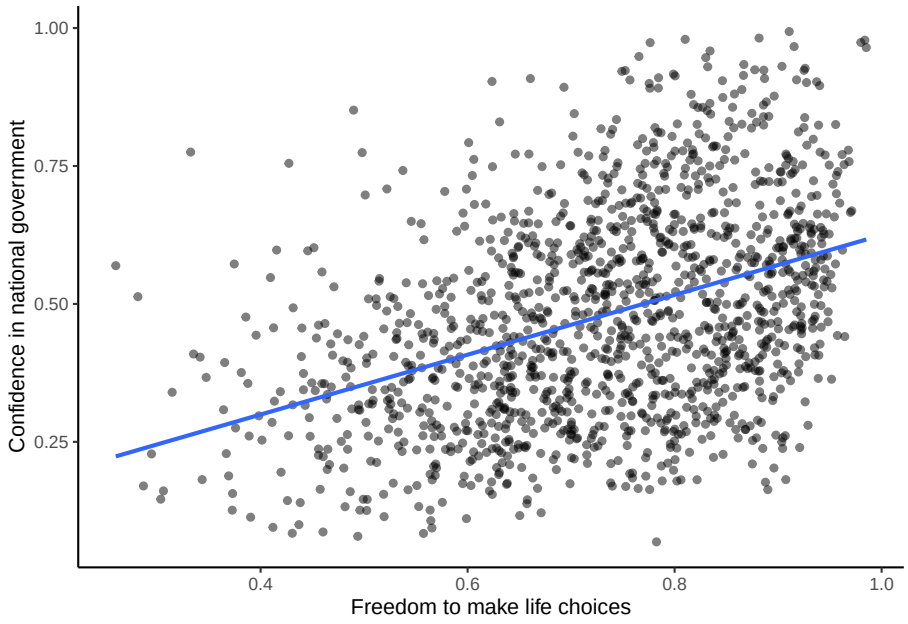
Now we see the correlation is .408.

Although the scatter plot 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

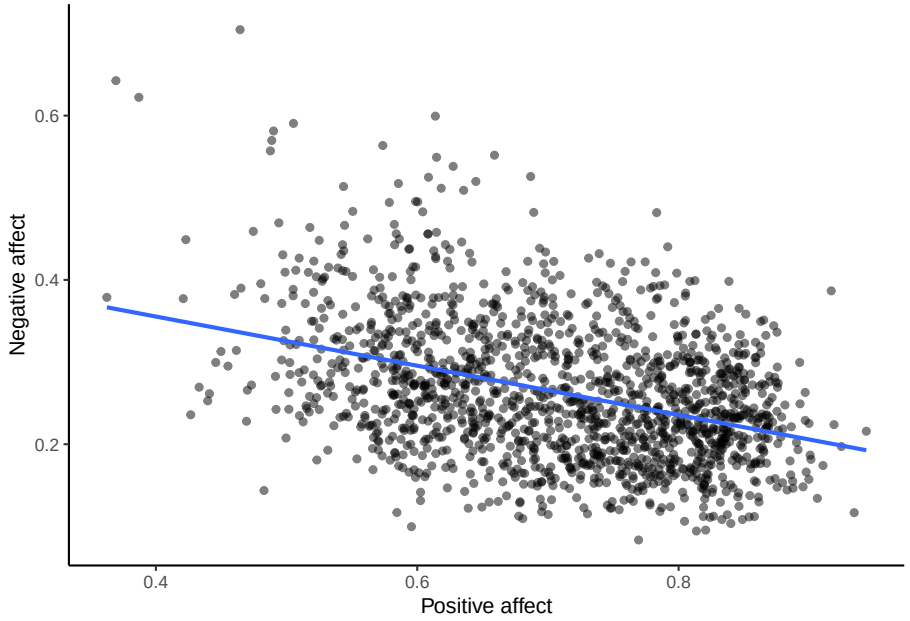
After all that work, we can now speedily answer more questions. For example, what is the relationship between positive affect in a country and negative affect in a country. I wouldn't be surprised if there was a negative correlation here: when positive feelings generally go up, shouldn't negative feelings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

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

This generalization exercise will explore the idea that correlations between two measures can arise by chance alone. There are two questions to answer. For each question you will be sampling random numbers from uniform distribution. To conduct the estimate, you will be running a simulation 100 times. The questions are:

1. Estimate the range (minimum and maximum) of correlations (using pearons's r) that could occur by chance between two variables with n=10.

2. Estimate the range (minimum and maximum) of correlations (using pearons's r) that could occur bychance between two variables with n = 100.

Use these tips to answer the question.

Tip 1: You can use the `runif()` function to sample random numbers between a minimum value, and maximum value. The example below sample 10 (n=10) random numbers between the range 0 (min = 0) and 10 (max=10). Everytime you run this code, the 10 values in x will be re-sampled, and will be 10 new random numbers

```
x <- runif(n=10, min=0, max=10)
```

Tip 2: You can compute the correlation between two sets of random numbers, by first sampling random numbers into each variable, and then running the `cor()` function.

```
x <- runif(n=10, min=0, max=10)
y <- runif(n=10, min=0, max=10)
cor(x,y)
```

```
## [1] -0.1745448
```

Running the above code will give different values for the correlation each time, because the numbers in x and y are always randomly different. We might expect that because x and y are chosen randomly that there should be a 0 correlation. However, what we see is that random sampling can produce "fake" correlations just by chance alone. We want to estimate the range of correlations that chance can produce.

Tip 3: One way to estimate the range of correlations that chance can produce is to repeat the above code many times. For example, if you ran the above code 100 times, you could save the correlations each time, then look at the smallest and largest correlation. This would be an estimate of the range of correlations that can be produced by chance. How can you repeat the above code many times to solve this problem?

We can do this using a `for` loop. The code below shows how to repeat everything inside the for loop 100 times. The variable `i` is an index, that goes from 1 to 100. The `saved_value` variable starts out as an empty variable, and then we put a value into it (at index position i, from 1 to 100). In this code, we put the sum of the products of x and y into the `saved_value` variable. At the end of the simulation, the `save_value` variable contains 100 numbers. The `min()` and `max()` functions are used to find the minimum and maximum values for each of the 100 simulations. You should be able to modify this code by replacing

`sum(x*y)` with `cor(x,y)`. Doing this will allow you to run the simulation 100 times, and find the minimum correlation and maximum correlation that arises by chance. This will be estimate for question 1. To provide an estimate for question 2, you will need to change `n=10` to `n=100`.

```
saved_value <- c() #make an empty variable
for (i in 1:100){
  x <- runif(n=10, min=0, max=10)
  y <- runif(n=10, min=0, max=10)
  saved_value[i] <- sum(x*y)
}

min(saved_value)
```

```
## [1] 93.29848
```

```
max(saved_value)
```

```
## [1] 424.6493
```

3.2.4 Writing assignment

Answer the following questions with complete sentences. 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.)

1. Imagine a researcher found a positive correlation between two variables, and reported that the r value was +.3. One possibility is that there is a true correlation between these two variables. Discuss one alternative possibility that would also explain the observation of +.3 value between the variables.
2. Explain the difference between a correlation of r = .3 and r = .7. What does a larger value of r represent?
3. Explain the difference between a correlation of r = .5, and r = -.5.

3.2.5 Practice Problems

---

1. For the year 2005 ONLY, find the correlation between "perceptions of corruption" and "positive affect". Create a scatterplot to visualize this relationship. What are your conclusions about the relationship between affect and perceived corruption? Is this surprising to you?

2. What has happened to log GDP (consider this a measure of GDP) in the United States ONLY with time (as the year has increased)? Explain this relationship and provide a scatterplot.

3. Which country (or countries) have seen a more consistent and strong increase in log GDP over time? Which country (or countries) have seen a decrease over time?

