Biology 305: Biostatistics

Dr. Jacob C. Cooper & Dr. Melissa Wuellner

Invalid Date

Table of contents

Preface

Welcome to Biology 305 at the University of Nebraska at Kearney! Material in this class was designed by Dr. Melissa Wuellner and adapted by Dr. Jacob C. Cooper for use in *R*.

In this class, you will learn:

1. The basics of study design, the importance of understanding your research situation before embarking on a full study, and practice creating research frameworks based on different scenarios.
2. The basics of data analysis, including understanding what kind of variables are being collected, why understanding variable types are important, and basic tests to understand univariate distributions.
3. Basic multivariate statistics, including ANOVA, correlation, and regression, for comparing multiple different groups.
4. The basics of coding and working in *R* for performing statistical analyses.

This site will help you navigate different homework assignments to perform the necessary *R* tests. Furthermore, this GitHub repository contains all of the homework dataframes, so you will *not* have to manually enter assignments if you use *R* to complete your assignments.

Welcome to class!

Dr. Jacob C. Cooper, BHS 321 Dr. Melissa Wuellner, BHS 345

1 Intro to *R*

In this class, we will be using *R* to perform statistical analyses. *R* is a free software program designed for use in a myriad of statistical and computational scenarios. It can handle extremely large datasets, can handle spatial data, and has wrappers for compatibility with *Python*, *Bash*, and other programs (even *Java*!).

1.1 Setup

First, we need to download *R* onto your machine. We are also going to download *RStudio* to assist with creating *R* scripts and documents.

1.1.1 Installing *R*

First, navigate to the *R* download and install page. Download the appropriate version for your operating system (Windows, Mac, or Linux). **Note** that coding will be formatted slightly different for Windows than for other operating systems. If you have a Chromebook, you will have to follow the online instructions for installing both programs on Chrome.

Follow the installation steps for *R*, and verify that the installation was successful by searching for *R* on your machine. You should be presented with a coding window that looks like the following:

```
R version 4.4.1 (2024-06-14) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
```

```
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

If that screen appears, congratulations! *R* is properly installed. If the install was not successful, please talk to Dr. Cooper and check with your classmates as well.

1.1.2 Installing *RStudio*

RStudio is a GUI (graphics user interface) that helps make *R* easier to use. Furthermore, it allows you to create documents in *R*, including websites (such as this one), PDFs, and even presentations. This can greatly streamline the research pipeline and help you publish your results and associated code in a quick and efficient fashion.

Head over the the *RStudio* download website and download "*RStudio* Desktop", which is free. Be sure to pick the correct version for your machine.

Open *RStudio* on your machine. You should be presented with something like the following:

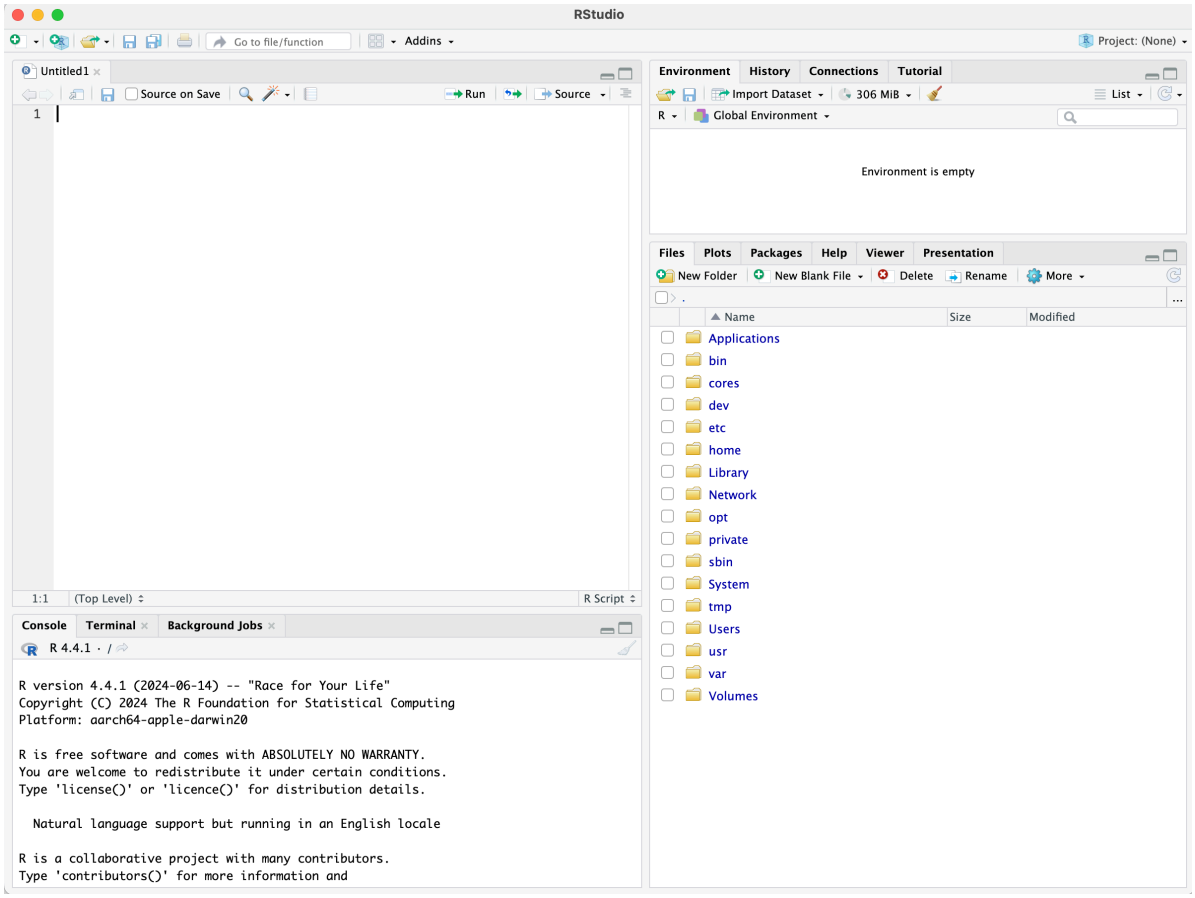


Figure 1.1: *RStudio* start window. Note that the screen is split into four different quadrants. Top left: *R* documents; bottom left: *R* program; top right: environment window; bottom right: plots, help, and directories.

In *RStudio*, the top left window is always going to be our coding window. This is where we will type all of our code and create our documents. In the bottom left we will see *R* executing the code. This will show what the computer is "thinking" and will help us spot any potential issues. The top right window is the "environment", which shows what variables and datasets are stored within the computers' memory. (It can also show some other things, but we aren't concerned with that at this point). The bottom right window is the "display" window. This is where plots and help windows will appear if they don't appear in the document (top left) window itself.

1.1.3 Setting up the "native pipe" operator

There are two ways to type commands in *R*: in a "nested" fashion and in a "piped" fashion.

Let's say that we have a random series of 100 numbers from a random normal distribution, which we can get by running the following:

```
# make repeatable
set.seed(42)
# get 100 values from normal
x <- rnorm(100)
```

Let's say that we want to get the mean of these values *and* round these values to two decimal places for our final answer. To do this in a nested fashion, we would do the following:

```
# round mean of x
round(mean(x), 2)
```

```
[1] 0.03
```

This works pretty well, but it can get confusion with so many parentheses over each other. Thus, we can use the "pipe" method instead.

In *RStudio*, click on the tabs at the top of the screen to go to Tools > Global Options, and in the pop up screen select Code. On this screen, you should see a tick box for "Use native pipe operator". Make sure this box is checked. Now, we can use `CTRL` + `SHIFT` + `M` to insert the "pipe" operator `|>`.

The pipe operator takes whatever came previous and puts it through the next command. For example, `mean(x)` could also be written as `x |> mean()`, where `x` would be placed inside of `mean()`.

```
x |>
  mean() |>
  round(2)
```

```
[1] 0.03
```

As you can see, the above breaks things down into a more step-by-step fashion, and makes code easier to follow.

1.2 Creating an *RMarkdown* document

1.2.1 Setup

In this class, we will be creating assignments in what is called *RMarkdown*. This is a rich-text version of *R* that allows us to create documents with the code embedded. In *RStudio*, click the "+" button in the far top left to open the `New Document` menu. Scroll down this list and click on `R Markdown`.

A screen such as this will appear:

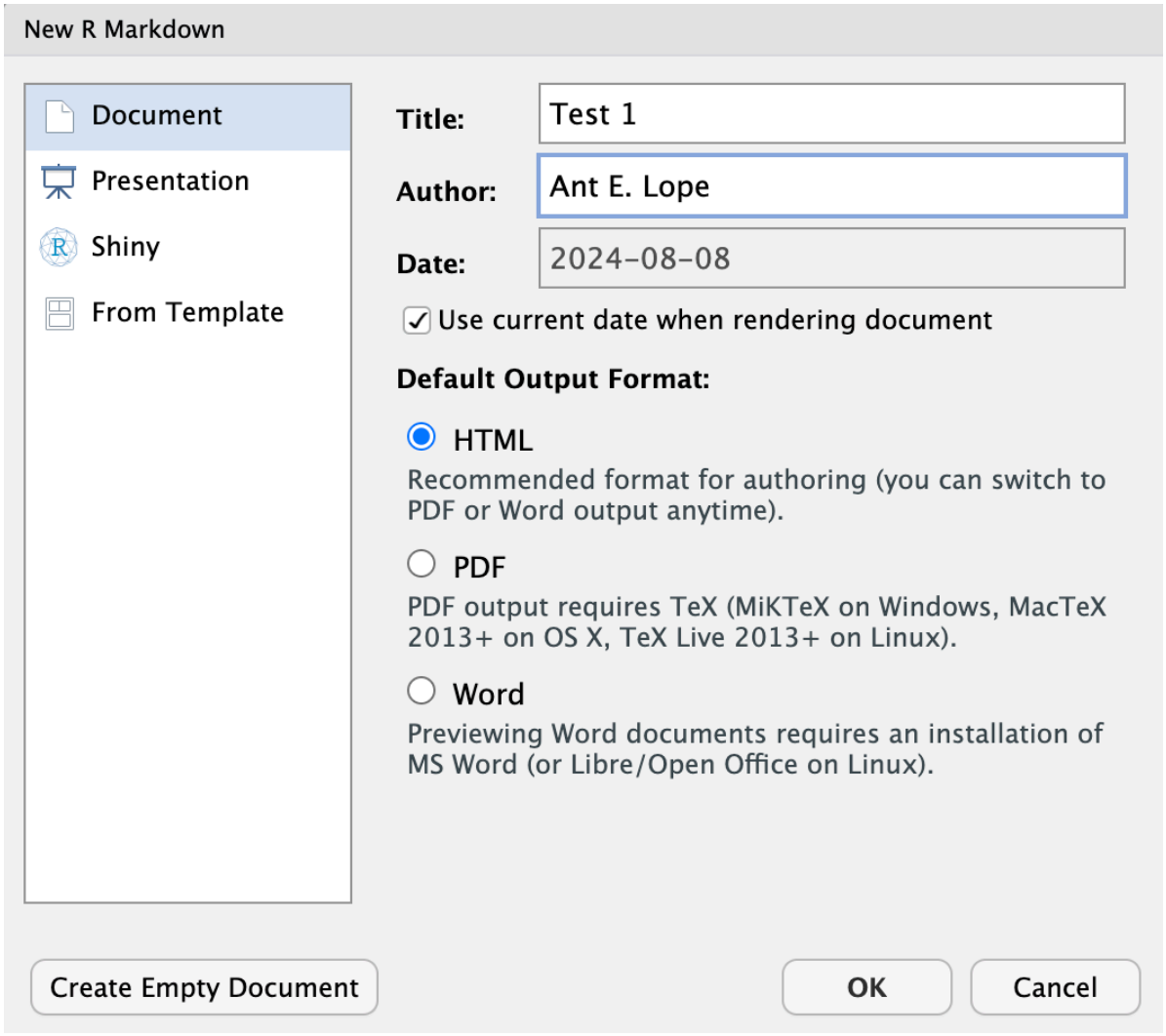


Figure 1.2: A new file window for an *RMarkdown* file.

After entering a title and your name and selecting `document` in the left hand menu, click `OK`.

Figure 1.3: An example of a markdown script.

In the image above, we can see what a "default" *RMarkdown* script looks like after creating the file. At the top of the document, between all of the dashes, we have the `yaml` header that tells *R* what kind of document will be created, who the author is, and tells it to use today's date. In this class, we will be saving documents as `html` as they are the easiest documents to create and save. These documents will include all of your code, text, and even any plots you may create!

Plain text in the document will be rendered as plain text in the document. (I.e., whatever you type normally will become "normal text" in the finished document). Lines preceded with `#` will become headers, with `##` being a second level header and `###` being a third level header, etc. Words can also be made italic by putting an asterisk on each side of the word (`*italic*`) and bold by putting two asterisks on each side (`**bold**`). URLs are also supported, with `<>` on each side of a URL making it clickable, and words being hyperlinked by typing `[words to show](target URL)`.

We also have code "chunks" that are shown above. A code chunk can be manually typed out or inserted by pressing `CTRL` + `ALT` + `I` (Windows, Linux) or `COMMAND` + `OPTION` + `I` (Mac).

Everything inside a "code chunk" will be read as *R* code and executed as such. Note that you can have additional commands in the *R* chunks, but we won't cover that for now.

1.2.2 Using code chunks

In your computer, erase all information except for the `yaml` header between the dashes on your computer. `Save` your file in a folder where you want your assignment to be located. It is important you do this step up front as the computer will sometimes save in random places if you don't specify a file location at the beginning. *Don't forget to save your work frequently*!

````
This is a test of the *RMarkdown* code.

```{r}
print("Hello world!")
```
````

Figure 1.4: Text to type in your *Rmarkdown* document.

After typing this into the document, hit `knit` near the top of the upper left window. *R* will now create an HTML document that should look like this:

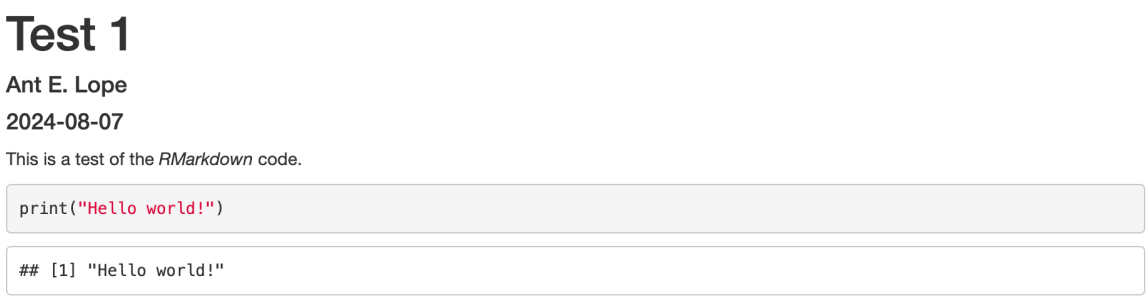


Figure 1.5: The output from the above code knitted into a document.

We can see now that the `HTML` document has the title of the document, the author's name, the date on which the code was run, and a greyed-out box with color coded *R* code followed by the output. Let's try something a little more complex. Create a new code chunk and type the following:

```
x <- 1:10
```

This will create a variable in *R*, `x`, that is sequentially each whole number between 1 and 10. We can see this by highlighting or typing only the letter `x` and running that line of code by clicking `CTRL` + `ENTER` (Windows / Linux) or `COMMAND` + `ENTER` (Mac).

```
x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

If you look at the top right window, you will also see the value `x` in the environment defined as `int [1:10] 1 2 3 4 5 6 7 8 9 10`. This indicates that `x` is integer data spanning ten positions numbered 1 to 10. Since the vector is small, it displays every number in the sequence.

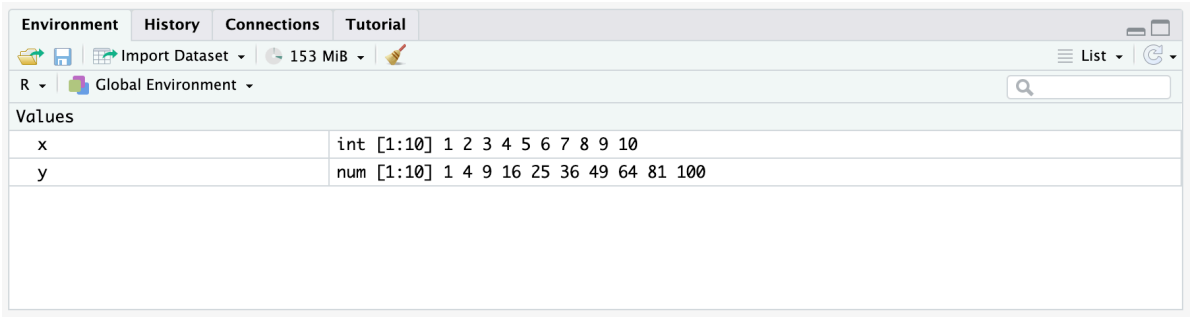


Figure 1.6: *RStudio* environment window showing saved objects. These are in the computer's memory.

Let's create another vector `y` that is the squared values of `x`, such that $y = x^2$. We can raise values to an exponent by using `^`.

```
y <- x^2
y
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

Now we have the value `y` in the environment that is the square of the values of `x`. This is a `numeric` vector of 10 values numbered 1 to 10 where each value corresponds to a square of the `x` value. We can raise things to any value however, including x^x!

```
x^x
```

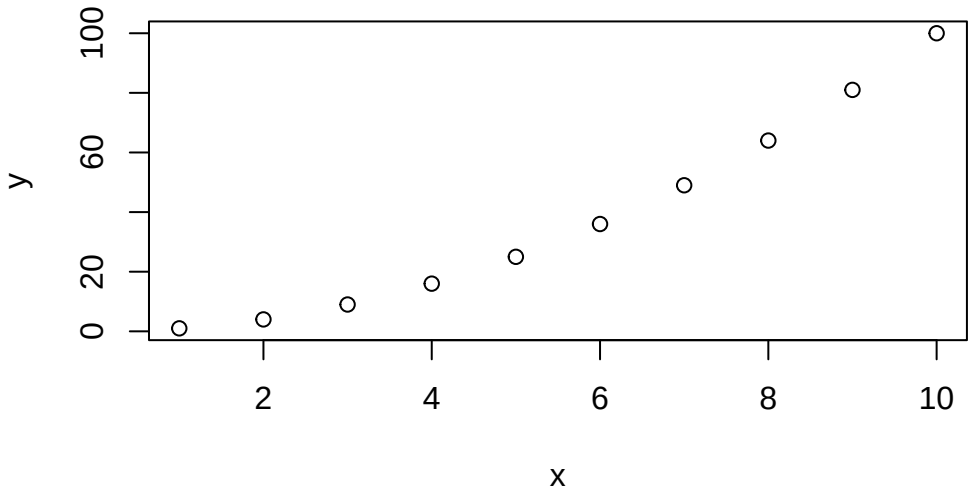
```
 [1]           1           4          27         256        3125       46656
 [7]      823543    16777216   387420489 10000000000
```

As we can see, since I didn't "store" this value as a variable in *R* using `<-`, the value is not in the environment.

1.2.3 Plotting

Now, let's try creating a plot. This is easy in *R*, as we just use the command `plot`.

```
plot(x = x, y = y)
```



By specifying the `y` and `x` components in `plot`, we can quickly generate a point plot. We can alter the visual parameters of this plot using a few different commands. I will outline these below with inline notes. Inline notes in the code can be made by using a `#` symbol before them, which basically tells *R* to ignore everything after the `#`. For example:

```
print("Test")
```

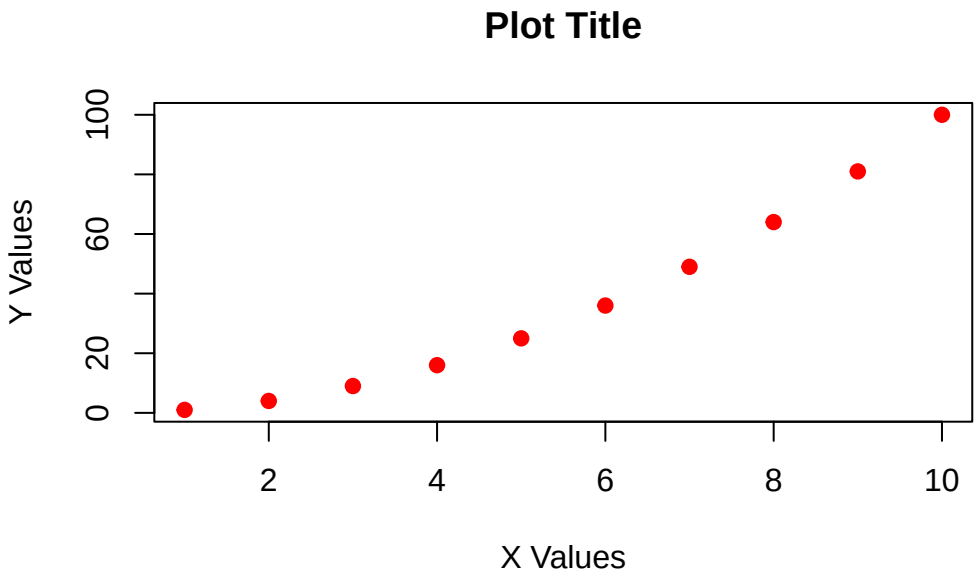
```
[1] "Test"
```

```
# print("Test 2")
```

This prints the word `Test`, but doesn't print `Test 2`.

Now let's make the plot with some new visual parameters.

```
plot(x = x, # specify x values
     y = y, # specify y values
     ylab = "Y Values", # specify Y label
     xlab = "X Values", # specify X label
     main = "Plot Title", # specify main title
     pch = 19, # adjust point style
     col = "red") # make points red
```



1.2.4 Tab complete

RStudio allows for "tab-completing" while typing code. Tab-completing is a way of typing the first part of a command, variable name, or file name and hitting "tab" to show all options with that spelling. You should use tab completing because it:

- reduces spelling mistakes
- reduces filepath mistakes
- increases the speed at which you code
- provides help with specific functions

1.2.5 Help

At any point in *R*, you can look up "help" for a specific function by typing `?functionname`. Try this on your computer with the following:

```
?mean
```

1.3 Working with data

Throughout this course, we are going to have to work with datasets that are from our book or other sources. Here, we are going to work through an example dataset. First, we need to install *libraries*. A *library* is a collated, pre-existing batch of code that is designed to assist with data analysis or to perform specific functions. These *libraries* make life a lot easier, and create short commands for completing relatively complex tasks.

In this class, there is one major library that you will need *almost every week*! First, we need to install this library:

1. `tidyverse`: this package is actually a group of packages designed to help with data analysis, management, and visualization.

NOTE: If you leave the install prompts in your RMarkdown document, *it will not knit!*

```
# run this code the first time ONLY
# DO NOT INCLUDE IN RMD FILE
# does not need to be run every time you use R!

# tidyverse has a bunch of packages in it!
# great for data manipulation
install.packages("tidyverse")

# if you ever need to update:
# leaving brackets open means "update everything"
update.packages()
```

After packages are installed, we will need to load them into our *R* environment. While we only need to `install.packages` once on our machine, we need to load libraries *every time we restart the program*!

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

You should an output like the above. What this means is:

1. The core packages that comprise the tidyverse loaded successfully, and version numbers for each are shown.
2. The conflicts basically means that certain commands will not work as they used to because *R* has "re-learned" a particular word.

To clarify the conflicts, pretend that you can only know one definition of a word at a time. You may know the word "cola" as a type of soda pop or as a drink in general. However, in Spanish, "cola" refers to a line or a tail. While we can learn both of these definitions and know which one is which because of context, a computer can't do that. In *R*, we would then have to specify which "cola" we are referring to. We do this by listing the package before the command; in this case, `english::cola` would mean a soda pop and `spanish::cola` would refer to a line or tail. If we just type `cola`, the computer will assume one of these definitions but not even consider the other.

We won't have to deal with conflicts much in this class, and I'll warn you (or help you) if there is a conflict.

1.3.1 Downloading data

Now, we need to download our first data set. These datasets are stored on GitHub. We are going to be looking at data from Dr. Cooper's dissertation concerning Afrotropical bird distributions (Cooper 2021). This website is in the data folder on this websites' GitHub page, accessible here.

```
# read comma separated file (csv) into R memory
# reads directly from URL
ranges <- read_csv("https://raw.githubusercontent.com/jacobccooper/biol305_unk/main/datasets/
```

```
Rows: 12 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): species
dbl (9): combined_current_km2, consensus_km2, bioclim_current_km2, 2050_comb...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Alternatively, we can use the operator `|>` to simplify this process. `|>` means "take whatever you got from the previous step and *pipe* it into the next step". So, the following does the exact same thing:

```
ranges <- "https://raw.githubusercontent.com/jacobccooper/biol305_unk/main/datasets/lacustri
  read_csv()
```

```
Rows: 12 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): species
dbl (9): combined_current_km2, consensus_km2, bioclim_current_km2, 2050_comb...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Using the `|>` is preferred as you can better set up a workflow and because it more closely mimics other coding languages, such as `bash`.

Let's view the data to see if it worked. We can use the command `head` to view the first few rows:

```
head(ranges)
```

```
# A tibble: 6 x 10
  species                combined_current_km2 consensus_km2 bioclim_current_km2
  <chr>                                 <dbl>         <dbl>                <dbl>
1 Batis_diops                          25209.         6694.               19241.
2 Chamaetylas_poliophrys               68171.         1106.               68158.
3 Cinnyris_regius                      60939.        13305.               53627.
4 Cossypha_archeri                     27021.         6409.               11798.
5 Cyanomitra_alinae                    78680.        34320.               63381.
```

```
6 Graueria_vittata                        8770.          861.               8301.
# i 6 more variables: `2050_combined_km2` <dbl>, `2050_consensus_km2` <dbl>,
#   `2070_combined_km2` <dbl>, `2070_consensus_km2` <dbl>,
#   alltime_consensus_km2 <dbl>, past_stable_km2 <dbl>
```

We can perform a lot of summary statistics in *R*. Some of these we can view for multiple columns at once using `summary`.

```
summary(ranges)
```

```
   species           combined_current_km2 consensus_km2      bioclim_current_km2
 Length:12          Min.   :  8770       Min.   :  861.3   Min.   :  3749
 Class :character   1st Qu.: 24800       1st Qu.: 4186.2   1st Qu.: 10924
 Mode  :character   Median : 43654       Median : 7778.1   Median : 31455
                    Mean   : 68052       Mean   :18161.8   Mean   : 42457
                    3rd Qu.: 70798       3rd Qu.:18558.7   3rd Qu.: 62835
                    Max.   :232377       Max.   :79306.6   Max.   :148753
 2050_combined_km2 2050_consensus_km2 2070_combined_km2  2070_consensus_km2
 Min.   :  1832    Min.   :    0.0    Min.   :   550.3   Min.   :    0.0
 1st Qu.:  6562    1st Qu.:  589.5    1st Qu.:  6583.8   1st Qu.:  311.4
 Median : 26057    Median : 6821.9    Median : 24281.7   Median : 2714.6
 Mean   : 33247    Mean   :14418.4    Mean   : 31811.0   Mean   : 8250.5
 3rd Qu.: 40460    3rd Qu.:18577.1    3rd Qu.: 38468.9   3rd Qu.:10034.4
 Max.   :132487    Max.   :79236.2    Max.   :129591.0   Max.   :53291.8
 alltime_consensus_km2 past_stable_km2
 Min.   :    0.0       Min.   :   0.0
 1st Qu.:  790.9       1st Qu.:   0.0
 Median : 8216.8       Median :   0.0
 Mean   :15723.3       Mean   : 127.3
 3rd Qu.:19675.0       3rd Qu.:   0.0
 Max.   :82310.5       Max.   :1434.8
```

As seen above, we now have information for the following statistics for each variable:

- `Min` = minimum
- `1st Qu.` = 1st quartile
- `Median` = middle of the dataset
- `Mean` = average of the dataset
- `3rd Qu.` = 3rd quartile
- `Max.` = maximum

We can also calculate some of these statistics manually to see if we are doing everything correctly. It is easiest to do this by using predefined functions in *R* (code others have written to perform a particular task) or to create our own functions in *R*. We will do both to determine the average of `combined_current_km2`.

1.3.2 Subsetting data

First, we need to select only the column of interest. In *R*, we have two ways of subsetting data to get a particular column.

- `var[rows,cols]` is a way to look at a particular object (`var` in this case) and choose a specific combination of `row` number and `column` number (`col`). This is great if you know a specific index, but it is better to use a specific name.
- `var[rows,"cols"]` is a way to do the above but by using a specific column name, like `combined_current_km2`.
- `var$colname` is a way to call the specific column name directly from the dataset.

```
# using R functions

ranges$combined_current_km2
```

```
 [1]  25209.4  68171.2  60939.2  27021.3  78679.9   8769.9 232377.2  17401.4
 [9]  51853.5  35455.1  23570.3 187179.1
```

As shown above, calling the specific column name with `$` allows us to see only the data of interest. We can also save these data as an object.

```
current_combined <- ranges$combined_current_km2

current_combined
```

```
 [1]  25209.4  68171.2  60939.2  27021.3  78679.9   8769.9 232377.2  17401.4
 [9]  51853.5  35455.1  23570.3 187179.1
```

Now that we have it as an object, specifically a numeric vector, we can perform whatever math operations we need to on the dataset.

```
mean(current_combined)
```

```
[1] 68052.29
```

Here, we can see the mean for the entire dataset. However, we should always round values to the same number of decimal points as the original data. We can do this with `round`.

```
round(mean(current_combined),1) # round mean to one decimal
```

```
[1] 68052.3
```

Note that the above has a nested set of commands. We can write this exact same thing as follows:

```
# pipe mean through round
mean(current_combined)  |>
  round(1)
```

```
[1] 68052.3
```

Use the method that is easiest for you to follow!

We can also calculate the mean manually. The mean is $\frac{\sum_{i=1}^{n} x}{n}$, or the sum of all the values within a vector divided by the number of values in that vector.

```
# create function
# use curly brackets to denote function
# our data goes in place of "x" when finally run
our_mean <- function(x){
  sum_x <- sum(x) # sum all values in vector
  n <- length(x) # get length of vector
  xbar <- sum_x/n # calculate mean
  return(xbar) # return the value outside the function
}
```

Let's try it.

```
our_mean(ranges$combined_current_km2)
```

```
[1] 68052.29
```

As we can see, it works just the same as **mean**! We can round this as well.

```
our_mean(ranges$combined_current_km2) |>
  round(1)
```

```
[1] 68052.3
```

1.4 Your turn!

With a partner or on your own, try to do the following:

1. Create an *RMarkdown document* that will save as an `.html`.
2. Load the data, as shown here, and print the summary statistics in the document.
3. Calculate the value of `combined_current_km2` divided by `2050_combined_km2` and print the results.
4. `knit` your results, with your name(s) and date, as an HTML document.

Let me know if you have any issues.

2 Descriptive Statistics

2.1 Purposes of descriptive statistics

Descriptive statistics enable researchers to quickly and easily examine the "behavior" of their datasets, identifying potential errors and allowing them to observe particular trends that may be worth further analysis. Here, we will cover how to calculate descriptive statistics for multiple different datasets, culminating in an assignment covering these topics.

2.2 Preparing *R*

As with every week, we will need to load our relevant packages first. This week, we are using the following:

```
# enables data management tools
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

2.3 Downloading the data

For the example this week, we will be using the `starbucks` dataset, describing the number of drinks purchased during particular time periods during the day.

```
starbucks <- read_csv("https://raw.githubusercontent.com/jacobccooper/biol105_unk/main/datas
```

```
Rows: 9 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): Hour
dbl (1): Frap_Num

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

2.4 Descriptive statistics

Descriptive statistics are statistics that help us understand the shape and nature of the data on hand. These include really common metrics such as *mean*, *median*, and *mode*, as well as more nuanced metrics like *quartiles* that help us understand if there is any *skew* in the dataset. (*Skew* refers to a bias in the data, where more data points lie on one side of the distribution and there is a long *tail* of data in the other direction).

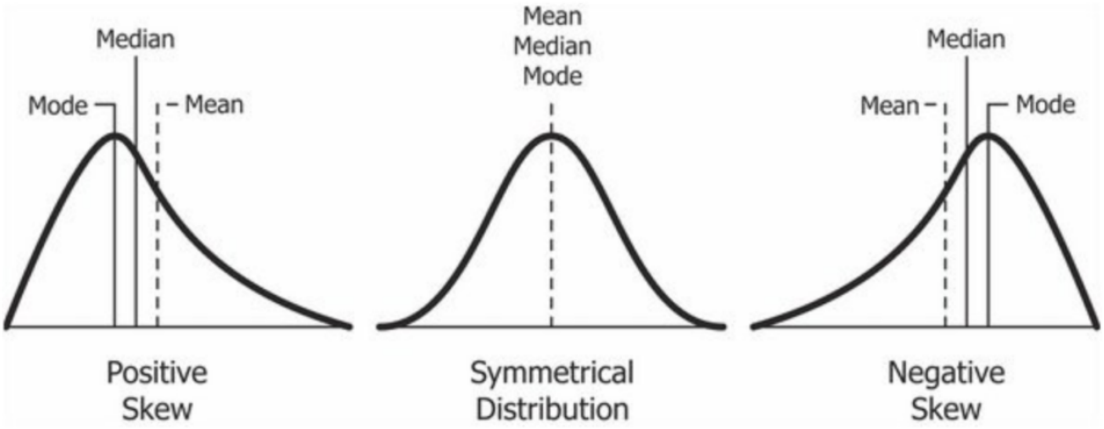


Figure 2.1: Examples of skew compared to a symmetrical, non-skewed distribution. Source: machinelearningparatodos.com

Note above that the relative position of the *mean*, *median*, and *mode* can be indicative of skew. Please also note that these values will rarely be exactly equal "in the real world", and thus you need to weigh differences against the entire dataset when assessing skew. There is a lot of nuance like this in statistics; it is not always an "exact" science, but sometimes involves judgment calls and assessments based on what you observe in the data.

Using the `starbucks` dataset, we can look at some of these descriptive statistics to understand what is going on.

2.4.1 Notation

As a quick reminder, we use Greek lettering for *populations* and Roman lettering for samples. For example:

- σ is a population, but s is a sample (both these variables refer to *standard deviation*).
- μ is a population, but $\bar{x}$ is a sample (both of these variables refer to the *mean*).

2.4.2 Mean

The mean is the "average" value within a set of data, specifically, the sum of all values divided by the length of those values: $\frac{\sum_{i=1}^{n} x}{n}$.

```
head(starbucks)
```

```
# A tibble: 6 x 2
  Hour      Frap_Num
  <chr>        <dbl>
1 0600-0659        2
2 0700-0759        3
3 0800-0859        2
4 0900-0959        4
5 1000-1059        8
6 1100-1159        7
```

Here, we are specifically interested in the number of frappuccinos.

```
# get vector of frappuccino number
fraps <- starbucks$Frap_Num

# get mean of vector
mean(fraps)
```

```
[1] 6.222222
```

Note that the above should be rounded to a whole number, since we were given the data in whole numbers!

```
mean(fraps) |>
  round(0)
```

```
[1] 6
```

```
# OR

round(mean(fraps),0)
```

```
[1] 6
```

We already covered calculating the average manually in our previous tutorial, but we can do that here as well:

```
# sum values
# divide by n, length of vector
# round to 0 places
round(sum(fraps)/length(fraps),0)
```

```
[1] 6
```

2.4.3 Range

The range is the difference between the largest and smallest units in a dataset. We can use the commands `min` and `max` to calculate this.

```
max(fraps) - min(fraps)
```

```
[1] 13
```

The range of our dataset is 13.

2.4.4 Median

The median is also known as the 50th percentile, and is the midpoint of the data when ordered from least to greatest. If there are an even number of data points, then it is the average point between the two center points. For odd data, this is the $\frac{n+1}{2}$th observation. For even data, since we need to take an average, this is the $\frac{\frac{n}{2}+(\frac{n}{2}+1)}{2}$. You should be able to do these by hand and by using a program.

```
median(fraps)
```

```
[1] 4
```

Now, to calculate by hand:

```
length(fraps)
```

```
[1] 9
```

We have an odd length.

```
# order gets the order
order(fraps)
```

```
[1] 1 3 7 2 4 6 5 9 8
```

```
# [] tells R which elements to put where
frap_order <- fraps[order(fraps)]

frap_order
```

```
[1]  2  2  2  3  4  7  8 13 15
```

```
# always use parentheses
# make sure the math maths right!
(length(frap_order)+1)/2
```

```
[1] 5
```

Which is the fifth element in the vector?

```
frap_order[5]
```

```
[1] 4
```

Now let's try it for an even numbers.

```
# remove first element
even_fraps <- fraps[-1]

even_fraps_order <- even_fraps[order(even_fraps)]

even_fraps_order
```

```
[1]  2  2  3  4  7  8 13 15
```

```
median(even_fraps)
```

```
[1] 5.5
```

Now, by hand: $\frac{\frac{n}{2}+(\frac{n}{2}+1)}{2}$.

