

Biology 305: Biostatistics

Dr. Jacob C. Cooper & Dr. Melissa Wuellner

Invalid Date

Table of contents

Preface	6
1 Intro to <i>R</i>	7
2 Setup	8
2.1 Installing <i>R</i>	8
2.2 Installing <i>RStudio</i>	9
3 Creating an <i>RMarkdown</i> document	11
3.1 Setup	11
3.2 Using code chunks	14
3.3 Plotting	16
3.4 Tab complete	17
3.5 Help	17
4 Working with data	19
4.1 Downloading data	20
4.2 Subsetting data	23
5 Your turn!	26
6 Descriptive Statistics	27
6.1 Purposes of descriptive statistics	27
6.2 Preparing <i>R</i>	27
6.3 Downloading the data	28
6.4 Descriptive statistics	28
6.4.1 Notation	29
6.4.2 Mean	29
6.4.3 Range	30
6.4.4 Median	31
6.4.5 Other quartiles and quantiles	33
6.4.6 Mode	35
6.4.7 Variance	37
6.4.8 Standard deviation	39
6.4.9 Standard error	39
6.4.10 Coefficient of variation	40

6.4.11	Outliers	41
6.5	Homework: Descriptive Statistics	41
6.5.1	Homework instructions	42
6.5.2	Data for homework problems	42
7	Diagnosing data visually	46
7.1	The importance of visual inspection	46
7.2	Sample data and preparation	46
7.3	Histograms	47
7.3.1	Histograms on numeric vectors	47
7.3.2	Histograms on frequency counts	48
7.3.3	ggplot histograms	49
7.4	Skewness	54
7.5	Kurtosis	55
7.6	Cumulative frequency plot	57
7.6.1	If data are not in histogram/frequency format	57
7.6.2	If data are in histogram/frequency format	59
7.7	Homework: Chapter 3	60
7.7.1	Helpful hint	60
7.7.2	Directions	61
7.8	Addendum	61
8	Normality & hypothesis testing	62
8.1	Normal distributions	62
8.1.1	Example in nature	63
8.1.2	Effect of sampling	67
8.2	Hypothesis testing	69
8.2.1	Critical Values - α	70
8.2.2	Visualizing a p value	70
8.2.3	Calculating a p value	72
8.2.4	Workthrough Example	74
8.3	Confidence Intervals	77
8.4	Homework: Chapter 8	79
9	Exam 2 practice	80
9.1	Exam 2 Practice	80
9.2	Question 1: Cyclones	80
9.3	Question 2: Minnows	81
10	Probability distributions	83
10.1	Probability distributions	83
10.2	Binomial distribution	83
10.3	Poisson distribution	87

10.4	χ^2 -squared distribution	88
10.4.1	Calculating the test statistic	89
10.4.2	χ^2 estimations in <i>R</i>	90
10.4.3	Calculating χ^2 by hand	90
10.5	Fisher's exact test	90
10.6	Homework	90
10.6.1	Chapter 5	90
10.6.2	Chapter 7	90
11	Single population means testing	91
11.1	Introduction	91
11.2	<i>t</i> -distribution	91
11.3	<i>t</i> -tests	91
11.4	Wilcoxon tests	91
11.5	Confidence intervals	91
11.6	Homework: Chapter 9	91
12	Two sample tests	92
12.1	Introduction	92
12.2	<i>t</i> -tests	92
12.3	Mann-Whitney <i>U</i> tests	92
12.4	Error	92
12.5	Homework: Chapter 10	92
13	ANOVA: Part 1	93
13.1	Introduction	93
13.2	ANOVA: By hand	93
13.3	ANOVA: By <i>R</i>	93
13.4	Kruskal-Wallis tests	93
13.5	Homework: Chapter 11	93
14	ANOVA: Part 2	94
14.1	Two-way ANOVA	94
14.2	Designs	94
14.2.1	Randomized block design	94
14.2.2	Repeated measures	94
14.2.3	Factorial ANOVA	94
14.3	Friedman's test	94
14.4	Homework: Chapter 12	94
15	Correlation & regression	95
15.1	Introduction	95

15.2	Correlation	95
15.2.1	Pearson's	95
15.2.2	Spearman's	95
15.2.3	Other non-parametric methods	95
15.3	Correlation	95
15.3.1	Parametric	95
15.3.2	Non-parametric	95
15.4	Homework	95
15.4.1	Chapter 13	95
15.4.2	Chapter 14	95
16	Final exam & review	96
16.1	Pick the test	96
16.2	Final review	96
17	Conclusions	97
17.1	97
	References	98
18	Glossary	99
18.1	Common Commands	99

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

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*!).

2 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.

2.1 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.

2.2 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 to 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:

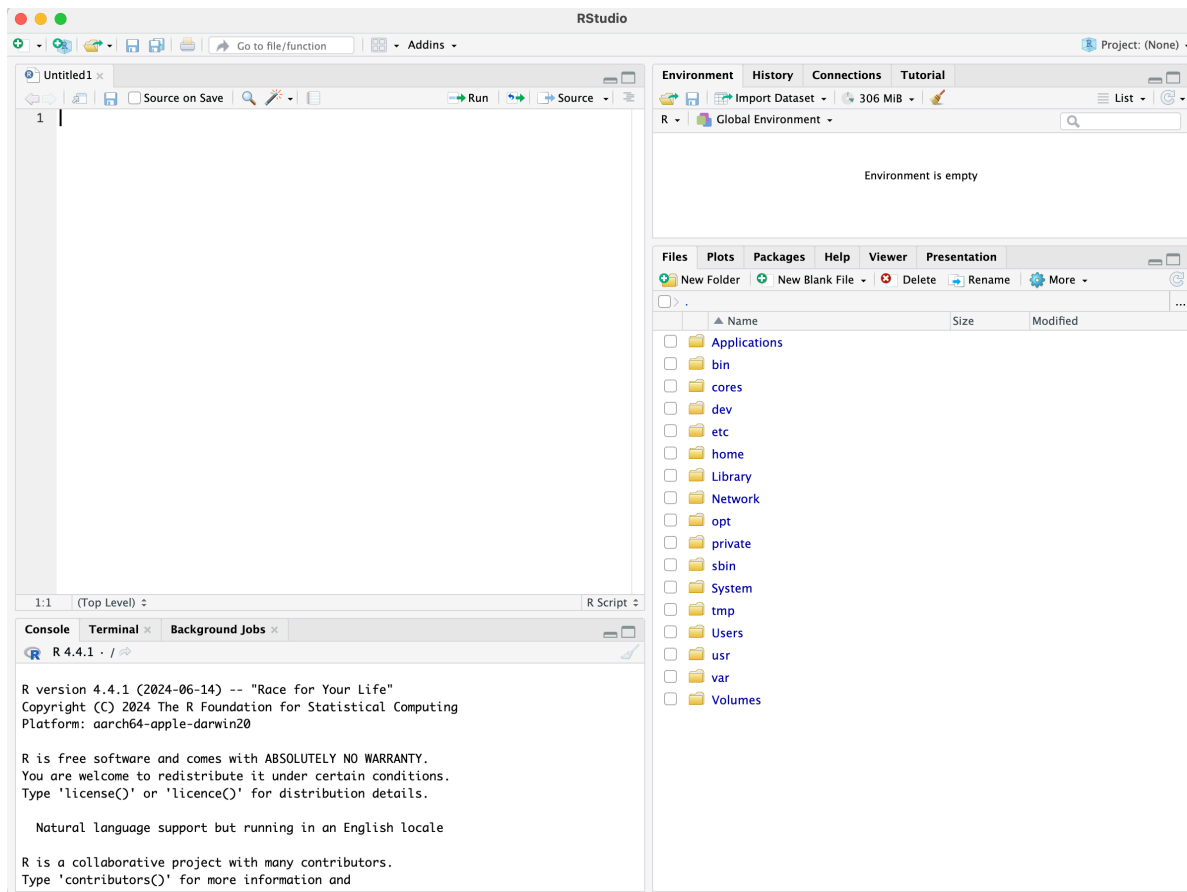


Figure 2.1: *RStudio* start window. Note that the screen is split into four different quadrants. Top left: *R* documents; bottom left: *R* program; top right: environment window; bottom right: plots, help, and directories.

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.

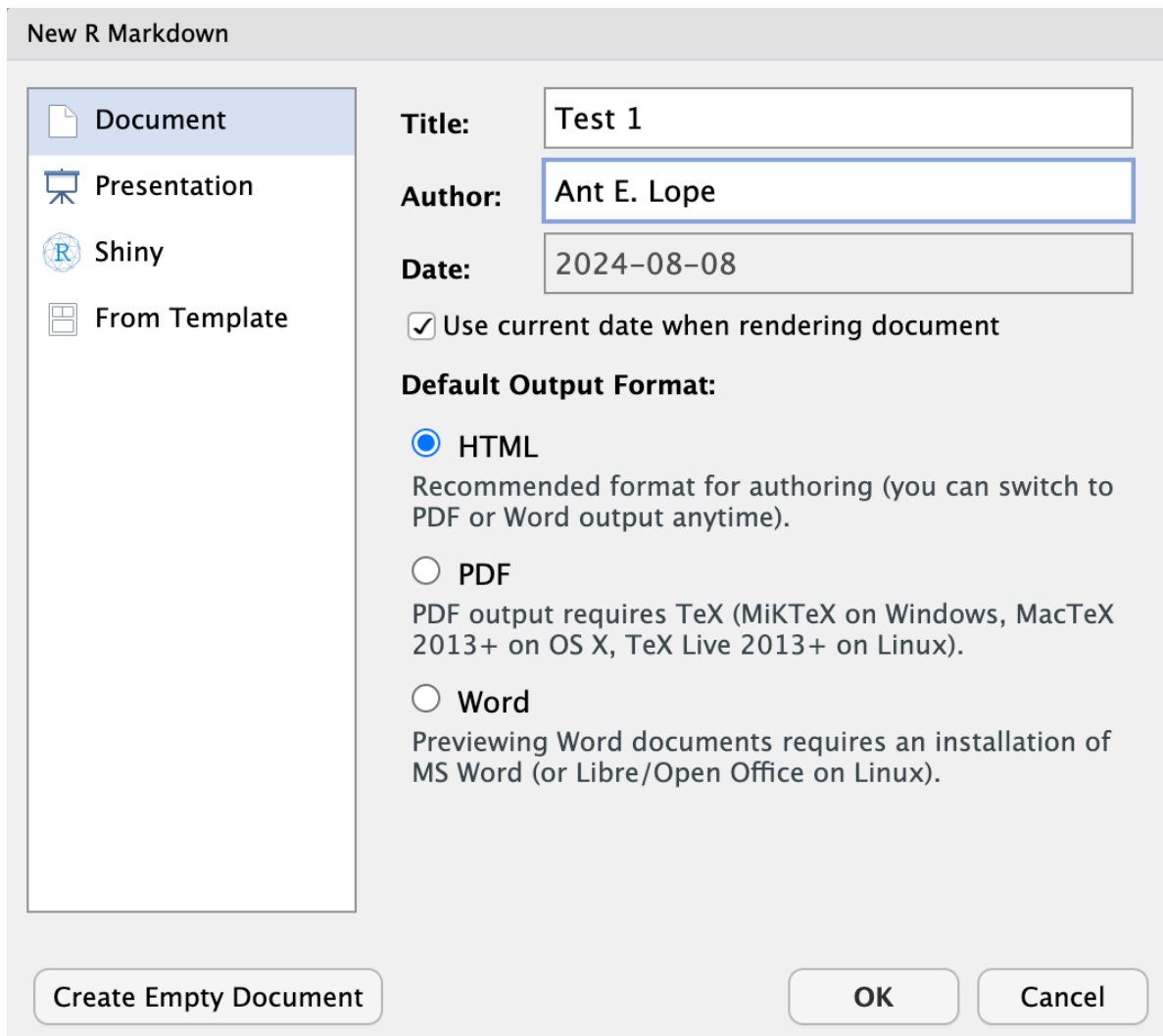
Now, we will create our first *R* document!