Chapter 4

Lab 4: Normal Distribution & Central Limit Theorem

By a small sample, we may judge of the whole piece. —Miguel de Cervantes from Don Quixote

4.1 General Goals

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

4.2 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 that 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 cases you 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" "B" "A" "B" "B" "B" "B" "B" "A" "B"
```

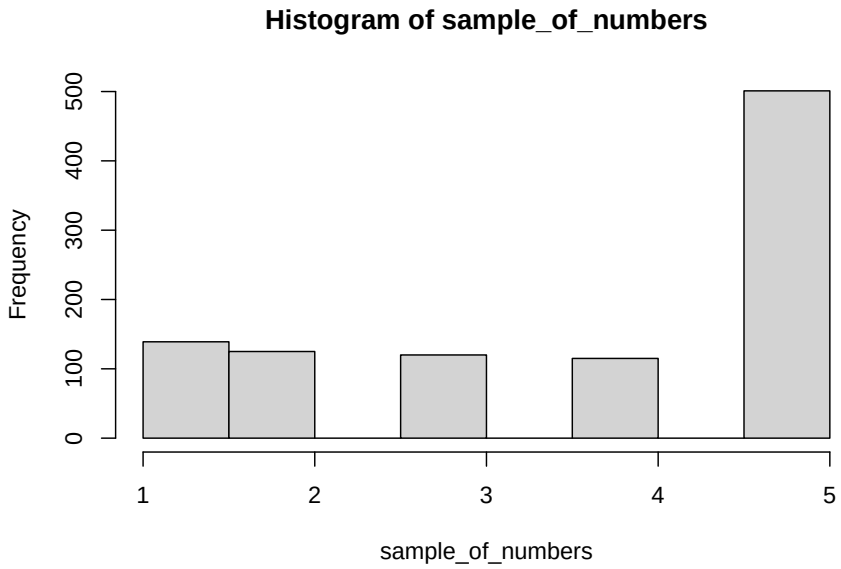
Here the sample function randomly picks A or B each time. We set it do this 10 times, so our sample has 10 things in it. We set `replace=TRUE` so that after each sample, we put the item back into the gumball machine and start again. Here's another example with numbers

```
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 5 4 5 5 5 1 2 3 5 5 4 4 5 5 5 5 5 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)
```

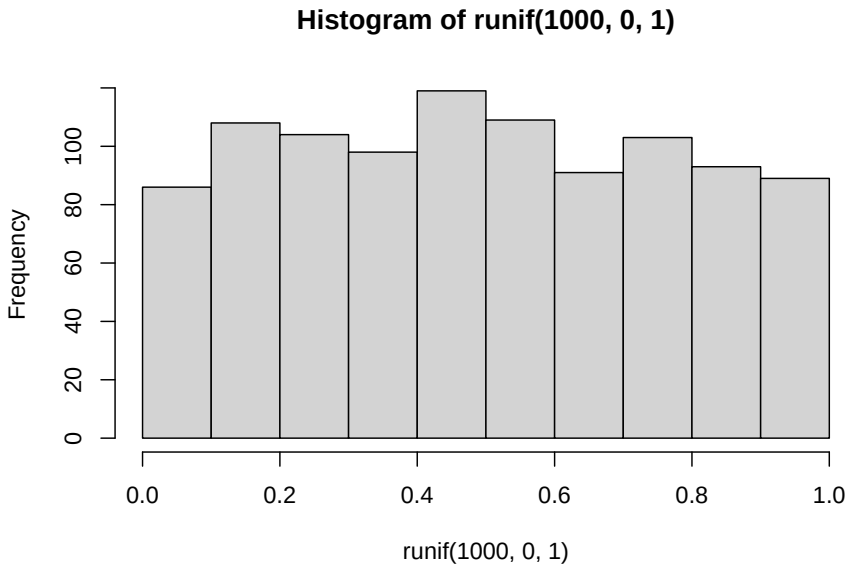


We are looking at lots of samples from our little gumball machine of numbers. We put more 5s in, and voila, more 5s come out of in our big sample of 1000.

4.2.1.2 runif uniform distribution

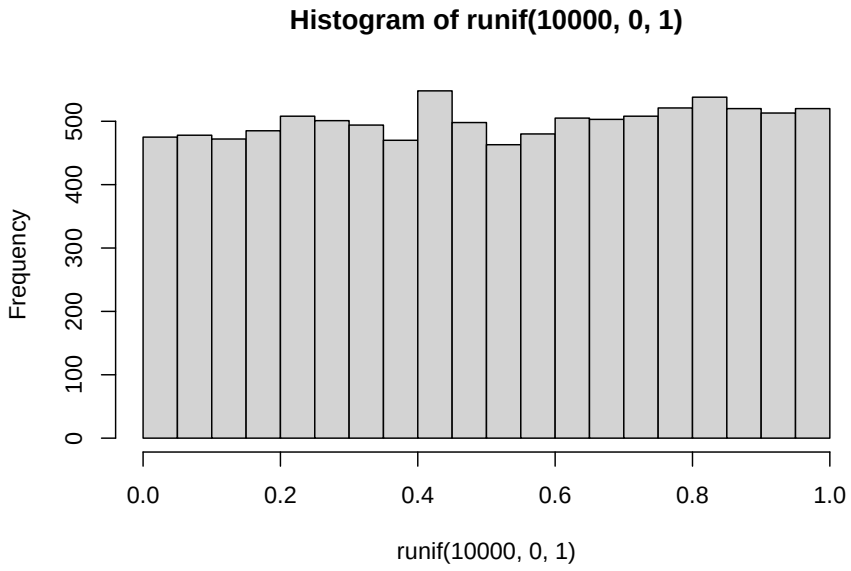
We can sample random numbers between any range using the `runif(n, min=0, max = 1)` function for the uniform distribution. We discussed this in the textbook. A uniform distribution is flat, and all the numbers between the min and max should occur roughly equally frequently. Let's take 1000 random numbers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

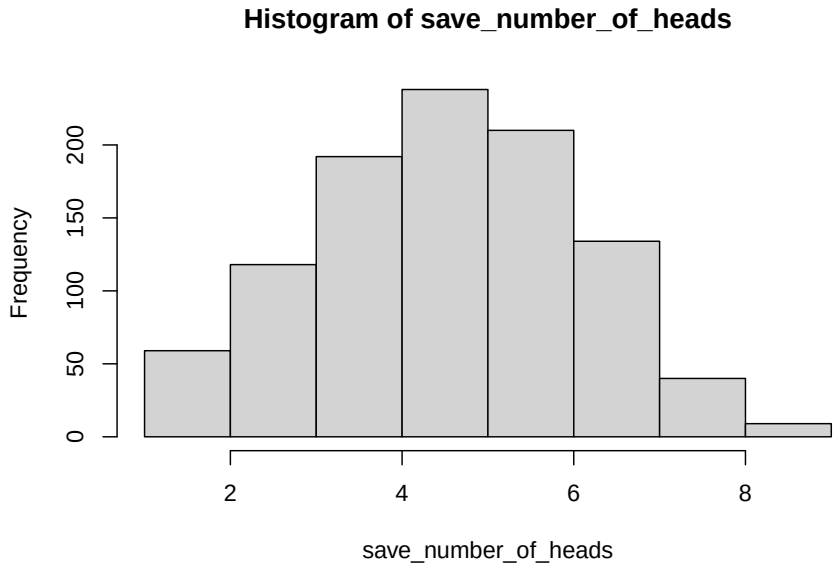
```
coin_flips <- rbinom(10,1,.5)
sum(coin_flips)
```

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

Alright, so we get the sum, which tells us the number of heads. But, should we always 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)
```

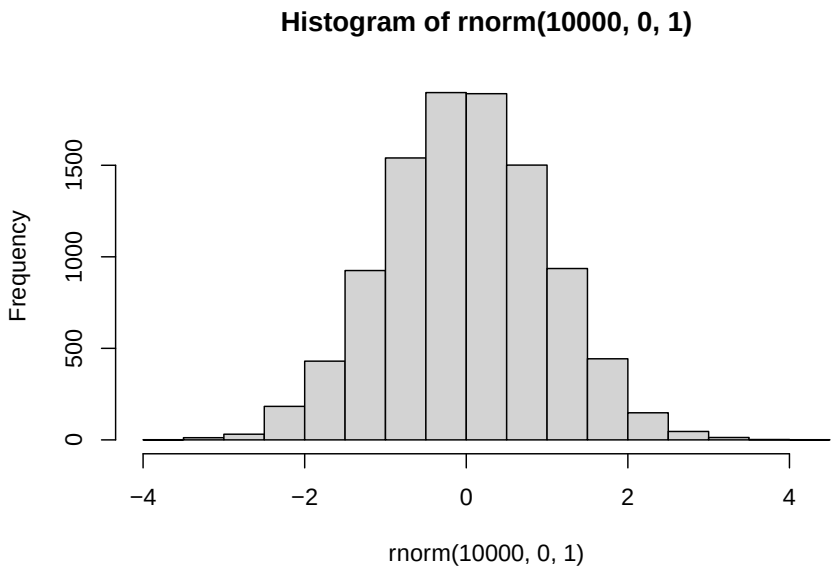


See, that wasn't too painful. Now we see another histogram. The histogram shows us the frequency observing different numbers of heads (for 10 flips) across the 1000 simulations. 5 happens the most, but 2 happens sometimes, and so does 8. All of the possibilities seem to happen sometimes, some more than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

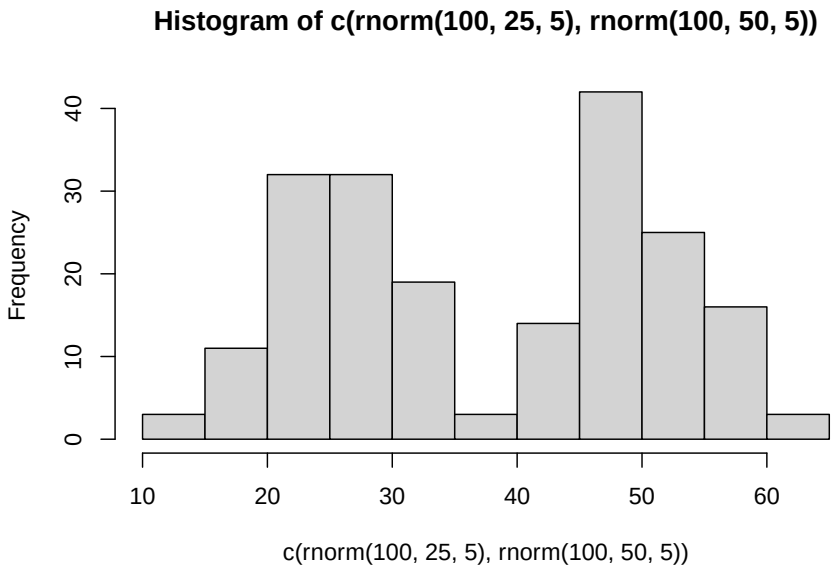


There it is, a bell-shaped normal distribution with a mean of 0, and a standard deviation of 1. You've probably seen things like this before. Now you can sample numbers from normal distributions with any mean or standard deviation, just by changing those parts of the `rnorm` function.