```
n <- length(even_fraps_order)

# get n/2 position from vector
m1 <- even_fraps_order[n/2]
# get n/2+1 position
m2 <- even_fraps_order[(n/2)+1]

# add these values, divide by two for "midpoint"
med <- (m1+m2)/2

med
```

```
[1] 5.5
```

As we can see, these values are equal!

2.4.5 Other quartiles and quantiles

We also use the 25th percentile and the 75th percentile to understand data distributions. These are calculated similar to the above, but the bottom quartile is only $\frac{1}{4}$ of the way between values and the 75th quartile is $\frac{3}{4}$ of the way between values. We can use the R function `quantile` to calculate these.

```
quantile(frap_order)
```

```
  0%  25%  50%  75% 100%
   2    2    4    8   15
```

We can specify a quantile as well:

```
quantile(frap_order, 0.75)
```

```
75%
  8
```

We can also calculate these metrics by hand. Let's do it for the even dataset, since this is more difficult.

```
quantile(even_fraps_order)
```

```
   0%   25%   50%   75%  100%
 2.00  2.75  5.50  9.25 15.00
```

Note that the 25th and 75th percentiles are also between two different values. These can be calculated as a quarter and three-quarters of the way between their respective values.

```
# 75th percentile

n <- length(even_fraps_order)

# get position
p <- 0.75*(n+1)

# get lower value
# round down
```

```r
m1 <- even_fraps_order[trunc(p)]

# get upper value
# round up
m2 <- even_fraps_order[ceiling(p)]

# position between
# fractional portion of rank
frac <- p-trunc(p)

# calculate the offset from lowest value
val <- (m2 - m1)*frac

# get value
m1 + val
```

```
[1] 11.75
```

Wait… why does our value differ?

R, by default, calculates quantiles using what is called `Type 7`, in which the quantiles are calculated by $p_k = \frac{k-1}{n-1}$, where n is the length of the vector and k refers to the quantile being used. However, in our book and in this class, we use `Type 6` interpretation - $p_k = \frac{k}{n+1}$. Let's try using `Type 6`:

```r
quantile(even_fraps_order, type = 6)
```

```
   0%   25%   50%   75%  100% 
 2.00  2.25  5.50 11.75 15.00 
```

Now we have the same answer as we calculated by hand!

This is a classic example of how things in *R* (and in statistics in general!) can depend on interpretation and are not always "hard and fast" rules.

In this class, we will be using Type 6 interpretation for the quantiles - you will have to specify this in the quantile function EVERY TIME! If you do *not* specify Type 6, you will get the questions incorrect and you will get answers that do not agree with the book, with Excel, or what you calculate by hand.

2.4.6 Mode

There is no default method for finding the mode in *R*. However, websites like Statology provide wraparound functions.

```
# Based on Statology function
# define function to calculate mode
find_mode <- function(x) {
  # get unique values from vector
  u <- unique(x)
  # count number of occurrences for each value
  tab <- tabulate(match(x, u))

  # if no mode, say so
  if(length(x)==length(u[tab == max(tab)])){
    print("No mode.")
  }else{
    # return the value with the highest count
    u[tab == max(tab)]
  }
}

find_mode(fraps)
```

```
[1] 2
```

We can also do this by hand, by counting the number of occurrences of each value. This can be done in a stepwise fashion using commands in the above function.

```
# unique counts
u <- unique(fraps)
u
```

```
[1]  2  3  4  8  7 15 13
```

```
# which elements match
match(fraps,u)
```

```
[1] 1 2 1 3 4 5 1 6 7
```

```
# count them
tab <- match(fraps,u) |>
  tabulate()

tab
```

```
[1] 3 1 1 1 1 1 1
```

Get the highest value.

```
u[tab==max(tab)]
```

```
[1] 2
```

Notice this uses `==`. This is a logical argument that means "is equal to" or "is the same as". For example:

```
2 == 2
```

```
[1] TRUE
```

These values are the same, so `TRUE` is returned.

```
2 == 3
```

```
[1] FALSE
```

These values are unequal, so `FALSE` is returned. *R* will read `TRUE` as `1` and `FALSE` as `ZERO`, such that:

```
sum(2==2)
```

```
[1] 1
```

and

```
sum(2==3)
```

```
[1] 0
```

This allows you to find how many arguments match your condition quickly, and even allows you to subset based on these indices as well. Keep in mind you can use greater than `<`, less than `>`, greater than or equal to `<=`, less than or equal to `>=`, is equal to `==`, and is not equal to `!=` to identify numerical relationships. Other logical arguments include:

- `&`: both conditions must be `TRUE` to match (e.g., `c(10,20) & c(20,10)`). Try the following as well: `fraps < 10 & fraps > 3`.
- `&&`: and, but works with single elements and allows for better parsing. Often used with `if`. E.g., `fraps < 10 && fraps > 3`. This will not work on our multi-element `frap` vector.
- `|`: or, saying at least one condition must be true. Try: `fraps > 10 | fraps < 3`.
- `||`: or, but for a single element, like `&&` above.
- `!`: not, so "not equal to" would be `!=`.

2.4.7 Variance

When we are dealing with datasets, the variance is a measure of the total spread of the data. The variance is calculated using the following:

$$\sigma^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

Essentially, this means that for every value of x, we are finding the difference between that value and the mean and squaring it, summing all of these quared differences, and dividing them by the number of samples in the dataset minus one. Let's do this for the frappuccino dataset.

```
frap_order
```

```
[1]  2  2  2  3  4  7  8 13 15
```

Now to find the differences.

```r
diffs <- frap_order - mean(frap_order)

diffs
```

```
[1] -4.2222222 -4.2222222 -4.2222222 -3.2222222 -2.2222222  0.7777778  1.7777778
[8]  6.7777778  8.7777778
```

Note that R is calculating the same thing for the entire vector! Since these are differences from the mean, they should sum to zero.

```r
sum(diffs)
```

```
[1] 3.552714e-15
```

This is not quite zero due to rounding error, but is essentially zero as it is 0.0000000000000036.

```r
# square differences
diffs_sq <- diffs^2

diffs_sq
```

```
[1] 17.8271605 17.8271605 17.8271605 10.3827160  4.9382716  0.6049383  3.1604938
[8] 45.9382716 77.0493827
```

Now we have the squared differences. We need to sum these and divide by $n - 1$.

```r
n <- length(frap_order)

var_frap <- sum(diffs_sq)/(n-1)

var_frap
```

```
[1] 24.44444
```

Let's check this against the built-in variance function in R.

```r
var(frap_order)
```

```
[1] 24.44444
```

They are identical! We can check this using a logical argument.

```
var_frap == var(frap_order)
```

```
[1] TRUE
```

Seeing as this is `TRUE`, we calculated it correctly.

2.4.8 Standard deviation

Another common measurement of spread is the standard deviation (σ). As you remember from class (or may have guessed from the notation on this site), the standard deviation is just the square root of the variance.

```
sqrt(var_frap)
```

```
[1] 4.944132
```

We can test this against the built in `sd` function in *R*:

```
sqrt(var_frap) == sd(frap_order)
```

```
[1] TRUE
```

As you can see, we calculated this correctly!

2.4.9 Standard error

The standard error is used to help understand the spread of data and to help estimate the accuracy of our measurements for things like the mean. The standard error is calculated thusly:

$$SE = \frac{\sigma}{\sqrt{n}}$$

There is not built in function for the standard error in excel, but we can write our own:

```r
se <- function(x){
  n <- length(x) # calculate n
  s <- sd(x) # calculate standard deviation
  se_val <- s/sqrt(n)
  return(se_val)
}
```

Let's test this code.

```r
se(frap_order)
```

```
[1] 1.648044
```

Our code works! And we can see exactly how the standard error is calculate. We can also adjust this code as needed for different situations, like samples.

Remember, the standard error is used to help reflect our *confidence* in a specific measurement (*e.g.*, how certain we are of the mean, and what values we believe the mean falls between). We want our estimates to be as precise as possible with as little uncertainty as possible. Given this, does having more samples make our estimates more or less confident? Mathematically, what happens as our sample size *increases*?

2.4.10 Coefficient of variation

The coefficient of variation, another measure of data spread and location, is calculated by the following:

$$CV = \frac{\sigma}{\mu}$$

We can write a function to calculate this in R as well.

```r
cv <- function(x){
  sigma <- sd(x)
  mu <- mean(x)
  val <- sigma/mu
  return(val)
}

cv(frap_order)
```

```
[1] 0.7945927
```

Remember that we will need to round values.

2.4.11 Outliers

Outliers are any values that are outside of the 1.5 times the interquartile range. We can calculate this for our example dataset as follows:

```
lowquant <- quantile(frap_order,0.25,type = 6) |> as.numeric()

hiquant <- quantile(frap_order,0.75,type = 6) |> as.numeric()

iqr <- hiquant - lowquant
```

We can also calculate the interquartile range using `IQR`. **Remember**, you must use `type = 6`!

```
iqr <- IQR(frap_order, type = 6)
```

Now, to calculate the "whiskers".

```
lowbound <- lowquant - (1.5*iqr)
hibound <- hiquant + (1.5*iqr)

# low outliers?
# select elements that match
# identify using logical "which"
frap_order[which(frap_order < lowbound)]
```

```
numeric(0)
```

```
# high outliers?
# select elements that match
# identify using logical "which"
frap_order[which(frap_order > hibound)]
```

```
numeric(0)
```

We have no outliers for this particular dataset.

2.5 Homework: Descriptive Statistics

Now that we've covered these basic statistics, it's your turn! For this week, you will be completing homework that involves methods from Chapter 4 in your book.

2.5.1 Homework instructions

Please create an *RMarkdown* document that will render as an `.html` file. You will submit this file to show your coding and your work. Please refer to the Introduction to *R* for refreshers on how to create an `.html` document in *RMarkdown*. You will need to do the following for each of these datasets:

- mean
- median
- range
- interquartile range
- variance
- standard deviation
- coefficient of variation
- standard error
- whether there are any "outliers"

Please show all of your work for full credit.

2.5.2 Data for homework problems

For each question, calculate the mean, median, range, interquartile range, variance, standard deviation, coefficient of variation, standard error, and whether there are any "outliers".

Please also write your own short response to the *Synthesis question* posed, which will involve thinking about the data and metrics you just analyzed.

2.5.2.1 1: UNK Nebraskans

Ever year, the university keeps track of where students are from. The following are data on the umber of students admitted to UNK from the state of Nebraska:

```
# create dataset in R
nebraskans <- c(5056,5061,5276,5244,5209,
                5262,5466,5606,5508,5540,5614)

years <- 2023:2013

nebraskans_years <- cbind(years,nebraskans) |>
  as.data.frame()

nebraskans_years
```

```
   years nebraskans
1   2023       5056
2   2022       5061
3   2021       5276
4   2020       5244
5   2019       5209
6   2018       5262
7   2017       5466
8   2016       5606
9   2015       5508
10  2014       5540
11  2013       5614
```

Using these data, please calculate the mean, median, range, interquartile range, variance, standard deviation, coefficient of variation, standard error, and whether there are any "outliers" for the number of UNK students from Nebraska.

In order to do this, we will need to first select the column that denotes the number of Nebraskans from the dataset. Remember, we need to save this as an object in *R* to do the analyses. Here is a demonstration of getting the column to look at the mean, so that you can repeat this for the other questions. This relies heavily on the use of `$`, used to get the data from a specific column.

```
# $ method
nebraskans <- nebraskans_years$nebraskans

nebraskans
```

```
 [1] 5056 5061 5276 5244 5209 5262 5466 5606 5508 5540 5614
```

Now we can get the `mean` of this vector.

```
mean(nebraskans) |>
  round(0) # don't forget to round!
```

```
[1] 5349
```

Synthesis question: Do you think that there are any really anomalous years? Do you feel data are similar between years? *Note* we are not looking at trends through time but whether any years are outliers.

2.5.2.2 2: Piracy in the Gulf of Guinea

The following is a dataset looking at oceanic conditions and other variables associated with pirate attacks within the region between 2010 and 2021 (Moura et al. 2023). Using these data, please calculate the mean, median, range, interquartile range, variance, standard deviation, coefficient of variation, standard error, and whether there are any "outliers" for distance from shore for each pirate attack (column `Distance_from_Coast`).

```
pirates <- read_csv("https://figshare.com/ndownloader/files/42314604")
```

```
New names:
Rows: 595 Columns: 40
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(18): Period, Season, African_Season, Coastal_State, Coastal_Zone, Navi... dbl
(20): ...1, Unnamed: 0, Year, Month, Lat_D, Lon_D, Distance_from_Coast,... dttm
(2): Date_Time, Date
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `` -> `...1`
```

Synthesis question: Do you notice any patterns in distance from shore? What may be responsible for these patterns? *Hint*: Think about what piracy entails and also what other columns are available as other variables in the above dataset.

2.5.2.3 3: Patient ages at presentation

The following is a dataset on skin sores in different communities in Australia and Oceania, specifically looking at the amount of time that passes between skin infections (Lydeamore et al. 2020a). This file includes multiple different datasets, and focuses on data from children in the first five years of their life, on household visits, and on data collected during targeted studies (Lydeamore et al. 2020b).

```
ages <- read_csv("https://doi.org/10.1371/journal.pcbi.1007838.s006")
```

```
Rows: 17150 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): dataset
dbl (1): time_difference

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Let's see what this file is like real fast. We can use the command `dim` to see the `rows` and `columns`.

```
dim(ages)
```

```
[1] 17150     2
```

As you can see, this file has only two columns but 17,150 rows! For the column `time_difference`, please calculate the mean, median, range, interquartile range, variance, standard deviation, coefficient of variation, standard error, and whether there are any "outliers".

Synthesis question: Folks will often think about probabilities of events being "low but never zero". What does that mean in the context of these data? What about these data make you feel like probabilities may decrease through time but never become zero?

3 Diagnosing data visually

3.1 The importance of visual inspection

Inspecting data visually can give us a lot of information about whether data are normally distributed and about whether there are any major errors or issues with our dataset. It can also help us determine if data meet model assumptions, or if we need to use different tests more appropriate for our datasets.

3.2 Sample data and preparation

First, we need to load our *R* libraries.

```
library(curl)
```

```
Using libcurl 8.7.1 with LibreSSL/3.3.6
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter()     masks stats::filter()
x dplyr::lag()        masks stats::lag()
x readr::parse_date() masks curl::parse_date()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

Next, we can download our data sample.

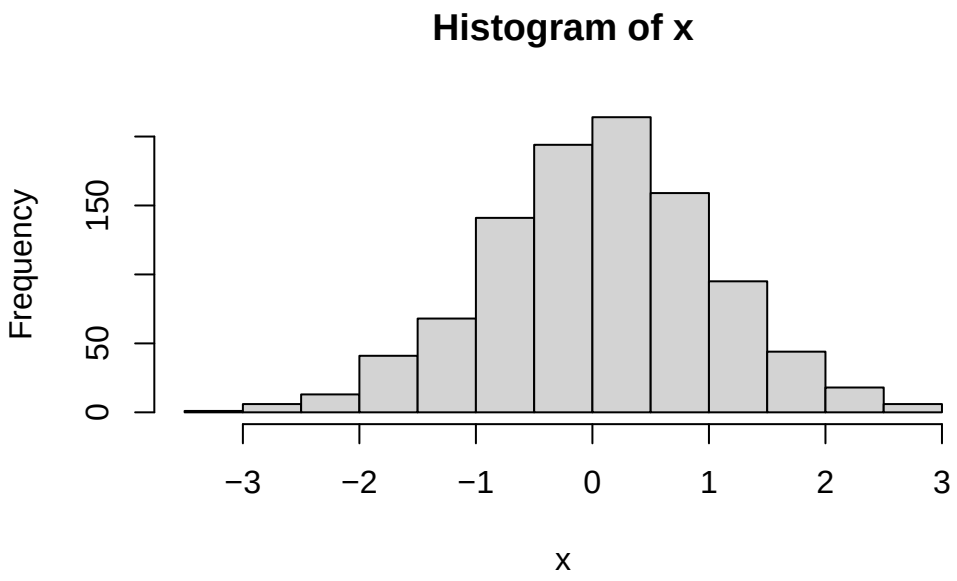
3.3 Histograms

A histogram is a frequency diagram that we can use to visually diagnose data and their distributions. We are going to examine a histogram using a random string of data. *R* can generate random (though, actually pseudorandom) strings of data on command, pulling them from different distributions. These distributions are pseudorandom because we can't actually program *R* to be random, so it starts from a wide variety of pseudorandom points.

3.3.1 Histograms on numeric vectors

The following is how to create default histograms on data. If you need to create custom bin sizes, please see the notes under *Cumulative frequency plots* for data that are not already in frequency format.

```
# create random string from normal distribution
# this step is not necessary for data analysis in homework
x <- rnorm(n = 1000, # 1000 values
           mean = 0,
           sd = 1)

# make histogram
hist(x)
```

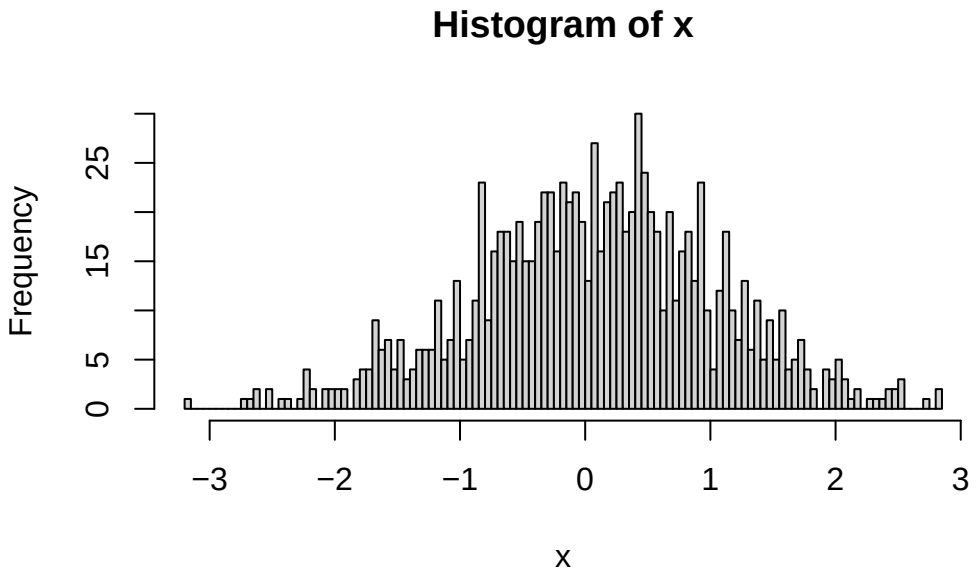


NOTE that a histogram can only be made on a vector of values. If you try to make a histogram on a data frame, *you will get an error and it will not work*. You have to specify which column

you wish to use with the `$` operator. (For example, for dataframe `xy` with columns `x` and `y`, you would use `hist(xy$y)`).

We can up the number of bins to see this better.

```
hist(x,breaks = 100)
```



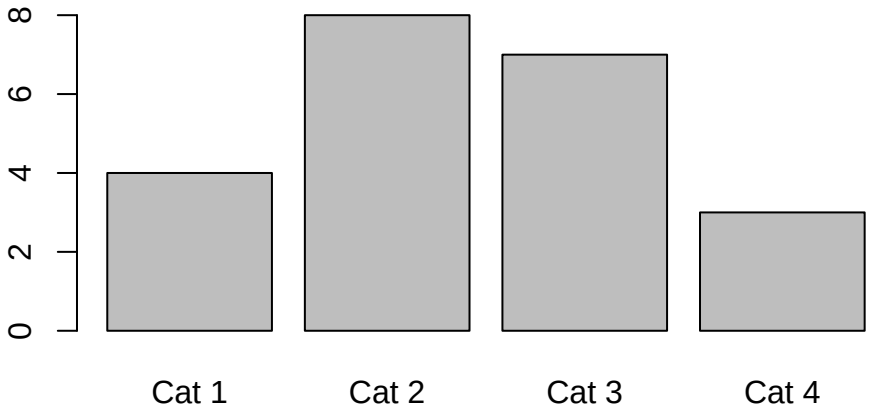
The number of bins can be somewhat arbitrary, but a value should be chosen based off of what illustrates the data well. *R* will auto-select a number of bins in some cases, but you can also select a number of bins. Some assignments will ask you to choose a specific number of bins as well.

3.3.2 Histograms on frequency counts

Say, for example, that we have a dataset where everything is already shown as frequencies. We can create a frequency histogram using `barplot`.

```
count_table <- matrix(nrow = 4, ncol = 2, byrow = T,
                      data = c("Cat 1", 4,
                               "Cat 2", 8,
                               "Cat 3", 7,
                               "Cat 4", 3)) |>
  as.data.frame()

colnames(count_table) <- c("Category","Count")
```

```r
# ensure counts are numeric data
count_table$Count <- as.numeric(count_table$Count)

# manually create histogram
barplot(count_table$Count, # response variable, counts for histogram
        axisnames = T, # make names on plot
        names.arg = count_table$Category) # make these the names
```



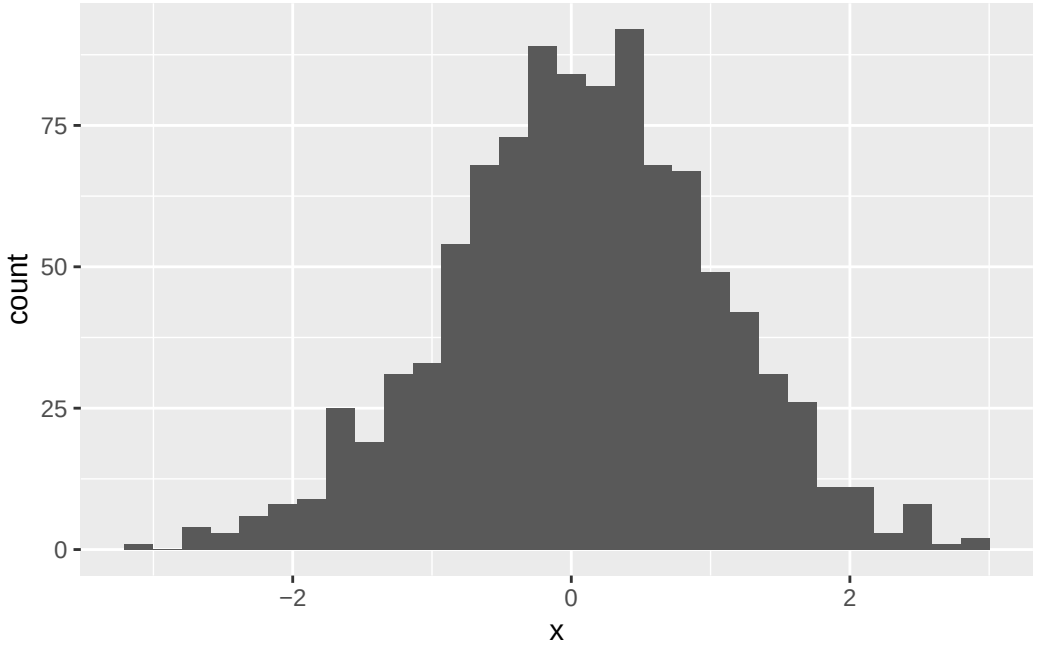
3.3.3 ggplot histograms

The following is an ***optional*** workthrough on how to make really fancy plots.

We can also use the program `ggplot`, part of the `tidyverse`, to create histograms.

```r
# ggplot requires data frames
x2 <- x |> as.data.frame()
colnames(x2) <- "x"

ggplot(data = x2, aes(x = x)) +
  geom_histogram()
```

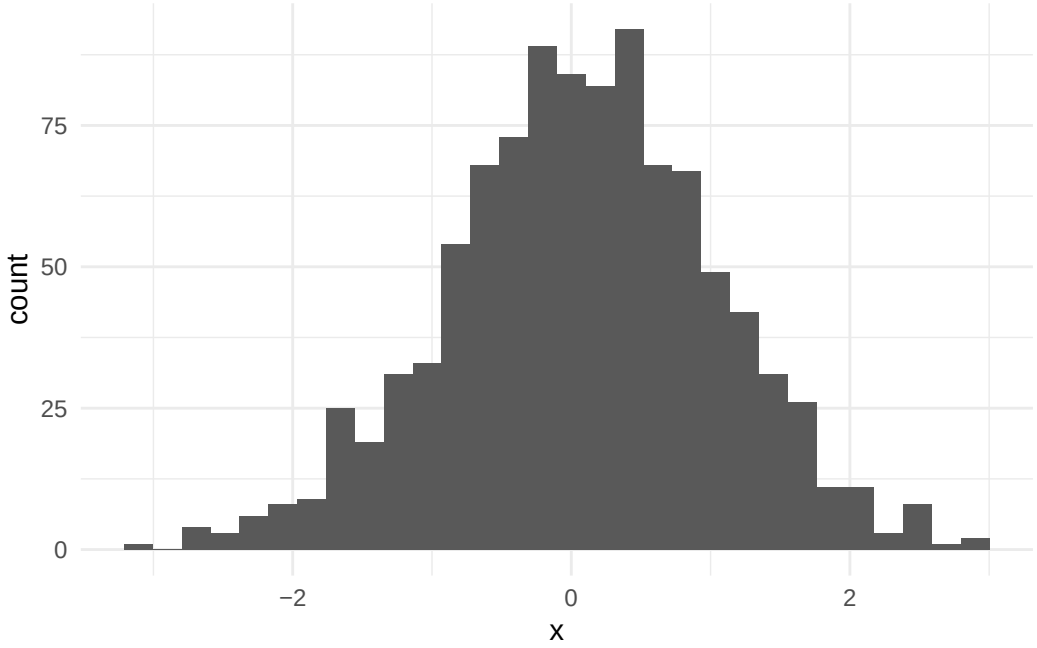
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



`ggplot` is nice because we can also clean up this graph a little.

```
ggplot(x2,aes(x=x)) + geom_histogram() +
  theme_minimal()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



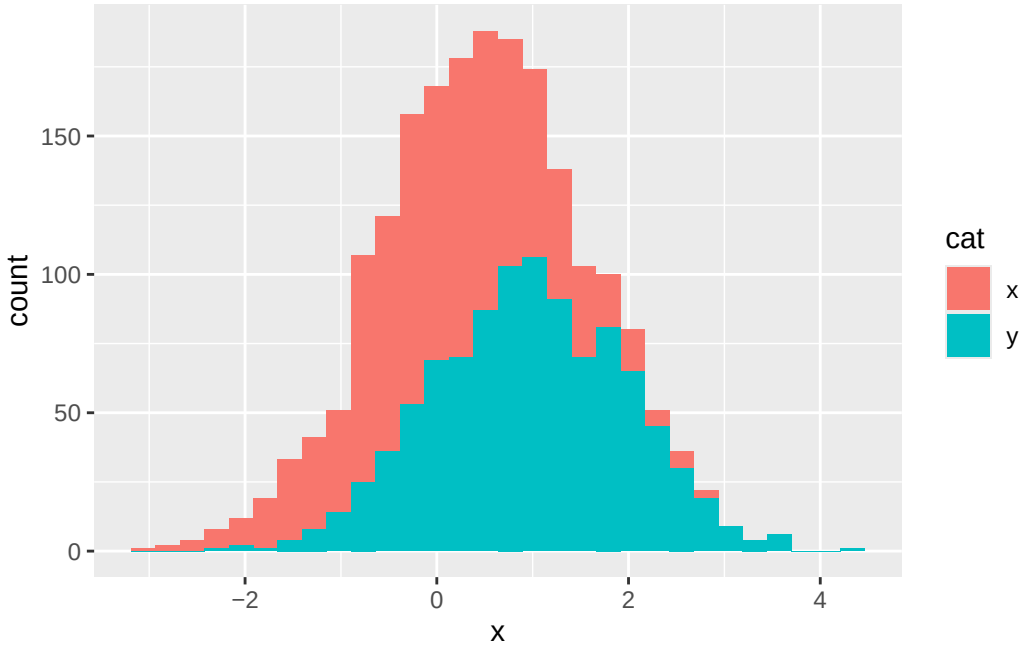
We can also do a histogram of multiple values at once in *R*.

```
x2$cat <- "x"

y <- rnorm(n = 1000,
           mean = 1,
           sd = 1) |>
  as.data.frame()

colnames(y) <- "x"
y$cat <- "y"

xy <- rbind(x2,y)

head(xy)
```

```
           x cat
1 -1.25952506   x
2  0.46794648   x
3 -0.04584308   x
4 -0.19237841   x
5 -0.44719363   x
6 -0.09679433   x
```

```
ggplot(xy,aes(x = x, fill = cat)) +
  geom_histogram()
```

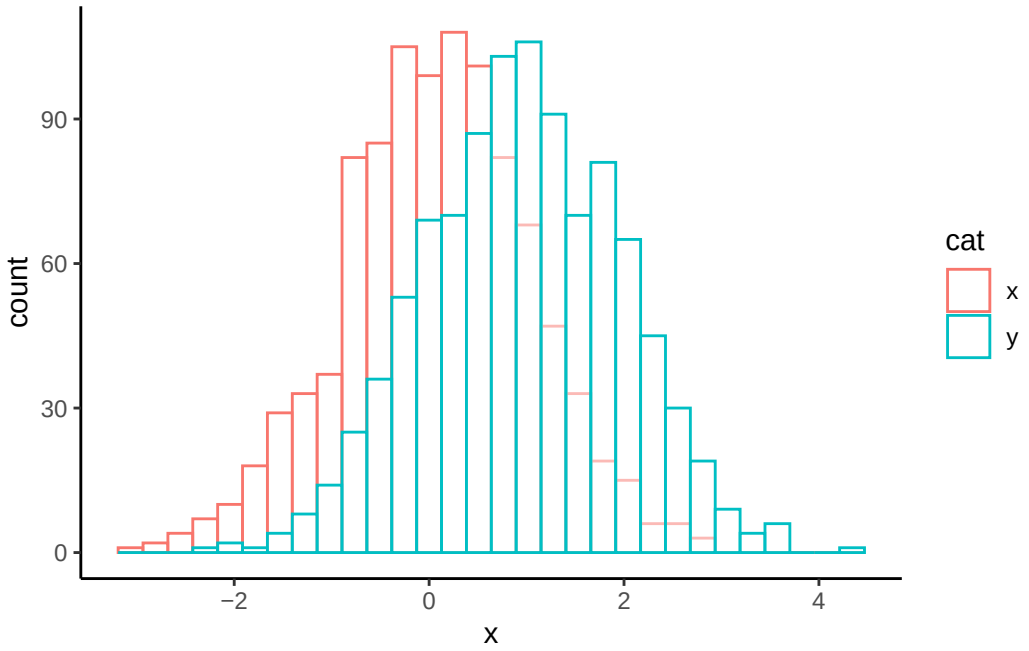
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can also make this look a little nicer.

```
ggplot(xy, aes(x = x, colour = cat)) +
  geom_histogram(fill = "white", alpha = 0.5, # transparency
                 position = "identity") +
  theme_classic()
```

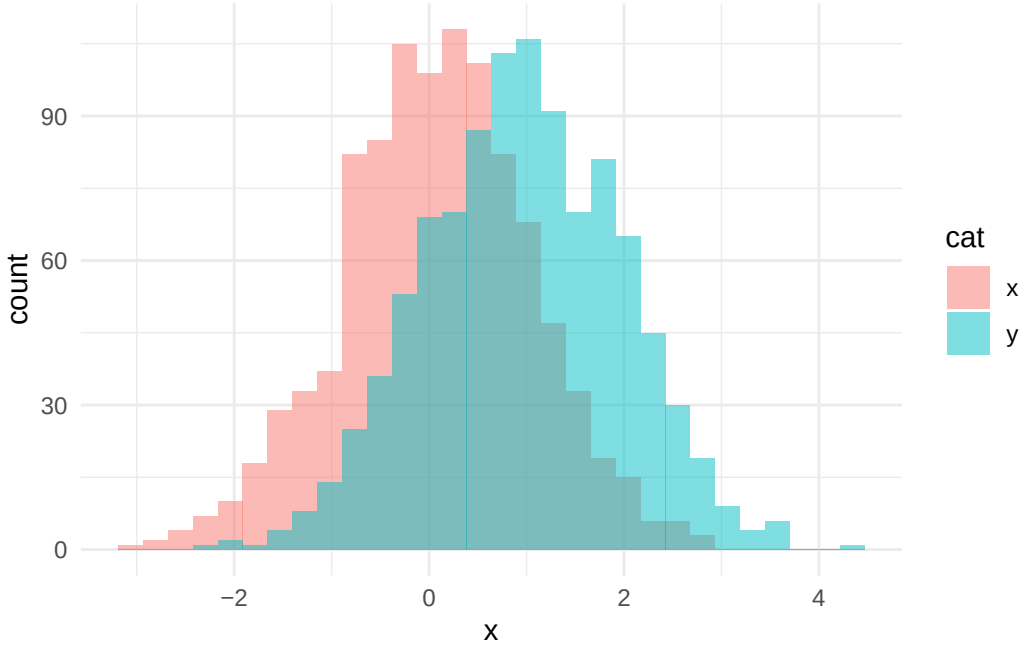
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can show these a little differently as well.

```
ggplot(xy, aes(x = x, fill = cat))+
  geom_histogram(position = "identity", alpha = 0.5) +
  theme_minimal()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

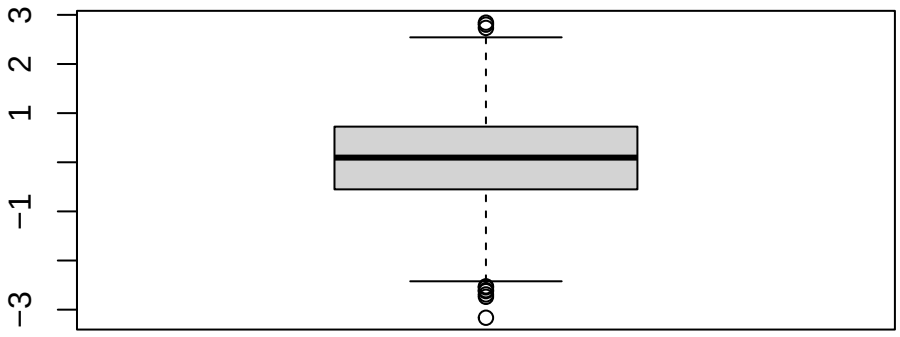


There are lots of other commands you can incorporate as well if you so choose; I recommend checking sites like this one.

3.4 Boxplots

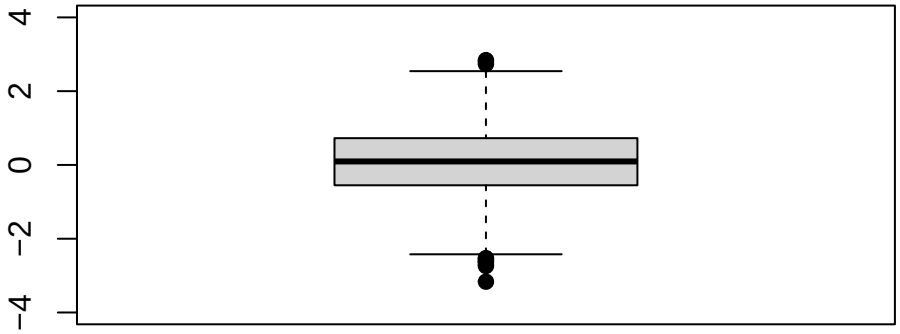
We can also create boxplots to visualize the spread of the data. Boxplots include a bar for the median, a box representing the interquartile range between the 25th and 75th percentiles, and whiskers that extend $1.5 \cdot IQR$ beyond the 25th and 75th percentiles. We can create a boxplot using the command `boxplot`.

```
# using pre-declared variable x

boxplot(x)
```



We can set the axis limits manually as well.