3 Creating an *RMarkdown* document

3.1 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:



The image shows a 'New R Markdown' dialog box. On the left is a vertical menu with four options: 'Document' (selected), 'Presentation', 'Shiny', and 'From Template'. Each option has a small icon to its left. To the right of the menu are three text input fields: 'Title:' with the value 'Test 1', 'Author:' with the value 'Ant E. Lope', and 'Date:' with the value '2024-08-08'. Below these fields is a checked checkbox labeled 'Use current date when rendering document'. Underneath is the section 'Default Output Format:' with three radio button options: 'HTML' (selected), 'PDF', and 'Word'. Each radio button option has a descriptive text block below it. The 'HTML' block says 'Recommended format for authoring (you can switch to PDF or Word output anytime)'. The 'PDF' block says 'PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux)'. The 'Word' block says 'Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux)'. At the bottom of the dialog are three buttons: 'Create Empty Document' on the left, and 'OK' and 'Cancel' on the right.

New R Markdown

Document

Presentation

Shiny

From Template

Title: Test 1

Author: Ant E. Lope

Date: 2024-08-08

☒ Use current date when rendering document

Default Output Format:

☒ **HTML**
Recommended format for authoring (you can switch to PDF or Word output anytime).

☐ **PDF**
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

☐ **Word**
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

Create Empty Document

OK Cancel

Figure 3.1: 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**.

```
---
title: "Test 1"
author: "Ant E. Lope"
date: "`r Sys.Date()`"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring
HTML, PDF, and MS Word documents. For more details on using R Markdown see
http://rmarkdown.rstudio.com.

When you click the Knit button a document will be generated that includes both
content as well as the output of any embedded R code chunks within the document. You can
embed an R code chunk like this:

```{r cars}
summary(cars)
```
```

Figure 3.2: 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.

3.2 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 *R*Markdown code.

```
```\n{r}\nprint("Hello world!")\n```
```

Figure 3.3: Text to type in your *R*markdown 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:

### Test 1

Ant E. Lope

2024-08-07

This is a test of the *R*Markdown code.

```
print("Hello world!")
```

```
[1] "Hello world!"
```

Figure 3.4: 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.

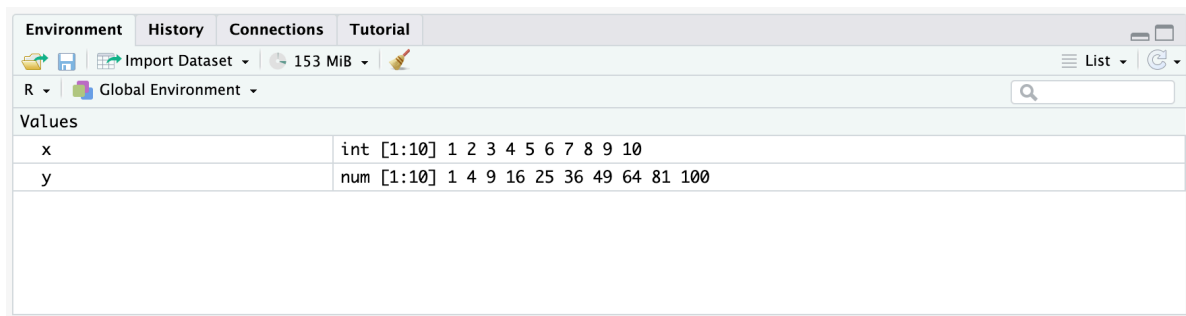


Figure 3.5: *RStudio* environment window showing saved objects. These are in the computer's memory.

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
```

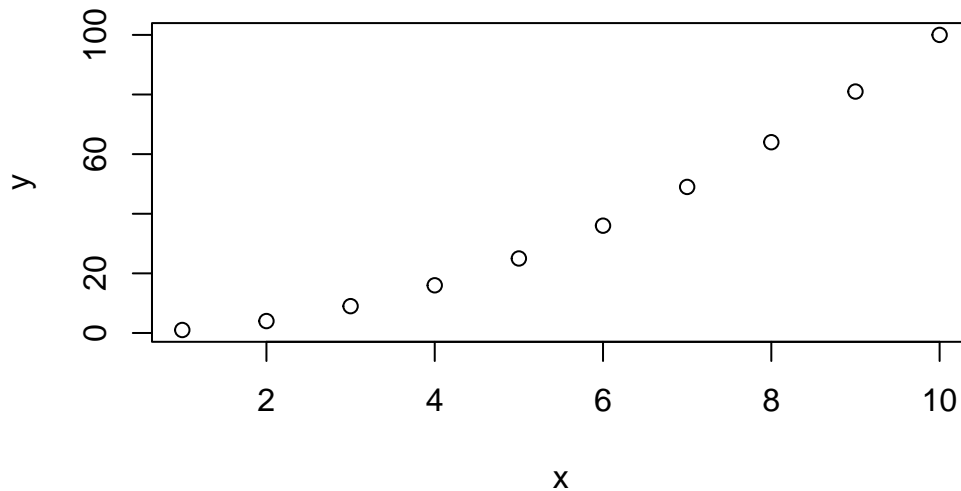
```
[1] 1 4 27 256 3125 46656
[7] 823543 16777216 387420489 10000000000
```

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

### 3.3 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
```



### 3.4 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

### 3.5 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

## 4 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 are two libraries that you will need *almost every week*! First, we need to install the libraries. The main libraries we need for this class are:

1. **curl**: this package allows us to download things from URLs. We will be using this to download data files. *Otherwise, you will have to enter all data by hand!*
2. **tidyverse**: this package is actually a [group of packages](#) designed to help with data analysis, management, and visualization.

```
run this code the first time ONLY
does not need to be run every time you use R!

curl allows for internet downloads
install.packages("curl")

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(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 1.9.3 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 become
```

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.

## 4.1 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](#).

```
first, declare filepath
I will try to give you the filepath for each assignment
if not, check the URL pattern for the file

create
ranges.url <- curl("https://raw.githubusercontent.com/jacobccooper/biol305_unk/main/datasets/la")
read comma separated file (csv) into R memory
ranges <- read_csv(ranges.url)
```

```
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 <- curl("https://raw.githubusercontent.com/jacobccooper/biol305_unk/main/datasets/la") %>%
 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`.

## 4.2 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
```

## 5 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.

# 6 Descriptive Statistics

## 6.1 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.

## 6.2 Preparing R

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

```
allows for internet downloading
library(curl)
```

Using libcurl 8.7.1 with LibreSSL/3.3.6

```
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 1.9.3 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 become
```

## 6.3 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 <- curl("https://raw.githubusercontent.com/jacobccooper/biol105_unk/main/datasets/starbucks.csv")
read_csv(starbucks)
```

```
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.

## 6.4 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).

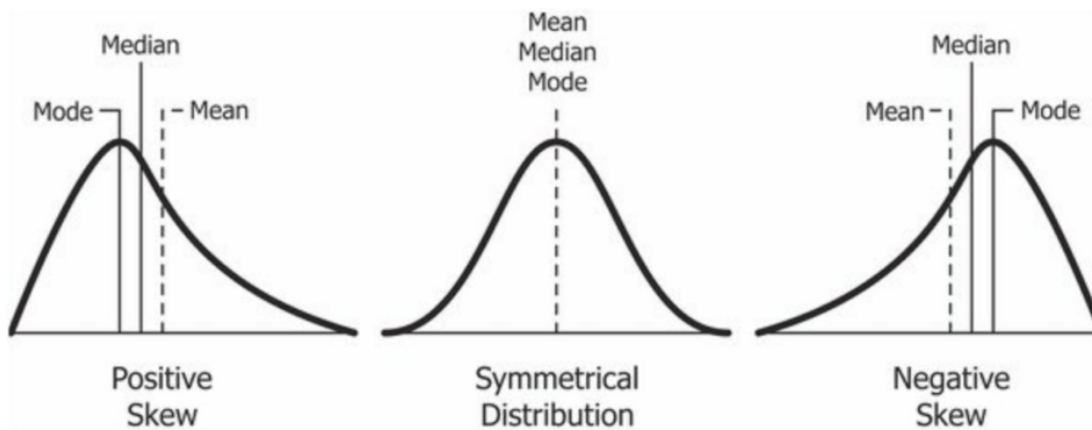


Figure 6.1: Examples of skew compared to a symmetrical, non-skewed distribution. Source: [machinelearningparatodos.com](http://machinelearningparatodos.com)

*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.

### 6.4.1 Notation

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

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

### 6.4.2 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
```

### 6.4.3 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.

### 6.4.4 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(flaps)
```

```
[1] 4
```

Now, to calculate by hand:

```
length(flaps)
```

```
[1] 9
```

We have an odd length.

```
order gets the order
order(flaps)
```

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

```
[] tells R which elements to put where
flap_order <- flaps[order(flaps)]
```

```
flap_order
```

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

```
always use parentheses
make sure the math is right!
(length(flap_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!



### 6.4.5 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
```

```

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**:

```
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.

### 6.4.6 Mode

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

```
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))
 # 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 `!=`.

### 6.4.7 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 squared 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.

```
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.

```
sum(diffs)
```

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

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

```
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  $n - 1$ .

```
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*.

```
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.

### 6.4.8 Standard deviation

Another common measurement of spread is the standard deviation ( $\sigma$ ). 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!

