Basic R for NUIG MSc Workshop

Augustus Pendleton

Nov. 19 and Dec. 3, 2021

Introduction

Hello! This document will serve as an introduction to working with data in RStudio. R is a powerful tool to manage, analyze, and plot your data. Before moving further, spend some time familiarizing yourself with the layout of RStudio. A benefit of R is that a wide online community exists where you can find answers to questions, YouTube videos walking you through concepts, and sample code with which to work. Let's begin with the basics of typing in R

Writing Code in R

You can write code in two places in RStudio, the Console (bottom-left) or the text-editor (top-left). I highly recommend you type your code in the text-editor, rather than the console, as these scripts can be saved, edited, and copy/pasted. When we type code into the text-editor, each line is a separate line of code. When you want this code to run, press ctrl+enter. The code will automatically be run in the console, where the output will be displayed. The most basic code in R is mathematical functions:

```
2+2
```

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

```
2*2
```

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

```
2^2
```

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

```
#For more advanced operations, we use functions
sqrt(4)
```

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

```
pi #Pi is also stored as a constant that can be called by typing pi
```

```
## [1] 3.141593
```

Notice how when I type # before a statement, it isn't counted a code, regardless of where I place it. # signifies a "comment", and is useful to label your code so you understand it later. Also, R doesn't care about spaces, so include them if you want to make your code easier to read, but don't include them in object names or in data.

Objects vs. Text

It's important to understand how R understands the code we write. Numbers are easy; 1 always means 1, and 2 always means 2. However, in most of what you'll be doing, you'll assign names to your numbers, your data frames, or your statistical outputs. We call these objects. Once you create an object, typing that object's name is the same as typing the values it represents. R assumes everything you type is an object (or a function), unless you put " " around it. See the code below for an example:

```
Example1<-10 #Here, I assign the value ten to an object named Example1.
#You can use either the = or the <-, but I prefer the arrow notation.
#When I type Example1, now:
Example1
```

```
## [1] 10
```

```
#I get the output I wanted! Note R IS case-sensitive:
example1
```

```
## Error in eval(expr, envir, enclos): object 'example1' not found
```

```
#Doesn't work. Finally, see what happens when I include quotation marks
"Example1"
```

```
## [1] "Example1"
```

Classes

You can see that Example1 and "Example1" returned different values in R. That is because R classifies objects, and then treats them differently depending on their class. Classes can get complex as you use R more, but most of your data (and objects) will fall into a few major classes:

1. Numeric. These are numbers, and they can be either "double", where they might have decimal points, or "integer", where they are whole numbers
2. Character. These are words or numbers that don't have a numerical value. They are enclosed in " " or ' ', and can also be called strings
3. Factor. These are words or numbers that don't have a numerical value, but they are part of defined set of levels. Imagine a survey response where participants picked their political party from a pre-defined list.
4. Boolean (also called "logical"). These are either TRUE or FALSE.
5. Finally, when an object has pieces of it "missing" (not zero), these are called NA. NA isn't a class of data, but a fill-in which tells R there is no data there.

Vectors and Dataframes

For our purposes, R understands data primarily as vectors, data frames, and lists, which are ordered strings of numbers or names. The command to make a vector is "c()". An important note is that vectors can only contain objects of the same class! Here's an example code where I make a few vectors and then combine them into a data frame. Finally, lists are a way to store any combination objects, regardless of class.

```r
name<-c("Augustus","Raymond","Pendleton", "Gus") #Here I make a vector with
#four items, separated by commas.
type<-c("First","Middle","Last","First")
length<-c(8,7,9,3)
#Now when I call these vectors:
name
```

```
## [1] "Augustus"  "Raymond"   "Pendleton" "Gus"
```

```r
type
```

```
## [1] "First"  "Middle" "Last"   "First"
```

```r
length
```

```
## [1] 8 7 9 3
```

```r
#I'm also going to combine them into a dataframe:
example.df<-data.frame(name,type,length)
example.df
```

```
##        name   type length
## 1  Augustus  First      8
## 2   Raymond Middle      7
## 3 Pendleton   Last      9
## 4       Gus  First      3
```

```r
#If I make a vector with mixed classes, they will all be coerced into characters
mix<-c("8",8,"Eight",TRUE)
mix
```

```
## [1] "8"     "8"     "Eight" "TRUE"
```

```r
#This isn't true for lists though!
mixed_list<-list("8",8,"Eight",TRUE)
mixed_list
```

```
## [[1]]
## [1] "8"
##
## [[2]]
## [1] 8
##
## [[3]]
## [1] "Eight"
##
## [[4]]
## [1] TRUE
```

Now's a good time to take a look over at the "Environment" pane. You can see the objects we made are there, and you can read more into the data frame you made! Note that some variables are "factors", meaning categorical variables, while "length" is numerical. You can also see how R understands a data frame as a collection of vectors, where each column is a variable (like type of name), and each row is an observation. If you read across a row then, you see all the attributes of an observation. For instance, row one is the name "Augustus", which is a "First" name, and has a length of 8 letters.

Below I'm including a few other commands that will be essential for finding data within a data frame. Note the use of the "$" symbol and the [ ] symbols

```
example.df[1,3] #Retrieving the value in row 1, column 3
```

```
## [1] 8
```

```
example.df[1,] #Retrieving row 1
```

```
##       name  type length
## 1 Augustus First      8
```

```
example.df[,1] #Retrieving column 1
```

```
## [1] "Augustus"  "Raymond"   "Pendleton" "Gus"
```

```
example.df[1] #This also retrieves column 1, by default
```

```
##        name
## 1  Augustus
## 2   Raymond
## 3 Pendleton
## 4       Gus
```

```
example.df$type #Retrieving the type column
```

```
## [1] "First"  "Middle" "Last"   "First"
```

Basic functions with dataframes

Most of what you'll do in R will rely on functions, which are previously made tools to help you *do things.* In the code below, I am going to make a new vector, attach it to my data frame using **cbind()**, delete that column, and then introduce some new functions. Note, you can also add new rows to your data frame, but this is a bit more complicated, because the headers and classes have to match! . Whenever you add new rows or new columns, they need to be the same length as the dataframe you're attaching them to. I will also show a method to select specific rows based on the attribute of a specific column

```
newcol<-c("bad","bad","good","best") #Make a new vector
example.df<-cbind(example.df,newcol) #Bind a new column onto our dataframe
example.df
```

```
##        name   type length newcol
## 1  Augustus  First      8    bad
## 2   Raymond Middle      7    bad
## 3 Pendleton   Last      9   good
## 4       Gus  First      3   best
```

```
example.df<-example.df[,-4] #Delete the fourth column (notice the negative!)
example.df
```

```
##        name   type length
## 1  Augustus  First      8
## 2   Raymond Middle      7
## 3 Pendleton   Last      9
## 4       Gus  First      3
```

```
newrow<-data.frame(name="Gustav",type="First",length=6) #Making a new dataframe, to add a
#new row of data
example.df<-rbind(example.df,newrow) #Binding that row onto our existing dataframe
example.df
```

```
##        name   type length
## 1  Augustus  First      8
## 2   Raymond Middle      7
## 3 Pendleton   Last      9
## 4       Gus  First      3
## 5    Gustav  First      6
```

```
sum(example.df$length) #Summing the column "length"
```

```
## [1] 33
```

```
mean(example.df$length[example.df$type=="First"]) #Find the average length of first names
```

```
## [1] 5.666667
```

```
#Find average length of all names that AREN'T last names (note the ! symbol)
mean(example.df$length[!example.df$type=="Last"])
```

```
## [1] 6
```

I threw a lot at you in that block! A big piece was learning how to select/subset data using *operators*. Some key operators are: 1. “==” which means “is equal to”

2. “!=” which means “is not equal to”
3. “>” or “<” are “is greater than” or “is less than”
4. “>=” or “<=” are “is greater than or equal to” or “is less than or equal to”

When you pass code to an operator, it returns a Boolean vector. That sentence sounds WILD, so let’s work through this in an example:

```
example.df
```

```
##        name   type length
## 1  Augustus  First      8
## 2   Raymond Middle      7
## 3 Pendleton   Last      9
## 4       Gus  First      3
## 5    Gustav  First      6
```

```
#First, we select the length column (remember, a column is just a vector!) with the $ notation
example.df$length
```

```
## [1] 8 7 9 3 6
```

```
#Now let's apply an operator
example.df$length==7
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

```
#See how we got a Boolean (logical) vector as a response?
#We can also use operators on characters
example.df$type=="First"
```

```
## [1]  TRUE FALSE FALSE  TRUE  TRUE
```

```
#Boolean operators can be used to subset data
first.names<-example.df$type=="First"
first.names
```

```
## [1]  TRUE FALSE FALSE  TRUE  TRUE
```

```
example.df[first.names,]
```

```
##       name  type length
## 1 Augustus First      8
## 4      Gus First      3
## 5   Gustav First      6
```

```
#Put it all together:
example.df[example.df$type=="First",]
```

```
##       name  type length
## 1 Augustus First      8
## 4      Gus First      3
## 5   Gustav First      6
```

