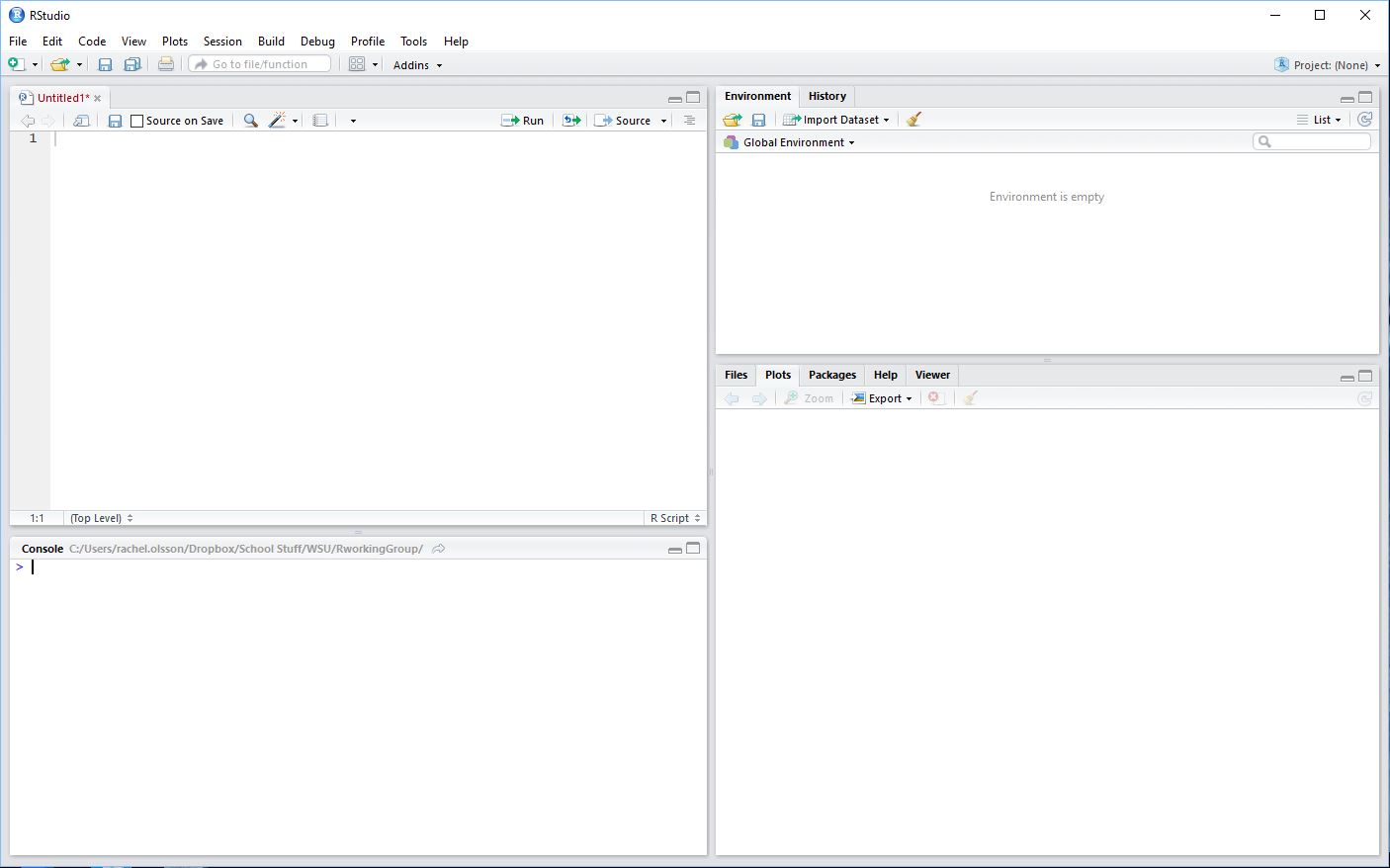
**What is R?**

For this course, we will introduce you to the R statistical software and programming language. R is free, open source, and can be used with Windows, Mac, or UNIX operating systems. R uses a console based, command line interface. R can be downloaded here (I used the Berkeley mirror): <https://www.r-project.org/>. We recommend (and will teach) using a secondary application called RStudio. RStudio is another open source program that makes editing code and visualizing output easier. RStudio can be downloaded here: <https://www.rstudio.com/>. Please download both R and RStudio before moving forward. Once you’ve downloaded both programs, you will only need to open RStudio when starting a new session because RStudio will run R in the background.