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.

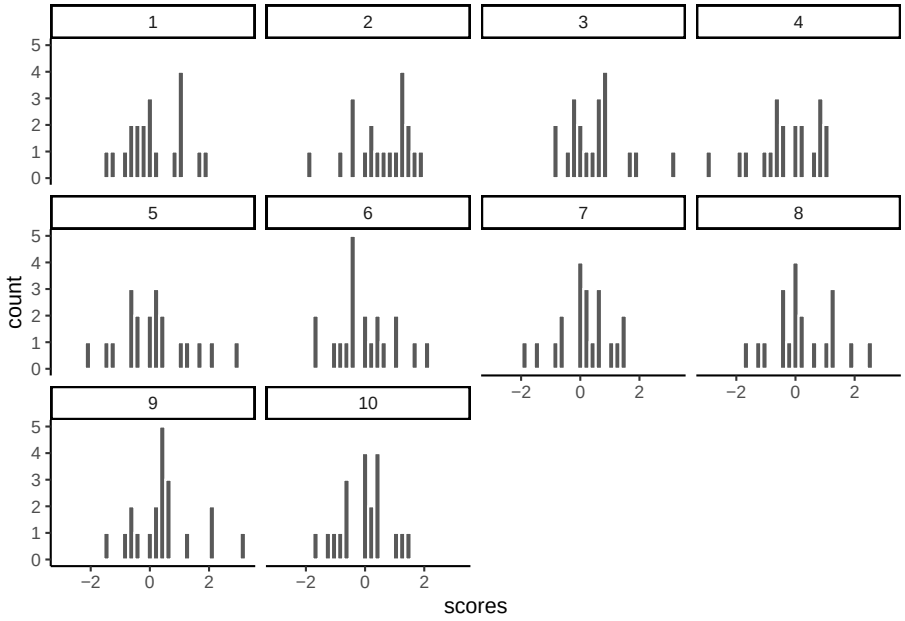
4.2.2.1 Taking multiple samples of the same size

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 histograms, 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 data frame. You can see there is a column with numbers 1 to 10, these are the sample names. There are also 20 scores for each in the 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)
```

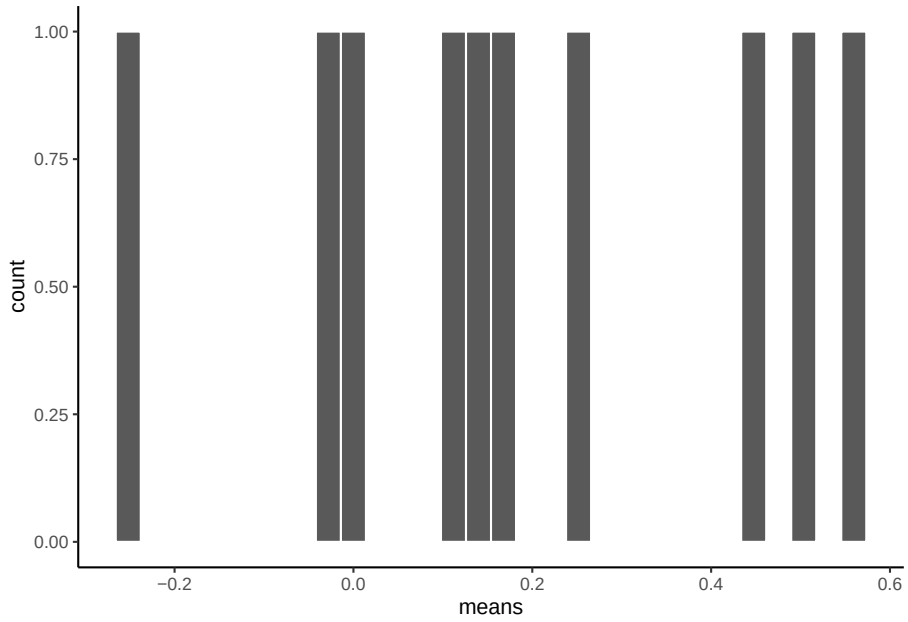
samples	means
1	0.1242773
2	0.5616046
3	0.5163987
4	-0.2496774
5	0.1534819
6	0.0060566
7	0.1632529
8	0.2536639
9	0.4376141
10	-0.0357752

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.

4.2.2.3 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] 0.1930898
```

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.

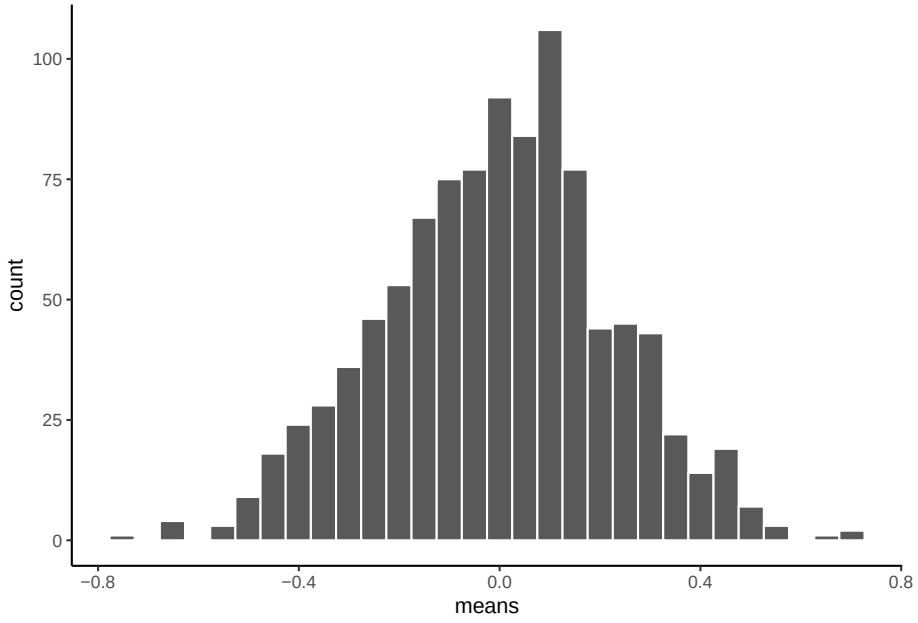
4.2.2.4 simulating the distribution of sample means