```
example.df[example.df$type!="First",]
```

```
##        name   type length
## 2   Raymond Middle      7
## 3 Pendleton   Last      9
```

Note you learned a new way to locate data based on specific attributes! You also learned that ! means “not.” The x==“ ” notation is very helpful and is worth practicing.

Lastly, let’s quickly learn how to save R Scripts. You should save the R-script you’ve made by hitting “ctrl+S” or clicked the save icon alone the top. Save it wherever you want to, with a .R file type. Go ahead and save this script.

Great job! I know this was really intimidating, but you have the basic skills necessary to write code in R! Moving forward, we’ll mostly focus on learning new functions to perform statistical analyses.

Importing Data into R

Creating our own data set was fun, but it’s much more likely that you’ll be working with data sets you’ve compiled in Excel. Excel is nice because it allows us to easily see and format our data. Once we feel good about our data in Excel, we can import it into R. You’re gonna need to download a sample datasheet I made by navigating to clicking this.[1] Download the Excel Document and open it. Spend some time familiarizing yourself with the variables. First thing we’ll do is clean-up some of the variables so that R can understand it. As you get better at R, you’ll definitely prefer to do this inside R. When you’re starting out, though, and if you have a small dataset, it can be a lot easier to just fix some stuff in Excel before importing it into R:

1. Make sure none of of the variable names (headers) have spaces or special characters. It is best to replace spaces with underscores or periods
2. Base R cannot read read .xlsx files. There are two approaches to getting around this. You can always create a .csv file; this will take up less space on your computer, and if your dataset is very large, it will read into R more quickly. The other option is to install the “readxl” package and use it to import .xlsx files. Because we haven’t gotten to installing packages yet, let’s just make a new file that is a .csv

Use Save As in Excel to save your .csv file somewhere you will be able to find it again. If you’re in the GIS lab, save it into the Documents Folder. While you’re at it, make sure our Excel file is also in this folder. If you’re working from home, save it to the Desktop. Finally, if you don’t have Excel on your home computer, you can just download the .csv file here.

We’re ready to import our data into R!

First, a brief foray into working directories. You need to tell R where on your computer it can search to find files that you want it to import. In the code below, we’ll find out working directory and then set it to a more specific directory, my documents. Note that this will be dependent on your computer, and Mac and Windows have different syntax, so do Google it when necessary. This is especially true if your Desktop folder is backed up in Cloud storage such as OneDrive.

```
getwd() #Find our working directory. Note that this is a function without any arguments
```

```
## [1] "C:/Users/guspe/OneDrive - Georgetown University/Desktop/R_Class"
```

[1] Thanks to Minhas Kamal’s DownGit tool for facilitating the download.

```
#If you're in the GIS lab
setwd("U:/My Documents") #Set our working directory to My documents in the U drive
#If you're on a Mac, your Desktop directory will look something like this:
setwd("/Users/<your username>/Desktop")
#If you're on Windows, your Desktop directory will look something like this:
setwd("C:/Users/<your username>/Desktop")
#If these aren't working or your don't know your username, open up either Finder (Mac) or
#File Explorer (Windows) and go to your Desktop folder. In Finder, the filepath will be
#listed on the bottom of the window pane, in Windows it will be at the top.
```

We'll pause briefly, to learn how to export data. It's good to export data **after** you've set your working directory, so you know where it'll be saved. To export data frames, we will use the "write.csv" function. Here's how we'll do it:

```
write.csv(example.df,file="Example.df.csv")
```

Feel free to go on your computer and confirm that a new .csv file was made. Now let's learn how to IMPORT data.

Now that our working directory is the same as the directory in which our practice data is stored, we are **finally** ready to import our data! We'll do it in two different ways

```
#This is a great way to import data if working directories are still a little intimidating.
#The downside is you have to do it manually every time!
water<-read.csv(file.choose())
```

```
water<-read.csv("Practice_R_Data.csv") #This method also works
head(water) #This shows the first few rows of our dataframe
```

```
##   X     WatershedName  IBI IBI_Rating Invertebra Invert_Rating   WQI WQI_Rating
## 1 1 Bowmanville Creek 86.8  Very Good      52.70          Fair 76.30       Good
## 2 2 Bowmanville Creek 34.1       Poor      51.40          Fair 74.50       Good
## 3 3 Bowmanville Creek 89.3  Very Good      47.70          Fair 83.93       Good
## 4 4 Bowmanville Creek 36.6       Poor      46.40          Fair 81.95       Good
## 5 5     Corbett Creek 50.0       Fair      36.70          Poor 31.00       Poor
## 6 6     Corbett Creek  2.0  Very Poor      34.74          Poor 18.70  Very Poor
##   Year Zone Temp Nutrients Urban Pollution.Source
## 1 2017 East   60  7.259477   YES              YES
## 2 2017 East   40  7.169379    NO               NO
## 3 2017 East   60  6.906519   YES              YES
## 4 2017 East   40  6.811755   YES              YES
## 5 2017 West   40  6.058052    NO              YES
## 6 2017 West   50  5.894065   YES              YES
```

Packages - what makes R great!

A major benefit of R is the large online community which uses and supports it. This is why you can always Google questions about R, and often find very helpful code to use on your own data. The other amazing thing is that many people have probably dealt with similar data, tests, or problems as you have, and come up with their own solutions or functions to accomplish these tasks. Anyone can store these functions in an openly available, online "package".

An R package is a set of functions that someone has created and stored online that is freely available for you to use. A great example is the readxl package. This package contains functions which allow R to read Excel files directly. Lots of your research will require you to download and use packages, so it's a good skill to learn.

```
#You use the function install.packages() to install packages
install.package("readxl")
#Don't worry about all the red text - as long as it doesn't say error, this is good!
```

```
#To load the functions in a package, we use the library() function
library(readxl)
read_excel(path = "Practice_R_Data.xlsx")
#If you don't want to load the entire package, or want to be very specific that you're using a specific
readxl::read_excel(path = "Practice_R_Data.xlsx")
```

If you want to see what other packages you have installed, you can click on the "Packages" tab lower-right pane.We'll talk about other great packages to use later on in the class.

Working with our data

Bam! We've got data in R now![2] Look over onto the environment tab, and you can see that R has a new object called "water." This is our data frame. You can see we have 104 observations (rows) of 11 variables (columns). R is so smart is already understood that the first row of our data was headers. Click the arrow next to water to see what variables are in our data frame.

You can see the variables names: their type, and their values. For instance. WatershedName is a "Factor" (categorical variable) with 11 different "levels", which are the unique names like "Black Creek." IBI is a "num" or numerical variable. Year is an integer (no decimals), but for our purposes we will want it to be categorical, or "factor." Here's the code to do it:

```
water$Year<-as.factor(water$Year)
```

Sorted! You can see it changed in the environment pane. One thing to keep in mind in R: there is no "ctrl z." So when you make changes to a data frame or a variable, those are permanent. Often, however, you can reverse them with code, and our original .csv file remains the same though, so don't be too scared! And you can always save something under a new name, so that you're not writing over any piece of your dataframe.

Basic summary statistics with our dataframe.

Let's explore our data a little bit further. Below I'll demonstrate code to find some basic summary statistics

```
sum(water$IBI) #Sum the column IBI
```

```
## [1] 2712.279
```

[2] Data was collected from the Central Lake Ontario Conservation open-source data port at http://open-data.cloca.com/datasets/cloca-aquatic-monitoring-stations. **THIS DATA WAS EDITED, CHANGED, AND DUPLICATED BY THE AUTHOR, AND SHOULD NOT BE CONSIDERED REPRESENTATIVE OF THE ORIGINAL DATASET. THE RESULTING DATASET IS FOR TEACHING PURPOSES ONLY AND SHOULD NOT BE ANALYZED, DISSEMINATED, NOR PUBLISHED**

```
mean(water$IBI) #Find the average of the column IBI
```

```
## [1] 26.07961
```

```
median(water$IBI) #Find the median of IBI
```

```
## [1] 13.81336
```

```
sd(water$IBI) #Find standard deviation of IBI
```

```
## [1] 28.24865
```

```
cor(water$IBI,water$WQI) #Find correlation between two variables, IBI and WQI
```

```
## [1] 0.5599344
```

Let’s get a little fancier, using our operators. . .

```
mean2017<-mean(water$IBI[water$Year=="2017"]) #Find average IBI in rows from Year 2017
mean2018<-mean(water$IBI[water$Year=="2018"]) #Do the same for 2018 rows
meanIBI<-data.frame(mean2017,mean2018) #Put them together in a dataframe
meanIBI
```

```
##   mean2017 mean2018
## 1 23.66111 27.35999
```

Do you notice how I’m constantly typing “water$” and then a variable? That’s because I always have to tell R to look for those variables in the data frame *water*. If you’re only working with one data frame in your script, you can get around that using the attach() function

```
sum(IBI) #Doesn't work
```

```
## Error in eval(expr, envir, enclos): object ’IBI’ not found
```

```
attach(water)
sum(IBI) #Works!
```

```
## [1] 2712.279
```

However, attach can be risky, and it’s easy to get mixed up as to where R is looking for a variable name. I generally prefer not to use it. You’ll see a lot of the $ operator for the rest of class. This can seem annoying, but it is well worth the specificity. You can also get around many of these difficulties using a set of packages called the tidyverse; we’ll try and discuss them at the end of class. For now, I’m going to “detach” the water dataframe so R will stop searching in it.

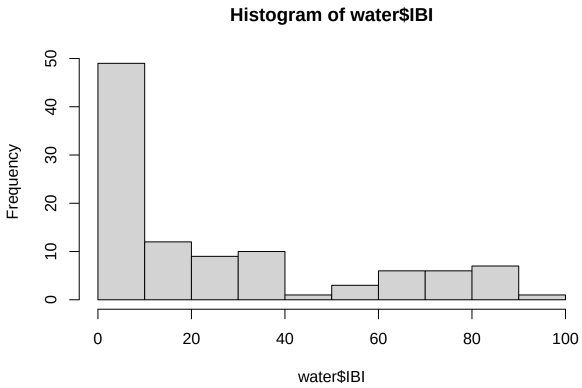
```
detach(water)
```

If you know a command you want to use, but aren’t quite sure how to use it, the help function is great:

