Sac State R Workshop Notes

Melissa McTernan, PhD

Fall 2018

Welcome

Hello! Welcome to the first Sac State Psychology Department R workshop. Some topics that we will cover today include:

- What is R, and how is it different from other stat programs (e.g. SPSS)?
- Some basic useful R functions for reading in data, data management, and calculating desriptive statistics
- t-tests, one-way ANOVA, correlation, and regression in R
- Data visualization using R

What is R?

R is a free, open source program for statistical computing. Beyond being free, there are many benefits to R:

- Open source software
    - developments are fast, so it is always on the cutting edge of new statistical methods
    - This is possible because R is made of of many user-developed ‘packages.’ A package is a set of functions, designed with a particular theme or purpose in mind. For example, you can install a package called ‘psych’ that consists of many useful functions for psychology researchers. Like R, all packages are free to install.
    - large community of online support
- Console-based software, syntax-driven
    - leads to better understanding of statistical procedures

Let’s take a peek…

There are two main platforms for running R. You can choose to work with *Base R* or you can work in *R Studio*. Base R is the most simple version, which you essentially operate by typing code directly into a console window, or a separate editor window called an editor, and results appear in the console. The graphics always appear in a separate graphics window.

R Studio was developed after R and is intended to be a more user-friendly platform. R-Studio displays all of the R windows in a single pane, and includes some buttons and menu-items for many common basic functions. There are also many other benefits to using R Studio that become apparent when you use R for more advanced tasks*. Note that we will be working in R Studio during this Worshop.

∗ For example, this document was created using R Markdown, which allows the user to integrate R syntax into a markdown document to create different types of output files, such as HTML, PDF, PowerPoint, and Word documents. There are other useful features that allow you to write complete APA style papers in R. With this functionality, you can write your papers and run your analyses in a single document, then plug statistical results directly into your manuscript, bypassing all the hassle (not to mention copy/paste errors and typos) of transferring results into a Word document and formatting it manually in APA.

R Studio

Console

- Contains output
- Can be used for trouble-shooting and during other times when you won't need to reproduce the syntax later
- Can be used as a giant calculator- try it! Type 2+2 into the console, then click 'Enter'.

Editor

- This is your clean, working syntax file
- You should be able to run all syntax in this document to complete a project, in theory
    - For example, in a single syntax file, you might read in data from Excel, rename a few variables, create a sum score, then with the new data, run summary statistics, run a t-test, create a bar chart, and export the bar chart as a .jpg to you hard drive

Environment/History/Connections

- *Environment* shows you what objects exist in R's 'working memory'
- *History* shows you what finctions you have run most recently

Files/Plots/Packages/Help/Viewer

- *Files* will display your folders from your hard-drive
- *Plots* your graphics will display in this window
- *Packages* allows you to manage packages in r
- *Help* you can get help with any function by typing `?` followed by the function name (with no space in between) directly into the console. The documentation for the function will appear in the 'Help' window.
- *Viewer* I forget what this window does…

Reading in Data, Data Management, and Descriptive Statistics

Reading Data into R

You can read almost any file type into R using the `read.table` series of functions. The two I use most often are `read.csv` and `read.xls` , which (as you might have guessed) can be used read in csv and Excel files, respectively.

Before you read data in from your computer, you need to tell R where to 'look' for it, or set the working directory, using the function `setwd`. You can get a list of all the files inside your working directory with the `list.files` function.

This is a good time to point out that R is **case-sensitive**, so you must make sure you type the file name exactly as it appears in your hard drive.

```
setwd("C:/Users/mcternan/Documents/R Workshop")
list.files()
```

```
## [1] "FAKE_mathdat.csv"    "FAKE_mathdat.xlsx"   "rsconnect"
## [4] "RWorkshop_F18.html"  "RWorkshop_F18.Rmd"   "RWorkshop_F18.tex"
```

I see the file I want! Now I will read it in to R. As I read the datafile into R, I want to save it as its own R object (R will read it in as an object type 'dataframe'). I will call the new data object "MathDat".

I am giving R two pieces of information to read in my datafile: the name of the file, and the information that the file contains a header row of variable names (this is the second **argument** in the code below, called `header`, and this argument is actually unnecessary because the default is `TRUE`).

```
MathDat <- read.csv("FAKE_mathdat.csv", header=TRUE)
```

Let's review - you can read the above syntax as "Hey, R! Can you please go look in my working directory for a file called Fake_mathdat.csv and bring it here? While you're at it, please put the data inside a new object and call it MathDat. Note that the first row contains variable names."

Note: If you forget to tell R to create a new object for the data, R will still read it in, but will print it to the console. (Well, what did you expect? You didn't tell R what to do with it!)

A Note about Function Arguments

Every function has at least one argument that must be passed to it. You can look at the function documentation (help page) to see what the argument names are. For example, see `?sd`. `sd` is the function for computing the standard deviation (which we will cover more later), and under Usage in the help page you can see that there are two arguments that can be passed to the function. The first argument is `x`, which is where you enter a list or vector of values for the calculation. The second argument is `na.rm`, which allows you to tell R whether it can ignore missing values in its calculation. **Importantly**, if you enter these arguments in order, you do not have to include the argument names (i.e. you could just say `sd(Age, TRUE)`) and R will know what to do with the information you are giving. However, if you enter the arguments *out of order* then you *must include the argument name* so R knows what's what. Example: `sd(na.rm=TRUE, x=Age)` will work but `sd(TRUE, Age)` will return an error. Why?

Data Management

Quick Checks

Sometimes you might want to take a quick look at the data to ensure that you brought in the file you intended to, that you read it in correctly, etc. In this section I will demonstrate some R functions for implementing those checks.

- Check the dimensions of a dataframe using the function `dim`
    - If for some reason you only need the number of rows or the number of columns, you could use `nrow` or `ncol` instead

```
dim(MathDat)
```

```
## [1] 60  8
```

- Take a peek at the structure of the data by displaying the top 6 rows with the `head` function

```
head(MathDat)
```

```
##   ID MathAnx SelfEfficacy MathScore College Age GenderID MathAfter
## 1  1      33           19        88    SSIS  19        0        88
## 2  2      29           17        76    SSIS  19        0        80
## 3  3      44           18        53    SSIS  18        0        70
## 4  4      32           16        94    SSIS  19        1        93
## 5  5      35           16        86    SSIS  18        1        88
## 6  6      28           18        90    SSIS  17        1        96
```

- Or, maybe you only need to check the variable names. You can view them using the `names` function. **This is pretty easy, isn't it?**

```
names(MathDat)
```

```
## [1] "ID"           "MathAnx"      "SelfEfficacy" "MathScore"
## [5] "College"      "Age"          "GenderID"     "MathAfter"
```

- Let's see how R is currently reading each variable. Note that R calls categorical variables 'factors' and numeric variables 'integers'. We can use the `str` function to peek at all the variables in our dataset:

```
str(MathDat)
```

```
## 'data.frame':    60 obs. of  8 variables:
##  $ ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ MathAnx     : int  33 29 44 32 35 28 31 33 34 31 ...
##  $ SelfEfficacy: int  19 17 18 16 16 18 15 14 15 17 ...
##  $ MathScore   : int  88 76 53 94 86 90 85 77 75 78 ...
##  $ College     : Factor w/ 3 levels "AL","NSM","SSIS": 3 3 3 3 3 3 3 3 3 3 ...
##  $ Age         : int  19 19 18 19 18 17 19 19 18 20 ...
##  $ GenderID    : int  0 0 0 1 1 1 1 0 0 1 ...
##  $ MathAfter   : int  88 80 70 93 88 96 95 82 81 80 ...
```

- A few things to note from this output:
    - Each variable is listed after a “$”. This operator denotes a variable inside a dataframe. We will come back to this in a bit.
    - All 6 variables in MathDat are currently being read as integer variables. We probably want GenderID to be categorical. We will fix this in the next section.
    - We get a peek at the first 10 values for each variable. This can serve as another quick check to make sure things appear as expected.