```
boxplot(x, # what to plot
        ylim = c(-4, 4), # set y limits
        pch = 19) # make dots solid
```



On the above plot, *outliers* for the dataset are shown as dots beyond the ends of the "whiskers".

3.5 Skewness

Skew is a measure of how much a dataset "leans" to the positive or negative directions (*i.e.*, to the "left" or to the "right"). To calculate skew, we are going to use the `moments` library.

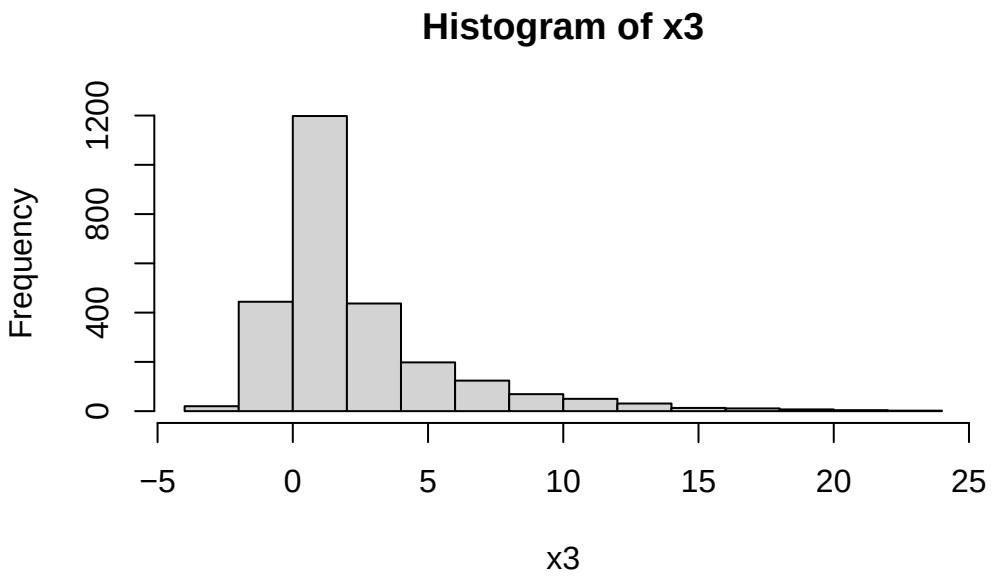
```
# don't forget to install if needed!
library(moments)

skewness(x)
```

```
[1] -0.09989024
```

Generally, a **value between** -1 **and** $+1$ **for skewness is "acceptable"** and not considered overly skewed. Positive values indicate "right" skew and negative values indicate a "left" skew. If something is too skewed, it may violate assumptions of normality and thus need *non-parametric* tests rather than our standard parametric tests - something we will cover later!

Let's look at a skewed dataset. We are going to artificially create a skewed dataset from our `x` vector.

```r
# create more positive values
x3 <- c(x,
        x[which(x > 0)]*2,
        x[which(x > 0)]*4,
        x[which(x > 0)]*8)

hist(x3)
```




```r
skewness(x3)
```

```
[1] 2.13453
```

As we can see, the above is a heavily skewed dataset with a positive ("right") skew.

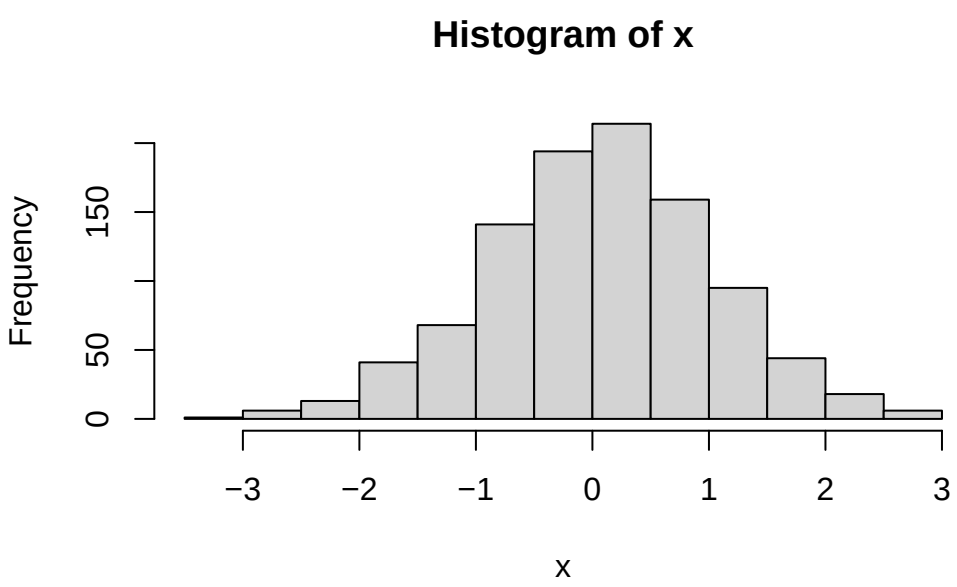
3.6 Kurtosis

Kurtosis refers to how sharp or shallow the peak of the distribution is (*platykurtic* vs. *leptokurtic*). Remember - *platy*kyrtic are *plateau*kurtic, wide and broad like a plateau, and *leptokurtic* distributions are sharp. Intermediate distributions that are roughly normal are *mesokurtic*.

Much like skewness, **kurtosis values of > 2 and < -2 are generally considered extreme**, and thus not mesokurtic. This threshold can vary a bit based on source, but for this class, we will use a threshold of ± 2 for both skewness and kurtosis.

Let's see the kurtosis of `x`. ***Note*** that when doing the equation, a normal distribution actually has a kurtosis of 3; **thus, we are doing kurtosis -3 to "zero" the distribution and make it comparable to skewness**.

```
hist(x)
```




```
# non-zeroed
kurtosis(x)
```

```
[1] 3.062539
```

```
# zeroed
kurtosis(x)-3
```

```
[1] 0.06253927
```

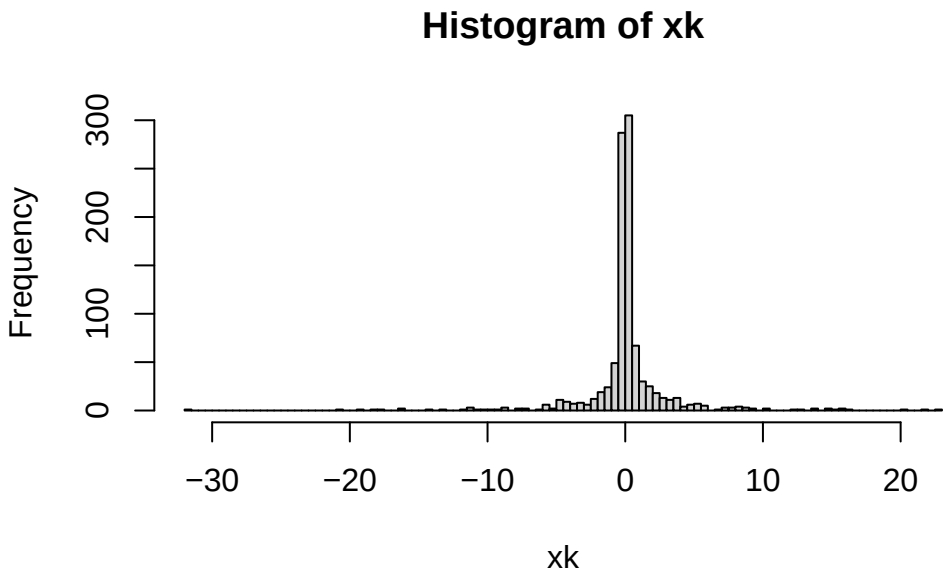
As expected, out values drawn from a normal distribution are not overly skewed. Let's compare these to a more kurtic distribution:

```
xk <- x^3

kurtosis(xk)-3
```

```
[1] 19.63253
```

What does this dataset look like?

```
hist(xk,breaks = 100)
```



As we can see, this is a very *leptokurtic* distribution.

3.7 Cumulative frequency plot

A *cumulative frequency plot* shows the overall spread of the data as a cumulative line over the entire dataset. This is another way to see the spread of the data and is often complementary to a histogram.

The following cumulative distribution plot is based on the method outlined in Geeks for Geeks.

3.7.1 If data are not in histogram/frequency format

You will need to create a frequency table to make them be in histogram format.

```
# declaring the break points
# make break points based on data
# always round UP with ceiling
range.x <- ceiling(max(x)-min(x))

range.x
```

```
[1] 7
```

```
range(x)
```

```
[1] -3.164889  2.841956
```

Based on this range, we need to create our bin sizes. We want our lowest bin to be below our lowest value, and our highest bin above our highest value. Here, I'm setting a step size of 1. **You will have to examine your data and determine the best break size for your datasets.**

```
# make bins based on range
# create sequential series
break_points <- seq(-3, # starting value, low
                    4, # end value, high
                    1) # increment size
```

```
# transforming the data
data_transform = cut(x, # your data!
                     break_points, # the breaks you defined
                     right=FALSE) # closed on the left for intervals
# creating the frequency table
freq_table = table(data_transform) # create a table

# printing the frequency table
print("Frequency Table")
```

```
[1] "Frequency Table"
```

```
print(freq_table)
```

```
data_transform
[-3,-2) [-2,-1)  [-1,0)   [0,1)   [1,2)   [2,3)   [3,4)
     19     109     335     373     139      24       0
```

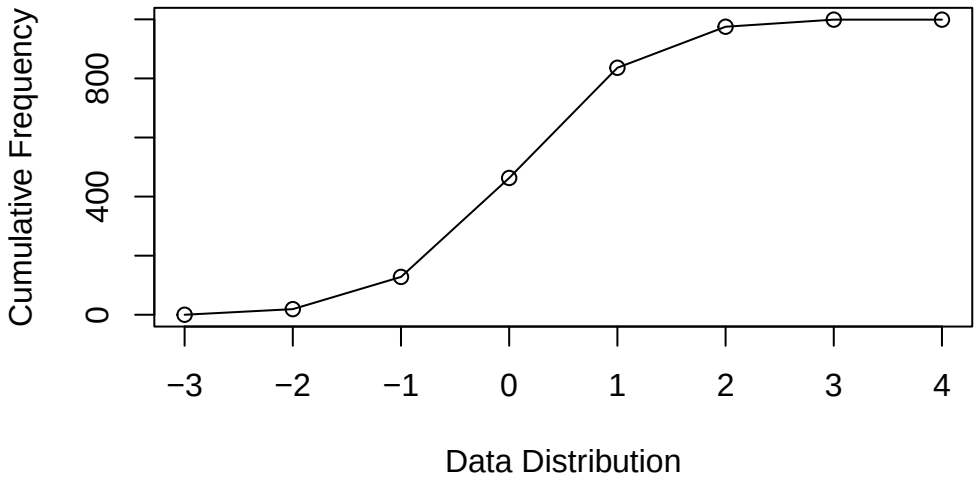
```
# calculating cumulative frequency
cumulative_freq = c(0, # start at 0, no points
                    cumsum(freq_table)) # get the cumulative frequency!
print("Cumulative Frequency")
```

```
[1] "Cumulative Frequency"
```

```
print(cumulative_freq)
```

```
        [-3,-2) [-2,-1)  [-1,0)   [0,1)   [1,2)   [2,3)   [3,4)
      0      19     128     463     836     975     999     999
```

```
# plotting the data
plot(break_points, # x axis is break_points
     cumulative_freq, # y axis is cumulative frequency
    xlab="Data Distribution", # x axis label
    ylab="Cumulative Frequency") # y axis label
# creating line graph
lines(break_points, # add lines to the graph, this is x
      cumulative_freq) # this is y for lines
```



3.7.2 If data are in histogram/frequency format

If you have a list of frequencies (say, *for river discharge over several years*), you only need to do the `cumsum` function. For example:

```
y <- c(1 ,2 ,4, 8, 16, 8, 4, 2, 1)

sum_y <- cumsum(y)

print(y)
```

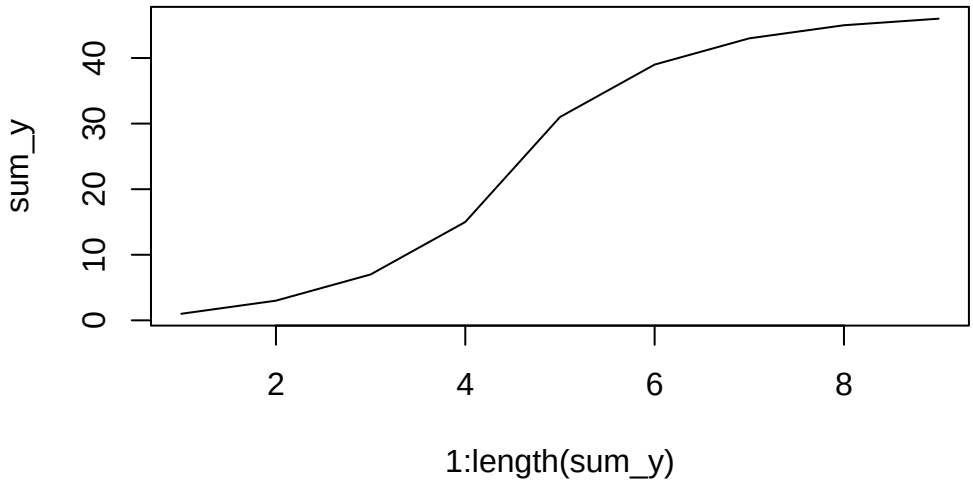
```
[1]  1  2  4  8 16  8  4  2  1
```

```
print(sum_y)
```

```
[1]  1  3  7 15 31 39 43 45 46
```

Now we can see we have out cumulative sums. Let's plot these. **NOTE** that this method will *not* have the x variables match the dataset you started with, it will only plot the curve based on the number of values given.

```
plot(x = 1:length(sum_y), # get length of sum_y, make x index
     y = sum_y, # plot cumulative sums
     type = "l") # make a line plot
```



3.8 Homework: Chapter 3

From your book, complete problems 3.1, 3.4 & 3.5. Data for these problems are available on *Canvas* and in your book.

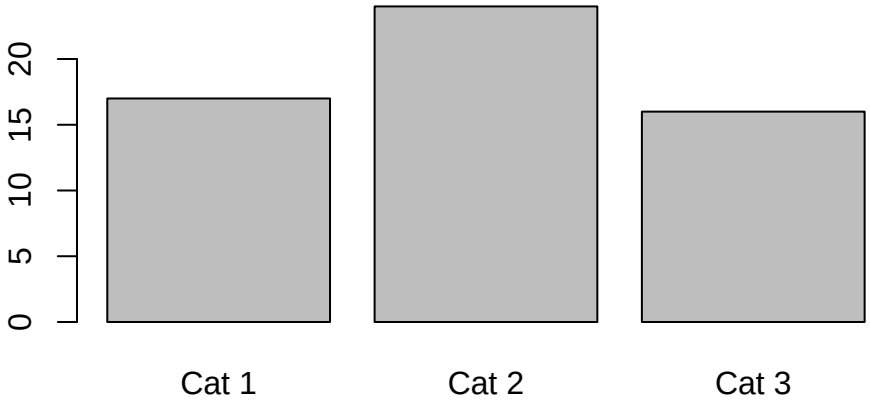
3.8.1 Helpful hint

HINT: For 3.5, consider just making a vector of the values of interest for a histogram.

For example, see the following. For reference:

- `c` means "concatenate", or place things together in an object.

```r
# numeric vector data for counts
y <- c(17,24,16)

# manually create a histogram using barplot
barplot(y,
        # axis names must be true
        axisnames = T,
        # input names here
        # each category as a separate quoted character string
        names.arg = c("Cat 1", "Cat 2", "Cat 3"))
```



3.8.2 Directions

Please complete all computer portions in an `rmarkdown` document knitted as an html. Upload any "by hand" calculations as images in the HTML or separately on *Canvas*.

3.9 Addendum

With thanks to Hernan Vargas & Riley Grieser for help in formatting this page. Additional comments provided by BIOL 305 classes.

4 Normality & hypothesis testing

4.1 Normal distributions

A *standard normal distribution* is a mathematical model that describes a commonly observed phenomenon in nature. When measuring many different kinds of datasets, the data being measured often becomes something that resembles a standard normal distribution. This distribution is described by the following equation:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

This equation is fairly well defined by the *variance* (σ^2), the overall spread of the data, and by the *standard deviation* (σ), which is defined by the square root of the variance.

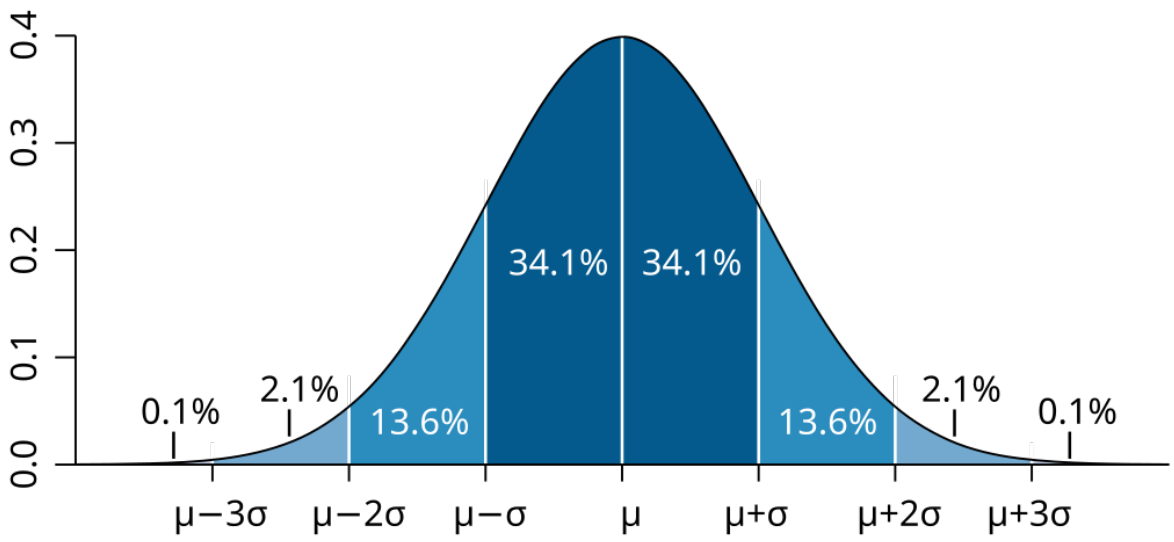


Figure 4.1: A standard normal distribution, illustrating the percentage of area found within each standard deviation away from the mean. By Ainali on Wikipedia; CC-BY-SA 3.0.

Standard normal distributions have a mean, median, and mode that are *equal*. The standard normal distribution is a *density function*, and we are interested in the "area under the curve" (AUC) to understand the relative probability of an event occurring. At the mean/median/mode, the probability on either side of the distribution is 50%. When looking at a normal distribution distribution, it is impossible to say the probability of a specific event occurring, but it is possible to state the probability of an event *as extreme or more extreme than the event observed* occurring. This is known as the p value.

4.1.1 Example in nature

In order to see an example of the normal distribution in nature, we are going to examine the BeeWalk survey database from the island of Great Britain (Comont 2020). We are not interested in the bee data at present, however, but in the climatic data from when the surveys were performed.

```
beewalk <- curl("https://figshare.com/ndownloader/files/44726902") |>
  read_csv()
```

```
Rows: 306550 Columns: 49
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (30): Website.ID, Website.RecordKey, SiteName, Site.section, ViceCounty,...
dbl (19): RecordKey, established, Precision, Transect.lat, Transect.long, tr...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Note that this is another massive dataset - 306,550 rows of data!

The dataset has the following columns:

```
colnames(beewalk)
```

```
 [1] "RecordKey"            "Website.ID"           "Website.RecordKey"
 [4] "SiteName"             "Site.section"         "ViceCounty"
 [7] "established"          "GridReference"        "Projection"
[10] "Precision"            "Transect.lat"         "Transect.long"
[13] "transect.OS1936.lat"  "Transect.OS1936.long" "transect_length"
[16] "section_length"       "section_grid_ref"     "H1"
[19] "H2"                   "H3"                   "H4"
[22] "habitat_description"  "L1"                   "L2"
```

```
[25] "land_use_description" "start_time"           "end_time"
[28] "sunshine"             "wind_speed"           "temperature"
[31] "TaxonVersionKey"      "species"              "latin"
[34] "queens"               "workers"              "males"
[37] "unknown"              "Comment"              "transect_comment"
[40] "flower_visited"       "StartDate"            "EndDate"
[43] "DateType"             "Year"                 "Month"
[46] "Day"                  "Sensitive"            "Week"
[49] "TotalCount"
```

We are specifically interested in `temperature` to determine weather conditions. Let's see what the mean of this variable is.

```
mean(beewalk$temperature)
```

```
[1] NA
```

Hmmm… we are getting an `NA` value, indicating that not every cell has data recorded. Let's view `summary`.

```
summary(beewalk$temperature)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00   16.00   19.00   18.65   21.00   35.00   16151
```

As we can see, 16,151 rows do not have temperature recorded! We want to remove these `NA` rows, which we can do by using using `na.omit`.

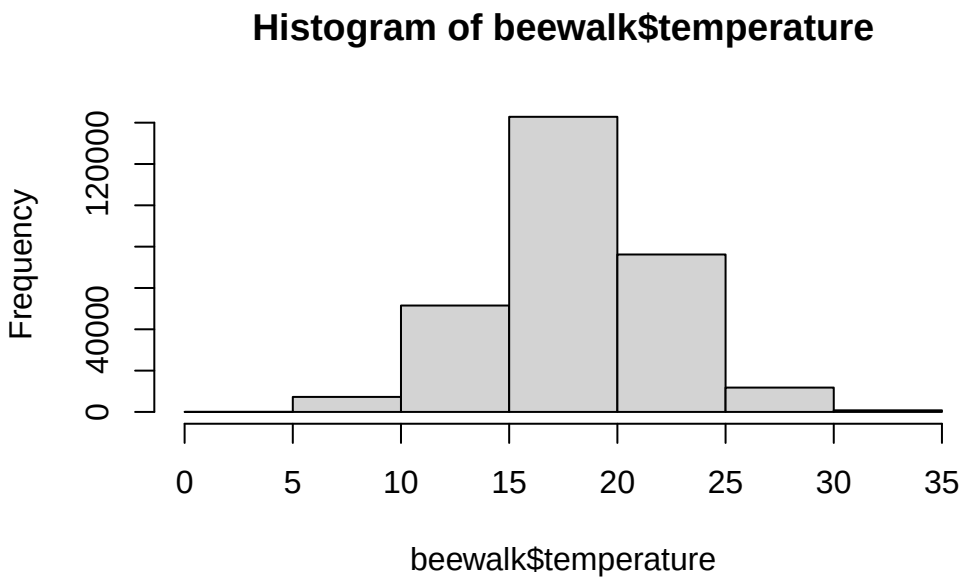
```
beewalk$temperature |>
  na.omit() |>
  mean() |>
  round(2) # don't forget to round!
```

```
[1] 18.65
```

Now we can record the mean.

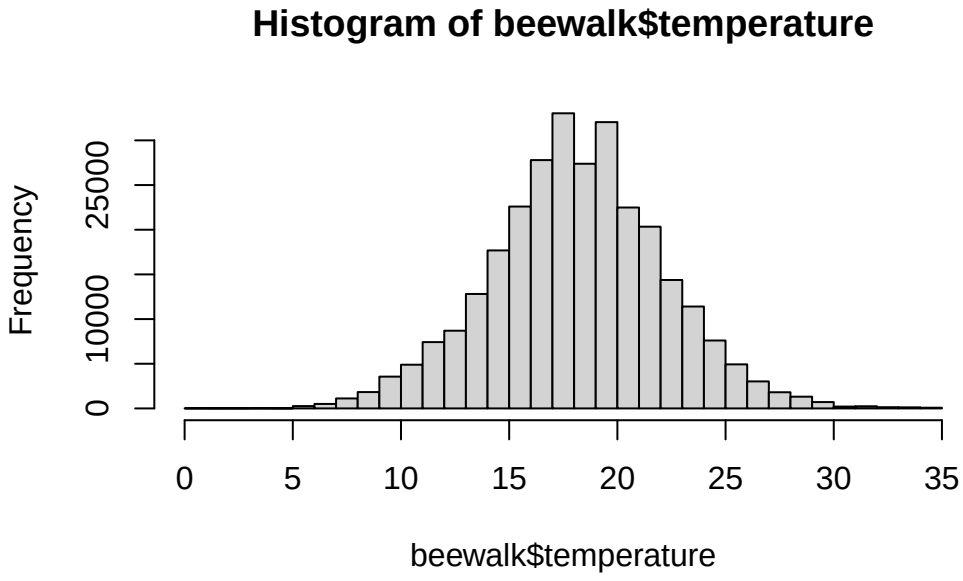
Let's visualize these data using a histogram. *Note* I do not use `na.omit` as the `hist` function automatically performs this data-cleaning step!

```
hist(beewalk$temperature,breaks = 5)
```



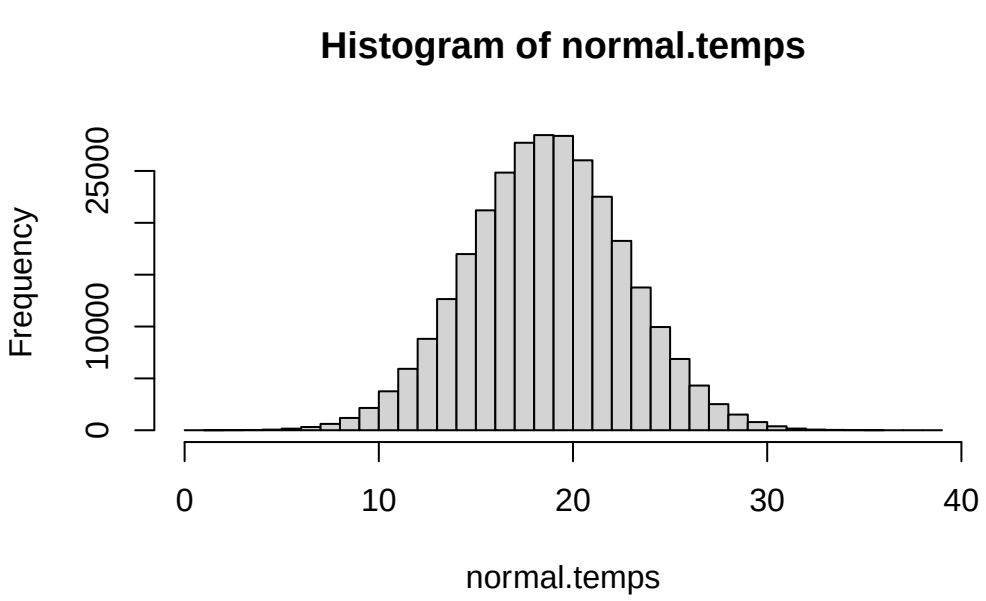
Even with only five breaks, we can see an interesting, normal-esque distribution in the data. Let's refine the bin number.

```
hist(beewalk$temperature,breaks = 40)
```



With forty breaks, the pattern becomes even more clear. Let's see what a *standard normal distribution* around these data would look like.

```r
# save temperature vector without NA values
temps <- beewalk$temperature |> na.omit()

mu <- mean(temps)
t.sd <- sd(temps)

# sample random values
normal.temps <- rnorm(length(temps), # sample same size vector
                      mean = mu,
                      sd = t.sd)

hist(normal.temps, breaks = 40)
```



As we can see, our normal approximation of temperatures is not too dissimilar from the distribution of temperatures we actually see!

Let's see what kind of data we have for temperatures:

```r
# load moments library
library(moments)

skewness(temps)
```

```
[1] 0.02393257
```

Data do not have any significant skew.

```
kurtosis(temps)-3
```

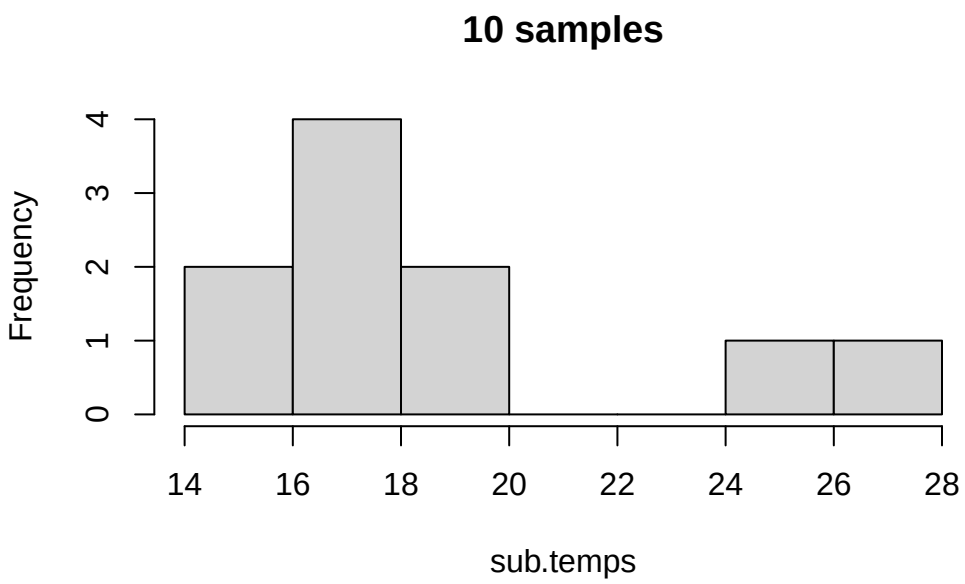
```
[1] 0.3179243
```

Data do not show any significant kurtosis.

4.1.2 Effect of sampling

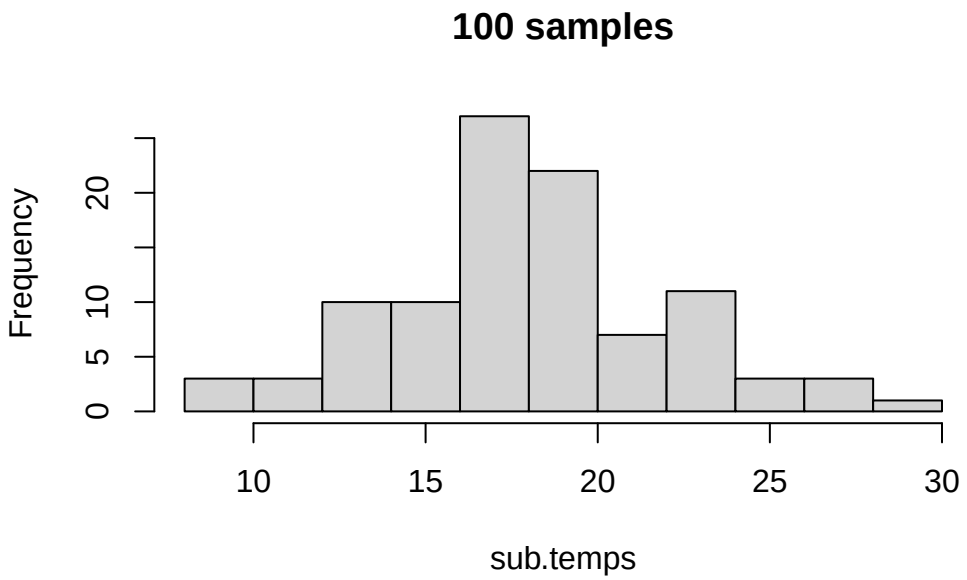
Oftentimes, we will see things approach the normal distribution as we collect more samples. We can model this by subsampling our temperature vector.

```
# make reproducible
set.seed(1839)

sub.temps <- sample(temps,
                    size = 10,
                    replace = FALSE)

hist(sub.temps, main = "10 samples")
```



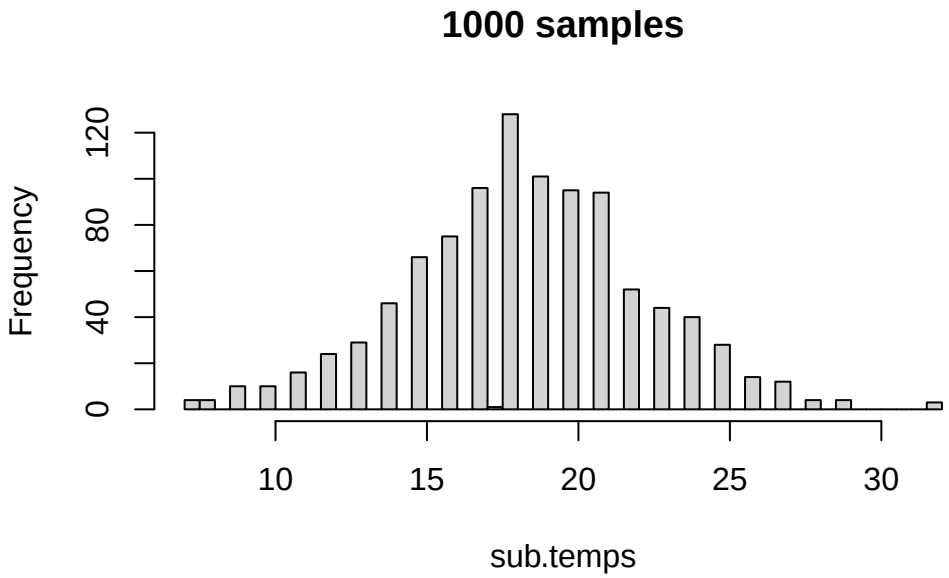
With only ten values sampled, we do not have much of a normal distribution. Let's up this to 100 samples.

```r
sub.temps <- sample(temps,
                    size = 100,
                    replace = FALSE)

hist(sub.temps, main = "100 samples",breaks = 10)
```



Now we are starting to see more of a normal distribution! Let's increase this to 1000 temperatures.

```r
sub.temps <- sample(temps,
                    size = 1000,
                    replace = FALSE)

hist(sub.temps, main = "1000 samples", breaks = 40)
```



Now the normal distribution is even more clear. As we can also see, the more we sample, the more we approach the true means and distribution of the actual dataset. Because of this, we can perform experiments and observations of small groups and subsamples and make inferences about the whole, given that most systems naturally approach statistical distributions like the normal!

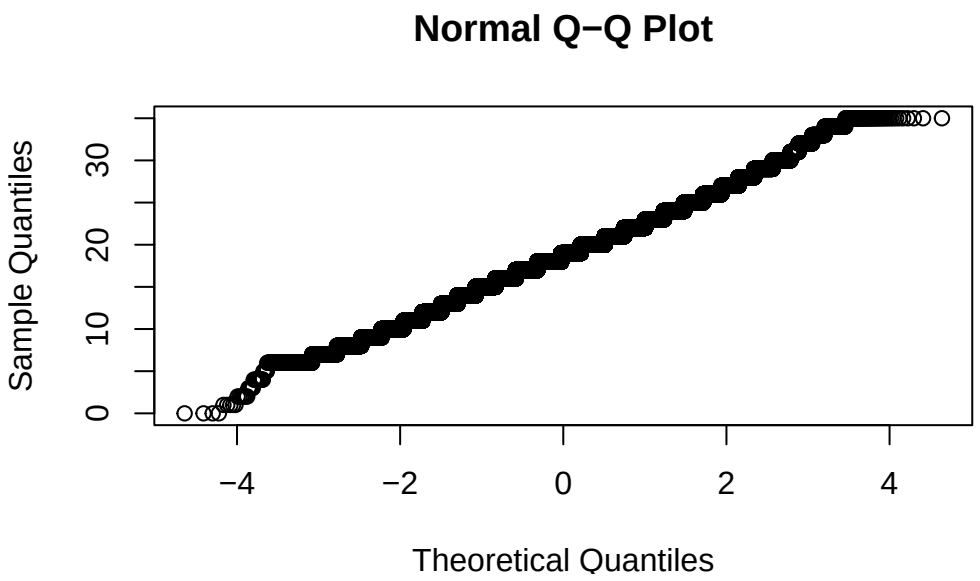
4.1.3 Testing if data are normal

There are two major methods we can use to see if data are normally distributed.

4.1.3.1 QQ Plots

Another way to see if data are normal is to use a QQ plot. These plots data quantiles to theoretical quantiles to see how well they align, with a perfectly normal distribution having a completely linear QQ plot. Let's look at these with our `beewalk` data.

```
qqnorm(temps)
```



As we can see above, the data are roughly linear, which means are data are very normal. The "stairsteps" are from the accuracy in measuring temperature, which was likely rounded and thus created a distribution that is not completely continuous.

4.1.3.2 Shapiro-Wilk test

Another way to test for normality is to use a Shapiro-Wilk test of normality. We will not get into the specific of this distribution, but this tests the null hypothesis that data originated in a normal distribution, with the alternative hypothesis that the data originated in a non-normal distribution. You will read next about the specifics of hypothesis testing, but this test uses an $\alpha = 0.05$, and we *reject the null hypothesis* if our $p < \alpha$, with p representing the probability of observing something as extreme or more extreme than the result we observe. Our `temps` dataset is too large, as `shapiro.test` requires vectors of < 5000. Let's take a random sample and try again.

```
sample(temps, 200) |>
  shapiro.test()
```

```
    Shapiro-Wilk normality test

data:  sample(temps, 200)
W = 0.98922, p-value = 0.1371
```

For our subsets of temperature, we find that the temperature is normally distributed. **Note** that having all 5000 temperatures included makes this test find a "non-normal" result, likely from the same stairstepping phenomenon from rounding that we discussed before, which may result in model overfitting to the rounded data.

4.2 Hypothesis testing

Since we can define specific areas under the curve within these distributions, we can look at the percentage of area within a certain bound to determine how likely a specific outcome would be. Thus, we can begin to test what the *probability of observing an event* is within a theoretical, probabilistic space. A couple of important conceptual ideas:

1. We may not be able to know the probability of a specific event, but we can figure out the probability of events more extreme or less extreme as that event.
2. If the most likely result is the mean, then the further we move away from the mean, the less likely an event becomes.
3. If we look *away* from the mean at a certain point, then the area represents the chances of getting a result *as extreme or more extreme than what we observe*. This probability is known as the p value.

Once we have a p value, we can make statements about the event that we've seen relative to the overall nature of the dataset, but we do not have sufficient information to declare if this result is *statistically significant*.

4.2.1 Critical Values - α

In order to determine if something is significant, we compare things to a *critical value*, known as α. This value is traditionally defined as 0.05, essentially stating that we deem an event as significant if 5% or fewer of observed or predicted events are as extreme or more extreme than what we observe.

