# Introduction to R

BUGS Workshop - Fall 2023

Dominique Maucieri

27 September, 2023

## Contents

---

Welcome to this Introduction to R workshop.

# R etiquette and setting up your code

## Set up

Lets start by making a new script for this tutorial. We will open a new script file and save it as BUGS_workshop.R

Now to set up this script, the first thing you always want to do in every new script, is put a description at the top of your script explaining what this script contains. Anything on a line following a # will be a comment, this code will not run, it will just be a note to anyone reading the code. For this workshop we can put a description kinda like this:

```
# This code is from the UVic BUGS Introduction to R Workshop
# Workshop led by Dominique Maucieri (dominiquemaucieri@gmail.com)
# 28-10-2023
```

The next thing you will put at the top of your code is **ALL** your packages. Packages are extensions to R what will contain things like code and data that we can use in our coding. They will often contain a bunch of different functions which will allow us to easily perform different tasks without coding them by hand. In the BEFORE_THE_WORKSHOP.pdf file, you would have been instructed to download a bunch of packages. We will be using these packages today to help us quickly perform task. It is proper etiquette to load all these packages at the start of your code so that someone else wanting to run your code will know what packages to download before starting. As you get better at coding you will know which packages you will want to use before you start, but as you learn, just keep scrolling to the top of your script and add packages as you need them.

For our workshop we will be using the dplyr, tidyr, ggplot2 and readxl packages, so lets load these and add them to the start of our script:

```
# Packages ----
library(dplyr)
library(ggplot2)
library(readxl)
library(tidyr)
library(car)
library(palmerpenguins)
#install.packages("dunn.test") #you may need to install this package if you haven't before
library(dunn.test)
```

## Creating an outline

If you notice after the comment were I specified the following code was packages, there are 4 "-". This will create a new section in your code. If you open the outline, which is located at the top right of your script, you can see the outline of your code. This is super helpful when you start getting long code scripts, because you can easily jump around your code.

## Importing data

### Setting the working directory

In order to import data, we have to know where the data is located on your computer. When your import data, you will use code to tell R the location of the data on your computer to open and load into your working environment so you can use it in R. First you need to know the location on your computer where R is currently "looking", known as your working directory. Good protocol is to have your data in the same folder as where you saved your script file, or in a data folder within the folder that your script file is saved in.

To figure out where your working directory is we can use this code:

```
# Importing data ----
getwd() #where is my current working directory
```

```
## [1] "/Users/dom/Library/CloudStorage/Dropbox/Documents/GitHub/UVic_BUGS_RIntro"
```

For me, this folder "UVIC_BUGS_RIntro" is the folder where I have put the script we are using for today's workshop and it is also where I have the data we are going to import for today's workshop. For you however, this might not be the right spot. So if we need to change the working directory, there are a few ways:

1. Code. This is the toughest to work your head around but it works. You would use the setwd() and the file path to your script folder (this should also be where your data is, if not move your data into the same folder). You always put the file path in quotations, and for my folder it would look like this:

```
setwd("~/Library/CloudStorage/Dropbox/Documents/GitHub/UVic_BUGS_RIntro")
```

Now often the issue with this method is your probably thinking well how do I get this file path? If you don't have a good understanding of file paths and how your computer works, this can seem very difficult, but there are a few short cuts to make this easier.

2. Using the File viewer pane. In the bottom right of your screen, there should be the files viewer. We can use this to move around on your computer and set your new working directory. Open files until you find the folder with your script and data. Then click the gear icon at the top of the file pane, and select "Set As Working Directory". This will run the line of code needed to set this folder as your working directory. Now important to remember to take the code outputted into your console, and paste it into your script so every time your come back to this code, you can set your working directory again.
3. Using the session options. At the top of your RStudio, go to Session > Set Working Directory > Choose Directory... Then you can move around on your computer until you find the folder with your script and data. This will run the line of code needed to set this folder as your working directory. Now important to remember to take the code outputted into your console, and paste it into your script so every time your come back to this code, you can set your working directory again.

**TASK**

Set your working directory to the folder where you have your script and data located.

### Importing csv and excel data

There are many types of files you may want to import into R to use, and for this tutorial we will go through two formats of data, comma separated values (csv) and excel files. The biggest thing to figure out is where your data is in relation to where you working directory has been sent. The easiest is for the working directory to be set to the same location as your data.

**.csv** To import data that has been saved as a .csv file, we will use the read.csv() function. If this data is located in the same folder as your working directory has been set to (which it should be after the task above) we will import the file as follows:

```
shark_data <- read.csv("LPD_Sharks_CPUE.csv")
```

But perhaps this data is in a folder called data, which is located in your working directory. This is how my data is always stored, so in order to import it, I have to set the file path as within a folder like this:

```
shark_data <- read.csv("data/LPD_Sharks_CPUE.csv")
```

Troubleshooting:

- Always make sure the files are spelt the same way
- Make sure you include the extension (.csv)
- Make sure it is in quotes

Notice that we stored this data into an object called "shark_data". This should now show up in your environment window as data. This means that anywhere you use the word shark_data in your code after now, it will reference this data.

This data comes from the Living Planet Database (https://www.livingplanetindex.org/) and is a cleaned up version where I have subset out some shark abundance data for us to look at.

**.excel** Importing excel files are very similar, they will just use a different function, read.excel() and the .xls or .xlsx file extension:

```
shark_data <- read_excel("data/LPD_Sharks_CPUE.xlsx")
```

Yay! Now we have data to work with in R. But what to do next? Well we need to look at this data and explore it a bit. There are many ways to do this. The first will be to just open the data frame and scroll around it. To open the data frame we can do a few things:

1. Over in the environment tab, we can click on the data frame we want to examine.
2. In the console or our script we can use the View() function to view our data, with the name of the data frame object we want to open and view within the brackets of the View() function.

**TASK**

Try to open the data frame to view it using a few different methods.

---

## Types of Objects and Elements

There are a few types of objects you will come across in R. These will be data frames, matrices, vectors and lists.

- A vector is like a single string of variables, like a single row of data, one dimension.
- Data frames are what we have already encountered, it has rows and columns, two dimensions, and it can be a mix of numbers and characters
- Matrices like data frames with rows and columns but they only contain one type of data, so all numbers or all characters
- Lists are like a bunch of data frames all stacked on top of each other, three dimensions, but we won't be working with lists for this.

As eluded to above, there are different types of data as well. We can have numeric, integer, logical, character and factor type data.

- Numeric is numbers, they contain decimals and are continuous. Examples are length measures or heights
- Integers are whole numbers, no decimals but still numbers. Example would be the year for your data.
- Logical data is TRUE / FALSE data
- Characters are text, known as strings in R, which can be words or a jumble of various letters and symbols, but this data is not being treated as a number and has no order to it. Characters are shown with quotes around them so R knows to treat it as a character. Examples could be site names or species names. "Site1"
- Factors are also text, like characters but they have a set order to them. The default order will be alphabetical but you can change this in R so it makes more sense. Example would be treatments, like before and after, you would want before to be order before after or it won't make sense for something like a plot.

To create a vector, we can use the c() function to combine a bunch of elements together and assign them to an object with the arrow operator <-

```r
fruit <- c("apple", "banana", "pear")

#then calling the object name will output the vector values in your console
fruit
```

```
## [1] "apple"  "banana" "pear"
```

```r
leaf_lengths <- c(4.5, 7.2, 4, 5.4)

leaf_lengths
```

```
## [1] 4.5 7.2 4.0 5.4
```

**TASK**

Try making your own object that contains both characters and numeric data. Then look at the structure of this object, what type of data does it contain?

---

It is easy to change the type of data to something else. Sometimes R will read in your numbers as characters or you might want to change your characters to a factor. So to this, first you need to know how R is currently classifying your data. To see this we can use class() or str(). I like str() because it gives a bit more information, but either works.

```r
class(leaf_lengths)
```

```
## [1] "numeric"
```

```r
str(fruit)
```

```
##  chr [1:3] "apple" "banana" "pear"
```

so leaf_length is a numeric vector and fruit is a character vector. To change to a different type of data you use as.******() with the asterisks being the new data type. So as.numeric() or as.factor() etc. Lets turn fruit into a factor, and assign it to a new operator called fruit_fact

```r
fruit_fact <- as.factor(fruit)

str(fruit_fact)
```

```
##  Factor w/ 3 levels "apple","banana",..: 1 2 3
```

Now when we want to change how a column in a data frame is being treated by R, we have to select that specific column. We can select a column within a data frame using the $

```r
shark_data$Common_name
```

```
##   [1] "Porbeagle shark"          "Porbeagle shark"
##   [3] "Porbeagle shark"          "Porbeagle shark"
##   [5] "Porbeagle shark"          "Porbeagle shark"
##   [7] "Porbeagle shark"          "Porbeagle shark"
##   [9] "Porbeagle shark"          "Porbeagle shark"
##  [11] "Porbeagle shark"          "Porbeagle shark"
##  [13] "Thresher shark"           "Thresher shark"
##  [15] "Thresher shark"           "Thresher shark"
##  [17] "Thresher shark"           "Thresher shark"
##  [19] "Thresher shark"           "Thresher shark"
##  [21] "Thresher shark"           "Thresher shark"
##  [23] "Thresher shark"           "Thresher shark"
##  [25] "Thresher shark"           "Thresher shark"
##  [27] "Thresher shark"           "Thresher shark"
##  [29] "Thresher shark"           "Thresher shark"
##  [31] "Thresher shark"           "Thresher shark"
##  [33] "Thresher shark"           "Thresher shark"
##  [35] "Blue shark"               "Blue shark"
##  [37] "Blue shark"               "Blue shark"
##  [39] "Blue shark"               "Blue shark"
##  [41] "Blue shark"               "Blue shark"
##  [43] "Blue shark"               "Blue shark"
##  [45] "Blue shark"               "Blue shark"
##  [47] "Blue shark"               "Blue shark"
##  [49] "Blue shark"               "Blue shark"
##  [51] "Blue shark"               "Blue shark"
##  [53] "Blue shark"               "Blue shark"
##  [55] "Blue shark"               "Blue shark"
##  [57] "Blue shark"               "Shortfin mako"
##  [59] "Shortfin mako"            "Shortfin mako"
##  [61] "Shortfin mako"            "Shortfin mako"
##  [63] "Shortfin mako"            "Shortfin mako"
##  [65] "Shortfin mako"            "Shortfin mako"
##  [67] "Shortfin mako"            "Shortfin mako"
##  [69] "Shortfin mako"            "Shortfin mako"
##  [71] "Shortfin mako"            "Sand tiger shark"
##  [73] "Sand tiger shark"         "Sand tiger shark"
##  [75] "Sand tiger shark"         "Sand tiger shark"
##  [77] "Sand tiger shark"         "Sand tiger shark"
##  [79] "Sand tiger shark"         "Sand tiger shark"
##  [81] "Sand tiger shark"         "Sand tiger shark"
##  [83] "Sand tiger shark"         "Sand tiger shark"
##  [85] "Sand tiger shark"         "Sand tiger shark"
##  [87] "Sand tiger shark"         "Sand tiger shark"
##  [89] "Sand tiger shark"         "Sand tiger shark"
##  [91] "Sand tiger shark"         "Sand tiger shark"
##  [93] "Sand tiger shark"         "Sand tiger shark"
##  [95] "Sand tiger shark"         "Sand tiger shark"
##  [97] "Sand tiger shark"         "Sand tiger shark"
##  [99] "Sand tiger shark"         "Sand tiger shark"
## [101] "Sand tiger shark"         "Sand tiger shark"
## [103] "Sand tiger shark"         "Sand tiger shark"
## [105] "Great white shark"        "Great white shark"
## [107] "Great white shark"        "Great white shark"
```

```
## [109] "Great white shark"         "Great white shark"
## [111] "Great white shark"         "Great white shark"
## [113] "Great white shark"         "Great white shark"
## [115] "Great white shark"         "Great white shark"
## [117] "Great white shark"         "Great white shark"
## [119] "Great white shark"         "Great white shark"
## [121] "Great white shark"         "Great white shark"
## [123] "Great white shark"         "Great white shark"
## [125] "Great white shark"         "Great white shark"
## [127] "Great white shark"         "Great white shark"
## [129] "Great white shark"         "Great white shark"
## [131] "Great white shark"         "Great white shark"
## [133] "Great white shark"         "Great white shark"
## [135] "Great white shark"         "Great white shark"
## [137] "Great white shark"         "Blue shark"
## [139] "Blue shark"                "Blue shark"
## [141] "Blue shark"                "Blue shark"
## [143] "Blue shark"                "Blue shark"
## [145] "Blue shark"                "Blue shark"
## [147] "Blue shark"                "Blue shark"
## [149] "Atlantic sharpnose shark"  "Atlantic sharpnose shark"
## [151] "Atlantic sharpnose shark"  "Blacktip shark"
## [153] "Blacktip shark"            "Blacktip shark"
## [155] "Spinner shark"             "Spinner shark"
## [157] "Spinner shark"             "Blacknose shark"
## [159] "Blacknose shark"           "Blacknose shark"
## [161] "Bonnethead shark"          "Bonnethead shark"
## [163] "Bonnethead shark"          "Sandbar shark"
## [165] "Sandbar shark"             "Sandbar shark"
## [167] "Scalloped hammerhead"      "Scalloped hammerhead"
## [169] "Scalloped hammerhead"      "Finetooth shark"
## [171] "Finetooth shark"           "Finetooth shark"
## [173] "Shortfin mako"             "Shortfin mako"
## [175] "Shortfin mako"             "Shortfin mako"
## [177] "Tope shark"                "Tope shark"
## [179] "Broadnose sevengill shark" "Broadnose sevengill shark"
## [181] "Smooth hammerhead"         "Smooth hammerhead"
## [183] "Smooth hammerhead"         "Smooth hammerhead"
## [185] "Smooth hammerhead"         "Smooth hammerhead"
## [187] "Smooth hammerhead"         "Smooth hammerhead"
## [189] "Smooth hammerhead"         "Lemon shark"
## [191] "Lemon shark"               "Lemon shark"
## [193] "Grey reef shark"           "Grey reef shark"
## [195] "Grey reef shark"           "Grey reef shark"
## [197] "Whitetip reef shark"       "Whitetip reef shark"
## [199] "Whitetip reef shark"       "Whitetip reef shark"
## [201] "Tiger shark"               "Tiger shark"
## [203] "Blacktip reef shark"       "Blacktip reef shark"
## [205] "Scalloped hammerhead"      "Scalloped hammerhead"
## [207] "Scalloped hammerhead"      "Scalloped hammerhead"
## [209] "Tawny nurse shark"         "Tawny nurse shark"
## [211] "Silky shark"               "Silky shark"
## [213] "Whale shark"               "Whale shark"
## [215] "Silvertip shark"           "Silvertip shark"
```

```
## [217] "Round ribbontail ray"      "Round ribbontail ray"
## [219] "Ribbontail stingray"       "Ribbontail stingray"
```