```
help(mean)
#You can also use the ? mark
?mean
```

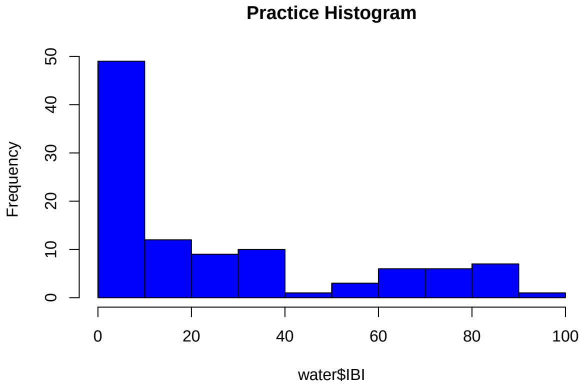
Look in the lower right pane. Here, you can see a help page popped up, explaining the function mean(). It shows what argument you can include in the mean function. So far, we've mostly been including only one argument in each function we use. To show how multiple arguments can be used in a function, let's make a histogram of our IBI data:

```
hist(water$IBI)
```



This is pretty darn ugly! R just used defaults for all aspects of this plot, including axis and main titles, and colors. Let's use multiple arguments in hist() to specify how we want our plot to look. Note that I will separate arguments using commas.

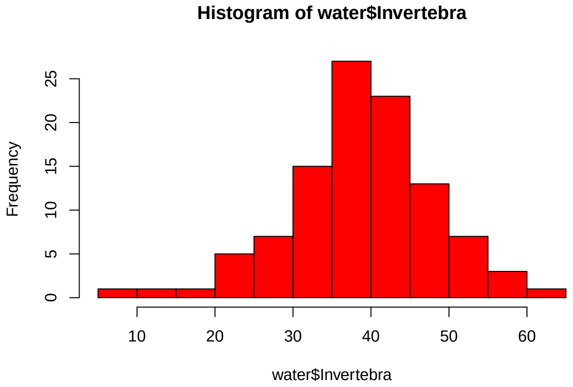
```
hist(water$IBI, col="blue",main="Practice Histogram") #Set color to blue, and give it a new title
```



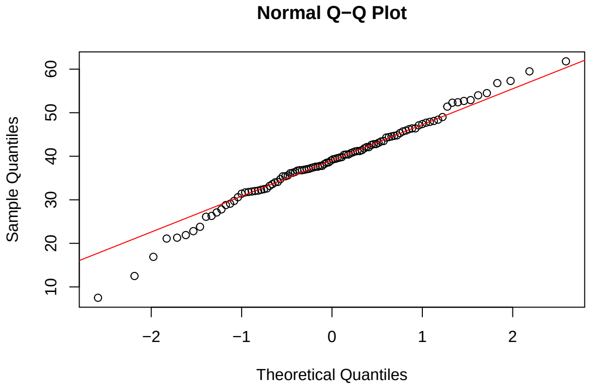
Wow! Look at that beautiful graph. Basic plotting in R is a good way to start analyzing your data, with other notable functions being plot(), boxplot(), or barplot(). If you really want to generate beautiful figures in R, however, I recommend using ggplot2. I'll introduce ggplot2 at the end of this tutorial, so don't worry too much about plotting yet.

However, we can draw some information regarding the IBI variable. Histograms show the distribution of your data, by stacking your values into "bins" of a set width. For instance, a frequency of 50 in the 0-5 bin means that there were 50 data points that fell within that range. From these, we can clearly see the variable IBI is NOT normally distributed, as it isn't shaped like a bell curve. Let's look at a different variable.

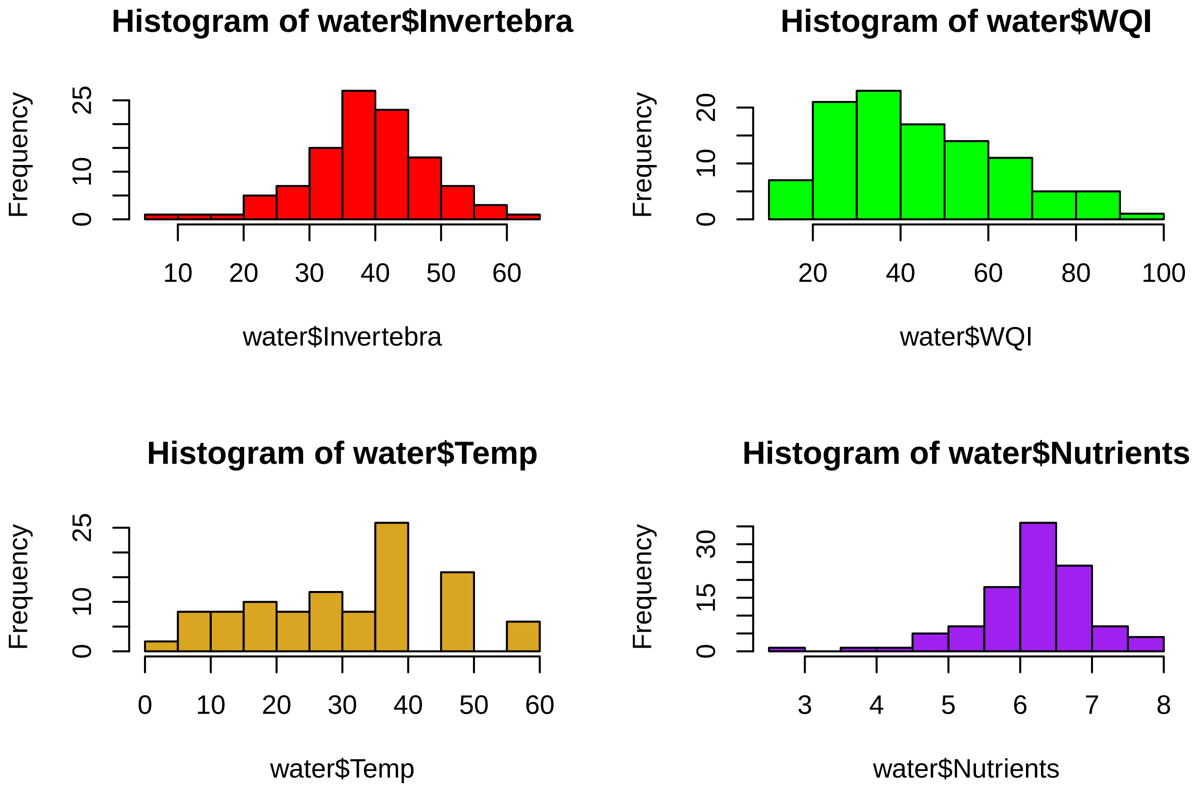
```
hist(water$Invertebra,col="red")
```



```
qqnorm(water$Invertebra) #These two commands are another way to test for normality;
#if the dots follow the line, it's normal. I'm pretty happy with the Q-Q plot
qqline(water$Invertebra, col="red")
```



Clearly, the variable Invertebra is much more normally-distributed. As many of the statistical tests we will run assume normality, it is important to analyze your variables separately with descriptive statistics in this way before proceeding to statistical tests. Analyzing a non-normal variable like IBI would likely require more advanced non-parametric statistics which we won't cover. Our other variables look a little messy but ok (histograms below), so we should be fine to continue with them. Also, as your sample size becomes larger, the assumption of normality becomes less important; since we have >100 observations, we'll be safe to use most parametric statistical tests with all of our variables.



A brief introduction to our data

So we've been exploring this data set for a bit, throwing around terms like IBI and WQI, but what does this mean? This data was gathered by the Central Lake Ontario Conservation Authority and can be accessed here[3]. I then added extra observations and variables like Temp and Nutrients: *this data is made-up, fictitious, not-real, etc.* However, it will be useful practice for data analysis.

CLOCA monitors several watersheds within Ontario and reports their data openly to the public. We are working with a few water quality indicator variables. IBI, or Index of Biological Integrity, is a measure of ecosystem health between 0 and 100. This correspond to the IBI-Rating, which ranges from Very Poor to Very Good. Invertebra is the average number of invertebrate species found at a site in a watershed. WQI is "Water Quality Index" and summarizes a wide variety of variables into a score from 0-100 (100 is good). Watersheds are grouped by geographic zones (Central, East, or West). Temp and Nutrients (I made these numbers up) should be fairly explanatory. I also added a binary variable called "Urban", which states whether the sample site was urban, and a binary variable called "Pollution.Source" which indicates if a point source of pollution was recorded nearby. What can this data set tell us about water quality in Ontario?

A brief introduction to hypothesis testing

All of the statistical tests we will run will have some aspect of hypothesis testing inherent in their nature. I am not a stats professor and so won't delve deeply into the theory of hypothesis testing, but good resources can be found here or in this video.

[3] CLOCA. "CLOCA Aquatic Monitoring Stations Access Data." Central Lake Ontario Conservation Authority, 23 Oct. 2019.

To understand hypothesis testing, we need to recognize the difference between a *sample* and a *population*. For instance, we could define a *population* as all the women in Ireland. And our research questions might be: "What is the average height of the women in Ireland?" If we went and measured the height of each and every woman in Ireland, we could determine this mean exactly. But often that is entirely impossible due to constraints on time and money. Accordingly, we take a *sample*, perhaps 500 women, out of the population. If our sample is *representative* of the population, for instance, the same ratio of age, race, etc in the sample as in the population, it is likely that our sample mean will be a good estimate for the true mean of the population. As such, the sample mean is an estimate of the population parameter: **average women's height in Ireland**. We use values like standard deviation and standard error to estimate how confident we are that the sample mean is a good estimate of the population parameter. Less variance in our sample or larger sample sizes reduce error and make our sample mean more likely to reflect the true mean of the population. A 95% confidence interval is a good example of this, and we can compute it using R code we mostly already know!:

```
#Let's determine the 95% confidence interval for Invertebra
sd.invertebra<-sd(water$Invertebra) #Find standard deviation
sample.size<-length(water$Invertebra) #Find sample size using "length"
SEM<-sd.invertebra/sqrt(sample.size) #Find standard error of the mean
conf.interval95<-SEM*qnorm(0.975) #Find the range of the confidence interval.
#Ignore the qnorm() command for our purposes, as this requires an
#understanding of statistics we won't cover today
upper.limit<-mean(water$Invertebra)+conf.interval95 #Find the top of the confidence
#interval
lower.limit<-mean(water$Invertebra)-conf.interval95 #Find the bottom of the confidence
#interval
Invertebra.conf.interval<-c(lower.limit,upper.limit) #Save them in a vector
Invertebra.conf.interval
```

```
## [1] 36.98653 40.64655
```

We would generally express this confidence interval as (36.99,40.65). We expect the true population mean of Invertebra (the value you would get if you measured every watershed at all times) to fall within this range at a 95% confidence level. Note that this does NOT mean the true mean has a 95% chance of falling within this range. This is confusing but you can read more here

A quick reminder that the stats we cover today encompassed more than two semesters of University for me. . . while I can give you the R code to run hypothesis tests, you will need to research their basis more completely to actually use them.

With that, let's get into our first hypothesis test: the t-test

t-tests

A good introduction to t-tests can be found here. Generally, a t-test will be used to test differences in mean. We will do two types of t-tests today, a one-sample t-test and an independent two-sample t-test. There are several types of t-tests and I highly recommend you ensure you use the correct one for your data - this is an ideal Google situation.

One-sample T-Test

In this test, we will test whether the average Nutrients concentration is greater than a predetermined value. This requires us to have some idea of what the mean should be before we begin our test. For instance, maybe a study in 2014 found the average nutrients concentration was 5.00, and we want to see if that's changed. We can perform a t-test to determine if our data indicates that the population parameter Nutrients is statistically different from 5.00. First, we define our hypotheses:

H_0: $\text{Mean}_{\text{Nutrients}} = 5.00$ THIS IS OUR NULL HYPOTHESIS

H_a: $\text{Mean}_{\text{Nutrients}} \neq 5.00$ THIS IS OUR ALTERNATIVE HYPOTHESIS

Most statisticians will use the Greek letter mu (μ) to denote the mean of a population. I will keep with the word mean for clarity, though. We are going to test the likelihood that we would have observed our sample mean *if* the null hypothesis is true. As such, if we assume the null hypothesis is true, and it turns out that it is *very unlikely (low probability)* that we would have seen our sample mean, we can reject the null hypothesis. We'll discuss this further after we run our test.

Next, we check our assumptions. All statistical tests have assumptions that we must satisfy if we are to consider the test meaningful. For our one-sample t-test, the assumptions are:

1. The variable is continuous (interval or ratio)
2. Observations were independently and randomly collected
3. Variable is normally distributed and lacks outliers OR our sample size is reasonably large (>30)

We know our variable is continuous (it's a number with decimals). We assume that collected one nutrient sample didn't effect the nutrients at another site, so they were independent samples. Plus we have over 100 observations! We previously showed the Nutrients data was reasonably normal and doesn't have outliers. So we're ready to run our test!

```
#Tell R to run a t-test with Nutrients as the variable.
#The mu argument allows us to specify the expected value of
#the null hypothesis
t.test(water$Nutrients, mu=5.00, alternative = "two.sided")
```

```
##
##  One Sample t-test
##
## data:  water$Nutrients
## t = 14.561, df = 103, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 5
## 95 percent confidence interval:
##  6.015934 6.336320
## sample estimates:
## mean of x
##  6.176127
```

Yay! This tells us a lot. It gives us a range of outputs. Don't worry about the t statistic (t=) or the degrees of freedom (df=), we won't cover that. The mean is nice, as well as the confidence interval (this is honestly an easier method to find the confidence interval than what I showed you). Finally, R gives us a p-value. This value is how often we would observe our sample mean (6.18) in a 103 count sample *if the null hypothesis was true*. So it would be **very** unlikely for us to take >100 samples and find a mean of 6.18 if the actual mean was 5.00. As we usually set limits for statistical significance at 0.05 (95% confidence), we easily reject the null hypothesis. We can conclude that at 95% confidence, the nutrient concentration in Ottawa rivers is different than it was in 2014. Two last bits of code for you:

```
#Find p-value assuming our mean is >5.00
t.test(water$Nutrients,mu=5.00, alternative="greater")
```

```
##
##  One Sample t-test
##
## data:  water$Nutrients
## t = 14.561, df = 103, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 5
## 95 percent confidence interval:
##  6.042063      Inf
## sample estimates:
## mean of x
##  6.176127
```

```
#Find p-value assuming our mean <5.00
t.test(water$Nutrients,mu=5.00,alternative="less")
```

```
##
##  One Sample t-test
##
## data:  water$Nutrients
## t = 14.561, df = 103, p-value = 1
## alternative hypothesis: true mean is less than 5
## 95 percent confidence interval:
##      -Inf 6.310192
## sample estimates:
## mean of x
##  6.176127
```

Note that we would also reject the null hypothesis in favor of the alternative hypothesis that the true mean is greater than 5, but would fail to reject the null hypothesis if our alternative hypothesis was that the true mean is less than 5. Usually, doing a two-sided (alternative hypothesis is mean ≠ 5.00) is safer, and you can just look at your data to see which is higher.

Two-sample T-test

Next, let's compare whether the means of two groups are significantly different. For instance, I want to find out if the average Invertebra was different in 2017 than in 2018. Let's write our hypotheses:

H_0: Mean of Invertebra_{2017} = Mean of Invertebra_{2018}

H_a: Mean of Invertebra_{2017} ≠ Mean of Invertebra_{2018}

Let's check our assumptions

1. Samples were independently and randomly collected? Yes
2. Samples are normally distributed? Yes (remember the histograms?)
3. Data is continuous? (Yes)
4. Is our sample size reasonably large (>30)? Yep! We have over thirty observations in each year!
5. Is variance equal between groups? Hmm, we need to check that one

```
sd(water$Invertebra[water$Year=="2017"])
```

```
## [1] 6.213114
```

```
sd(water$Invertebra[water$Year=="2018"])
```

```
## [1] 10.84775
```

Ehh, this is kinda hard to tell. A t-test assuming unequal variance (Welch's t-test) is perfectly valid, and easy to do in R anyways, so I'll show both. Welch's is usually safer and will return a similar result as a "classic"" (equal variance) t-test, so I often just stick with a Welch's t-test, which is the R default. Note as well that the t-test is pretty robust to non-normal distributions as long as our sample size is >30, so these assumptions are guidelines less than hard rules. Always research which test you're doing before you run it to make sure your test is valid!

Let's run our test:

```
#T-test of Invertebra, grouped by Year, assuming equal variance
t.test(Invertebra~Year, data = water, var.equal=TRUE, alternative="two.sided")
```

```
##
##  Two Sample t-test
##
## data:  Invertebra by Year
## t = 1.0684, df = 102, p-value = 0.2878
## alternative hypothesis: true difference in means between group 2017 and group 2018 is not equal to 0
## 95 percent confidence interval:
##  -1.794656  5.985637
## sample estimates:
## mean in group 2017 mean in group 2018
##           40.18667           38.09118
```