Your value should always set your α critical value before you do your experiments and analyses.

Our critical value of α represents our criterion for *rejecting the null hypothesis*. We set our α to try to minimize the chances of error.

Type I Error is also known as a **false-positive**, and is when we **reject the null hypothesis when the null is true**.

Type II Error is also known as a **false-negative**, and is when we **support the null hypothesis when the null is false**.

By setting an α, we are creating a threshold of probability at which point we can say, with confidence, that results are different.

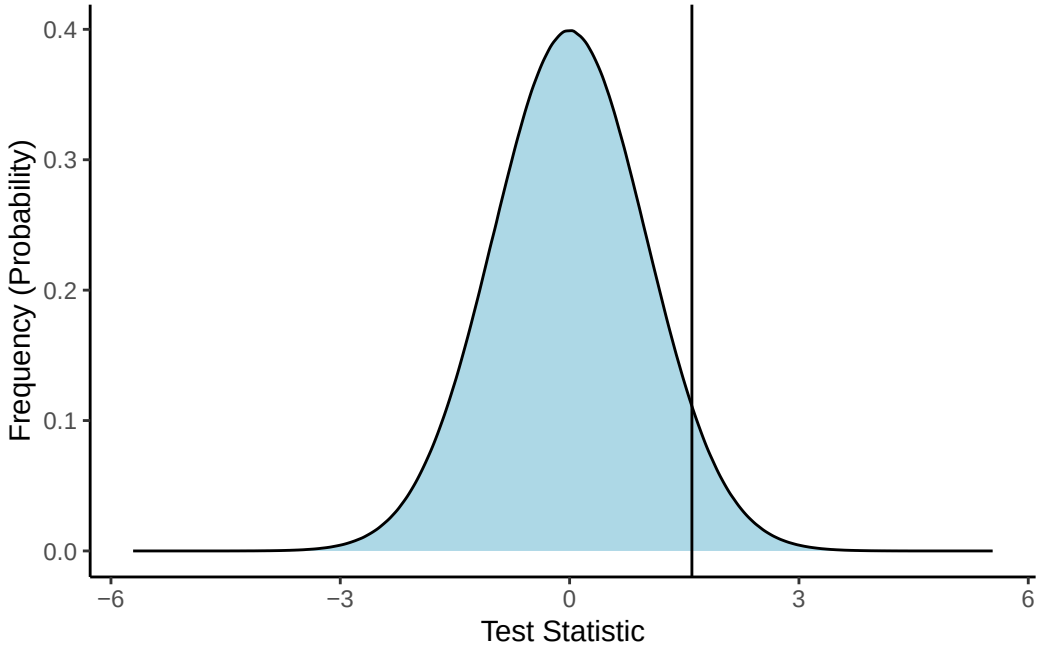
4.2.2 Introduction to p values

Let's say that we are looking at a dataset defined by a standard normal distribution with $\mu = 0$ and $\sigma = 1$. We draw a random value, x, with $x = 1.6$. What is the probability of drawing a number this extreme or more extreme from the dataset?

First, let's visualize this distribution:

```
###THIS WILL TAKE A WHILE TO RUN###

# create gigantic normal distribution dataset
# will be essentially normal for plotting
# rnorm gets random values
x <- rnorm(100000000)

# convert to data frame
x <- as.data.frame(x)
# rename column
colnames(x) <- c("values")

# thank you stack overflow for the following
# Creating density plot
p = ggplot(x,
           aes(x = values)
          ) +
  # generic density plot, no fill
  geom_density(fill="lightblue")

# Building shaded area
# create new plot object
p2  <-  p + # add previous step as a "backbone"
  # rename axes
  geom_vline(xintercept = 1.6) +
  xlab("Test Statistic") +
  ylab("Frequency (Probability)") +
  # make it neat and tidy
  theme_classic()

# plot it
```

```r
# can use ggsave function to save
plot(p2)
```

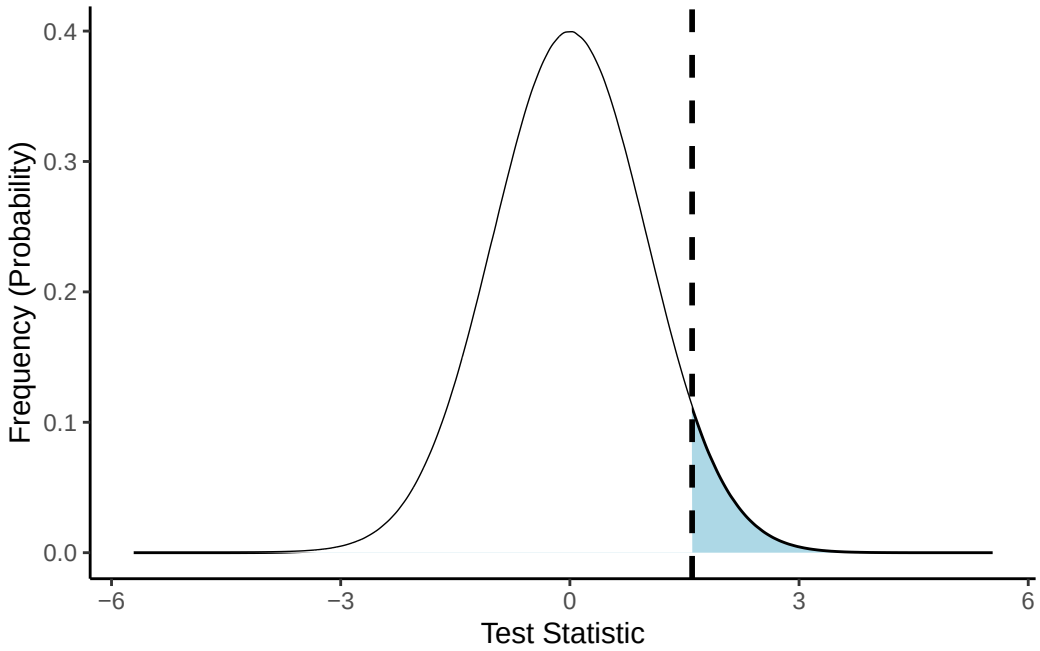


Above, the solid black line represents x, with the illustrated standard normal distribution being filled in blue.

Let's see how much of the area represents values *as extreme or more extreme* as our value x.

```r
### THIS WILL TAKE A WHILE TO RUN ###

# Getting the values of plot
# something I wasn't familiar with before making this!
d  <-  ggplot_build(p)$data[[1]]

# Building shaded area
# create new plot object
p2  <-  p + # add previous step as a "backbone"
  # add new shaded area
  geom_area(data = subset(d, x < 1.6), # select area
            # define color, shading intensity, etc.
            aes(x=x,y=y), fill = "white", alpha = 1) +
  # add value line
  geom_vline(xintercept = 1.6, colour = "black",
             linetype = "dashed",linewidth = 1) +
```

```
  # rename axes
  xlab("Test Statistic") +
  ylab("Frequency (Probability)") +
  # make it neat and tidy
  theme_classic()

# plot it
# can use ggsave function to save
plot(p2)
```



Now we can see that it is only a portion of the distribution *as extreme or more extreme* than the value we placed on the graph. The area of this region is our p value. This represents the *probability of an event as extreme or more extreme occurring* given the random variation observed in the dataset or in the distribution approximating the dataset. This is the value we compare to α - our threshold for rejecting the null hypothesis - to determine whether or not we are going to reject the null hypothesis.

Let's look at the above graph again, but let's visualize a two-tailed α around the mean with a 95% confidence interval. First, we need to get the Z scores for our $\alpha = 0.05$, which we calculate by taking $\frac{\alpha}{2}$ to account for the two tails. (Two tails essentially meaning we reject the null mean if we see things *greater than* or *less than* our expected value to a significant extent). We can calculate Z scores using `qnorm`.

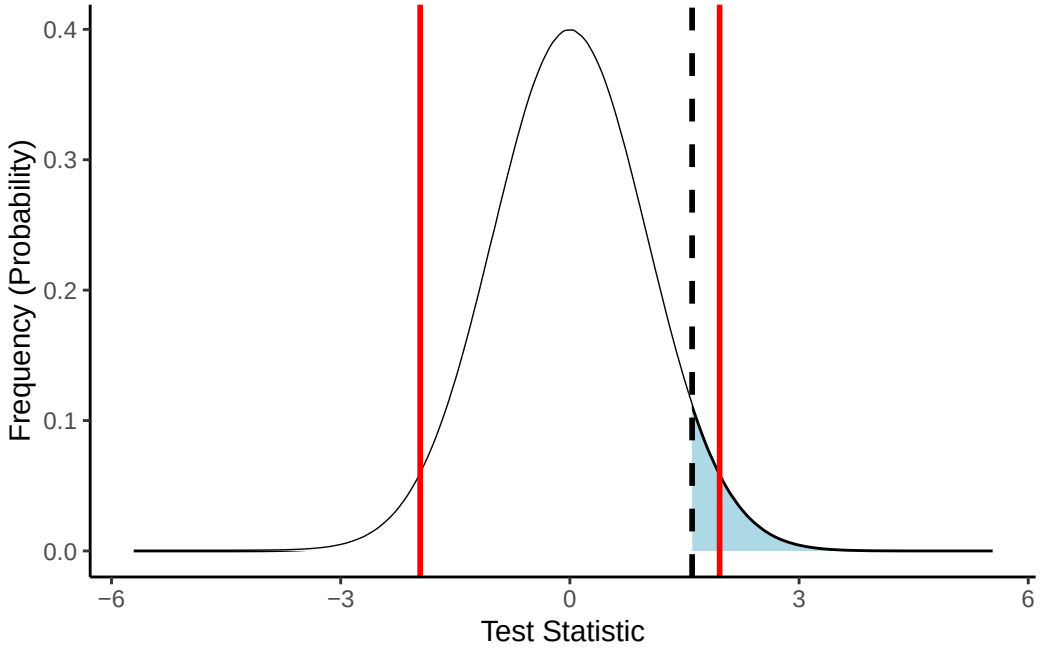
```r
low_alpha <- qnorm(0.025) # looks left
hi_alpha <- qnorm(0.975) # looks left

print(paste0("Our Z scores are: ",
             round(low_alpha,2),
             " & ",
             round(hi_alpha,2)))
```

```
[1] "Our Z scores are: -1.96 & 1.96"
```

The above values make sense, given the distribution is symmetrical. Our above dashed line is as $Z = 1.6$, which means we should have a $p = 0.05$, so the dashed line should be *closer* to the mean than our cutoffs.

```r
### THIS WILL TAKE A WHILE TO RUN ###

# Getting the values of plot
# something I wasn't familiar with before making this!
d  <-  ggplot_build(p)$data[[1]]

# Building shaded area
# create new plot object
p2  <-  p + # add previous step as a "backbone"
  # add new shaded area
  geom_area(data = subset(d, x < 1.6), # select area
            # define color, shading intensity, etc.
            aes(x=x,y=y), fill = "white", alpha = 1) +
  # add value line
  geom_vline(xintercept = 1.6, colour = "black",
             linetype = "dashed",linewidth = 1) +
  geom_vline(xintercept = low_alpha, colour = "red",
             linetype = "solid",linewidth = 1) +
  geom_vline(xintercept = hi_alpha, colour = "red",
             linetype = "solid",linewidth = 1) +
  # rename axes
  xlab("Test Statistic") +
  ylab("Frequency (Probability)") +
  # make it neat and tidy
  theme_classic()

# plot it
```

```
# can use ggsave function to save
plot(p2)
```



Exactly as we calculated, we see that our $p < \alpha$ and thus we do not see an area (in blue) less than the area that would be further than the mean as defined by the red lines.

4.2.3 Calculating a Z score

When we are trying to compare our data to a normal distribution, we need to calculate a Z score for us to perform the comparison. A Z score is essentially a measurement of the number of standard deviations we are away from the mean on a standard normal distribution. The equation for a Z score is:

$$Z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Where $\bar{x}$ is either a sample mean or a sample value, μ is the population mean, σ is the population standard deviation and n is the number of individuals in your sample (1 if comparing to a single value).

We can calculate this in R using the following function:

```r
zscore <- function(xbar, mu, sd.x, n = 1){
  z <- (xbar - mu)/(sd.x/sqrt(n))
  return(z)
}
```

NOTE that is the above isn't working for you, *you have a mistake somewhere in your code.* Try comparing - character by character - what is listed above to what you have.

Let's work through an example, where we have a sample mean of 62 with 5 samples compared to a sample mean of 65 with a standard deviation of 3.5.

```r
Z <- zscore(xbar = 62, # sample mean
            mu = 65, # population mean
            sd.x = 3.5, # population standard deviation
            n = 5) # number in sample

print(Z)
```

```
[1] -1.91663
```

Now, we can calculate the p value for this Z score.

```r
pnorm(Z)
```

```
[1] 0.0276425
```

After rounding, we get $p = 0.03$, a p that is significant if for a one-tailed $\alpha = 0.05$ but insignificant for a two-tailed $\alpha = 0.05$.

4.2.4 Calculated the p value

We have two different methods for calculating a p value:

4.2.4.1 Comparing to a z table

We can compare the z value we calculate to a z table, such as the one at ztable.net. On this webpage, you can scroll and find tables for positive and negative z scores. *Note* that normal distributions are symmetrical, so you can also transform from negative to positive to get an idea of the area as well. Given that a normal distribution is centered at 0, a z score of 0 will have a p value of 0.50.

On the z tables, you will find the tenths place for your decimal in the rows, and then go across to the columns for the hundredths place. For example, go to the website and find the p value for a z score of -1.33. You should find the cell marked 0.09176. *Note* the website uses naked decimals, which we do not use in this class.

For values that aren't on the z table, we can approximate its position between different points on the z table or, if it is extremely unlikely, denote that $p < 0.0001$.

4.2.4.2 Using *R*

In *R*, we can calculate a p value using the function `pnorm`. This function uses the arguments of `p` for our p value, `mean` for the mean of our distribution, `sd` for the standard deviation of our distribution, and also information on whether we want to log-transform p or if we are testing a specific hypothesis (lower tail, upper tail, or two-tailed). The function `pnorm` defaults to a standard normal distribution, which would be a z score, but it can also perform the z transformations for us if we define the mean and standard deviation.

For example, if we have a z score of -1.33:

```
pnorm(-1.33)
```

```
[1] 0.09175914
```

As we can see, we get the same result as our z table, just with more precision!

There are other functions in this family as well in *R,* including `dnorm` for quantiles, `qnorm` for determining the z score for a specific p value, and `rnorm` for getting random values from a normal distribution with specific dimensions. For now, we will focus on `pnorm`.

4.2.5 Workthrough Example

When this class was being designed, Hurricane Milton was about to make contact with Florida. Hurricane Milton is considered one of the strongest hurricanes of all time, so we can look at historical hurricane data to determine just how powerful this storm really was. We can get information on maximum wind speeds of all recorded Atlantic hurricanes as of 2024 from Wikipedia.

```
hurricanes <- read_csv("https://raw.githubusercontent.com/jacobccooper/biol305_unk/main/assig
```

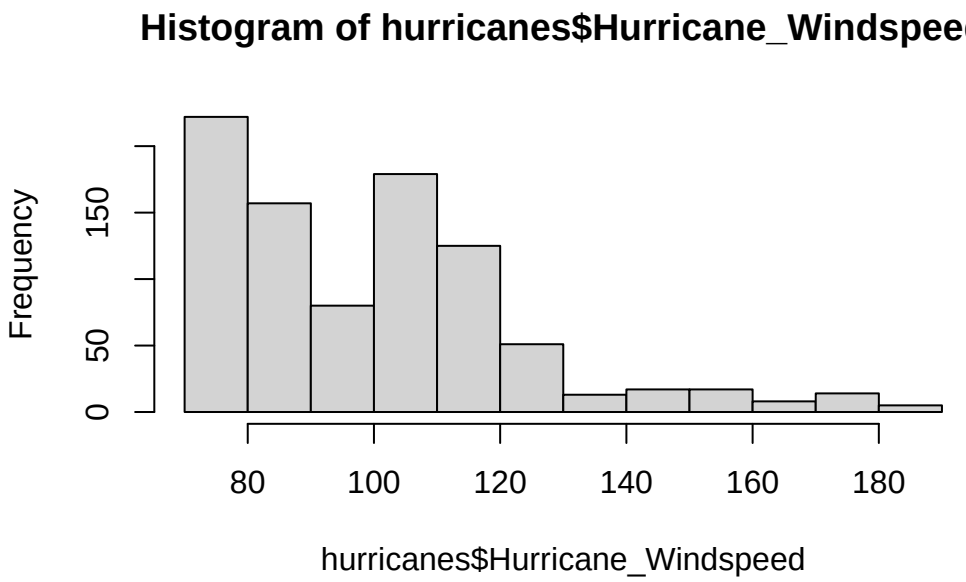
```
Rows: 889 Columns: 1
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl (1): Hurricane_Windspeed

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Above, we have loaded a `.csv` of one column that has all the hurricane speeds up to 2024. Hurricane Milton is the last row - the most recent hurricane. Let's separate this one out. We will use `[ , ]`, which defines `[rows,columns]` to subset data.

```
milton <- hurricanes$Hurricane_Windspeed[nrow(hurricanes)]

other_hurricanes <- hurricanes[-nrow(hurricanes),]
```

We want to compare the windspeed of Milton (180 mph) to the overall distribution of hurricane speeds. We can visualize this at first.

```
# all hurricanes
hist(hurricanes$Hurricane_Windspeed)
```



Windspeeds are more towards the lower end of the distribution, with strong storms being rarer.

For the sake of this class, we will assume we can use a normal distribution for these data, but if we were doing an official study we would likely need to use a non-parametric test (we will cover these later, but they cover non-normal data).

```
mu <- mean(other_hurricanes$Hurricane_Windspeed)

mu
```

```
[1] NA
```

Hmmm… we need to use `na.omit` to be sure we do this properly.

```
other_hurricanes_windspeed <- na.omit(other_hurricanes$Hurricane_Windspeed)

mu <- mean(other_hurricanes_windspeed)
mean(other_hurricanes_windspeed)
```

```
[1] 102.5254
```

Next, we need the standard deviation.

```r
sd.hurricane <- sd(other_hurricanes_windspeed)

sd.hurricane
```

```
[1] 23.08814
```

Now, we can calculate our Z value.

```r
Z <- (milton - mu)/sd.hurricane

Z
```

```
[1] 3.355603
```

How significant is this?

```r
pnorm(Z)
```

```
[1] 0.999604
```

This is greater than 0.5, so we need to do $1 - p$ to figure things out.

```r
1 - pnorm(Z)
```

```
[1] 0.000395961
```

This rounds to 0.0004, which means that this is an *extremely* strong hurricane.

4.2.5.1 Non-normality, for those curious

We can do a Shapiro-Wilk test of normality to see if this dataset is normal.

```r
shapiro.test(other_hurricanes_windspeed)
```

```
    Shapiro-Wilk normality test

data:  other_hurricanes_windspeed
W = 0.88947, p-value < 2.2e-16
```

A $p < 0.05$ indicates that these data are *non-normal*.

We can do a Wilcoxon-Test since these data are extremely non-normal.

```
wilcox.test(other_hurricanes_windspeed,
            milton)
```

```
    Wilcoxon rank sum test with continuity correction

data:  other_hurricanes_windspeed and milton
W = 6.5, p-value = 0.08678
alternative hypothesis: true location shift is not equal to 0
```

Using non-normal corrections, we find that this is *not* an extremely strong hurricane, but it is near the upper end of what we would consider "normal" under historical conditions. Still an extremely bad hurricane!

4.3 Confidence Intervals

Because we can figure out the probability of an event occurring, we can also calculate *confidence intervals*. A *confidence interval* provides a range of numbers around a value of interest that indicates where we believe the mean of a population lies and our confidence that it lies within that range. **Note** that nothing is ever 100% certain, but this helps us determine where a mean is and demonstrates our confidence in our results.

Specifically, if the tails of the distribution, our α, are 0.05, then we have an area of 0.95 around the mean where we do not reject results. Another perspective on this area is that we can say with 95% certainty that a mean is within a certain area, and that if values fall within that confidence area then we do not reject the null hypothesis that the means are equal.

We will cover several different ways to calculate confidence intervals, but for normal distributions, we use the following equation, with the 0.95 confidence interval shown as an example:

$$CI = \bar{x} \pm Z_{1-\frac{\alpha}{2}} \cdot \frac{\sigma}{\sqrt{n}}$$

This interval gives us an idea of where the mean should lie. For example, if we are looking at the aforemention `beewalk` temperature data, we can calulate a 0.95 confidence interval around the mean.

```r
temps <- na.omit(beewalk$temperature)

xbar <- mean(temps)
n <- length(temps)
sdtemp <- sd(temps)
# Z for 0.95 as P, so for 0.975, 0.025
# get value from P
Z <- qnorm(0.975)

CI <- Z*(sdtemp/sqrt(n))

CI
```

```
[1] 0.01458932
```

We have a very narrow confidence zone, because we have so many measurements. Let's round everything and present it in a good way.

If I want numbers to show up *in text* in RMarkdown, I can add code to a line of plain text using the following syntax:

```r
# DO NOT RUN
# Format in plaintext
`r xbar`
```

```
Typing that into the plaintext should render as the following: `r xbar`. Then I can also type my answer as follows:

The $95$% Confidence Interval for the mean for this temperature dataset is `r round(xbar,2)` $\pm$ `r round(CI,2)`.
```

Figure 4.2: This is the "coded" version of the text below. Compare the above window to the text below this image.

Typing that into the plain text should render as the following: 18.645963. Then I can also type my answer as follows:

The 95% Confidence Interval for the mean for this temperature dataset is:

```r
print(paste(round(xbar, 2),
      "+/-",
      round(CI, 2)))
```

```
[1] "18.65 +/- 0.01"
```

Note that the above is rounded to two decimal places to illustrative purposes ONLY, and should be rounded to one decimal place if it was a homework assignment because the original data has only one decimal place.

4.4 Homework: Chapter 8

Please complete problems 8.1, 8.2, 8.3 & 8.6. Follow the directions as written in the book. Submit one `html` file, as derived from *RStudio*. For maximum clarity, create headings to separate your problems. (Remember, a header can be invoked by placing '`#`' in front of a line of text. For example: the header here is written as `# Homework: Chapter 8`).

5 Exam 2 practice

5.1 Exam 2 Practice

Exam 2 practice is available on Canvas.

6 Probability distributions

6.1 Probability distributions

We rely on multiple different probability distributions to help us understand what probable outcomes are for a specific scenario. All of the tests that we are performing are comparing our results to what we would expect under perfectly random scenarios. For example, if we are flipping a coin, we are interested in whether the the observation we have of the flips on our coin matches our expectation given the probability of getting heads or tails on a perfectly fair coin. While it is possible to get all heads or all tails on a coin flip, it is highly unlikely and may lead us to believe we have an unfair coin. The more trails we perform, the more confident we can be that out coin is atypical.

We perform similar comparisons for other distributions. If we are comparing sets of events, we can look at the probability of those events occurring if events are occurring randomly. If we are comparing counts, we can compare our counts to our expectation of counts if events or subjects are distributed randomly throughout the matrix or whether two sets of counts are likely under the same sets of assumptions.

Remember, for our specific tests, we are setting an α value in advance (traditionally 0.05, or 5%) against which we compare our p value, with p representing the probability of observing an event *as extreme* or *more extreme* than the event we observe given a specific probability distribution.

Previously, we talked about the *normal distribution*, which is used to approximate a lot of datasets in nature. However, several other probability distributions are also useful for biological systems, which are outlined here.

6.2 Binomial distribution

A *binomial distribution* is one in which only two outcomes are possible - often coded as 0 and 1 and usually representing failure and success, respectively. The binomial is described by the following function:

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

where n = number of trials, x = the number of successes, and p = the probability of a success under random conditions.

In *R*, the binomial distribution is represented by the following functions:

- `dbinom`: the density of a binomial distribution
- `pbinom`: the distribution function, or the probability of a specific observation
- `qbinom`: the value at which a specific probability is found (the *quantile function*)
- `rbinom`: generates random values according to a binomial.

6.2.1 Binomial examples

Let's see what this looks like. Let's consider a scenario where we flip a coin 10 times and get 9 heads. How likely is this outcome?

```
x <- pbinom(q = 9, # number successes, 9 heads
            size = 10, # number of trials, 10 flips
            prob = 0.5) # probability with a fair coin

round(x,4)
```

```
[1] 0.999
```

NOTE that the trailing 0 is dropped, such that the real answer is 0.9990. However, we mentioned before that the p value should be the probability of a result as extreme or more extreme, meaning that it should *always* be less than 0.5. If we are reporting a value of *greater* than 0.5, then we are comparing to the upper tail of the distribution. For a one-tailed α of 0.05, this would mean that we are looking for a value *greater than* 0.95 $(1 - \alpha)$.

So, our real p is:

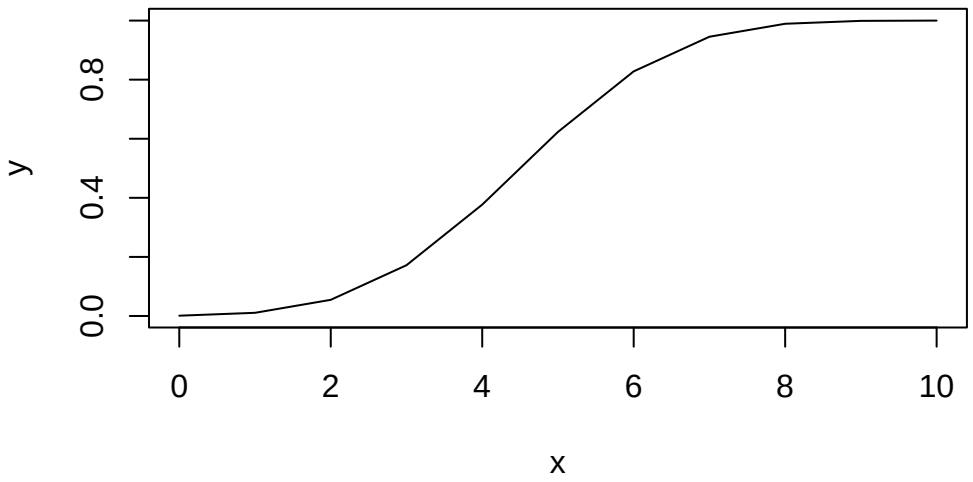
```
1 - round(x,4)
```

```
[1] 0.001
```

Again, the trailing zero is missing. Given that $p < \alpha$, we *reject the null hypothesis* that this is a fair coin.

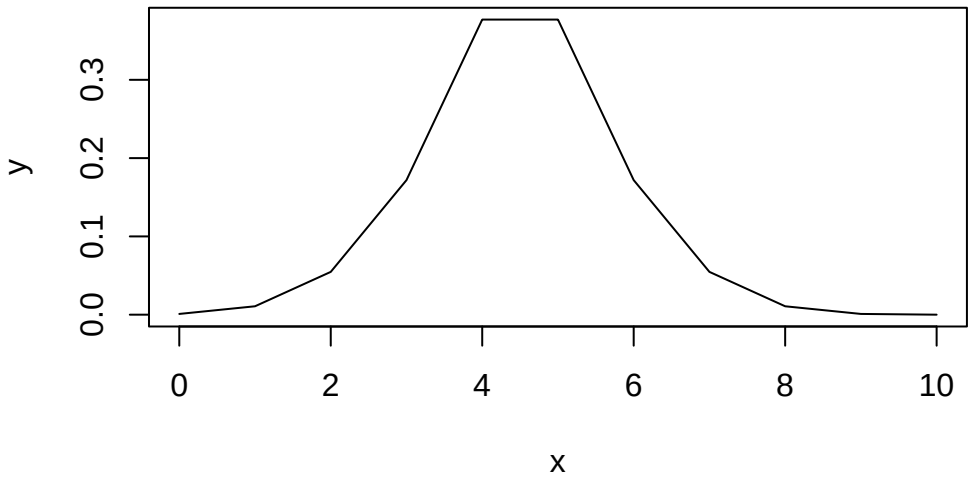
How does this distribution look?

```r
# number of successes
# start at 0 for no heads
x <- 0:10
# cumulative probability to left of outcome
y <- pbinom(x,
            size = 10,
            prob = 0.5,
            lower.tail = T)

# cumulative probability of results to the left
plot(x,
     y,
     type="l") # line plot
```



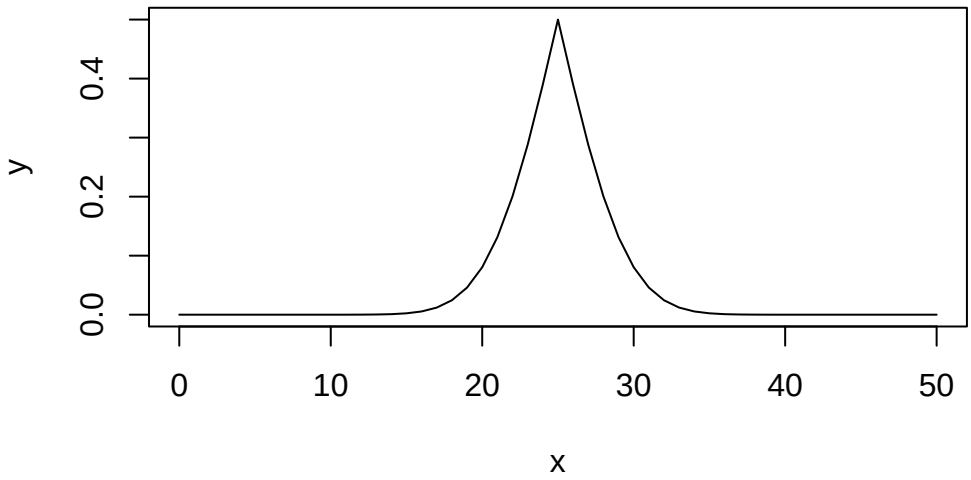
What about if we always have p less than 0.5 to reflect two tails?

```r
# any value greater than 0.5 is subtracted from 1
y[y > 0.5] <- 1 - y[y > 0.5]

plot(x,
     y,
     type="l")
```



What if we do this with a bigger dataset, like for 50 flips?

```
# number of successes
# start at 0 for no heads
x <- 0:50
# cumulative probability to left of outcome
y <- pbinom(x,
            size = length(x),
            prob = 0.5,
            lower.tail = T)

# any value greater than 0.5 is subtracted from 1
y[y > 0.5] <- 1 - y[y > 0.5]

plot(x,
     y,
     type="l")
```



As we increase the number of flips, we can see that the probability of success forms a *normal distribution* centered on the outcome given the default probability. Thus, as we deviate from our *expected outcome* (initial probability multiple by the number of trials), then our results become less likely.

6.2.2 Binomial exact tests

We can perform exact binomial tests by using the *R* function `binom.test`. This is a built in function within *R*. This test requires the following arguments:

- `x`: number of successes (success = outcome of interest)
- `n`: number of trials (number of events)
- `p`: probability of success in a typical situation (*i.e.*, for a fair coin, this is 50%)
- `alternative`: the hypothesis to be tested, whether `two.sided`, `greater`, or `less`.
- `conf.level` is the *confidence level* to be returned; default is 95%.

Let's say you flip a coin ten times, randomly assigning one side as a "success" and one side as a "failure". We do ten flips, and get 3 "successes". How likely is this outcome?

```
binom.test(x = 3,   # three successes
           n = 10,  # ten flips
           p = 0.5) # 50% chance on fair coin
```

```
    Exact binomial test
```

```
data:  3 and 10
number of successes = 3, number of trials = 10, p-value = 0.3438
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.06673951 0.65245285
sample estimates:
probability of success
                   0.3
```

Now let's say we do 1000 flips, and we get 300 successes.

```
binom.test(x = 300,
           n = 1000,
           p = 0.5)
```

```
    Exact binomial test

data:  300 and 1000
number of successes = 300, number of trials = 1000, p-value < 2.2e-16
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.2717211 0.3294617
sample estimates:
probability of success
                   0.3
```

As we can see, both of these return a confidence interval among other things. If we save the object, we can access these "slots" of data using the $ character.

```
binom_result <- binom.test(x = 3,
                           n = 10,
                           p = 0.5)

binom_result$p.value |> round(2)
```

```
[1] 0.34
```

```
binom_result$conf.int
```

```
[1] 0.06673951 0.65245285
attr(,"conf.level")
[1] 0.95
```

This test is easily implemented, but always double check and make sure you are setting it up correctly.

6.3 Poisson distribution

The *Poisson distribution* is used to reflect random count data. Specifically, the Poisson is used to determine if success events are *overdispersed* (*i.e.*, regularly spaced), *random*, or *underdispersed* (i.e., *clustered*). The Poisson introduces the variable *lambda* (λ) which represents the mean (μ) and the variance (σ^2), which are equal in a Poisson distribution. A Poisson distribution is described by the following function:

$$p(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

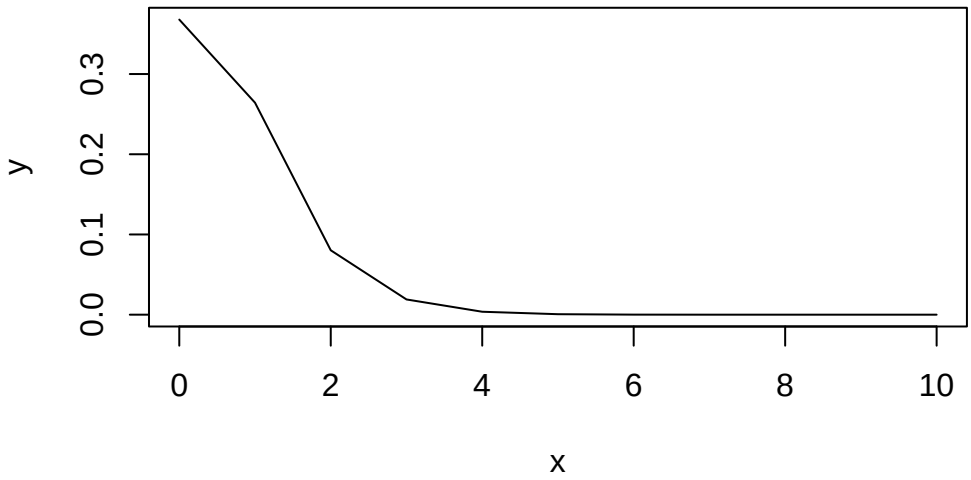
6.3.1 Poisson example

The Poisson is represented by the following functions in *R* which closely resemble the functions for the normal and binomial distributions:

- `dpois`: the log density function
- `ppois`: log distribution (probability) function
- `qpois`: quantile function
- `rpois`: random values from a Poisson.

Let's look at the probability of 0 to 10 successes when we have our $\lambda = 1$.

```
x <- 0:10
y <- ppois(x,lambda = 1)

# any value greater than 0.5 is subtracted from 1
y[y > 0.5] <- 1 - y[y > 0.5]

plot(x,y,type="l")
```



As we can see, the probability of rare events is high, whereas the probability quickly decreases as the number of successes increases.

6.3.2 Poisson test

Much like the Binomial Distribution and its `binom.test`, we can use `poisson.test` to analyze data via a Poisson Distribution. This command uses the arguments:

- `x`: number of events of interest
- `T`: time base (if for an event count)
- `r`: hypothesized rate or ratio
- `alternative` and `conf.level` are the same as for `binom.test`

We will not often use the `poisson.test` in this class, but it is good to be aware of.

6.4 Cumulative Probabilities

With both the Binomial and the Poisson, we can calculate cumulative probabilities. Both of these require the density (`d`) versions of the arguments.

6.4.1 Binomial cumulative

Let's say we have ten trials with three successes and a base probability of 0.5. We can calculate the probability to the *left* by using the following:

```
pbinom(q = 3,
       size = 10,
       prob = 0.5)
```

```
[1] 0.171875
```

As we can see, this is ~17.19%. Now let's try using `dbinom`. This command gives us the value at an individual bin, given that it is a more discrete distribution for these smaller sample sizes.

```
dbinom(x = 0:3,
       size = 10,
       prob = 0.5)
```

```
[1] 0.0009765625 0.0097656250 0.0439453125 0.1171875000
```

Above, we can see the probability of each number of successes three and fewer, for 0, 1, 2, and 3. Let's sum these probabilities.

```
dbinom(x = 0:3,
       size = 10,
       prob = 0.5) |>
  sum()
```

```
[1] 0.171875
```

As we can see, we get the same value as for `pbinom`! We can use this method for finding very specific answers, like what is the probability of getting between 3 and 6 successes in ten trials?

```
dbinom(x = 3:6,
       size = 10,
       prob = 0.5) |>
  sum() |>
  round(4)
```

```
[1] 0.7734
```

The probability of getting one of these outcomes is 77.34%.

6.4.2 Poisson cumulative probability

Likewise, we can use `ppois` to get the p value and `dpois` to get the distribution function of specific outcomes. So, let's say we have a scenario with a $\lambda = 0.5$ and we are looking at the probability of 2 successes or greater. In this case, we have an infinite series, which we can't calculate. However, we can calculate the probability of what it *isn't* and then subtract from 1. In this case, we are looking for the probability of *not* having 0 or 1 successes.