This will select only one column within the data frame. So we could examine they type of data for the whole data frame or just a single column. However, class() of a data frame will tell you that it is a data frame, not the type of data in each column like str() will.

```
class(shark_data)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
str(shark_data)
```

```
## tibble [220 x 15] (S3: tbl_df/tbl/data.frame)
##  $ ID          : num [1:220] 1934 1934 1934 1934 1934 ...
##  $ Binomial    : chr [1:220] "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" ...
##  $ Class       : chr [1:220] "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" ...
##  $ Order       : chr [1:220] "Lamniformes" "Lamniformes" "Lamniformes" "Lamniformes" ...
##  $ Family      : chr [1:220] "Lamnidae" "Lamnidae" "Lamnidae" "Lamnidae" ...
##  $ Genus       : chr [1:220] "Lamna" "Lamna" "Lamna" "Lamna" ...
##  $ Species     : chr [1:220] "nasus" "nasus" "nasus" "nasus" ...
##  $ Common_name: chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" "Porbeagle shark"
##  $ Location    : chr [1:220] "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-6, Canada" "NAFO Subareas 3
##  $ Country     : chr [1:220] "Canada" "Canada" "Canada" "Canada" ...
##  $ Region      : chr [1:220] "North America" "North America" "North America" "North America" ...
##  $ Latitude    : num [1:220] 43.3 43.3 43.3 43.3 43.3 ...
##  $ Longitude   : num [1:220] -54.8 -54.8 -54.8 -54.8 -54.8 ...
##  $ Year        : num [1:220] 1989 1990 1991 1992 1993 ...
##  $ Abundance   : num [1:220] 0.0035 0.0041 0.0052 0.0068 0.0034 0.0035 0.0036 0.0025 0.0005 0.0015 ..
```

```
class(shark_data$Common_name)
```

```
## [1] "character"
```

```
str(shark_data$Common_name)
```

```
##  chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" ...
```

Now did you see what Year was being treated as? It was treated as numeric, but maybe we want it as an integer. So to change it to an integer we have to use the as.integer() on only that column and assign it to a single column.

```
shark_data$Year <- as.integer(shark_data$Year)
str(shark_data$Year)
```

```
##  int [1:220] 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 ...
```

# Math in R

R can be used as a fancy calculator, and common math expressions will work within your code.

- a * will multiply
- a / will divide
- a - will subtract
- a + will add
- log() will take the log of a number
- pi is the complete un-rounded pi
- brackets work using BEDMAS

**TASK**

complete some simple math expressions like 4 + 7

---

# Summary Functions

With new data, it can be helpful to generate some summary statistics to explore your data and understand what the data looks like. Some helpful summary statistic functions can be:

## nrow() and ncol()

These two functions will tell you the number of rows and number of columns in a data frame respectively.

```
ncol(shark_data)
```

```
## [1] 15
```

**TASK**

Determine the number of rows in the shark data frame.

---

## unique()

This is one of my favorites and it will help to identify all the unique values within an object. However, it might at first seem unhelpful like this:

```
unique(shark_data)
```

```
## # A tibble: 220 x 15
##       ID Binomial  Class Order Family Genus Species Common_name Location Country
##    <dbl> <chr>     <chr> <chr> <chr>  <chr> <chr>   <chr>       <chr>    <chr>
##  1  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  2  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  3  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  4  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  5  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  6  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  7  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  8  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  9  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
## 10  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
## # i 210 more rows
## # i 5 more variables: Region <chr>, Latitude <dbl>, Longitude <dbl>,
## #   Year <int>, Abundance <dbl>
```

This will just output the whole data frame, not helpful. But if we select one column using a $ it will give us more information.

```
unique(shark_data$Common_name)
```

```
##  [1] "Porbeagle shark"           "Thresher shark"
##  [3] "Blue shark"                "Shortfin mako"
##  [5] "Sand tiger shark"          "Great white shark"
##  [7] "Atlantic sharpnose shark"  "Blacktip shark"
##  [9] "Spinner shark"             "Blacknose shark"
## [11] "Bonnethead shark"          "Sandbar shark"
## [13] "Scalloped hammerhead"      "Finetooth shark"
## [15] "Tope shark"                "Broadnose sevengill shark"
## [17] "Smooth hammerhead"         "Lemon shark"
## [19] "Grey reef shark"           "Whitetip reef shark"
## [21] "Tiger shark"               "Blacktip reef shark"
## [23] "Tawny nurse shark"         "Silky shark"
## [25] "Whale shark"               "Silvertip shark"
## [27] "Round ribbontail ray"      "Ribbontail stingray"
```

Now we can see all the different common names for sharks in this data set.

**TASK**

What are the different Countries in this data set?

---

## names()

This function is useful for determining what the column names in your data set are.

```
names(shark_data)
```

```
##  [1] "ID"          "Binomial"    "Class"       "Order"       "Family"
##  [6] "Genus"       "Species"     "Common_name" "Location"    "Country"
## [11] "Region"      "Latitude"    "Longitude"   "Year"        "Abundance"
```

## mean(), max(), min(), median()

These will compute these basic mathematical equations on the data given to the function to summarize your data. Note for these you will need to give the function ONLY numeric data (numbers only no words). So to use mean() we should select the Abundance column.

```
mean(shark_data$Abundance)
```

```
## [1] 0.6136064
```

Now we know the mean abundance (catch per unit effort) for sharks in the whole data frame is 0.614

**TASK**

Determine the min, max and median for the abundance of sharks in the whole data frame.

---

## length()

This function will give the length of elements within an object. However this will tell us different things based on whether we have a data frame object or a vector.

Lets start by looking at the length of the shark data

```
length(shark_data)
```

```
## [1] 15
```

15? Does this look like a number we have seen before? It should because its the number of columns. So in a data frame, length() will tell us the number of columns. Now lets try selecting a single column in this function.

```
length(shark_data$ID)
```

```
## [1] 220
```

This should also look familiar because its the number of rows, so there are 220 elements in this column, which would be 220 rows.

We can also get the length of vectors.

**TASK**

Determine the length of the vector fruit

---

## sum()

This can be used to sum a column or a vector.

**TASK**

Try using sum() to sum our leaf_lengths vector.

---

# Data Manipulation

One initial thing that can be very helpful is manipulating your data so you can get it into the format you want to calculate more summary statistics. Like what if you wanted to know the average abundance of sharks for each country? or in each year? To help with this we can use:

## filter()

The filter() function can be used to filter your data set. So you can filter out certain rows so you only have attributes you want. We will take our shark data and filter out the "Whale shark" rows from the Common_name column and name this object whale_sharks. This is very useful for creating subsets of your original data. So we are looking for where Common_name is equal to "Whale shark"

```
whale_sharks <- filter(shark_data, Common_name == "Whale shark")

View(whale_sharks)
```

We can also filter with multiple criteria, we could filter out all "Shortfin mako" data that also has an abundance over 1 CPUE.

```
abund_short_mako <- filter(shark_data,
                           Common_name == "Shortfin mako" &
                             Abundance >= 1)
```

Some important symbols for the filter() are:

- "==" is equals exactly (a single = will not work in this usage)
- "!=" is does not equal
- ">" is greater than
- "<" is less than
- ">=" is greater than or equal to
- "<=" is less than or equal to
- "&" is and
- "|" is or

you can use any of these within the filter function to define what you are filtering out of a column.

**TASK**

Filter the shark data so there is only data from after 2010, and that the data set does not include the "International Waters" Region. Name this object Recent_Coastal_Sharks. Then check using functions to ensure this is correct

---

## select()

The select() can be used to select columns from your data, or to remove columns from your data. We will select the common name and abundance column for one data frame and for another we will select all except the ID column.

```
simple_shark_data <- select(shark_data, Common_name, Abundance)
# you can list as many columns as you want, just make sure they are
# spelled correctly and separated by a comma

head(simple_shark_data) # this will show us the first 6 rows of our data frame
```

```
## # A tibble: 6 x 2
##   Common_name     Abundance
##   <chr>               <dbl>
## 1 Porbeagle shark    0.0035
## 2 Porbeagle shark    0.0041
## 3 Porbeagle shark    0.0052
## 4 Porbeagle shark    0.0068
## 5 Porbeagle shark    0.0034
## 6 Porbeagle shark    0.0035
```

```
shark_data_noID <- select(shark_data, -ID)

head(shark_data_noID)
```

```
## # A tibble: 6 x 14
##   Binomial  Class Order Family Genus Species Common_name Location Country Region
##   <chr>     <chr> <chr> <chr>  <chr> <chr>   <chr>       <chr>    <chr>   <chr>
## 1 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 2 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 3 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 4 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 5 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 6 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## # i 4 more variables: Latitude <dbl>, Longitude <dbl>, Year <int>,
## #   Abundance <dbl>
```