Quick Fixes / Data Management

- We noted in the previous section that R currently thinks GenderID is a numeric variable. One implication of this is that R will try to compute means, standard deviations, etc. for GenderID when we ask for summary statistics. It would be more appropriate to treat GenderID as categorical so we can look at the counts/frequencies instead. Here is some brief syntax that says to R, “Make my GenderID variable categorical, and then name this new cateogorical version”GenderID_c“.
    - BUT, we will have to do something special to tell R *where* to find the object ‘GenderID’. Remember that $ operator I mentioned? Here is comes:

    - In R, variables are stored as objects *inside* other objects. To refer to a variable then, we must tell R which object that variable exists within. `MathDat$GenderID` in R language means “the GenderID variable inside the MathDat dataframe.”

    - SO, we can now proceed with creating a categorical gender variable:

```
MathDat$GenderID_c <- as.factor(MathDat$GenderID)
```

Note: if you don’t want to create a new variable, and you just want to overwrite the original variable, just name the new object the same as the original object

Let’s check:

```
str(MathDat)
```

```
## 'data.frame':    60 obs. of  9 variables:
##  $ ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ MathAnx     : int  33 29 44 32 35 28 31 33 34 31 ...
##  $ SelfEfficacy: int  19 17 18 16 16 18 15 14 15 17 ...
##  $ MathScore   : int  88 76 53 94 86 90 85 77 75 78 ...
##  $ College     : Factor w/ 3 levels "AL","NSM","SSIS": 3 3 3 3 3 3 3 3 3 3 ...
##  $ Age         : int  19 19 18 19 18 17 19 19 18 20 ...
##  $ GenderID    : int  0 0 0 1 1 1 1 0 0 1 ...
##  $ MathAfter   : int  88 80 70 93 88 96 95 82 81 80 ...
##  $ GenderID_c  : Factor w/ 4 levels "0","1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
```

Everything looks good here! Let’s move on…

Calculating Descriptive Statistics

- The `summary` function is a quick way to get basic descriptives.
- If you have a small enough dataframe (not too many variables), you can get summary statistics for all variables at once by getting a summary of the entire data object:

```
summary(MathDat)
```

```
##        ID           MathAnx       SelfEfficacy     MathScore     College
##  Min.   : 1.00   Min.   :24.00   Min.   :11.00   Min.   :33.00   AL  :20
##  1st Qu.:15.75   1st Qu.:30.00   1st Qu.:15.00   1st Qu.:73.00   NSM :20
##  Median :30.50   Median :31.00   Median :16.00   Median :76.50   SSIS:20
##  Mean   :30.50   Mean   :31.65   Mean   :15.73   Mean   :76.97
##  3rd Qu.:45.25   3rd Qu.:34.00   3rd Qu.:17.00   3rd Qu.:83.00
##  Max.   :60.00   Max.   :44.00   Max.   :20.00   Max.   :98.00
##
##       Age           GenderID         MathAfter     GenderID_c
##  Min.   :17.00   Min.   :0.0000   Min.   :58.00   0:27
##  1st Qu.:19.00   1st Qu.:0.0000   1st Qu.:77.00   1:28
##  Median :19.00   Median :1.0000   Median :81.00   2: 3
##  Mean   :19.13   Mean   :0.6667   Mean   :82.66   3: 2
##  3rd Qu.:19.00   3rd Qu.:1.0000   3rd Qu.:89.75
##  Max.   :24.00   Max.   :3.0000   Max.   :99.00
##                                   NA's   :2
```

- Or, if you only want a summary of a particular variable, what could you do?

. . . . .

- That’s right! You could use the $ operator!

```
summary(MathDat$MathScore)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   33.00   73.00   76.50   76.97   83.00   98.00
```

```
summary(MathDat$GenderID_c)
```

```
##  0  1  2  3 
## 27 28  3  2
```

We often report standard deviations (*SD*) or variances along with the mean. The `sd` and the `var` functions can be used to calculate the *SD* and variance, respectively.

```
sd(MathDat$MathScore)
```

```
## [1] 10.33009
```

```
var(MathDat$MathScore)
```

```
## [1] 106.7107
```

Calculating New Variables / Data Manipulation

Notice that the dataset includes a before score and an after score for a math test. These were taken before and after participants completed a semester-long math course. Perhaps we are interested in whether the change in math performance is related to self efficacy. It might be helpful to calculate each individual's change score. You can do this using simple math:

```
MathDat$Diff <- MathDat$MathAfter - MathDat$MathScore
head(MathDat)
```

```
##   ID MathAnx SelfEfficacy MathScore College Age GenderID MathAfter
## 1  1      33           19        88    SSIS  19        0        88
## 2  2      29           17        76    SSIS  19        0        80
## 3  3      44           18        53    SSIS  18        0        70
## 4  4      32           16        94    SSIS  19        1        93
## 5  5      35           16        86    SSIS  18        1        88
## 6  6      28           18        90    SSIS  17        1        96
##   GenderID_c Diff
## 1          0    0
## 2          0    4
## 3          0   17
## 4          1   -1
## 5          1    2
## 6          1    6
```

Challenge: Take a moment to think about how you would describe the above syntax in plain English. What would have happened if I typed `Diff <- MathAfter - MathScore` ?

Subsetting a dataset

Another common practice in research data management is subsetting. For example, suppose we want to only focus on SSIS majors for part of our study. We can tell R that we want to create a new dataframe that only includes a subset of the original rows:

```
SSIS <- subset(MathDat, College == "SSIS")
```

This translates to “Hi again, R! Can you take a look at the MathDat object and pull out all the rows for which College has a value of ‘SSIS’? Can you put those into their own dataset and call it ’SSIS”? Thanks! xoxo, me"

Notice that nothing will “happen” in your console, but if the code was properly run, you should see a new data object listed in your Global Environment pane.

What if you wanted to keep SSIS majors *and* Arts and Letters majors in your subset? You can modify your syntax to tell R you want to keep all observations for which College = SSIS *or* AL:

```
SSIS_AL <- subset(MathDat, College == "SSIS" | College == "AL")
```

Tip: Always look at the dimensions of the new data object after subsetting and make sure they are what you expect! Sometimes you might not get an error for this command, but the data may not actually have subsetted properly. What other function(s) might you use to reassure yourself that the subset function worked the way you wanted it to?

You might also want to subset your dataframe by columns rather than observations. For example, maybe I want to only look at the summary statistics for MathAnx, SelfEfficacy, and MathScore. These variables are in columns 2, 3, and 4, respectively. A simple way to subset to only these columns is to use square brackets (`[ ]`) and indexing. Try this:

```
newdat <- MathDat[2:4]
```

Don’t forget to check the dimensions!