This walkthrough is intended to introduce you to some of the very basic functions of R, some of the specific wording related to the program, and how to work with very basic scripts and code. I have highlighted lines that can be copied directly into the R console in yellow.



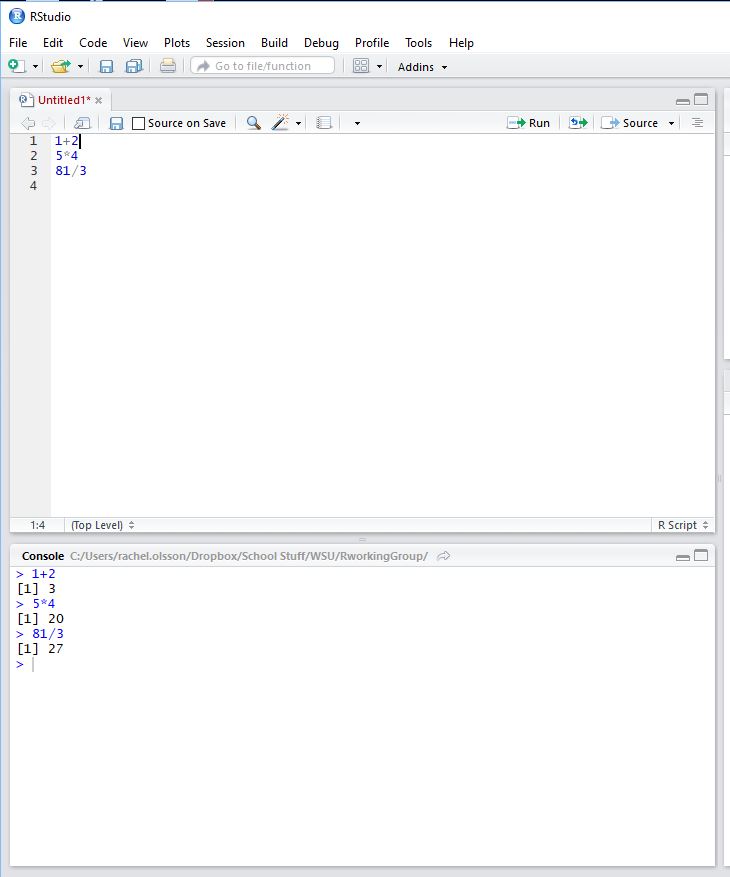
**Scripts and the R Console**

When you open a new session in RStudio, you will likely see something like this. The top left pane is your script editor. The bottom left pane is your console. The top right pane holds the environment and history tabs, and the bottom left pane holds many useful tools to interact with the R program. Any of these panes can be collapsed or expanded as needed.

The script editor is where you can write and save scripts you will use repeatedly. This pane effectively acts as a word processor, but with a bit more function. You can run code from the script editor, but you can also annotate your code. Editing scripts in the script editor makes it much easier to return to your work.

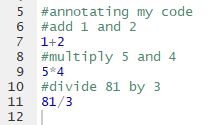
The console is a command line. You can write individual pieces of code here, and run them immediately. Once you run code in the console, you cannot edit the same line of code without rewriting it again. If you are writing very simple code, you can easily write this directly into the console. The console will also show you the output of the code you run.

This shows an example of writing script in the editor, and using the console directly. I wrote these simple equations into the script editor. When you write code in the script editor, pieces of the code will be different colors. For example, text will be black, and numbers will be blue. If you write directly into the console, the all the code will be blue, and the output will be black.



Once you have a complete line of code in the editor, you can run it by selecting the line and clicking the run button. You can also use Ctrl+Enter (or Cmd+Enter for Mac). If you write the code directly into the console, you can run that line by hitting Enter. As shown, R can perform basic calculations without additional packages. If you want to clear the console, click within the pane and use Ctrl+L.

#Annotation is a way to write notes about what your code is doing without affecting the code. Using the # symbol tells the program that anything coming after # for the rest of the line is not part of the code. In RStudio, annotated lines are green. When you run the code, R will skip the green lines. Annotation can be incredibly useful because code isn’t always super clear, and code can get very long. Annotation can help you find errors, or remind you how different pieces of code are affecting your output. We’ll explore this a bit later.