## group_by()

The group_by() is useful when paired with a summarize function which we will learn next. This function will group your data based on the different values in a column. So for our data we could group our data by Common_name. This will mean we can then apply the summarize function to each different group in the Common_name column. So we could do something like average the abundance of each species of shark. Very useful. First lets group our data by the Common_name and call it sharks_grouped.

```
sharks_grouped <- group_by(shark_data, Common_name)
# you can group by as many columns as you would like, just make sure they
# spelled correctly and separated by a comma

#now to check that our data is grouped we can call the structure
str(sharks_grouped)
```

```
## gropd_df [220 x 15] (S3: grouped_df/tbl_df/tbl/data.frame)
##  $ ID         : num [1:220] 1934 1934 1934 1934 1934 ...
##  $ Binomial   : chr [1:220] "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" ...
##  $ Class      : chr [1:220] "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" ...
##  $ Order      : chr [1:220] "Lamniformes" "Lamniformes" "Lamniformes" "Lamniformes" ...
##  $ Family     : chr [1:220] "Lamnidae" "Lamnidae" "Lamnidae" "Lamnidae" ...
##  $ Genus      : chr [1:220] "Lamna" "Lamna" "Lamna" "Lamna" ...
##  $ Species    : chr [1:220] "nasus" "nasus" "nasus" "nasus" ...
##  $ Common_name: chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" "Porbeagle shark"
##  $ Location   : chr [1:220] "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-6, Canada" "NAFO Subareas 3
##  $ Country    : chr [1:220] "Canada" "Canada" "Canada" "Canada" ...
##  $ Region     : chr [1:220] "North America" "North America" "North America" "North America" ...
##  $ Latitude   : num [1:220] 43.3 43.3 43.3 43.3 43.3 ...
##  $ Longitude  : num [1:220] -54.8 -54.8 -54.8 -54.8 -54.8 ...
##  $ Year       : int [1:220] 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 ...
##  $ Abundance  : num [1:220] 0.0035 0.0041 0.0052 0.0068 0.0034 0.0035 0.0036 0.0025 0.0005 0.0015 ..
##  - attr(*, "groups")= tibble [28 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ Common_name: chr [1:28] "Atlantic sharpnose shark" "Blacknose shark" "Blacktip reef shark" "Bla
##   ..$ .rows      : list<int> [1:28]
##   .. ..$ : int [1:3] 149 150 151
##   .. ..$ : int [1:3] 158 159 160
##   .. ..$ : int [1:2] 203 204
##   .. ..$ : int [1:3] 152 153 154
##   .. ..$ : int [1:34] 35 36 37 38 39 40 41 42 43 44 ...
##   .. ..$ : int [1:3] 161 162 163
##   .. ..$ : int [1:2] 179 180
##   .. ..$ : int [1:3] 170 171 172
##   .. ..$ : int [1:33] 105 106 107 108 109 110 111 112 113 114 ...
##   .. ..$ : int [1:4] 193 194 195 196
##   .. ..$ : int [1:3] 190 191 192
##   .. ..$ : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ : int [1:2] 219 220
##   .. ..$ : int [1:2] 217 218
##   .. ..$ : int [1:33] 72 73 74 75 76 77 78 79 80 81 ...
##   .. ..$ : int [1:3] 164 165 166
##   .. ..$ : int [1:7] 167 168 169 205 206 207 208
##   .. ..$ : int [1:18] 58 59 60 61 62 63 64 65 66 67 ...
##   .. ..$ : int [1:2] 211 212
##   .. ..$ : int [1:2] 215 216
##   .. ..$ : int [1:9] 181 182 183 184 185 186 187 188 189
##   .. ..$ : int [1:3] 155 156 157
##   .. ..$ : int [1:2] 209 210
##   .. ..$ : int [1:22] 13 14 15 16 17 18 19 20 21 22 ...
##   .. ..$ : int [1:2] 201 202
##   .. ..$ : int [1:2] 177 178
##   .. ..$ : int [1:2] 213 214
##   .. ..$ : int [1:4] 197 198 199 200
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

Great, now we can see that they type of data set has changed to a grouped data set.

## summarize()

Now lets use the summarize function to average the abundance of our sharks. We get to pick the column name we want to have R put these new averages in, I am going to choose Average_Abund

```
shark_averages <- summarize(sharks_grouped, Average_Abund = mean(Abundance))

shark_averages
```

```
## # A tibble: 28 x 2
##    Common_name                Average_Abund
##    <chr>                              <dbl>
##  1 Atlantic sharpnose shark         1.44
##  2 Blacknose shark                  0.162
##  3 Blacktip reef shark              0.115
##  4 Blacktip shark                   0.847
##  5 Blue shark                       1.57
##  6 Bonnethead shark                 0.810
##  7 Broadnose sevengill shark        0.0162
##  8 Finetooth shark                  0.641
##  9 Great white shark                0.551
## 10 Grey reef shark                  1.00
## # i 18 more rows
```

Our final data frame should look like this, if you only have one single value, you probably used the original shark_data data frame so it just took an average of the entire data set, make sure you use the grouped data frame.

## mutate()

The last function we will talk about before discussing how to make this process quicker is the mutate(). This can be used to mutate columns and create new columns. Maybe we want to get a log transformed abundance column, we can do this with the mutate function. We get to specify what the new column will be, for use we will call it log_Abundance and then you specify what the formula will be to mutate a column.

```
shark_data_log <- mutate(shark_data, log_Abundance = log(Abundance))

str(shark_data_log)
```

```
## tibble [220 x 16] (S3: tbl_df/tbl/data.frame)
##  $ ID           : num [1:220] 1934 1934 1934 1934 1934 ...
##  $ Binomial     : chr [1:220] "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" ...
##  $ Class        : chr [1:220] "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" ..
##  $ Order        : chr [1:220] "Lamniformes" "Lamniformes" "Lamniformes" "Lamniformes" ...
##  $ Family       : chr [1:220] "Lamnidae" "Lamnidae" "Lamnidae" "Lamnidae" ...
##  $ Genus        : chr [1:220] "Lamna" "Lamna" "Lamna" "Lamna" ...
##  $ Species      : chr [1:220] "nasus" "nasus" "nasus" "nasus" ...
##  $ Common_name  : chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" "Porbeagle shark
##  $ Location     : chr [1:220] "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-6, Canada" "NAFO Subareas
##  $ Country      : chr [1:220] "Canada" "Canada" "Canada" "Canada" ...
##  $ Region       : chr [1:220] "North America" "North America" "North America" "North America" ...
##  $ Latitude     : num [1:220] 43.3 43.3 43.3 43.3 43.3 ...
```

```
##  $ Longitude    : num [1:220] -54.8 -54.8 -54.8 -54.8 -54.8 ...
##  $ Year         : int [1:220] 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 ...
##  $ Abundance    : num [1:220] 0.0035 0.0041 0.0052 0.0068 0.0034 0.0035 0.0036 0.0025 0.0005 0.0015
##  $ log_Abundance: num [1:220] -5.65 -5.5 -5.26 -4.99 -5.68 ...
```

### Combining with a Pipe

Doing each of these data manipulation steps individually is fine, but it will take a lot of intermediate data objects. This can clutter up your environment and make it quite hard to pick a new name for you data set after you have gone through 12 different names. Instead we can use a pipe (%>%) to feed our first line into the second and so forth to combine all these different steps together. So instead of doing a new data frame after a filter, group_by and summarize, we can do all three of those steps together and name them sharks_summarized. With this, the intermediate data frames don't need to be saved as an object, and you don't have to specify the data frame for each set.

```
sharks_summarized <- shark_data %>%
  filter(Region != "International Waters") %>%
  group_by(Common_name) %>%
  summarise(Average_Abund = mean(Abundance))

head(sharks_summarized)
```

```
## # A tibble: 6 x 2
##   Common_name              Average_Abund
##   <chr>                            <dbl>
## 1 Atlantic sharpnose shark         1.44
## 2 Blacknose shark                  0.162
## 3 Blacktip reef shark              0.115
## 4 Blacktip shark                   0.847
## 5 Blue shark                       1.28
## 6 Bonnethead shark                 0.810
```

In this pipe you can now see that we only specify shark_data at the start, then that data set is fed into the next line with the pipe to filter our data, that filtered data is fed into the group_by() and that grouped data is fed into the final summarize() function and that result is what is saved into the object sharks_summarized.

**TASK**

Your turn, create one object regional_shark_abund where I want you to use pipes to filter all data after 2000 from the shark_data data frame, group by Region and Year, then sum all the abundance so we get the total shark abundance in CPUE for each region in each year after 2000.

---

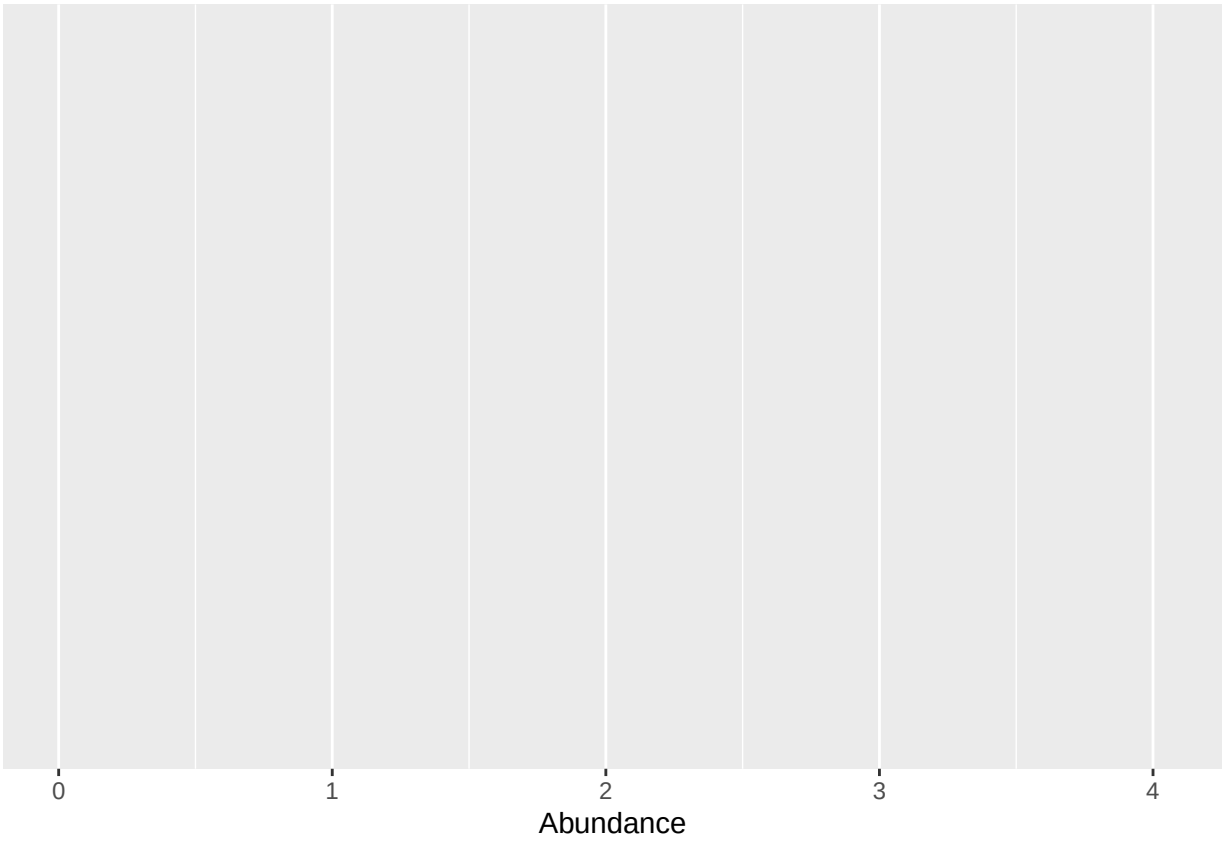
## Plotting Data

To plot data we are going to use the ggplot2 package, which is a great package for data visualization. All ggplots are built up in layer, and you keep adding layers to your plot until it looks like the plot you want. You can save your plots as an object so you can export them later, or just have them output without saving.

## Histogram

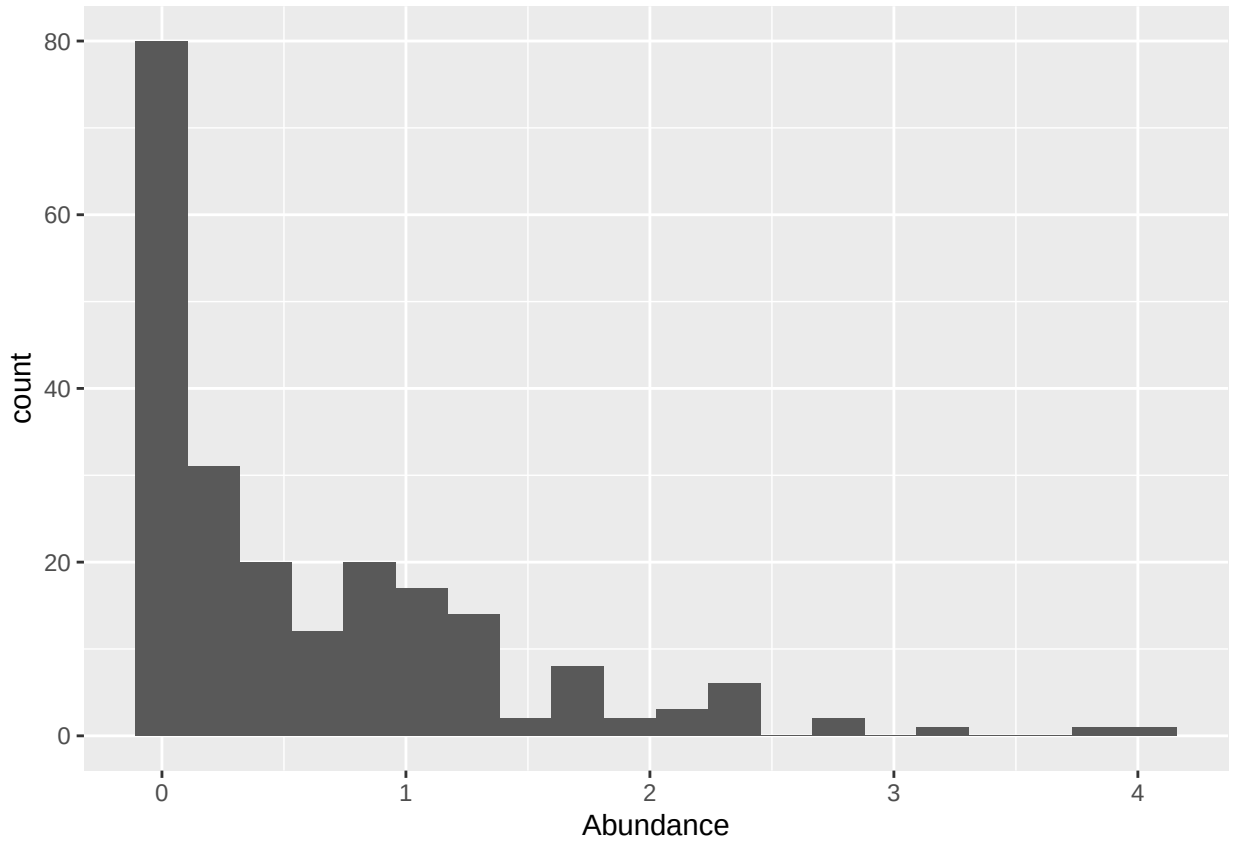
The steps for making these plots is to first specify you are making a plot with the ggplot(). You specify which data frame you want to plot data from, and then specify the aes() or aesthetics of your plot. This is where you tell R which columns in the data set are going on the x axis, y axis, or are to be used as a color or shape argument. We will start by making a histogram of our abundance data, so this will only have an x axis, which will be abundance.