Now we can use the `summary` function on newdat:

```
summary(newdat)
```

```
##     MathAnx       SelfEfficacy     MathScore   
##  Min.   :24.00   Min.   :11.00   Min.   :33.00  
##  1st Qu.:30.00   1st Qu.:15.00   1st Qu.:73.00  
##  Median :31.00   Median :16.00   Median :76.50  
##  Mean   :31.65   Mean   :15.73   Mean   :76.97  
##  3rd Qu.:34.00   3rd Qu.:17.00   3rd Qu.:83.00  
##  Max.   :44.00   Max.   :20.00   Max.   :98.00  
```

How would we have to modify this syntax if we wanted to also include Age and MathAfter in the subset? The syntax gets a little uglier, I will admit:

```
newdat2 <- MathDat[c(2:4, 6, 8)]
```

Let's use the `names` function to make sure we got the right variables.

```
names(newdat2)
```

```
## [1] "MathAnx"      "SelfEfficacy" "MathScore"    "Age"         
## [5] "MathAfter"
```

Looks right! we could now use the summary function on newdat2 to calculate summary statistics for these 5 variables. However, I will take this opportunity to instead show you how you can nest functions in R. We will nest the subsetting feature inside the summary feature to get a summary of only these 5 columns *without actually having to create a permanent new data object*.

```
summary(MathDat[c(2:4, 6, 8)])
```

```
##     MathAnx       SelfEfficacy     MathScore          Age       
##  Min.   :24.00   Min.   :11.00   Min.   :33.00   Min.   :17.00  
##  1st Qu.:30.00   1st Qu.:15.00   1st Qu.:73.00   1st Qu.:19.00  
##  Median :31.00   Median :16.00   Median :76.50   Median :19.00  
##  Mean   :31.65   Mean   :15.73   Mean   :76.97   Mean   :19.13  
##  3rd Qu.:34.00   3rd Qu.:17.00   3rd Qu.:83.00   3rd Qu.:19.00  
##  Max.   :44.00   Max.   :20.00   Max.   :98.00   Max.   :24.00  
##                                                                 
##    MathAfter    
##  Min.   :58.00  
##  1st Qu.:77.00  
##  Median :81.00  
##  Mean   :82.66  
##  3rd Qu.:89.75  
##  Max.   :99.00  
##  NA's   :2
```

Challenge: run the following syntax. What do you think is happening? What does the `round` function do? Remember you could type `?round` to get help on this function.

```
round(mean(MathDat$Age), 2)
```

```
## [1] 19.13
```

```
round(mean(MathDat$Age), 4)
```

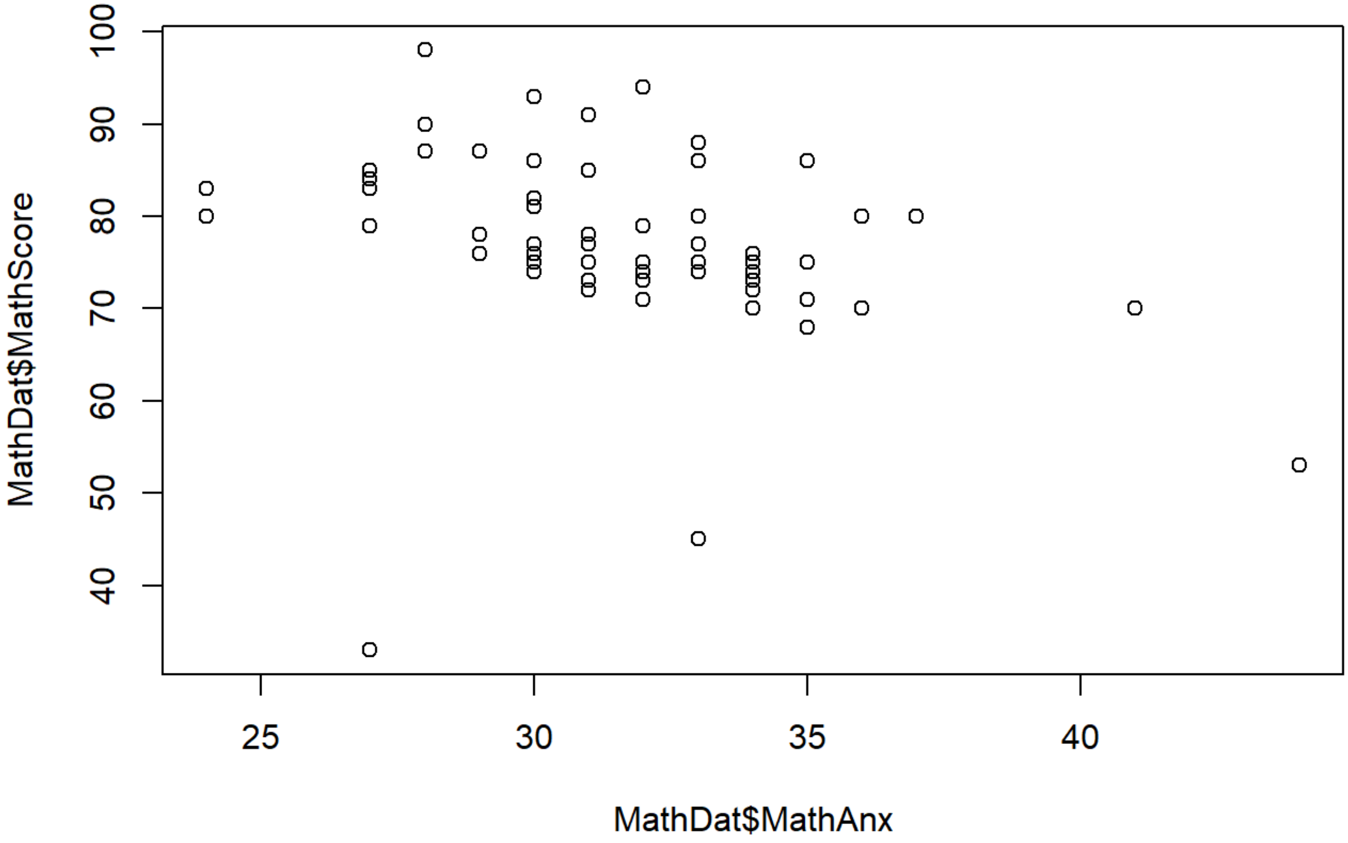
```
## [1] 19.1333
```

Data Visualization in R

Base graphics

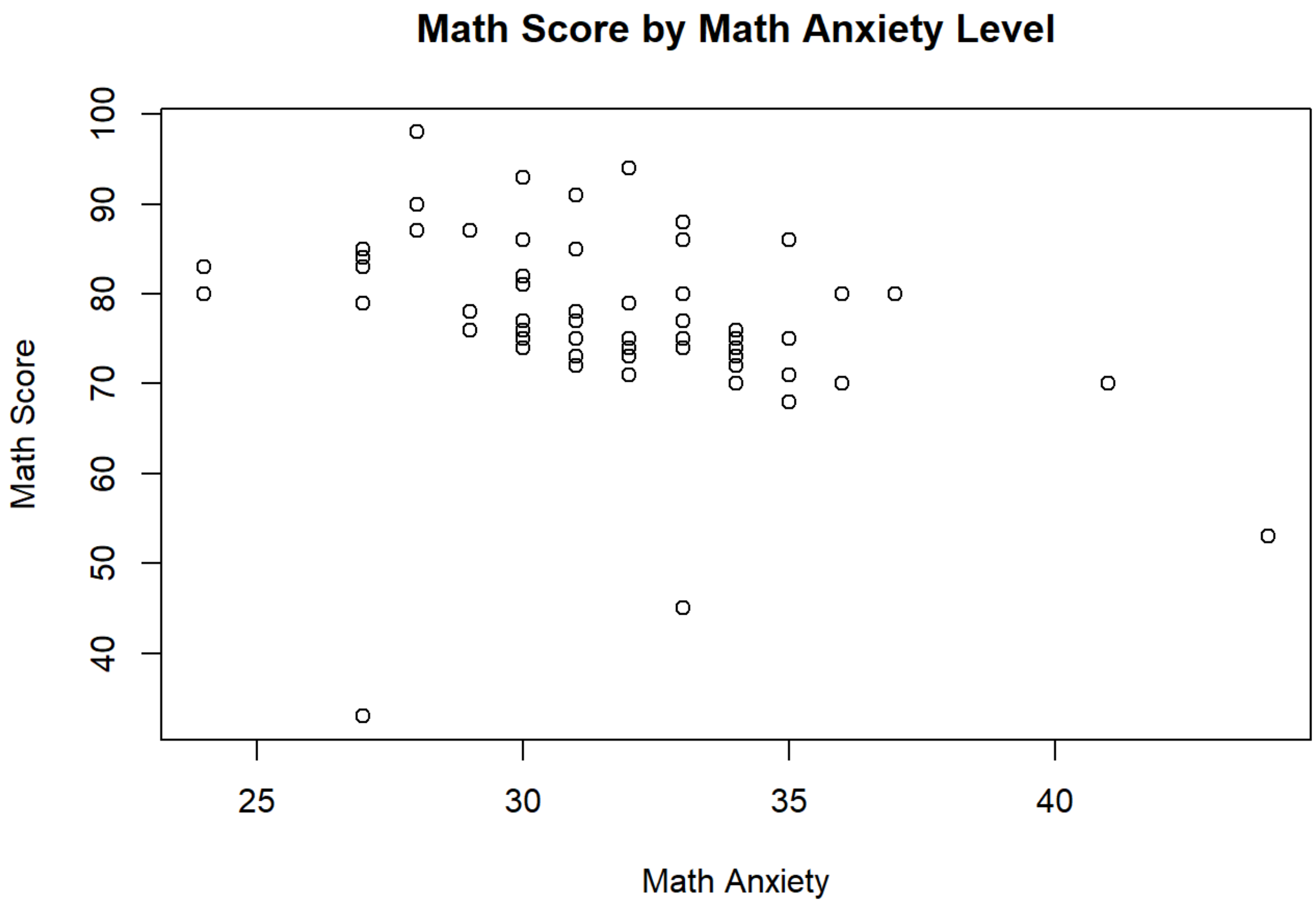
Scatter Plot

```
plot(MathDat$MathAnx, MathDat$MathScore)
```

