Answering questions with data: Lab Manual

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

2018-07-20

Contents

Preface

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

0.1 R

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

Statistics Software

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

0.2 Why R?

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

0.3 Installing R and R Studio

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

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

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

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

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

0.4 R studio notes and tips

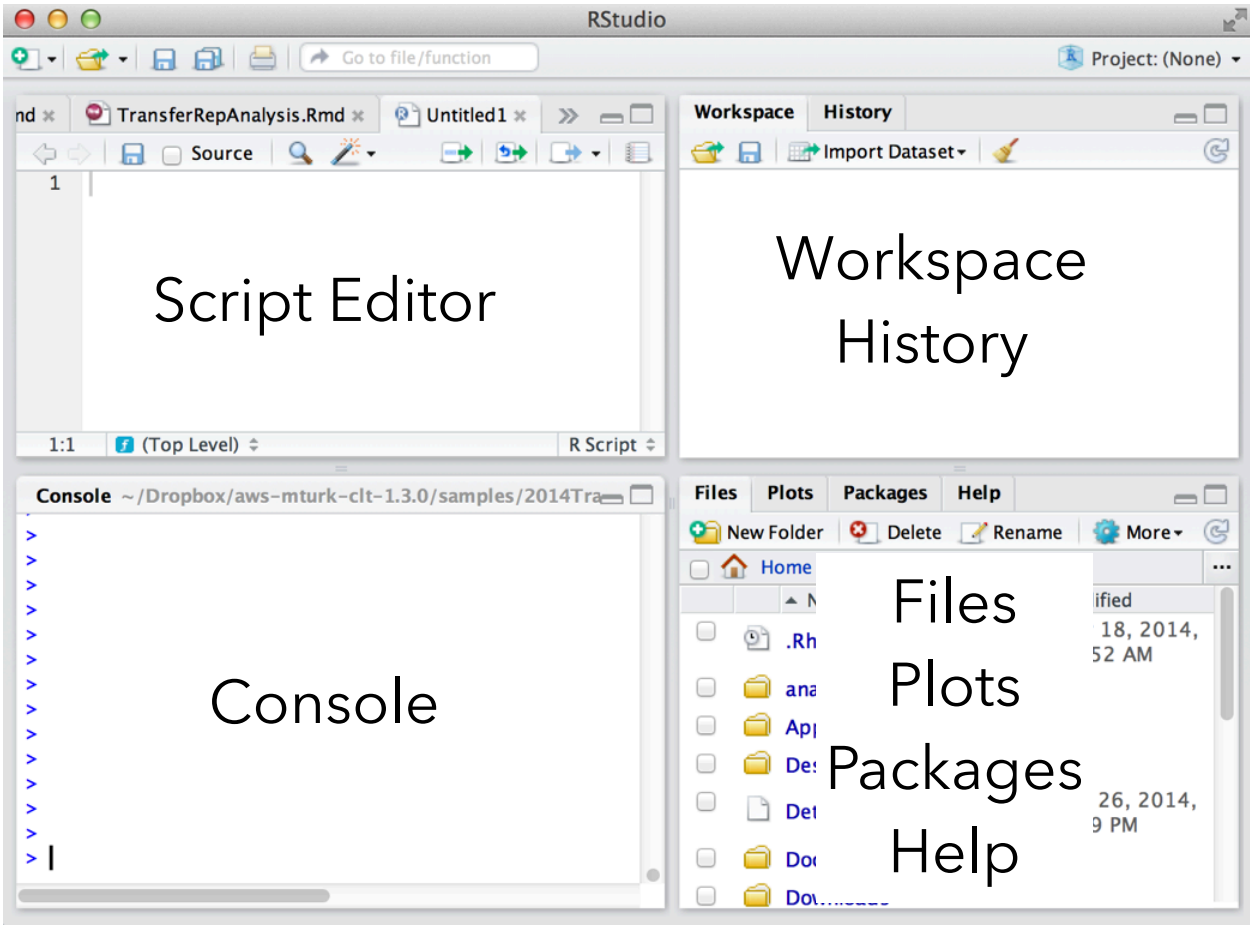


Figure 1: The R-studio workspace

0.4.1 Console

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

0.4.2 Script Editor

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

0.4.3 Workspace and History

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

0.4.4 File, Plot, Packages, Help

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

0.5 Final comments

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

Chapter 1

Lab 1: Graphing Data

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

1.1 General Goals

1.1.1 Other things if needed

1.2 R

1.2.1 General Goals

1. A brief tour of R-studio
2. Some R basics
3. Graphing data in R
4. Graphing data using ggplot2

1.2.1.1 R basics Checklist

1. Create a new R project
2. Execute commands in the console
3. Open and save new R script in the editor
4. Write a short script, and run the commands in the script in the console
5. View the contents of variables in the environment window
6. View the contents of variables using the console
7. Use the console like a calculator
8. Store numbers in variables

1.2.2 A clean start

Good organization is key to data analysis. To explain, let me tell you a story to help you avoid being like me when I started learning how to analyze data. As a graduate student, I would collect data from experiments I was running. I stored the data in different files in different folders on my computer (all over the place, hard to remember where sometimes). I would copy the data into excel so I could look at it, do basic analysis, and

reformat it so I could run statistics on the data using programs like SPSS. I would often have many different versions of excel spreadsheets with different versions of the data, sometimes stored in different folders. I would have many different SPSS outputs for each analysis I performed, sometimes in different locations. I would create tables and graphs in new excel spreadsheets, and edit the graphs in programs like Adobe Illustrator. All of these files would be all over the place. Sometimes, weeks, or months, or years later, I would revisit the data. After spending time to find it all, I would have to retrace my steps, doing detective work to figure out what analyses I had done. Ultimately, my previous work was so hard to understand, that I would end up redoing the analysis again. This was a messy process, and it took a lot of time. Wouldn't it be nice if everything was in one place? R provides this solution.

1.2.2.1 Making an R project

R projects are a convenient way to organize everything you do in R. To create an R project in R-studio you can go to the file menu, and choose "New Project…". Or, if you look in the top, right-side of the screen, you should see a little blue cube with an R in it. This shows your current R project. You can click this to create a new R project.

If you are using a lab computer, then insert a USB stick. Then click to create a new R project. Navigate to your USB stick drive. Give your project a name, like "StatsLab". This will create a new folder on your USB stick. Inside the folder will be a new R project file.

Once you have loaded an R project, all of the new files that you make will be saved in this project folder. You can also put data that you want to analyse in this folder. Additionally, the output of your analyses (including figures etc.) will be saved into this folder. This great, because everything is in one place, and you know where that is. When you want to return to work on your R project, you just have to load it up. You can make as many R projects as you like as a way to organize your work in R.

1.2.3 R console

R does things using scripts, which involves typing in commands to R. To begin, we will learn how to type in a command and execute it using the console.

The console is an interface to using R. You type in a command, then press enter to execute it. Then, R will show you the result in the console.

The console should be located in the bottom-left window of R-studio. You should see a tab that says "Console". If you do not see the console window, then click on the word Console, and the window should appear.

Inside the console you will a bunch of text telling you what version of R you are using. Scroll down to the bottom of the console and you should see a blue arrow (>) followed by a cursor. If you click into the console, then you will be able to type commands. For example, click into the console and type 1+1, then press enter.

```
1+1
```

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

Above you should see two grey boxes. The first grey box is example code, showing what I typed into the console (e.g., 1+1). The second grey box is output given by R after pressing enter. You can see it gave the answer 2.

1.2.3.1 Using the console as a calculator

R can be used just a like a calculator. Here are some examples:

```
7+100
```

```
## [1] 107
```

```
43-23
```

```
## [1] 20
```

```
34*4
```

```
## [1] 136
```

```
22/2
```

```
## [1] 11
```

```
1+(2*3)+5
```

```
## [1] 12
```

Try using the R console as a calculator for yourself.

Using the console is a quick and easy way to enter one command at a time. However, what if you want to enter more than one command? In this case, we want to write a script. A script is a recipe of multiple commands that tells R to do more than one thing, one after another.

1.2.4 R editor

We will use the R editor to write, save, and work on our scripts. The editor appears in the top-left window of R-Studio. When you open new scripts in R, you will see them appear as new tabs in the Editor window.

To open a new R script, look to the top left-hand side of R-studio. You should see a white square with a green plus sign. Click this button, and you can create a new R script.

The first thing that happens is a new, blank, R script is loaded, with the name "Untitled.R". If you save this file (file menu->save), then you will be asked to give your new script a name. Give it a new name. If you are working in an R Project, then R-studio will automatically save your new script in your R project folder. All ".R" files a just plain text files.

1.2.5 An example script

After you create a new script, you can click into the editor window, and write anything you want, just like a word processor. In general, the scripts we will write, will give R instructions one line at a time. Below is an example:

```
# this is a comment
a <- 1+1
b <- 2*3
c <- a+b
```

You can run this entire script in a few different ways:

1. Highlight all of the lines of text, copy them to the clipboard, then paste them into the console, and press enter (or return).
2. Highlight all of the lines of text, and press the "run" button at the top of the editor window (this automatically copies and pastes the selected lines, and runs them in the console).

After you run the script, you should see some output in the R console. Specifically, you should see each of lines of code that you asked R to run.

Notice, however, that we do not see any of the answers of our this script. What has happened?

Let's step through each line. The first line says "# this is a comment". Anything text that follows a "#" tells R not to run that line as code. Instead, R knows this is just a comment. Comments are very useful to insert into your scripts to explain, in plain english what is going on. For example, I will add more comments to the above script to explain what is going on.

```
# this is a comment
a <- 1+1 # puts 1+1 into new variable a
b <- 2*3 # puts 2 times 3 into new variable b
c <- a+b # puts the sum of variable a and b into c
```

The comments give more insight into what R is doing here. For example, each line does a simple calculation, and stores the result into a new variable. Variables are a way to store our data in R. You can think of them as containers with names.

Let's look more closely at this line:

```
a <- 1+1
```

1.2.5.1 Variable name

The 'a' is the name of the variable. In general, you can choose any name that you want. It is best to give descriptive names that are meaningful, and that help you remember what the variable is being used for.

1.2.5.2 <-

The '<-' command tells R to put something into the variable. Anything that is to the right of the '<-' command will be put into the variable named on the left-hand side of the '<-' command

1.2.5.3 1+1

1+1 is an operation that we are asking R to compute. The output of this operation is put into '<-' the variable named 'a'.

1.2.5.4 Where are the variables?

Where are these variables, and how can we see them? There are two ways to see what is inside variables.

1. In the top right window, you should see a tab called "Environment". This tab lists all of the variables that you have currently stored in R. You should see an a, b, and c, along with the numbers inside them.
2. You can type the name of the variable into the R console, and then press enter. The console will display the contents of the variable.

1.2.6 A bunch of numbers

Data comes in all shapes and sizes. Usually, there are so many numbers that it is difficult to make sense of them. Let me show you 100 numbers.

Where did these numbers come from? What kind of properties do these numbers have? Are there any patterns in the numbers? Do some kinds of numbers happen more often than other kinds of numbers? What can we say about these numbers just by looking at them?

If you take some time to look at the above numbers, you might start noticing some regularities. When I quickly look at them I see:

1. Most the numbers are around 100.
2. The numbers all appear to be different by big or small amounts
3. There are no negative numbers
4. There are no really small numbers (e.g., close to 0)
5. There are no really huge numbers

At least we can get some sense of the numbers by eyeballing them. If there were 1000s or 100000s of numbers, eyeballing them one at a time would take forever.