```
Hist_step1 <- ggplot(shark_data, aes(x = Abundance))

Hist_step1
```



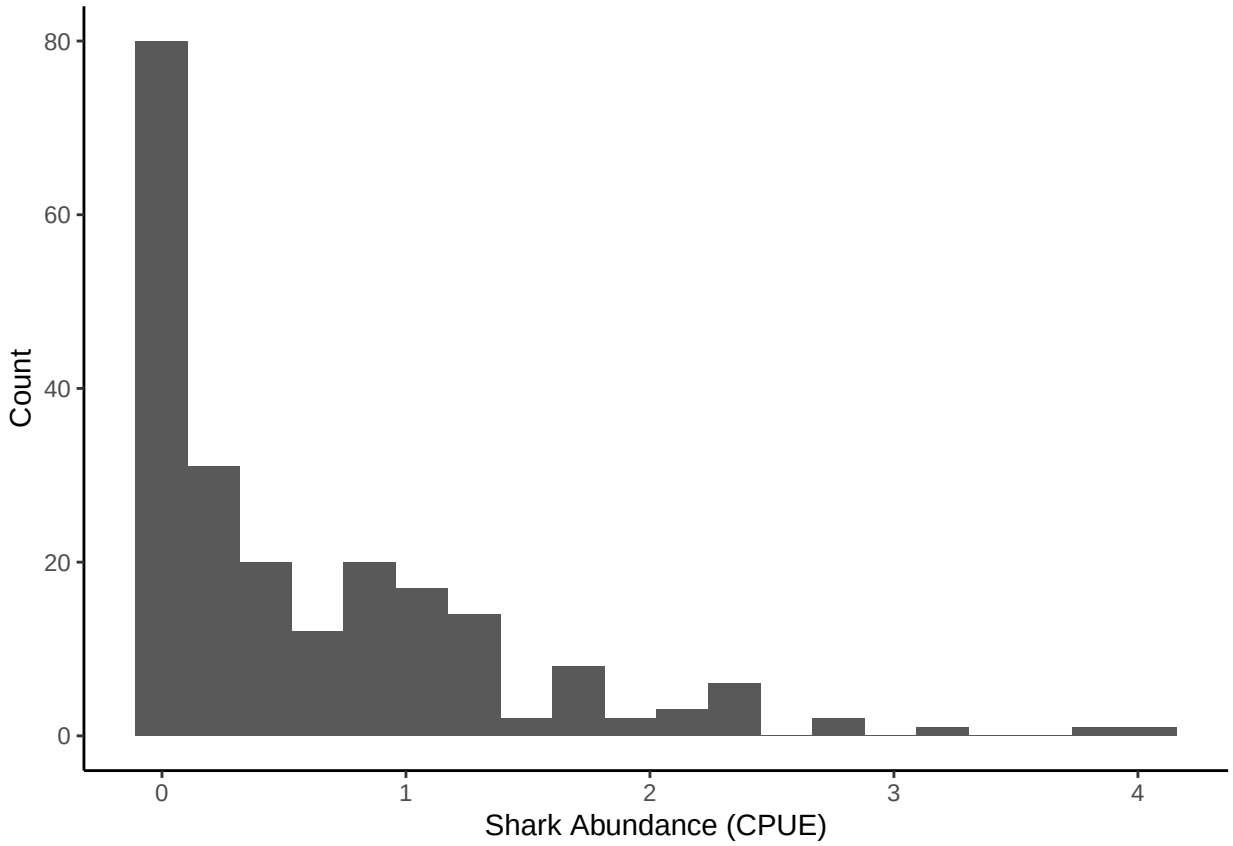

Now we have a blank plot, because R doesn't know what kind of plot it is making, so the next line will be to add a geometric argument to specify it is a histogram we are making. We will also add in an argument about number of bins which you can play with the change how many histogram bins are being made.

```
Hist_step2 <- ggplot(shark_data, aes(x = Abundance)) +
  geom_histogram(bins = 20)

Hist_step2
```



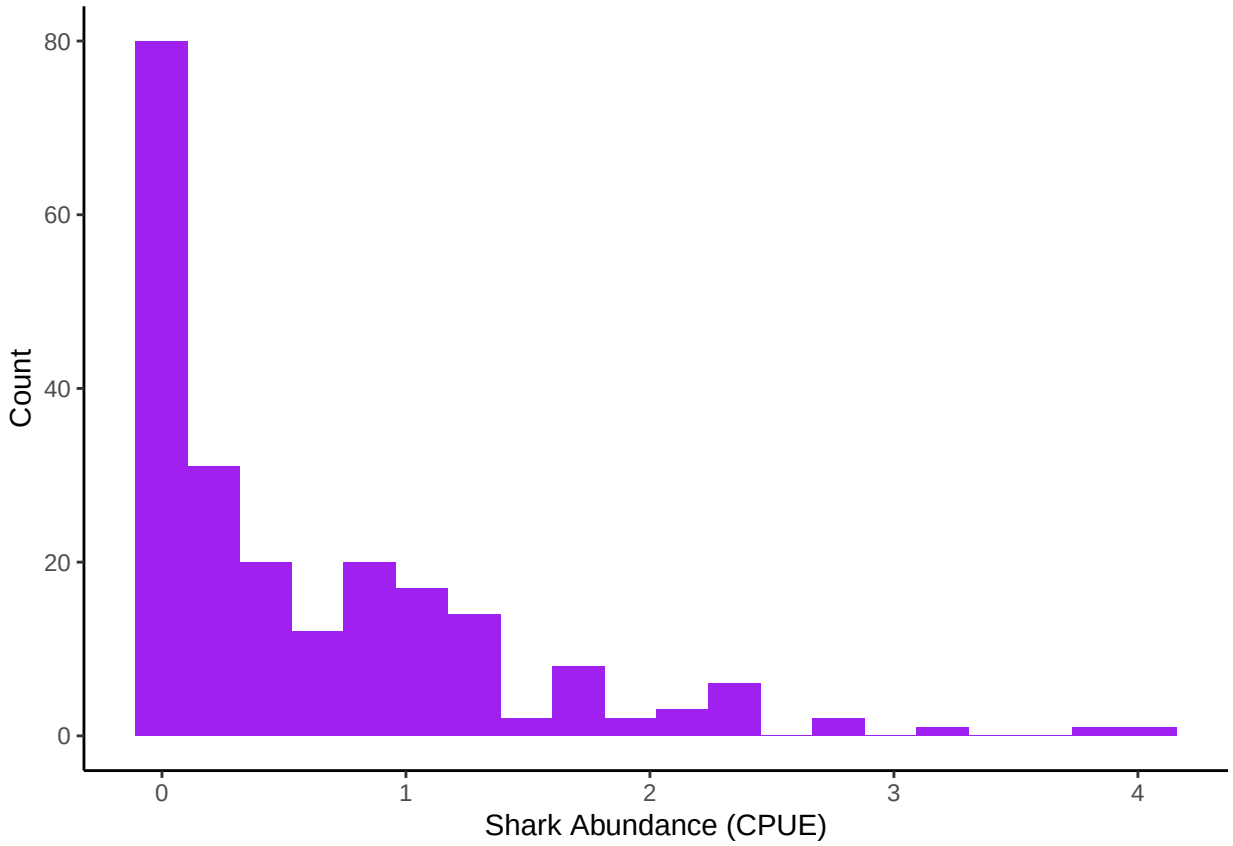

Yay, its a plot! Now lets make it a bit nicer and more customized. There are many themes that have been made already which we can use to make our plots more aesthetic, I like theme_classic(). We can also use the labs() to change our axis labels.

```
Hist_step3 <- ggplot(shark_data, aes(x = Abundance)) +
  geom_histogram(bins = 20) +
  theme_classic() +
  labs(x = "Shark Abundance (CPUE)",
       y = "Count")

Hist_step3
```



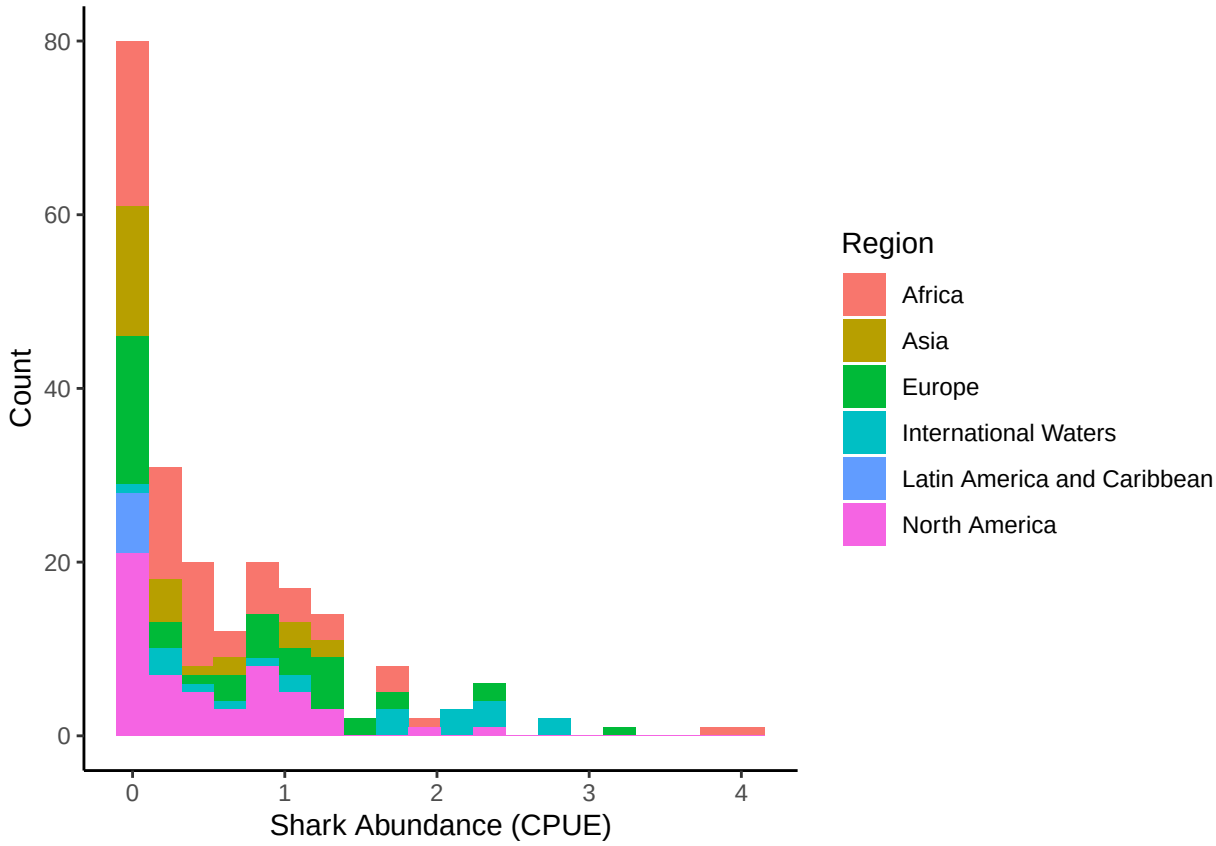

Another thing we can do is add a color to this plot. We could change all the color of the bins to say a darkpurple by adding a fill argument to the geom_histogram()