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()
```



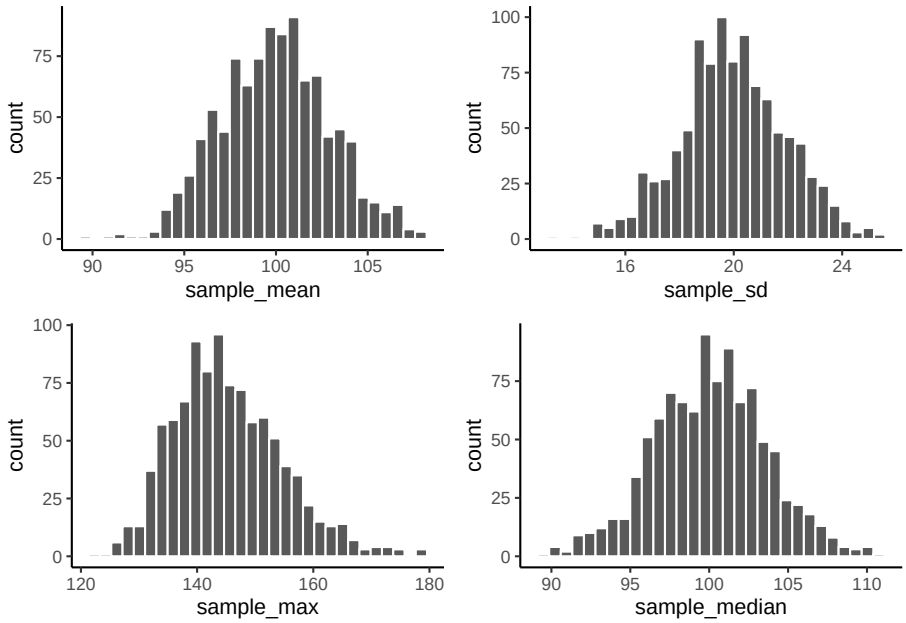
There, that looks more like a sampling distribution of the sample means. Notice it's properties. It is centered on 0, which tells us that sample means are mostly around zero. It is also bell-shaped, like the normal distribution it came from. It is also quite narrow. The numbers on the x-axis don't go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



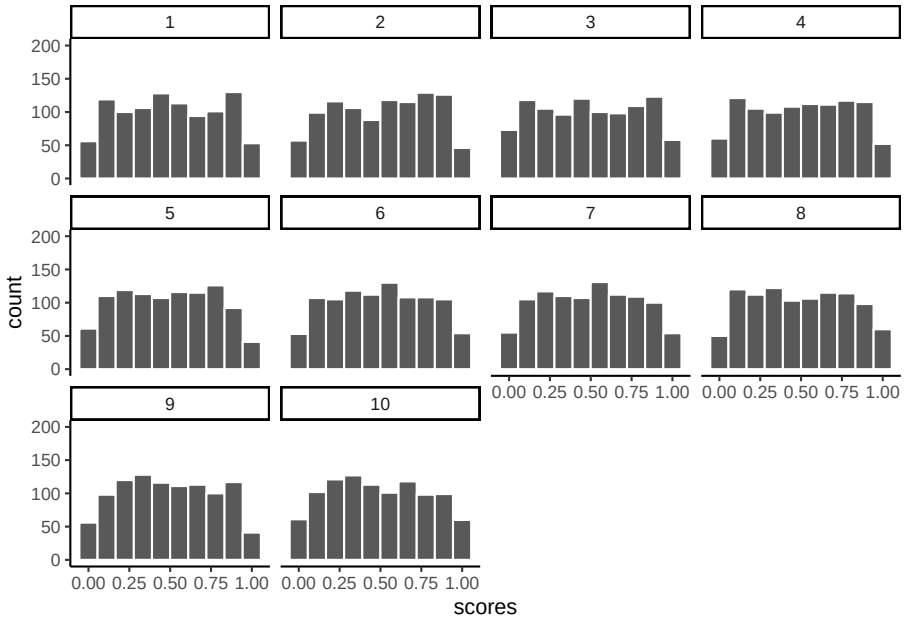
From reading the textbook and attending lecture, you should be able to start thinking about why these sampling statistic distributions might be useful…For now, just know that you can make a sampling statistic for pretty much anything in R, just by simulating the process of sampling, measuring the statistic, 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 demonstrate this the next bit of code is modified from what we did earlier. 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 histograms 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)
```

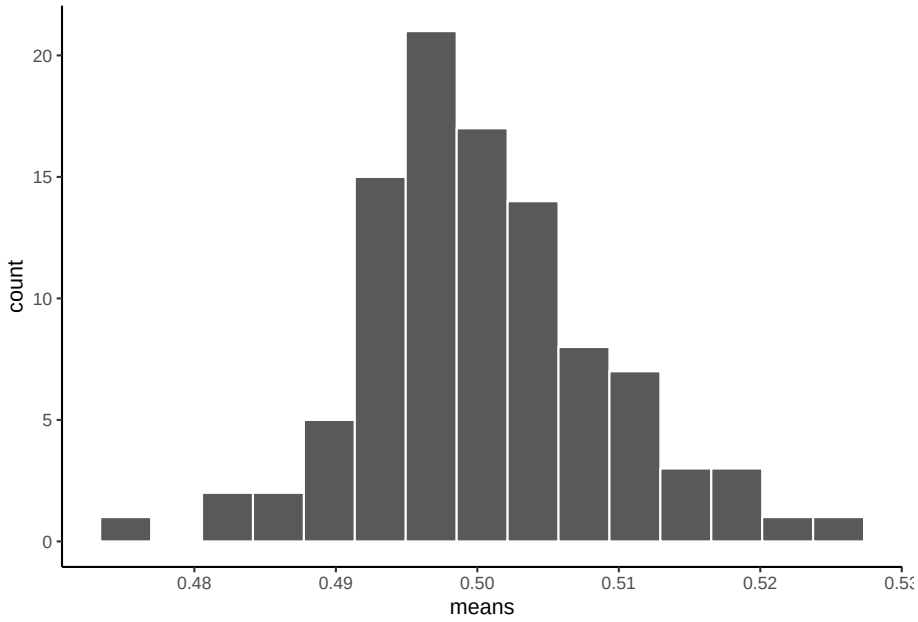


We took samples from a flat uniform distribution, and the samples themselves look like that same flat distribution.

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 normal-ish. 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).

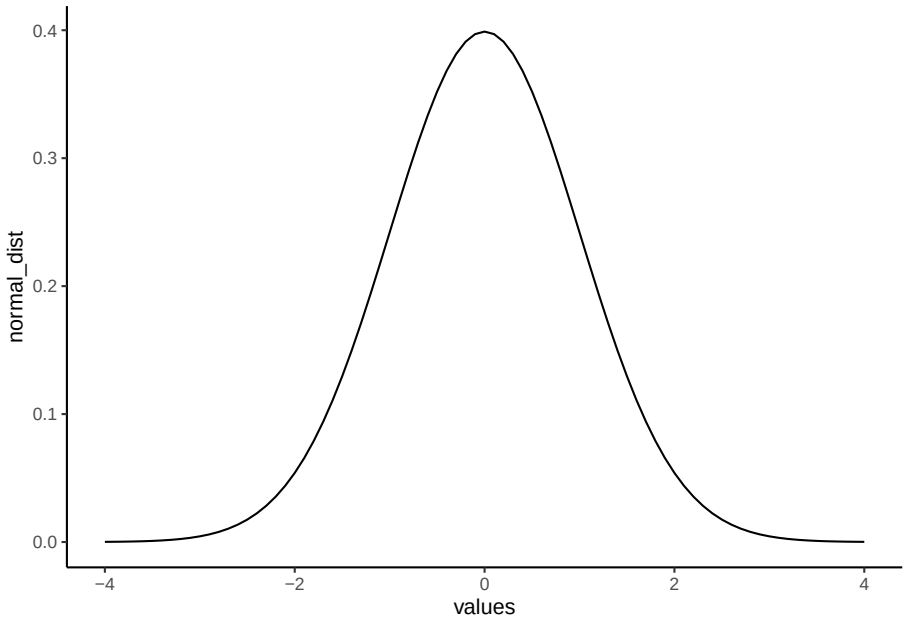
We can use this knowledge. If we learn a little bit more about normal distributions, and how they behave and work, we can take that and use it to understand our sample means better. This will become more clear as head into the topic of statistical inference next week. This is all a build up for that.

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 already 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 distribution 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 probabilities.

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 a lot.

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 a lot. Numbers bigger or smaller than 1 also happen fairly often, but less often. Number bigger than 2 don't happen a lot, 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 a lot or not? The numbers don't 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] 58.05807
```

```
print(my_sd)
```

```
## [1] 29.28992
```

3. subtract the mean from your numbers

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

```
##  [1] -48.167182  14.980089  42.267161  29.866269 -25.555337 -24.259125
##  [7]  23.072389  29.735354 -23.105184 -44.203498   3.726066   2.305146
## [13]  33.643191  26.586656 -20.377513 -32.471566  16.368551 -28.228258
## [19] -13.144142  36.960934
```

4. divide by the standard deviation

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