1.2.7 Making numbers in R

Before we go ahead and use R to make plots and graphs of data, we need to first have some data to plot. And, before we start using real data, it is worth pointing out that we can use R to create numbers. So, after we create our own sets of numbers, we can then plot them to see how graphing works.

1.2.7.1 rep function

Let's say you wanted to create a variable that stored the number 43, 100 times. You can do this using the rep function (rep is short for repeat). Below is an example of how this function is used.

```
my_numbers <- rep(43,100)
```

We can check to see what is inside the new variable *my_numbers* by typing it into the console, or looking at it in the environment tab. You should see that it contains the number 43, repeated 100 times. Using the rep function you can repeat anything, any number of times

1.2.7.2 seq function

Let's say you want to create a sequence of numbers. You can do this using the seq function. The example below starts at 23, and goes to 56, in increments of 1. You can modify the starting value, the ending value, and the increment value to create many different kinds of sequences.

```
my_sequence <- seq(23,56,1)
```

1.2.7.3 runif function

R can generate numbers in much more sophisticated ways. In particular, we can use R to sample numbers from distributions with particular properties. We will introduce distributions in the next lab.

R can generate random numbers using the runif function. In the example below, R generates 100 numbers that are randomly between 0 and 1. You can generate as many numbers as you want, between any two numbers that you want.

```
my_randoms <- runif(100,0,1)
```

1.2.7.4 rnorm function

R can generate numbers from a normal distibrution. In the example belwo, we generate 100 numbers from a distribution with a mean of 10, and a standard deviation of 20.

```
my_normal <- rnorm(100,10,20)
```

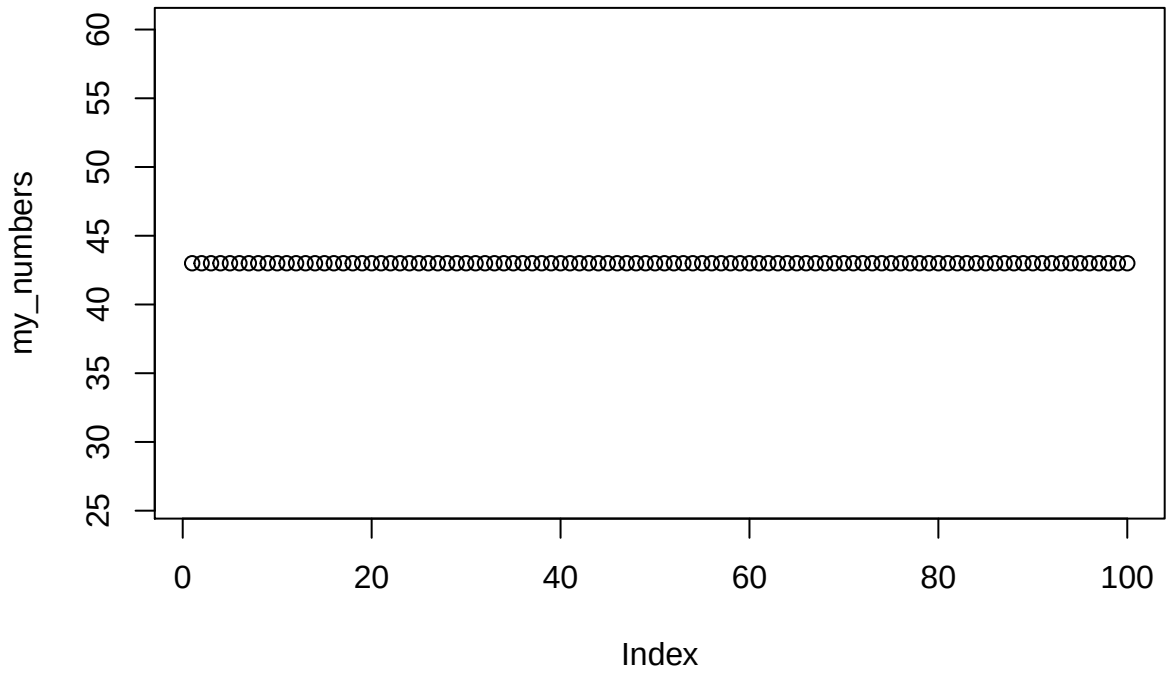
1.2.8 Graphing Data in R

Graphs, or visual displays, of numbers can be very useful for interpreting data. Fortunately, we can use R to create many kinds of visual displays, that can help us interpret the data. To begin we will look at the plot and histogram functions.

1.2.8.1 Plot function

Previously, we created a *my_numbers* variable, that contains the number 43, repeated 100 times. If we plot this in R, what should we see? Let's do it and find out.

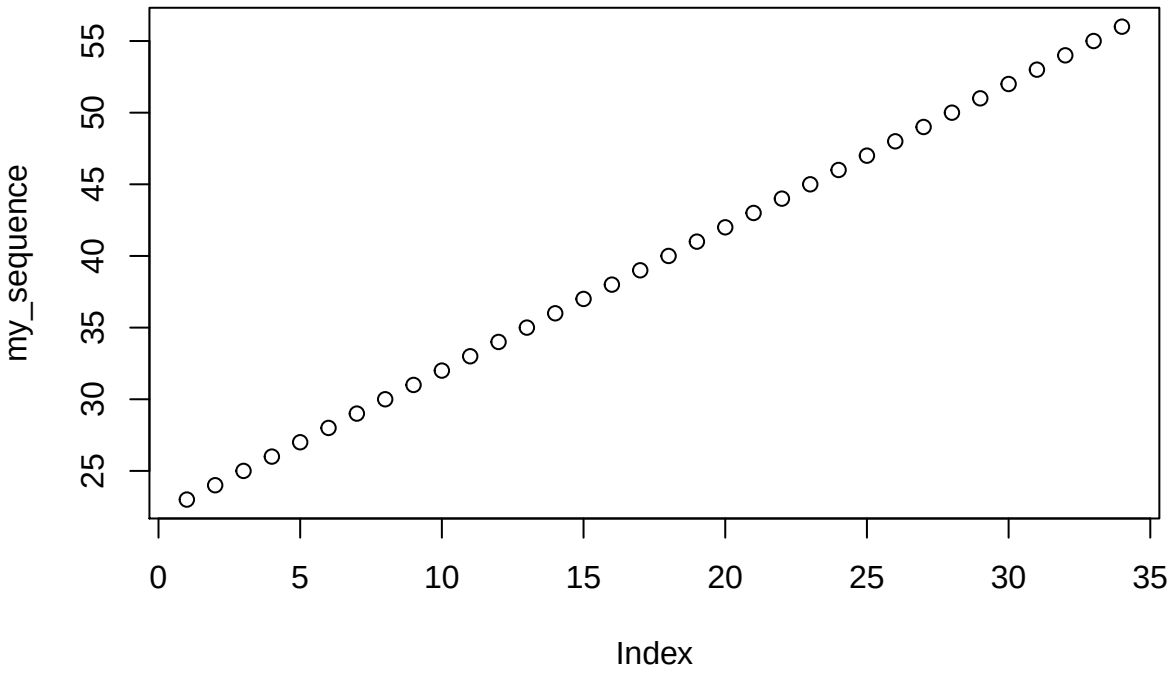
```
plot(my_numbers)
```



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

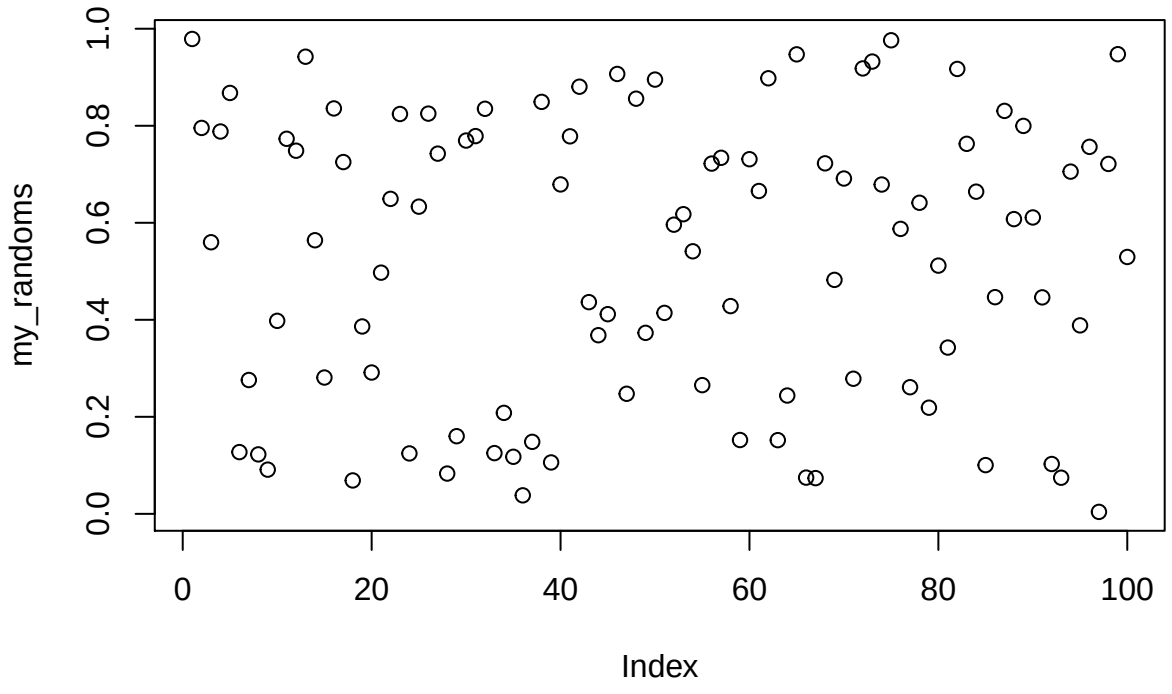
Let's plot some of the other variables we made.

```
plot(my_sequence)
```



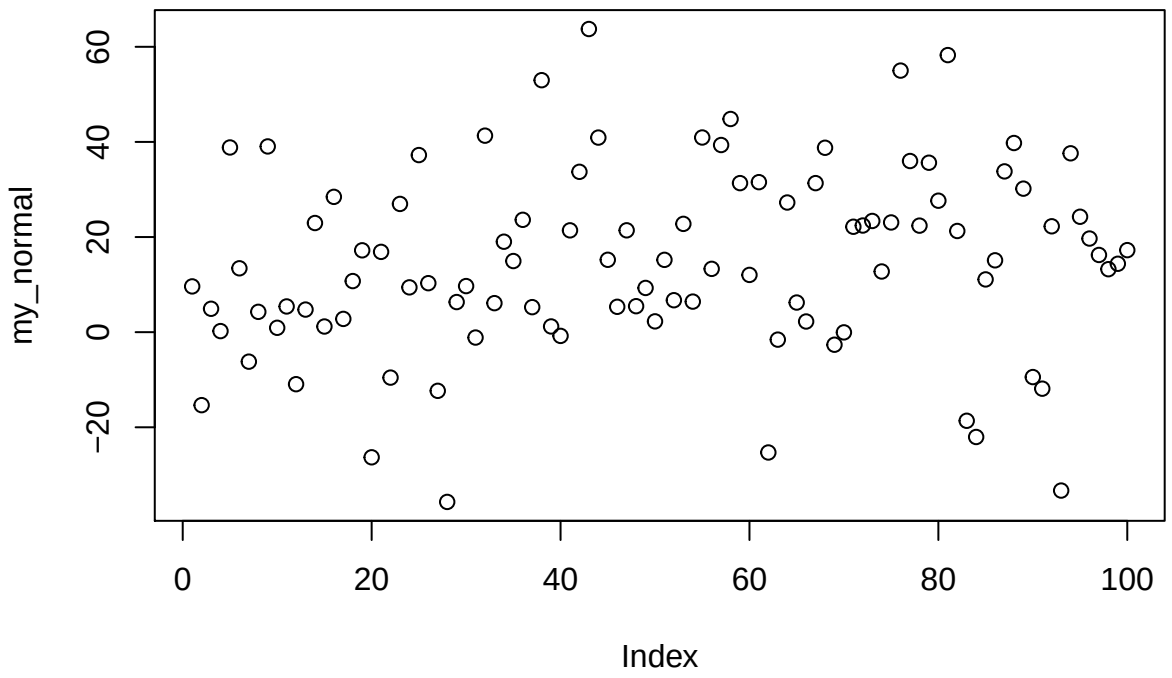
The variable *my_sequence* contains the numbers 23 to 56, going up by one. We see in the plot, the first number (on the x-axis) is a 23 on the y-axis. As we go across the x-axis, the numbers go up by one until we get to 56. We see a straight diagonal line.

```
plot(my_randoms)
```