```
Hist_darkpurple <- ggplot(shark_data, aes(x = Abundance)) +
  geom_histogram(bins = 20, fill = "purple") +
  theme_classic() +
  labs(x = "Shark Abundance (CPUE)",
       y = "Count")
Hist_darkpurple
```



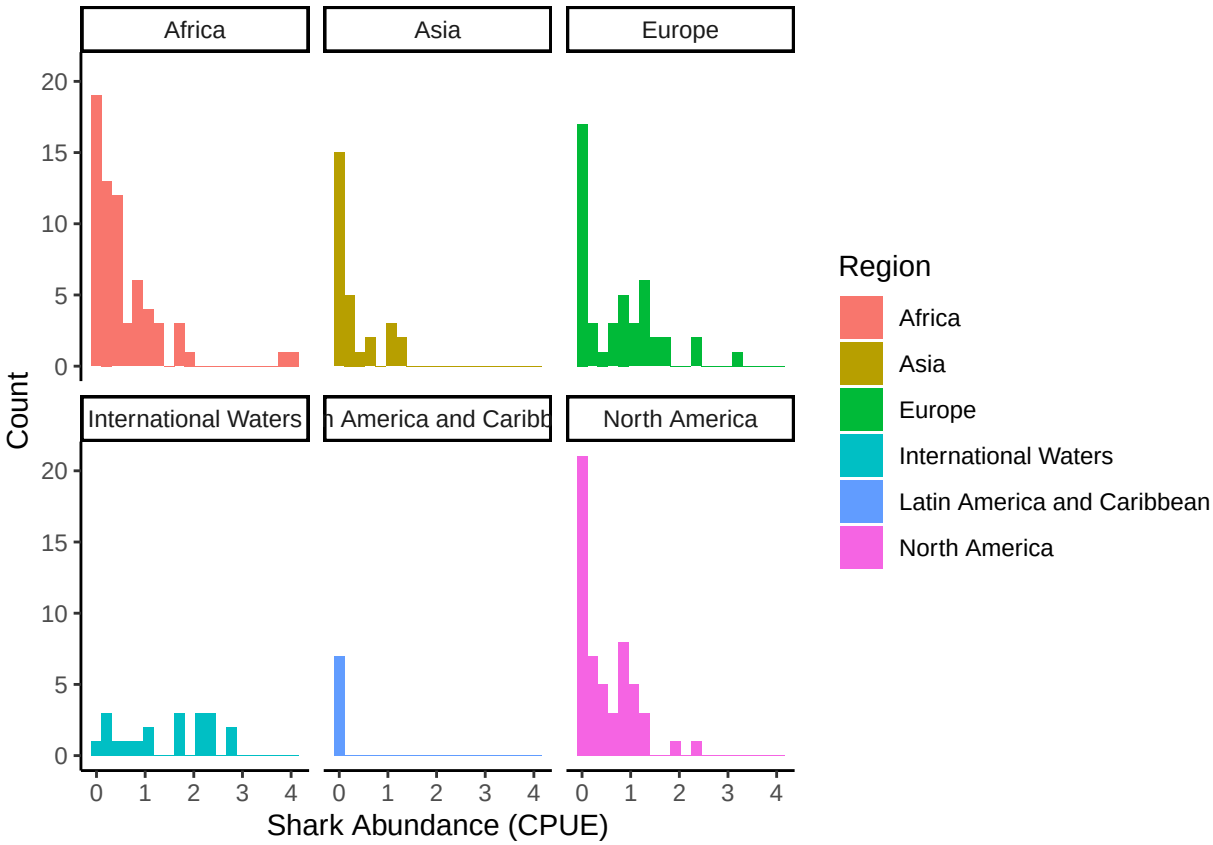

or maybe we want each region to be its own fill color. If we want the fill to change based on a variable, this needs to be put in the aesthetics statement.

```
Hist_regions <- ggplot(shark_data, aes(x = Abundance, fill = Region)) +
  geom_histogram(bins = 20) +
  theme_classic() +
  labs(x = "Shark Abundance (CPUE)",
       y = "Count")
Hist_regions
```




This if getting there but it is a bit difficult to actually see the different distributions of the various regions. The last think we will add to this will be a facet_wrap() which will create a new mini figure for each attribute of a variable.

```
Hist_wrap <- ggplot(shark_data, aes(x = Abundance, fill = Region)) +
  geom_histogram(bins = 20) +
  theme_classic() +
  labs(x = "Shark Abundance (CPUE)",
       y = "Count") +
  facet_wrap(~Region)
Hist_wrap
```




Beautiful! There are ways to customize which colors are used so if you are interested in that, I would recommend a little google. The final step will be to save this figure. We will use the jpeg() to save as it is very simple and we can customize the size, resolution and location we save the figure to. Since we are using jpeg() we will have to save our image with a .jpeg extension. You could use png() or tiff() just make sure you change the file extension. We will name the file Histogram_Example.jpeg, and specify the width and height in inches and a resolution in DPI. Then the next line is where we call the plot, and the final line is dev.off() which will just close the saving process. You have to run all the lines together for it to work best. This will save the figure right to your working directory, so make sure this is set right, and if you want it in a folder, make sure you change the file name to reflect that.

```
jpeg(filename = "Histogram_Example.jpeg",
     units = "in", width = 8, height = 6, res = 300)
Hist_wrap
dev.off()
```

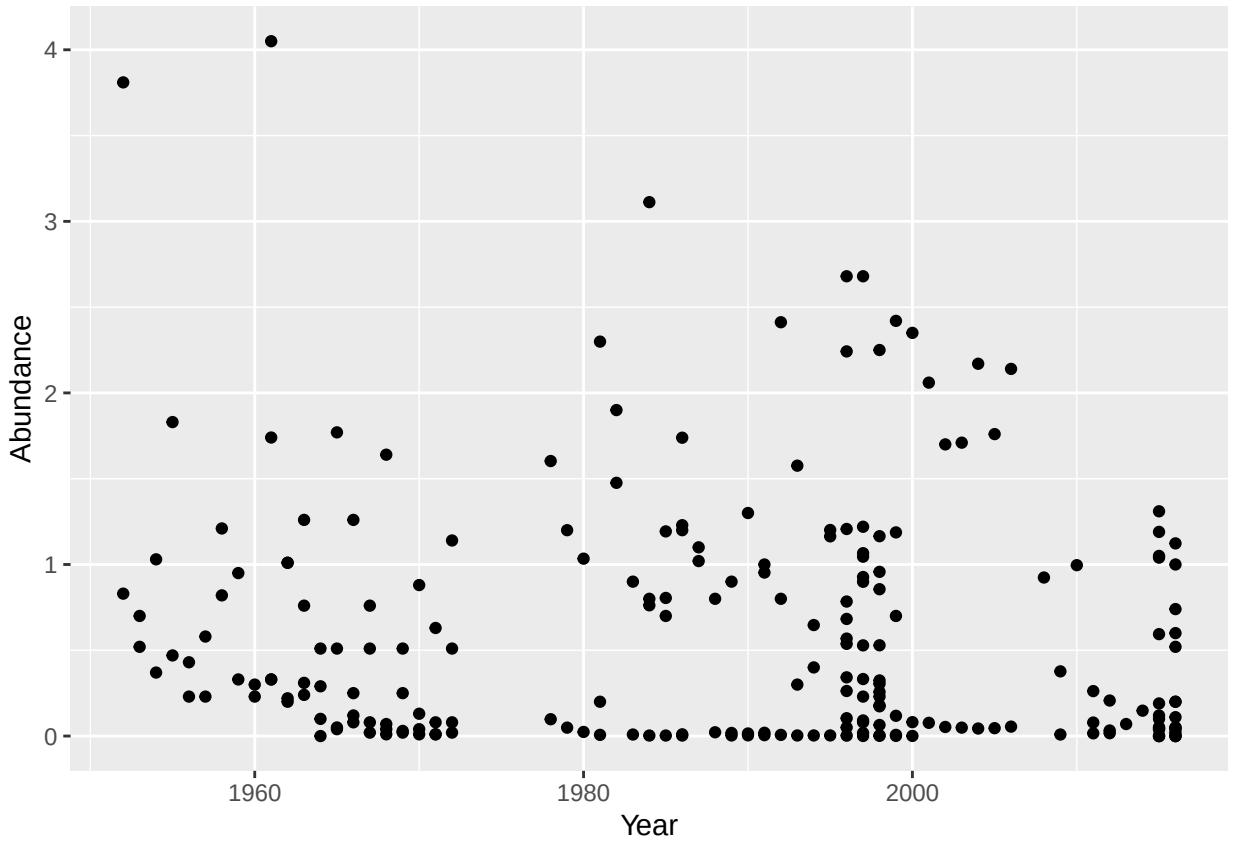
```
## pdf
##   2
```

## Scatter Plot

Scatter plots are very similar to histograms, except you have to specify which variable is on the y axis and also instead of geom_histogram() you use geom_point() to add points to your plot. We will plot the abundance by year.

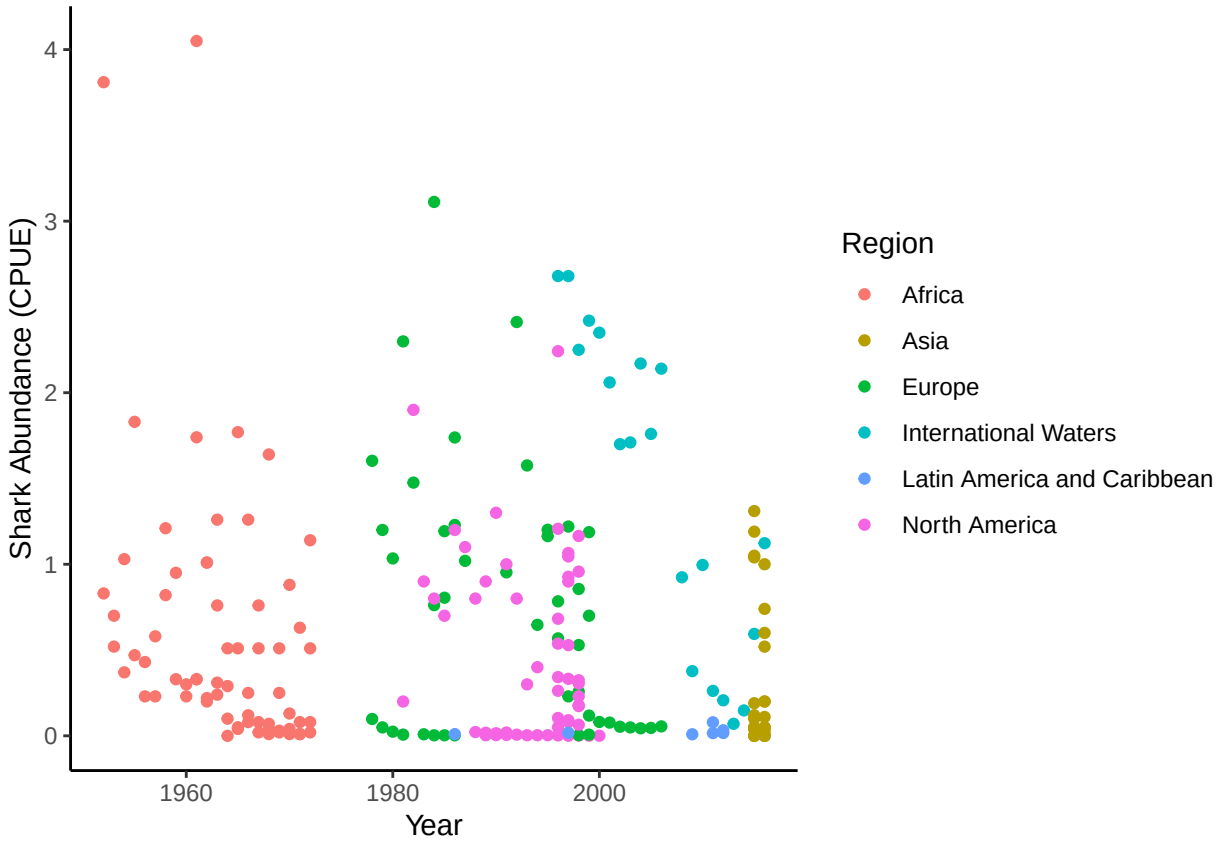
```
Scatter_step1 <- ggplot(shark_data, aes(x = Year, y = Abundance)) +
  geom_point()
```