### 6.4.9 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:

```
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.

```
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*?

#### 6.4.10 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.

```
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.



### 6.4.11 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

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.

## 6.5 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.

### 6.5.1 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.**

### 6.5.2 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.

#### 6.5.2.1 1: UNK Nebraskans

Every year, the university keeps track of where students are from. The following are data on the number 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.

### 6.5.2.2 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`).

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

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,... dtm
(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.

### 6.5.2.3 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).

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

```
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?

## 7 Diagnosing data visually

### 7.1 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.

### 7.2 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 1.9.3 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 become
```

Next, we can download our data sample.

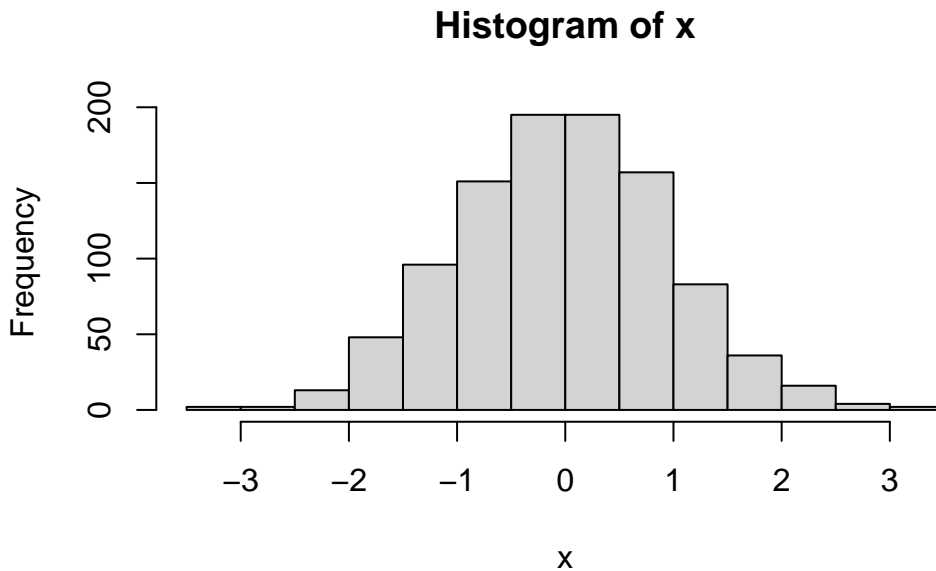
## 7.3 Histograms

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.

### 7.3.1 Histograms on numeric vectors

```
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)

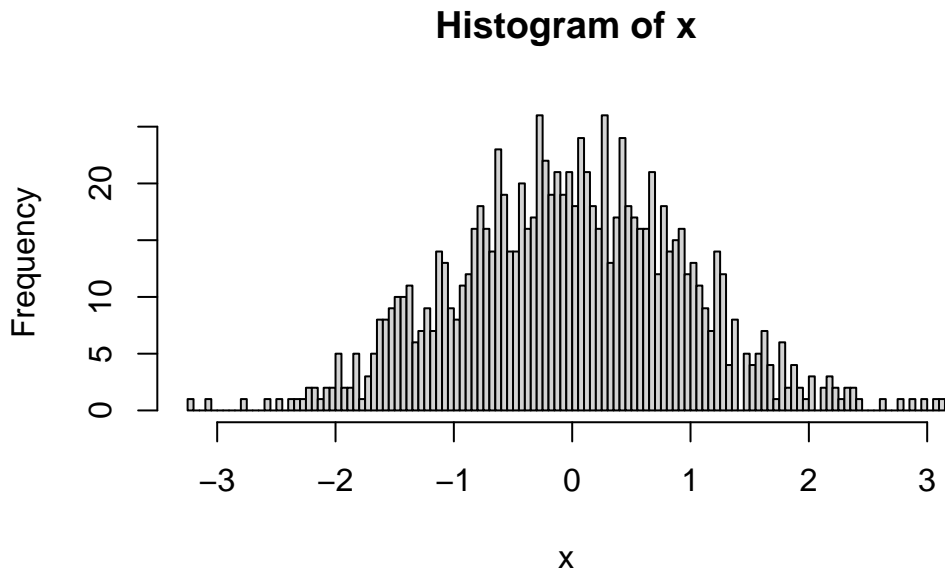
make histogram
hist(x)
```



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

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.

### 7.3.2 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()

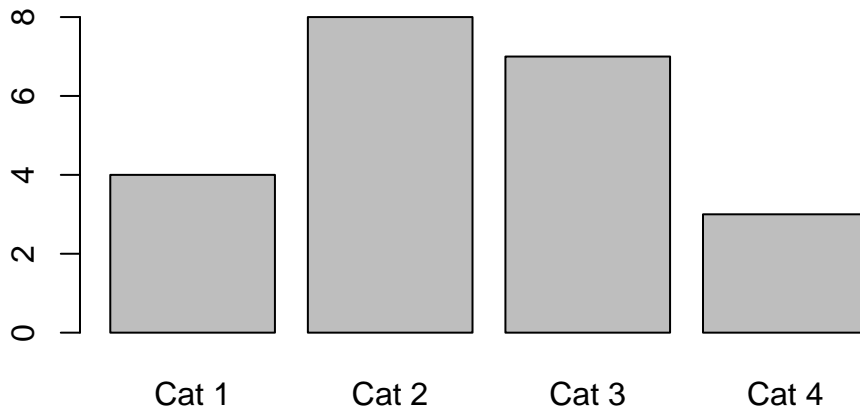
colnames(count_table) <- c("Category", "Count")

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
```



### 7.3.3 ggplot histograms

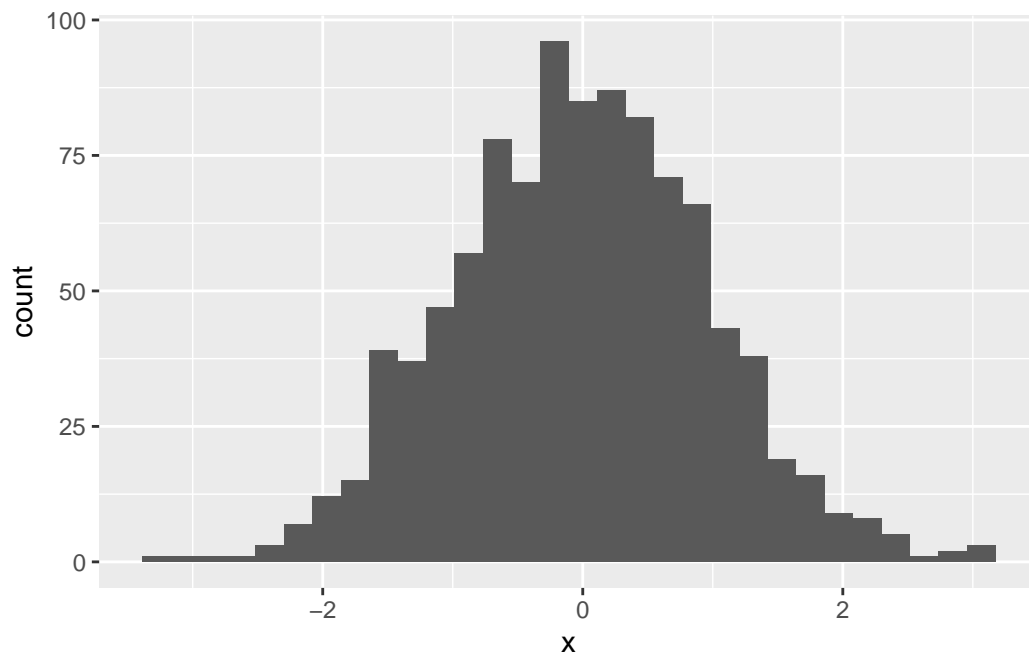
The following is an *optional* walkthrough on how to make really fancy plots.

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

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

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

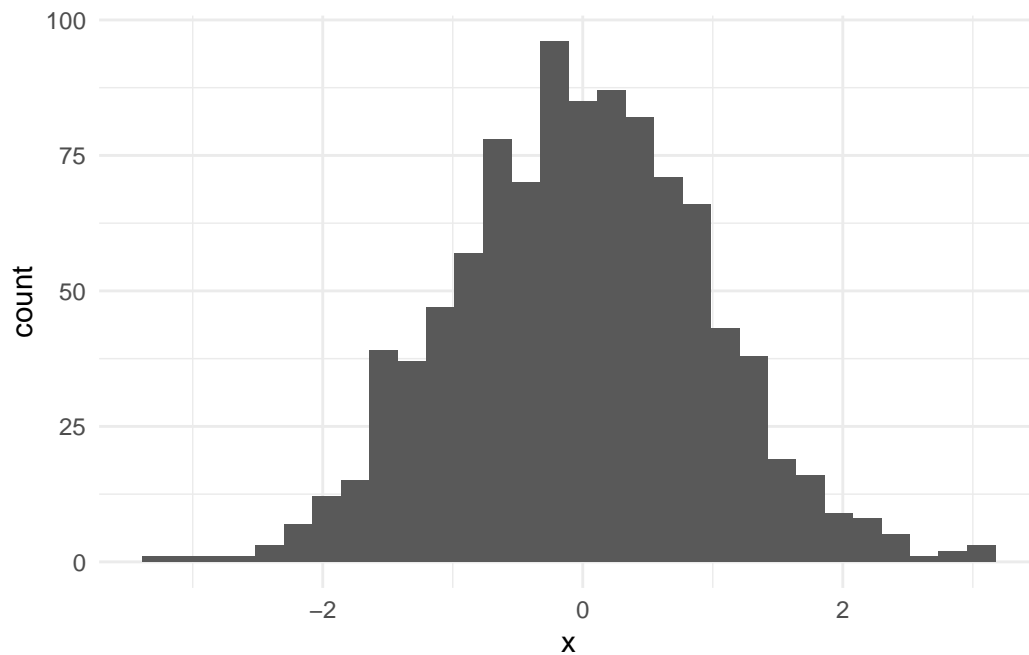
``stat_bin()`` using ``bins = 30``. Pick better value with ``binwidth``.