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

```
plot(my_normal)
```



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

1.2.8.2 Histograms

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

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

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

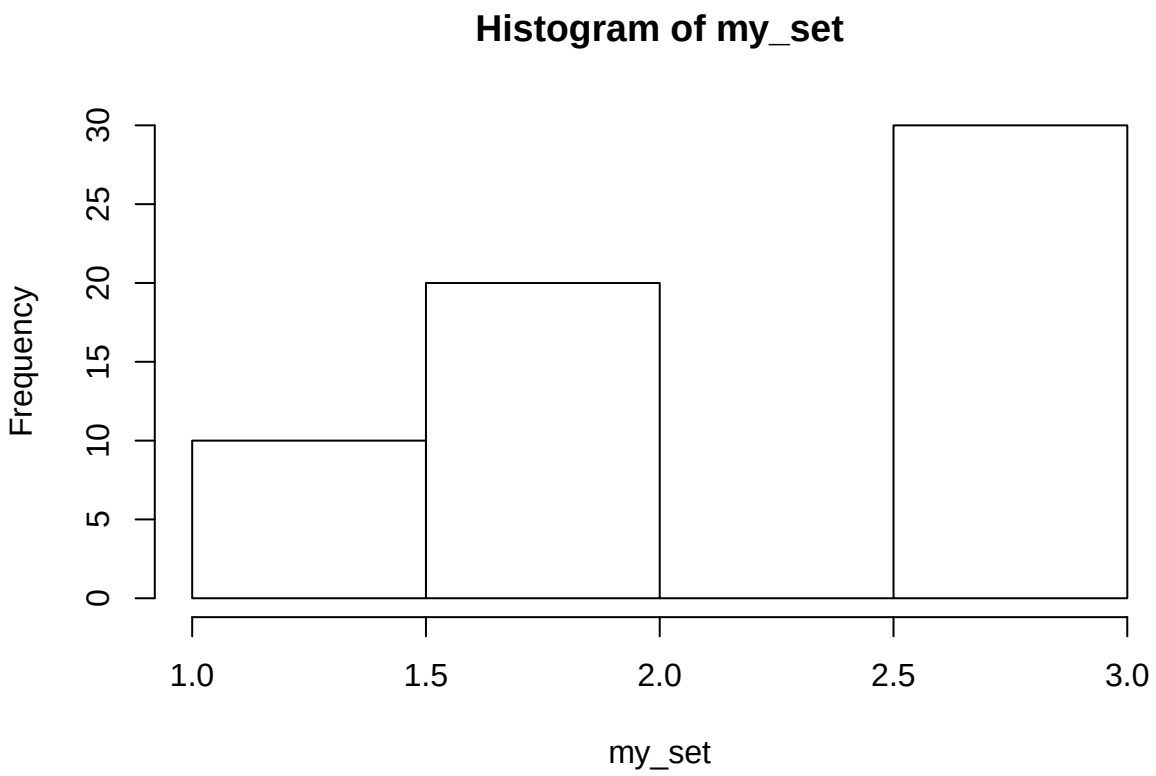
The above line of code uses two R functions, rep(), and c(). We already know how rep works. The c() function is short for combine. So, the above line of code, combines 10 1s, 20 2s and 30 3s, all into one variable. The contents of the variable looks like this:

```
my_set
```

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

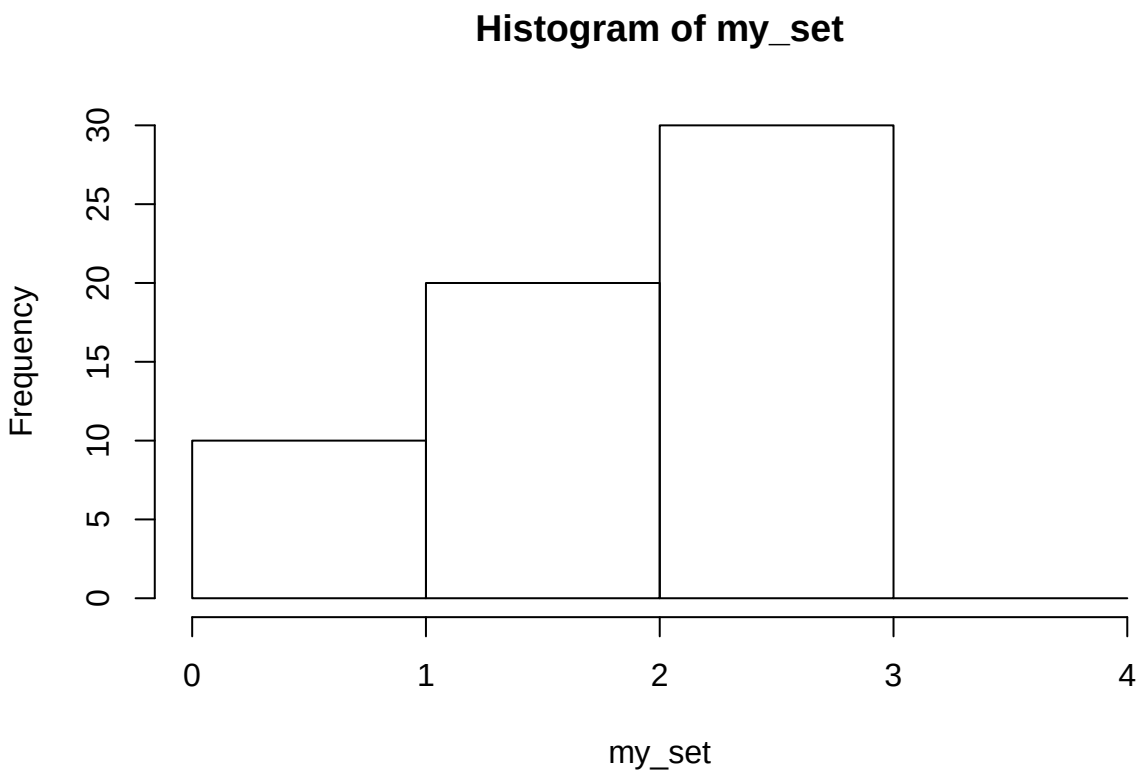
Because, we made this variable, we already know what is inside it. Let's make a histogram of the variable, to see what that looks like:

```
hist(my_set)
```



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

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



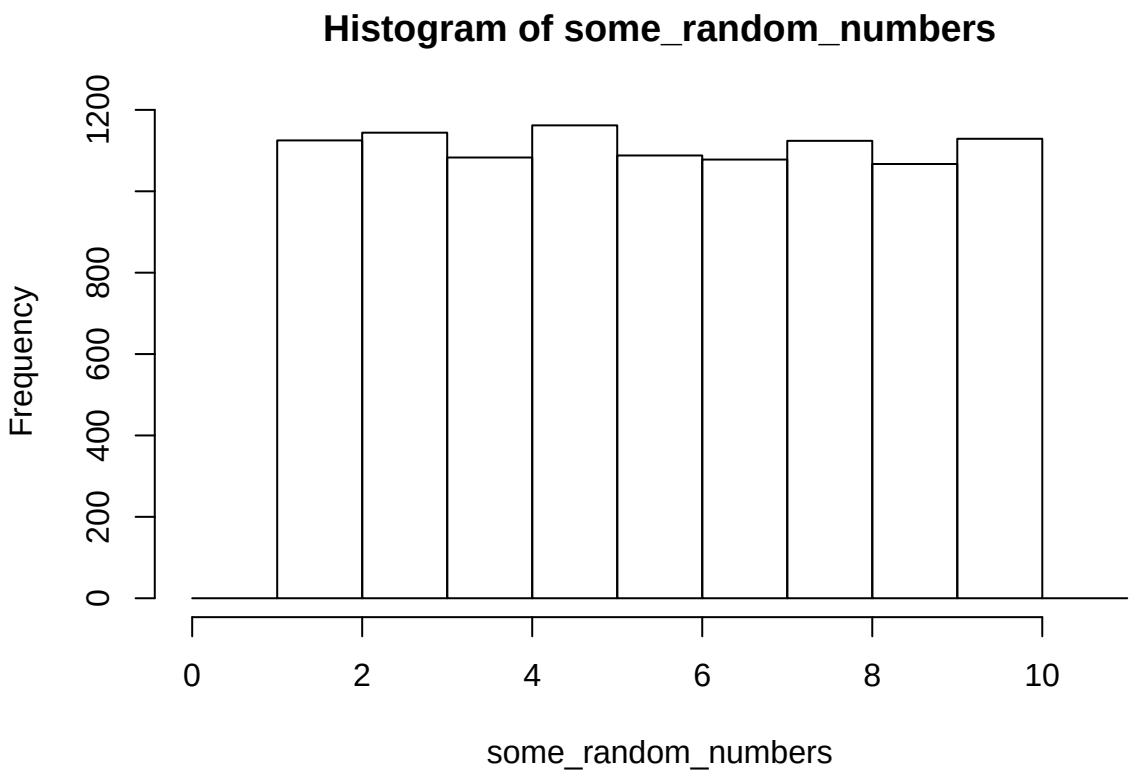
Let's spend a moment interpreting this new histogram. The first bar is between 0 and 1 on the x-axis, and has a value of 10 on the y-axis. This means that there are 10 numbers inside the my_set variable that have a value between 0 and 1; specifically, a value greater than zero up to and equalling 1.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