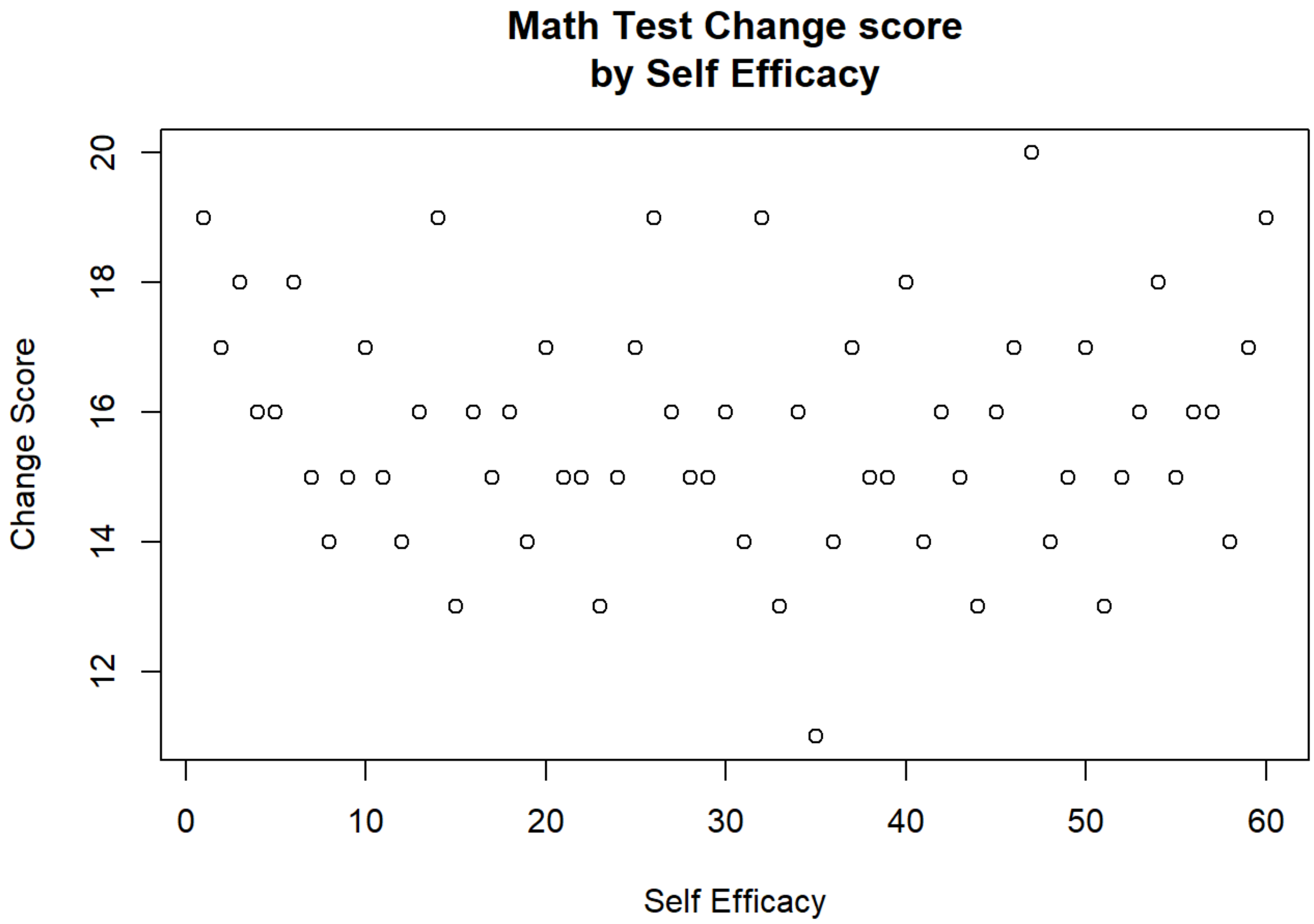


Notice that these plots are so simple to create, but are also very visaully simplistic and the axis labels are pretty atrocious to look at. You can add axis labels and a title using the argumants “xlab”, “ylab”, and “main”:

```
plot(MathDat$MathAnx, MathDat$MathScore, ylab="Math Score", xlab="Math Anxiety", main
="Math Score by Math Anxiety Level")
```



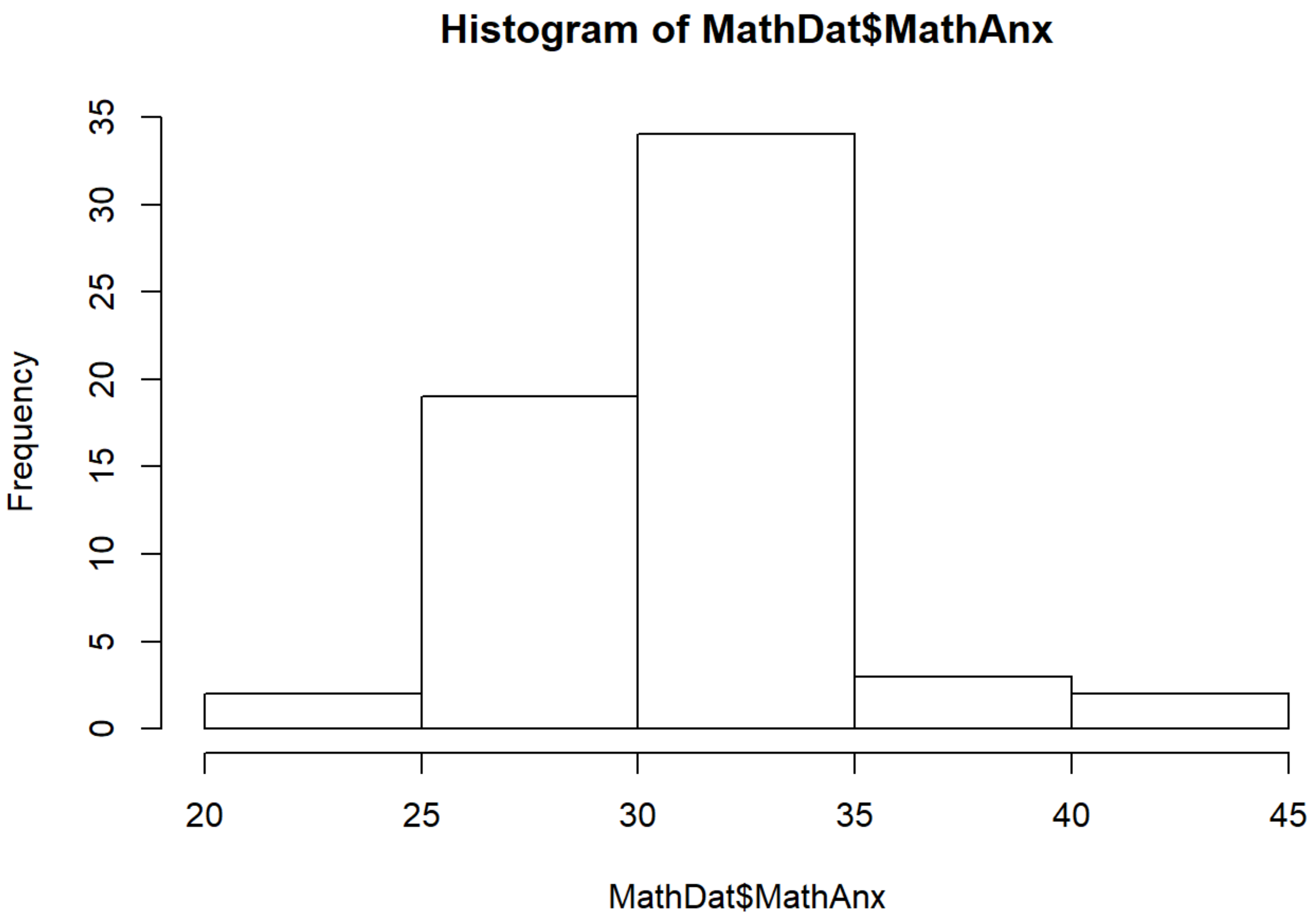
```
plot(MathDat$SelfEfficacy, MathDat$MathDiff, ylab="Change Score", xlab="Self Efficacy
", main="Math Test Change score\nby Self Efficacy")
```