```
dpois(x = 0:1,
      lambda = 0.5)
```

```
[1] 0.6065307 0.3032653
```

Now, let's sum this and subtract it from 1.

```
1 - dpois(x = 0:1, lambda = 0.5) |>
  sum()
```

```
[1] 0.09020401
```

The probability is only about 9%.

6.5 Homework

6.5.1 Chapter 5

Complete problems 5.1, 5.2, 5.3, 5.5, 5.6, 5.11, 5.12, 5.13, 5.14, and 5.20 as written in your textbook.

Be sure to show your work, and submit your assignment as a knitted `html` document.

7 2 (Chi-squared) tests

7.1 χ^2-squared distribution

χ^2-squared (pronounced "*kai*", and spelled "*chi*") is a distribution used to understand if count data between different categories matches our expectation. For example, if we are looking at students in the class and comparing major vs. number of books read, we would expect *no association*, however we may find an association for a major such as English which required reading more literature. The χ^2 introduces a new term *degrees of freedom* (df) which reflects the number of individuals in the study. For many tests, df are needed to reflect how a distribution changes with respect the number of individuals (and amount of variation possible) within a dataset. The equation for the χ^2 is as follows, with the χ^2 being a special case of the *gamma* (γ or Γ) distribution that is affected by the df, which is defined as the number of rows minus one multiplied by the number of columns minus one $df = (rows - 1)(cols - 1)$:

$$f_n(x) = \frac{1}{2^{\frac{n}{2}}\Gamma(\frac{n}{2})} x^{\frac{n}{2-1}} e^{\frac{-x}{2}}$$

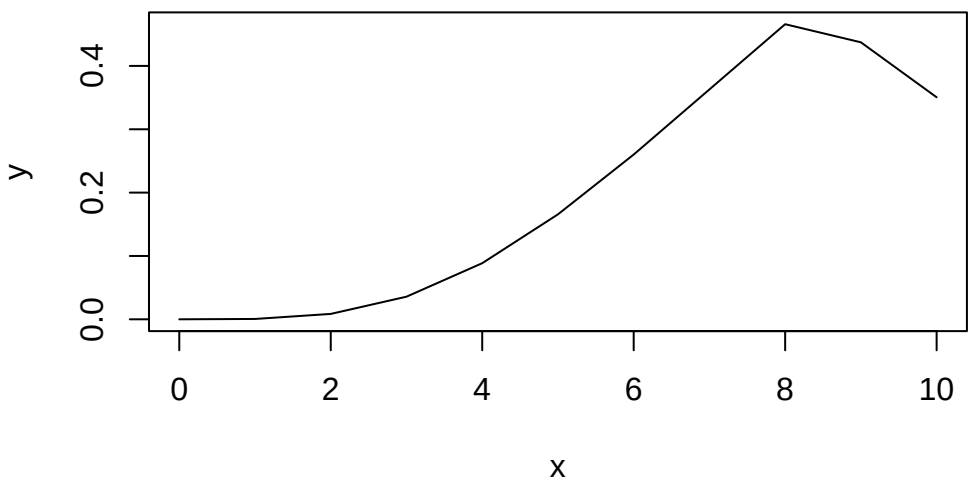
The χ^2-squared distribution is also represented by the following functions, which perform the same things as the previous outlined equivalents for Poisson and binomial:

- `dchisq`
- `pchisq`
- `qchisq`
- `rchisq`

We can view these probabilities as well:

```
x <- 0:10

y <- pchisq(x, df = 9)

# any value greater than 0.5 is subtracted from 1
y[y > 0.5] <- 1 - y[y > 0.5]
```

```r
plot(x,y,type="l")
```



7.1.1 Calculating the test statistic

We evaluate χ^2 tests by calculating a χ^2 value based on our data and comparing it to an expected χ^2 distribution. This *test statistic* can be evaluated by looking at a χ^2 table or by using *R*. **Note** that you need to know the degrees of freedom in order to properly evaluate a χ^2 test. We calculate our test statistic as follows:

$$\chi^2 = \Sigma \frac{(o-e)^2}{e}$$

where e = the number of expected individuals and o = the number of observed individuals in each category. Since we are squaring these values, we will only have positive values, and thus this will **always be a one-tailed test**.

There are multiple types of χ^2 test, including the following we will cover here:

- χ^2 Goodness-of-fit test
- χ^2 test of independence

7.1.2 χ^2 goodness-of-fit test

A χ^2 goodness-of-fit test looks at a vector of data, or counts in different categories, and asks if the observed frequencies vary from the expected frequencies.

7.1.2.1 χ^2 estimate by hand

Let's say, for example, we have the following dataset:

Hour	No. Drinks Sold
6-7	3
7-8	8
8-9	15
9-10	7
10-12	5
12-13	20
13-14	18
14-15	8
15-16	10
16-17	12

Now, we can ask if the *probability* of selling drinks is the same across all time periods.

```
drinks <- c(3, 8, 15, 7, 5, 20, 18, 8, 10, 12)
```

We can get the expected counts by assuming an equal probability for each time period; thus, $Exp(x) = \frac{N}{categories}$.

```
# sum all values for total number
N <- sum(drinks)

# get length of vector for categories
cats <- length(drinks)

# repeat calculation same number of times as length
exp_drinks <- rep(N/cats, cats)

exp_drinks
```

```
 [1] 10.6 10.6 10.6 10.6 10.6 10.6 10.6 10.6 10.6 10.6
```

Now, we can do out χ^2 calculation.

```r
chi_vals <- ((drinks - exp_drinks)^2)/exp_drinks

chi_vals
```

```
 [1] 5.44905660 0.63773585 1.82641509 1.22264151 2.95849057 8.33584906
 [7] 5.16603774 0.63773585 0.03396226 0.18490566
```

```r
sum(chi_vals)
```

```
[1] 26.45283
```

And now, to get the probability.

```r
chi_vals |>
  # get chi statistic
  sum() |>
  # get p value
  pchisq(df = length(chi_vals) - 1,
         # looking RIGHT
         lower.tail = F)
```

```
[1] 0.001721825
```

Here, we get $p = 0.002$, indicating that there is not an equal probability for selling drinks at different times of day.

7.1.2.2 χ^2 estimation by code

We can use the test `chisq.test` to perform this analysis as well.

```r
chisq.test(drinks)
```

```
    Chi-squared test for given probabilities

data:  drinks
X-squared = 26.453, df = 9, p-value = 0.001722
```

As we can see, these values are exactly the same as we just calculated by hand! ***Note*** that we can define the probability `p` if we want, otherwise it defaults to `p = rep(1/length(x), length(x))`.

7.1.3 χ^2 test of independence

Usually when we use a χ^2, we are looking at count data. Let's consider the following hypothetical scenario, comparing experience with *R* between non-biology majors (who, in this theoretical scenario, do not regularly use *R*) and Biology majors who are required to take *R* for this class:

Table 7.2: The above table of counts is also known as a **contingency table**. Intuitively, we can see a difference, but we want to perform a statistical test to see just how likely these counts would be if both groups were equally likely. We can calculate this both "by hand" and using built in *R* functions.

Major	*R* experience	No *R* experience
Non-biology	3	10
Biology	9	2

7.1.3.1 χ^2 estimations by hand

First, we can enter the data into *R*.

```
data <- matrix(data = c(3,10,9,2), nrow = 2, ncol = 2, byrow = T)

colnames(data) <- c("R", "No R")
rownames(data) <- c("Non-biology", "Biology")

data
```

```
            R No R
Non-biology 3   10
Biology     9    2
```

Next, we need the *observed* - *expected* values. We determine expected values either through probability ($0.5 \cdot n$ for equal probability for two categories) or via calculating the the expected values (see later section on *expected counts*). In this case, since we are looking at equally likely in each cell, we have an expected matrix as follows:

```
# total datapoints
N <- sum(data)

expected <- matrix(data = c(0.25*N,0.25*N,0.25*N,0.25*N), nrow = 2, ncol = 2, byrow = T)
```

```
colnames(expected) <- c("R", "No R")
rownames(expected) <- c("Non-biology", "Biology")

expected
```

```
            R No R
Non-biology 6    6
Biology     6    6
```

Now we need to find our *observed - expected*.

```
o_e <- data - expected

o_e
```

```
             R No R
Non-biology -3    4
Biology      3   -4
```

Note that in *R* we can add and subtract matrices, so there's no reason to reformat these data!

Now, we can square these data.

```
o_e2 <- o_e^2

o_e2
```

```
            R No R
Non-biology 9   16
Biology     9   16
```

Next, we take these and divide them by the expected values and then sum those values.

```
chi_matrix <- o_e2/expected

chi_matrix
```

```
              R     No R
Non-biology 1.5 2.666667
Biology     1.5 2.666667
```

```r
sum(chi_matrix)
```

```
[1] 8.333333
```

Here, we get a χ^2 value of 8.3333333. We can use our handy family functions to determine the probability of this event:

```r
chi_matrix |>
  # get chi statistic
  sum() |>
  pchisq(df = 1,
         # looking RIGHT
         lower.tail = F)
```

```
[1] 0.003892417
```

Here, we get a p value of:

```r
chi_matrix |>
  sum() |>
  pchisq(df = 1, lower.tail = F) |>
  round(3)
```

```
[1] 0.004
```

Alternatively, we can calculate this using *expected counts*. For many situations, we don't know what the baseline probability *should* be, so we calculate the expected counts based on what we do know. Expected counts are calculated as follows:

$$Exp(x) = \frac{\Sigma(row_x) \cdot \Sigma(col_x)}{N}$$

where N is the sum of all individuals in the table. For the above example, this would look like this:

```
data_colsums <- colSums(data)
data_rowsums <- rowSums(data)
N <- sum(data)

expected <- matrix(data = c(data_colsums[1]*data_rowsums[1],
                            data_colsums[2]*data_rowsums[1],
                            data_colsums[1]*data_rowsums[2],
                            data_colsums[2]*data_rowsums[2]),
                   nrow = 2, ncol = 2, byrow = T)

# divide by total number
expected <- expected/N

colnames(expected) <- colnames(data)
rownames(expected) <- rownames(data)

expected
```

```
             R No R
Non-biology 6.5  6.5
Biology     5.5  5.5
```

Here, we can see our expected number are not quite 50/50! this will give us a different result than our previous iteration.

```
e_o2 <- ((data - expected)^2)/expected

sum(e_o2)
```

```
[1] 8.223776
```

Now we have a χ^2 of:

```
e_o2 |>
  sum() |>
  round(2)
```

```
[1] 8.22
```

As we will see below, is the exact same value as we get for an uncorrected `chisq.test` from *R*'s default output.

7.1.3.2 χ^2 estimations in *R*

We can calculate this in R by entering in the entire table and using `chisq.test`.

```
data
```

```
            R No R
Non-biology 3   10
Biology     9    2
```

```
chi_data <- chisq.test(data, correct = F)

chi_data
```

```

    Pearson's Chi-squared test

data:  data
X-squared = 8.2238, df = 1, p-value = 0.004135
```

Here, we get the following for your p value:

```
chi_data$p.value |>
  round(3)
```

```
[1] 0.004
```

This is significant with $\alpha = 0.05$.

Note that these values are slightly different. they will be even more different if `correct` is set to `TRUE`. By default, R, uses a Yate's correction for continuity. This accounts for error introduced by comparing the discrete values to a continuous distribution.

```
chisq.test(data)
```

```

    Pearson's Chi-squared test with Yates' continuity correction

data:  data
X-squared = 6.042, df = 1, p-value = 0.01397
```

Applying this correction *lowers* the degrees of freedom, and increases the p value, thus making it harder to get $p < \alpha$.

Note that the Yate's correction is only applied for 2 x 2 contingency tables.

Given the slight differences in calculation between by hand and what the functions of R are performing, *it's important to always show your work.*

7.2 Fisher's exact test

χ^2 tests don't work in scenarios where we have very small count sizes, such as a count size of 1. For these situations with small sample sizes and very small count sizes, we use Fisher's exact test. This test gives us the p value directly - no need to use a table of any kind! Let's say we have a $2x2$ contingency table, as follows:

a	b
c	d

Where row totals are $a+b$ and $c+d$ and column totals are $a+c$ and $b+d$, and $n = a+b+c+d$. We can calculate p as follows:

$$p = \frac{(a+b)!(c+d)!(a+c)!(b+d)!}{a!b!c!d!n!}$$

In R, we can use the command `fisher.test` to perform these calculations. For example, we have the following contingency table, looking at the number of undergrads and graduate students in introductory and graduate level statistics courses:

	Undergrad	Graduate
Intro Stats	8	1
Grad Stats	3	5

```
stats_students = matrix(data = c(8,1,3,5),
                        byrow = T, ncol = 2, nrow = 2)

stats_students
```

```
     [,1] [,2]
[1,]    8    1
[2,]    3    5
```

```r
fisher.test(stats_students)
```

```
    Fisher's Exact Test for Count Data

data:  stats_students
p-value = 0.04977
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
   0.7934527 703.0167380
sample estimates:
odds ratio
  11.10917
```

In this situation, $p = 0.05$ when rounded, so we would fail to reject but note that this is borderline.

7.2.1 Chapter 7

Complete problems 7.1, 7.2, 7.3, 7.5, and 7.7 as written in your text books. For these problems, please also state your *null* and *alternative* hypotheses, as well as a conclusion as to whether you *support* or *reject* the null.

Next, do problems 7.9 and 7.11. For these two problems, pick which test is best and then perform said test. State your hypothesis and make a conclusion with respect to your p value.

Be sure to submit your homework as a knitted `html` document.

8 Testing means with t-tests

8.1 Introduction

Previously, we talked about normal distributions as a method for comparing samples to overall populations or comparing individuals to overall populations. However, sample sizes can introduce some error, and oftentimes we may not have access to an entire population. In these situations, we need a better test that can account for this changing error and the effect of different sample sizes. This is especially important when comparing two samples to each other. We may find a small sample from one population and a small sample for another, and we want to determine if these came from the same overall population as effectively as possible.

The distribution that we commonly refer to as a t-distribution is also sometimes known as a "Student's t-distribution" as it was first published by a man with the pseudonym of "Student". Student was in fact William Sealy Gossett, an employee of the Guinness corporation who was barred from publishing things by his employer to ensure that trade secrets were not made known to their competitors. Knowing that his work regarding statistics was important, Gossett opted to publish his research anyway under his pseudonym.

8.2 Dataset

For all of the examples on this page, we will be using a dataset on the morphology of canine teeth for identification of predators killing livestock (Courtenay 2019).

```
canines <- read_csv("https://figshare.com/ndownloader/files/15070175")
```

We want to set up some of these columns as "factors" to make it easier to process and parse in *R*. We will look at the column `OA` for these examples. Unfortunately, it is unclear what exactly `OA` stands for since this paper is not published at the present time.

```
canines$Sample <- as.factor(canines$Sample)

# we will be examining the column "OA"

canines$OA <- as.numeric(canines$OA)
```

```
summary(canines)
```

```
  Sample        WIS               WIM               WIB          
 Dog :34   Min.   :0.1323   Min.   :0.1020   Min.   :0.03402  
 Fox :41   1st Qu.:0.5274   1st Qu.:0.3184   1st Qu.:0.11271  
 Wolf:28   Median :1.1759   Median :0.6678   Median :0.25861  
           Mean   :1.6292   Mean   :1.0233   Mean   :0.44871  
           3rd Qu.:2.4822   3rd Qu.:1.5194   3rd Qu.:0.74075  
           Max.   :4.8575   Max.   :3.2423   Max.   :1.51721  
       D                 RDC               LDC                OA       
 Min.   :0.005485   Min.   :0.05739   Min.   :0.02905   Min.   :100.7  
 1st Qu.:0.034092   1st Qu.:0.28896   1st Qu.:0.22290   1st Qu.:139.2  
 Median :0.182371   Median :0.61777   Median :0.55985   Median :149.9  
 Mean   :0.250188   Mean   :0.88071   Mean   :0.84615   Mean   :148.4  
 3rd Qu.:0.361658   3rd Qu.:1.26417   3rd Qu.:1.26754   3rd Qu.:158.0  
 Max.   :1.697461   Max.   :3.02282   Max.   :3.20533   Max.   :171.5  
```

8.3 t-distribution

For these scenarios where we are testing a single sample mean from one or more samples we use a t-distributions. A t-distribution is a specially altered normal distribution that has been adjusted to account for the number of individuals being sampled. Specifically, a t-distributions with infinite degrees of freedom is the same as a normal distribution, and our degrees of freedom help create a more platykurtic distribution to account for error and uncertainty. The distribution can be calculated as follows:

$$t = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{\pi\nu}\Gamma(\frac{\nu}{2})}(1 + \frac{t^2}{\nu})^{-\frac{(v+1)}{2}}$$

These t-distributions can be visualized as follows:

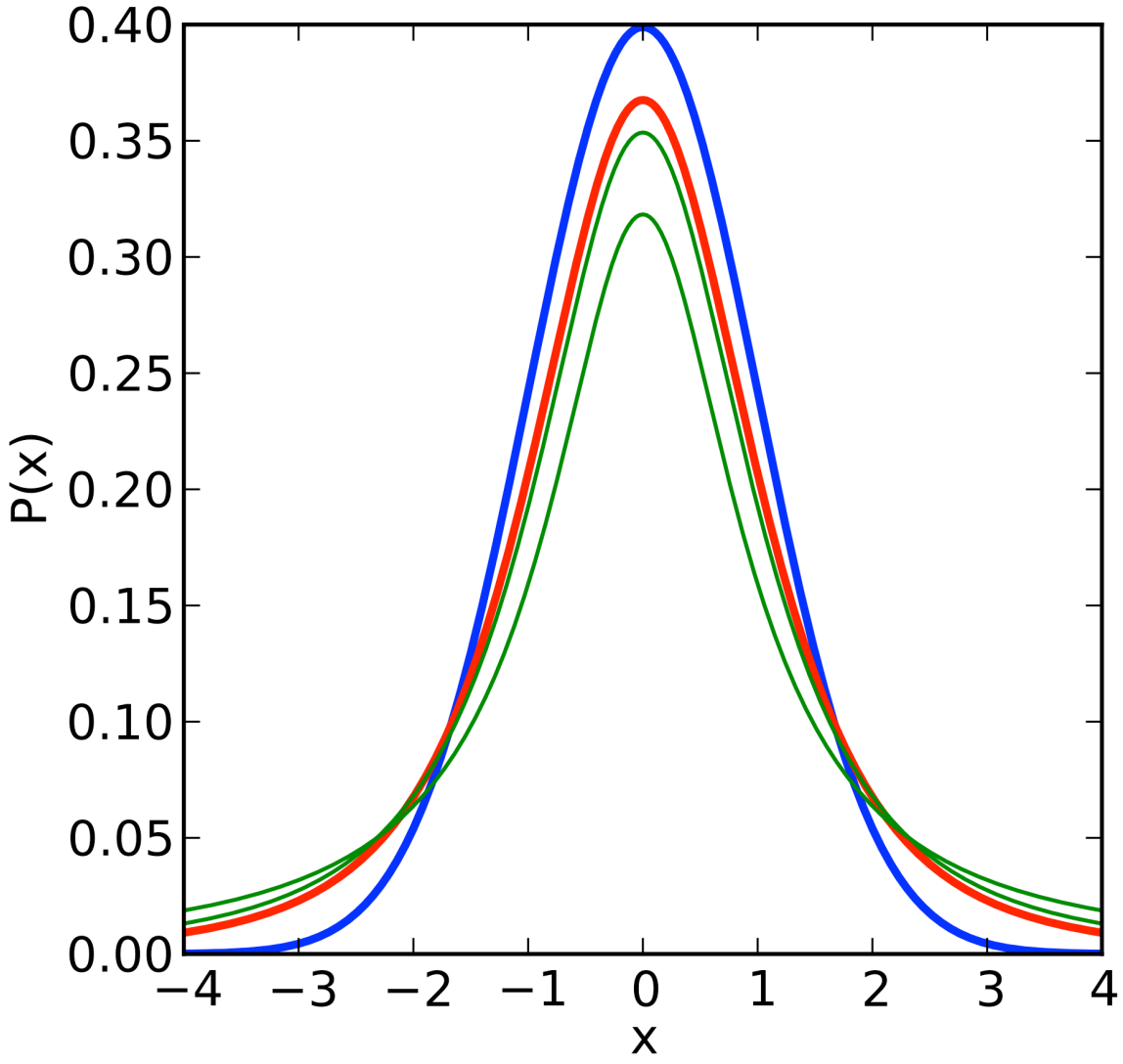


Figure 8.1: IkamusumeFan - Wikipedia

For all t-tests, we calculate the degrees of freedom based on the number of samples. If comparing values to a single sample, we use $df = n - 1$. If we are comparing two sample means, then we have $df = n_1 + n_2 - 2$.

Importantly, we are testing to see if the means of the two distributions are equal in a t-test. Thus, our hypotheses are as follows:

$H_0 : \mu_1 = \mu_2$ ***or*** $H_0 : \mu_1 - \mu_2 = 0$

$H_A : \mu_1 \neq \mu_2$ ***or*** $H_A : \mu_1 - \mu_2 \neq 0$

When asked about hypotheses, remember the above as the statistical hypotheses that are being directly tested.

In *R*, we have the following functions to help with t distributions:

- `dt`: density function of a t-distribution
- `pt`: finding our p value from a specific t in a t-distribution
- `qt`: finding a particular t from a specific p in a t-distribution
- `rt`: random values from a t-distribution

All of the above arguments required the degrees of freedom to be declared. Unlike the normal distribution functions, these can not be adjusted for your data; tests must be performed using `t.test`.

8.4 t-tests

We have three major types of t-tests:

- **One-sample t-tests**: a single sample is being compared to a value, or *vice versa*
- **Two-sample t-tests**: two samples are being compared to one another to see if they come from the same population
- **Paired t-tests**: before-and-after measurements of the same individuals are being compared. This is necessary to account for a repeat in the individuals being measured, and different potential baselines at initiation. In this case, we are looking to see if the difference between before and after is equal to zero.

We also have what we call a "true" t-test and "Welch's" t-test. The formula for a "true" t is as follows:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Where s_p is based on the "pooled variance" between the samples. This can be calculated as follows:

$$s_p = \sqrt{\frac{(n_1 - 1)(s_1^2) + (n_2 - 1)(s_2^2)}{n_1 + n_2 - 2}}$$

Whereas the equation for a "Welch's" t is:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Welch's t also varies with respect to the degrees of freedom, calculated by:

$$df = \frac{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}{\frac{(\frac{s_1^2}{n_1})^2}{n_1 - 1} + \frac{(\frac{s_2^2}{n_2})^2}{n_2 - 1}}$$

OK, so why the difference?

A t-test works well under a certain set of assumptions, include *equal variance* between samples and roughly equal *sample sizes*. A Welch's t-test is better for scenarios with *unequal variance* and *small sample sizes*. If sample sizes and variances are equal, the two t-tests should perform the same.

Because of this, some argue that "Welch's" should be the default t-test, and **in *R*, Welch's *is* the default t-test**. If you want to specify a "regular" t-value, you will have to set the option `var.equal = TRUE`. (The default is `var.equal = FALSE`).

8.4.1 One-sample t-tests

Let's look at the values of all of the dog samples in our `canines` dataset.

```
dogs <- canines |>
  filter(Sample == "Dog") |>
  select(Sample, OA)

xbar <- mean(dogs$OA)
sd_dog <- sd(dogs$OA)
n <- nrow(dogs)
```

Now we have stored all of our information on our dog dataset. Let's say that the overall populations of dogs a mean OA score of 143 with a $\sigma = 1.5$. Is our sample different than the overall population?

```
t.test(x = dogs$OA,
       alternative = "two.sided",
       mu = 143)
```

```
    One Sample t-test

data:  dogs$OA
t = -0.74339, df = 33, p-value = 0.4625
alternative hypothesis: true mean is not equal to 143
95 percent confidence interval:
 138.4667 145.1070
sample estimates:
mean of x
 141.7869
```

As we can see above, we fail to reject the null hypothesis that our sample is different than the overall mean for dogs.

8.4.2 Two-sample t-tests

Now let's say we want to compare foxes and dogs to each other. Since we have all of our data in the same data frame, we will have to *subset* our data to ensure we are doing this properly.

```
# already got dogs
dog_oa <- dogs$OA

foxes <- canines |>
  filter(Sample == "Fox") |>
  select(Sample, OA)

fox_oa <- foxes$OA
```

Now, we are ready for the test!

```
t.test(dog_oa, fox_oa)
```

```
    Welch Two Sample t-test
```

```
data:  dog_oa and fox_oa
t = -6.3399, df = 72.766, p-value = 1.717e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -19.62289 -10.23599
sample estimates:
mean of x mean of y
 141.7869  156.7163
```

As we can see, the dogs and the foxes significantly differ in their `OA` measurement, so we reject the null hypothesis that $\mu_{dog} = \mu_{fox}$.

8.4.3 Paired t-tests

I will do a highly simplified version of a paired t-test here just for demonstrations sake. Remember that you want to used paired tests when we are looking at the *same individuals* at different points in time.

```
# create two random distributions
# DEMONSTRATION ONLY

# make repeatable
set.seed(867)

t1 <- rnorm(20,0,1)
t2 <- rnorm(20,2,1)
```

Now we can compare these using `paired = TRUE`.

```
t.test(t1, t2, paired = TRUE)
```

```
    Paired t-test

data:  t1 and t2
t = -7.5663, df = 19, p-value = 3.796e-07
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -3.107787 -1.760973
sample estimates:
mean difference
       -2.43438
```

As we can see, we reject the null hypothesis that these distributions are equal in this case. Let's see how this changes though if we set `paired = FALSE`.

```
t.test(t1, t2)
```

```
    Welch Two Sample t-test

data:  t1 and t2
t = -8.1501, df = 37.48, p-value = 8.03e-10
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.039333 -1.829428
sample estimates:
  mean of x   mean of y
-0.07258938  2.36179080
```

This value differs because, in a paired test, we are looking to see if the difference between the distributions is 0, while in the independent (standard) test we are comparing the overall distributions of the samples.

8.5 Wilcoxon tests

When data (and the differences among data) are non-normal, they violate the assumptions of a t-test. In these cases, we have to do a Wilcoxon test (also called a Wilcoxon signed rank test). In R, the command `wilcox.test` also includes the Mann-Whitney U test for unpaired data and the standard Wilcoxon test W for paired data.

8.5.1 Mann-Whitney U

For this test, we would perform the following procedures to figure out our statistics:

1. Rank the pooled dataset from smallest to largest, and number all numbers by their ranks
2. Sum the ranks for the first column and the second column
3. Compute U_1 and U_2, comparing the smallest value to a Mann-Whitney U table.

The equations for these statistics are as follows, where R represents the sum of the ranks for that sample:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2$$

In R, this looks like so:

```
wilcox.test(t1, t2, paired = FALSE)
```

```
    Wilcoxon rank sum exact test

data:  t1 and t2
W = 11, p-value = 2.829e-09
alternative hypothesis: true location shift is not equal to 0
```

8.5.2 Wilcoxon signed rank test

For paired samples, we want to do the Wilcoxon signed rank test. This is performed by:

1. Finding the difference between sampling events for each sampling unit.
2. Order the differences based on their *absolute value*
3. Find the sum of the *positive ranks* and the *negative ranks*
4. The *smaller* of the values is your W statistic.

In R, this test looks as follows:

```
wilcox.test(t1, t2, paired = TRUE)
```

```
    Wilcoxon signed rank exact test

data:  t1 and t2
V = 0, p-value = 1.907e-06
alternative hypothesis: true location shift is not equal to 0
```

8.6 Confidence intervals

In t tests, we are looking at the difference between the means. Oftentimes, we are looking at a confidence interval for the difference between these means. This can be determined by:

$$(\bar{x}_1 - \bar{x}_2) \pm t_{crit}\sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}}$$

This is very similar to the CI we calculated with the Z statistic. **Remember** that we can use the following function to find our desired t, which requires degrees of freedom to work:

```
qt(0.975, df = 10)
```

```
[1] 2.228139
```

8.7 Homework: Chapter 9

For Chapter 9, complete problems 9.1, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, and 9.10. For problems 9.3 - 9.8, be sure to state the null and alternative hypotheses and whether the test is one- or two-tailed.

8.8 Homework: Chapter 10

Two-sample means are practiced in Chapter 10. Please see *Canvas* for more information.

9 ANOVA: Part 1

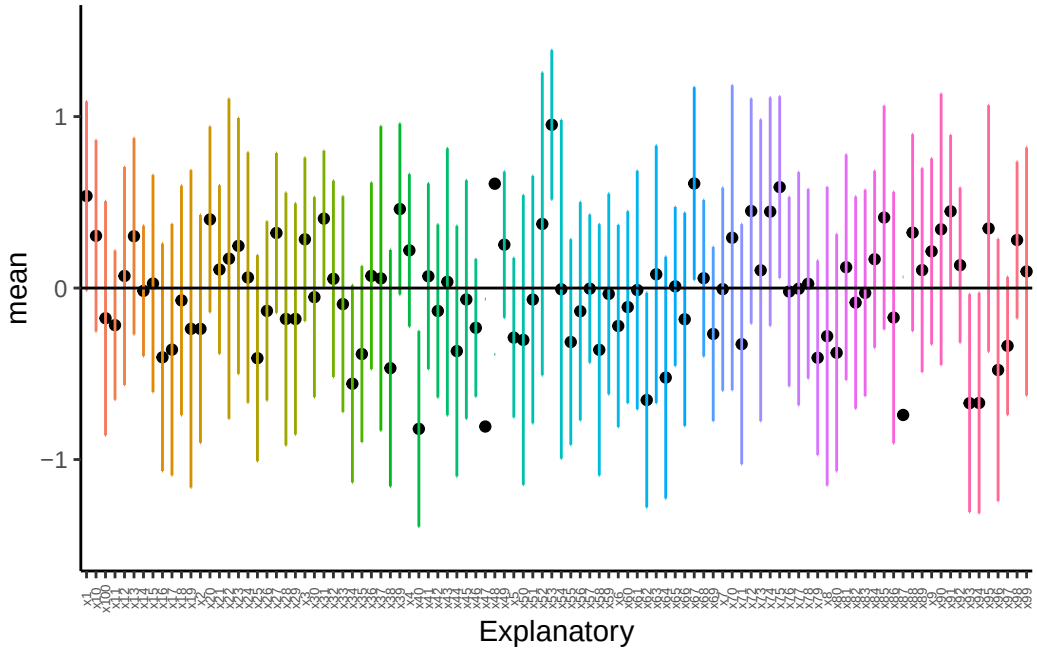
9.1 Introduction

When we are comparing multiple (2+) populations, we perform what is called an *analysis of variance* - or an ANOVA. We opt for this different method because we are trying to minimize error. As you'll recall, we use α to minimize our chances of making an error and coming to an incorrect conclusion regarding our data. In our previous tests (t-tests) we are comparing the means between two different populations, such that $H_0 : \mu_1 = \mu_2$. When comparing multiple populations, comparing the means in this direct fashion can increase the probability of introducing error into a system. Consider the following:

```
library(tidyverse)

# needed for summarizing data
library(plyr)

# needed for better Tukey tests
library(agricolae)
```

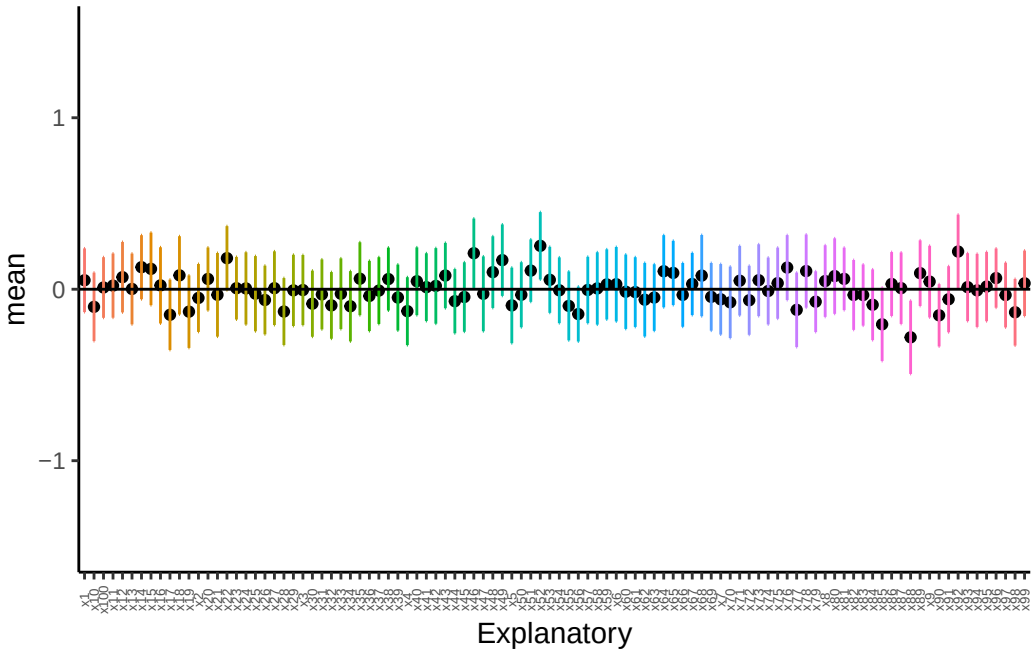
```
# This creates a reproducible example
# rnorm creates random datasets

set.seed(8675309)

for(i in 1:100){
  x <- rnorm(10)
  if(i == 1){
    data <- x |> as.data.frame()
    colnames(data) <- "Response"
    data$Explanatory <- paste0("x",i)
  }else{
    newdat <- x |> as.data.frame()
    colnames(newdat) <- "Response"
    newdat$Explanatory <- paste0("x",i)
    data <- rbind(data,newdat)
```

```
  }
}

# summarize by group
summary_data <- ddply(data, "Explanatory", summarise,
                  N = length(Response),
                  mean = mean(Response),
                  sd = sd(Response),
                  se = sd / sqrt(N))

ggplot(summary_data, aes(x = Explanatory, y = mean, group = Explanatory)) +
  geom_point() +
  geom_errorbar(data = summary_data, aes(ymin = mean - 2*se, ymax = mean+2*se,
                                         color = Explanatory), width = 0.1) +
  geom_hline(yintercept = 0, col = "black", linewidth = 0.5) +
  ylim(c(-1.5,1.5)) +
  theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, size = 5))
```



As we can see above, with just ten random samples and 100 sampling events, we get some datasets that do not have the mean included within the interquartile range, and thus have means that would be statistically different than what we draw. As we increase the number of draws, we get closer to the mean:

```r
set.seed(8675309)

for(i in 1:100){
  x <- rnorm(100)
  if(i == 1){
    data <- x |> as.data.frame()
    colnames(data) <- "Response"
    data$Explanatory <- paste0("x",i)
  }else{
    newdat <- x |> as.data.frame()
    colnames(newdat) <- "Response"
    newdat$Explanatory <- paste0("x",i)
    data <- rbind(data,newdat)
  }
}

# summarize by group
summary_data <- ddply(data, "Explanatory", summarise,
                 N = length(Response),
                 mean = mean(Response),
                 sd = sd(Response),
                 se = sd / sqrt(N))

ggplot(summary_data, aes(x = Explanatory, y = mean, group = Explanatory)) +
  geom_point() +
  geom_errorbar(data = summary_data, aes(ymin = mean - 2*se, ymax = mean+2*se,
                                         color = Explanatory), width = 0.1) +
  geom_hline(yintercept = 0, col = "black", linewidth = 0.5) +
  ylim(c(-1.5,1.5)) +
  theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, size = 5))
```



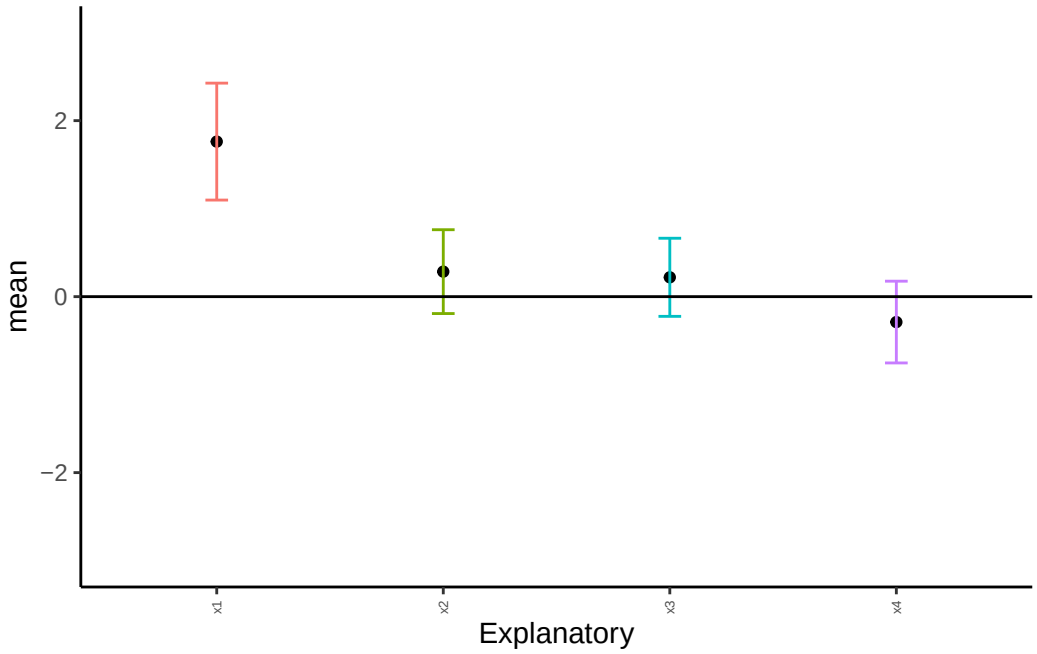
As we can see, even with 100 sample, we still have some chances of having groups that are different! When we do pairwise comparisons, we are compounding the error and the possibility of coming to an incorrect conclusion. Thus, when comparing multiple groups, we use the variances to see if groups come from the same distribution rather than the mean.

9.2 ANOVA: By hand

For this workthrough, we will use the following example dataset:

```
set.seed(8675309)

for(i in 1:4){
  x <- rnorm(10)
  if(i == 1){
    x <- rnorm(10, mean = 2)
    data <- x |> as.data.frame()
    colnames(data) <- "Response"
    data$Explanatory <- paste0("x",i)
  }else{
    newdat <- x |> as.data.frame()
    colnames(newdat) <- "Response"
    newdat$Explanatory <- paste0("x",i)
    data <- rbind(data,newdat)
```

```
  }
}

# split into "typical" table
expanded_data <- NULL
expanded_data$x1 <- data$Response[which(data$Explanatory=="x1")]
expanded_data$x2 <- data$Response[which(data$Explanatory=="x2")]
expanded_data$x3 <- data$Response[which(data$Explanatory=="x3")]
expanded_data$x4 <- data$Response[which(data$Explanatory=="x4")]

expanded_data <- expanded_data |>
  as.data.frame()

# summarize by group
summary_data <- ddply(data, "Explanatory", summarise,
                 N = length(Response),
                 mean = mean(Response),
                 sd = sd(Response),
                 se = sd / sqrt(N))

ggplot(summary_data, aes(x = Explanatory, y = mean, group = Explanatory)) +
  geom_point() +
  geom_errorbar(data = summary_data, aes(ymin = mean - 2*se, ymax = mean+2*se,
                                     color = Explanatory), width = 0.1) +
  geom_hline(yintercept = 0, col = "black", linewidth = 0.5) +
  ylim(c(-3,3)) +
  theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, size = 5))
```




```
expanded_data |> round(2)
```

```
     x1    x2    x3    x4
1  0.45  1.99  0.38 -0.64
2  3.02  0.04  1.15  0.00
3  2.15 -0.40  1.57  0.05
4  1.34 -0.47  0.59  0.68
5  1.01 -0.41 -0.62 -0.25
6  3.97  0.68 -0.23 -0.15
7  1.56  0.69  0.06 -0.87
8  1.10  0.53 -0.31 -1.98
9  1.85 -0.19 -0.25  0.24
10 1.17  0.38 -0.15  0.04
```

Above, we can see the made-up dataset where it appears as though one population differs from the other populations in our measurements. Let's calculate an ANOVA and find out if this is the case!

NOTE throughout this process that I am trying to name variables in a straightforward fashion so as not to lose my way.

9.2.1 Calculate group means and Grand Mean

Let us assume we have a dataset, x, that is k columns and n rows, with N data points in the entire data frame. We first want to take the column means for each group k, such that we have $\bar{x}_k$. We also need to find the mean of the entire dataset, $\bar{x}_n$. We can calculate this by taking $\frac{\Sigma x}{n}$, which we have to calculate by column as follows.