1.2.8.3 What are histograms useful for?

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

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

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



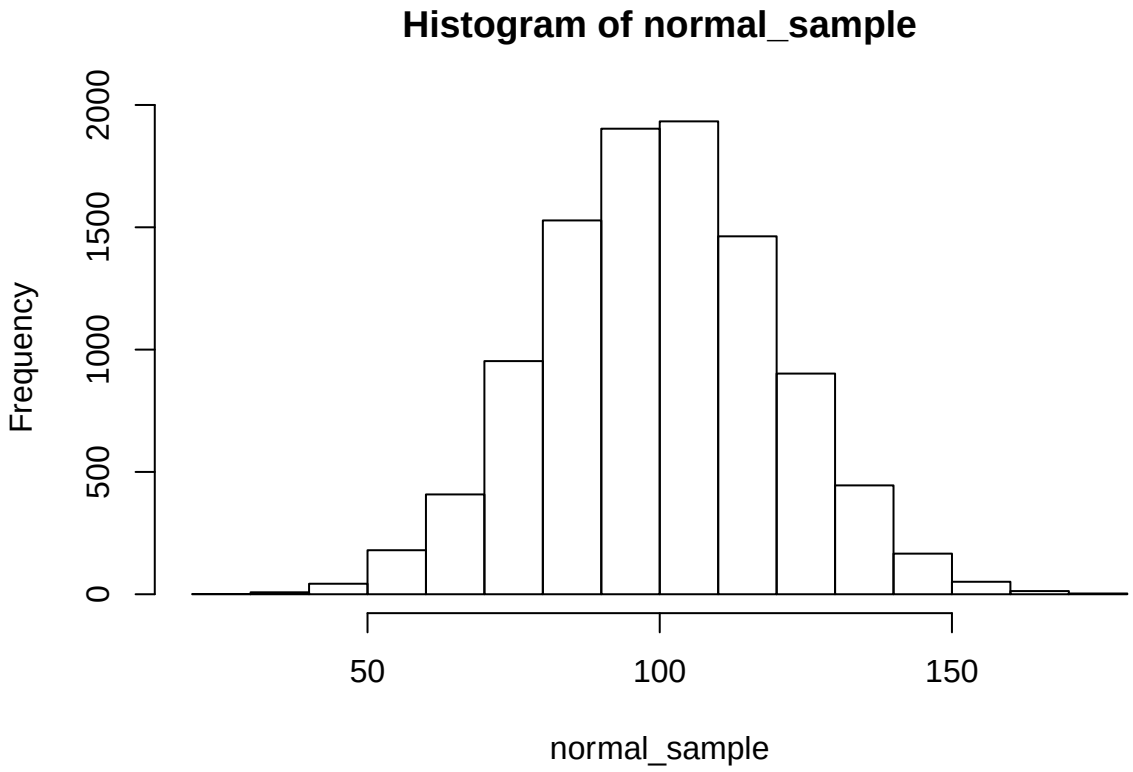

```
#notice I set the breaks using the seq() function
```

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

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

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

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



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

1.3 Excel

1.4 SPSS

1.5 Matlab

Chapter 2

Lab 2: Descriptive Statistics

Some inspiring quote —Inspiring Person

2.1 Outline of Problem to solve

Stuff we need to say in general

2.1.1 important things

Other things to say

2.2 R

How to do it in R

2.3 Excel

How to do it in Excel

2.4 SPSS

How to do it in SPSS

2.5 Matlab

How to do it in Matlab

Chapter 3

Lab 3: Correlation

Some inspiring quote —Inspiring Person

3.1 Outline of Problem to solve

Stuff we need to say in general

3.1.1 important things

Other things to say

3.2 R

How to do it in R

3.3 Excel

How to do it in Excel

3.4 SPSS

How to do it in SPSS

3.5 Matlab

How to do it in Matlab

Chapter 4

Lab 4: Normal Distribution & Central Limit Theorem

Some inspiring quote —Inspiring Person

4.1 Outline of Problem to solve

Stuff we need to say in general

4.1.1 important things

Other things to say

4.2 R

How to do it in R

4.3 Excel

How to do it in Excel

4.4 SPSS

How to do it in SPSS

4.5 Matlab

How to do it in Matlab

Chapter 5

Lab 5: Fundamentals of Hypothesis Testing

Some inspiring quote —Inspiring Person

5.1 Outline of Problem to solve

Stuff we need to say in general

5.1.1 important things

Other things to say

5.2 R

How to do it in R

5.3 Excel

How to do it in Excel

5.4 SPSS

How to do it in SPSS

5.5 Matlab

How to do it in Matlab

Chapter 6

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

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

6.1 Does Music Convey Social Information to Infants?

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

6.1.1 STUDY DESCRIPTION

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

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

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

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

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

6.2 Lab skills learned

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

6.3 Important Stuff

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

6.4 R

6.4.1 Loading the data

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

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

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

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

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

6.4.2 Inspect the data frame

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

6.4.2.1 summarytools

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

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

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

6.4.3 Get the data for Experiment one

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

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

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

6.4.4 Baseline phase: Conduct a one sample t-test

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

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

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

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

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

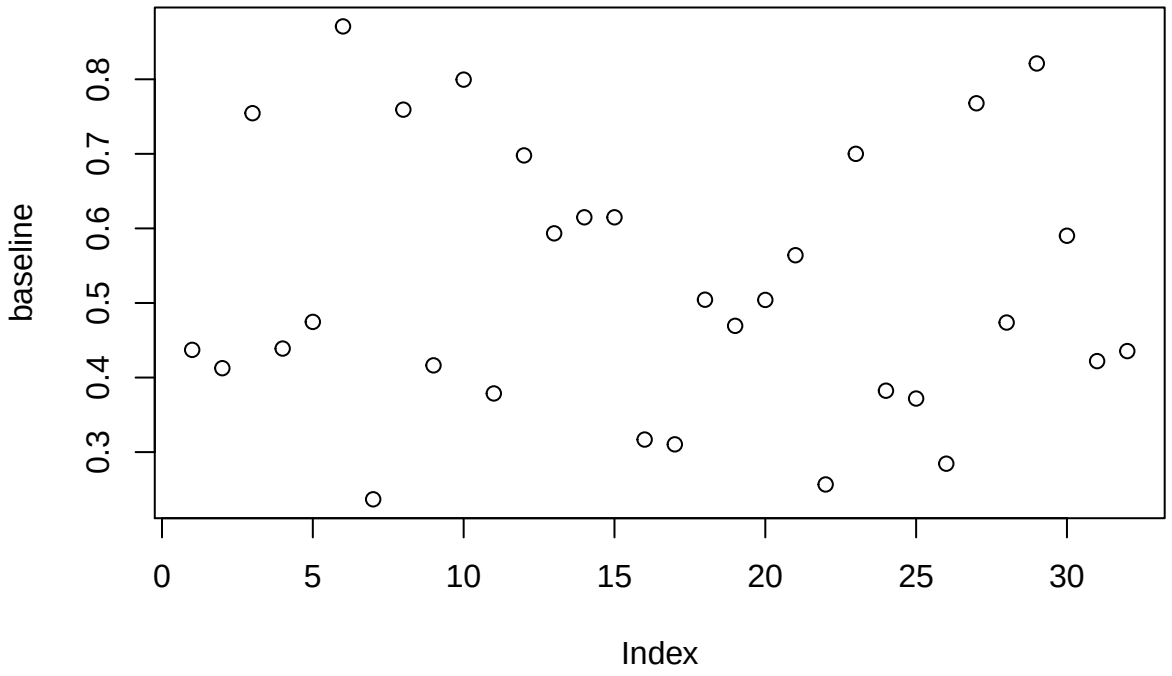
```
baseline <- experiment_one$Baseline_Proportion_Gaze_to_Singer
```

6.4.4.1 Look at the numbers

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

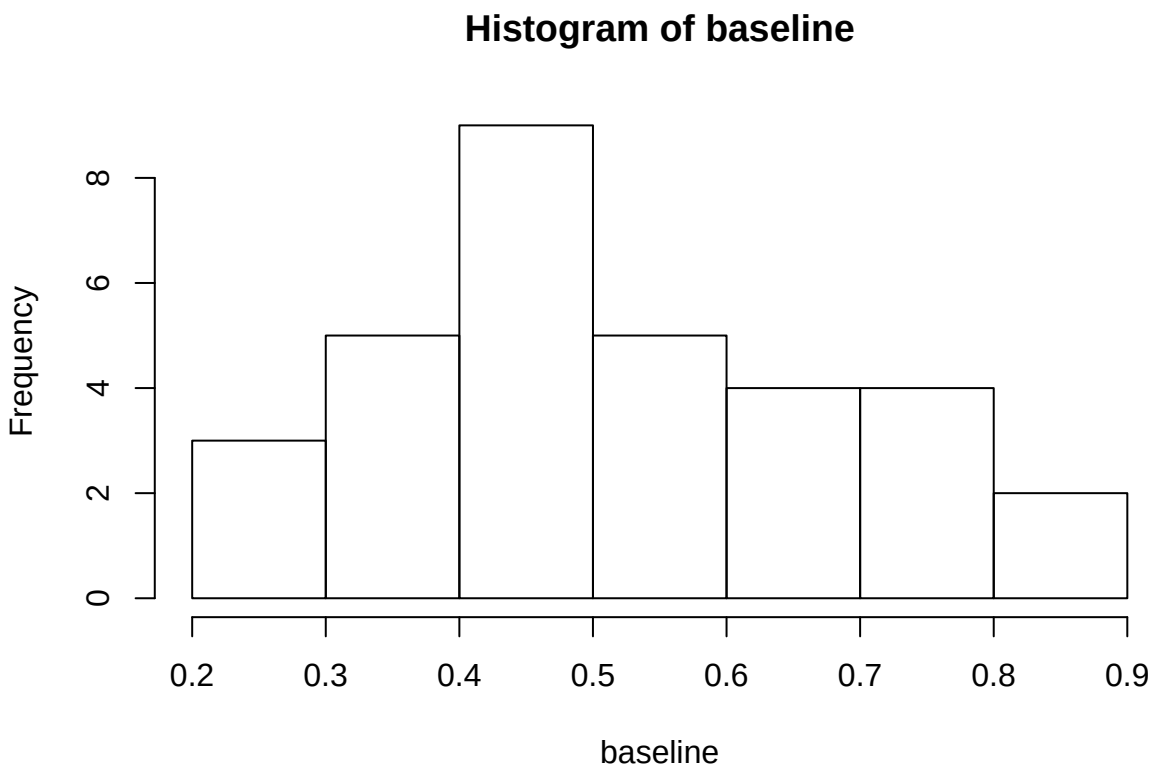
Ok, we could just do the t-test right away, it's really easy, only one line of code. But, we haven't even looked at the numbers yet. Let's at least do that. First, we'll just use plot. It will show every data point for each infant as a dot.

```
plot(baseline)
```



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

```
hist(baseline)
```



6.4.4.2 Look at the descriptives

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

```
mean(baseline)
```

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

```
sd(baseline)
```

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

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

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

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

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

6.4.4.3 Conduct t.test

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

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

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

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

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

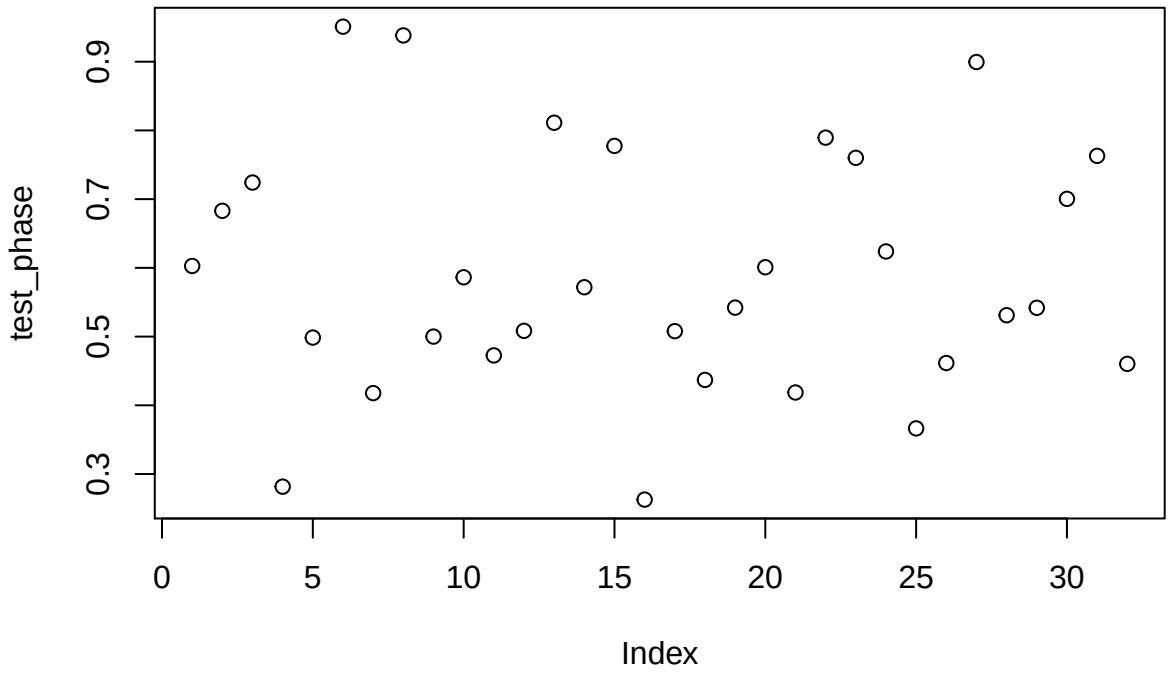
You should take the time to check this result, and see if it is the same one that was reported in the paper.