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"

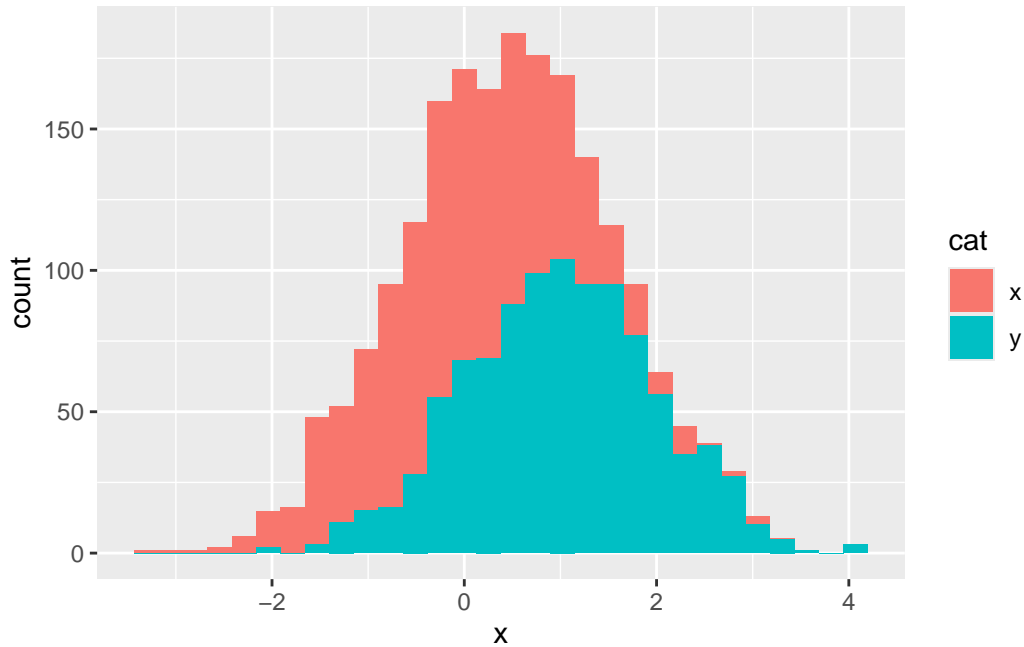
xy <- rbind(x2,y)

head(xy)
```

|   | x           | cat |
|---|-------------|-----|
| 1 | -1.07435774 | x   |
| 2 | -0.82433014 | x   |
| 3 | 0.76803047  | x   |
| 4 | 0.34248267  | x   |
| 5 | 0.84234017  | x   |
| 6 | -0.00990365 | x   |

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

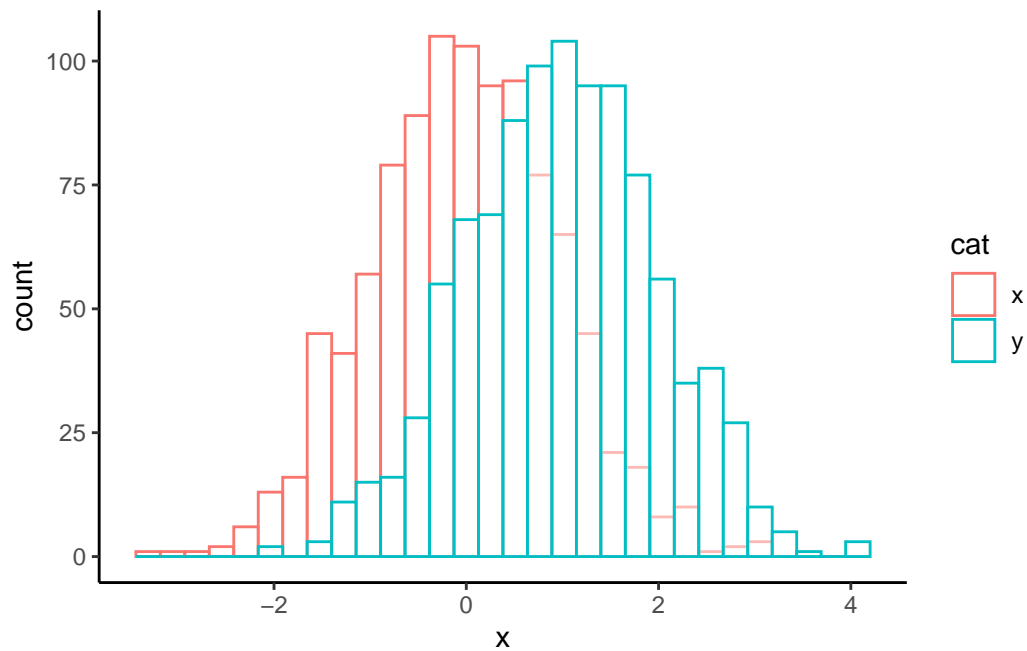
``stat_bin()`` using ``bins = 30``. Pick better value with ``binwidth``.



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()
```

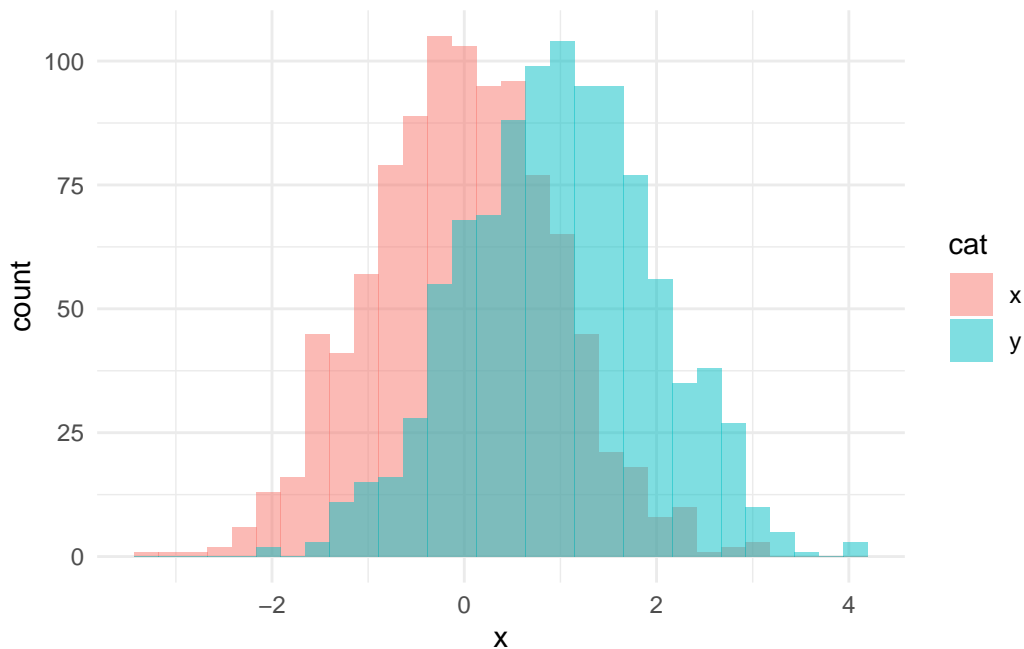
``stat_bin()`` using ``bins = 30``. Pick better value with ``binwidth``.



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](#).

## 7.4 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] 0.04627593
```

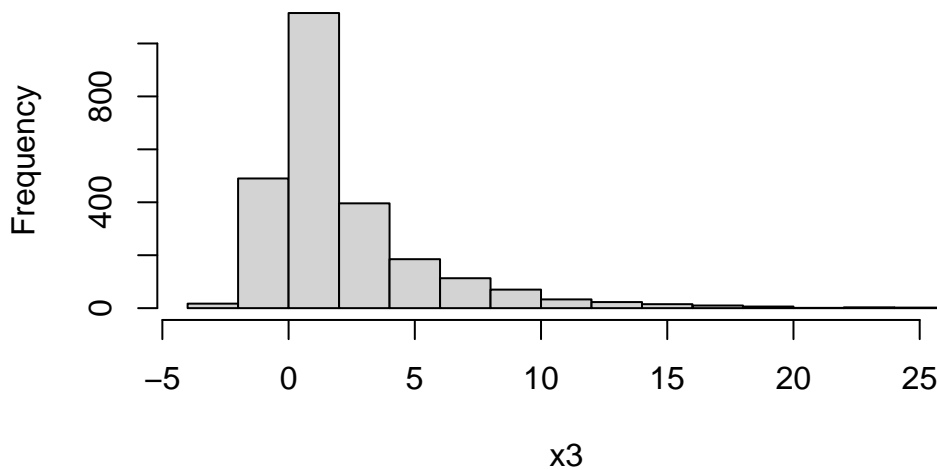
Generally, a **value between  $-1$  and  $+1$  for skewness is “acceptable”** and not considered overly skewed. Positive values indicate “right” skew and negative values indicate a “left” skew. If something is too skewed, it may violate assumptions of normality and thus need *non-parametric* tests rather 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.

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

hist(x3)
```

**Histogram of x3**



```
skewness(x3)
```

```
[1] 2.303814
```

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

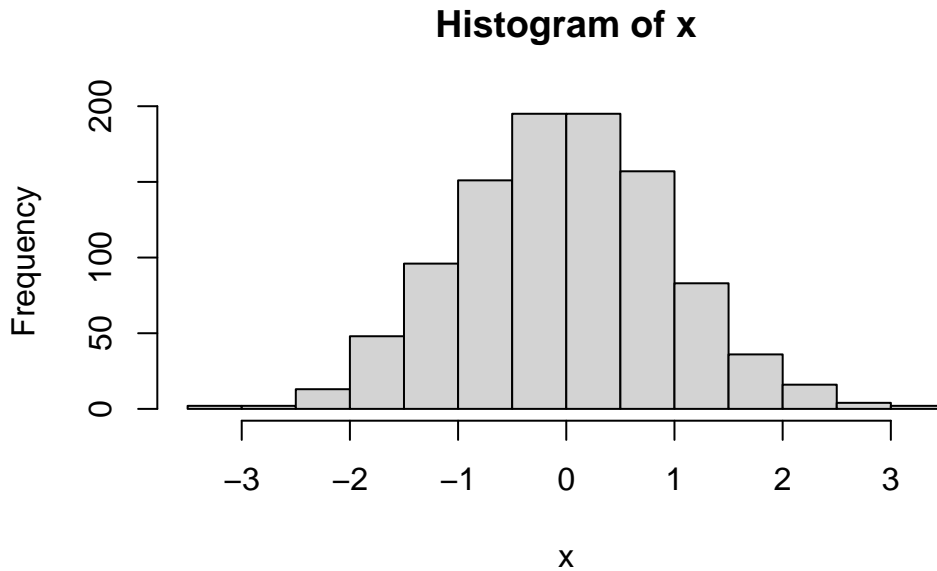
## 7.5 Kurtosis

Kurtosis refers to how sharp or shallow the peak of the distribution is (*platykurtic* vs. *leptokurtic*). Remember - *platykurtic* are *plateaukurtic*, 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  $\pm 2$  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  $\text{kurtosis} - 3$  to “zero” the distribution and make it comparable to skewness.

```
hist(x)
```