```
##  [1] -1.64449664  0.51144172  1.44306147  1.01967723 -0.87249583 -0.82824131
##  [7]  0.78772444  1.01520761 -0.78884410 -1.50917080  0.12721323  0.07870098
## [13]  1.14862678  0.90770653 -0.69571752 -1.10862582  0.55884579 -0.96375318
## [19] -0.44875986  1.26189926
```

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 a lot, bigger numbers closer to 1 happen less often, but still fairly often, and numbers bigger than 2 or 3 hardly happen at all.

4.2.7 Generalization Exercise

(1 point - Pass/Fail)

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

1. Simulate the sampling distribution of the mean for sample-size =10, from a normal distribution with mean =100, and standard deviation = 25. Run the simulation 1000 times, taking 1000 samples, and computing the sample mean each time.

a. Plot the sampling distribution of the mean in a histogram
b. Report the mean of the sampling distribution of the mean
c. Report the standard deviation of the sampling distribution of the mean

2. Repeat all of the above, except change the sample-size to 100 for all simulations

a. Plot the sampling distribution of the mean in a histogram
b. Report the mean of the sampling distribution of the mean
c. Report the standard deviation of the sampling distribution of the mean

4.2.8 Writing assignment

(2 points - Graded)

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.)

1. Explain the concept of sampling error (1 point)

2. Explain why the standard deviation of the sampling distribution of mean gets smaller as sample-size increases (1 point)

General grading.

- You must write in complete sentences. Point form sentences will be given 0 points.
- Completely incorrect answers will receive 0 points.
- If your answer is generally correct but very difficult to understand and unclear you may receive half points for the question

4.2.9 Practice Problems

"Professor, do you grade on a curve?"

This is probably the most commonly-asked question in Statistics class. Everyone assumes that grading on a curve will benefit them. But does it?

Here is a link to an SPSS file containing 50 students' exam grades (let's say it's the final exam for a Statistics class).

1. Create a table containing the mean and standard deviation for this sample of scores. Now, produce a frequency histogram of the score data. Describe the distribution.

2. Transform each student's score into a Z-score (you can use either method shown in this tutorial). Now, plot the frequency histogram of this Z-score distribution. Compare it to the raw score distribution. How are they the same? How are they different?

3. Imagine you are a student in this class who received a 90 on this exam. However, the Professor has decided to GRADE ON A CURVE (shock! awe!), such that only the top 10% of the class receives an A (this professor only gives whole grades, no minuses or pluses). Calculate the z-score that corresponds to a raw score of 90 on this exam. Will you get an A with this grade? Why or why not?

Chapter 5

Lab 5: Fundamentals of Hypothesis Testing

The null hypothesis is never proved or established, but is possibly disproved, in the course of experimentation. Every experiment may be said to exist only to give the facts a chance of disproving the null hypothesis. —R. A. Fisher

5.1 R

From here on, we will be focusing on making sense of data from experiments. In all of this, we use experiments to ask a question about whether one thing causes change (influences) another thing. Then, we look at the data to help us answer that question. In general, we expect to find a difference in our measurement between the conditions of the experimental manipulation. We expect to find a difference when the manipulation works, and causes change in our measure. We expect not to find a difference when the manipulation does not work, and does not cause change.

However, as you well know from reading the textbook, and attending the lectures. Experimental manipulations are not the only thing that can cause change in our measure. Chance alone can cause change. Our measures are usually variable themselves, so they come along with some change in them due to sampling error.

At a minimum, when we conduct an experiment, we want to know **if the change we observed is bigger than the change that can be produced by chance**. Theoretically, random chance could produce most any change we might measure in our experiment. So, there will always be uncertainty about whether our manipulation caused the change, or chance caused the change. But, we can

reduce and evaluate that uncertainty. When we do this, we make **inferences** about what caused change in our experiments. This process is called **statistical inference**. We use **inferential statistics** as tools to help us make these inferences.

In this lab we introduce you to foundational concepts in **statistical inference**. This is also commonly termed **hypothesis testing**. But, for various reasons using that language to describe the process is tied to particular philosophies about doing statistical inference. We use some of that language here, so that you know what it means. But, we also use our own plain language, so you know what the point is, without the statistical jargon.

The textbook describes a few different statistical tests for building your conceptual understanding for statistical inference. In this lab, we work through some of them. In particular, we work through the Crump test, and the Randomization test. We show you how to conduct these tests in R on fake data, and real data.

5.1.1 The Crump Test

The Crump test is described more fully in the textbook here, but you already read that in preparation for this lab, right! I hope you did.

The big idea behind the Crump test is this. You find out what kind of differences between two conditions can be found by chance alone. This shows you what chance can do. Then, you compare what you actually found in one experiment, with the chance distribution, and make an inference about whether or not chance could have produced the difference.

5.1.1.1 Make assumptions about the distribution for your measurement

The first step in conducting the Crump test is to make a guess at the distribution behind your measurement. We will see in the next part how to do this from real data. For now, we just pick a distribution. For example, let's say we are measuring something that comes from a normal distribution with mean = 75 and standard deviation = 5. Perhaps, this is a distribution for how people perform on a particular test. The mean on the test is 75%, with a standard deviation of 5%. We know from last lab that 3 standard deviations away from the mean is pretty unlikely with this distribution. So, for example, most people never score above 90% (5*3=15, 75+15 = 90) on this test.

In this example situation, we might imagine an experiment that was conducted to determine whether manipulation A improves test performance, compared to a control condition where no manipulation took place. Using the Crump test, we can simulate differences that can occur by chance. We are formally simulating

the differences that could be obtained between two control conditions, where no manipulation took place.

To, restate our assumptions, we assume a single score for each subject is sampled from:

```
rnorm(n, mean=75, sd=5)
```

5.1.1.2 Make assumptions about N

In the real world, experiments have some number of subjects in each condition, this number is called N. For our simulation we, need to choose the number of subjects that we have. For this demonstration, we choose N = 20 in each condition.

5.1.1.3 Choose the number of simulations to run

We are going to run a fake experiment with no manipulation, and do this many times over (doing it many times over is called **monte carlo simulation**). Each time we will do this:

1. Sample 20 numbers for control group A using `rnorm(20, mean=75, sd=5)`
2. Sample 20 numbers for control group B using `rnorm(20, mean=75, sd=5)`
3. Compute the means for control group A and B
4. Compute the difference between the mean for group A and B
5. Save the differences score in a variable
6. Repeat as many times as we want

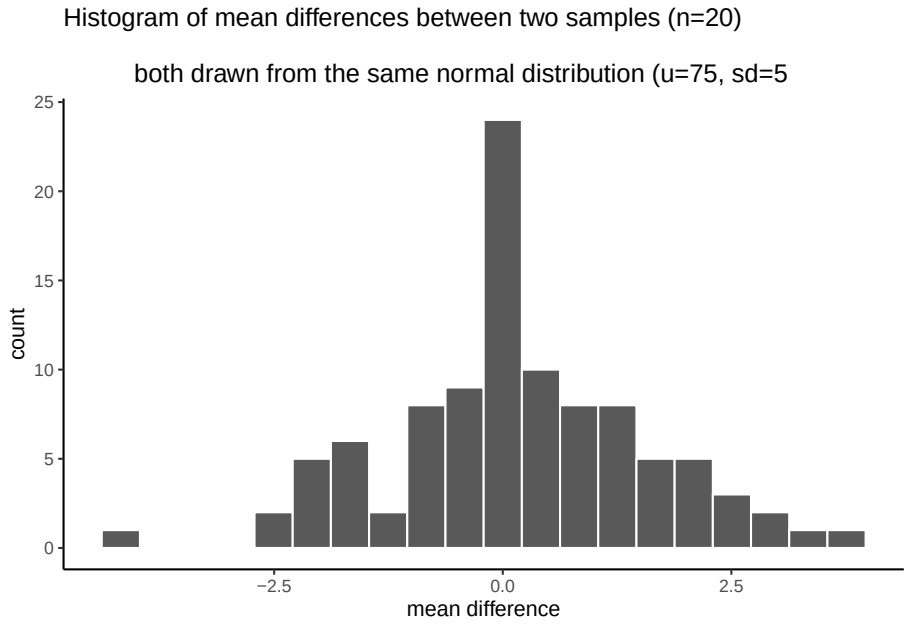
If we repeat the simulation 100 times, we will see the differences that can be produced by chance, when given the opportunity 100 times. For example, in a simulation like this, the biggest difference (the maximum value) only happens once. We can find that difference, and then roughly conclude that a difference of that big happens 1 out of 100 times just by chance. That's not a lot.

If we want to be more restrictive, we can make the simulation go to 1,000, or 10,000, or greater. Each time the maximum value will tell us what is the biggest thing chance did 1 out of 1000 times, or 1 out of 10,000 times.

The textbook uses 10,000 times. Let's use 100 times here to keep things simple.

5.1.1.4 Run the simluation

```r
library(ggplot2)

# set paramaters of simulation

sims_to_run <- 100
sample_n   <- 20
dist_mean  <- 75
dist_sd    <- 5

# run simulation

mean_differences <- length(sims_to_run)
for(i in 1:sims_to_run){
  mean_control_A      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_control_B      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_differences[i] <- mean_control_A - mean_control_B
}

# plot the  distribution of mean difference scores

plot_df <- data.frame(sim=1:sims_to_run,mean_differences)

ggplot(plot_df,aes(x=mean_differences))+
  geom_histogram(bins=20, color="white")+
  theme_classic()+
  ggtitle("Histogram of mean differences between two samples (n=20) \n
          both drawn from the same normal distribution (u=75, sd=5")+
  xlab("mean difference")
```



5.1.1.5 find the range

We can see that chance produces some differences that are non-zero. The histogram shows all the mean differences that were produced by chance. Most of the differences are between -2 and +2, but some of them are bit more negative, or a bit more positive. If we want to know what chance **did** do in this one simulation with 100 runs, then we need to find the range, the minimum and maximum value. This will tell us the most negative mean difference that chance did produce, and the most positive mean difference that chance did produce. Then, we will also know that chance **did not** produce any larger negative, or larger positive differences, in this simulation.

We use the `min()` and `max()` functions to get the minimum and maximum value.