**Working Directory**

Every time you start a new session in R, you’ll need to set your working directory. The working directory is the folder in which you are working during any given session. Any files you want to access during the session will need to be in the working directory. To find out what your working directory is, you can use the code

getwd() #tells you your current working directory

To set your working directory to a new folder, you can use the code

setwd() #note: if you use this, you will need to type in the folder pathway in quotes between the parentheses for the folder you’d like to set as your WD. Example:

setwd(“C:/Users/rachel.olsson/Dropbox/School Stuff/WSU/Crowder Lab/ Rachel PhD Work/ R\_Directory”) #This won’t work for you, because your file pathways will be unique to the computer that you are using.

However, it is much easier to use the user interface for this. In the top bar, drop the Session menu, and choose Set working directory> Choose Directory. This will allow you to navigate through your computer’s files the same way you would if you were opening or saving a new file.

\*\*This is a good time to point out that the R language is sensitive to letter case\*\*

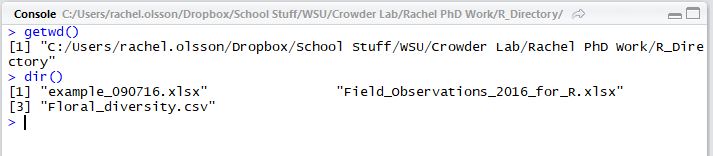
getwd() #will tell you what folder is being used as your working directory.

Getwd() #will give you an error.

You can change your working directory at any time during a session, but you can only be working in one directory at a time.

Once you have set your working directory, you can use the *dir* function to access the files in your directory.

dir() #this will show you the file names of everything in your current working directory folder.



This is showing the file names of three different files in my folder, and assigning a number to each one. I can use the file name or the number to access the files.

**Packages**

Packages are user-created tool sets available for use in R. These packages extend the types of statistical tests you can use, allow you to find different datasets, and use R with other programs. Clicking on the Packages tab in the lower right pane of RStudio will show you a list of packages in the User Library (these are the packages you have downloaded) and the System Library (these are packages that are included with R). These package names have very brief descriptions. You can click their names and read more about the packages (clicking will open the Help tab).

To download and install a new package, you can either click the “Install” button and search for the package name, or use the *install.packages* function.

install.packages(“ggplot2”)

If you want to load a package, you can either click the check box next to the package name, or you can use the *library* function.

library() #you will then include the package name in quotes inside the parentheses. Example: library(“ggplot2”)

**Help!**

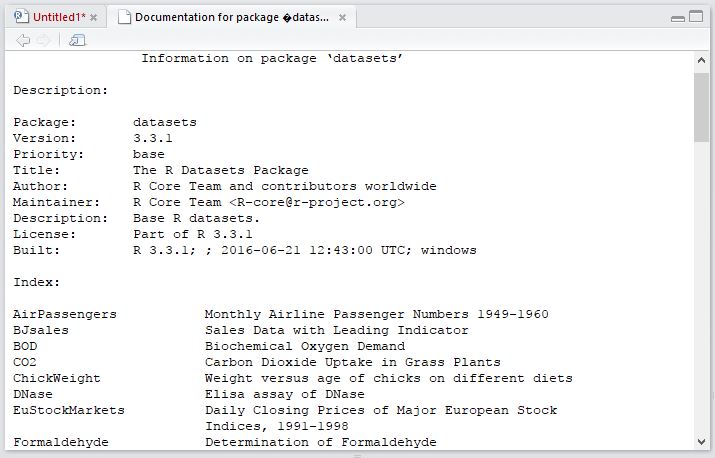
The help tab in the lower right pane of RStudio is the first place to look if you need help. Here you will find information about packages, functions, and examples for using code.

**Datasets in R**

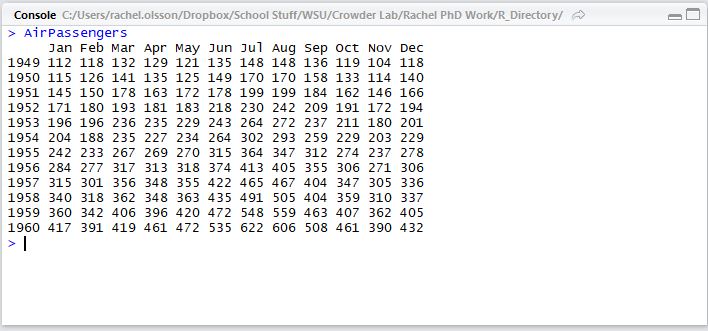
R has several datasets preloaded and accessible to you. To access a list and description of the datasets, use the code

library(help=”datasets”)

This will open the documentation for the datasets in the upper left panel of RStudio.