```
# calculate the mean of each group
# each group is in a single column
group_means <- colMeans(expanded_data)

# rounding to two decimal places
group_means |> round(2)
```

```
   x1    x2    x3    x4
 1.76  0.28  0.22 -0.29
```

Next, we need to calculate the number of total entries in the dataset. We have written a function to accomplish this incase some rows have different numbers of entries from others.

```
n <- 0

for(i in 1:ncol(expanded_data)){
  # account for unequal row length, if exists
  sample <- expanded_data[,i] |>
    as.numeric() |>
    na.omit()
  n <- n + length(sample)
}

n
```

```
[1] 40
```

Next, we can calculate the `grand_mean` of all of the data.

```
# sum up all the data
dataset_sum <- colSums(expanded_data) |>
  sum()

# divide by the number of data points
```

```r
grand_mean <- dataset_sum/n

# display mean
grand_mean |> round(2)
```

```
[1] 0.49
```

9.2.2 Total sum of squares

To calculate the *total sum of squares* (TSS), we need to take the deviations (differences) of each point from the grand mean $\bar{x}_n$, square them, and them take the sum of them.

```r
# calculate deviates
# can calculate across all table at once
grand_deviates_squared <- (expanded_data - grand_mean)^2

# round output for here
grand_deviates_squared |> round(2)
```

```
      x1   x2   x3   x4
1   0.00 2.22 0.01 1.28
2   6.39 0.20 0.43 0.25
3   2.74 0.81 1.17 0.20
4   0.72 0.94 0.01 0.04
5   0.26 0.83 1.23 0.56
6  12.10 0.04 0.52 0.42
7   1.13 0.04 0.19 1.87
8   0.37 0.00 0.65 6.11
9   1.84 0.46 0.55 0.07
10  0.46 0.01 0.42 0.21
```

```r
# calculate the sum of all the deviates
ss_total <- rowSums(grand_deviates_squared) |>
  sum()

ss_total |> round(2)
```

```
[1] 47.73
```

9.2.3 Within-group sum of squares

For each data point, we need to calculate its deviation from its own group mean, squaring these deviations and then summing them together. We can't calcualte this quite as elegantly as the aforementioned data, but we can write a function that will operate across the table and create a new dataset on our behalf.

```
# replicate dataset
# replace columns with deviate data
group_deviates <- expanded_data

# loop through each column
for(i in 1:ncol(group_deviates)){
  # get the data in each group
  dat <- group_deviates[,i]
  # calculate the group mean
  mu <- mean(dat)
  # calculate the group deviates
  dev.dat <- (dat - mu)^2
  # save into table
  group_deviates[,i] <- dev.dat
}

group_deviates |> round(2)
```

```
     x1   x2   x3   x4
1  1.72 2.90 0.02 0.12
2  1.59 0.06 0.87 0.08
3  0.15 0.47 1.84 0.11
4  0.18 0.57 0.14 0.95
5  0.57 0.49 0.70 0.00
6  4.89 0.16 0.20 0.02
7  0.04 0.16 0.02 0.34
8  0.44 0.06 0.28 2.85
9  0.01 0.22 0.22 0.28
10 0.35 0.01 0.14 0.11
```

```
# calculate sum of data table
ss_within <- colSums(group_deviates) |>
  sum()

ss_within |> round(2)
```

```
[1] 24.33
```

9.2.4 Among-group sum of squares

The total sum of squares is equal to the among groups sum of squares and the within groups sum of squares added together; thus, we can solve this part with some easy arithmetic.

```
ss_among <- ss_total - ss_within

ss_among |> round(2)
```

```
[1] 23.4
```

9.2.5 Calculate degrees of freedom

Our degrees of freedom for the "between" group is the number of categories minus one ($K - 1$).

```
ss_among_df <- ncol(expanded_data) - 1

ss_among_df
```

```
[1] 3
```

Our degrees of freedom for the within group are the number of total samples minus the number of categories ($N - K$).

```
ss_within_df <- n - ncol(expanded_data)

ss_within_df
```

```
[1] 36
```

Our degrees of freedom for the total sum of squares is the number of samples minus one ($N - 1$).

```
ss_total_df <- n - 1

ss_total_df
```

```
[1] 39
```

9.2.6 Calculate mean squares

For each category (among and within), the mean square is equal to the sum of squares divided by the degrees of freedom.

```
ms_among <- ss_among/ss_among_df

ms_among |> round(2)
```

```
[1] 7.8
```

```
ms_within <- ss_within/ss_within_df

ms_within |> round(2)
```

```
[1] 0.68
```

9.2.7 Get F statistic

We divide the sum of squares among data point by the sum of squares within data points to obtain our F statistic.

```
f_stat <- ms_among/ms_within

f_stat |> round(2)
```

```
[1] 11.54
```

9.2.8 Get p value

We can use the function `pf` to calculate the p value for any given F. *Note* that this function requires two different degrees of freedom to work correctly, **and we are always looking right** since this is a unidirectional distribution.

```
pf(f_stat,
   df1 = ss_among_df,
   df2 = ss_within_df,
   lower.tail = F)
```

```
[1] 1.894073e-05
```

Given how small our p value is, we want to round this to $p < 0.0001$. As we can see, it is very unlikely that these are the same population.

9.3 ANOVA: By *R*

For this, we need to use the dataframe where we have all data in a single column and all ID's in the other columns. We then show how we want the ANOVA to operate across the data using the ~ symbol.

```
data_aov <- aov(Response ~ Explanatory, data = data)

summary(data_aov)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
Explanatory  3  23.40   7.801   11.54 1.89e-05 ***
Residuals   36  24.33   0.676
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we can see, all these values match what we did by hand above!

9.4 Post-hoc Tukey test

ANOVA tells us *if* a test is different, but it doesn't tell us *which* test is different. To do this, we have to perform a Tukey test.

9.4.1 Tukey test by hand

To do this by hand, we will need a lot of data from our aforementioned ANOVA test.

We need to calculate pairwise differences between each set of means.

```r
x1_mean <- mean(data$Response[data$Explanatory == "x1"])
x2_mean <- mean(data$Response[data$Explanatory == "x2"])
x3_mean <- mean(data$Response[data$Explanatory == "x3"])
x4_mean <- mean(data$Response[data$Explanatory == "x4"])

group_means <- c(x1_mean, x2_mean, x3_mean, x4_mean)

# calculate all pairwise differences
pairwise_mean_diffs <- dist(group_means)

pairwise_mean_diffs |> round(2)
```

```
     1    2    3
2 1.48
3 1.54 0.06
4 2.05 0.57 0.51
```

Next, we need a critical Q value against which we can compare. For Tukey, our degrees of freedom are the same as the degrees of freedom for SS_{within}: $N - K$.

```r
# set p value
tukey_q <- qtukey(p = 0.95,
                  # get length of categories / columns
                  nmeans = ncol(expanded_data),
                  df = ss_within_df)

tukey_q |> round(2)
```

```
[1] 3.81
```

We need to multiply Q by the pooled variance.This is the same as the average of the variances for each group.

```r
var_data <- 0

# calculate pooled variance
for(i in 1:ncol(expanded_data)){
  var_data <- var_data + var(expanded_data[,i])
}
```

```
pooled_var_dat <- sqrt(var_data/n)

pooled_var_dat |> round(2)
```

```
[1] 0.26
```

We can calculate the Tukey critical value by multiplying the pooled variance by Q.

```
tukey_critical <- tukey_q*pooled_var_dat

tukey_critical |> round(2)
```

```
[1] 0.99
```

Remember, we are comparing to the actual value, not the rounded value.

Which mean differences are difference compared to our critical value?

```
pairwise_mean_diffs[pairwise_mean_diffs < tukey_critical] <- 0

pairwise_mean_diffs
```

```
         1        2        3
2 1.477866
3 1.542282 0.000000
4 2.051068 0.000000 0.000000
```

As we can see above, three differences cross our threshold - all associated with `x1`.

When we graph things, we want to label this group as different. We will cover this a little later in the tutorial.

9.4.2 Tukey test in *R*

Tukey tests in the *R* program are a bit more straightforward.

```
TukeyHSD(data_aov)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Response ~ Explanatory, data = data)

$Explanatory
             diff       lwr        upr     p adj
x2-x1 -1.47786579 -2.468021 -0.4877106 0.0015554
x3-x1 -1.54228164 -2.532437 -0.5521265 0.0009388
x4-x1 -2.05106768 -3.041223 -1.0609125 0.0000147
x3-x2 -0.06441585 -1.054571  0.9257393 0.9980525
x4-x2 -0.57320189 -1.563357  0.4169533 0.4141599
x4-x3 -0.50878604 -1.498941  0.4813691 0.5173399
```

As we can see above, only three comparisons have a p value of < 0.05, and thus only those three are significantly different. All involve `x1`.

We can also use `HSD.test` to get more specific results:

```r
tukey_data_aov <- HSD.test(data_aov,
                           # what to group by?
                           "Explanatory",
                           # significance level?
                           alpha = 0.05,
                           # are data unbalanced
                           unbalanced = FALSE,
                           # show answer?
                           console = TRUE)
```

```
Study: data_aov ~ "Explanatory"

HSD Test for Response

Mean Square Error:  0.6758192

Explanatory,  means

     Response       std  r        se        Min      Max        Q25         Q50
x1  1.7620153 1.0505466 10 0.2599652  0.4504476 3.972459  1.1175485  1.44911720
x2  0.2841495 0.7532422 10 0.2599652 -0.4729986 1.985826 -0.3497379  0.21347543
x3  0.2197337 0.7019368 10 0.2599652 -0.6150452 1.574903 -0.2436023 -0.04493909
```

```
x4 -0.2890524 0.7345336 10 0.2599652 -1.9769014 0.684072 -0.5394534 -0.07741642
          Q75
x1 2.07491533
x2 0.64579865
x3 0.53544151
x4 0.04323417

Alpha: 0.05 ; DF Error: 36
Critical Value of Studentized Range: 3.808798

Minimun Significant Difference: 0.9901551

Treatments with the same letter are not significantly different.

     Response groups
x1  1.7620153      a
x2  0.2841495      b
x3  0.2197337      b
x4 -0.2890524      b
```

```
print(tukey_data_aov)
```

```
$statistics
    MSerror Df      Mean       CV       MSD
  0.6758192 36 0.4942115 166.3422 0.9901551

$parameters
   test      name.t ntr StudentizedRange alpha
  Tukey Explanatory   4         3.808798  0.05

$means
     Response       std  r        se        Min      Max        Q25         Q50
x1  1.7620153 1.0505466 10 0.2599652  0.4504476 3.972459  1.1175485  1.44911720
x2  0.2841495 0.7532422 10 0.2599652 -0.4729986 1.985826 -0.3497379  0.21347543
x3  0.2197337 0.7019368 10 0.2599652 -0.6150452 1.574903 -0.2436023 -0.04493909
x4 -0.2890524 0.7345336 10 0.2599652 -1.9769014 0.684072 -0.5394534 -0.07741642
          Q75
x1 2.07491533
x2 0.64579865
x3 0.53544151
x4 0.04323417
```

```
$comparison
NULL

$groups
     Response groups
x1  1.7620153      a
x2  0.2841495      b
x3  0.2197337      b
x4 -0.2890524      b

attr(,"class")
[1] "group"
```

This output is nice because it labels groups based on which groups belong together! **This will be important for plotting, and is cumbersome if you have a lot of means.**

9.5 Plotting our ANOVA results

When we plot our ANOVAs, we want to show which mean is different from all of the others. Below, I will show the full pipeline for performing an ANOVA on a dataset and plotting the data at the end using out `data` object.

```
data_aov <- aov(Response ~ Explanatory,data)

summary(data_aov)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
Explanatory  3  23.40   7.801   11.54 1.89e-05 ***
Residuals   36  24.33   0.676
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now, we need to summarize things based on our `plyr` function.

```
# summarize by group
summary_data <- ddply(data, "Explanatory", summarise,
                 N = length(Response),
                 mean = mean(Response),
                 sd = sd(Response),
                 se = sd / sqrt(N))
```

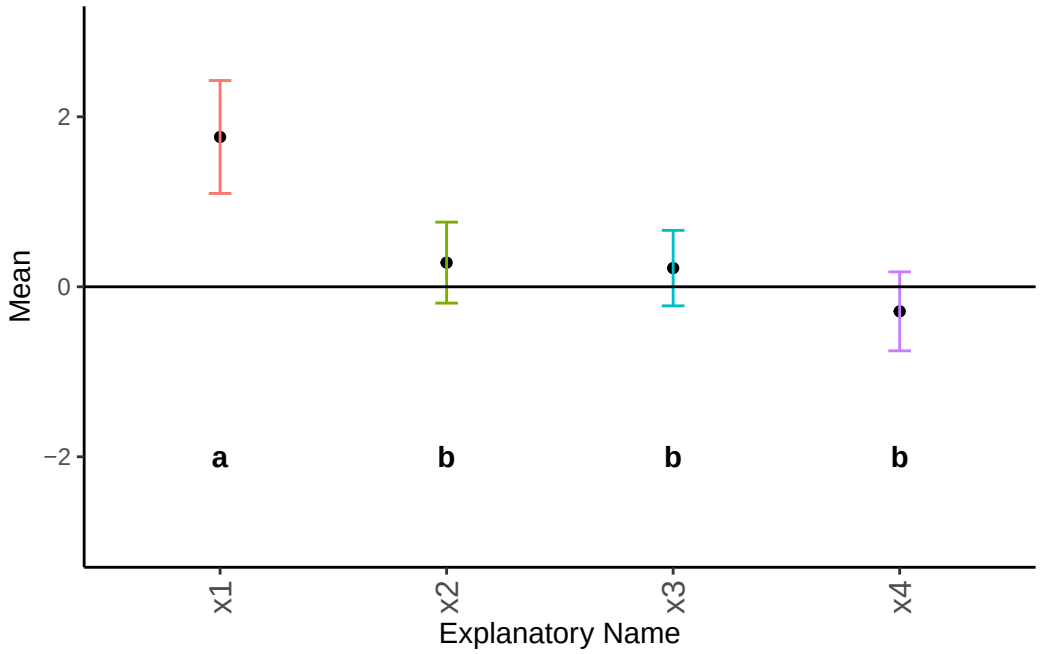
Now, we can use the aforementioned Tukey results to assign to groups. Because `agricolae` functions define the groups in a `$groups` slot, we can pull out these data and perform some minimal transformations to make them ready to plot. *This will save us a lot of time and hassle.*

```
# note first group must be EXACT MATCH to your summary_data object
# groups are saved in the Tukey object
# this is true for Tukey later as well

# the following is a function that will make the significant label table
sig.label.maker <- function(tukey_test, group_name){
  sig.labels <- tukey_test$groups |>
    # convert to a data.frame
    as.data.frame() |>
    # create a new column - place rownames into the column
    # converts to a format better for ggplot
    mutate(Explanatorys = rownames(tukey_test$groups)) |>
    # rename column to prevent confusion
    # specify dplyr; default function may be from plyr and not work
    dplyr::rename(significance = groups)
  colnames(sig.labels)[which(colnames(sig.labels) == "Explanatorys")] <- group_name
  return(sig.labels)
}

# pull out the groups slot from tukey
# same for Kruskal later on!
sig.labels <- sig.label.maker(tukey_data_aov, "Explanatory")

sig.labels
```

```
     Response significance Explanatory
x1  1.7620153            a          x1
x2  0.2841495            b          x2
x3  0.2197337            b          x3
x4 -0.2890524            b          x4
```

Note that in the above, the row labels and the column `Explanatory` are identical. We moved these data into a column to make it easier to use `ggplot`. If you want letters to be different - capitalized or something else - you will have to do this yourself. Now we can plot our data and add the labels!

```r
ggplot(summary_data, # plot summary data
       # Define plotting - x by group, y is mean, grouping by group
       aes(x = Explanatory, y = mean, group = Explanatory)) +
  # add points to plot for y values
  geom_point() +
  # add error bars around points
  geom_errorbar(data = summary_data,
                # define error bars
                aes(ymin = mean - 2*se, ymax = mean+2*se,
                    # define color, size
                                    color = Explanatory), width = 0.1) +
  # add line at average for the main group
  # this is not always known - nor requires!
  geom_hline(yintercept = 0, col = "black", linewidth = 0.5) +
  # set vertical limits for plot
  ylim(c(-3,3)) +
  # make it a classic theme - more legible
  theme_classic() +
  # add text to plot
  geom_text(data = sig.labels,
            # make bold
            fontface = "bold",
            # define where labels should go
            aes(x = Explanatory,
                # define height of label
                y = -2,
                # what are the labels?
                label = paste0(significance))) +
  xlab("Explanatory Name") +
  ylab("Mean") +
  # remove legend - not needed here
  theme(legend.position = "none",
        # make label text vertical, easier to read
        axis.text.x = element_text(angle = 90,
                                   # vertical offset of text
                                   vjust = 0.5,
                                   # text size
                                   size = 12))
```



Note that I place the labels below here, but they could also be placed above. You have to position them based on your own judgment.

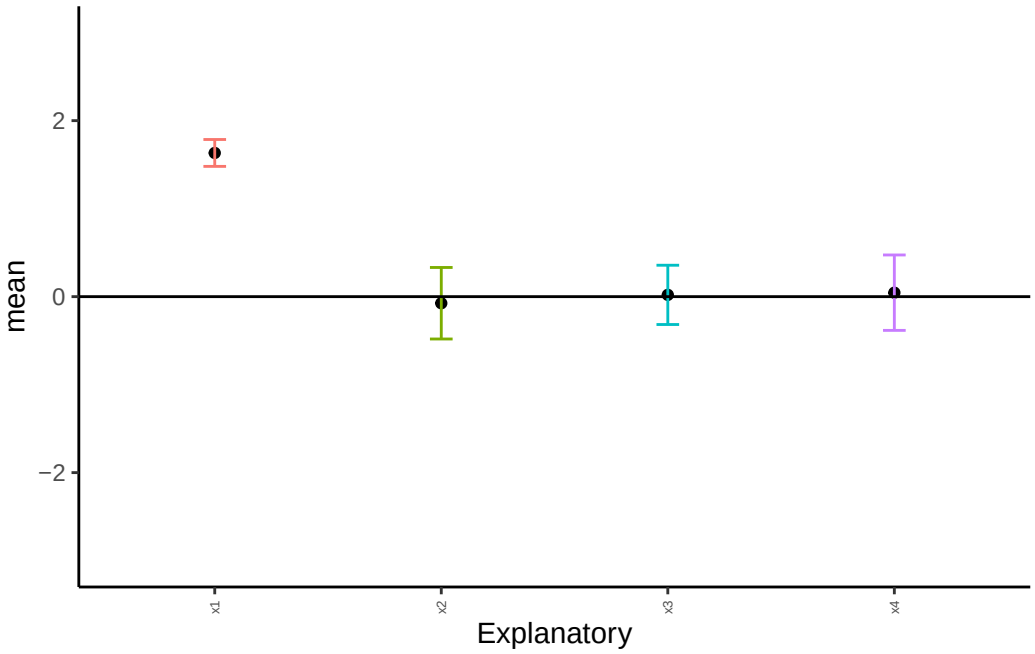
Addendum: if you see a mean labeled `a` and `b`, it would be statistically indistinguishable from means labeled `a` and means labeled `b`, despite means with only an `a` or a `b` being different from each other.

9.6 Kruskal-Wallis tests

The Kruskal-Wallis test is the non-parametric version of an ANOVA. To demonstrate this, we will be creating a non-normal distribution by pulling random values from a *uniform* distribution, using the random uniform function `runif`. **Note** we are rounding the data here to make it more similar to non-normal datasets you may encounter, and to increase the probability of ties.

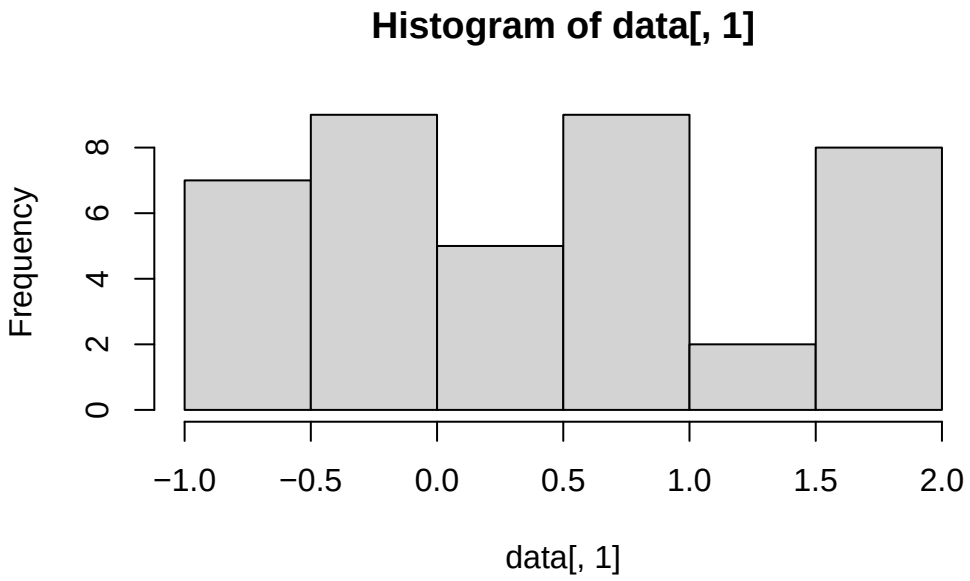
```
set.seed(8675309)

for(i in 1:4){
  x <- runif(10, min = -1, max = 1) |>
    round(2)
  if(i == 1){
    x <- runif(10, min = 1, max = 2) |>
      round(2)
```

```
    data <- x |> as.data.frame()
    colnames(data) <- "Response"
    data$Explanatory <- paste0("x",i)
  }else{
    newdat <- x |> as.data.frame()
    colnames(newdat) <- "Response"
    newdat$Explanatory <- paste0("x",i)
    data <- rbind(data,newdat)
  }
}

# split into "typical" table
expanded_data <- NULL
expanded_data$x1 <- data$Response[which(data$Explanatory=="x1")]
expanded_data$x2 <- data$Response[which(data$Explanatory=="x2")]
expanded_data$x3 <- data$Response[which(data$Explanatory=="x3")]
expanded_data$x4 <- data$Response[which(data$Explanatory=="x4")]

expanded_data <- expanded_data |>
  as.data.frame()

# summarize by group
summary_data <- ddply(data, "Explanatory", summarise,
                 N = length(Response),
                 mean = mean(Response),
                 sd = sd(Response),
                 se = sd / sqrt(N))

ggplot(summary_data, aes(x = Explanatory, y = mean, group = Explanatory)) +
  geom_point() +
  geom_errorbar(data = summary_data, aes(ymin = mean - 2*se, ymax = mean+2*se,
                                     color = Explanatory), width = 0.1) +
  geom_hline(yintercept = 0, col = "black", linewidth = 0.5) +
  ylim(c(-3,3)) +
  theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, size = 5))
```



We can confirm these data are non-normal with Shapiro-Wilk tests and histograms. Demonstrating with the first column only:

```
hist(data[,1])
```




```
shapiro.test(data[,1])
```

```
    Shapiro-Wilk normality test

data:  data[, 1]
W = 0.93404, p-value = 0.02187
```

The above histogram appears non-normal, and the Shapiro-Wilk also indicates this is non-normal.

9.6.1 By hand

First, we need to order all the data in the entire dataset. **This is easiest to do if we use the dataset with all data in a single column.**

```
data$ranks <- rank(data$Response, ties.method = "average")

# view first couple of rows
head(data)
```

```
  Response Explanatory ranks
1     1.84          x1    39
2     1.58          x1    34
3     1.51          x1    33
4     1.26          x1    32
5     1.75          x1    37
6     1.92          x1    40
```

Now, we need to calculate the sum of the ranks for each category. The below function will take the sum of the ranks for the rows which match the condition of having the group be `x1`.

```
x1_sum <- sum(data$ranks[which(data$Explanatory=="x1")])
x2_sum <- sum(data$ranks[which(data$Explanatory=="x2")])
x3_sum <- sum(data$ranks[which(data$Explanatory=="x3")])
x4_sum <- sum(data$ranks[which(data$Explanatory=="x4")])
```

Now, we need to calculate our test statistic. The test statistic is H, with:

$$H = \frac{12}{N(N+1)} \cdot \Sigma \frac{R_j^2}{n_j} - 3(N+1)$$

In this equation, R is the sum of the ranks for a given category. This follows a χ^2 distribution with $k-1$ degrees of freedom, with k referring to categories.

For these, we need to know what n is for each category.

```
n1 <- length(data$ranks[which(data$Explanatory=="x1")])
n2 <- length(data$ranks[which(data$Explanatory=="x2")])
n3 <- length(data$ranks[which(data$Explanatory=="x3")])
n4 <- length(data$ranks[which(data$Explanatory=="x4")])
```

Next, we can calculate the sums of the $\frac{R^2}{n}$ term.

```
r2_sum <- sum(x1_sum^2/n,
              x2_sum^2,n,
              x3_sum^2/n,
              x4_sum^2/n)
```

Now, we can calculate H.

```
N <- sum(n1, n2, n3, n4)

H <- ((12/(N*(N+1)))*r2_sum)-(3*(N+1))

H |> round(2)
```

```
[1] 57.43
```

Now, we can evaluate this with a χ^2 p value.

```
pchisq(q = H,
       df = ncol(data)-1,
       # remember, looking right!
       lower.tail = FALSE)
```

```
[1] 3.382831e-13
```

As we can see, the probability is extremely low with $p < 0.0001$. One distribution is different, and we can proceed with Tukey tests.

9.6.2 Using *R*

We can also use *R* for this test.

```
kruskal_data <- kruskal.test(Response ~ Explanatory, data)

print(kruskal_data)
```

```
    Kruskal-Wallis rank sum test

data:  Response by Explanatory
Kruskal-Wallis chi-squared = 22.149, df = 3, p-value = 6.073e-05
```

This test is similar, but not quite the same, as what we calculated above. The package `agricolae` provides a more in-depth Kruskal-Wallis test that gives us more metadata. We have to use the wrapper `with` to get this to work properly, per the help page.

```
# use "with", defining your dataset first
data_kruskal <- with(data,
                     # define parameters for Kruskal-Wallis test
                     # First, variable of interest
                     kruskal(Response,
                             # second, grouping variable
                             Explanatory,
                             # do you want group designations returned?
                             group = TRUE,
                             # what is this being done on?
                             # must match dataset name in "with"!!!
                             main = "data"))

# display results
# summary WILL NOT show the right thing
print(data_kruskal)
```

```
$statistics
     Chisq Df     p.chisq  t.value      MSD
  22.14917  3 6.07312e-05 2.028094 7.252228

$parameters
            test p.ajusted      name.t ntr alpha
```

```
  Kruskal-Wallis      none Explanatory   4  0.05

$means
   Response  rank       std  r   Min  Max     Q25    Q50    Q75
x1    1.633 35.50 0.2415252 10  1.21 1.92  1.5275  1.720 1.8025
x2   -0.075 14.20 0.6424130 10 -0.88 0.69 -0.6325 -0.105 0.5575
x3    0.020 16.15 0.5327080 10 -0.63 0.95 -0.3250 -0.110 0.3775
x4    0.045 16.15 0.6774011 10 -0.92 0.95 -0.3500 -0.135 0.6475

$comparison
NULL

$groups
   Response groups
x1    35.50      a
x3    16.15      b
x4    16.15      b
x2    14.20      b

attr(,"class")
[1] "group"
```

As we can see, the above gives us our group designations under the `$group` section. This is what we need to be able to plot things, as with ANOVA above. I do not repeat those steps here.

9.7 Homework: Chapter 11

Your homework is to complete problems 11.1, 11.2, 11.3, and 11.4. The first two will require ANOVA, and the last two will require Kruskal-Wallis tests. For each pair of tests, you must also complete the problem by hand in addition to using the default *R* methods. For *all* problems:

- State your hypothesis in words
- State your hypothesis mathematically (hint: μ)
- **Answer in complete sentences** and don't forget to round your answers.
- If you reject the null hypothesis, you must plot the results and label them to show which means are different.

10 ANOVA: Part 2

10.1 Two-way ANOVA

Previously, we discussed one-way ANOVAs, where we are looking at a single factor split across three or more groups and trying to determine if the means of these groups are equal (*i.e.*, $H_0 : \mu_1 = \mu_2 = ...\mu_i$). ANOVA specifically allows us to analyze the variance of these different groups to ascertain which factors are most responsible for the variation we observe in the data. Because of the way ANOVA operates, we can actually test multiple different combinations of variables simultaneously in what we call a *two-way ANOVA*.

Don't forget to load your required packages - some we have used before, like `agricolae`, `plyr`, and `tidyverse`, but others are new for this section: `multcomp` and `nlme`! As a reminder, these packages are designed for the following:

- `agricolae`: originally written as a Master's thesis at the Universidad Nacional de Ingeniería (Lima, Perú), this package is designed to help with agricultural research.
- `plyr`: tools for common problems, including splitting data, applying functions across data, and combining datasets together.
- `tidyverse`: one we are already familiar with; a wrapper for installing `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, and `forcats`.
- `multcomp`: more in depth and better **MULT**iple **COMP**arisons via linear models and related models.
- `nlme`: a package for fitting Gaussian and **n**on-**l**inear **m**ixed-**e**ffect models.
- `PMCMRplus`: a math package with post-hoc tests for Friedman's test

```
library(agricolae)
library(plyr)
library(tidyverse)

# NEW PACKAGES NEEDED
# Don't forget to install these on your machine
library(multcomp)
library(nlme)
library(PMCMRplus)
```

10.2 Designs

There are several different designs for two-way ANOVAs, and we will cover some of the most common designed here.

For these examples, we are going to randomly generated examples. I will refer to the variables as `Response` and `Explanatory` for simplicity's sake.

10.2.1 Randomized block design

Randomized block designs look at combinations of variables that could be affecting the results. More specifically, we are looking at two *strata* or *factors* and their effects on a *continuous* response variable.

```
set.seed(8675309)

# random example

# Blocking variable
Blocking_Variable <- c("Group 1", "Group 2", "Group 3")

# explanatory variables
# these are your columns
# these are your primary hypothesis
Explanatory_1 <- c(10.1, 9.4, 11.1)
Explanatory_2 <- c(12, 13.0, 15.4)
Explanatory_3 <- c(11.2, 10.1, 11.9)

# create "data table" as we normally see it
# combine all columns
data_expanded <- cbind(Blocking_Variable,
                       Explanatory_1,
                       Explanatory_2,
                       Explanatory_3) |>
  as.data.frame() # create data frame

data_expanded
```

```
  Blocking_Variable Explanatory_1 Explanatory_2 Explanatory_3
1           Group 1          10.1            12          11.2
2           Group 2           9.4            13          10.1
3           Group 3          11.1          15.4          11.9
```

Note that this table is in the format that we most often see, but we need to reshape these data to make it easier for us to perform our analyses. I created the data here as a matrix with named columns and rows; the following code may need to be adjusted if you do things differently.

```
# expand to "long" format
# if not done earlier, convert to data frame

data <- data_expanded |>
  # !by column for aggregating
  # names_to = what to name column aggregation
  # values_to = what the measurements should be called
  pivot_longer(!Blocking_Variable, names_to = "Explanatory_Variables", values_to = "Measureme

data
```

```
# A tibble: 9 x 3
  Blocking_Variable Explanatory_Variables Measurements
  <chr>             <chr>                 <chr>
1 Group 1           Explanatory_1         10.1
2 Group 1           Explanatory_2         12
3 Group 1           Explanatory_3         11.2
4 Group 2           Explanatory_1         9.4
5 Group 2           Explanatory_2         13
6 Group 2           Explanatory_3         10.1
7 Group 3           Explanatory_1         11.1
8 Group 3           Explanatory_2         15.4
9 Group 3           Explanatory_3         11.9
```

Now we can do our ANOVA. ***Note*** that I put `factor` around the blocking variable.

```
# mark block by factor
# best to always use
data_aov <- aov(Measurements ~ Explanatory_Variables + factor(Blocking_Variable), data)

summary(data_aov)
```

```
                         Df Sum Sq Mean Sq F value Pr(>F)
Explanatory_Variables     2 17.182   8.591  14.412 0.0149 *
factor(Blocking_Variable) 2  6.829   3.414   5.728 0.0670 .
Residuals                 4  2.384   0.596
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this particular example, the blocking variable does not significantly differ, however the explanatory variable does differ.

Remember, the columns represent your *primary hypothesis*. You will only plot your results if your *primary hypothesis is significant!*

Given that our primary null hypothesis is rejected (that is to say, not all means are equal), we need to plot our results.