6.4.5 Test phase

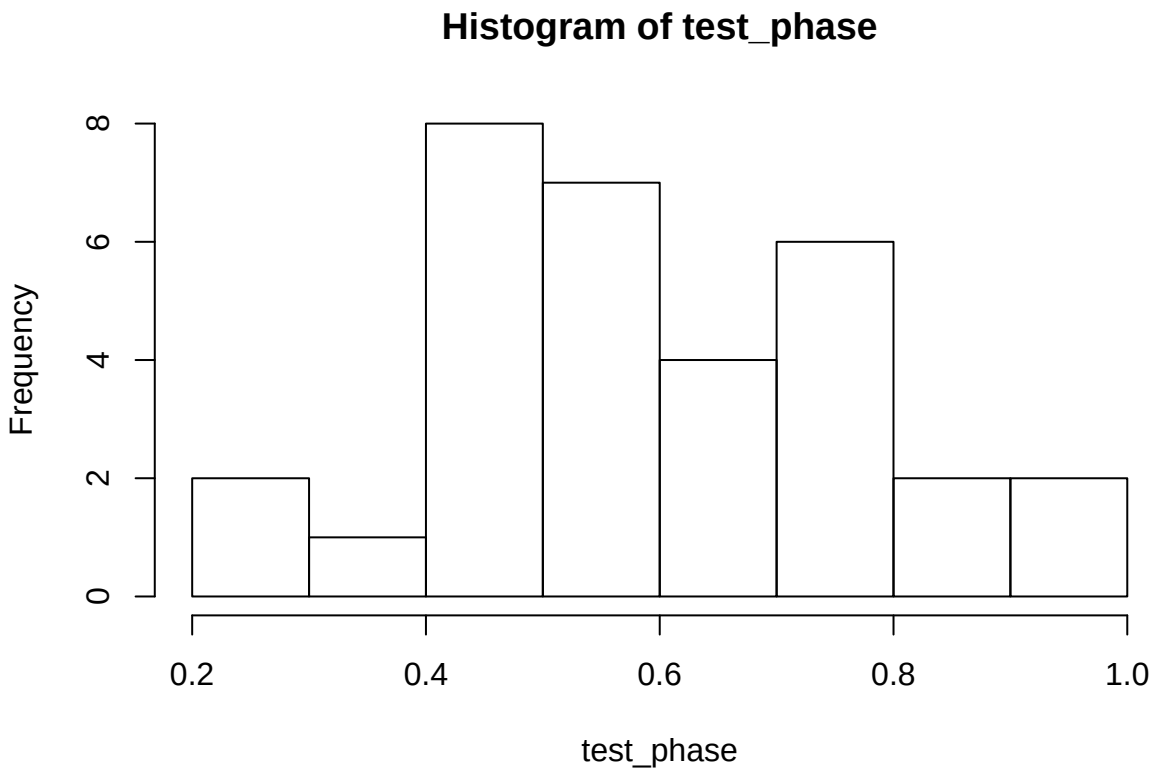
Remember how the experiment went. Infants watched silent video recordings of two women (Baseline). Then each person sung a song, one was familiar to the infant (their parents sung the song to them many times), and one was unfamiliar (singing phase). After the singing phase, the infants watched the silent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

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

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




```
hist(test_phase)
```




```
mean(test_phase)
```

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

```
sd(test_phase)
```

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

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

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

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

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

6.4.6 Paired-samples t-test

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

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

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

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

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

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

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

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

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

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

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

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

6.4.6.2 Usefulness of difference scores

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

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

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

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

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

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

6.4.7 Graphing the findings

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

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

6.4.7.1 Make the dataframe for plotting

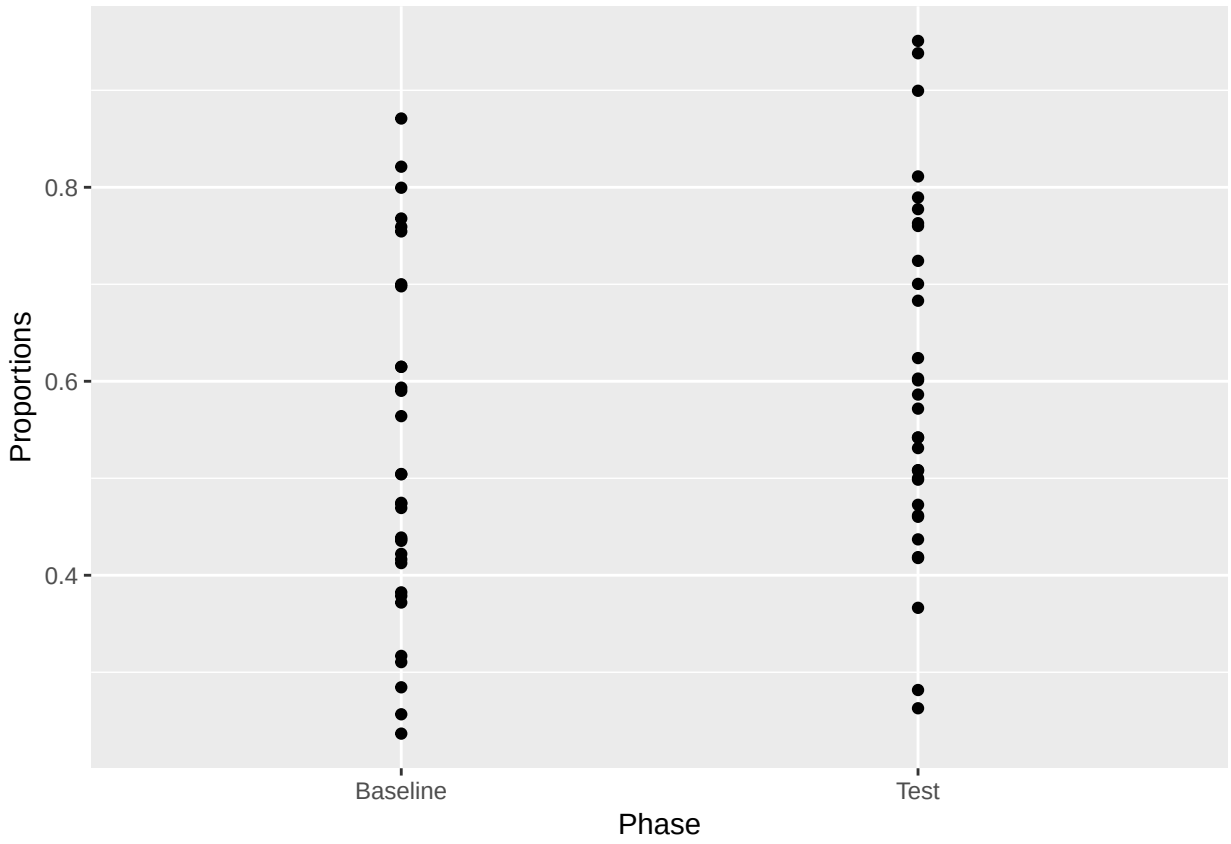
To start we will need 2 columns. One column will code the experimental phase, Baseline or Test. There are 32 observations in each phase, so we want the word `Baseline` to appear 32 times, followed by the word `Test` 32 times. Then we want a single column with each of the proportions for each infant.

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

6.4.7.2 Dot plot of raw scores

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

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



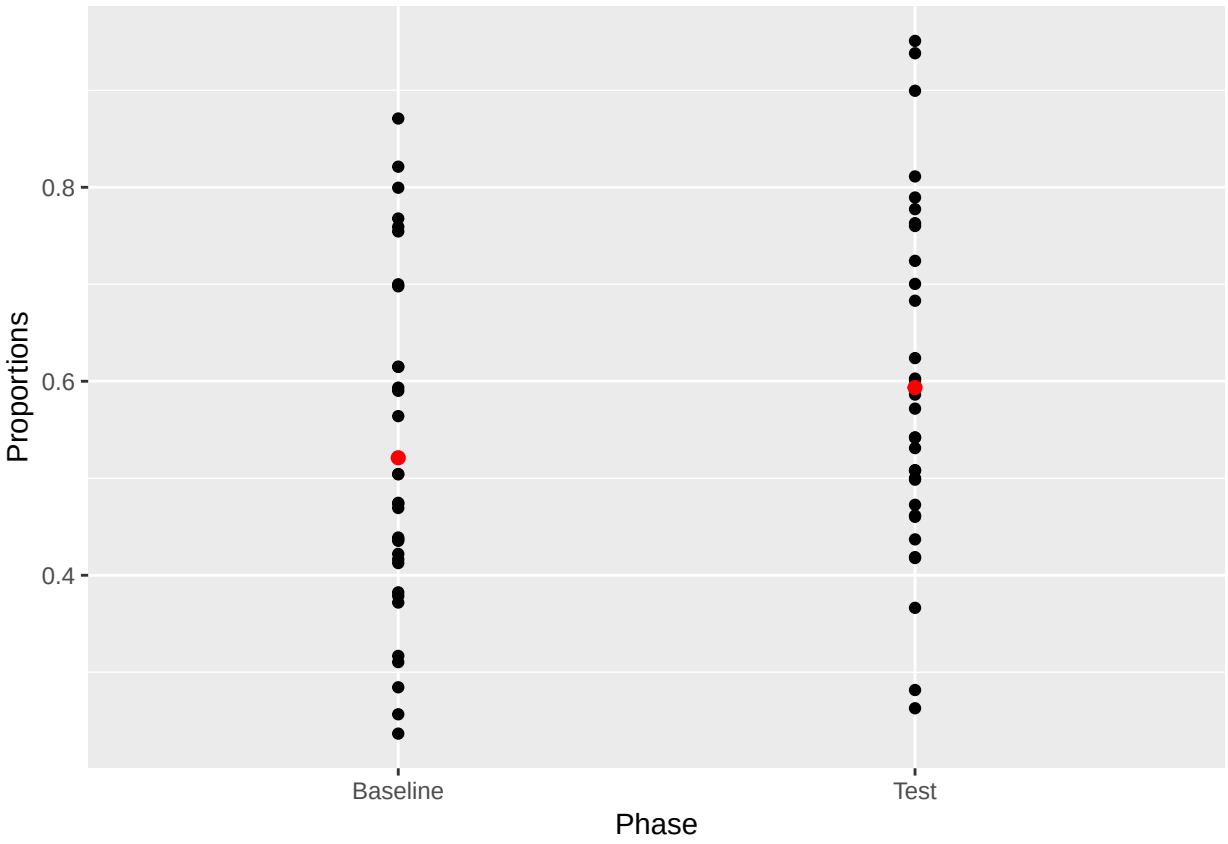
6.4.7.3 Dot plot with means and raw scores