To view a dataset, simply type the index name into the console and press enter. Remember that this is case sensitive, so typing in AirPassengers will open the dataset, but airpassengers will not.



The descriptions of the datasets will help you understand what you are looking at. The AirPassengers dataset shows the number of airline passengers per month from 1949-1960.

To read more information about a dataset, you can use the *help* function. This will open a help window in the lower right pane. From this you can learn more about the data, its source, and how to use it. There is also example code of how to run certain tests using this dataset.

help(AirPassengers)

You can also use a question mark in place of the word help to access the help function. If you use the question mark, then you do not use parentheses

?AirPassengers

**Data Structures**

There are several different types of data that can be used in R. For example, you can use numeric data, (1, 15, 35, etc), character data (yellow, B, March), date, factor (character data that has a meaning- we’ll get into this in lectures), logical (true/false, 0,1) and some others. To store these data so that we can use them, we use data structures like vectors, data frames, and matrices.

A **vector** is a one dimensional piece of data, separated by commas.

A vector might look like this

(4, 8, 5, 12, 15, 8)

To create this vector, we will need to create an object, and then assign values to that object. For this exercise, we will create an object called a. We will also use the *concatenate* function, which is abbreviated c. Using the = sign tells are that we are going to create an object called a.

a= c(4, 8, 5, 12, 15, 8)

Once you’ve run this line of code, you have a vector called a. You can view this vector by typing a into the command line.

You can use the *structure* function, abbreviated str to view the structure of the object a.

str(a) #this will tell you what type of data is in your vector

You can also use other basic functions to interact with your object.

length(a) #this will tell you how many values are in your vector

is.vector(a) #this is a true/false test on whether the dataset is a vector

is.character(a) #this will tell you if the values in your dataset are character (true/false)

is.numeric(a) #this will tell you if the values in your dataset are numeric (true/false)

mean(a) #this will give you the mean of the values in your object

sum(a) #this will give you the sum of all values in your object

summary(a) #this will summarize your data

A vector can also be made up of character data, for example:

b=c(“red”, “blue”, “yellow”, “yellow”, “green”, “purple”)

Anything that isn’t a numeric value needs to be in quotes for R to recognize it as a character value, including dates.

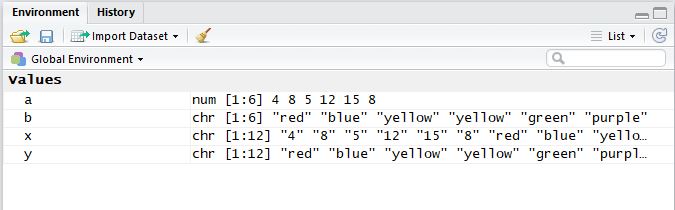
You can try some of the above functions to interact with the b object. Because this object is made up of characters, some of these functions will not interact, because you can’t logically get the mean of a bunch of colors.

You can also combine vectors into a single vector by creating a new object. The order of this depends on the order within parentheses. Note that when combining numeric data with other types of data, the numbers will sort of cease to be numbers, and will become characters. We can go over what this means more in the lab.

x= c(a,b)

y= c(b,a)

Now that you’ve created some objects, you’ll notice the upper right pane, the Global Environment, has been populated with some of these values. This list can be helpful in remembering what objects you’ve created and what type of information is contained in these objects.



Other information will populate in the Global Environment as we move through different uses of R.

**Matrices** are two dimensional pieces of data with rows and columns, just like you would see in a spreadsheet. The rows and columns can both be made up of vectors. Matrices cannot contain more than one type of data, for example, character and numeric. There are several ways to build a matrix. The first is to combine vectors using the *columnbind* (cbind) or *rowbind* (rbind) functions. You’ll create a new object that combines the previous objects, or vectors.

data1= cbind(a,b)

You’ll notice that now the Global Environment has been populated with data1 in the Data section. There is a small icon on the right that looks like a spreadsheet. If you click it, you can view the data. You can also do this by typing View(data1) into the console. You can view the data in the console by typing

data1

data2= rbind(b,a)

data2

You can view the structure of these datasets using the *str* function again.