```
min(mean_differences)
```

```
## [1] -4.228719
```

```
max(mean_differences)
```

```
## [1] 3.707135
```

We now know, that biggest negative difference was -4.229, and the biggest positive difference was 3.707. We also know that any mean difference inside the

range **was produced by chance** in our simulation, and any mean difference outside the range **was not produced by chance** in our simulation

5.1.1.6 Make inferences

This part requires you to think about the answers. Let's go through some scenario's.

1. You sample 20 numbers from a normal distribution with mean = 75, and standard deviation =5. The mean of your sample is 76. Then, you take another sample of the same size, from the same distribution, and the mean of your second sample is 78. The mean difference is +1 (or -1, depending on how you take the difference)

    a. **Question**: According to the histogram did a mean difference of 1 or -1 occur by chance?
    b. **Answer**: Yes, it is inside the range

2. Same as above, but the mean of your first sample is 74, and the mean of your second sample is 80, showing a mean difference of 6, or -6.

    a. **Question**: According to the histogram did a mean difference of 6 or -6 occur by chance?
    b. **Answer**: No, it is outside the range

3. You run an experiment. Group A receives additional instruction that should make them do better on a test. Group B takes the test, but without the instruction. There are 20 people in each group. You have a pretty good idea that group B's test scores will be coming from a normal distribution with mean = 75, and standard deviation = 5. You know this because you have given the test many times, and this is what the distribution usually looks like. You are making an educated guess. You find that the mean test performance for Group A (with additional instruction) was 76%, and the mean test performance for Group B (no additional instruction) was 75%. The mean difference has an absolute value of +1.

    a. **Question #1**: According to the histogram, could chance alone have produced a mean absolute difference of +1?
    b. **Answer**: Yes, it is inside the range
    c. **Question #2**: It looks like Group A did better on the test (on average), by 1%, compared to the control group B. Are you willing to believe that your additional instruction **caused the increase in test performance**?
    d. **Answer**: The answer is up to you. There is no correct answer. It could easily be the case that your additional instruction did not do anything at all, and that the difference in mean test performance

was produced by chance. My inference is that I do not know if my instruction did anything, I can't tell it's potential influence from chance.

4. Same as 3, except the group mean for A (receiving instruction) is 90%. The group mean for B (no instruction control) is 75%. The absolute mean difference is 15%.

    a. **Question #1**: According to the histogram, could chance alone have produced a mean absolute difference of +15?
    b. **Answer**: No, it is well outside the range
    c. **Question #2**: It looks like Group A did better on the test (on average), by 15%, compared to the control group B. Are you willing to believe that your additional instruction **caused the increase in test performance**?
    d. **Answer**: The answer is up to you. There is no correct answer. You know from the simulation that chance never produced a difference this big, and that producing a difference this big by chance would be like winning the lottery (almost never happens to you). My inference is that I believe chance did not produce the difference, I'm willing to believe that my instructional did cause the difference.

5.1.1.7 Planning your experiment

We've been talking about a hypothetical experiment where an instructor tests whether group A does better (when receiving additional instruction) on a test, compared to a group that does receives no additional instruction and just takes the test.

If this hypothetical instructor wanted to make an experiment, they would get to choose things like how many subjects they will put in each condition. How many subjects should they plan to get?

The number of subjects they plan to get will change what chance can do, and will change the sensitivity of their experiment to detect differences of various sizes, that are not due to chance.

We can use the simulation process to make informed decisions about how many subjects to recruit for an experiment. This is called **sample-size planning**. There are two goals here. The instructor might have a first goal in mind. They may be only interested in adopting a new method for instruction, if it actually improves test performance beyond more than 1% (compared to control). Differences of less than 1% are just not worth it for the instructor. They want bigger differences, they want to help their students improve more than 1%.

One problem for the instructor, is that they just don't know in advance how good their new teaching materials will be. Some of them will be good and

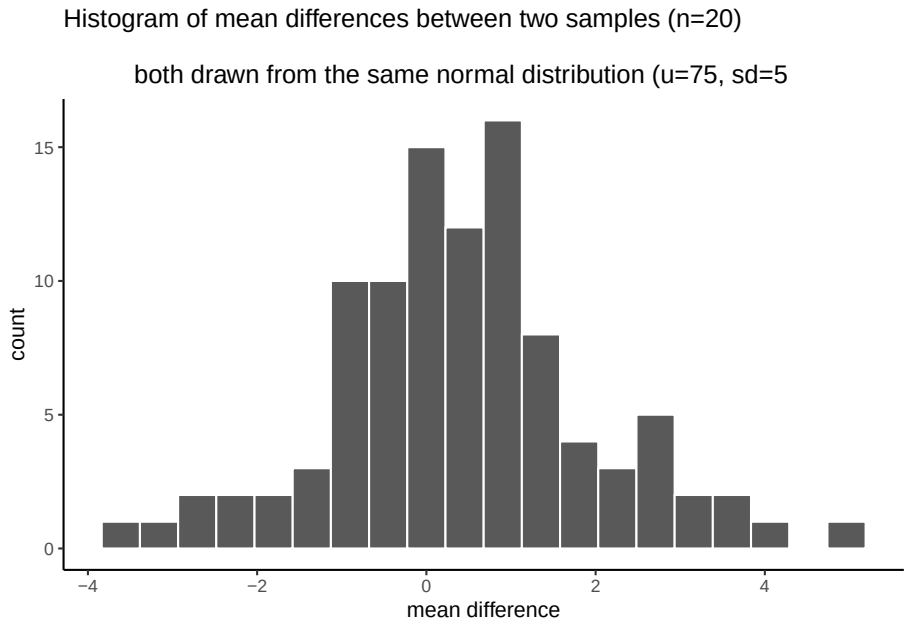
produce bigger differences, and some of them won't. The size of the difference from the manipulation can be unknown. However, this doesn't really matter for planning the experiment. The instructor wants to know that they can **find** or detect any real difference (not due to chance) that is say 2% or bigger. We can use the simulation to figure out roughly (or more exactly, depending on how much we work at it) how many subjects are needed to detect difference of at least 2%.

Notice, from our prior simulation, chance does produce differences of 2% some of the time (given 100 runs). The task now is to re-run the simulation, but use different numbers of subjects to figure out how many subjects are needed to always detect differences of 2%. To be simple about this, we are interested in producing a distribution of mean differences that never produces a mean difference of -2% to + 2% (not once out of 100 times). You can re-run this code, and change N until the min and max are always smaller than -2 to +2.

The code starts out exactly as it was before. You should change the number for `sample_n`. As you make the number bigger, the range (min and max) of the mean differences by chance will get smaller and smaller. Eventually it will be smaller than -2 to +2. When you get it this small, then the N that you used is your answer. Use that many subjects. If you run your experiment with that many subjects, **AND** you find a difference or 2 or greater, then you know that chance does not do this even 1 times out of 100.

```
library(ggplot2)

# set paramaters of simulation

sims_to_run <- 100
sample_n   <- 20
dist_mean  <- 75
dist_sd    <- 5

# run simulation

mean_differences <- length(sims_to_run)
for(i in 1:sims_to_run){
  mean_control_A      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_control_B      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_differences[i] <- mean_control_A - mean_control_B
}

# plot the  distribution of mean difference scores

plot_df <- data.frame(sim=1:sims_to_run,mean_differences)

ggplot(plot_df,aes(x=mean_differences))+
```

```
  geom_histogram(bins=20, color="white")+
  theme_classic()+
  ggtitle("Histogram of mean differences between two samples (n=20) \n
          both drawn from the same normal distribution (u=75, sd=5")+
  xlab("mean difference")
```