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

```
[1] 3.055901
```

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

```
[1] 0.05590072
```

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

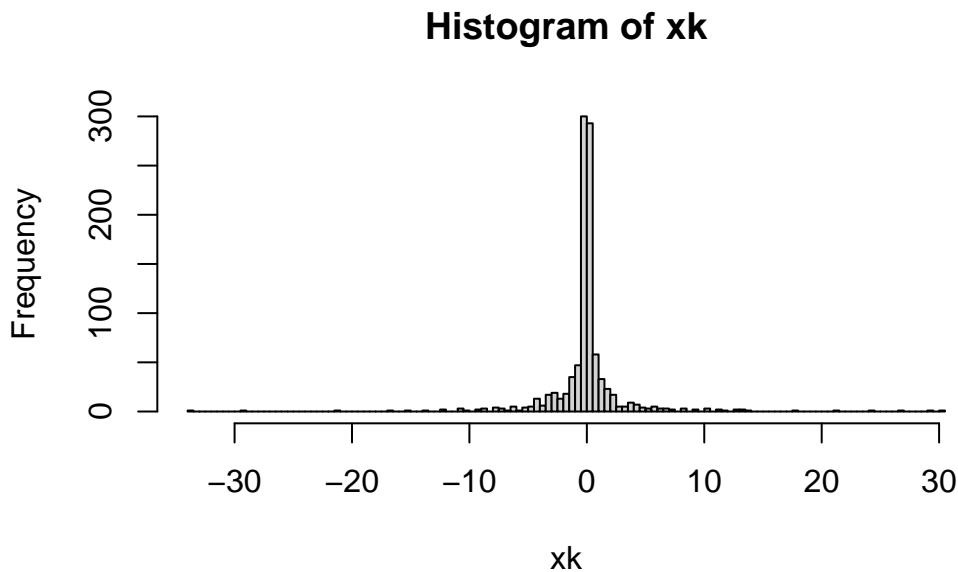
```
xk <- x^3
kurtosis(xk)-3
```

```
[1] 29.58594
```



What does this dataset look like?

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



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

## 7.6 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](#).

### 7.6.1 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] -3.234498 3.110551
```

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)
 15 144 346 352 119 20 2
```

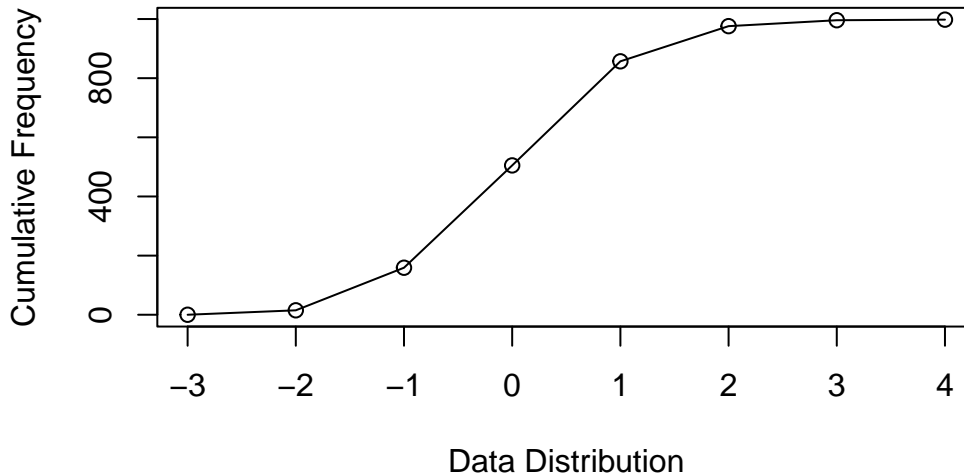
```
calculating cumulative frequency
cumulative_freq = c(0, # start at 0, no points
 cumsum(freq_table)) # get the cumulative frequency!
print("Cumulative Frequency")
```

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

```
print(cumulative_freq)
```

|  | $[-3,-2)$ | $[-2,-1)$ | $[-1,0)$ | $[0,1)$ | $[1,2)$ | $[2,3)$ | $[3,4)$ |
|--|-----------|-----------|----------|---------|---------|---------|---------|
|  | 0         | 15        | 159      | 505     | 857     | 976     | 996     |

```
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
```



### 7.6.2 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)
```

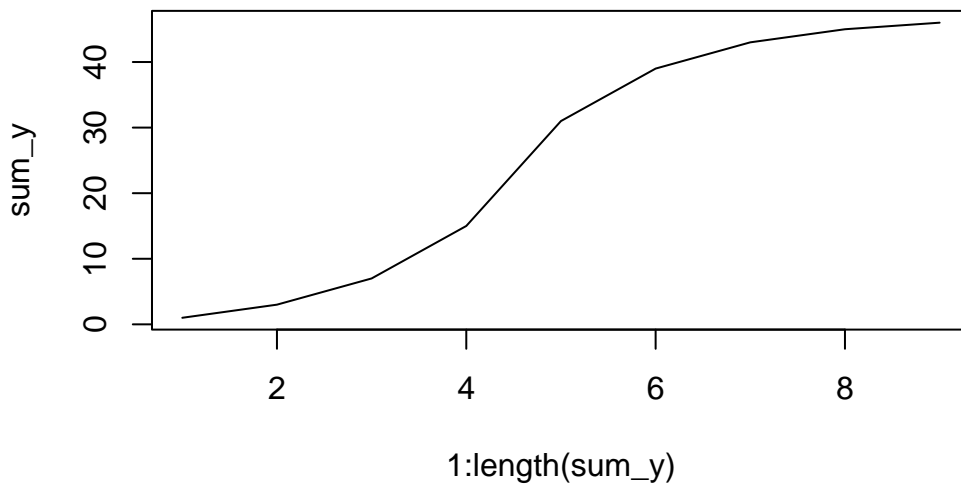
```
[1] 1 2 4 8 16 8 4 2 1
```

```
print(sum_y)
```

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

Now we can see we have our 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
```



## 7.7 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.

### 7.7.1 Helpful hint

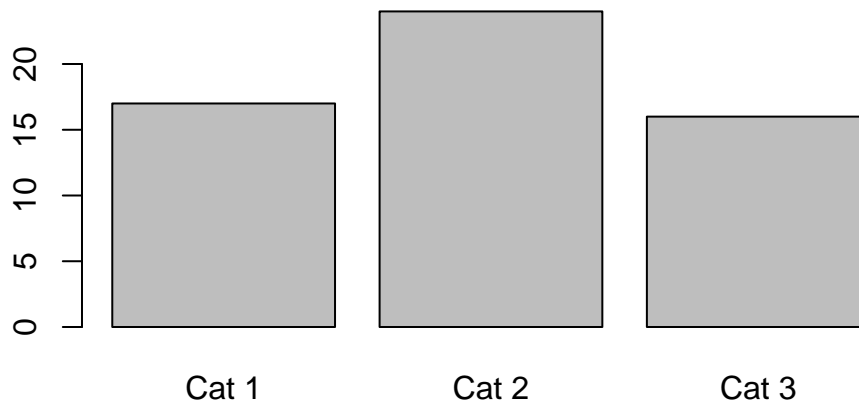
**HINT:** For 3.5, consider just making a vector of the values of interest for a histogram.

For example, see the following. For reference:

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

```
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"))
```



### 7.7.2 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*.

## 7.8 Addendum

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

## 8 Normality & hypothesis testing

### 8.1 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* ( $\sigma^2$ ), the overall spread of the data, and by the *standard deviation* ( $\sigma$ ), which is defined by the square root of the variance.

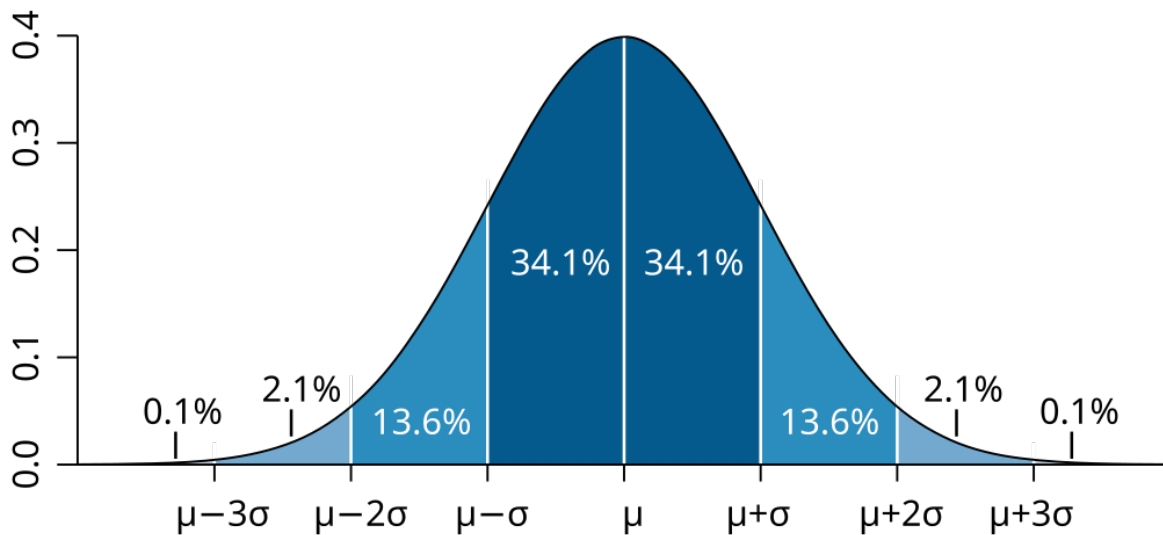


Figure 8.1: A standard normal distribution, illustrating the percentage of area found within each standard deviation away from the mean. By Ainali on Wikipedia; CC-BY-SA 3.0.

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.

### 8.1.1 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 `na.omit`.

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

```
[1] 18.65
```

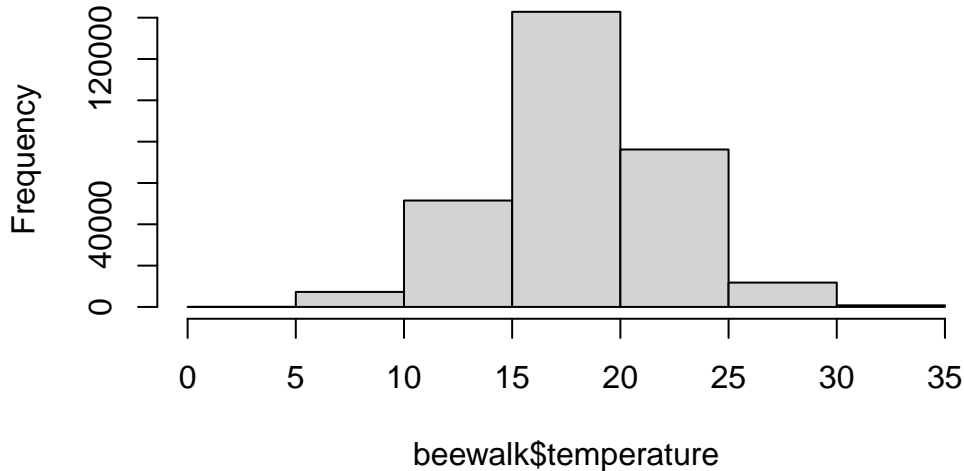
Now we can record the mean.