```
#T-test not assuming equal variance
t.test(Invertebra~Year, data = water,alternative="two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  Invertebra by Year
## t = 1.2517, df = 101.3, p-value = 0.2136
## alternative hypothesis: true difference in means between group 2017 and group 2018 is not equal to 0
## 95 percent confidence interval:
##  -1.225463  5.416444
## sample estimates:
## mean in group 2017 mean in group 2018
##           40.18667           38.09118
```

As you can see, our results aren't all the different. Let's look at the lower one. We see right away that our p-value, 0.2136, is greater than 0.05. As such, we fail to reject the null hypothesis and did not find that the average Invertebra was different in 2018 than in 2017. This does not PROVE that the average Invertebra was equal each year, just that our data don't show a statistically significant difference. Notice that their alternative hypothesis is "true difference in means is not equal to 0." What this test is actually doing is finding the p-value for whether the $\text{Mean}_{\text{Invertebra2017}}$ - $\text{Mean}_{\text{Invertebra2018}}$ is significantly different from 0. If those two means are equal, obviously they would subtract to zero. The confidence interval R gives, (-1.22,5.42) is the confidence interval of the **difference** between means, not the means themselves. Notice that the 95% confidence interval contains zero, and we failed to reject our null hypothesis at 95% confidence.

That's all for t-tests for now! The other t-test worth researching is a paired t-test, if you have sites or individuals you sampled twice at different times. This t-test can let you get away with smaller sample sizes.

What happens, however, if we want to compare means between more than two groups?

Analysis of Variance (ANOVA)

To understand ANOVA, let's stay with our Invertebra variable. We will call this our dependent, or response, variable. In the t-test above, we tested whether varying an independent variable, Year, affected the response variable Invertebra. Here, we will use Zone as our independent (or predictor) variable. So, does Invertebra vary significantly between the three zones?

We could just run t-tests between each zone. Since we have three zones, we would need to run three t-tests (Central-East, Central-West, East-West). This isn't too bad. However, the more zones we would add, the worse this approach would become, for two reasons. First, it would be inefficient and time-consuming. Second, every time we run a t-test, we risk something called a type-1 error. A type-1 error is where we reject a null hypothesis ($p<0.05$) which was *actually true in the population.* Our probability of having type-1 errors increases the more t-tests we run. For instance, if we had 100 zones and ran t-tests between all of them, very likely we'd find a significant difference between at least two zones, even if that was just by random chance.

This is where ANOVA comes in. Let's look at our hypotheses for ANOVA:

H_0: $\text{Mean}_{\text{Invertebra}}$ is equal for all zones; or, $\text{Mean}_{\text{Central}}=\text{Mean}_{\text{East}}=\text{Mean}_{\text{West}}$

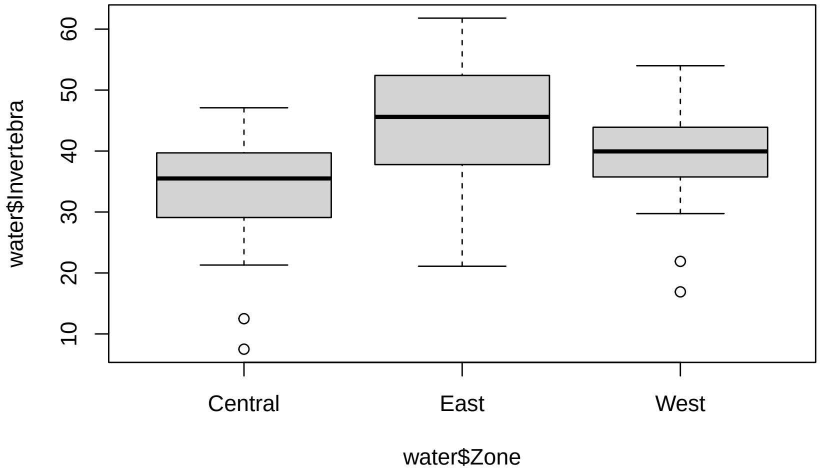
H_a: Means of at least two groups are different; or, not all means are equal

What we test in ANOVA is whether all means are equal between groups. We do this **first** before doing t-tests. **If** we reject the null hypothesis in ANOVA, we find that at least one mean is different from another mean. Then we can do t-tests to find out which groups are different.

There are times when, if you had a large number of groups, you would find t-tests that show significant differences just by chance. The ANOVA test, however, is much less likely to commit a type-1 error. As such, ANOVA safeguards you from committing type-1 errors just due to a large number of t-tests. Let's run through an ANOVA:

Does our data look like there might be differences in the mean? Let's see:

```
boxplot(water$Invertebra~water$Zone)
```



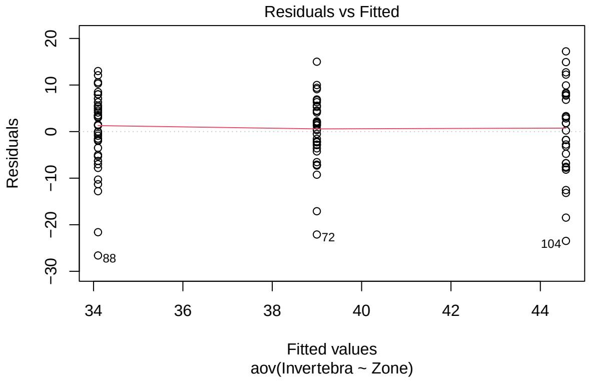
It does look like the East zone might have higher invertebra scores! I think this qualifies well for an ANOVA.

We already stated our hypotheses above. Next let's run through our assumptions:

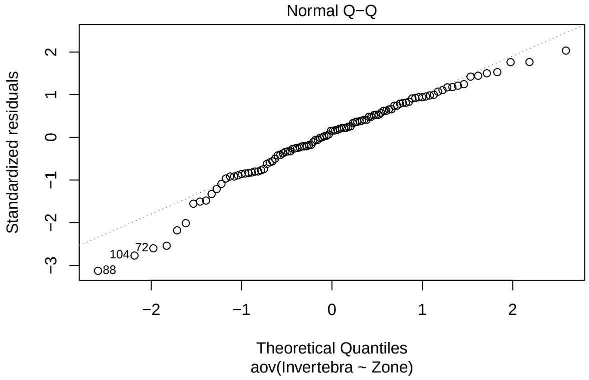
1. Our data is normally distributed or our sample size is large (>30). Yup! All good here!
2. Our samples were independently and randomly sampled. Yep!
3. Equal variances between treatments
4. The residuals in our data are normally distributed

Ok the last two get a little stats-y. I will show the R-code to test for them, but won't go deeper into their meaning

```
aov1<-aov(Invertebra~Zone, data = water)
plot(aov1,1)
```



```
plot(aov1,2)
```



Please don't worry about what these plots are actually showing, but you can get more information here. However, the first plot helps us test assumption #3. If the red-line is flat, that's good and that's all we're going to worry about. The second plot helps us test assumption #4. If the dots fall on the line, that's good. It looks ok, though some points in the lower left are getting out there. Still, we're going to proceed with our test.

Running the ANOVA:

```
#Make an ANOVA model with Invertebra as the response variable
#and Zone as the independent or grouping variable
Invertebra.anova<-aov(Invertebra~Zone, data = water)
summary(Invertebra.anova) #Examine our ANOVA model
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## Zone          2   1841   920.4    12.4 1.53e-05 ***
## Residuals   101   7498    74.2
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
```

What does this output tell us? Look at the Pr(>F), which is our p-value. It is extremely small, so we can reject our null hypothesis. At least one zone has a different mean than another zone. To find out which zones are significantly different, we run a Tukey HSD test:

```
TukeyHSD(Invertebra.anova) #Note the argument for TukeyHSD is the aov object we made before!
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = Invertebra ~ Zone, data = water)
##
## $Zone
##                   diff        lwr        upr     p adj
## East-Central 10.473333   5.467691 15.4789752 0.0000079
## West-Central  4.897778   0.130938  9.6646176 0.0426013
## West-East    -5.575556 -10.642170 -0.5089414 0.0273505
```

What can we learn from this output? The furthest column shows which two zones we are are comparing. The furthest right column (p adj) shows the p-value of the null hypothesis that the mean in those two zones are equal. All of our p-values are less than 0.05! That means that at 95% confidence, we can conclude that all of the zones have different average invertebra counts.

That's as far as we will go with ANOVAs! You can run other ANOVAs with multiple grouping variables (for instance, grouping by both zone and year). More information can be found here

Chi-Squared Tests for Independence

So far we've used hypothesis tests to see if the average of a continuous variable (like Invertebra) is dependent on a categorical variable like zone or year[4]. What happens if we want to compare two categorical variables, though? For this test, we will ask: Do urban areas predict significantly more pollution point sources? We won't have independent and dependent variables; rather, we will simply see if the two variables seem to interact with each other in a non-random way.

A good description on the chi-squared test in R can be found here. Like any hypothesis test, we will first form hypotheses:

H_0: The two variables are independent

H_a: The variables relate/interact/depend on another

Assumptions of Chi-squared:

[4] We treated "Year" as a categorical variable, as we split our data into two groups, 2017 and 2018. If we had multiple years of data, we would treat it as a continuous variable, and analyze its effects using the regression techniques found later in this document

1. Data was randomly/independently collected (Yep)
2. Levels within each category are mutually exclusive and each observation (site) can only occupy one level within each category

We satisfy assumption two, because each sampling site only has one response, and can only be YES or NO in urban and YES or NO in Pollution.Source.

Let's look at our data:

```
table(water$Urban, water$Pollution.Source)
```

```
##
##       NO YES
##   NO  24   6
##   YES  8  66
```

It looks like there are a lot more pollution sources in urban areas. Let's run the test to find out!

```
chisq.test(water$Urban,water$Pollution.Source)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  water$Urban and water$Pollution.Source
## X-squared = 44.778, df = 1, p-value = 2.207e-11
```

If we find our p-value we immediately can see that it is FAR less than 0.05. As such we can reject the null hypothesis and conclude that the variables Urban and Pollution.Source do appear to affect one another. Looking at our table, we can conclude that there are significantly more pollution sources nearby urban sampling sites.

Pearson and Spearman Correlation Coefficients

You might be like, wait, what? We already know how to do correlations with the cor() function. However, it's worth returning to them in this section and applying hypothesis tests to their values.

Pearson's Correlation Coefficient

A correlation coefficient, often denoted r or ρ is a measure of how strongly the change in one variable predicts the change in another variable. Correlation coefficients have values between -1 and 1. An r of -1 means when one variable increases, the other decreases in a perfectly proportional manner. An r of +1 means that as one variable increases, the other increases in a perfectly proportional manner. For instance, the temperature and volume of a gas are *positively* correlated, while the temperature and density of liquid water are *negatively* correlated.

Let's find the correlation between a few variables, IBI, Invertebra, Temp, and Nutrients. Note I can only use **continuous** variables in this command, and R calculates Pearson's correlation coefficient by default:

```
#I use the data.frame() function as cor can only accept multiple variables
#in a dataframe
cor(data.frame(water$IBI,water$Invertebra,water$Temp,water$Nutrients))
```

```
##                    water.IBI water.Invertebra  water.Temp water.Nutrients
## water.IBI         1.00000000        0.5162282 -0.08119929      0.46153050
## water.Invertebra  0.51622823        1.0000000 -0.11451504      0.99096884
## water.Temp       -0.08119929       -0.1145150  1.00000000     -0.09240962
## water.Nutrients   0.46153050        0.9909688 -0.09240962      1.00000000
```

As we move across the first row, we see r values for IBI vs. IBI, IBI vs. Invertebra, IBI vs. Temp, and IBI vs. Nutrients. Some variables seem very highly correlated (Invertebra and Nutrients), some have very weak correlations (IBI and Temp), and others have "medium" correlations (IBI and Nutrients). Most of our correlations are positive.

However, like any sample metric, the correlation shown here is just a picture we gathered using a small sample of a larger population. As such, just because we have a "strong" correlation like Nutrients and Invertebra, is it actually statistically significant? For example, if we only had two data points, we would likely get a strong correlation, but it wouldn't be significant. We can perform a hypothesis test to find out if our correlation coefficient is significant.