```
min(mean_differences)
```

```
## [1] -3.63058
```

```
max(mean_differences)
```

```
## [1] 4.94561
```

5.1.2 Crumping real data

In this example we look at how you can run a Crump test to evaluate the results in a published paper. The goal of this is to build up your intuitions about whether or not an observed difference could have been caused by chance. We take many liberties, and this is not an exact test of anything. However, we will see how a series of rough, and reasonable assumptions can be made to simulate the results of published experiments, even when exact information is not provided in the paper.

5.1.2.1 Test-enhanced learning

We have already been using an educational example. We've been talking about a manipulation that might be employed to help students learn something better than they otherwise would without the manipulation.

Research in Cognitive Psychology has discovered clear evidence of some teaching practices that really work to enhance memory and comprehension. One of these practices is called test-enhanced learning. Students who study material, and take a quick test (quiz) while they study, do better at remembering the material compared to students who just studied the whole time and did not take a quiz (why do you think we have so many quizzes, in the textbook and for this class? This is why. Prior research shows this will improve your memory for the content, so we are asking you to take the quizzes so that it helps you learn!).

Here is a link to a paper demonstrating the test-enhanced learning effect.

The citation is: Roediger III, H. L., & Karpicke, J. D. (2006). Test-enhanced learning: Taking memory tests improves long-term retention. Psychological science, 17(3), 249-255.

5.1.2.2 Brief summary

The subjects learned about some things in two conditions. In one condition (study-study) they studied some things, then studied them again. In the other condition (study-test) they studied some things, then took a quiz about the things rather then studying them one more time.

Everyone received follow up tests to see what they learned and remembered. They came back one week later and took the test. The researchers measured the mean proportion of things remembered in both conditions. They found the study-test condition had a higher mean proportion of remembered idea units than the study-study condition. So, the difference between the mean proportions suggest that taking a quick test after studying was beneficial for remembering the content. The researchers also conducted statistical tests, and they concluded the difference they found was not likely due to chance. Let's apply the Crump test to their findings, and see if we come to the same conclusion about the role of chance.

5.1.2.3 Estimate the paramaters of the distribution

To do the Crump test, we need to make assumptions about where the sample data is coming from. Download the paper from the link, then look at Figure 1. We will estimate our distribution by looking at this figure. We will be doing **informed guesstimation** (a real word I just made up).

Look only at the two bars for the `1 week` condition.

The mean for the study-study group is about .4. The mean for the study-test group is about .55. The results section reports that the actual mean for the study-study group was .42 (42%) and the mean for the study-test group was .56 (56%). Pretty close to our visual guesstimate.

The data show that the study-test group remembered .14 (14%, or .56-.42=.14) more idea units than the study-study group.

Estimating the mean

We can imagine for the moment that this difference could have been caused by chance. For example, in this case, both samples would have been drawn from the same distribution. For example, we might say that on average people remember about .49 idea units after a one week delay. I got .49 by averaging the .42 and the .56 together. We can use this as the mean for our distribution in the Crump test.

N

The paper says there were 120 subjects in total, and that different groups of subjects were measured in the three different delay conditions (5 minutes, 2 Days and 1 week). We will assume there were an equal number of subjects in each group. There were 3 groups, $120/3 = 40$, so we assume there were 40 subjects in the 1 week delay group

Estimating Standard Deviation

The paper doesn't directly report the standard deviation for the measurement of proportion idea units. But, we can guesstimate it visually. Look at the little bars with a line on them coming out of each bar. These are called error bars, and they represent the standard error of the mean.

These look like they are about .033 in length. We know the standard error of the mean (SEM) is the standard deviation divided by the sqare root of N. So, we can infer that the standard deviation is $.033 * \sqrt{40} = .21$.

The paper also reports cohen's D, which is a measure of effect size using the mean difference divided the standard deviation. Cohen's D was .83, so the standard deviation must have been $.14/.83 = .168$, which is pretty close to our guesstimate of .21.

5.1.2.4 Findings from the original study

One week after the initial learning conditions, subjects came back and took a retention test to see how much they learned. The mean proportion "idea units" recalled was:

1. study-study : 42% or .42
2. study-test : 56% or .56

The mean difference was .56-.42 = .14 or a whopping 14% improvements. That's actually pretty big.

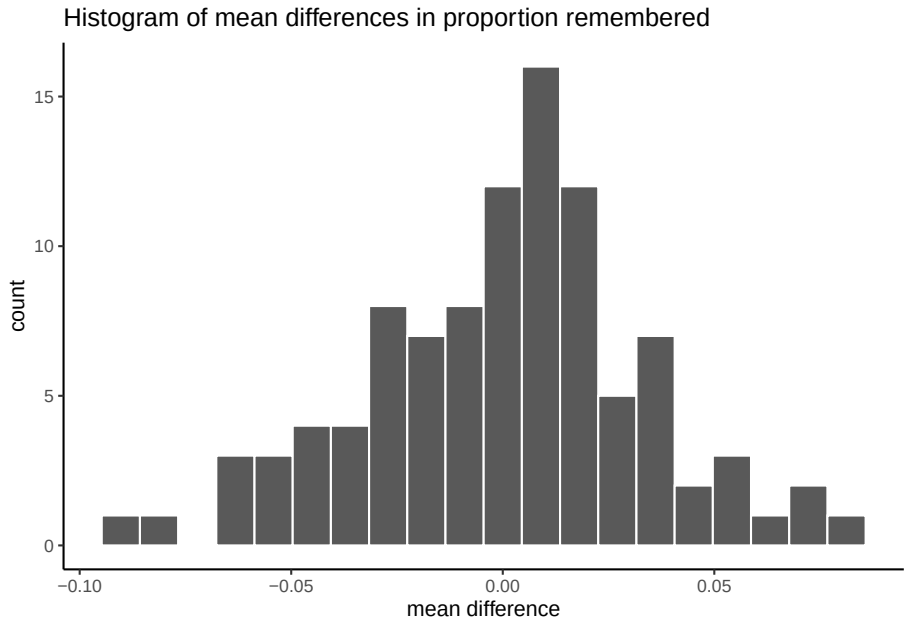
What we want to know is whether chance could produce a difference of 14%, or .14, just by itself.

5.1.2.5 Run the simulation

Now we can plug our numbers in to the Crump test simulation.

1. We run the simulation 100 times
2. Our sample N is 40
3. The mean of our distribution is .49
4. The standard deviation of the distribution is .168

```
# set paramaters of simulation

sims_to_run <- 100
sample_n   <- 40
dist_mean  <- .49
dist_sd    <- .168

# run simulation

mean_differences <- length(sims_to_run)
for(i in 1:sims_to_run){
  mean_control_A      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_control_B      <- mean(rnorm(sample_n, dist_mean, dist_sd))
  mean_differences[i] <- mean_control_A - mean_control_B
}

# plot the  distribution of mean difference scores

plot_df <- data.frame(sim=1:sims_to_run,mean_differences)

ggplot(plot_df,aes(x=mean_differences))+
  geom_histogram(bins=20, color="white")+
  theme_classic()+
  ggtitle("Histogram of mean differences in proportion remembered")+
  xlab("mean difference")
```




```
min(mean_differences)
```

```
## [1] -0.08802108
```

```
max(mean_differences)
```

```
## [1] 0.08350093
```

According to the simulation, the biggest negative difference was -0.088, and the biggest positive difference was 0.084. We also know that any mean difference inside the range **was produced by chance** in our simulation, and any mean difference outside the range **was not produced by chance** in our simulation.

A difference of .14 or 14% was never produced by chance, it was completely outside the range. Based on this analysis we can be fairly confident that the test-enhanced learning effect was not a fluke, it was not produced by chance. We have evidence to support the claim that testing enhances learning, and because of this we test you while you are learning to enhance your learning while you learn.

5.1.3 The Randomization Test

Unlike the Crump test, which was made to help you understand some basic ideas about how chance can do things, the Randomization test is a well-known,

very real, statistical test for making inferences about what chance can do. You will see that it is very similar to the Crump test in many respects. In fact, we might say that the Crump test is really just a randomization test.

You read about the randomization test in the textbook. We won't repeat much about that here, but we will show you how to do one in R.

To briefly remind you, in a randomization test we first obtain some data from an experiment. So we have a sample of numbers for group A, and Group B. We calculate the means for both groups (or other statistic we want to know about), and then see if they are different. We want to know if the difference we found could be produced by chance, so we conduct the randomization test on using the sample data that we have.

The major difference between the Crump test and the Randomization test, is that we make no assumption about the distribution that the sample data came from. We just randomize the sample data. We do:

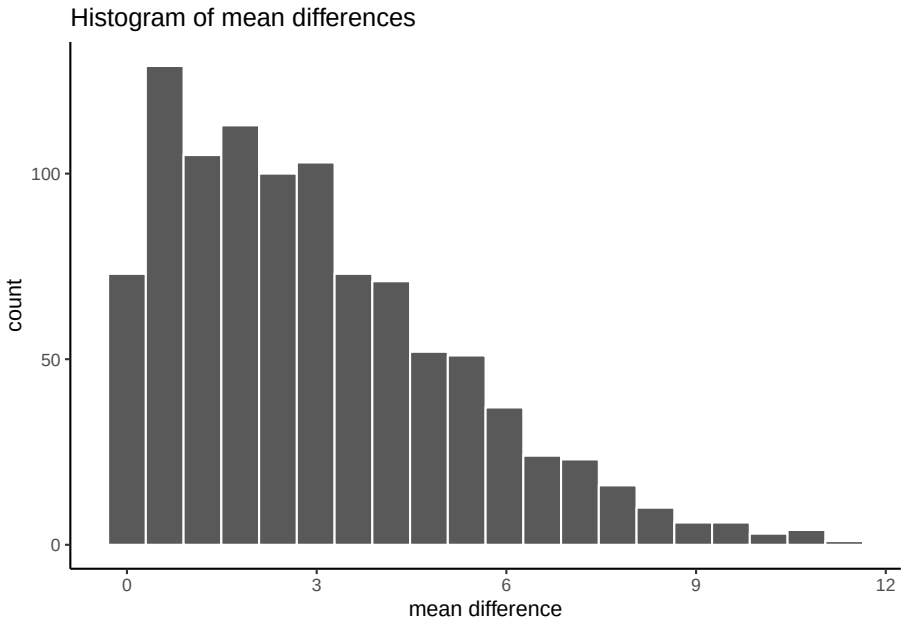
1. Take all the numbers from group A and B, put them in the same pot. Then randomly take the numbers out and assign them back into A and B. Then, compute the means, and the difference, and save the difference
2. Do this over and over
3. Plot the histogram of the mean differences obtained by shuffling the numbers. This shows you what chance can do.

5.1.3.1 Run the randomization test

Note, this time when we calculate the mean differences for each new group, we will take the absolute value of the mean difference.