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)
```

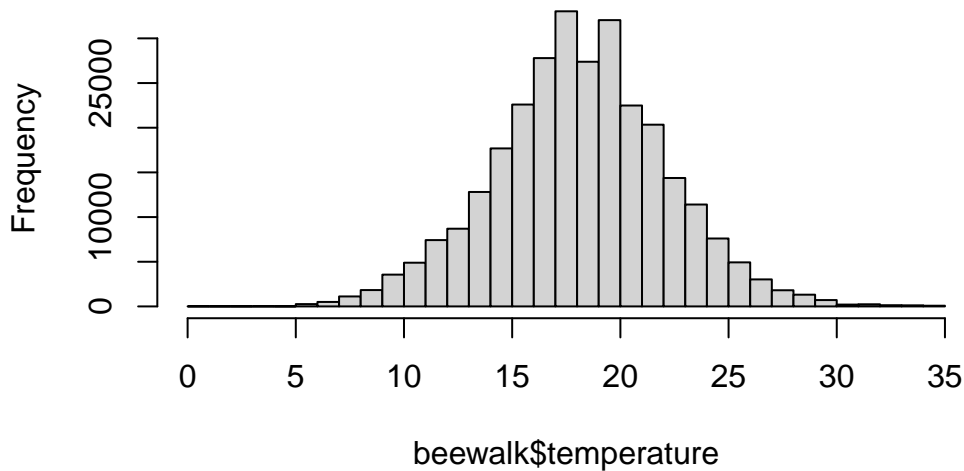
**Histogram of beewalk\$temperature**



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)
```

**Histogram of beewalk\$temperature**



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

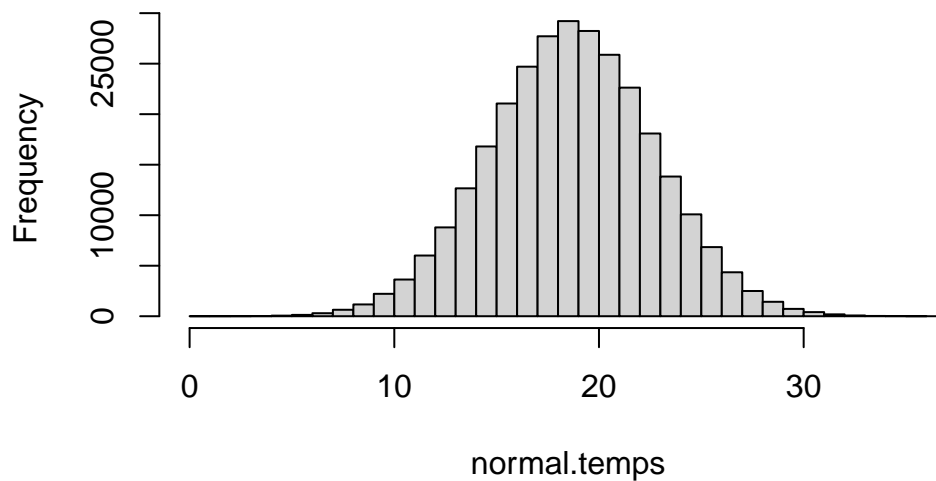
```
save temperature vector without NA values
temps <- beewalk$temperature %>% na.omit()

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)
```

## Histogram of normal.temps



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:

```
load moments library
library(moments)

skewness(temps)
```

```
[1] 0.02393257
```

Data do not have any significant skew.

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

```
[1] 0.3179243
```

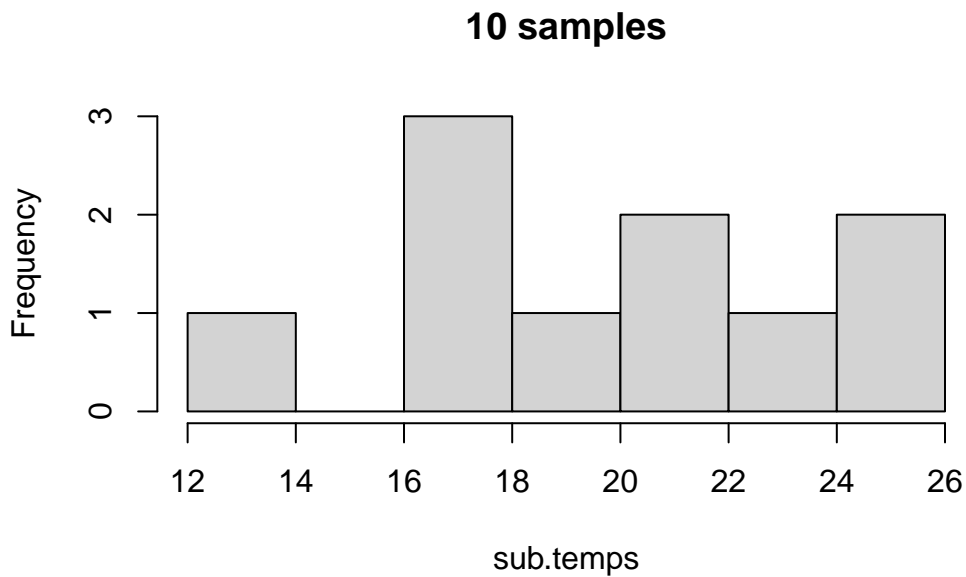
Data do not show any significant kurtosis.

### 8.1.2 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.

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

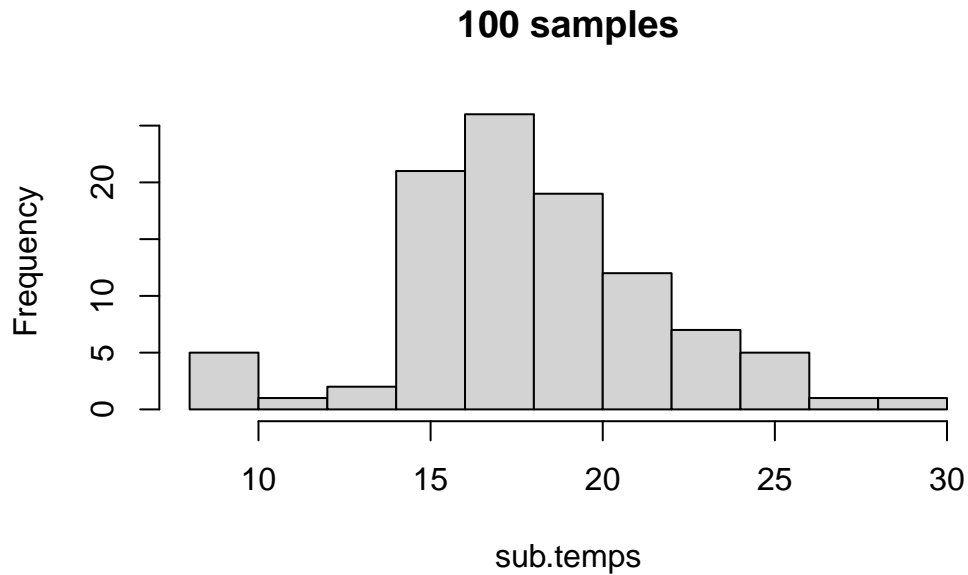
hist(sub.temps, main = "10 samples")
```



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

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

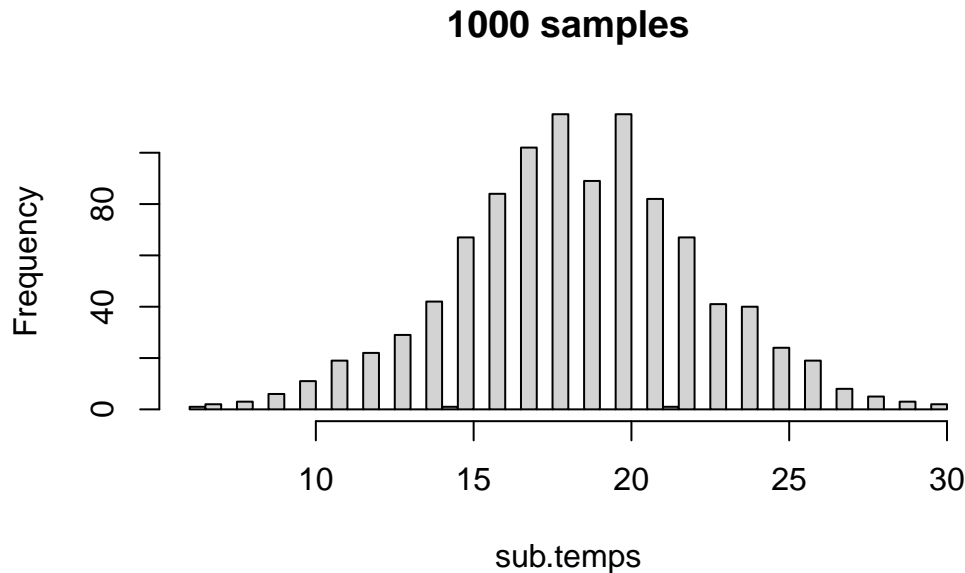
hist(sub.temps, main = "100 samples", breaks = 10)
```