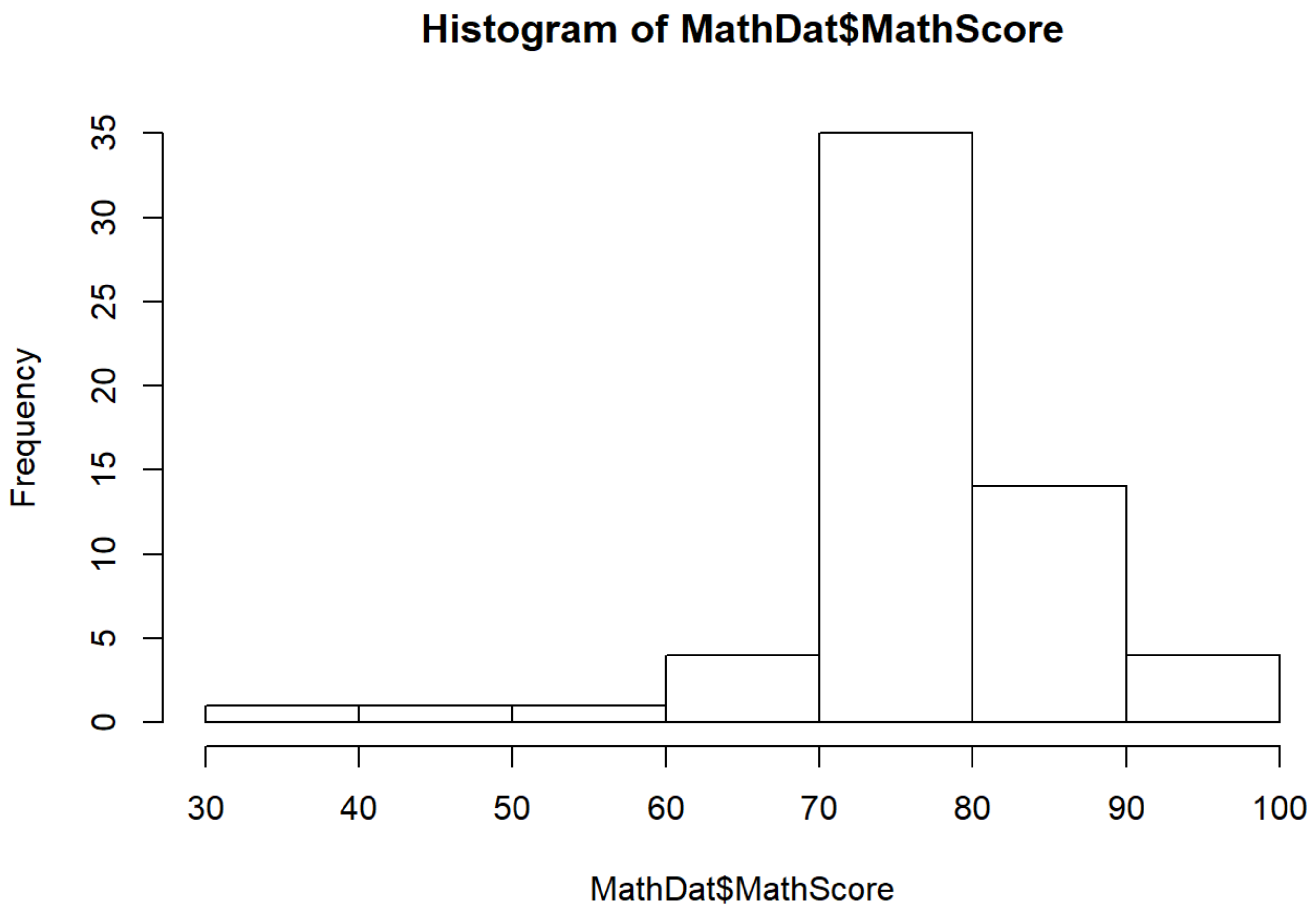
Notice something funny about the title specification in the "main" title argument above? The `\n` is code for "new line" and allows you to split a long plot title across two lines of text.