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

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

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

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

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



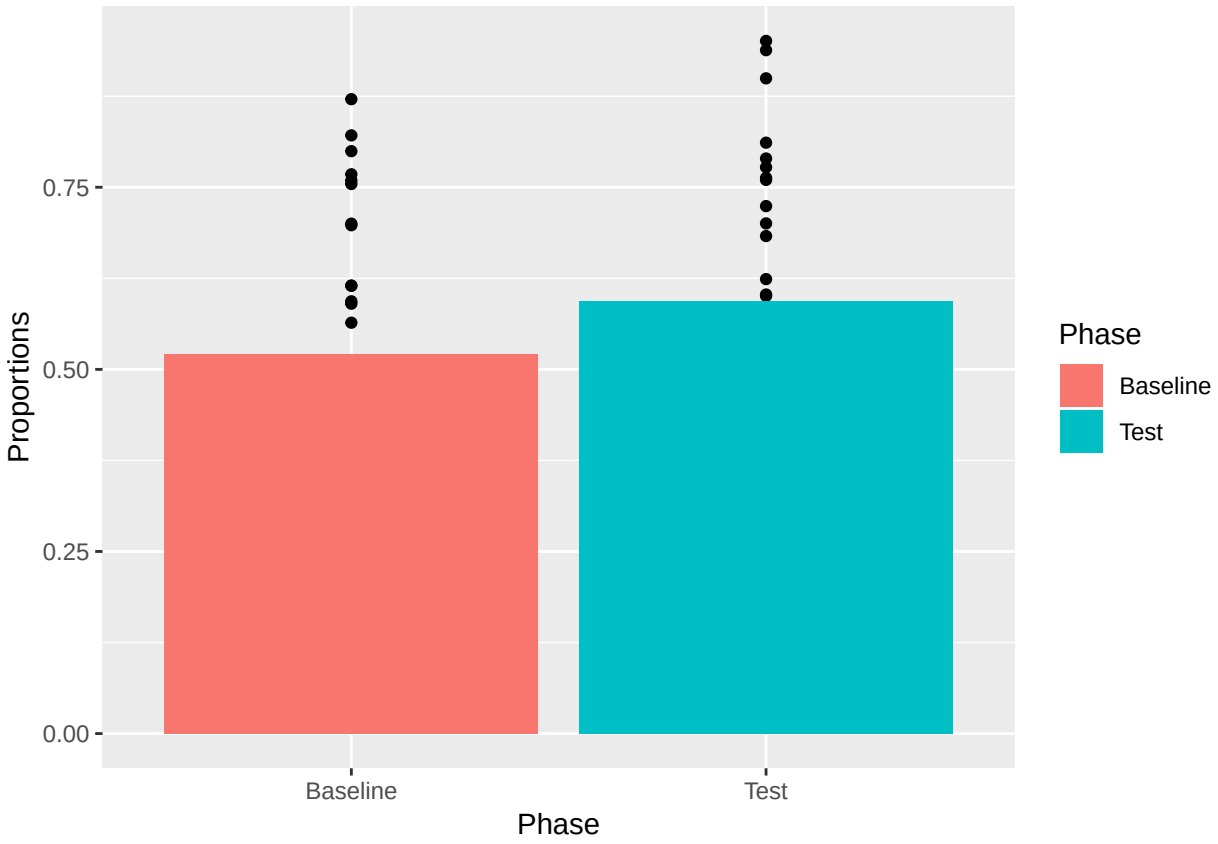
6.4.7.4 Bar plot

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

Also look for these changes.

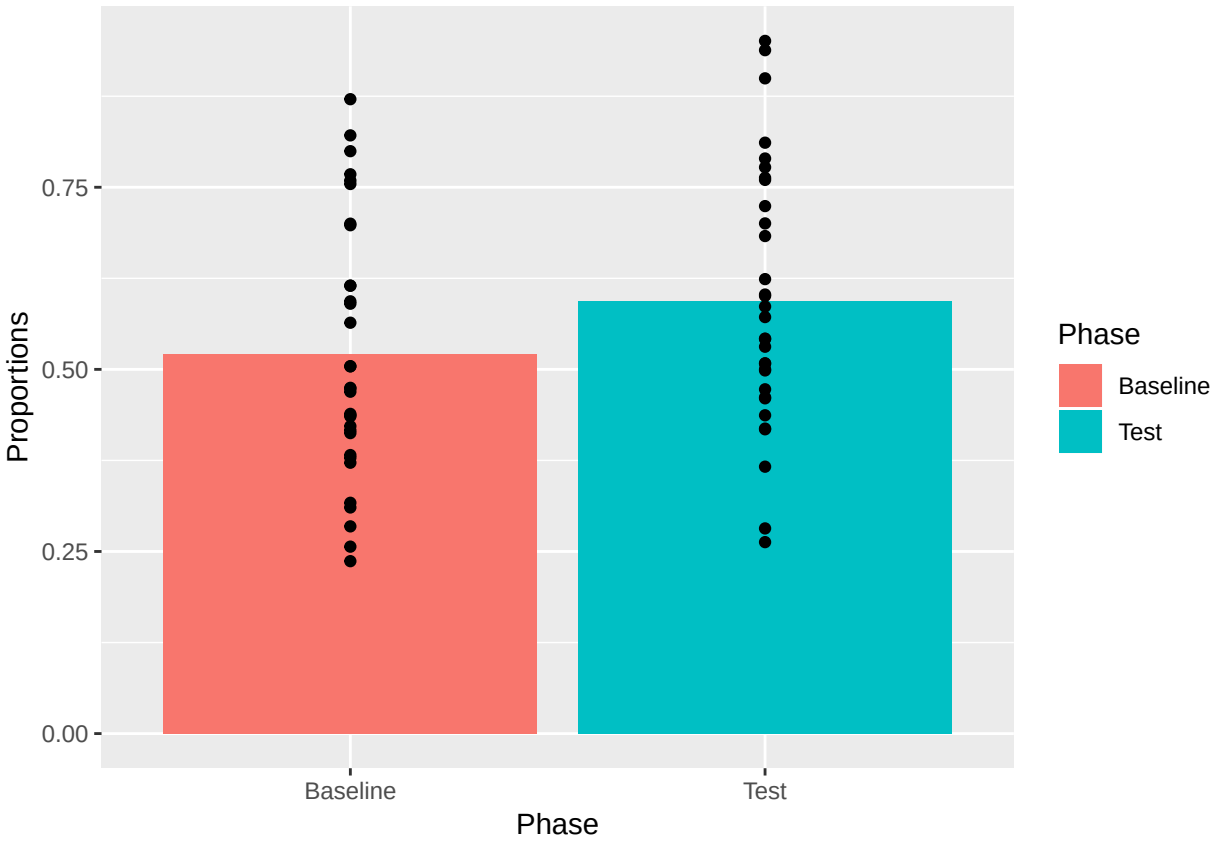
1. added `stat="identity"` Necessary for bar plot to show specific numbers
2. added `aes(fill=Phase)` Makes each bar a different color, depending on which phase it comes from

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



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

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



6.4.7.5 Bar plot with error bars

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

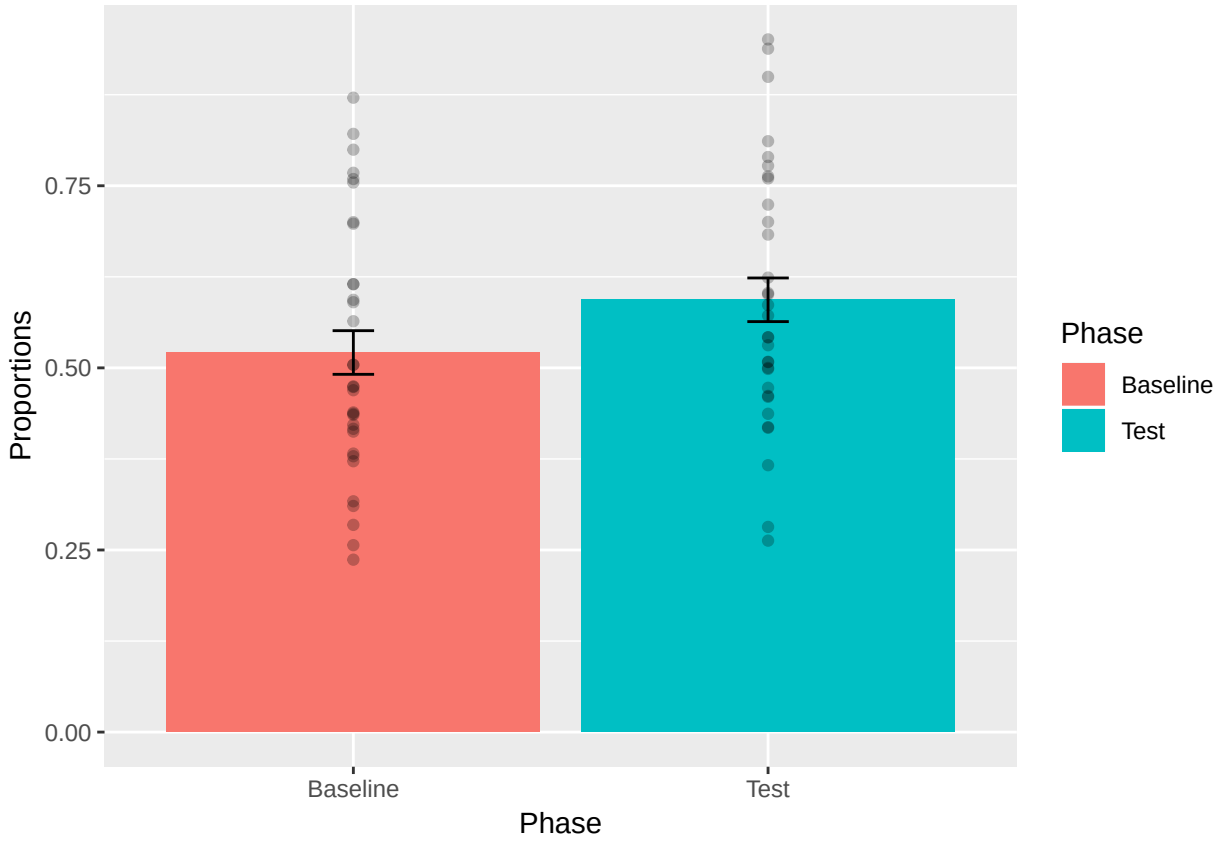
Question: Why would the standard deviation of each mean, or the standard error of each mean be inappropriate to use in this case?