```
cor.test(water$Invertebra, water$Nutrients)
```

```
##
##  Pearson's product-moment correlation
##
## data:  water$Invertebra and water$Nutrients
## t = 74.637, df = 102, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.9866892 0.9938768
## sample estimates:
##       cor
## 0.9909688
```

The null hypothesis we are testing is "true correlation is not equal to 0." From the p-value displayed, we reject this null hypothesis and have sufficient statistical evidence to conclude Invertebra and Nutrients are correlated.

In this example, we have only used variables that are continuous and defined as interval or ratio data. This means the distances between levels is set and constant. For instance, the change from 10 degrees Celsius to 20 degrees Celsius is the same change as from 20 degrees Celsius to 30 degrees Celsius. However, there are unfortunately other, messier types of data:

Ordinal Data and Spearman's Rank Order Correlation

A great example of ordinal data you may interact with in your research or Methods class is questions on a survey. For instance, there will be a statement, and then five options from strongly disagree to strongly agree. We can rank this data (strongly agree is "above" strongly disagree), as opposed to purely character data like last names. However, the distance between "strongly disagree" and "disagree" is subjective and different for each respondent. As such, even if we convert the "strongly disagree" to a 1 and the "strongly agree" to a 5, we can't treat this data as a continuous variable. We need different statistical tests to analyze it. The only one we will learn in this lesson is the Spearman's Rank Order Correlation

I want to find out if IBI is correlated with the IBI_Rating (ideally, it should be). I also want to see if the IBI_rating and the WQI_Rating are correlated. To run this test, we first need to put our data in a numerical form and make sure R knows it is ordinal data.

```r
#First, we need to turn IBI_Rating into a factor
water$IBI_Rating<-factor(water$IBI_Rating)
levels(water$IBI_Rating)
```

```
## [1] "Fair"      "Good"      "Poor"      "Very Good" "Very Poor"
```

```r
#You can see R has defined the levels alphabetically, which is incorrect
#We need to change the levels manually
water$IBI_Rating<-ordered(water$IBI_Rating, levels=c("Very Poor","Poor","Fair","Good","Very Good"))
levels(water$IBI_Rating)
```

```
## [1] "Very Poor" "Poor"      "Fair"      "Good"      "Very Good"
```

```r
#Better! Now when we change it to a numerical value, R will
#assign 1 to Very poor, 2 to poor, and so on.
#Make a new variable that is a numerical form of IBI_Rating
water$IBI_Rating_num<-as.numeric(water$IBI_Rating)
head(water$IBI_Rating_num)
```

```
## [1] 5 2 5 2 3 1
```

```r
head(water$IBI_Rating)
```

```
## [1] Very Good Poor      Very Good Poor      Fair      Very Poor
## Levels: Very Poor < Poor < Fair < Good < Very Good
```

Yay! Wasn't that fun? But consider how obnoxious this would be to do by hand in excel! Let's do the same for WQI_Rating:

```r
water$WQI_Rating<-ordered(water$WQI_Rating,levels=c("Very Poor","Poor","Fair","Good","Very Good"))
water$WQI_Rating_num<-as.numeric(water$WQI_Rating)
```

If you look over in the environment tab, you can see we've added two new variables to our data frame! Note you might have to attach() our water data frame again so R finds out those variables are there. And R is very smart and recognizes that those numbers also correspond to ordered levels! Now we are finally ready to run out correlations, which is the easy part!

```r
cor.test(water$IBI,water$IBI_Rating_num,method="spearman")
```

```
## Warning in cor.test.default(water$IBI, water$IBI_Rating_num, method =
## "spearman"): Cannot compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  water$IBI and water$IBI_Rating_num
## S = 21768, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.8838802
```

What does this output tell us? We see a strong, positive correlation between IBI and IBI_Rating, which is good. Also, the p-value is very small, so the correlation is statistically significant. Let's do another one:

```
cor.test(water$IBI_Rating_num,water$WQI_Rating_num,method="spearman")
```

```
## Warning in cor.test.default(water$IBI_Rating_num, water$WQI_Rating_num, : Cannot
## compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  water$IBI_Rating_num and water$WQI_Rating_num
## S = 94106, p-value = 7.493e-08
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.4979942
```

We see a positive correlation of medium strength between IBI_Rating and WQI_Rating, which is statistically significant.

A brief digression about warnings

At this point, you've noticed we're picking up a lot of red warning messages in the console. I get that these can be scary! It's important to recognize that red text can mean two things:

1. Errors. If it says "Error:", that means that R didn't have sufficient information to complete the task you gave it. Essentially, the command failed. Check you syntax, read the help documentation, and google the Error message to troubleshoot.
2. Warning Messages. Commands will often give you "Warning message:", which means that the command was run, but the people who wrote the command want you to know something important. Think of these warnings like reminders; they don't mean you did something wrong, but it's worthwhile checking everything twice to make sure you did it right. Feel free to google warning messages too. This will help you make sure you're using functions correctly.

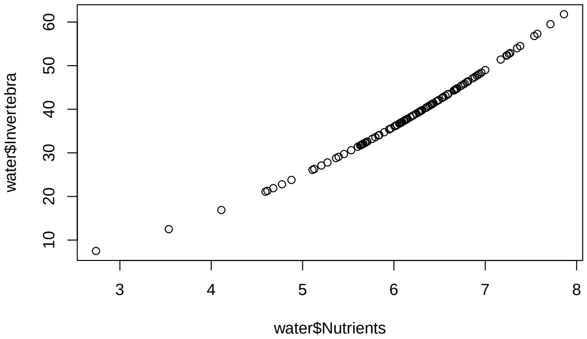
Next, let's say goodbye to our ordinal variables, and go back into continuous variables to cover some basic regression analysis!

Regressions

Here we go folks - this is the last statistical topic we will cover! It's also the most complex. Here, as everywhere else, I recommend you find online video tutorials and descriptions to understand the theory of regressions. For the sake of time, I'm only going to cover the *code* of regression in R, not the theory.

First, we are going to build a regression model between Invertebra and Nutrients. Let's examine the data in a scatter plot:

```
plot(water$Invertebra~water$Nutrients)
```



Recall these variables had a very high correlation before, and we can see how the points seem to fall in a straight-ish line across the graph. This relationship indicates that a linear regression model is appropriate. I'm going to make a linear model using the lm() function. Note how I define Invertebra as the dependent variable and Nutrients as the independent variable.