Histogram

```
hist(MathDat$MathAnx)
```

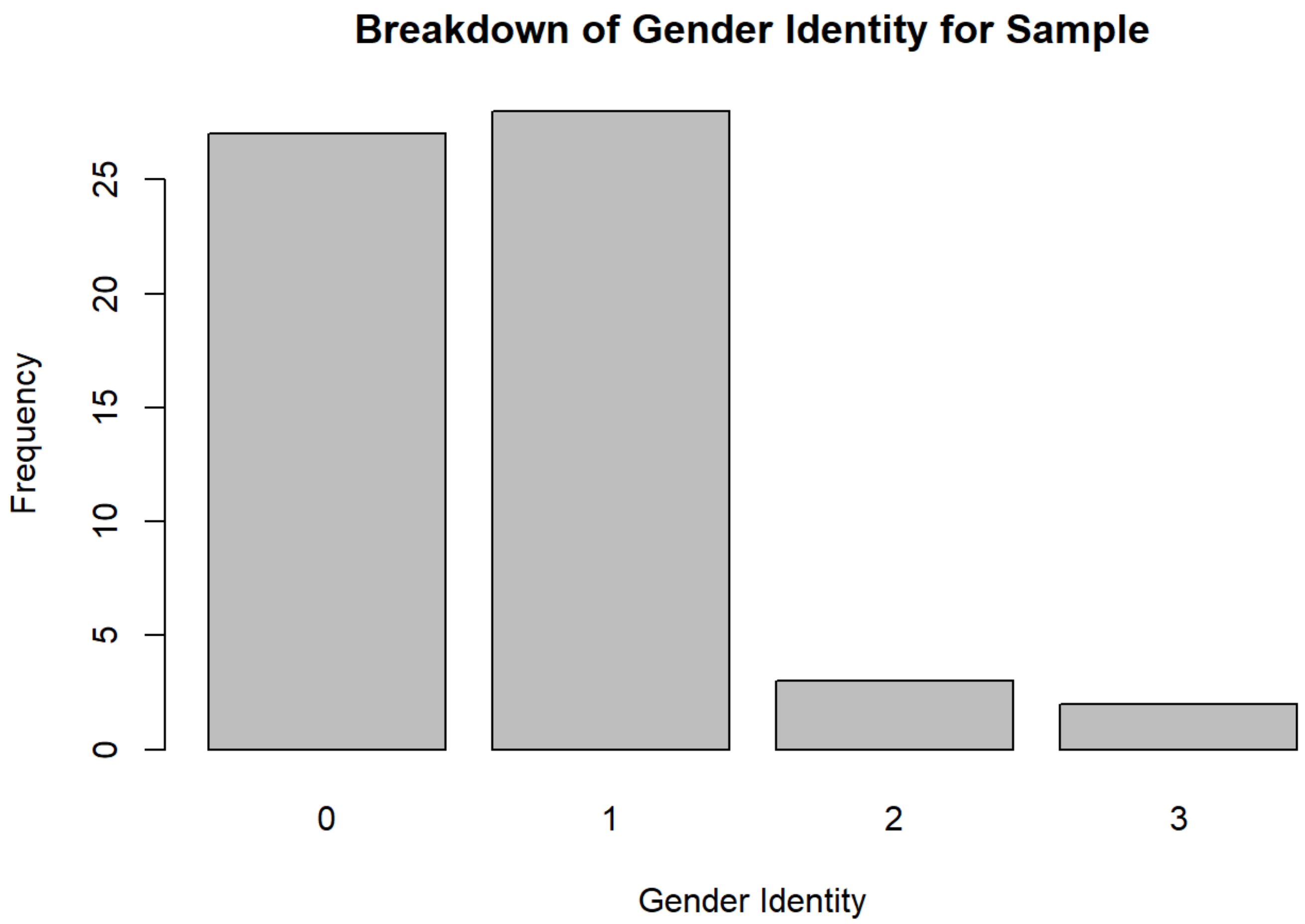


```
hist(MathDat$MathScore)
```



Bar Chart

```
plot(MathDat$GenderID_c, xlab="Gender Identity", ylab="Frequency", main="Breakdown of
Gender Identity for Sample")
```



Let’s pretty these up a bit using package `ggplot2`

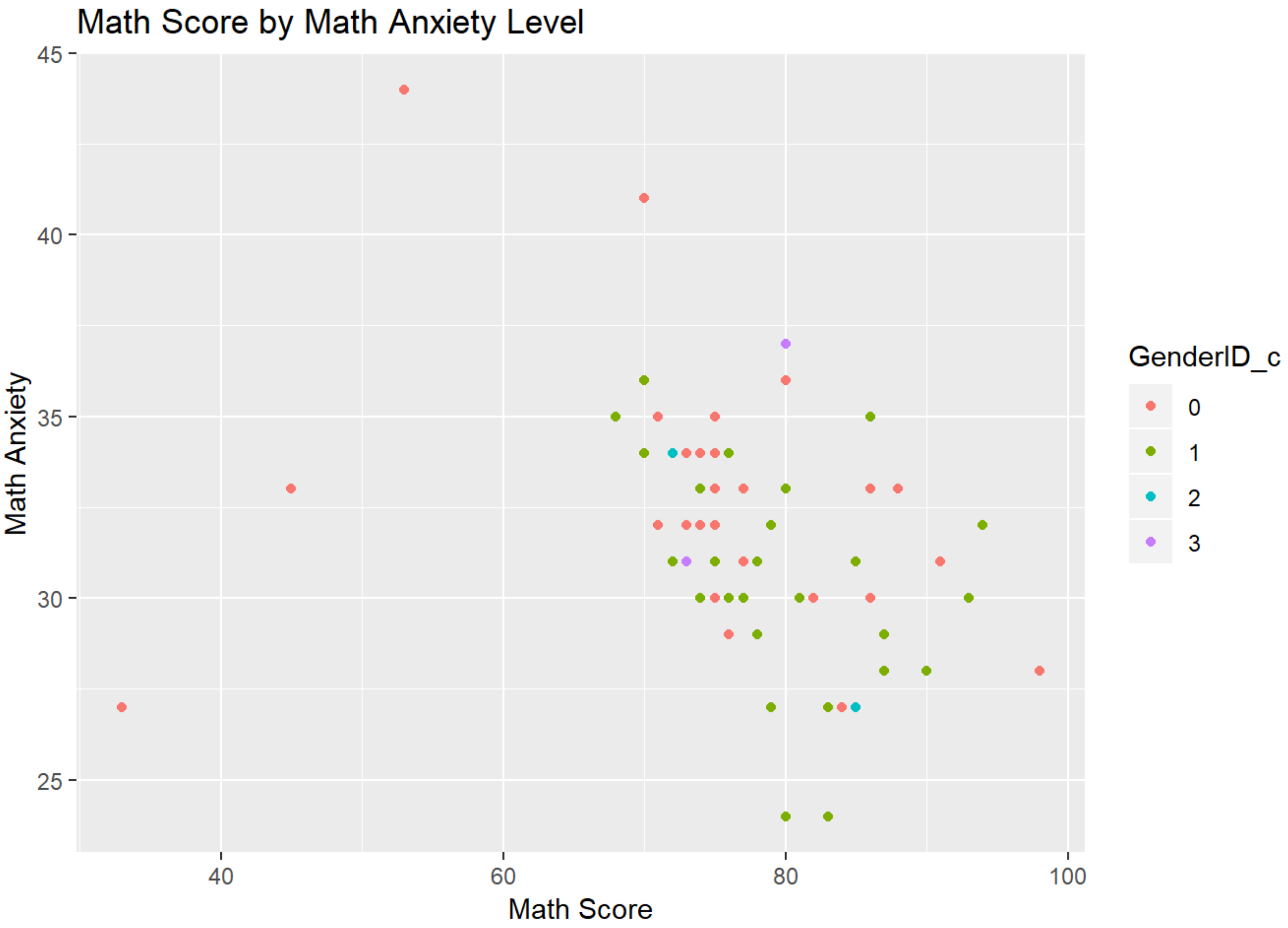
If you haven’t installed the package (you only have to do this one time), you can install it using: `install.packages("ggplot2")`.

Once installed, you must load the package every time you want to use it. You can load a package using `library("ggplot2")`.

The syntax for ggplot can be a little daunting, but there are many helpful resources onlin (linked at the end of this document) and, just as with anything else, practice helps!