```
Scatter_step1
```




And you can use all the same code from the histogram example to customize this plot too. For geom_point() to change the color you will use a color argument in the aesthetic (instead of fill)

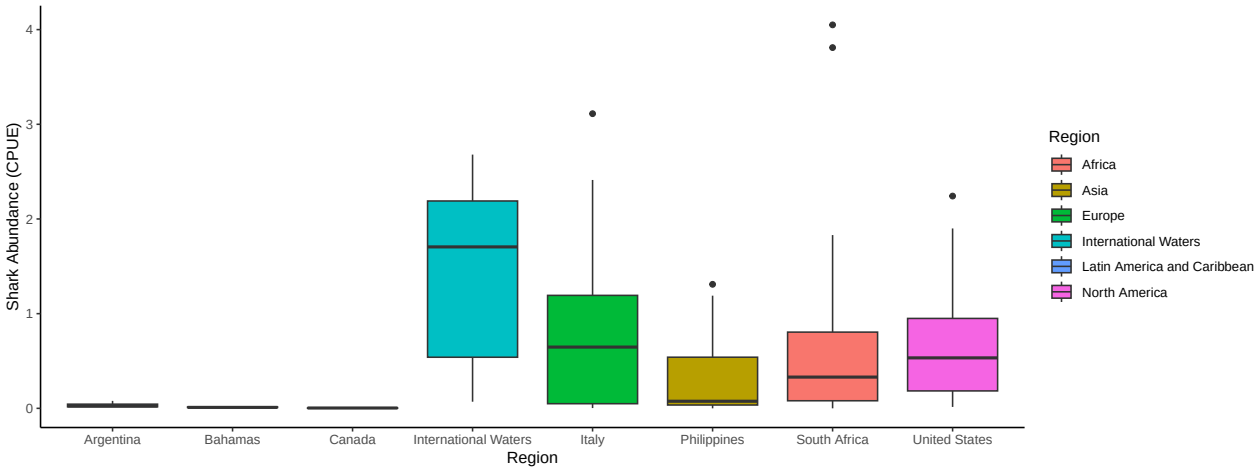
```
Scatter_clean <- ggplot(shark_data, aes(x = Year, y = Abundance, color = Region)) +
  geom_point() +
  theme_classic() +
  labs(x = "Year",
       y = "Shark Abundance (CPUE)")

Scatter_clean
```




## Box Plot

Box plot will work the same way but instead of geom_histogram() or geom_point(), it is geom_box plot() and it will use a fill aesthetic argument instead of color. Here we will plot abundance by country

```
Boxplot_clean <- ggplot(shark_data, aes(x = Country, y = Abundance, fill = Region)) +
  geom_boxplot() +
  theme_classic() +
  labs(x = "Region",
       y = "Shark Abundance (CPUE)")

Boxplot_clean
```




Making figures can start very simple and you can slowly increase the complexity.

**TASK**

Time to try to make your own figure. Try making a scatter plot that plots shark abundance by year, but has a facet of each region and each point is colored by the country.

---

# Stats Intro

Now I don't want to go into the theory behind each of these test for this workshop as the goal is to learn how to code. So if you don't know what these tests are or what the different assumptions are of these tests, I highly recommend reading up on them before you conduct them on your own data. We are going to use some different data for the stats portion, penguins, this data is already in R and we have already loaded the package from R so to access it all we have to do is call on the data. There are a lot of fun variables in this data but we will be examining the flipper_length_mm to start and will add in bill_length_mm later. But feel free to explore this data more on your own.

```
data(penguins)

head(penguins)
```

```
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
## 1 Adelie  Torgersen           39.1          18.7               181        3750
## 2 Adelie  Torgersen           39.5          17.4               186        3800
## 3 Adelie  Torgersen           40.3          18                 195        3250
## 4 Adelie  Torgersen           NA            NA                  NA          NA
## 5 Adelie  Torgersen           36.7          19.3               193        3450
## 6 Adelie  Torgersen           39.3          20.6               190        3650
## # i 2 more variables: sex <fct>, year <int>
```

## T-test and Wilcoxon rank sum test

First we will examine a two-sample t-test and its non-parametric equivalent, the Wilcoxon rank sum test. These tests are for determining differences between two groups of numeric data.

There are three kinds of t-tests. A two sample t-test means we have two groups, while a one sample t-test looks at one group compared to a constant, usually a known "true" mean. The other kind is a paired test, which is usually a before and after style experiment where you want to know if something had an effect so you compare before and after application.

Two sample t-tests are the most common kind of t-tests because we are often trying to figure out if the mean of two groups differ from each other. For our example we will look at the difference in mean flipper length between the Adelie and Chinstrap penguins. We will create two data sets, one for each penguin species, as well one that contains both penguin species.

The difference between a two-sample t-test and a Wilcoxon test is the underlying assumptions. T-tests have the assumption that each group in the data is normal and that the groups have equal variance between them. Alternatively the Wilcoxon test does not have these assumptions, but may not be as powerful of a test, so you may not find a difference that exists. So to determine which test to use we need to determine if our data is normal and if they have equal variance.

```
unique(penguins$species)
```

```
## [1] Adelie    Gentoo    Chinstrap
## Levels: Adelie Chinstrap Gentoo
```

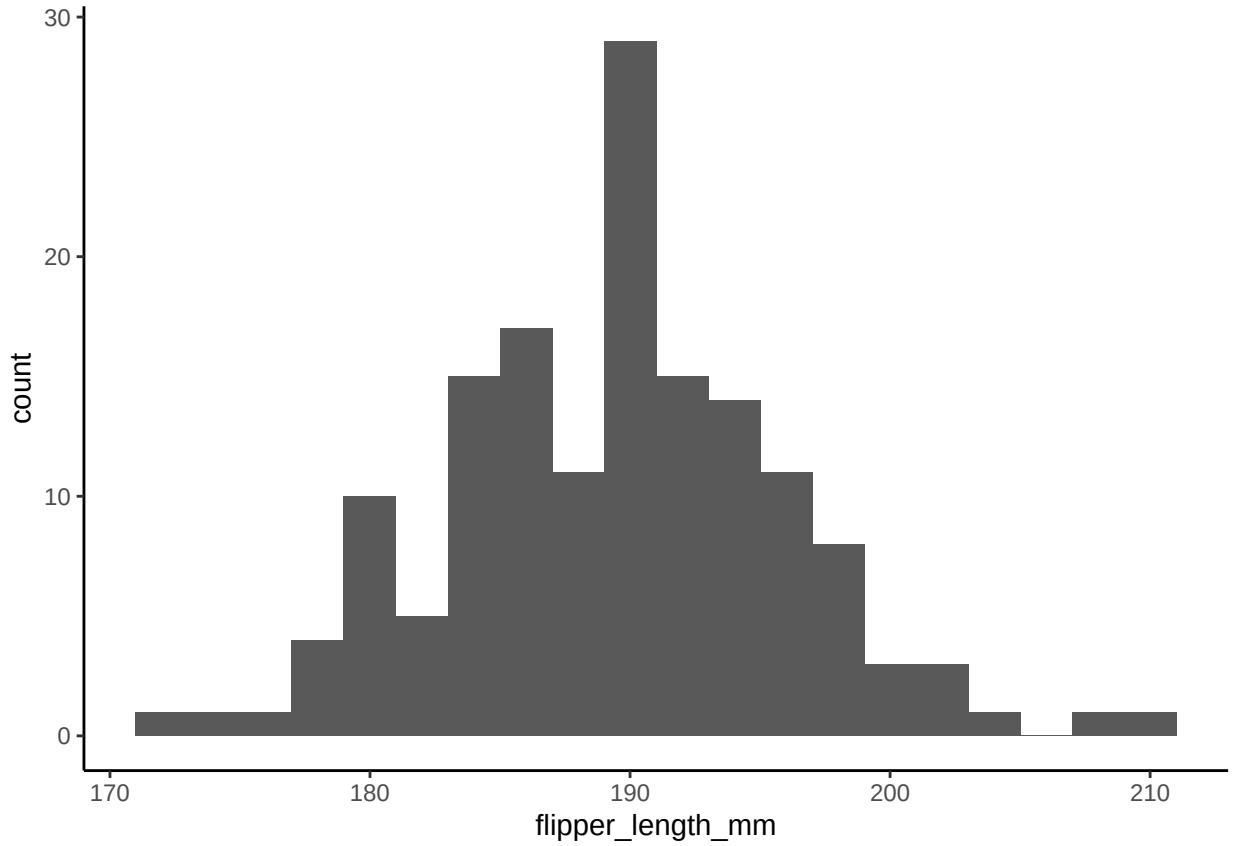
```
Adelie <- penguins %>%
  filter(species == "Adelie")

Chinstrap <- penguins %>%
  filter(species == "Chinstrap")

Adelie_n_Chinstrap <- penguins %>%
  filter(species == "Adelie" | species == "Chinstrap")
```

To assess the normality of this data for our t-test, you can do this visually with a histogram as we did above, or if you want to use a statistical test, we can use a Shapiro Wilks test. This is for testing if there is normality in a set of continuous numeric data. If you are using different groups in your analysis, you will want to subset your data into your various groups and conduct a Shapiro Wilks test on each group. We are going to do this for each species of penguin to examine their flipper length.

```
shapiro.test(Adelie$flipper_length_mm)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Adelie$flipper_length_mm
## W = 0.99339, p-value = 0.72
```

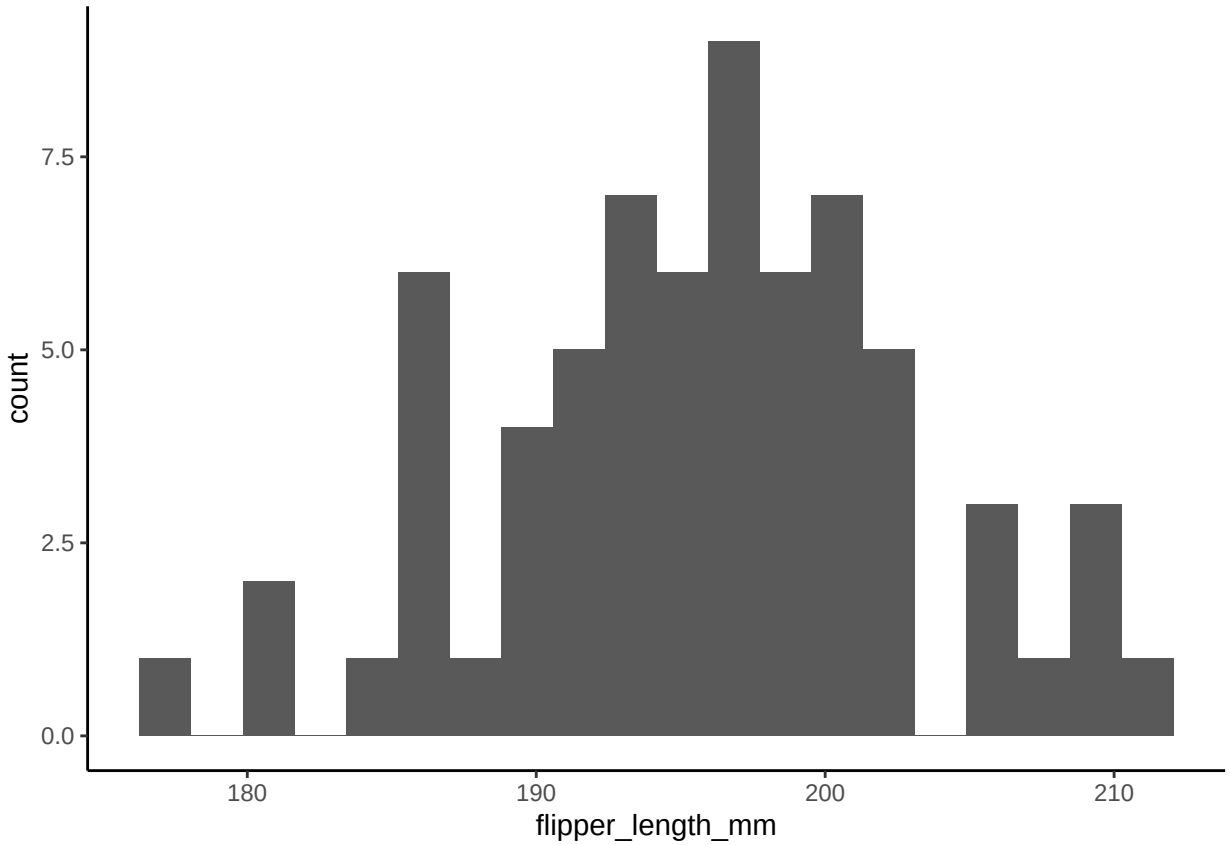
```
ggplot(Adelie, aes(x = flipper_length_mm)) +
  geom_histogram(bins = 20) +
  theme_classic()
```




```
shapiro.test(Chinstrap$flipper_length_mm)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Chinstrap$flipper_length_mm
## W = 0.98891, p-value = 0.8106
```

```
ggplot(Chinstrap, aes(x = flipper_length_mm)) +
  geom_histogram(bins = 20) +
  theme_classic()
```




These results show us that the Adelie and Chinstrap data is normal ($p > 0.05$) and by looking at the histogram it is approximately normal, meaning it follows a nice bell shaped distribution.

Now to test if there are even variances between the different groups, we can conduct a LeveneTest(). This does not need to be done on each subset of data, instead you can specify the variable ~ group.