To determine which mean(s) differ, we will use a Tukey Test. Unfortunately, the `agricolae` function `HSD.test` does not work as well for these multi-directional ANOVAs, so we need to use `TukeyHSD`.

```
tukey_data_aov <- TukeyHSD(data_aov)

tukey_data_aov
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Measurements ~ Explanatory_Variables + factor(Blocking_Variable), data = d

$Explanatory_Variables
                                   diff       lwr        upr     p adj
Explanatory_2-Explanatory_1  3.2666667  1.019919  5.5134144 0.0144034
Explanatory_3-Explanatory_1  0.8666667 -1.380081  3.1134144 0.4333834
Explanatory_3-Explanatory_2 -2.4000000 -4.646748 -0.1532523 0.0406301

$`factor(Blocking_Variable)`
                      diff        lwr      upr     p adj
Group 2-Group 1 -0.2666667 -2.5134144 1.980081 0.9082398
Group 3-Group 1  1.7000000 -0.5467477 3.946748 0.1118461
Group 3-Group 2  1.9666667 -0.2800810 4.213414 0.0745795
```

As we can see above, each pairwise comparison is given a p value for the level of difference. We need to manually label these groups based on these p values, with groups being considered different if $p < 0.05$. We can do this as follows, but unfortunately, we need to do it by hand since we don't have a short-form code (*yet*) for this conversion.

```
# change Explanatory_Variables to your data
Treatments <- unique(data$Explanatory_Variables)

sig_labels <- Treatments |>
```

```
  as.data.frame() |>
  mutate(Significance = rep(NA, length(Treatments)))

# Change Explanatory_Variables to your data
colnames(sig_labels) <- c("Explanatory_Variables", # MUST BE SAME AS DATA
                          "Significance")

tukey_data_aov$Explanatory_Variables
```

```
                                diff       lwr        upr      p adj
Explanatory_2-Explanatory_1  3.2666667  1.019919  5.5134144 0.01440339
Explanatory_3-Explanatory_1  0.8666667 -1.380081  3.1134144 0.43338343
Explanatory_3-Explanatory_2 -2.4000000 -4.646748 -0.1532523 0.04063012
```

NOTE that I am going to have to adjust column names and variable names a few times. Based on the above, we can see that `Explanatory_3` and `Explanatory_1` are not different from each other, but everything else is in a different group relative to each other. We can label these by hand.

```
sig_labels$Significance <- c("A", "B", "A")

sig_labels
```

```
  Explanatory_Variables Significance
1         Explanatory_1            A
2         Explanatory_2            B
3         Explanatory_3            A
```

As we can see above, now only `Explanatory_2` is given a different letter category.

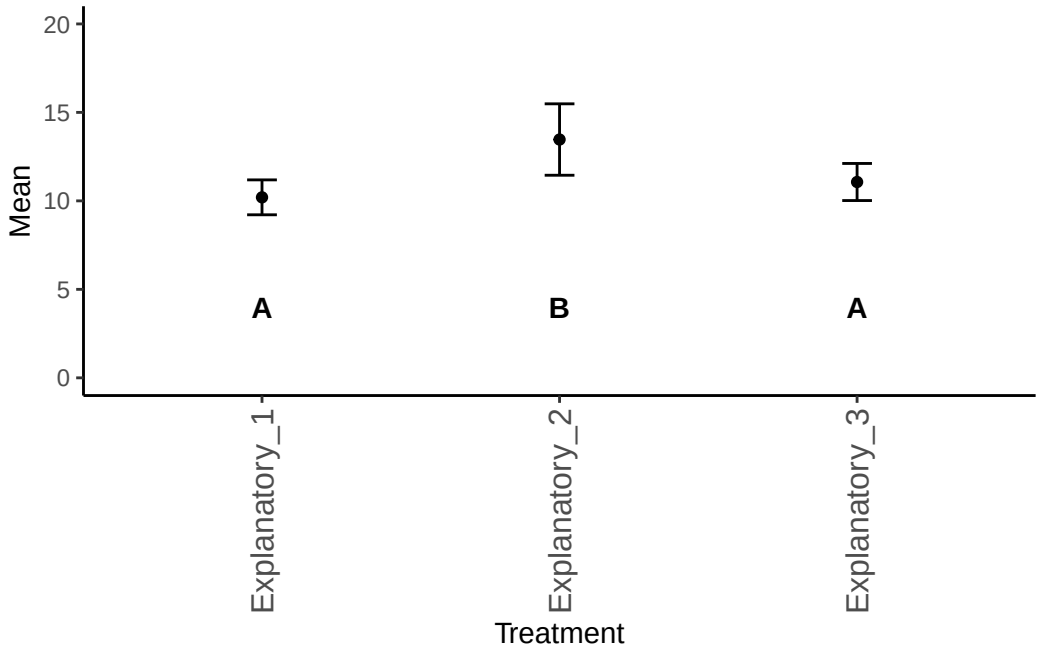
Now, we can plot these different factors.

```
# summarize by group
# slight adjustment from previous
summary_data <- ddply(data, "Explanatory_Variables", summarise,
                          N = length(as.numeric(Measurements)),
                          mean = mean(as.numeric(Measurements)),
                          sd = sd(as.numeric(Measurements)),
                          se = sd / sqrt(N))

summary_data
```

```
  Explanatory_Variables N     mean        sd        se
1         Explanatory_1 3 10.20000 0.8544004 0.4932883
2         Explanatory_2 3 13.46667 1.7473790 1.0088497
3         Explanatory_3 3 11.06667 0.9073772 0.5238745
```

```r
# SET Y LIMITS
# change based on observed data
ylims <- c(0, 20)

# set label height, can change before plotting function
label_height <- 4

ggplot(summary_data, # plot summary data
       # Define plotting - x by group, y is mean, grouping by group
       aes(x = Explanatory_Variables, y = mean)) +
  # add points to plot for y values
  geom_point() +
  # add error bars around points
  geom_errorbar(data = summary_data,
                # define error bars
                aes(ymin = mean - 2*se, ymax = mean+2*se),
                # width of bar
                width = 0.1) +
  # set vertical limits for plot
  ylim(ylims) +
  # make it a classic theme - more legible
  theme_classic() +
  # add text to plot
  geom_text(data = sig_labels,
            # make bold
            fontface = "bold",
            # define where labels should go
            aes(x = Explanatory_Variables,
                # define height of label
                y = 4,
                # what are the labels?
                label = paste0(Significance))) +
  xlab("Treatment") +
  ylab("Mean") +
  # remove legend - not needed here
  theme(legend.position = "none",
        # make label text vertical, easier to read
```

```
        axis.text.x = element_text(angle = 90,
                                   # vertical offset of text
                                   vjust = 0.5,
                                   # text size
                                   size = 12))
```



10.2.2 Repeated measures

Now, we are going to do a *repeated measures ANOVA*, where we have the same individuals being measured multiple times. Consider the following imaginary dataset:

```
Visit_1 <- c(5.5,6.2,5.8)
Visit_2 <- c(4.6,5.4,5.2)
Visit_3 <- c(3.8,4.0,3.9)

Individuals <- c(paste0("Individual"," ",1:3))

data <- cbind(Individuals,
                  Visit_1,
                  Visit_2,
                  Visit_3) |>
  as.data.frame() |>
```

```
  pivot_longer(!Individuals,
               names_to = "Visits",
               values_to = "Measurements")

data
```

```
# A tibble: 9 x 3
  Individuals  Visits  Measurements
  <chr>        <chr>   <chr>
1 Individual 1 Visit_1 5.5
2 Individual 1 Visit_2 4.6
3 Individual 1 Visit_3 3.8
4 Individual 2 Visit_1 6.2
5 Individual 2 Visit_2 5.4
6 Individual 2 Visit_3 4
7 Individual 3 Visit_1 5.8
8 Individual 3 Visit_2 5.2
9 Individual 3 Visit_3 3.9
```

We need to perform the ANOVA again, but we need to account for the factor of which locations are repeated.

```
repeated_aov <- aov(Measurements ~ factor(Visits) + Error(factor(Individuals)), data)

summary(repeated_aov)
```

```
Error: factor(Individuals)
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  2 0.4867  0.2433

Error: Within
               Df Sum Sq Mean Sq F value   Pr(>F)
factor(Visits)  2  5.687  2.8433   89.79 0.000475 ***
Residuals       4  0.127  0.0317
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Unfortunately, because of the model this is, we cannot perform a Tukey Test on the "object" that is created from this ANOVA analysis. We can, however, approach this from a different

direction and get our Tukey results (thanks to Henrik on StackOverflow!). **For this to work, we need to install the packages `nlme` and `multcomp`.**

```
# ensure data is proper format
data$Individuals <- as.factor(data$Individuals)
data$Visits <- as.factor(data$Visits)
data$Measurements <- as.numeric(data$Measurements)
```

The next part of the code fits a *linear model* to the data. A linear model, which we will cover later in the class, is mathematically very similar to an ANOVa. However, we can data from this model and extract the ANOVA data to understand more about the interactions. We need to use a linear model for this to account with the relationships between the two.

```
# fit a linear mixed-effects model
# similar to ANOVA

lme_data <- lme(Measurements ~ Visits,
                          data = data,
                          # define repeated section
                          random = ~1|Individuals)

# perform ANOVA on model
anova(lme_data)
```

```
            numDF denDF  F-value p-value
(Intercept)     1     4 900.1633  <.0001
Visits          2     4  89.7895   5e-04
```

As we can see above, we can get the ANOVA results from this linear mixed-effects model fit to the dataset. Now, we need to know post-hoc which sets are different:

```
lme_data |>
  # "general linear hypothesis"
  # define a comparison to make
  # can add corrections like test = adjusted (type = "bonferroni")
  glht(linfct = mcp(Visits = "Tukey")) |>
  # return a summary of the above
  summary()
```

```
     Simultaneous Tests for General Linear Hypotheses
```

```
Multiple Comparisons of Means: Tukey Contrasts


Fit: lme.formula(fixed = Measurements ~ Visits, data = data, random = ~1 |
    Individuals)

Linear Hypotheses:
                       Estimate Std. Error z value Pr(>|z|)
Visit_2 - Visit_1 == 0  -0.7667     0.1453  -5.277   <1e-06 ***
Visit_3 - Visit_1 == 0  -1.9333     0.1453 -13.306   <1e-06 ***
Visit_3 - Visit_2 == 0  -1.1667     0.1453  -8.030   <1e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

We can see that every visit is different.

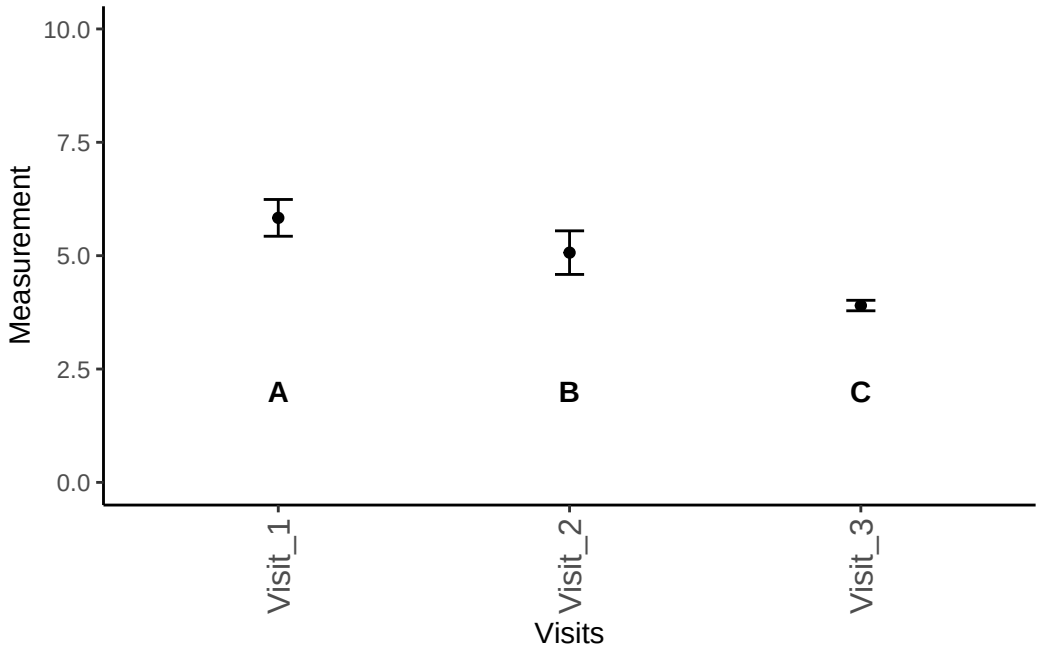
```
# manually labeling

sig_levels_repeated <- matrix(data = c("Visit_1", "A",
                                       "Visit_2", "B",
                                       "Visit_3", "C"),
                              byrow = T, ncol = 2) |> as.data.frame()

# make labels match!
colnames(sig_levels_repeated) <- c("Visits", "Significance")
```

Let's plot these. *Note* that we are not summarizing these the same way, since things are varying based on individual as well.

Note: For reasons I am not certain, you need to put the locations and then `ggplot` uses these colors to define everything. I really don't know why this is happening, so if you have a solution, let me know.

```
summary_data <- ddply(data, "Visits", summarise,
                          N = length(as.numeric(Measurements)),
                          mean = mean(as.numeric(Measurements)),
                          sd = sd(as.numeric(Measurements)),
                          se = sd / sqrt(N))
```

```r
ggplot(summary_data,
       aes(x = Visits, y = mean)) +
  geom_point() +
  geom_errorbar(data = summary_data,
                # define error bars
                aes(ymin = mean - 2*se, ymax = mean+2*se),
                # width of bar
                width = 0.1) +
  # set vertical limits for plot
  ylim(c(0,10)) +
  # make it a classic theme - more legible
  theme_classic() +
  # add text to plot
  geom_text(data = sig_levels_repeated,
            # make bold
            fontface = "bold",
            # define where labels should go
            aes(x = Visits,
                # define height of label
                y = 2,
                # what are the labels?
                label = paste0(Significance))) +
  xlab("Visits") +
  ylab("Measurement") +
  # remove legend - not needed here
  theme(legend.position = "none",
        # make label text vertical, easier to read
        axis.text.x = element_text(angle = 90,
                                   # vertical offset of text
                                   vjust = 0.5,
                                   # text size
                                   size = 12))
```



10.2.3 Similar ANOVA

Mathematically, a factorial ANOVA is the same as a randomized block ANOVA; please see that section for information on how to run this test.

10.2.4 ANOVA with interaction

Sometimes when we running a model, we want to look for *interactive effects*. Interactive effects are situations where one (or both) variables on their own do not effect the data, but there is a cumulative effect between variables that effects things. Let's look at an example, based on our initial example but with the data altered.

```
set.seed(8675309)

# we are using data from the randomized black ANOVA again

data_expanded
```

```
  Blocking_Variable Explanatory_1 Explanatory_2 Explanatory_3
1           Group 1          10.1            12          11.2
2           Group 2           9.4            13          10.1
3           Group 3          11.1          15.4          11.9
```

```r
### YOU DO NOT NEED TO DO THIS
### CREATING DATA FOR EXAMPLE

data_expanded$Explanatory_1 <- as.numeric(data_expanded$Explanatory_1)
data_expanded$Explanatory_2 <- as.numeric(data_expanded$Explanatory_2)
data_expanded$Explanatory_3 <- as.numeric(data_expanded$Explanatory_3)

# create some pseudorandom data
# [,-1] excludes first column - group data
data_expanded2 <- cbind(Blocking_Variable,
                        data_expanded[,-1] - 0.75)
data_expanded3 <- cbind(Blocking_Variable,
                        data_expanded[,-1]*1.05)

data_expanded <- rbind(data_expanded,
                       data_expanded2,
                       data_expanded3)

# expand to "long" format
data <- data_expanded |>
  # convert to data frame
  as.data.frame() |>
  # !by column for aggregating
  # names_to = what to name column aggregation
  # values_to = what the measurements should be called
  pivot_longer(!Blocking_Variable, names_to = "Treatments", values_to = "Measurements")

# specifying factor to be safe

interactive_aov <- aov(Measurements ~ factor(Treatments) +
                         factor(Blocking_Variable) +
                         factor(Treatments)*factor(Blocking_Variable),
                       data)

summary(interactive_aov)
```

```
                                          Df Sum Sq Mean Sq F value   Pr(>F)
factor(Treatments)                         2  53.28  26.640  59.680 1.14e-08
factor(Blocking_Variable)                  2  21.18  10.588  23.719 9.01e-06
factor(Treatments):factor(Blocking_Variable)  4   7.39   1.848   4.141    0.015
Residuals                                 18   8.03   0.446
```

```
factor(Treatments)                             ***
factor(Blocking_Variable)                      ***
factor(Treatments):factor(Blocking_Variable) *
Residuals
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we can see above, we have very significant effects for `Treatment` and `Blocking_Variable`, but a less significant effect for the interaction between the two. **Remember - we only need to plot our primary hypothesis**. **Note** however, that Tukey gives us our differences and p values for each set of tests and comparisons:

```
TukeyHSD(interactive_aov)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Measurements ~ factor(Treatments) + factor(Blocking_Variable) + factor(Tre

$`factor(Treatments)`
                                   diff         lwr       upr     p adj
Explanatory_2-Explanatory_1  3.3211111  2.51730768  4.124915 0.0000000
Explanatory_3-Explanatory_1  0.8811111  0.07730768  1.684915 0.0304521
Explanatory_3-Explanatory_2 -2.4400000 -3.24380343 -1.636197 0.0000011

$`factor(Blocking_Variable)`
                      diff        lwr       upr     p adj
Group 2-Group 1 -0.2711111 -1.0749145 0.5326923 0.6710682
Group 3-Group 1  1.7283333  0.9245299 2.5321368 0.0000928
Group 3-Group 2  1.9994444  1.1956410 2.8032479 0.0000159

$`factor(Treatments):factor(Blocking_Variable)`
                                                          diff          lwr
Explanatory_2:Group 1-Explanatory_1:Group 1  1.931667e+00  0.02027781
Explanatory_3:Group 1-Explanatory_1:Group 1  1.118333e+00 -0.79305553
Explanatory_1:Group 2-Explanatory_1:Group 1 -7.116667e-01 -2.62305553
Explanatory_2:Group 2-Explanatory_1:Group 1  2.948333e+00  1.03694447
Explanatory_3:Group 2-Explanatory_1:Group 1 -1.776357e-15 -1.91138886
Explanatory_1:Group 3-Explanatory_1:Group 1  1.016667e+00 -0.89472219
Explanatory_2:Group 3-Explanatory_1:Group 1  5.388333e+00  3.47694447
Explanatory_3:Group 3-Explanatory_1:Group 1  1.830000e+00 -0.08138886
Explanatory_3:Group 1-Explanatory_2:Group 1 -8.133333e-01 -2.72472219
```

```
Explanatory_1:Group 2-Explanatory_2:Group 1 -2.643333e+00 -4.55472219
Explanatory_2:Group 2-Explanatory_2:Group 1  1.016667e+00 -0.89472219
Explanatory_3:Group 2-Explanatory_2:Group 1 -1.931667e+00 -3.84305553
Explanatory_1:Group 3-Explanatory_2:Group 1 -9.150000e-01 -2.82638886
Explanatory_2:Group 3-Explanatory_2:Group 1  3.456667e+00  1.54527781
Explanatory_3:Group 3-Explanatory_2:Group 1 -1.016667e-01 -2.01305553
Explanatory_1:Group 2-Explanatory_3:Group 1 -1.830000e+00 -3.74138886
Explanatory_2:Group 2-Explanatory_3:Group 1  1.830000e+00 -0.08138886
Explanatory_3:Group 2-Explanatory_3:Group 1 -1.118333e+00 -3.02972219
Explanatory_1:Group 3-Explanatory_3:Group 1 -1.016667e-01 -2.01305553
Explanatory_2:Group 3-Explanatory_3:Group 1  4.270000e+00  2.35861114
Explanatory_3:Group 3-Explanatory_3:Group 1  7.116667e-01 -1.19972219
Explanatory_2:Group 2-Explanatory_1:Group 2  3.660000e+00  1.74861114
Explanatory_3:Group 2-Explanatory_1:Group 2  7.116667e-01 -1.19972219
Explanatory_1:Group 3-Explanatory_1:Group 2  1.728333e+00 -0.18305553
Explanatory_2:Group 3-Explanatory_1:Group 2  6.100000e+00  4.18861114
Explanatory_3:Group 3-Explanatory_1:Group 2  2.541667e+00  0.63027781
Explanatory_3:Group 2-Explanatory_2:Group 2 -2.948333e+00 -4.85972219
Explanatory_1:Group 3-Explanatory_2:Group 2 -1.931667e+00 -3.84305553
Explanatory_2:Group 3-Explanatory_2:Group 2  2.440000e+00  0.52861114
Explanatory_3:Group 3-Explanatory_2:Group 2 -1.118333e+00 -3.02972219
Explanatory_1:Group 3-Explanatory_3:Group 2  1.016667e+00 -0.89472219
Explanatory_2:Group 3-Explanatory_3:Group 2  5.388333e+00  3.47694447
Explanatory_3:Group 3-Explanatory_3:Group 2  1.830000e+00 -0.08138886
Explanatory_2:Group 3-Explanatory_1:Group 3  4.371667e+00  2.46027781
Explanatory_3:Group 3-Explanatory_1:Group 3  8.133333e-01 -1.09805553
Explanatory_3:Group 3-Explanatory_2:Group 3 -3.558333e+00 -5.46972219
                                                       upr     p adj
Explanatory_2:Group 1-Explanatory_1:Group 1  3.84305553 0.0464895
Explanatory_3:Group 1-Explanatory_1:Group 1  3.02972219 0.5317567
Explanatory_1:Group 2-Explanatory_1:Group 1  1.19972219 0.9173224
Explanatory_2:Group 2-Explanatory_1:Group 1  4.85972219 0.0010200
Explanatory_3:Group 2-Explanatory_1:Group 1  1.91138886 1.0000000
Explanatory_1:Group 3-Explanatory_1:Group 1  2.92805553 0.6439270
Explanatory_2:Group 3-Explanatory_1:Group 1  7.29972219 0.0000003
Explanatory_3:Group 3-Explanatory_1:Group 1  3.74138886 0.0667267
Explanatory_3:Group 1-Explanatory_2:Group 1  1.09805553 0.8457970
Explanatory_1:Group 2-Explanatory_2:Group 1 -0.73194447 0.0032271
Explanatory_2:Group 2-Explanatory_2:Group 1  2.92805553 0.6439270
Explanatory_3:Group 2-Explanatory_2:Group 1 -0.02027781 0.0464895
Explanatory_1:Group 3-Explanatory_2:Group 1  0.99638886 0.7519790
Explanatory_2:Group 3-Explanatory_2:Group 1  5.36805553 0.0001577
Explanatory_3:Group 3-Explanatory_2:Group 1  1.80972219 0.9999999
```

```
Explanatory_1:Group 2-Explanatory_3:Group 1  0.08138886 0.0667267
Explanatory_2:Group 2-Explanatory_3:Group 1  3.74138886 0.0667267
Explanatory_3:Group 2-Explanatory_3:Group 1  0.79305553 0.5317567
Explanatory_1:Group 3-Explanatory_3:Group 1  1.80972219 0.9999999
Explanatory_2:Group 3-Explanatory_3:Group 1  6.18138886 0.0000097
Explanatory_3:Group 3-Explanatory_3:Group 1  2.62305553 0.9173224
Explanatory_2:Group 2-Explanatory_1:Group 2  5.57138886 0.0000766
Explanatory_3:Group 2-Explanatory_1:Group 2  2.62305553 0.9173224
Explanatory_1:Group 3-Explanatory_1:Group 2  3.63972219 0.0947841
Explanatory_2:Group 3-Explanatory_1:Group 2  8.01138886 0.0000000
Explanatory_3:Group 3-Explanatory_1:Group 2  4.45305553 0.0047491
Explanatory_3:Group 2-Explanatory_2:Group 2 -1.03694447 0.0010200
Explanatory_1:Group 3-Explanatory_2:Group 2 -0.02027781 0.0464895
Explanatory_2:Group 3-Explanatory_2:Group 2  4.35138886 0.0069891
Explanatory_3:Group 3-Explanatory_2:Group 2  0.79305553 0.5317567
Explanatory_1:Group 3-Explanatory_3:Group 2  2.92805553 0.6439270
Explanatory_2:Group 3-Explanatory_3:Group 2  7.29972219 0.0000003
Explanatory_3:Group 3-Explanatory_3:Group 2  3.74138886 0.0667267
Explanatory_2:Group 3-Explanatory_1:Group 3  6.28305553 0.0000070
Explanatory_3:Group 3-Explanatory_1:Group 3  2.72472219 0.8457970
Explanatory_3:Group 3-Explanatory_2:Group 3 -1.64694447 0.0001097
```

I do not plot this here, but it would be similar to the other parts of this test.

10.3 Friedman's test

10.3.1 Using *R*

Friedman's test is a non-parametric alternative to a two-way ANOVA, so as you would guess, it can be painful to implement. We will use an altered version of the same test we've used before:

```
# set seed - make reproducible
set.seed(8675309)

### DO NOT NEED TO REPEAT THIS
### CREATING DATA FOR EXAMPLE

# new set of foods - this time, ten of them
Treatments <- c(paste("Treatment",1:10)) |>
  as.factor()
```

```r
# pre-created data frame of locations from earlier
Blocking_Factor <- c(paste("Block", 1:10)) |>
  as.factor()

long_data <- crossing(Blocking_Factor, Treatments)

long_data$Measurements <- NA

for(i in 1:length(unique(long_data$Treatments))){
  subset_rows <- which(long_data$Treatments==long_data$Treatments[i])
  long_data$Measurements[subset_rows] <- runif(n = length(subset_rows),
                                               min = i-2, max = i+2) |>
    round(1)
}

long_data
```

```
# A tibble: 100 x 3
   Blocking_Factor Treatments   Measurements
   <fct>           <fct>               <dbl>
 1 Block 1         Treatment 1          -0.4
 2 Block 1         Treatment 10          3.4
 3 Block 1         Treatment 2           1.2
 4 Block 1         Treatment 3           5.9
 5 Block 1         Treatment 4           6.9
 6 Block 1         Treatment 5           7
 7 Block 1         Treatment 6           7.6
 8 Block 1         Treatment 7           7.6
 9 Block 1         Treatment 8           8.1
10 Block 1         Treatment 9           9.8
# i 90 more rows
```

Now that we have our expanded and randomized table, we can get started with our test.

Our calculation for the Friedman's test statistic Q (not to be confused with Tukey's q!) is:

$$Q = \frac{12}{nk(k+1)} \cdot \Sigma R_j^2 - 3n(k+1)$$

where n is the total number of individuals in each sample in the dataset, k is the number of groups, and R_j^2 is the sum of the ranks.

In this class, we will do this in *R*.

```r
friedman_long_data <- friedman.test(y = long_data$Measurements,
                                    groups = long_data$Treatments,
                                    blocks = long_data$Blocking_Factor)

print(friedman_long_data)
```

```
    Friedman rank sum test

data:  long_data$Measurements, long_data$Treatments and long_data$Blocking_Factor
Friedman chi-squared = 79.983, df = 9, p-value = 1.629e-13
```

Note you will get a different answer if you are switching the blocks and the groups.

We can use the following, from package `PMCMRplus`, to find the adjacent and non-adjacent groups.

```r
# find differences
frdAllPairsConoverTest(y = long_data$Measurements,
                       groups = long_data$Treatments,
                       blocks = long_data$Blocking_Factor,
                       p.adjust.method = "bonf")
```

```
    Pairwise comparisons using Conover's all-pairs test for a two-way balanced complete bloc

data: y, groups and blocks

             Treatment 1 Treatment 10 Treatment 2 Treatment 3 Treatment 4
Treatment 10 1.00000     -           -           -           -
Treatment 2  1.00000     1.00000     -           -           -
Treatment 3  0.00094     0.10167     0.80301     -           -
Treatment 4  3.8e-09     1.6e-06     3.4e-05     0.19031     -
Treatment 5  3.2e-11     1.6e-08     4.0e-07     0.00637     1.00000
Treatment 6  < 2e-16     9.2e-16     2.5e-14     1.5e-09     0.00042
Treatment 7  < 2e-16     < 2e-16     3.6e-16     2.0e-11     9.3e-06
Treatment 8  < 2e-16     < 2e-16     < 2e-16     6.1e-15     3.8e-09
Treatment 9  < 2e-16     < 2e-16     < 2e-16     < 2e-16     1.1e-12
             Treatment 5 Treatment 6 Treatment 7 Treatment 8
Treatment 10 -           -           -           -
```

```
Treatment 2  -          -          -          -
Treatment 3  -          -          -          -
Treatment 4  -          -          -          -
Treatment 5  -          -          -          -
Treatment 6  0.01886    -          -          -
Treatment 7  0.00063    1.00000    -          -
Treatment 8  4.0e-07    0.34617    1.00000    -
Treatment 9  1.4e-10    0.00094    0.02676    1.00000


P value adjustment method: bonferroni
```

As we can see, some pairs are inseparable and others are separable. We can now plot as for the other problems.

10.4 Homework: Chapter 12

For problems 12.1, 12.2, 12.3, 12.4, and 12.5, state your hypotheses in sentence form and mathematically. Then, identify the appropriate ANOVA and perform the analysis. If you reject the null, complete a Tukey test and plot your results, showing letters denoting each group. **Note** that 12.4 requires a Friedman's test, but all other problems require some form of ANOVA.

Next, for problems 12.7, 12.8, and 12.9, identify the appropriate test and justify your reasoning. State the null and alternative hypothesis in word form and mathematically, and perform your analysis. If you perform an ANOVA *and* you reject the null hypothesis, plot your results and label the groups by letter.

Remember, only plot the results if you reject your *primary hypothesis*.

11 Correlation & regression

```
# load required libraries

library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

11.1 Introduction

When we are comparing *two continuous variables*, we use two forms of tests: *correlation* to understand *if* there is a relationship between two variables, and *linear regression* to determine *what* that relationships is.

Remember - "if" is always correlation, and "what is it" is always linear regression when choosing a test for an exam.

11.2 Correlation

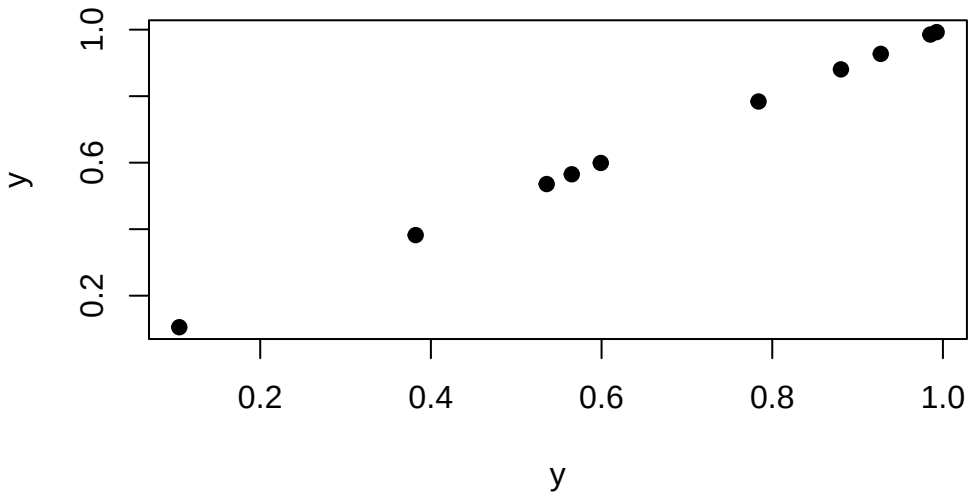
Correlation - denoted by ρ ("rho") and not to be confused with p - is a measure of how closely related to continuous variables appear to be. This can vary from a purely negative relationship to a purely positive relationship, such that $-1 \leq \rho \leq 1$ with $\rho = 0$ indicating a random relationship between data.

We can visualize these as follows:

```
### ILLUSTRATIVE PURPOSES ONLY

# create two random uniform distributions
y <- runif(10)
x <- runif(10)
```

For example, two things compared to themselves have a $\rho = 1$.

```
plot(y, y, pch = 19)
```



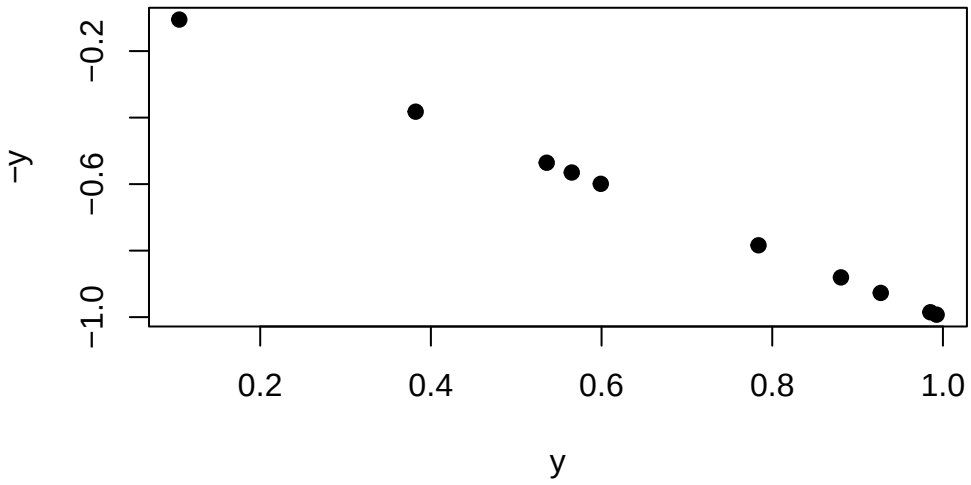

```
cor(y, y)
```

```
[1] 1
```

As we can see above, the correlation is 1.

Plotting by the negative will be a correlation of -1.

```
plot(y, -y, pch = 19)
```

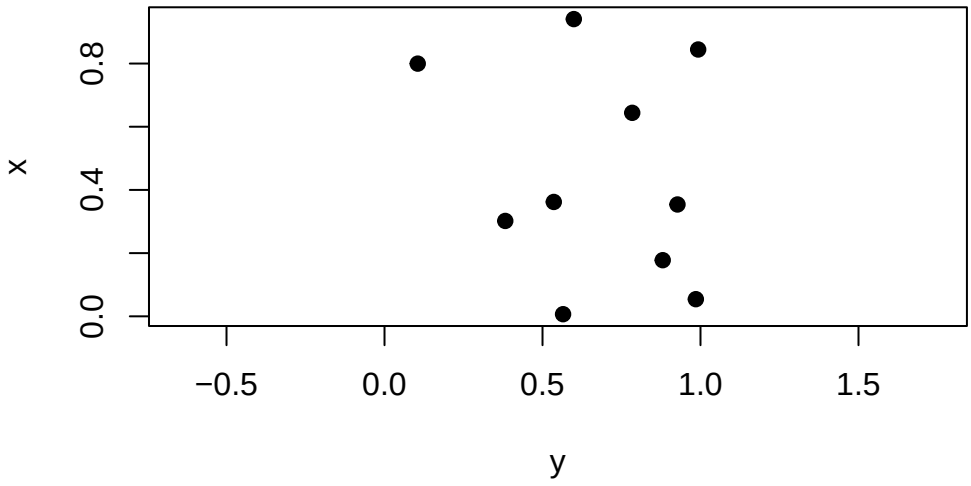



```
cor(y, -y)
```

```
[1] -1
```

Lastly, two random variables plotted against each other should have $\rho \approx 0$.

```
plot(y, x,
     pch = 19,
     asp = 1) # aspect ratio
```




```
cor(y, x) |>
  round(2)
```

```
[1] -0.21
```

11.2.1 Pearson's

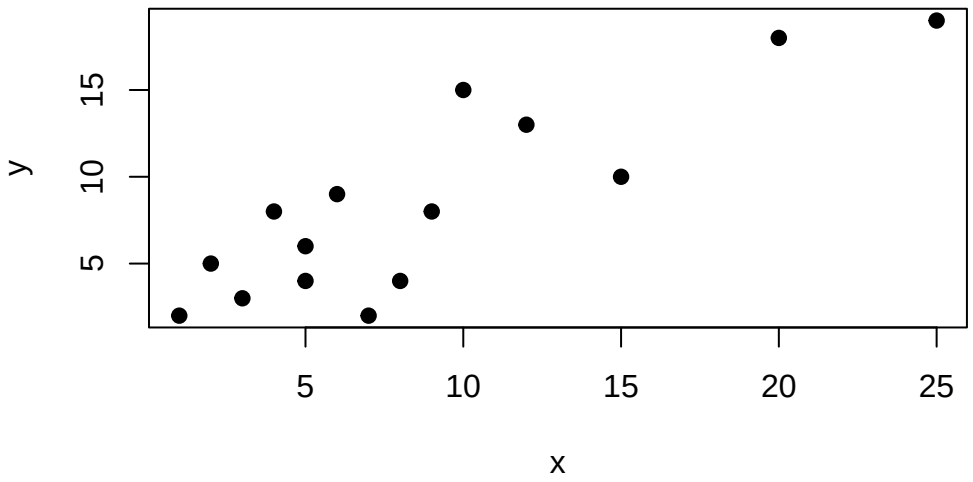
Pearson's correlation coefficient is our value for *parametric tests*. We often denote our correlation coefficient as r and not ρ for this particular test. It is calculated as follows:

$$r = \frac{\Sigma xy - (\frac{\Sigma x \Sigma y}{n})}{\sqrt{(\Sigma x^2 - \frac{(\Sigma x)^2}{n})(\Sigma y^2 - \frac{(\Sigma y)^2}{n})}}$$

where x is variable 1, y is variable 2, and n is the total number of data point pairs.