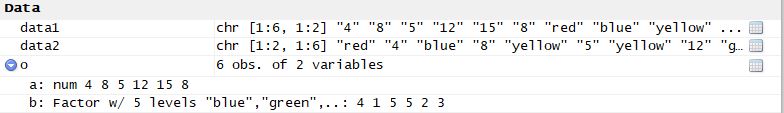
str(data1)

Since a matrix cannot contain more than one type of data, R will force the data to be all one type, in this case, character. This means that the numeric values no longer have numeric value, they are simply a label.

**Dataframes** are also a two dimensional data structure, but they can contain more than one type of data. You can create dataframes by combining vectors, but you use the *data.frame* function to do so. This still requires you to create an object.

o= data.frame(a,b)

You can view the data by typing o into the console. Your Global Environment has now populated an o in the data section, with the same spreadsheet icon as the matrix had. There is also now a blue triangle. You can click this triangle to view more information about the values in the dataframe.

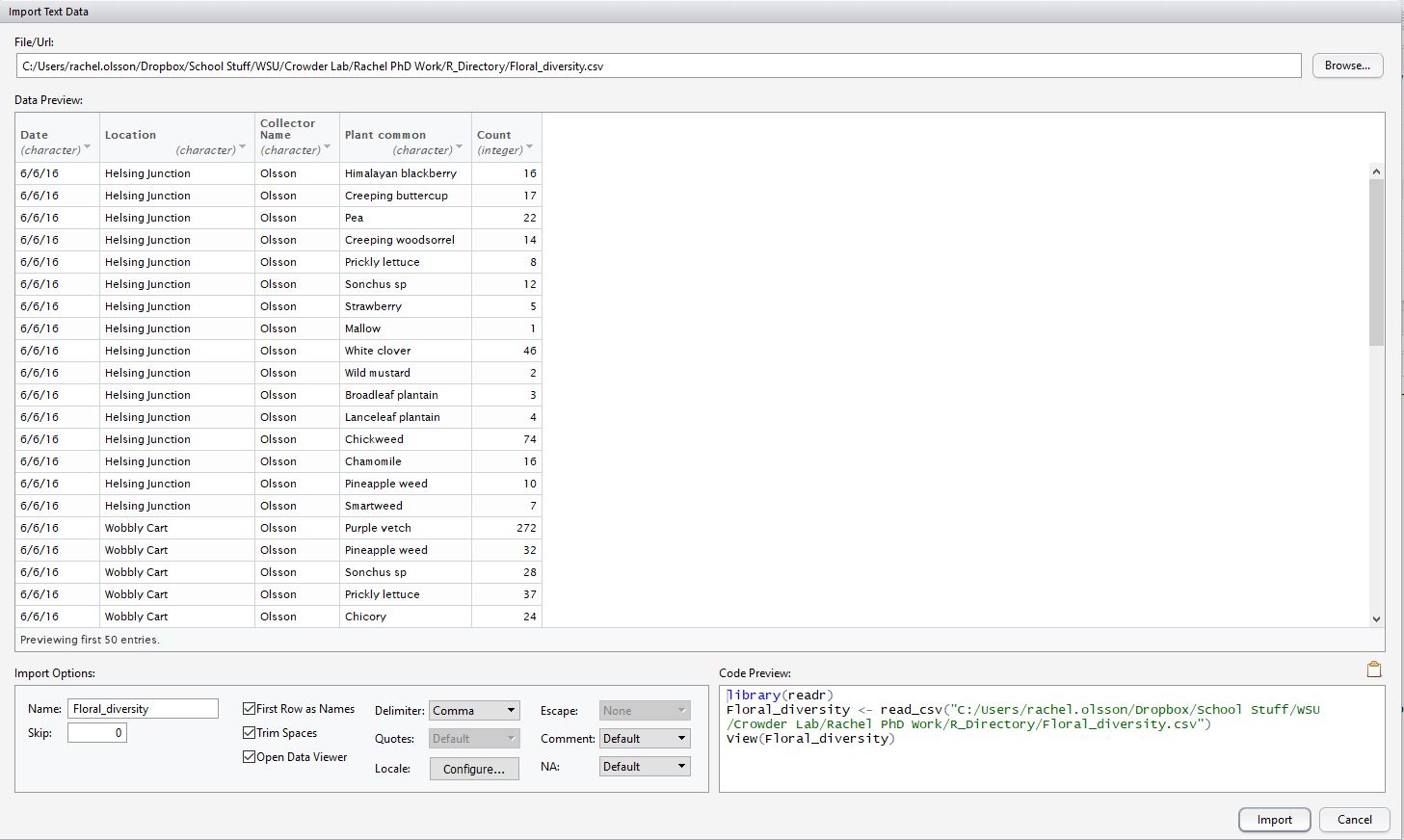


This shows us that the dataframe has kept the values differentiated in their data types. We have numeric data and factor data in the same dataframe.