```
leveneTest(Adelie_n_Chinstrap$flipper_length_mm ~ Adelie_n_Chinstrap$species)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   1  0.6238 0.4305
##       217
```

This test shows we have equal variance between our groups as our p-value is large ($p > 0.05$).

Our data meets the assumptions for a test, so we can go ahead and conduct this test. We will specify the variable ~ groups and which data set the variables are coming from. Finally we will specify that var.equal = TRUE which means we have equal variances, which we know based on the results of the Levene's test above.

```
t.test(flipper_length_mm ~ species , data = Adelie_n_Chinstrap,
       var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  flipper_length_mm by species
## t = -5.974, df = 217, p-value = 9.379e-09
```

```
## alternative hypothesis: true difference in means between group Adelie and group Chinstrap is not equa
## 95 percent confidence interval:
##  -7.806481 -3.933293
## sample estimates:
##    mean in group Adelie mean in group Chinstrap
##                189.9536                195.8235
```

Here we can see that there is a very low p-value ($p < 0.05$) so there is a significant difference between the mean flipper length of Adelie and Chinstrap penguins.

**TASK**

We can see by the means that Adelie have on average smaller flippers but you could also try plotting this with a box plot to visualize it. Make a pretty box plot to display this result.

---

Now we were able to conduct a t-test because our data was normal and homoscedastic, but if it wasn't we would perform a Wilcoxon test instead. We can still do this, though it won't be as powerful as the t-test. To code this test, it will look a lot like the t-test, but using wilcox.test() and this test is examining if the medians are different between the two groups, instead of the means like a t-test is doing.

```
wilcox.test(flipper_length_mm ~ species , data = Adelie_n_Chinstrap)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  flipper_length_mm by species
## W = 2733.5, p-value = 3.028e-08
## alternative hypothesis: true location shift is not equal to 0
```

This gives us the same result, that there is a difference in the median flipper length between penguin species ($p < 0.05$).

## ANOVA and Kruskal-Wallis tests

Now a t-test and Wilcoxon test are great if you only have 2 groups, but often we have more than 2 groups, and that is where the ANOVA and Kruskal-Wallis tests come in. Because ANOVA is like a t-test but it can have 2 or more groups, we can examine all three penguin species. The Kruskal-Wallis test is the nonparametric equivalent of an ANOVA, so it does not require data to be normal or homoscedastic, and it is like the Wilcoxon test but with more than 2 groups.

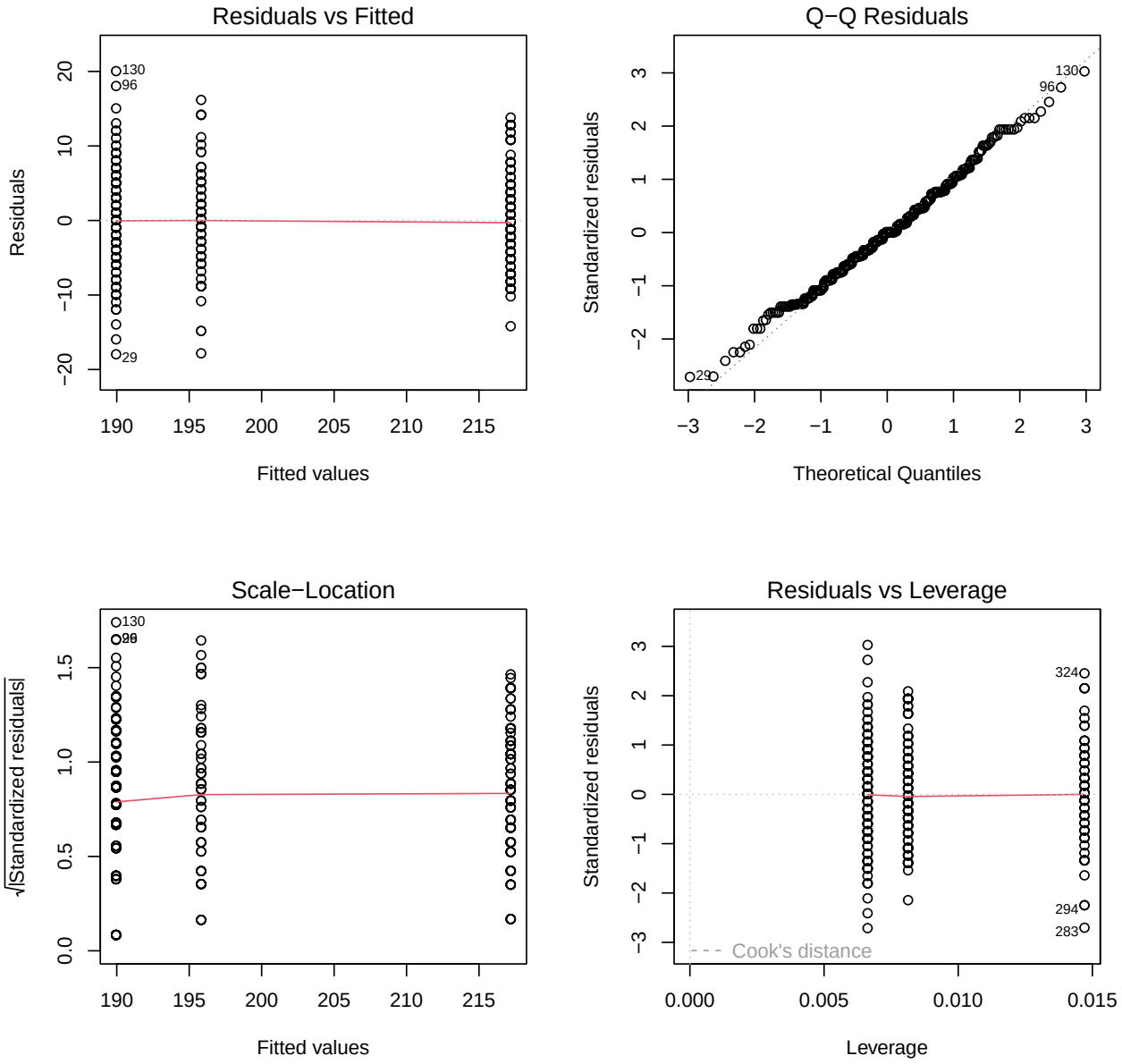
In order to test the assumptions of our data (normality and homeostaticity) we could do what we did before, subset the groups and use a Shapiro Wilks test on each group, but just imagine if we had 10 groups, that would take forevvvvver. Instead we can build the ANOVA, and then plot and test the residuals. You can look more into the theory behind this, but by examining the residuals after being fit by the ANOVA, we can see if our data will follow a normal distribution and is homoscedastic without having to break it apart into each group.

To build the ANOVA, we will use the aov() and save it as an object. We will code the same kind of equation like we did with the t-test.

```
penguin_aov <- aov(flipper_length_mm ~ species, data = penguins)
```

Now we can plot this object and look at the first two plots

```
par(mfrow = c(2,2)) # this will just output the plots together in a 2x2 grid
plot(penguin_aov)
```




With the first plot, the residuals vs fitted plot, we are looking for the red line to be centered on 0, and for the data to be even below and above the red line, we don't want any kind of wedge shape going on. This shows our data meets the assumption of homoscedasticity visually. Then the second plot, we want our data to nicely follow that red 1:1 line. We don't want it to show an S shape. This plot shows our data meets the assumption of normality visually. This is the most common way that ecologists will check that their data meets the assumptions of statistical tests.

If we want to check these assumptions with a test, we can use the Shapiro.test() and leveneTest() again, but this time the Shapiro test will be on the residuals of the anova object.

```r
shapiro.test(penguin_aov$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  penguin_aov$residuals
## W = 0.99452, p-value = 0.2609
```

```r
leveneTest(flipper_length_mm ~ species, data = penguins)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##        Df F value Pr(>F)
## group   2  0.3306 0.7188
##       339
```

This test also shows that our data meets the assumptions of an ANOVA.

Since we have already made the ANOVA object, we don't need to conduct the test again, but now we can call the results with the summary() of that ANOVA object.

```r
summary(penguin_aov)
```

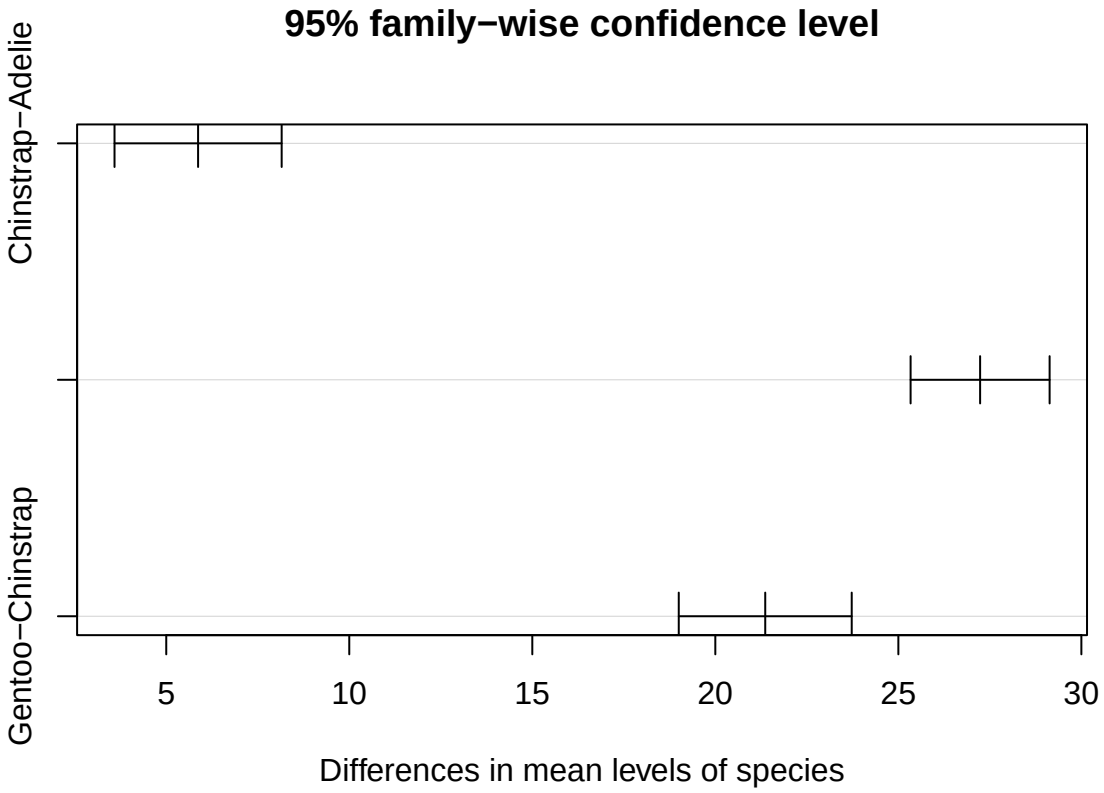
```
##              Df Sum Sq Mean Sq F value Pr(>F)
## species       2  52473   26237   594.8 <2e-16 ***
## Residuals   339  14953      44
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 2 observations deleted due to missingness
```

This p-value for species is very small, showing that there is a significant difference in the mean flipper length of at least one species of penguin, but as you can see, we don't know where that difference is, or even how many there are. That is where the Tukey HSD test comes in, it will essentially do a bunch of pairwise t-tests to determine which groups differ from each other

```r
TukeyHSD(penguin_aov)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = flipper_length_mm ~ species, data = penguins)
##
## $species
##                       diff       lwr       upr p adj
## Chinstrap-Adelie  5.869887  3.586583  8.153191     0
## Gentoo-Adelie    27.233349 25.334376 29.132323     0
## Gentoo-Chinstrap 21.363462 19.000841 23.726084     0
```

```r
plot(TukeyHSD(penguin_aov))
```




Now to tell if there are differences between each species pair, we look at the p adj column which shows the adjusted p value after accounting for this being a *post hoc* test (I suggest a little google to learn more about why this p value would be adjusted). All of these show as 0, which means they are really really small. So with p « 0.05 we can say there is a significant difference in flipper length between all 3 penguin species.

Just like with the t-test, because our data was normal and homoscedastic, it is more powerful to use an ANOVA, but we could also do a Kruskal-Wallis, which is the test you would use if your data was not normal or had uneven variances.

This code is super similar to the Wilcoxon rank sum test code and instead of a Tukey test we will use a Dunn Test, which is the nonparametric equivalent. For the Dunn test we will specify we want to use the bonferroni correction and that we want to view it in a list, this will make it look very similar to a Tukey test.