```
# get sample numbers from one experiment
Group_A <- rnorm(20,50,10)
Group_B <- rnorm(20,50,10)

# randomize the numbers, compute mean difference, save the mean difference
mean_differences <- length(1000)
for(i in 1:1000){
  shuffle_numbers <- sample(c(Group_A, Group_B), replace=FALSE)
  new_group_A <- shuffle_numbers[1:20]
  new_group_B <- shuffle_numbers[21:40]
  mean_differences[i] <- abs(mean(new_group_A)-mean(new_group_B))
}

# plot the histogram
plot_df <- data.frame(sim=1:1000,mean_differences)
```

```
ggplot(plot_df,aes(x=mean_differences))+
  geom_histogram(bins=20, color="white")+
  theme_classic()+
  ggtitle("Histogram of mean differences")+
  xlab("mean difference")
```



The histogram shows us the kinds of absolute mean difference that happen by chance alone. So, in this data, a mean difference of 10 hardly wouldn't happen very often. A difference between 0 and 2.5 happens fairly often by chance.

5.1.3.2 Decision criteria

When you are determining whether or not chance could have produced a difference in your data, you might want to consider how stringent you want to be about accepting the possibility that chance did something. For example chance could produce a difference of 0 or greater 100% of the time. Or, it produces a difference of 10 or greater, a very small (less than .00001% of the time). How big does the difference need to be, before you will consider the possibility that chance probably didn't cause the difference.

Alpha. Researchers often set what is called an **alpha** criteria. This draws a line in the histogram, and says I will be comfortable assuming that chance did not produce my differences, when chance produces difference less than X percent of the time. This X percent, is the alpha value. For example, it is often set to 5%, or **p <= .05**.

Where is 5% of the time on our histogram? What mean difference or greater happens less than or equal to 5% of the time.

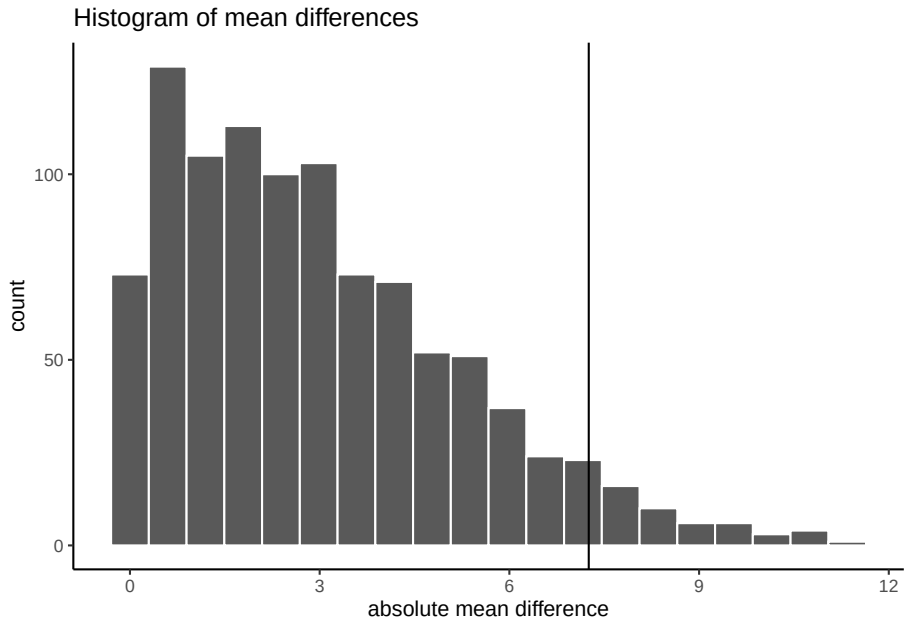
5.1.3.3 Finding the critical region

1. Take the simulated mean difference scores and order them from smallest to largest
2. We have 1000 numbers, ordered from smallest to largest.
3. The number in position 950 is the alpha location. All of the numbers from position 0 to 950, reflect 95% of the numbers. What is left over is 5% of the numbers.
4. Find the alpha cut-off, and plot it on the histogram

```
ordered_differences <- sort(mean_differences) # sort
alpha_cutoff <- ordered_differences[950] # pick 950th number
alpha_cutoff
```

```
## [1] 7.255284
```

```
# add to histogram using vline

ggplot(plot_df,aes(x=mean_differences))+
  geom_histogram(bins=20, color="white")+
  geom_vline(xintercept=alpha_cutoff)+
  theme_classic()+
  ggtitle("Histogram of mean differences")+
  xlab("absolute mean difference")
```



OK, so our alpha criterion or cutoff is located at 7.26 on the x-axis. This shows us that mean differences of 7.26 or larger happen only 5% of the time by chance.

You could use this information to make an inference about whether or not chance produced the difference in your experiments

1. When the mean difference is 7.26 or larger, you might conclude that chance did not produce the difference
2. When the mean difference is less than 7.26, you might conclude that chance could have produced the mean difference.

It's up to you to set your alpha criterion. In practice it is often set at p=.05. But, you could be more conservative and set it to p=.01. Or more liberal and set it to p=.1. It's up to you.

5.1.4 Generalization Exercise

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

1. Consider taking measurements from a normal distribution with mean = 100 and standard deviation = 25. You will have 10 subjects in two conditions (20 subject total). You will take 1 measurement (sample 1 score) of each subject.

Use the process for simulating the Crump test. Pretend that there are now differences between the conditions. Run a Crump test simulation 100 times.

a. Report the maximum mean difference in the simulation
b. Report the minimum mean difference in the simulation
c. Report a mean difference that, according to the simulation, was not observed by chance
d. Report a mean difference that, according to the simulation was observed by chance
e. What is the smallest mean difference that, if you found this difference in an experiment, you would be willing to entertain the possibility that the difference was unlikely to be due to chance (note this is a subjective question, give your subjective answer)

5.1.5 Writing assignment

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.)

Imagine you conduct an experiment. There is one independent variable with two levels, representing the manipulation. There is one dependent variable representing the measurement. Critically, the measurement has variability, so all of the samples will not be the same because of variability. The researcher is interested in whether the manipulation causes a change in the dependent measure. Explain how random chance, or sampling error, could produce a difference in the dependent measure even if the manipulation had no causal role.

a. Explain how the sampling distribution of the mean differences from the Crump or Randomization test relates to the concept that chance can produce differences (0.5 points)

b. Imagining a sampling distribution of the mean differences from a Crump test simulation, which values in the distribution are most likely to be produced by chance (0.5 points)

c. From above, which values are least likely to be produced by chance (0.5 points)

d. If you obtained a mean difference that was very large and it fell outside the range of the sampling distribution of the mean differences from a Crump test, what would you conclude about the possibility that chance produced your difference. Explain your decision. (0.5 points)

General grading.

- You will receive 0 points for missing answers
- You must write in complete sentences. Point form sentences will be given 0 points.
- Completely incorrect answers will receive 0 points.
- If your answer is generally correct but very difficult to understand and unclear you may receive half points for the question

5.1.6 Practice Problems

---

1. In this lab, you will be conducting a sign test to determine whether a coin is weighted. Individually or in small groups, take a coin out of your pocket. If working in groups, use only one coin for the whole group. Now, flip the coin 25 times. Write down how many heads and tails you observed.

2. Enter this into your SPSS spreadsheet and run a sign test using an alpha level of .05. What is your result?

3. Have students in the class (or groups) announce their results as well. Did anyone have a trick coin? Do you think some of the coins used were actually weighted? Why or why not?

Chapter 6

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

Any experiment may be regarded as forming an individual of a 'population' of experiments which might be performed under the same conditions. A series of experiments is a sample drawn from this population. —William Sealy Gossett

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 samples t-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/MehrSongS
```

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` data frame, 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 unfamiliar people, one of the left and one on the right. There was no sound during the baseline. 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` data frame 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