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

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

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


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

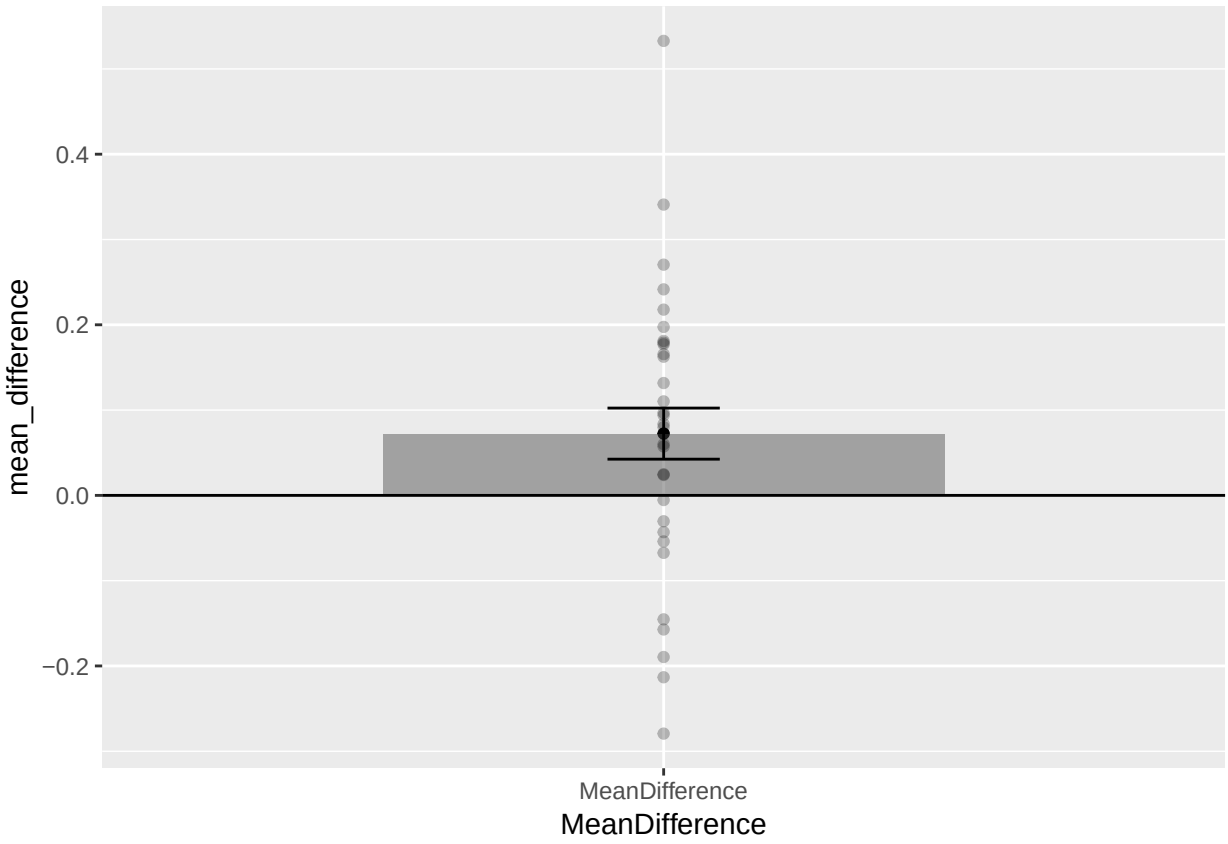
```
  geom_bar(data=mean_df, stat="identity",aes(fill=Phase))+
  geom_errorbar(data=mean_df, aes(ymin=Proportions-standard_error,
                                  ymax=Proportions+standard_error), width=.1) +
  geom_point(alpha=.25)
```



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

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

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

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



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

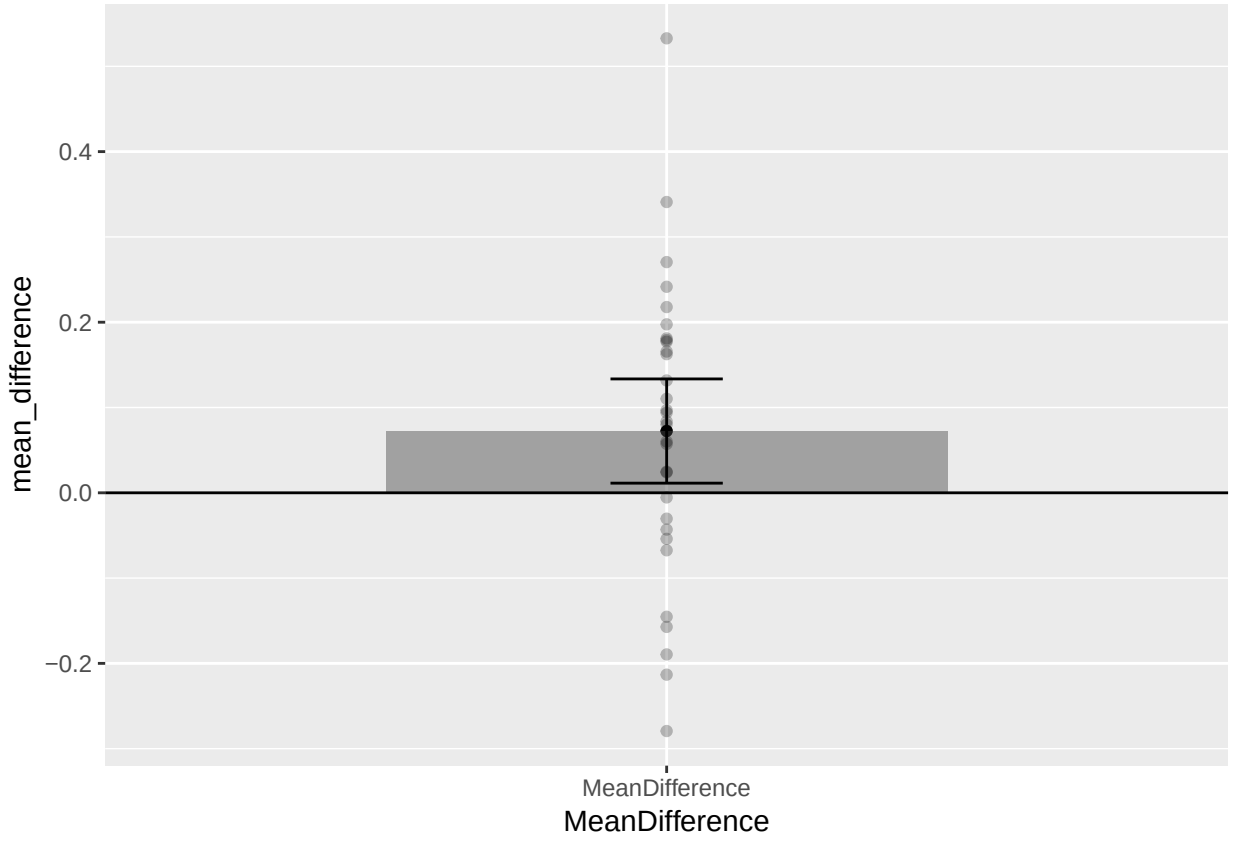
This plot is very useful too, it gives us some new information. We can see that the mean difference (test - baseline) was greater than 0. And, we are plotting the standard error of the mean differences, which are the error bars that more formally belong to this graph. Still, we are in a position to make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

6.4.7.6 Bar plot with confidence intervals

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

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

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

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



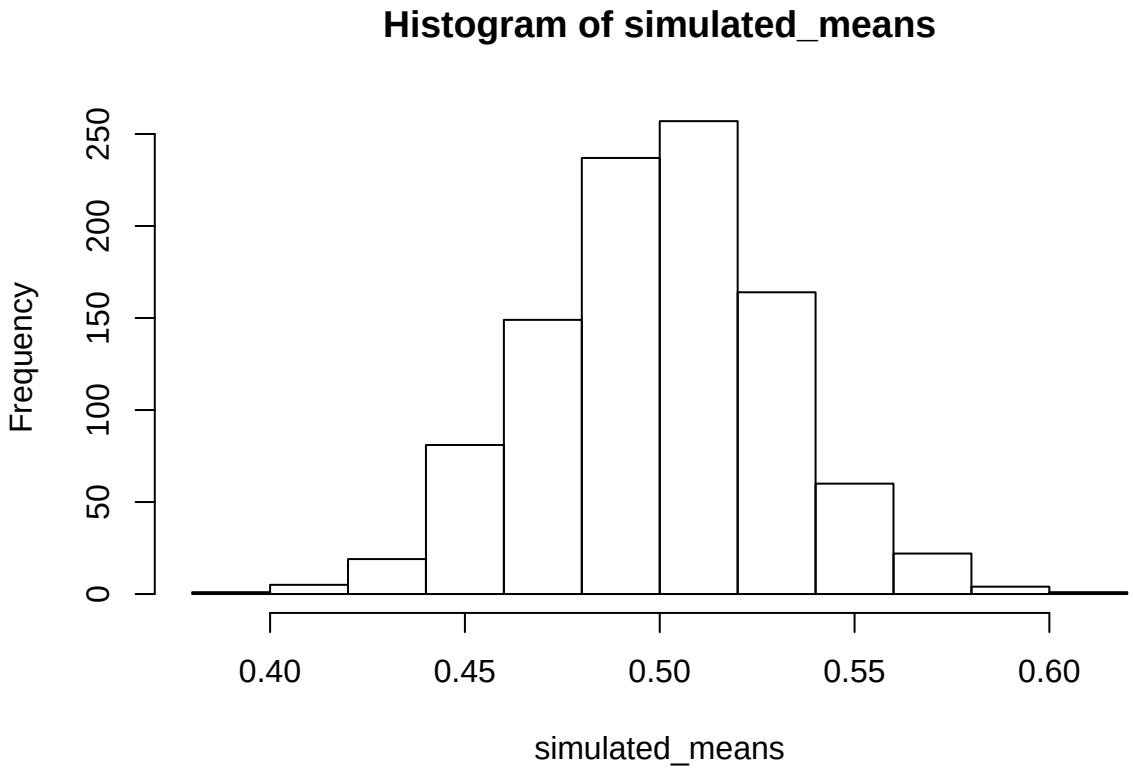
Notice that the 95% confidence intervals around the mean are wider than the SEM error bars from the previous graph. These new confidence intervals tell us that 95% of the time our sample mean will fall between the two lines. The bottom line is slightly above 0, so we can now visually see support for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

6.4.8 Data-simulation

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

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

simulated_means <- length(1000)
```

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

hist(simulated_means)
```