```
Invert.lm1<-lm(Invertebra~Nutrients, data = water)
summary(Invert.lm1)
```

```
##
## Call:
## lm(formula = Invertebra ~ Nutrients, data = water)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.8729 -0.7542 -0.4631  0.2363  8.0608
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -31.9322     0.9562  -33.39   <2e-16 ***
## Nutrients    11.4552     0.1535   74.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.283 on 102 degrees of freedom
## Multiple R-squared:  0.982,  Adjusted R-squared:  0.9818
## F-statistic:  5571 on 1 and 102 DF,  p-value: < 2.2e-16
```

This output is a lot. Let's ignore the residuals. The coefficients are important, however. We made a linear model, so our end-product will be an equation in the form y=mx+b. Using the output from above, I would write the equation for a best-fit line like this:

$Invertebra = 11.4552(Nutrients) - 31.9322$

Can you find where I got those numbers? It's the "estimate" column. The (intercept) defines what Invertebra would be if nutrients was zero. This value isn't always meaningful in the real world but that's ok. The "estimate" for Nutrients is 11.4552. This means that for every one unit Nutrients increases, Invertebra will increase by 11.4552. This is also known as the slope.

You'll notice R also calculated p-values for our intercept, and nutrients. That is a hypothesis test as to whether those values are significantly different from zero. Because the p-value for the Nutrients coefficient is <0.05, we say there is a statistically significant relationship between Nutrients and Invertebra. There's also a p-value at the bottom of the output. This is a p-value for the entire model as a whole - this becomes more important as you add more independent variables.

Finally, R gives us R-squared values (coefficients of determination). This is the percentage of variance in our response variable (Invertebra) that is explained by variance in our predictor variable (Nutrients). You can think of this as a measure of how closely the data points stick to the regression line we calculated. R-squared values fall between 0 and 1. Values closer to one are grouped more closely to our regression line. The difference between Multiple and Adjusted R-squared values is only pertinent if we add more independent variables. Better linear models have higher r-squared values.

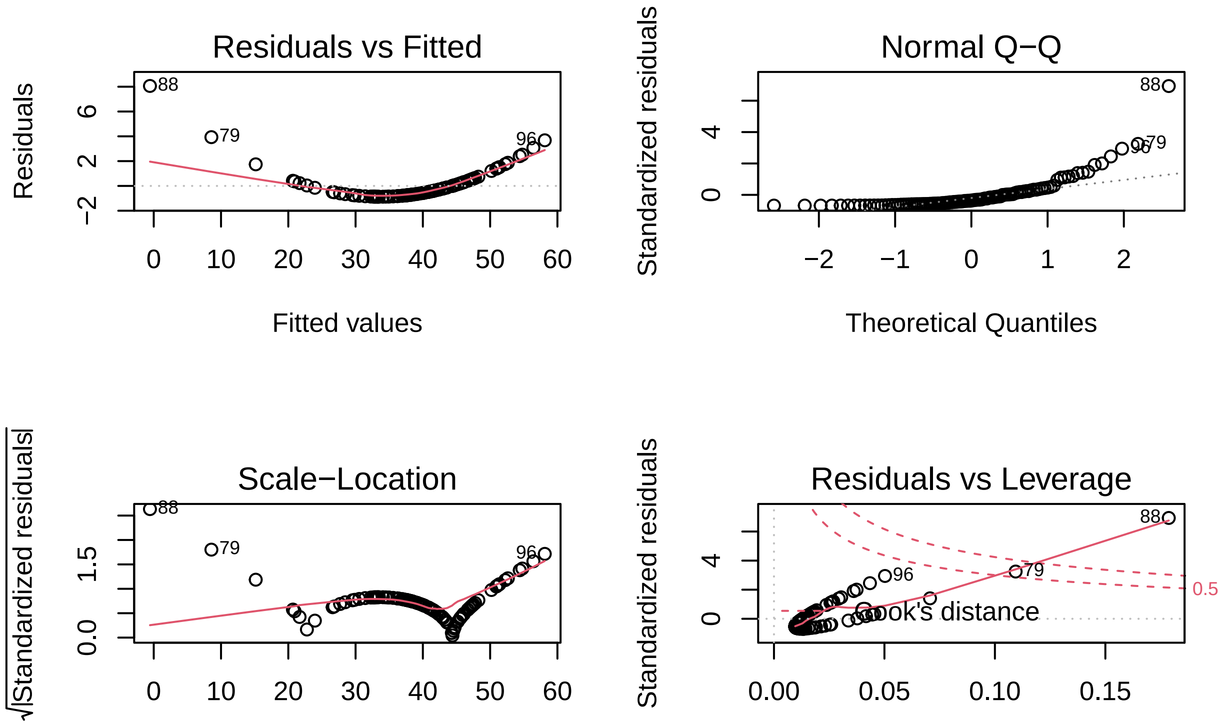
Why use R for regression?

Regression is extremely easy in Excel - you just make a scatter plot and stick a trend-line on it. It'll even give you an R-squared value. So why deal with the pain of R for your regressions?

First, R provides us p-values on our coefficients and linear models, allowing us to estimate whether our model is statistically significant. It also allows us to perform multivariate regressions, where multiple independent variables are used to predict one dependent variable (we'll cover this in a bit). Finally, R allows you to better analyze your linear model.

As with any statistical test, there are assumptions/conditions which need to be met for a regression model to be appropriate. A discussion on these assumptions can be found here. We won't discuss them here, but I can show you R code to produce the diagnostic plots necessary to test these assumptions:

```
par(mfrow=c(2,2))
plot(Invert.lm1)
```



```
par(mfrow=c(1,1))
```

Note that some of our assumptions appear to be violated. Recall the scatter plot between these two variables actually looked like it was curving, so a linear model might not be appropriate. Regardless, this was a good start! Now you're ready to run basic regressions. In the next section, I'm just showing some R code using to produce different linear models and linking sources to understand them

```


Invert.lm2<-lm(Invertebra~Temp, data = water)
summary(Invert.lm2)
```

Statistically Weak Regression

```
##
## Call:
## lm(formula = Invertebra ~ Temp, data = water)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -30.9914  -4.5836   0.3267   5.8179  21.3755
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 41.10085    2.17226  18.921   <2e-16 ***
## Temp        -0.07204    0.06188  -1.164    0.247
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 9.505 on 102 degrees of freedom
## Multiple R-squared:  0.01311,    Adjusted R-squared:  0.003438
## F-statistic: 1.355 on 1 and 102 DF,  p-value: 0.2471
```

This is an example of a bad linear model. Notice the p-values are high and the R-squared values are low.

Multiple Linear Regression   Source for understanding multiple linear regression

Code:

```
lm3<-lm(Invertebra~WQI+Temp, data = water) #Make a linear model to predict Invertebra using
#WQI and Temp as independent variables
summary(lm3)
```

```
##
## Call:
## lm(formula = Invertebra ~ WQI + Temp, data = water)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -25.5614  -3.8273   0.8274   4.4011  19.2114
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 30.35042    2.15282  14.098  < 2e-16 ***
## WQI          0.32324    0.03980   8.122 1.17e-12 ***
## Temp        -0.18038    0.05017  -3.595 0.000503 ***
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 7.429 on 101 degrees of freedom
## Multiple R-squared:  0.403,  Adjusted R-squared:  0.3912 
## F-statistic: 34.09 on 2 and 101 DF,  p-value: 4.852e-12
```

Multiple Linear Regression with Interaction terms Explanantion Here, you’re exploring how Invertebra changes alongside WQI and Temp, adding the possibility that WQI’s impact on Invertebra may be different at different temperatures. Code:

```
lm4<-lm(Invertebra~WQI*Temp, data = water)
summary(lm4)
```

```
##
## Call:
## lm(formula = Invertebra ~ WQI * Temp, data = water)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -25.6862  -3.7658   0.5741   4.6386  19.2177 
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 29.238896   4.327775   6.756 9.53e-10 ***
## WQI          0.348238   0.093296   3.733 0.000315 ***
## Temp        -0.143445   0.134374  -1.068 0.288315    
## WQI:Temp    -0.000788   0.002658  -0.297 0.767457    
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 7.463 on 100 degrees of freedom
## Multiple R-squared:  0.4035, Adjusted R-squared:  0.3857 
## F-statistic: 22.55 on 3 and 100 DF,  p-value: 3.113e-11
```

Multiple Linear Regression with a Categorical Predictor Source That’s right, you can also do regression with categorical variables!

Code:

```
lm5<-lm(Invertebra~Temp+Urban, data = water)
summary(lm5)
```

```
##
## Call:
## lm(formula = Invertebra ~ Temp + Urban, data = water)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.095  -4.673   0.405   5.703  21.623
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 40.85952    2.56855  15.908   <2e-16 ***
## Temp        -0.07271    0.06229  -1.167    0.246
## UrbanYES     0.36911    2.07101   0.178    0.859
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 9.551 on 101 degrees of freedom
## Multiple R-squared:  0.01342,    Adjusted R-squared:  -0.006112
## F-statistic: 0.6871 on 2 and 101 DF,  p-value: 0.5054
```

Non-Linear Data - We can still do regression! What if our data follows a quadratic relationship? We can either linearize our data and put it into our model, or include a new term in our linear model.

Quadratic Models

Linearizing Data

Note you can also linearize most types of data, including exponential, logarithmic, and cubic relationships.

```r
lm.linnearized<-lm(WQI~sqrt(Nutrients), data = water) #Linearized data
summary(lm.linnearized)
```

```
##
## Call:
## lm(formula = WQI ~ sqrt(Nutrients), data = water)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.549 -11.871  -1.016  10.938  38.839
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -98.361     22.778  -4.318 3.65e-05 ***
## sqrt(Nutrients)   57.380      9.166   6.260 9.18e-09 ***
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 16.3 on 102 degrees of freedom
## Multiple R-squared:  0.2776, Adjusted R-squared:  0.2705
## F-statistic: 39.19 on 1 and 102 DF,  p-value: 9.182e-09
```

```r
#Quadratic model
water$Temp2<-water$Temp^2
lm.quadratic<-lm(WQI~Temp+Temp2, data = water)
summary(lm.quadratic)
```

```
##
## Call:
```

```
## lm(formula = WQI ~ Temp + Temp2, data = water)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.654 -13.273  -2.524   7.964  47.550
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.74085    7.23267   5.909 4.67e-08 ***
## Temp        -0.43908    0.49584  -0.886    0.378
## Temp2        0.01223    0.00760   1.609    0.111
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 18.34 on 101 degrees of freedom
## Multiple R-squared:  0.09391,    Adjusted R-squared:  0.07597
## F-statistic: 5.234 on 2 and 101 DF,  p-value: 0.006872
```

We’d write the last equation as: $WQI = 42.741 - 0.439(Temp) + 0.012(Temp^2)$ Keep in mind this isn’t a very good model (p-values of coefficients and r-squared values).

Logarithmic Regression Use a continuous variable to predict a categorical response Urban

Source

```
glm1<-glm(as.factor(Urban)~IBI,family="binomial", data = water)
summary(glm1)
```

```
##
## Call:
## glm(formula = as.factor(Urban) ~ IBI, family = "binomial", data = water)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5985  -1.5688   0.8238   0.8303   0.8319
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 0.8831843  0.2945697   2.998  0.00272 **
## IBI         0.0007584  0.0077369   0.098  0.92191
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 124.96  on 103  degrees of freedom
## Residual deviance: 124.95  on 102  degrees of freedom
## AIC: 128.95
##
## Number of Fisher Scoring iterations: 4
```

Other statistical topics you my want to look into Multiple Logistic Regression

Bootstrapping and other resampling techniques

That's it for stats

That's all the stats which I wanted to give you. Always happy to chat in class or work through code with you!

Bonus Information!

Packages, commands, and sources I have found useful.

Tidyverse

I highly recommend you install the package "tidyverse," which is itself a collection of packages. You may also need to install the package "magrittr." The tidyverse will greatly help you format, analyze, and plot data if you continue to work in R. Remember to install the package (this can take awhile, since it is quite a few packages)