**Importing your own data**

R can read data in a few different formats. Most commonly, R can use .csv or .xls files. You’ll notice a button in the Global Environment pane called Import Dataset. If you click it, you’ll have a few different options. For now, we will stick with Excel and CSV file types. Clicking one of the options may require you to download and install a new package, and R will prompt you to do so, and guide you through that process.

Once you click either Excel or CSV, a new window will open. Click the Browse button to view files in your computer. You can navigate through your files and folders outside of your working directory through this menu. A bunch of useful information will pop up in the preview:



You’ll see a preview of the data, including the type of data in each column, you’ll see code preview that can be saved into your script, and you’ll see a number of import options. If you are satisfied with the file, click Import.

This has created an object called Floral\_diversity.

You can interact with the values in imported dataframes the same way you did with dataframes we created, but this file is bigger than the one we created earlier. If you’d like to interact with the values in a specific column, you can express that by using the $ symbol.

mean(Floral\_diversity$Count) #this specifies that you want R to use the Floral\_diversity dataset, but only look at the values in the Count column.

**Plotting**

R has approximately 8 squillion different ways of plotting, and we will not get into most of them because they are far more complicated and intricate than this class will allow. We will go into a couple of them later on, but for now, I will just show you really quickly how to use the *plot* function. We can plot the number of counts of plants by the location using the plot function. However, plotting can’t use characters as values, so we need to convert the location names to factors instead. To do this, use the code

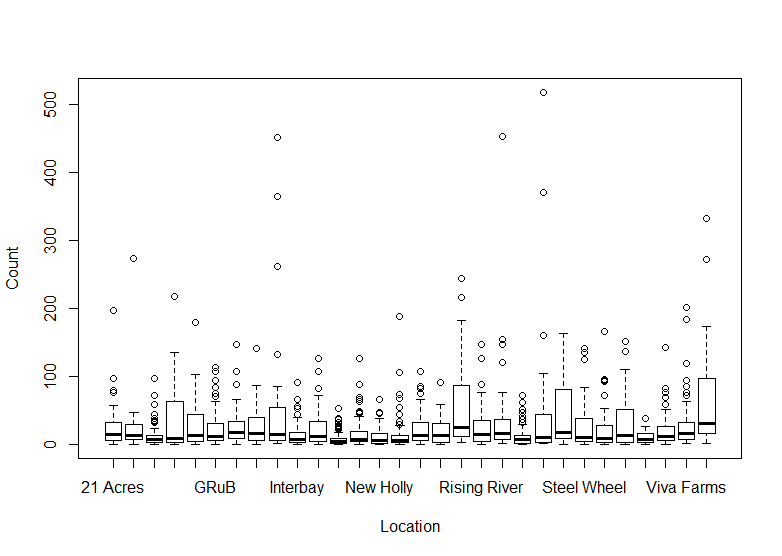
Floral\_diversity$Location=as.factor(Floral\_diversity$Location) #this looks a bit redundant, but it tells R to look at the Location column in the Floral\_diversity dataframe, and then change the type of data in that column to be a factor.

Once you’ve done that, you can use this code to plot out a graph.

plot(formula=Count~Location, data=Floral\_diversity) #the ~ acts as the term “by” for plotting.

The code now reads like this: make a plot, count by location, using the Floral\_diversity data.

This will generate a graph in the plots window of the lower right pane of RStudio. You can save the graph by clicking the Export button and saving it as an image or PDF.



This is a very basic overview of things we can do in R, but it covers a lot of the language and concepts that you will need for this class. When we begin doing more complex stats, we will continue to introduce you to packages, functions, and how to write more complicated code.

I recommend you check out the R users working group hosted by CEREO. We meet Wednesdays at noon in PACCAR 202. These group meetings focus on different packages, different ways people use R, and some troubleshooting help.

More helpful resources: These are all either books or blogs that are related to different ways you can use R.

<https://leanpub.com/rprogramming>

<https://www.manning.com/books/r-in-action-second-edition?a_bid=5c2b1e1d&a_aid=RiA2ed>

<http://www.cookbook-r.com/>

<https://www.r-bloggers.com/>