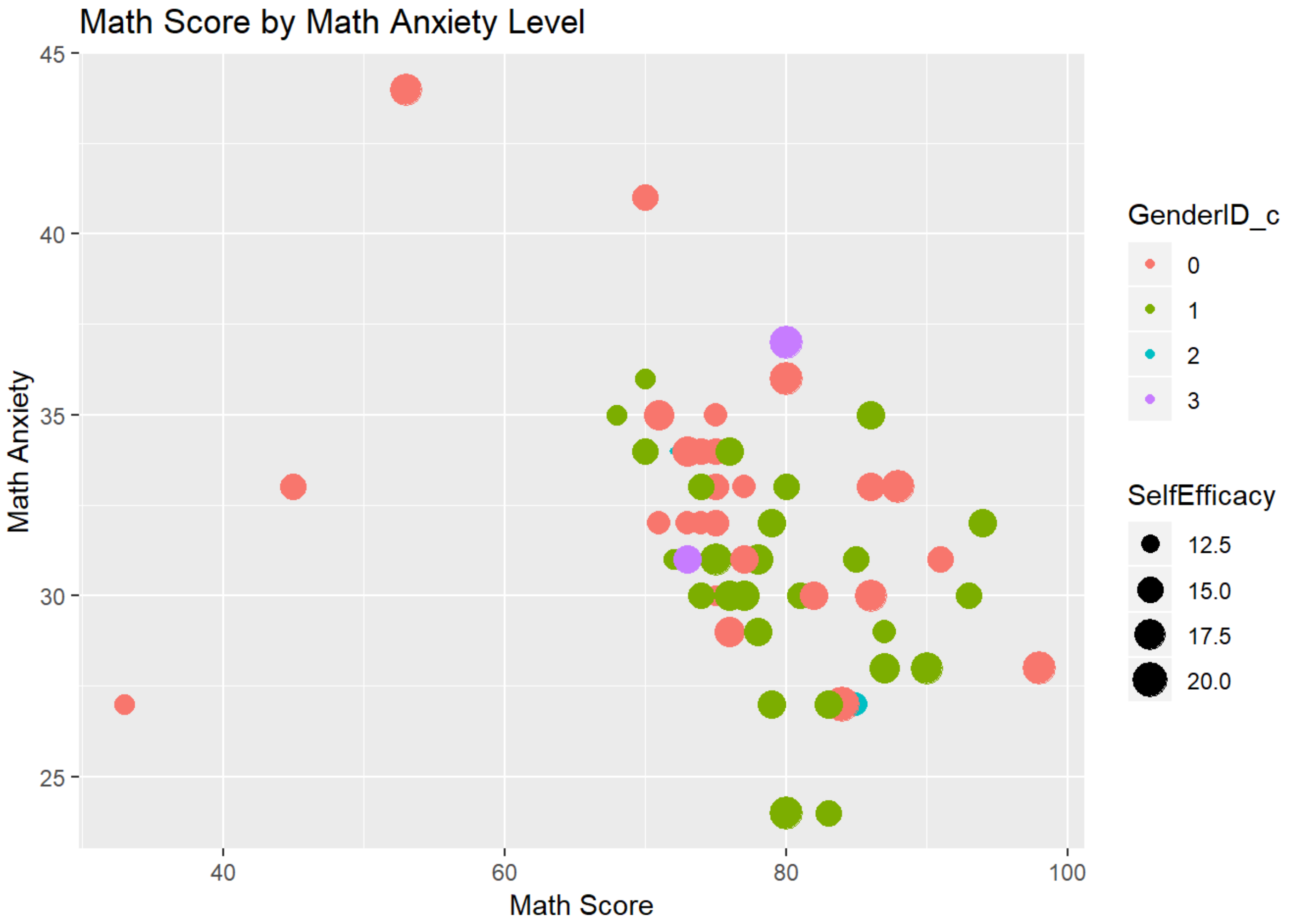
gg Scatter Plot

```
ggplot(data=MathDat, aes( x=MathDat$MathScore, y=MathDat$MathAnx, color=GenderID_c))
+ geom_point() + xlab("Math Score") + ylab("Math Anxiety") + ggtitle("Math Score by M
ath Anxiety Level")
```



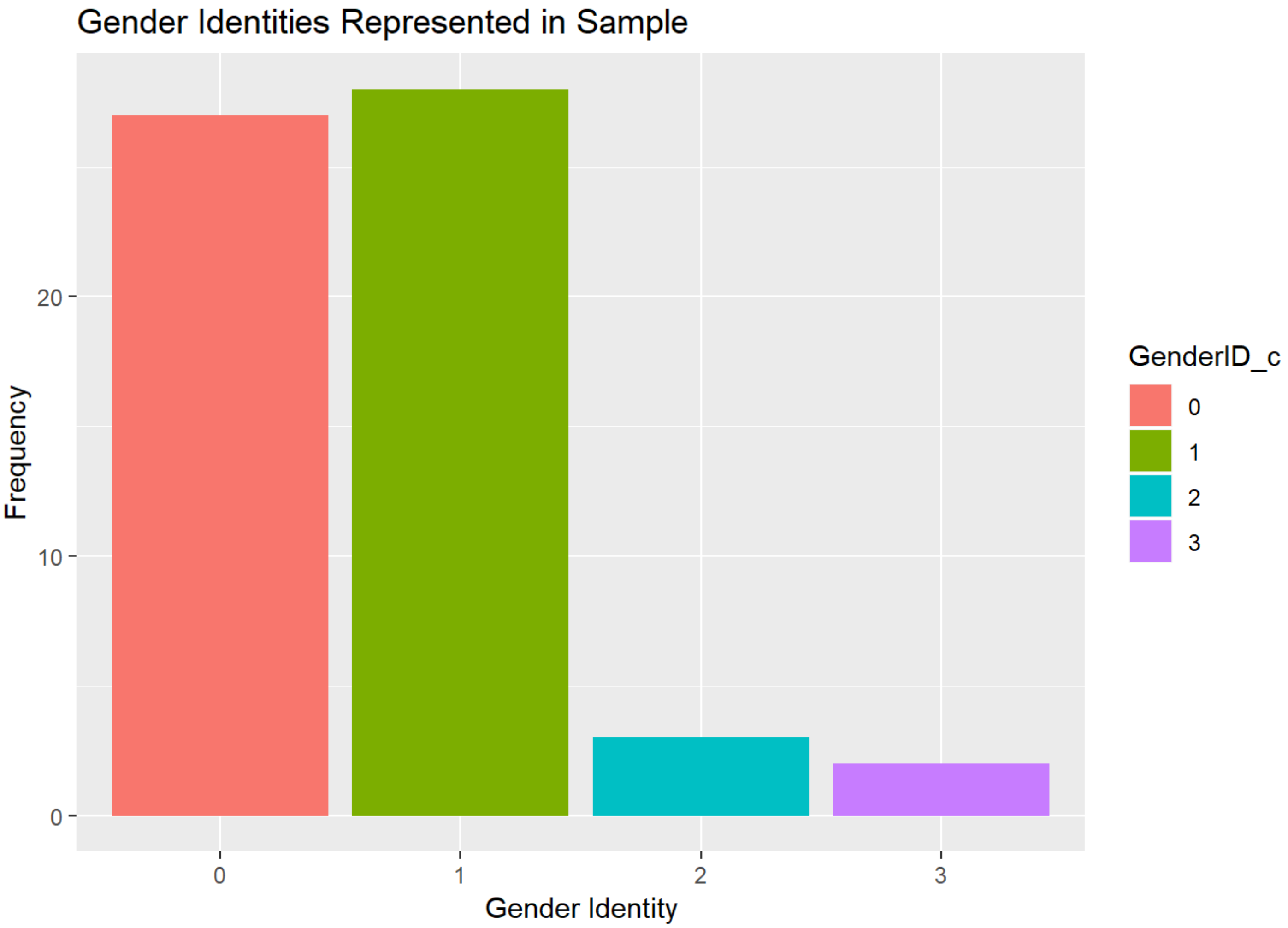
Let’s add Another variable. Suppose we also want to visualize the relation between Self Efficacy and Math Scores with Math Anxiety levels. One way to do this is to vary the point size based on participant Self Efficacy score.

```
ggplot(data=MathDat, aes( x=MathDat$MathScore, y=MathDat$MathAnx, size=SelfEfficacy,
color=GenderID_c)) + geom_point() + xlab("Math Score") + ylab("Math Anxiety") + ggtit
le("Math Score by Math Anxiety Level")
```