```
install.packages("tidyverse")
```

And remember you need to load the package using library()

```
library(tidyverse)
```

The tidyverse is in fact a collection of several very helpful packages. The main ones you'll be using are dplyr and ggplot2.

dplyr

Dplyr is incredibly helpful for managing your datasets. A great tutorial for dplyr is here. I want to show you a few commands and the pipe operator (%>%). You'll see it a lot in R code online and it is very helpful. A fabulous pipe operator tutorial is available here. Look at the code below to see why we like the pipe operator:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
water%>%
  filter(Zone=="Central")%>%
  select(IBI)%>%
  summarize(Mean = mean(IBI),
            St.Dev = sd(IBI))%>%
  round(2)
```

```
##    Mean St.Dev
## 1 17.35  18.42
```

What the pipe operator does it let's us specify the order of functions in an intuitive way. For instance, the above code can be translated to "Take the data frame water, THEN filter for observations in the central zone, THEN select the IBI column, THEN compile these statistics, THEN round to two decimal places." Ok, maybe this seems confusing. However, without pipe operators, the same function would look like this:

```
round(summarize(water,Mean=mean(water$IBI[Zone=="Central"]),St.Dev=sd(water$IBI[Zone=="Central"])),2)
```

```
##    Mean St.Dev
## 1 17.35  18.42
```

Or like this:

```
water_filt<-filter(water, Zone=="Central")
water_select<-select(water_filt, IBI)
water_summ<-summarize(water_select,
                      Mean = mean(IBI),
                      St.Dev = sd(IBI))
round(water_summ,2)
```

```
##    Mean St.Dev
## 1 17.35  18.42
```

Now you can see why sometimes the pipe operator makes things simpler to read. There are lots of very useful functions in the dplyr package (like for real, I use them every day). Below I list a few:

```
select() #Select a column
filter() #Choose observations based on an attribute (like being in Zone==East). These will use operator
group_by() #Group your data by a variable, like Year
mutate() #Make a new column with a function. For instance:
```

```
water%>%
  select(IBI, WQI)%>%
  mutate(Index_sum = IBI+WQI)%>%
  head
```

```
##    IBI   WQI Index_sum
## 1 86.8 76.30    163.10
## 2 34.1 74.50    108.60
## 3 89.3 83.93    173.23
## 4 36.6 81.95    118.55
## 5 50.0 31.00     81.00
## 6  2.0 18.70     20.70
```

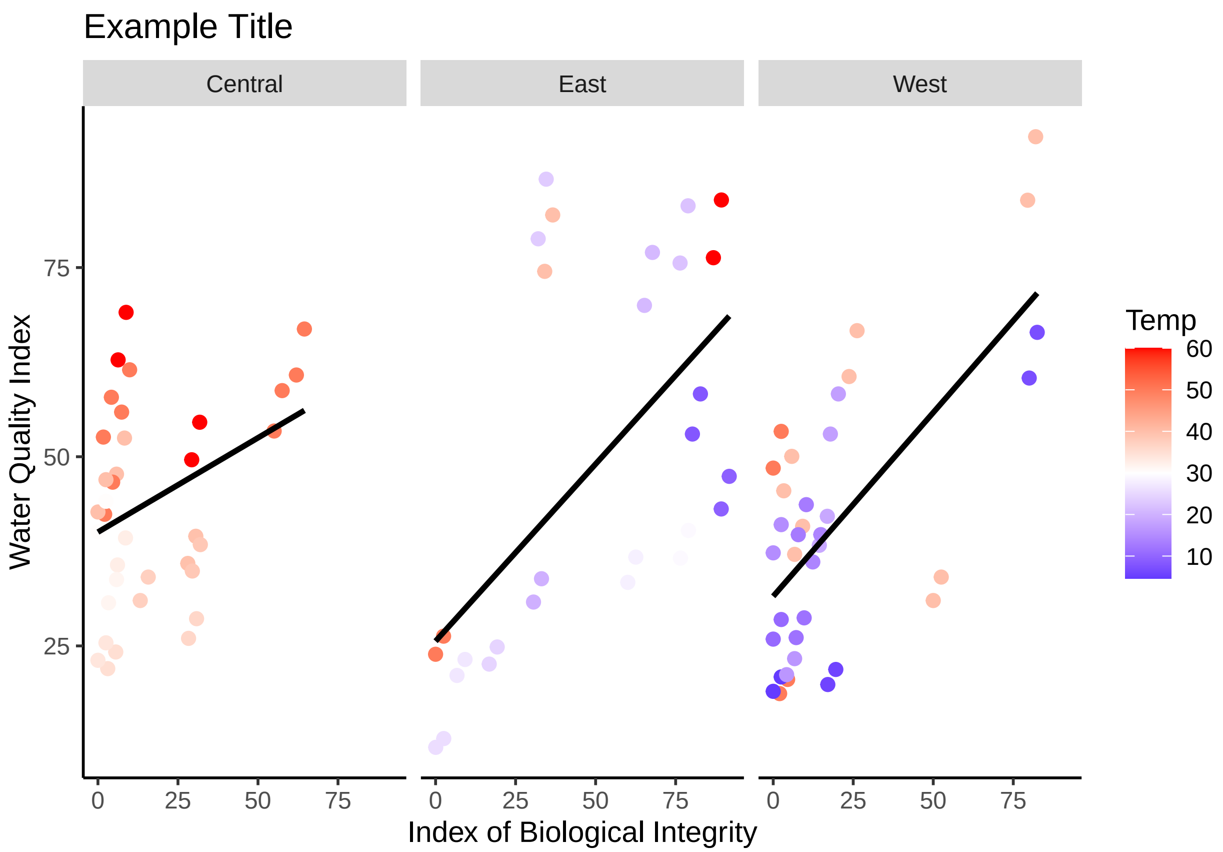
Now we've got a new variable! Dplyr will get you through lots a everyday tasks like cleaning up a data frame, calculating summary statistics, and as you get more advanced, running statistics using grouping variables and list columns.

ggplot2

The other tidy package that I use almost everyday is ggplot2. The package ggplot2 allows you to make very beautiful and customizable graphs. You have complete control over all aspects of your graph which makes it very flexible. It is also easy to graph several variables as once by mapping aesthetic traits like color or size to variable attributes.The syntax is a little confusing at first, so I recommend you start with a step-by-step tutorial that teaches you from the basic to the advanced. I cannot recommend this tutorial enough. It is a great place to learn ggplot2 and teaches you almost everything you need to know to get started plotting. A sample bit of ggplot2 code is shown below:

```r
library(ggplot2)
#Make a new plot using our dataframe water, with IBI on the x-axis and WQI on the y axis
ggplot(data=water,aes(x=IBI,y=WQI))+
  #ggplot uses + to link layers of your graph together
  #geoms tell ggplot what type of graph you want. Here we want a scatterplot (points). We also say that
  geom_point(aes(color=Temp),size=2)+
  #In theme(), we can specify attributes of the plot like the axes, the background, and the fonts
  theme(panel.background = element_rect(fill = "white"),
        axis.line = element_line(color = "black"))+
  #With scale commands, we can specify how aesthetics are mapped. Here, I'm editing the color palette w
  scale_color_gradient2(low = "blue",mid = "white",high="red",midpoint = 30)+
  #Specifying our axis and plot titles
  labs(title="Example Title", x = "Index of Biological Integrity", y = "Water Quality Index")+
  #Adding trendlines
  geom_smooth(method="lm",col="black", se=F)+
#And finally, I decide to facet (or split) my plot into multiple plots by zones
  facet_wrap(~Zone)
```

```
## ‘geom_smooth()‘ using formula ’y ~ x’
```



This is just an introduction to ggplot2 - it's wild what you can do with it!

Other tidyverse packages that will be helpful in certain cases

1. lubridate. Makes working with dates and times (already a bit of a headache) much more manageable
2. stringr. Very helpful functions for working character strings. Helps search for specific words or patterns, split strings, cut out white space, and other useful transformations
3. tidyr. These functions require a little more confidence working with data frames, but pivot_wider(), pivot_longer(), and separate() are functions I use constantly.
4. purrr. As you get more advanced in R, you'll do a lot of work with lists and list columns. purrr provides map() functions which make working across lists and vectors easier, and lets you avoid confusing for() loops. You'll also combine tidyr's nest() with mutate(map()) to powerfully group data frames, run analyses within groups, and then extract relevant outputs.
5. This isn't a package, but dplyr's _join() functions (inner_join(), left_join(), anti_join()) will be essential if you are combining or comparing multiple data frames.

Nonparametric testing

For much of our data, which may be stubbornly non-normal or with smaller sample sizes, we'll need to use non-parametric testing. Here is a good start

R Projects and the "here" package

As your coding projects get bigger, you'll often be working with multiple datasets, analyses, and outputs. It can be very helpful to organize relevant data and R scripts together in a RProject. Coupled with RPojects is the here package. This helps you reference files and directories within a project easily, and makes your code more robust to changes in file paths. I'd go right ahead and make an RProject for your dissertation now, and keep everything R-related in it. It can't hurt!

You're done!

Thank you for sticking with me this long! I hope this tutorial was useful. As you work with your own datasets, feel free to contact me over email or twitter (links on the class website) and I'll be happy to talk over some approaches to analyzing your data.