In this class, we will be using R to calculate r, which is done using the command `cor`. To ensure we are using the correct method, we need to set `method = "pearson"`.

```
set.seed(8675309)

### EXAMPLE DATA
x <- c(1,2,5,3,4,5,8,7,9,6,10,12,15,20,25)
y <- c(2,5,4,3,8,6,4,2,8,9,15,13,10,18,19)

plot(x, y, pch = 19)
```




```
cor(x, y, method = "pearson") |>
  round(2)
```

```
[1] 0.86
```

As we can see, these data are fairly positively correlated. As `x` increases, so does `y`. But how *significant* is this relationship?

Well, we can calculate two things - the *amount of variation explained*, which is r^2, and the *significance of the relationships*, which is determined via a t test and the equation $t = r\sqrt{\frac{n-2}{1-r^2}}$. This is a two-tailed distribution, with $df = n - 2$.

We can write a function to perform all of these options:

```
biol305_cor <- function(x=NA, y=NA, method = "pearson"){
  if(is.data.frame(x)==T){
    if(ncol(x)==2){
      r <- cor(x[,1], x[,2], method = method)
    }else{
      r <- cor(x, method = method)
    }

    r2 <- r
    r[r==1|r==-1] <- 0

    n <- 2*nrow(x)
  }else{
    r <- cor(x, y, method = method)

    n <- 2*length(x)
  }

  t_val <- r*sqrt((n-2)/(1-r^2))

  p <- pt(t_val, df = n - 2)

  p[p > 0.5] <- 1 - p[p > 0.5]
  p[p > 0.005] <- round(p[p > 0.005],2)
  p[p > 0.0005] <- round(p[p > 0.0005],3)
  p[p > 0.00005] <- round(p[p > 0.00005],4)
  p[p < 0.00005] <- "< 0.0001"

  if(is.data.frame(x)==T){
    print("Correlation:")
    print(round(r, 2))
    if(ncol(x) == 2){
      print(paste0("Degrees of freedom: ", n - 2))
      print(paste0("t value: ", round(t_val, 2)))
```

```
      print(paste0("P value: ", p))
    }
    if(ncol(x) > 2){
      print(paste0("Degrees of freedom: ", n - 2))
      print("")
      print("t value: ")
      print(round(t_val, 2))
      print("")
      print("P value: ")
      print(p)
    }
  }else{
    print(paste0("Correlation: ", round(r, 2)))
    print(paste0("Degrees of freedom: ", n - 2))
    print(paste0("t value: ", round(t_val, 2)))
    print(paste0("P value: ", p))
  }
}
```

Let's test our function.

```
biol305_cor(x, y, method = "pearson")
```

```
[1] "Correlation: 0.86"
[1] "Degrees of freedom: 28"
[1] "t value: 8.93"
[1] "P value: < 0.0001"
```

There we go! Our function printed out everything that we need.

11.2.2 Spearman's

Spearman's correlation is one of the non-parametric methods for our correlation tests. We can use this for ranked data or for non-parametric datasets. We do this the exact same way, except we change `method = "spearman"`.

11.2.3 Other non-parametric methods

To be expanded upon, but not necessary for the class at present.

11.3 Regression

Regression is used when we want to know *what* the relationship is between two variables. Regression operates similar to ANOVA and correlation, providing us with the nature of the relationship, the strength of the relationship, and gives us values for calculating the relationship. **For this class, we are only focusing on linear regression for relationships between linear variables**.

The equation for a regression line is often written as $y_i = \alpha + \beta x_i + e_i$, where α is the y intercept, β is the slope, and e is the error around each point. We will not perform regression calculations by hand in this class.

11.3.1 Parametric

We will use out previous established `x` and `y` datasets that are strongly positively correlated for this example. The equation for calculating a linear relationship is `lm`, which stands for "linear model". This uses equations like ANOVA, but can also use two vectors of data.

```
xy_linear <- lm(y ~ x)

summary(xy_linear)
```

```

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-5.1189 -1.4838 -0.5423  1.9757  5.7460

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.1370     1.2825   1.666     0.12
x             0.7117     0.1169   6.086 3.87e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.964 on 13 degrees of freedom
Multiple R-squared:  0.7402,    Adjusted R-squared:  0.7202
F-statistic: 37.04 on 1 and 13 DF,  p-value: 3.867e-05
```

As we can see, this returned an ANOVA table to use that tells us the value and significance of our *intercept* as well as the value and significance of the the slope (here, shown as `x`; it will always show the explanatory variable in this slot for the name).

Looking at the above, we can see that the slope is not significantly non-zero with a $p = 0.12$, but that the slope is significantly non-zero with $p < 0.0001$. We also have our R^2 values returned, which is similar to the r we got for correlation. Indeed, our correlation was $r = 0.86$, with $r^2 = 0.74$, which is very similar to the `Multiple R-squared` shown in the above ANOVA table.

R has a built in function within `ggplot` that will add a linear model to our plot and will show error regions as well. First, we need to make sure our data are in a `data.frame`.

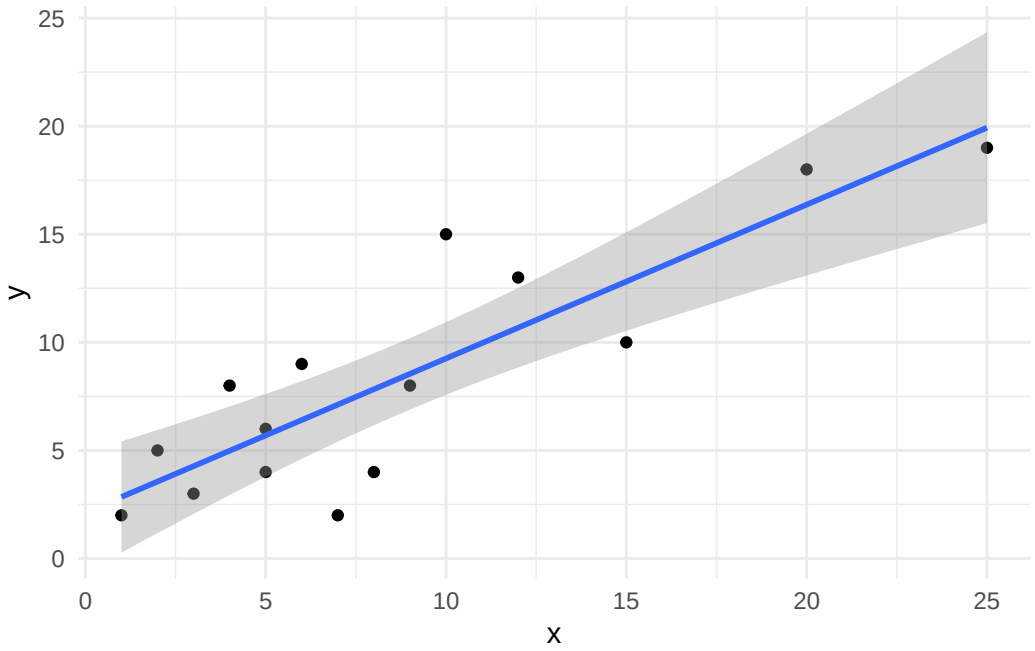
```
data <- data.frame(x, y)

data
```

```
    x  y
1   1  2
2   2  5
3   5  4
4   3  3
5   4  8
6   5  6
7   8  4
8   7  2
9   9  8
10  6  9
11 10 15
12 12 13
13 15 10
14 20 18
15 25 19
```

Next, we can plot the data in `ggplot`.

```
ggplot(data, aes(x = x, y = y)) +
  geom_point() +
  stat_smooth(method = "lm") +
  theme_minimal()
```

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



Just like that, we have created a plot of our linear regression. **Note** that you should always plot the lines only within the extent of the data; this is harder to do in other programs, but *R* does it for us!

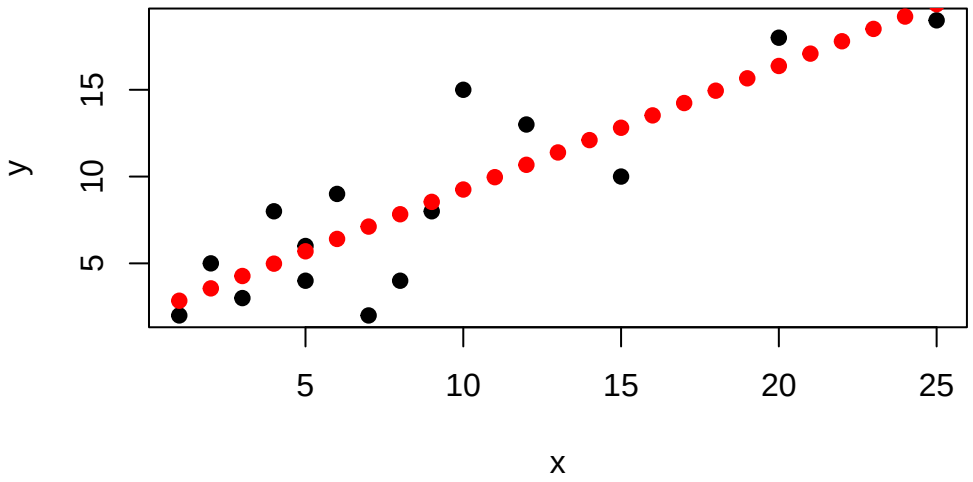
R can also allow us to predict different values using our linear model:

```
# must be in data frame format
test_data <- data.frame(1:25)

colnames(test_data) <- "x" # must be same as explanatory in data

predict(xy_linear, test_data)
```

```
       1        2        3        4        5        6        7        8
2.848692 3.560399 4.272105 4.983811 5.695517 6.407223 7.118929 7.830635
       9       10       11       12       13       14       15       16
8.542341 9.254047 9.965753 10.677460 11.389166 12.100872 12.812578 13.524284
      17       18       19       20       21       22       23       24
14.235990 14.947696 15.659402 16.371108 17.082814 17.794521 18.506227 19.217933
      25
19.929639
```

Let's visualize these points for illustration's sake.

```
plot(x, y, pch = 19)
points(1:25, as.numeric(predict(xy_linear, test_data)),
       pch = 19, col = "red")
```



As we can see above, our points follow the line from the plot. By using this format, however, we can make predictions of value for any point we want.

11.3.2 Non-parametric

We are not doing non-parametric linear regression in this class.

11.4 Homework

11.4.1 Chapter 13

11.4.2 Chapter 14

12 Pick the test

12.1 Picking the test

A major component of the final will be picking the correct test to run on some data. Here, I cover the specific ways in which you can determine what test to use.

Please use this web page as an *interactive* way to pick the test.

12.2 Exceptions

Below is a walk through for the most common statistical analyses, but keep in mind there are a few "less common" ones that we are using as well:

- **Binomial test** - if we are looking at something with discrete outcomes - like coin tosses, die rolls, etc. - we are doing a binomial test to determine the probability of a specific outcome. You can do this with `binom.test`.
- **Poisson test** - if we are looking at the probability of obtaining certain counts over events - specifically, looking at the probability of *rare* events - we will use a Poisson. You can do this with `poisson.test`.

12.3 Overview of picking the test

Below is an overview of picking basic statistical tests. There are more complex tests, but these are the main ones to consider for exams in this course.

Table 12.1: Which test to pick for which combinations of variables.

	Categorical Explanatory	Continuous Explanatory
Categorical Response	• χ^2 tests, especially if count data for category – Yate's correction auto-applied for 2x2 tables • Fisher test is a 2x2 table • Null is that counts match a known proportion (goodness-of-fit) or counts match each other (independence)	• Logistic regression, modeling categorical responses across continuous variables. – Not covered in BIOL 305 at present

	Categorical Explanatory	Continuous Explanatory
Continuous Response	• Single mean to population – This is a Z-score comparison, **when the population parameters are known** * $H_0 : \bar{x} = \mu$ – Almost always better to resort to t-test comparison if unsure is a population - t approaches a Z as $df \to \infty$ * $H_0 : \mu_1 = \mu_2$ • Two measurements – **Remember - all t-tests default to Welch's in *R*, assuming unequal variance** * When $\sigma_1^2 = \sigma_2^2$, Welch's t-test is the same as a standard t-test * Usually better to use Welch's, but justify reasoning – One-sample t-test, comparing the two groups * $H_0 : \mu_1 = \mu_2$ • Two samples – Two-sample t-test, comparing the two samples * $H_0 : \mu_1 = \mu_2$ • Paired samples – Special case - used when there are repeated measurements of the same sampling units * Need to set `paired = TRUE` * $H_0 : \mu_d = 0$ • ≥ 3 samples – ANOVA - analysis of variance across multiple samples, computing variance within samples and between samples * For all ANOVA, $H_0 : \mu_1 = \mu_2 = ...\mu_i$ * Single type of measurement is a one-way ANOVA * When things are "blocked" into categories - like litters - that are unique per row, use a randomized block ANOVA (remember to put `factor()` around anything you are worried won't be read as categorical!) · `aov(response ~ explanatory + block, data)` * When things are individuals being	• When looking at *if* there is a relationship, use correlation – $H_0 : \rho = 0$ – Evaluated using a t statistic • When looking at *what* the relationship is, use linear regression – Line formula is $y = \beta x + \alpha + \epsilon_i$ – Very similar to your high-school $y = mx + b$ – $H_0 : \beta = 0$ • Remember - linear regression works like an ANOVA, and has similar outputs – Check the ANOVA page for more information!

12.4 Another method - checklist

Feel free to go through the following headings to also help you pick a test.

12.4.1 Explanatory Variable

The explanatory variable is continuous and numeric. Pick this one for a continuous measurement variable, for example, such as Longitude, concentration, or other ratio or integer data.

The explanatory variable is discontinuous and categorical. In these cases, the explanatory variable is a condition, like a control and a treatment.

12.4.2 Continuous explanatory variable

12.4.2.1 Continuous response variable

If you have a continuous response variable, then you need to see what the question is asking:

- **Is there a relationship?**

If you are looking at a problem and it is simply asking *if* there is a relationship, you are looking at a **correlation analysis**.

- **What is the relationship?**

If you are asking *what* the relationship is or looking to be able to *predict* a value based on what you know, you are doing a **linear regression**. Note there are other kinds of regression, but in this class, we focus on linear regression.

If we have multiple response variables, we can do multiple regression, which we do not cover here.

12.4.2.2 Discrete response variable

If you are looking at a discrete response variable, such as a state of 1 or 0 in response to a certain amount of stimulus, then you are doing a **logistic regression**. We did not cover this analysis in this class.

12.4.3 Discrete explanatory variable

For discrete explanatory variables, we are often looking at categorical treatments or distinct groups, like species or geographic locations.

12.4.3.1 Continuous response variable

For a continuous response variable, we need to ask ourselves how many categories we are dealing with.

- If we are dealing with **two categories** or **two measurements from the same individual**, we are using a ***t*-test**.

Make sure you check the *t*-test page to understand what kind of *t*-test is being performed. For repeated measurements from the same individuals or populations under different conditions, we have the **paired *t*-test**. Otherwise, we have the **Welch's *t*-test** as the default in *R*. *NOTE* that the default assumes unequal variance; you must set `var.equal = TRUE` to perform a **"true" *t*-test**. Make sure you familiarize yourself with *why* our code is assuming variances are unequal.

If we *know* what the population is, we will use a Z-score, but bear in mind that we will almost always use a *t*-test to account for error. A *t*-test with infinite degrees of freedom is the same as a Z-score, so it is better to default to a *t*-test.

- If we have **three or more categories or treatments** then we need to perform an **ANOVA**.

If we are simply comparing multiple groups, we are performing a one-way ANOVA. We also need to make sure we aren't doing some sort of factorial ANOVA, repeated-measures ANOVA, or interactive ANOVA. Please read the ANOVA page to ensure you are using the correct format.

Remember to label ANOVA plots with letters to indicate the separate groups.

If we have multiple response variables, we can use a MANOVA; we do not cover that in this class.

12.4.3.2 Discrete response variable

If our response variable *and* our explanatory variable are discrete (i.e., categorical or nominal), then we are doing a χ^2 **test**. This tests looks at counts in different categories. For example, looking at proportions of men and women who do and do not smoke would be a classic χ^2 test.

13 Final exam & review

13.1 Introduction

These are questions that are posed as they would be on a final. Please complete each part of each question; we will review the answers. The following data are either imaginary or pulled from publicly available sources like Wikipedia.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

13.2 Happiness

You decide to ask your friends to rate their happiness on a scale of 0-100 before and after you bother them to ask how happy they are, thereby inciting an existential crises within each of them. You predict that asking this question will decrease the happiness they are feeling at that moment. These data are ranked data, but assume they are parametric given the size of the scale being used.

```
before_asking <- c(96,80,86,92,100,92,95,91,87)
after_asking <- c(72,53,90,85,90,86,83,79,62)

happiness <- cbind(before_asking, after_asking) |>
  as.data.frame()
```

```
happiness
```

```
  before_asking after_asking
1            96           72
2            80           53
3            86           90
4            92           85
5           100           90
6            92           86
7            95           83
8            91           79
9            87           62
```

13.2.1 What are the null and statistical hypotheses of this study?

13.2.2 What statistical test should be used for this test? Justify your answer, and be specific.

13.2.3 Calculate the appropriate test statistic, showing *all* or your work.

13.2.4 Assume that $\alpha = 0.05$. State your final conclusion from the survey.

13.3 Running and grades

You hear that going running is good for your grades. You decide to look and see if the amount of time people spend running is related to their grades. You obtain the following data:

```
run_time <- c(20,25,22,50,30,10,6)
grade <- c(94,84,95,72,88,90,85)

run_grades <- cbind(run_time, grade) |>
  as.data.frame()

run_grades
```

```
  run_time grade
1       20    94
2       25    84
3       22    95
```

```
4        50     72
5        30     88
6        10     90
7         6     85
```

13.3.1 What is the appropriate analysis for this question?

13.3.2 Perform the test. What is the *test statistic*?

13.3.3 State your conclusion about this scenario, using $\alpha = 0.05$.

13.4 Fosbury Flop

High-jumpers use the "Fosbury Flop" because it improves their performance by allowing their center of mass to pass *under* the high-jump bar while their bodies pass *over* the bar. Below are jump heights (in meters) for world records from before the Fosbury Flop was widely used and after the Fosbury Flop was widely used. Does the flop significantly improve athlete performace?

```
pre_flop <- c(2.09,2.12,2.15,2.18,2.17,2.28,2.29)
post_flop <- c(2.30,2.33,2.39,2.34,2.42,2.45,2.44)

jump_heights <- cbind(pre_flop, post_flop) |>
  as.data.frame()

jump_heights
```

```
  pre_flop post_flop
1     2.09      2.30
2     2.12      2.33
3     2.15      2.39
4     2.18      2.34
5     2.17      2.42
6     2.28      2.45
7     2.29      2.44
```

13.4.1 What is the appropriate test for this analysis? Justify your answer, and be specific.

13.4.2 Perform the test. What is the *test statistic*?

13.4.3 State your conclusion about this scenario. Set $\alpha = 0.05$.

13.5 Sandhills, Stonehills

You decide to look at concentrations of Greater Prairie-Chickens *Tympanuchus cupido* in fields a set amount of years after control burns. The following table shows the count of prairie-chickens in each field from the year of the burn until 5 years after the burn. Conditions are lettered to ensure sorting is performed correctly.

```
Location <- c(rep("Sandhills", 4), rep("Stonehills", 4))
a_burn_year <- c(1,0,3,2,1,0,0,2)
b_one_year_post_burn <- c(3,5,7,6,5,3,2,5)
c_five_years_post_burn <- c(20,21,15,8,8,7,10,8)

prairie_chickens <- cbind(Location, a_burn_year, b_one_year_post_burn, c_five_years_post_burn
  as.data.frame()

prairie_chickens
```

```
    Location a_burn_year b_one_year_post_burn c_five_years_post_burn
1  Sandhills           1                    3                     20
2  Sandhills           0                    5                     21
3  Sandhills           3                    7                     15
4  Sandhills           2                    6                      8
5 Stonehills           1                    5                      8
6 Stonehills           0                    3                      7
7 Stonehills           0                    2                     10
8 Stonehills           2                    5                      8
```

13.5.1 What is the appropriate test?

13.5.2 What is / are the explanatory variables?

13.5.3 Perform the appropriate test. Make a graph if necessary.

13.5.4 State your conclusions about this test. Set $\alpha = 0.05$.

14 Conclusions

14.1 Parting thoughts

In this class, we have covered two major things: (1) the basics of statistics for biological research and (2) the basics of using *R* to solve different computational problems. It is my hope that this class helps you both become a better researcher and also a more efficient researcher and student by using code to help you with your future projects.

14.2

(pronounced *doh-dah-dah-go-huh-ee*) is a traditional Cherokee farewell. It does not mean goodbye, but rather reflects a parting of ways until a group of folks meet again.

I enjoyed getting to know all of you in class, and please feel free to reach out or stop by and say hi if you are ever passing through Kearney in the future or if you need help with something biology related.

Wishing you the best,

Dr. Cooper

15 Functions & Glossary

15.1 Common Commands

The following are common useful commands used in *R*, with examples of their use.

- `<-` / `=` - save a value as an object

```
x <- 10
x
```

```
[1] 10
```

- `|>` - "pipe" a command or output into another command. You can use the shortcut `CTRL SHIFT M` on Windows or Mac.

```
# make repeatable
set.seed(930)

# random string
x <- rnorm(20)

x |>
  # pass to summary
  summary() |>
  # pass summary through round
  round(2)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  -1.94   -0.48   -0.06   -0.05    0.36    1.80
```

- `c` - concatenate, place two values together

```
x <- c(10,11)
x
```

```
[1] 10 11
```

15.2 Basic statistics

For these examples, we will create a random vector of number to demonstrate how they work.

```
x <- rnorm(1000)
```

- `mean` - get the mean / average of a set of data

```
mean(x)
```

```
[1] 0.04962364
```

15.3 Custom functions from class and elsewhere

The following functions are those developed for this class or adapted from code posted to sources like StackOverflow.

15.3.1 Basic stats

For these basic stats, we are using the following example data:

```
### EXAMPLE DATA ###
x <- c(1,2,3,5,5,4,6,8,7,9,5)
```

15.3.1.1 Mode calculations

```
# Based on Statology function
# define function to calculate mode
# works on vectors of data
find_mode <- function(x) {
  # get unique values from vector
  u <- unique(x)
  # count number of occurrences for each value
  tab <- tabulate(match(x, u))

  # if no mode, say so
  if(length(x)==length(u[tab == max(tab)])){
    print("No mode.")
```

```
  }else{
    # return the value with the highest count
    u[tab == max(tab)]
  }
}

find_mode(x)
```

```
[1] 5
```

15.3.1.2 Standard error

```
se <- function(x){
  n <- length(x) # calculate n
  s <- sd(x) # calculate standard deviation
  se_val <- s/sqrt(n)
  return(se_val)
}

se(x)
```

```
[1] 0.7385489
```

15.3.1.3 Coefficient of variation

```
cv <- function(x){
  sigma <- sd(x)
  mu <- mean(x)
  val <- sigma/mu
  return(val)
}

cv(x)
```

```
[1] 0.4898979
```

15.3.2 Normal distributions

15.3.2.1 Z score

Remember - in the Z-score code below, if no n is specified, then it will default to $n = 1$.

```
zscore <- function(xbar, mu, sd.x, n = 1){
  z <- (xbar - mu)/(sd.x/sqrt(n))
  return(z)
}

zscore(xbar = 62,
       mu = 65,
       sd.x = 3.5,
       n = 5)
```

```
[1] -1.91663
```

15.3.3 ANOVA

The following example data are going to be used to illustrate these functions.

```
#### EXAMPLE DATA ####
set.seed(8675309)

for(i in 1:4){
  x <- rnorm(10)
  if(i == 1){
    x <- rnorm(10, mean = 2)
    data <- x |> as.data.frame()
    colnames(data) <- "Response"
    data$Explanatory <- paste0("x",i)
  }else{
    newdat <- x |> as.data.frame()
    colnames(newdat) <- "Response"
    newdat$Explanatory <- paste0("x",i)
    data <- rbind(data,newdat)
  }
}

# split into "typical" table
```

```r
expanded_data <- NULL
expanded_data$x1 <- data$Response[which(data$Explanatory=="x1")]
expanded_data$x2 <- data$Response[which(data$Explanatory=="x2")]
expanded_data$x3 <- data$Response[which(data$Explanatory=="x3")]
expanded_data$x4 <- data$Response[which(data$Explanatory=="x4")]

expanded_data <- expanded_data |>
  as.data.frame()
```

The above is a *one-way ANOVA*. As a reminder, we would calculate the test as follows:

```r
# pivot longer does not work for one-way ANOVA, requires blocking factor
# can rbind things with same colnames to make longer

example_aov <- aov(Response ~ Explanatory, data)

summary(example_aov)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
Explanatory  3  23.40   7.801   11.54 1.89e-05 ***
Residuals   36  24.33   0.676
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Above, we can see a significant result of the ANOVA. We can follow this up with a Tukey test. **This requires the package `agricolae`!** Check the ANOVA pages, however, as not all ANOVA can use this `agricolae` shortcut method.

```r
example_tukey <- HSD.test(example_aov,
                          # what to group by?
                          "Explanatory",
                          # significance level?
                          alpha = 0.05,
                          # are data unbalanced
                          unbalanced = FALSE,
                          # show answer?
                          console = TRUE)
```

```
Study: example_aov ~ "Explanatory"
```

```
HSD Test for Response

Mean Square Error:  0.6758192

Explanatory,  means

     Response       std  r        se        Min      Max        Q25         Q50
x1  1.7620153 1.0505466 10 0.2599652  0.4504476 3.972459  1.1175485  1.44911720
x2  0.2841495 0.7532422 10 0.2599652 -0.4729986 1.985826 -0.3497379  0.21347543
x3  0.2197337 0.7019368 10 0.2599652 -0.6150452 1.574903 -0.2436023 -0.04493909
x4 -0.2890524 0.7345336 10 0.2599652 -1.9769014 0.684072 -0.5394534 -0.07741642
          Q75
x1 2.07491533
x2 0.64579865
x3 0.53544151
x4 0.04323417

Alpha: 0.05 ; DF Error: 36
Critical Value of Studentized Range: 3.808798

Minimun Significant Difference: 0.9901551

Treatments with the same letter are not significantly different.

     Response groups
x1  1.7620153      a
x2  0.2841495      b
x3  0.2197337      b
x4 -0.2890524      b
```

15.3.3.1 Summarize data (for plotting)

Remember - you need to change `"Explanatory"` to your explanatory variable (in quotes!) and you need to change `Response` to your response column (no quotes!). The following requires `plyr` to work, but the function itself should call up `plyr` if you do not yet have it loaded.

```
# summarize by group
summary_data <- function(data, explanatory){
  require(plyr)
  ddply(data, paste(explanatory), summarise,
                N = length(Response),
```

```
                  mean = mean(Response),
                  sd = sd(Response),
                  se = sd / sqrt(N))
}

example_summary <- summary_data(data = data, explanatory = "Explanatory")
```

```
Loading required package: plyr
```

```
------------------------------------------------------------------------------
```

```
You have loaded plyr after dplyr - this is likely to cause problems.
If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
library(plyr); library(dplyr)
```

```
------------------------------------------------------------------------------
```

```

Attaching package: 'plyr'
```

```
The following objects are masked from 'package:dplyr':

    arrange, count, desc, failwith, id, mutate, rename, summarise,
    summarize
```

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

    compact
```

```
example_summary
```

```
  Explanatory  N       mean        sd        se
1          x1 10  1.7620153 1.0505466 0.3322120
2          x2 10  0.2841495 0.7532422 0.2381961
3          x3 10  0.2197337 0.7019368 0.2219719
4          x4 10 -0.2890524 0.7345336 0.2322799
```

15.3.3.2 Significant label maker

This command requires a Tukey HSD object from `agricolae`. You can manually create a table like this for some other scenarios; see relevant pages for documentation.

```
# note first group must be EXACT MATCH to your summary_data object
# groups are saved in the Tukey object
# this is true for Tukey later as well

# the following is a function that will make the significant label table
sig.label.maker <- function(tukey_test, group_name){
  sig.labels <- tukey_test$groups |>
    # convert to a data.frame
    as.data.frame() |>
    # create a new column - place rownames into the column
    # converts to a format better for ggplot
    mutate(Explanatorys = rownames(tukey_test$groups)) |>
    # rename column to prevent confusion
    # specify dplyr; default function may be from plyr and not work
    dplyr::rename(Significance = groups)
  colnames(sig.labels)[which(colnames(sig.labels) == "Explanatorys")] <- group_name
  return(sig.labels)
}

# Function requires explanatory groups in quotes
sig_labels <- sig.label.maker(example_tukey, "Explanatory")

sig_labels
```

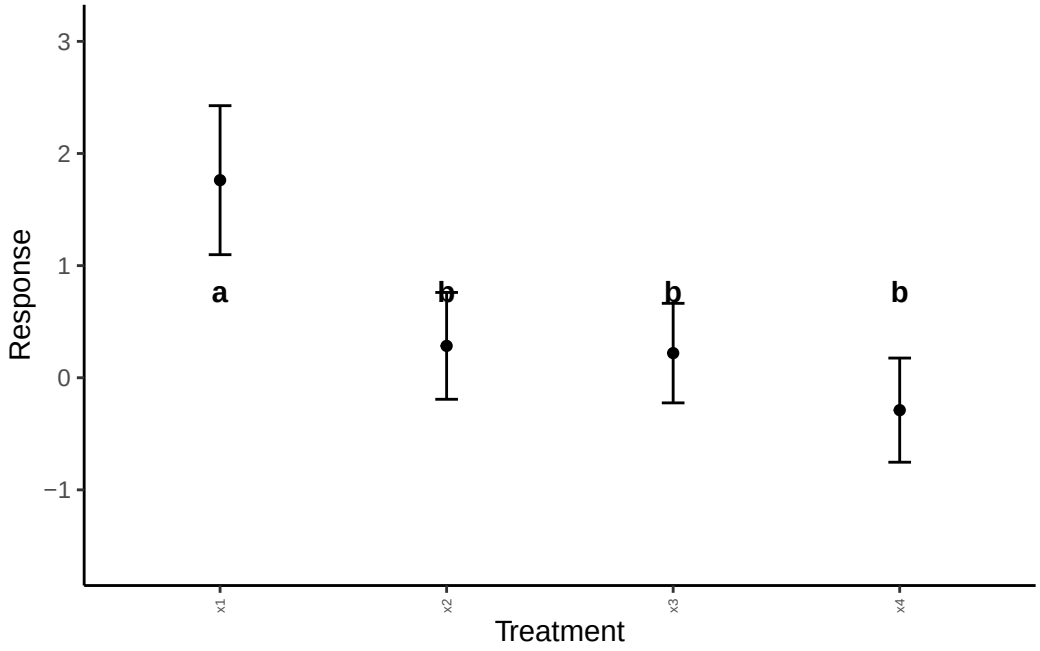
```
     Response Significance Explanatory
x1  1.7620153            a          x1
x2  0.2841495            b          x2
x3  0.2197337            b          x3
x4 -0.2890524            b          x4
```

15.3.3.3 ANOVA plotter

The following function plots ANOVAS if you have a `summary_data` object and a `sig_labels` object, as shown above. **This does not work on ANOVA with interactive components**.

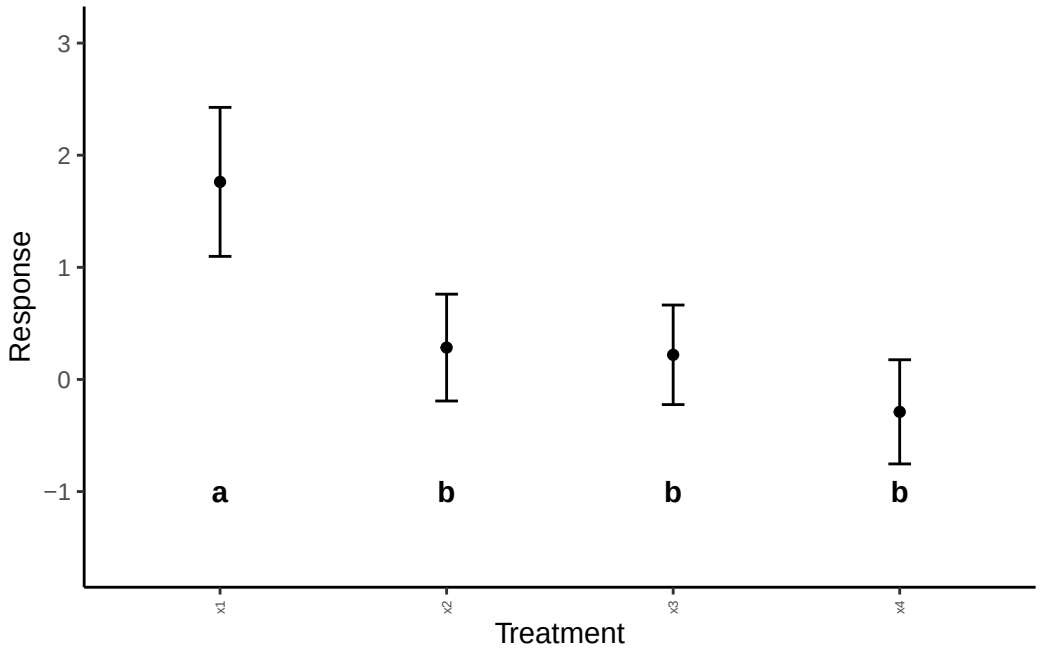
```
anova_plotter <- function(summary_data, explanatory,
                          response, sig_labels,
                          y_lim=NA, label_height=NA,
                          y_lab=NA, x_lab=NA){
  require(tidyverse)

  plot_data_1 <- summary_data[,c(explanatory, response, "se")]
  plot_data_2 <- sig_labels[,c(explanatory,"Significance")]

  colnames(plot_data_1) <- c("explanatory", "response", "se")
  colnames(plot_data_2) <- c("explanatory", "Significance")

  plot_data <- plot_data_1 |>
    full_join(plot_data_2, by = "explanatory")

  if(is.na(y_lim)){
    if(min(plot_data$response) < 0){
      y_lim <- c(min(plot_data$response) -
          4*max(plot_data$se),
        max(plot_data$response) +
          4*max(plot_data$se))
    }else{
      y_lim <- c(0,max(plot_data$response) +
                                4*max(plot_data$se))
    }
  }
  if(is.na(label_height)){label_height <- 0.25*max(y_lim)}
  if(is.na(y_lab)){y_lab <- "Response"}
  if(is.na(x_lab)){x_lab <- "Treatment"}

  plot_1 <- ggplot(plot_data,
         aes(x = explanatory, y = response)) +
    geom_point() +
    geom_errorbar(data = plot_data,
                  aes(ymin = response - 2*se,
                      ymax = response + 2*se,
                      width = 0.1)) +
    ylim(y_lim) +
    theme_classic() +
  theme(legend.position = "none",
        axis.text.x = element_text(angle = 90, vjust = 0.5, size = 5)) +
  geom_text(data = plot_data,
```

```
            # make bold
            fontface = "bold",
            # define where labels should go
            aes(x = explanatory,
                # define height of label
                y = label_height,
                # what are the labels?
                label = paste0(Significance))) +
  xlab(x_lab) +
  ylab(y_lab)

  print(plot_1)
}

anova_plotter(summary_data = example_summary,
              explanatory = "Explanatory",
              response = "mean", # from summary_data table! What is to be plotted
              sig_labels = sig_labels)
```



Note that in the above, the default label height is not working for us. We can adjust this with `label_height`.

```r
anova_plotter(summary_data = example_summary,
              explanatory = "Explanatory",
              response = "mean", # from summary_data table! What is to be plotted
              sig_labels = sig_labels,
              label_height = -1)
```



Much better!

References

Comont, R. (2020). BeeWalk dataset 2008-23. https://doi.org/10.6084/m9.figshare.12280547.v4

Cooper, J. C. (2021). Biogeographic and Ecologic Drivers of Avian Diversity. [Online.] Available at https://doi.org/10.6082/uchicago.3379.

Courtenay, L. (2019). Measurements on Canid Tooth Scores. https://doi.org/10.6084/m9.figshare.8081108.v1

Lydeamore, M. J., P. T. Campbell, D. J. Price, Y. Wu, A. J. Marcato, W. Cuningham, J. R. Carapetis, R. M. Andrews, M. I. McDonald, J. McVernon, S. Y. C. Tong, and J. M. McCaw (2020b). Patient ages at presentation. https://doi.org/10.1371/journal.pcbi.1007838.s006

Lydeamore, M. J., P. T. Campbell, D. J. Price, Y. Wu, A. J. Marcato, W. Cuningham, J. R. Carapetis, R. M. Andrews, M. I. McDonald, J. McVernon, S. Y. C. Tong, and J. M. McCaw (2020a). Estimation of the force of infection and infectious period of skin sores in remote Australian communities using interval-censored data. PLOS Computational Biology 16:e1007838.

Moura, R., N. P. Santos, and A. Rocha (2023). Processed csv file of the piracy dataset. https://doi.org/10.6084/m9.figshare.24119643.v1