```
penguin_kruskal <- kruskal.test(flipper_length_mm ~ species, data = penguins)
penguin_kruskal
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  flipper_length_mm by species
## Kruskal-Wallis chi-squared = 244.89, df = 2, p-value < 2.2e-16
```

```
dunn.test(penguins$flipper_length_mm, penguins$species,
          method = "bonferroni", list=TRUE)
```

```
##   Kruskal-Wallis rank sum test
##
## data: x and group
```

```
## Kruskal-Wallis chi-squared = 244.8905, df = 2, p-value = 0
##
##
##                             Comparison of x by group
##                                   (Bonferroni)
## Col Mean-|
## Row Mean |     Adelie   Chinstra
## ---------+----------------------
## Chinstra |  -3.629336
##          |    0.0004*
##          |
##   Gentoo |  -15.47661  -8.931938
##          |    0.0000*    0.0000*
##
##
## List of pairwise comparisons: Z statistic (adjusted p-value)
## ----------------------------------------
## Adelie - Chinstrap : -3.629336 (0.0004)*
## Adelie - Gentoo    : -15.47661 (0.0000)*
## Chinstrap - Gentoo : -8.931938 (0.0000)*
##
## alpha = 0.05
## Reject Ho if p <= alpha/2
```

And our results show that the median flipper length for each species pair is significantly different from the other since we have such small p values (p « 0.05)

**TASK**

Determine which test to use as you examine if there is a significant difference in body mass between penguin sex. Then plot this relationship with a box plot

---

## cor.test

If your data doesn't have groups, but is two continuous variables, one thing you can look at is if there is a correlation between the two variables. Now a correlation doesn't mean there is causation, so this test isn't looking at if one variable affects the other, its just looking at if the variables change together.

There are two kinds of correlation tests that we will discuss, parametric (Pearson) and nonparametric (Spearman). We will first determine if our two variables are normal and then decide which test to use. We will look at if beak length differs with flipper length for our penguins.

```
shapiro.test(penguins$bill_length_mm)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  penguins$bill_length_mm
## W = 0.97485, p-value = 1.12e-05
```

```r
shapiro.test(penguins$flipper_length_mm)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  penguins$flipper_length_mm
## W = 0.95155, p-value = 3.54e-09
```

These tests show that all our data together are not normally distributed for either variable ($p < 0.05$), so we will use the nonparametric correlation test the Spearman's rank correlation

```r
cor.test(penguins$bill_length_mm, penguins$flipper_length_mm, method = "spearman")
```

```
##
##  Spearman's rank correlation rho
##
## data:  penguins$bill_length_mm and penguins$flipper_length_mm
## S = 2181594, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.6727719
```

This shows there is a significant correlation between bill length and flipper length. Now this doesn't mean larger bill length causes larger flipper length, that doesn't make sense anyway, it just means the two variables change together, which probably has something to do with penguins getting larger.

**TASK**

Plot bill length by flipper length to see the correlation between the variables. You could even look into adding a line with geom_smooth() to the plot if you want for bonus learning.

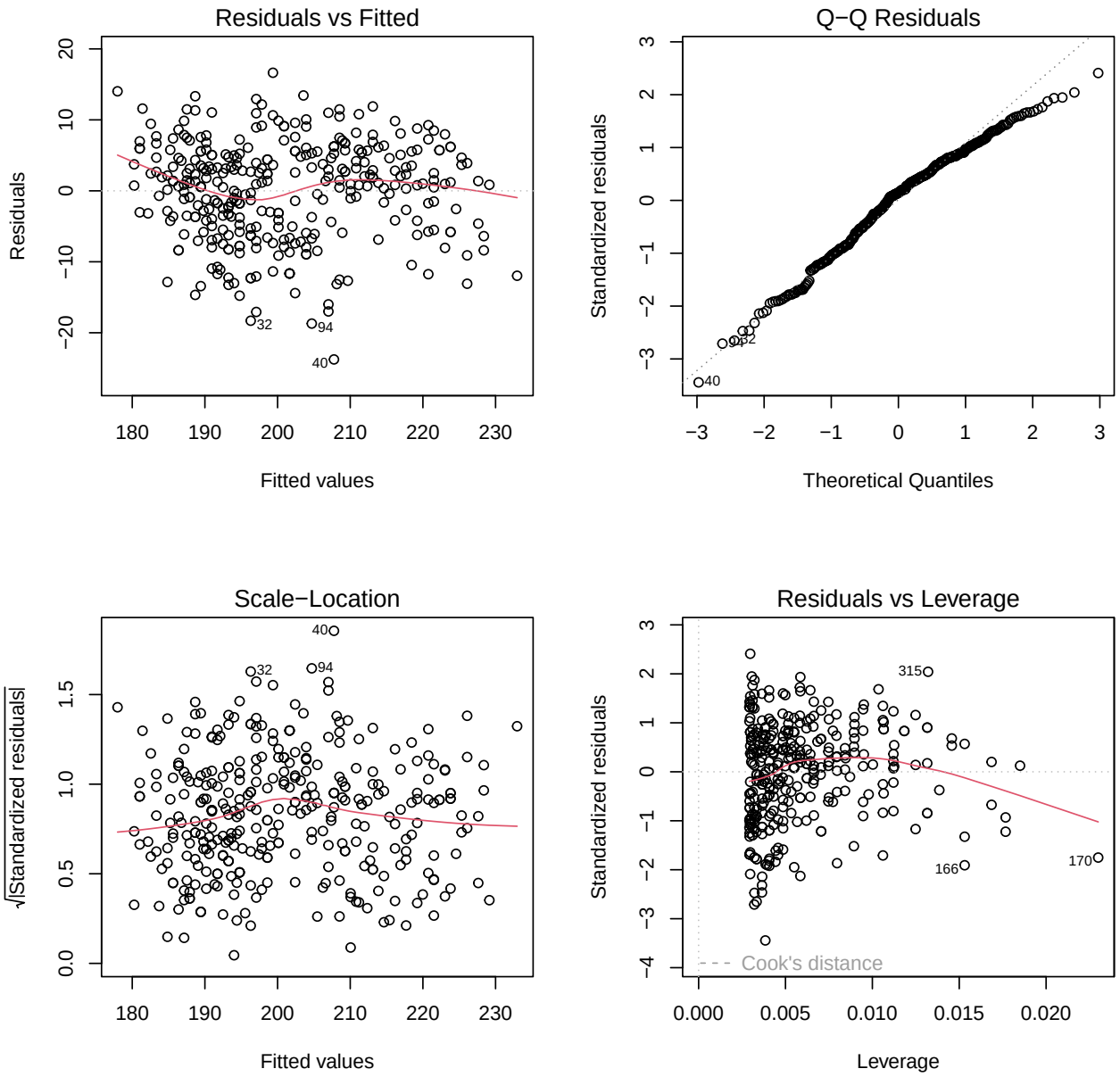
---

## regression

The final stat we will discuss is a linear Regression or a linear model, lm(). This requires two normally distributed numeric variables and you are examining the effect that one variable has on the other variable. There is the implication of causation with this data so make sure you know which you are predicting will affect the other.

For this we are going to examine if flipper length is effected by body size. But first we will make the model, then check for normality and homoscedasticity of the residuals visually like we did with the anova.

```r
penguins_lm <- lm(flipper_length_mm ~ body_mass_g, data = penguins)


par(mfrow = c(2,2))
plot(penguins_lm)
```




Visually this is meeting the assumptions of this linear regression. Now we can call the summary of the model object.

```
summary(penguins_lm)
```

```
##
## Call:
## lm(formula = flipper_length_mm ~ body_mass_g, data = penguins)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.7626  -4.9138   0.9891   5.1166  16.6392
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

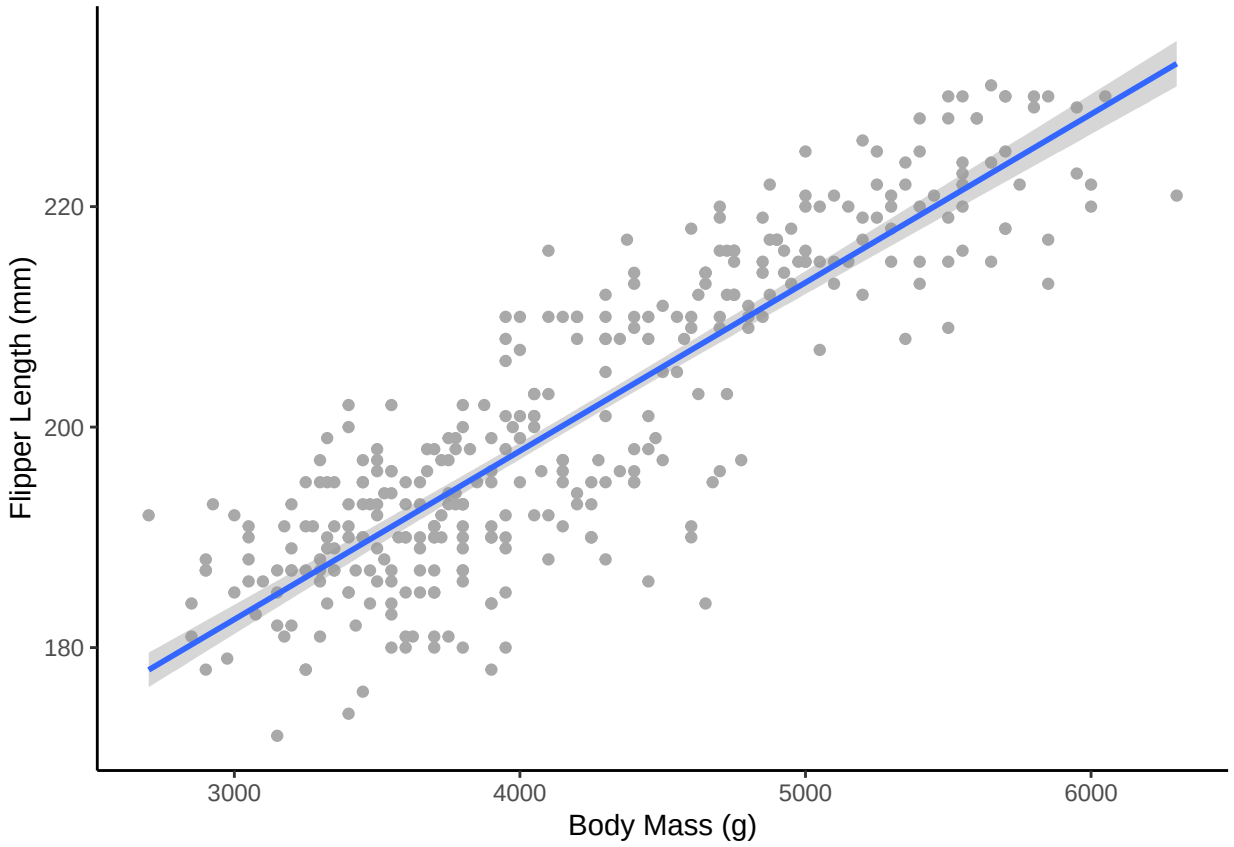
```
## (Intercept) 1.367e+02  1.997e+00   68.47   <2e-16 ***
## body_mass_g 1.528e-02  4.668e-04   32.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.913 on 340 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.759,  Adjusted R-squared:  0.7583
## F-statistic:  1071 on 1 and 340 DF,  p-value: < 2.2e-16
```

This output is showing us that we have a significant effect of body mass on the flipper length of penguins ($p < 0.05$).

And if we plot this, or look at the sign of the Estimate value for body_mass_g, there is a positive relationship between the two. So as body size increases in penguins, so does their flipper length.

Another thing to look at with this output is the Adjusted R-squared value. This is telling us how much of the variation in our dependent variable (flipper length) is explained by our independent variable (body mass). For this model it is ~76% which for ecological/environmental data, is really good.

```
ggplot(penguins, aes(x = body_mass_g, y = flipper_length_mm)) +
  geom_point(color = "darkgray") +
  geom_smooth(method = "lm") +
  theme_classic() +
  labs(x = "Body Mass (g)",
       y = "Flipper Length (mm)")
```



## Now What?

Obviously this isn't a comprehensive tutorial on everything R and by now, maybe you are realizing just how much there is that you can learn about stats and coding in R. To give you some direction with how to continue to learn on your own I have complied some resources for you.

- If you need help troubleshooting code, Stack Overflow is a great place to go to for worked through solutions (https://stackoverflow.com/)
- https://www.datacamp.com/
- https://r4ds.had.co.nz/index.html
- https://ourcodingclub.github.io/
- https://r-graph-gallery.com/
- https://posit.co/resources/cheatsheets/
- https://www.quantitative-biology.ca/git-and-github.html