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

```
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!

## 8.2 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*.

### 8.2.1 Critical Values - $\alpha$

In order to determine if something is significant, we compare things to a *critical value*, known as  $\alpha$ . 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  $\alpha$  critical value before you do your experiments and analyses.**

### 8.2.2 Visualizing a $p$ value

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")

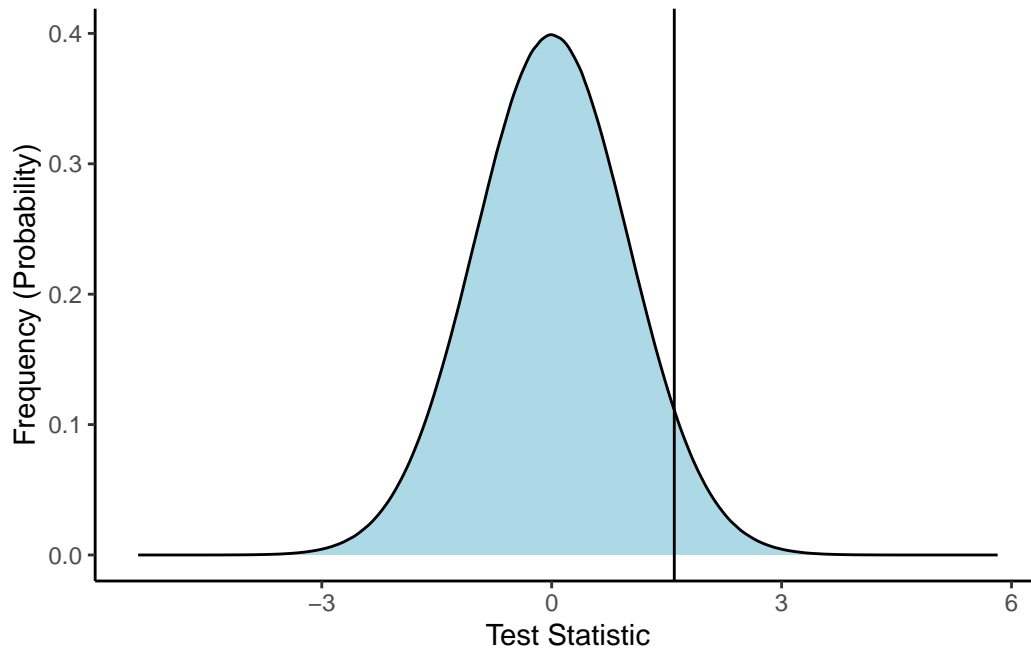
Building shaded area
create new plot object
p2 <- p + # add previous step as a "backbone"
 # rename axes
 geom_vline(xintercept = 1.6) +
```

```

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

plot it
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  $x$ .

```

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

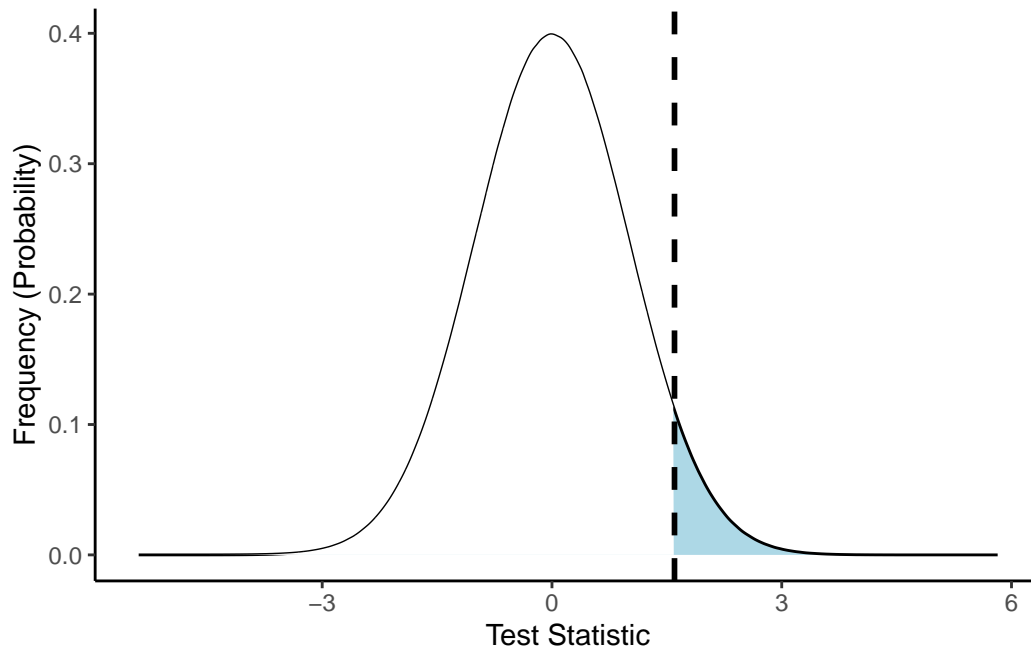
```

```

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 that is within our  $p$  value, but we don't know exactly what this area is! We can find out in one of two ways.

### 8.2.3 Calculating a $p$ value

We have two different methods for calculating a  $p$  value:



### 8.2.3.1 Comparing to a $z$ table

We can compare the  $z$  value we calculate to a  $z$  table, such as the one at [ztable.net](http://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  $p < 0.0001$ .

### 8.2.3.2 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`.

## 8.2.4 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/assign1/hurricanes.csv")
```

```
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)]
```

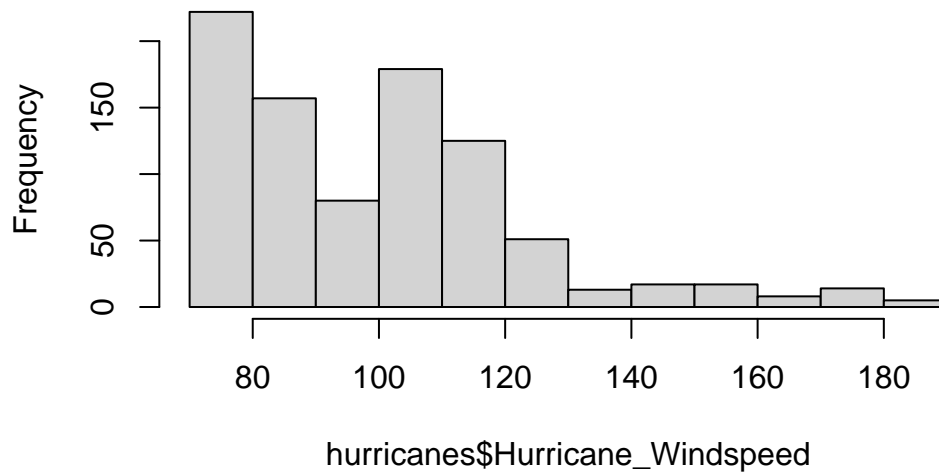
```
other_hurricanes <- hurricanes[-nrow(hurricanes),]
```

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)
```

## Histogram of 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.

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

sd.hurricane
```

```
[1] 23.08814
```

Now, we can calculate our  $Z$  value.

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

Z
```

```
[1] 3.355603
```

How significant is this?

```
pnorm(Z)
```

```
[1] 0.999604
```

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

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

```
[1] 0.000395961
```

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

#### 8.2.4.1 Non-normality, for those curious

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

```
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!

## 8.3 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  $\alpha$ , 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 aforementioned **beewalk** temperature data, we can calculate a 0.95 confidence interval around the mean.

```

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:

```

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 8.2: This is the “coded” version of the text below. Compare the above window to the text below this image.

Typing that into the plaintext 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  $18.65 \pm 0.01$ .

*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.

## 8.4 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`).

## 9 Exam 2 practice

### 9.1 Exam 2 Practice

The following is practice for the exam. Please work through these problems and be ready to discuss them in class.

### 9.2 Question 1: Cyclones

Consider this short data set:

| Latitude band | Season | Number of cyclones |
|---------------|--------|--------------------|
| 40 - 49° S    | Fall   | 370                |
| 40 - 49° S    | Winter | 452                |
| 40 - 49° S    | Spring | 273                |
| 40 - 49° S    | Summer | 422                |
| 50 - 59° S    | Fall   | 526                |
| 50 - 59° S    | Winter | 624                |
| 50 - 59° S    | Spring | 513                |
| 50 - 59° S    | Summer | 1,059              |
| 60 - 69° S    | Fall   | 980                |
| 60 - 69° S    | Winter | 1,200              |
| 60 - 69° S    | Spring | 995                |
| 60 - 69° S    | Summer | 1,751              |

All of the data was collected in the same year to determine whether the occurrences of cyclones differed by latitude and season around Antarctica. Use this data to answer the following questions.

- Classify each of the three variables above as either an explanatory or a response variable. Justify your answer.
- Classify each of the three variables above as either a nominal, ordinal, interval, or ratio variable. Justify your answer.



- c) State the null and alternative hypotheses for this scenario. Note: there are two sets of null/alternative hypotheses.
- d) Calculate the mean, median, and mode of your response variable. Based on your results, would you expect your data to be normally distributed or not? Justify your answer.
- e) Let's say now that you're testing the null hypothesis that the mean number of cyclones in this one year is similar to the average year. The mean number of cyclones per year across all years of data collected is 650 with a standard deviation of 425. Calculate the z-score and make a decision regarding the null hypothesis.

### 9.3 Question 2: Minnows

Consider this scenario: You have discovered a never-before-documented population of minnow in the Kearney Canal near campus. During your first sampling trip, you notice that the total length (i.e., measured from the tip of the snout to the very tip of the tail) of the fish you measure appear to be smaller than the average total length of the species as recorded among all known individuals across their range. The mean total length noted in one publication is 85 mm with a standard deviation of 4.50. Below is your data the data from 20 minnows that you captured during your first sampling trip to the Kearney Canal:

| Fish ID | Length (mm) |
|---------|-------------|
| 1       | 89          |
| 2       | 75          |
| 3       | 86          |
| 4       | 74          |
| 5       | 69          |
| 6       | 100         |
| 7       | 73          |
| 8       | 69          |
| 9       | 96          |
| 10      | 79          |
| 11      | 61          |
| 12      | 62          |
| 13      | 95          |
| 14      | 98          |
| 15      | 100         |
| 16      | 57          |
| 17      | 70          |
| 18      | 78          |
| 19      | 65          |
| 20      | 65          |

- a) State the null and alternative hypothesis for your study.
- b) Calculate the median, first and third quartiles, and interquartile range of your response variable. Create a boxplot. Based on your results, are there any outliers in your data? Explain.
- c) Calculate the  $z$ -score for this scenario. Please show all of your work.
- d) What is the probability that, by random chance alone, you would find your observed mean or something more extreme?
- e) Assume that you set your  $\alpha$  for your study prior to your data collection to 0.05. Based on this information and the  $p$ -value you obtained for part  $c$  above, is your null hypothesis supported or rejected?

# 10 Probability distributions

## 10.1 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 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 trials we perform, the more confident we can be that our 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  $\alpha$  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.

## 10.2 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.

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  $\alpha$  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.

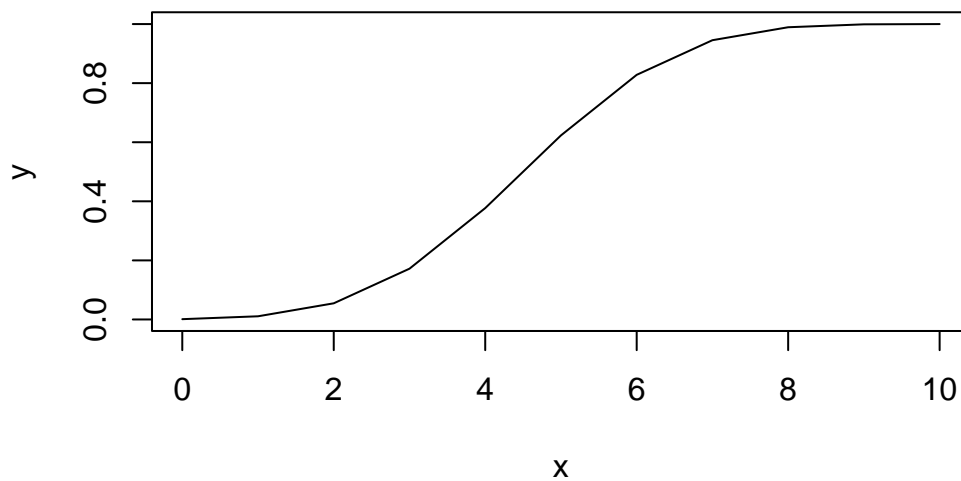
How does this distribution look?