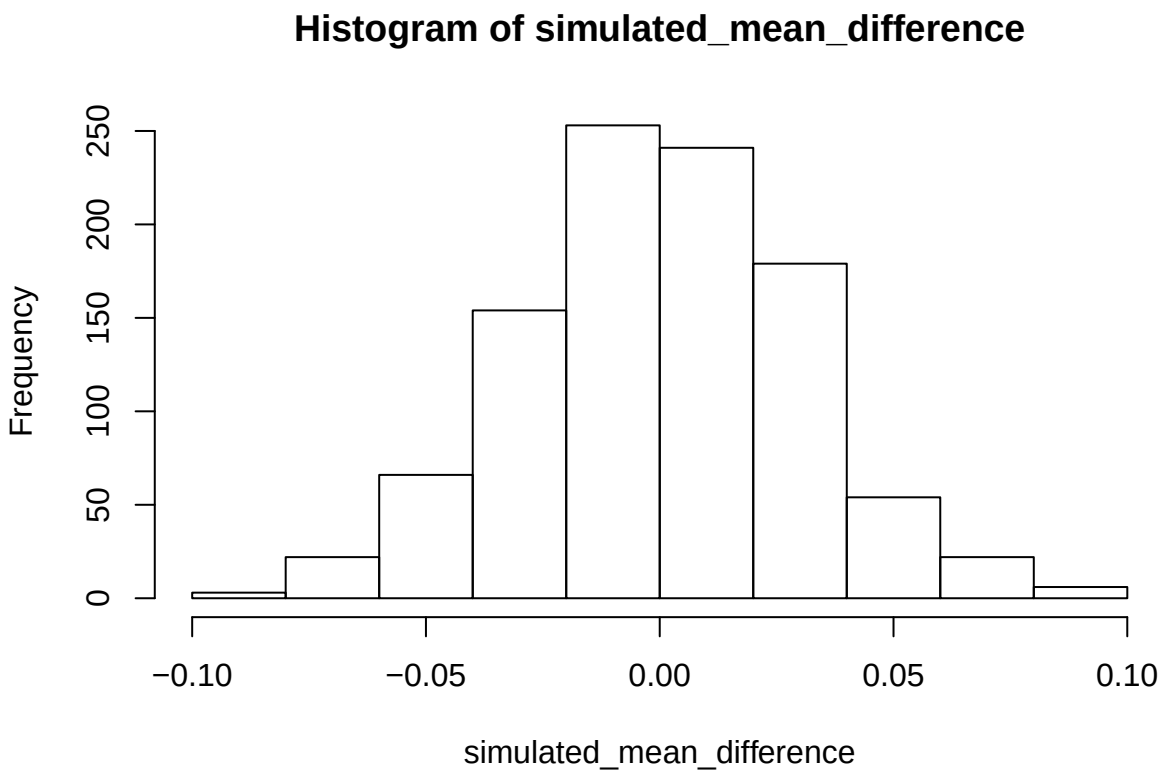


####

Simulating the mean differences

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

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

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

hist(simulated_mean_difference)
```



6.5 Excel

How to do it in Excel

6.6 SPSS

How to do it in SPSS

6.7 Matlab

How to do it in Matlab

Chapter 7

Lab 7: t-test (Independent Sample)

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

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

7.1.1 STUDY DESCRIPTION

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

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

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

7.2 Lab skills learned

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

7.3 Important Stuff

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

7.4 R

7.4.1 Load the data

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

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

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

7.4.2 Inspect data frame

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

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

7.4.3 Find the data you need

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

7.4.3.1 Condition variable

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

```
all_data$CONDITION
```

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

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

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

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

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

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

7.4.3.2 Dependent Measures

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

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

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

7.4.4 Look at the dependent variable.

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

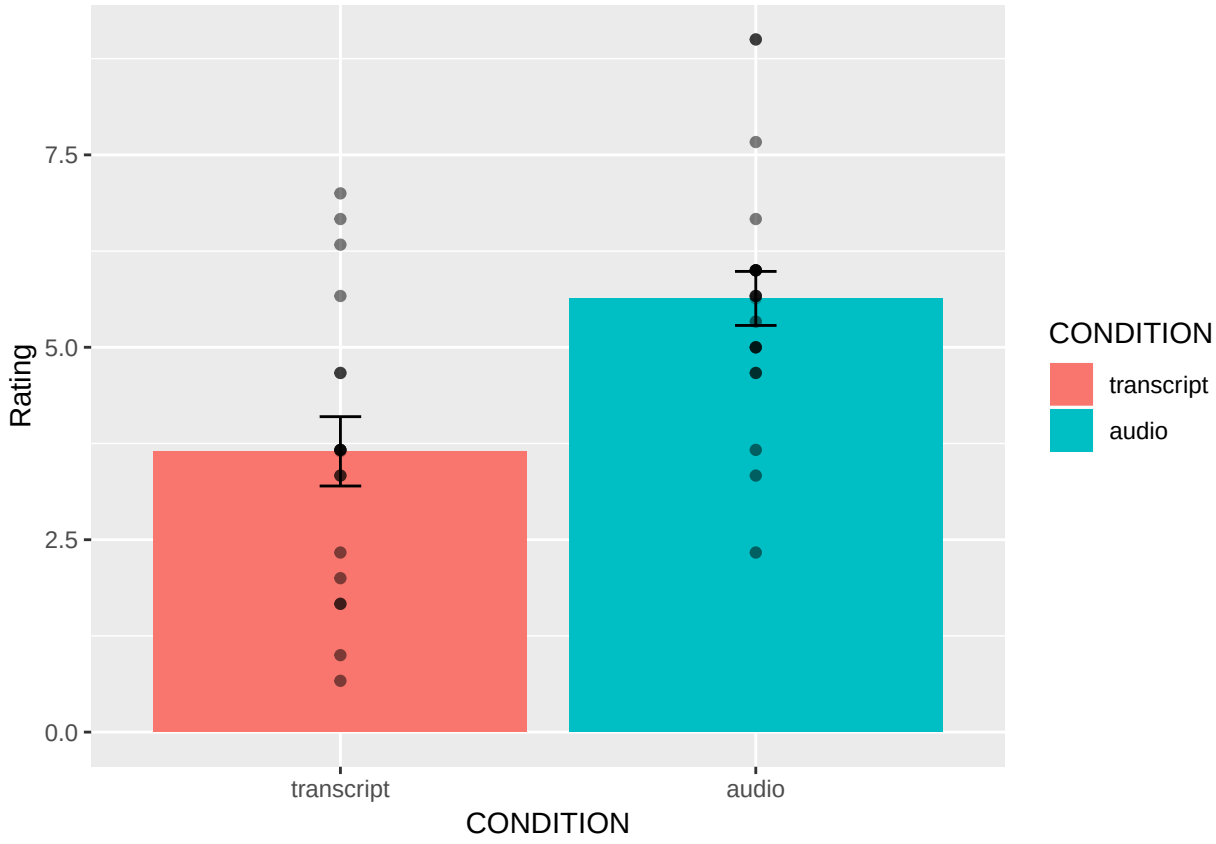
What is the first thing we do before even considering an inferential test? Look at the data. Always look at the data. We could make a dot plot or histogram of the data from the `Intellect_ratings`. But, from our last lab we already learned how to make graphs showing most of the information we would want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

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

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

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

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

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



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

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

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

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

7.4.5 Conduct Independent samples t-test

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

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

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

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

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

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

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

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

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

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

7.4.6 Remaining ratings

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

7.4.7 Reconstructing the graph from the paper

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

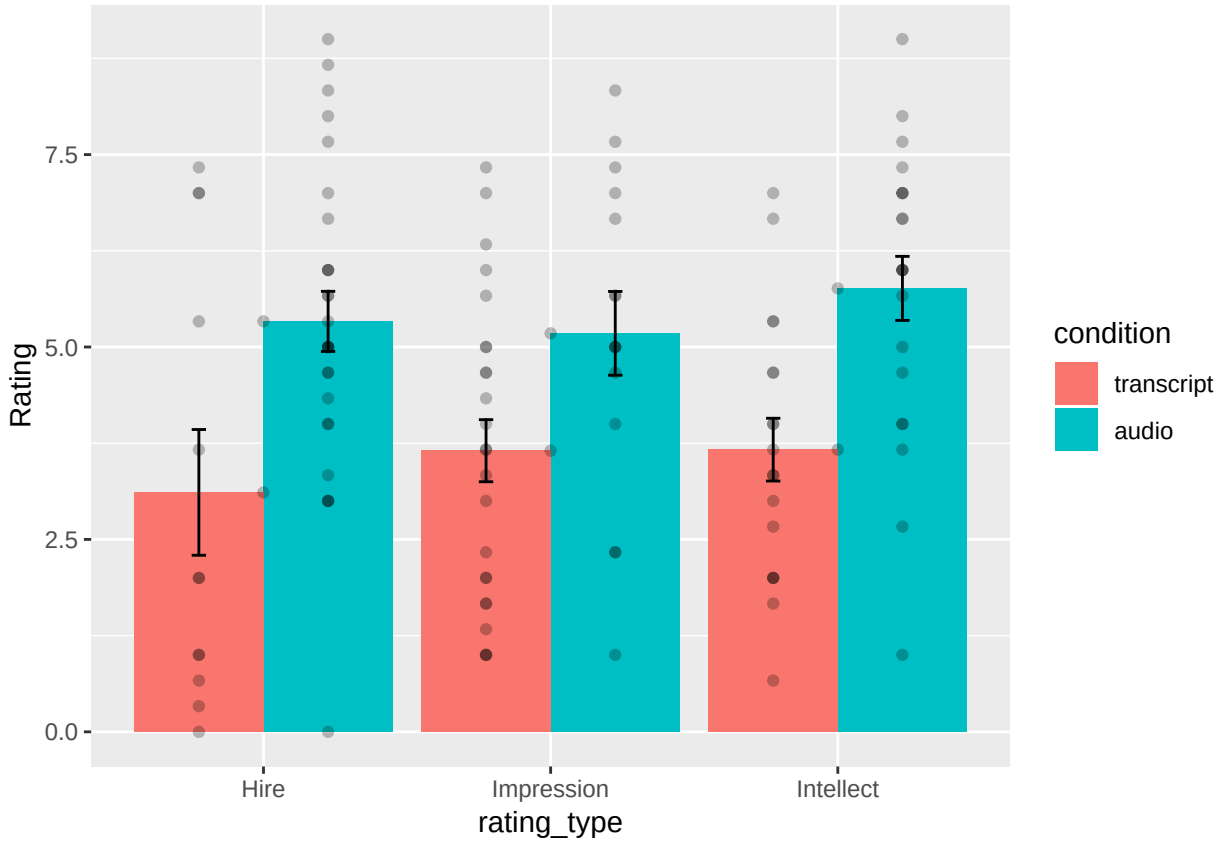
The `all_data` frame is partly long and partly wide. If we are only interested in one dependent measure, then it is a long data frame for that measure. For example, example if we are only interested in plotting `Intellect_Rating`, then we already have one observation of that dependent measure for each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

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

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

```
# repeat CONDITION column three times

condition <- rep(all_data$CONDITION,3)

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

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

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

num_to_repeat <- length(all_data$CONDITION)

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

# put the new variables into a data frame

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

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

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

# Make the plot

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



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

7.5 Excel

How to do it in Excel

7.6 SPSS

How to do it in SPSS

7.7 Matlab

How to do it in Matlab

Chapter 8

Lab 8: One-way ANOVA

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

8.1 Lab Skills Learned

8.2 Important Stuff

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

8.3 Outline of Problem to solve

Stuff we need to say in general

8.3.1 important things

Other things to say

8.4 R

How to do it in R

8.5 Excel

How to do it in Excel

8.6 SPSS

How to do it in SPSS

8.7 Matlab

How to do it in Matlab

Chapter 9

Lab 9: One-way ANOVA

9.1 Lab Skills Learned

9.2 Important Stuff

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

9.3 Outline of Problem to solve

Stuff we need to say in general

9.3.1 important things

Other things to say

9.4 R

How to do it in R

9.5 Excel

How to do it in Excel

9.6 SPSS

How to do it in SPSS

9.7 Matlab

How to do it in Matlab

Chapter 10

Lab 10: Factorial ANOVA

10.1 Lab Skills Learned

10.2 Important Stuff

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

10.3 Outline of Problem to solve

Stuff we need to say in general

10.3.1 important things

Other things to say

10.4 R

How to do it in R

10.5 Excel

How to do it in Excel

10.6 SPSS

How to do it in SPSS

10.7 Matlab

How to do it in Matlab

Chapter 11

Lab 11: Mixed Factorial ANOVA

11.1 Lab Skills Learned

11.2 Important Stuff

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

Stuff we need to say in general

11.2.1 important things

Other things to say

11.3 R

How to do it in R

11.4 Excel

How to do it in Excel

11.5 SPSS

How to do it in SPSS

11.6 Matlab

How to do it in Matlab