```

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

```



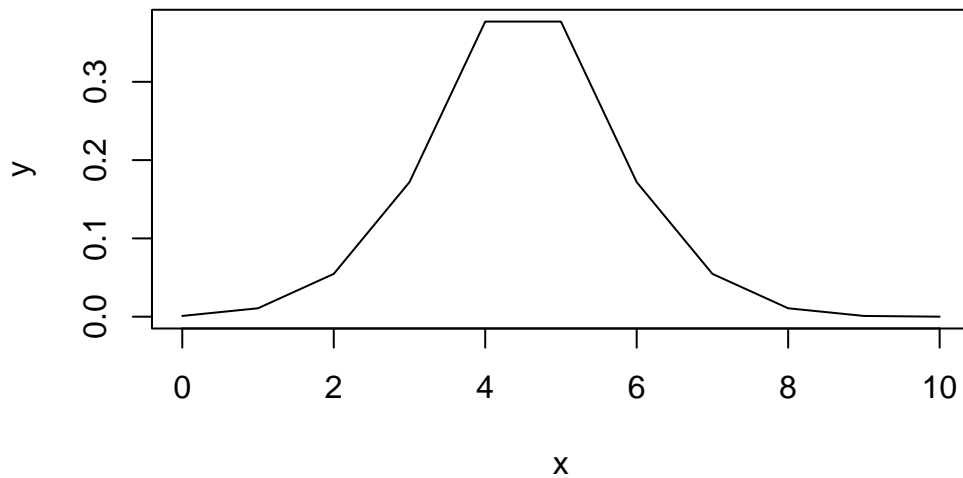
What about if we always have  $p$  less than 0.5 to reflect two tails?

```

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

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

```

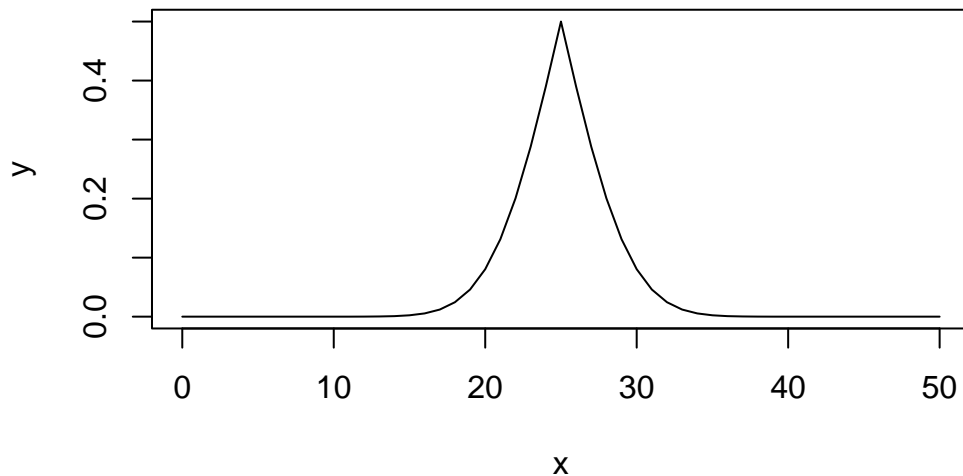


What if we do this with a bigger dataset, like for 50 flips?

```
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 = F)

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.

## 10.3 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* ( $\lambda$ ) which represents the mean ( $\mu$ ) and the variance ( $\sigma^2$ ), which are equal in a Poisson distribution. A Poisson distribution is described by the following function:

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

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

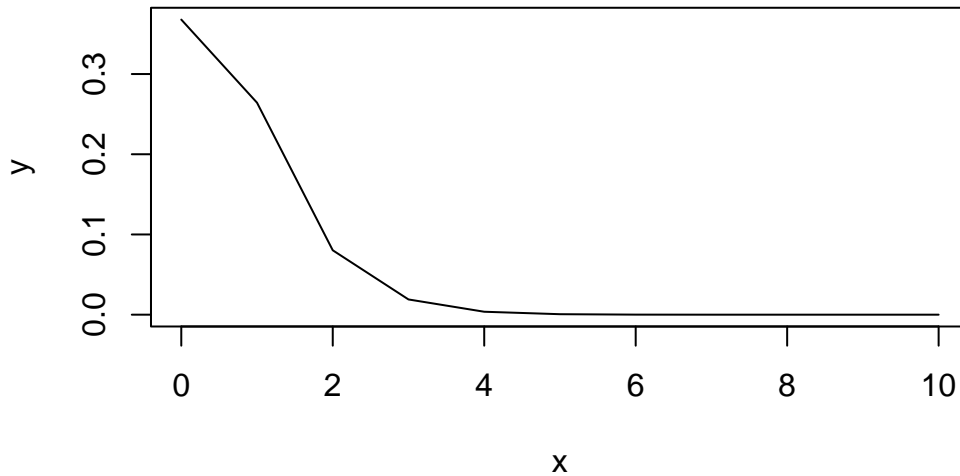
- `dpois`: the log density function
- `ppois`: log distribution (probability) function
- `qpois`: quantile function
- `rpois`: random values from a Poisson.

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.

## 10.4 $\chi^2$ -squared distribution

$\chi^2$ -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  $\chi^2$  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  $\chi^2$  is as follows, with the  $\chi^2$  being a special case of the *gamma* ( $\gamma$  or  $\Gamma$ ) 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  $\chi^2$ -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`



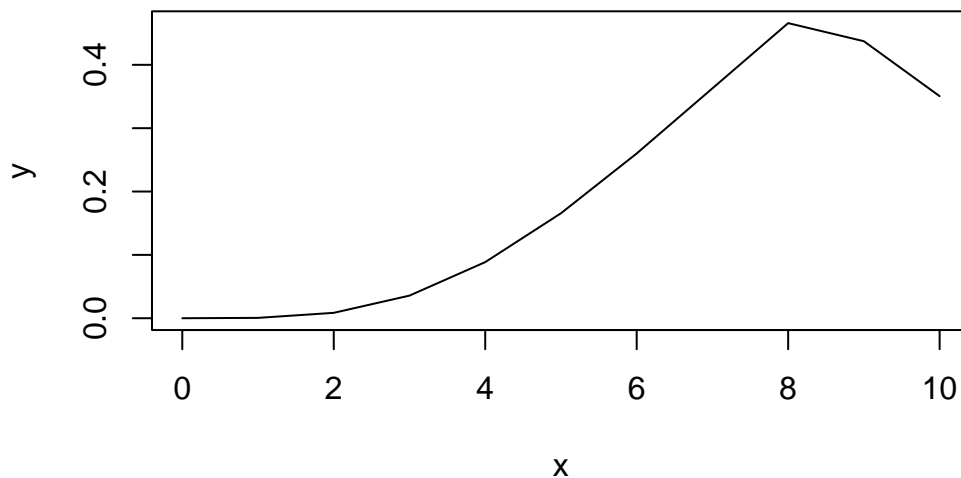
We can view these probabilities as well:

```
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]

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



#### 10.4.1 Calculating the test statistic

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

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

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**.

### 10.4.2 $\chi^2$ estimations in $R$

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

| Major          | $R$ experience | No $R$ experience |
|----------------|----------------|-------------------|
| <i>English</i> | 2              | 8                 |
| <i>Biology</i> | 9              | 1                 |

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 to use  $R$ :

```
chisq.test(x = c(2,8),
 y = c(9,1))
```

```
Warning in chisq.test(x = c(2, 8), y = c(9, 1)): Chi-squared approximation may
be incorrect
```

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

```
data: c(2, 8) and c(9, 1)
X-squared = 0, df = 1, p-value = 1
```

Here we get  $p = 1$ , which reflects a one-tailed interpretation.

### 10.4.3 Calculating $\chi^2$ by hand

## 10.5 Fisher's exact test

To be completed; involves calculations on smaller datasets.

## 10.6 Homework

### 10.6.1 Chapter 5

### 10.6.2 Chapter 7

# **11 Single population means testing**

## **11.1 Introduction**

## **11.2 $t$ -distribution**

## **11.3 $t$ -tests**

## **11.4 Wilcoxon tests**

## **11.5 Confidence intervals**

## **11.6 Homework: Chapter 9**

## **12 Two sample tests**

### **12.1 Introduction**

### **12.2 $t$ -tests**

### **12.3 Mann-Whitney $U$ tests**

### **12.4 Error**

### **12.5 Homework: Chapter 10**

## **13 ANOVA: Part 1**

### **13.1 Introduction**

### **13.2 ANOVA: By hand**

### **13.3 ANOVA: By *R***

### **13.4 Kruskal-Wallis tests**

### **13.5 Homework: Chapter 11**

## **14 ANOVA: Part 2**

### **14.1 Two-way ANOVA**

### **14.2 Designs**

#### **14.2.1 Randomized block design**

#### **14.2.2 Repeated measures**

#### **14.2.3 Factorial ANOVA**

### **14.3 Friedman's test**

### **14.4 Homework: Chapter 12**

# **15 Correlation & regression**

## **15.1 Introduction**

## **15.2 Correlation**

### **15.2.1 Pearson's**

### **15.2.2 Spearman's**

### **15.2.3 Other non-parametric methods**

## **15.3 Correlation**

### **15.3.1 Parametric**

### **15.3.2 Non-parametric**

## **15.4 Homework**

### **15.4.1 Chapter 13**

### **15.4.2 Chapter 14**

## **16 Final exam & review**

### **16.1 Pick the test**

### **16.2 Final review**



# 17 Conclusions

Parting thoughts about the course.

## 17.1

(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

## 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>.
- 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>

# 18 Glossary

## 18.1 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
```

- `c` - concatenate, place two values together

```
x <- c(10,11)
x
```

```
[1] 10 11
```

-