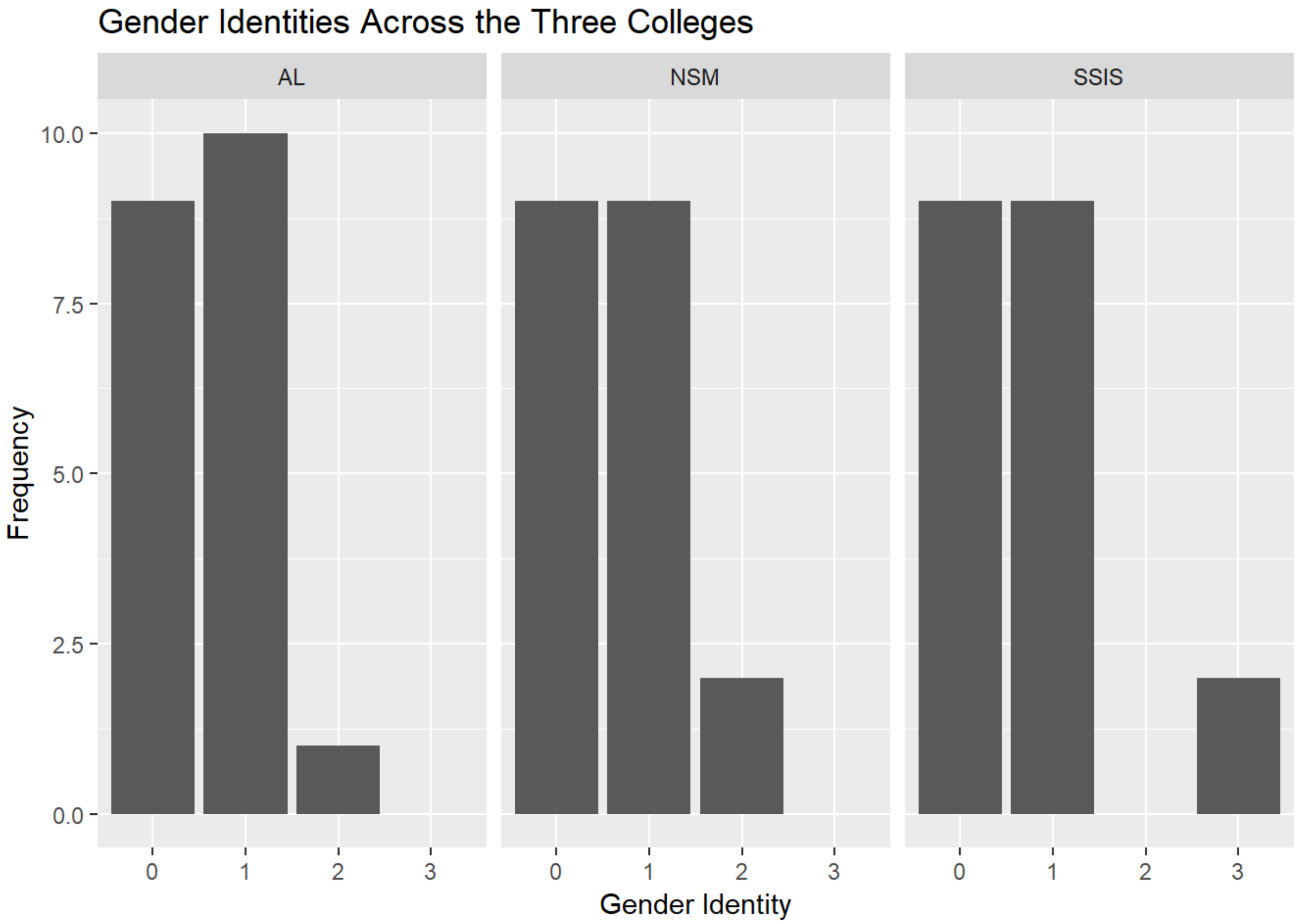
gg Barchart

```
ggplot(data=MathDat, aes(MathDat$GenderID_c, fill=GenderID_c)) + geom_bar() + xlab("Gender Identity") + ylab("Frequency") + ggtitle("Gender Identities Represented in Sample")
```



`ggplot` offers so many different kinds of plots. One of my favorite features is the `facet_wrap` feature, allowing the user to make grids of related plots. For example, we could plot the gender identity representation across the three colleges for a side-by-side comparison by adding the variable College to the `facet_wrap` :

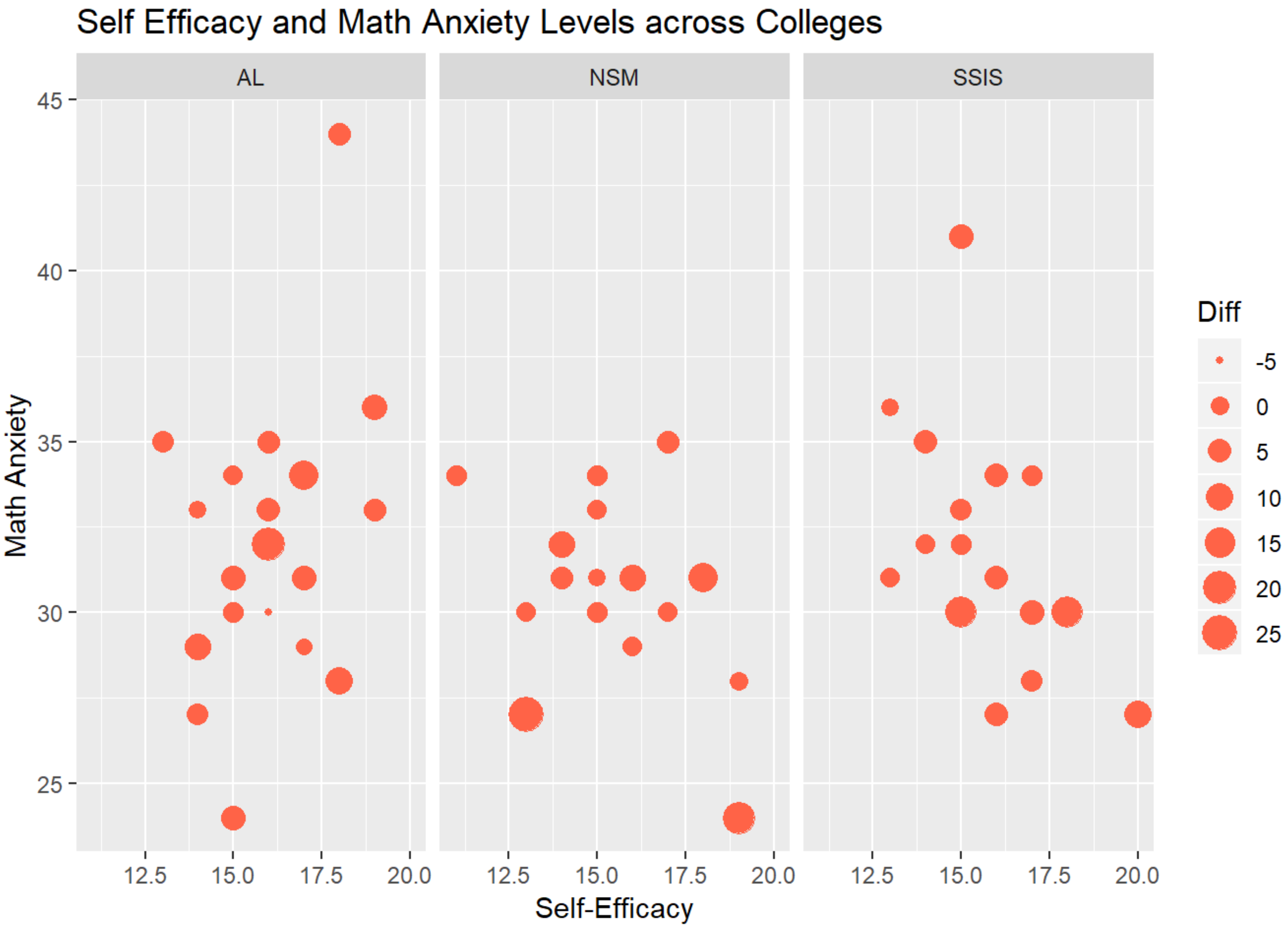
```
ggplot(data=MathDat, aes(MathDat$GenderID_c)) + geom_bar() + xlab("Gender Identity")
+ ylab("Frequency") + ggtitle("Gender Identities Across the Three Colleges") + facet_
wrap(MathDat$College)
```



or we might want to plot the relationship between self-efficacy and math anxiety across the three colleges:

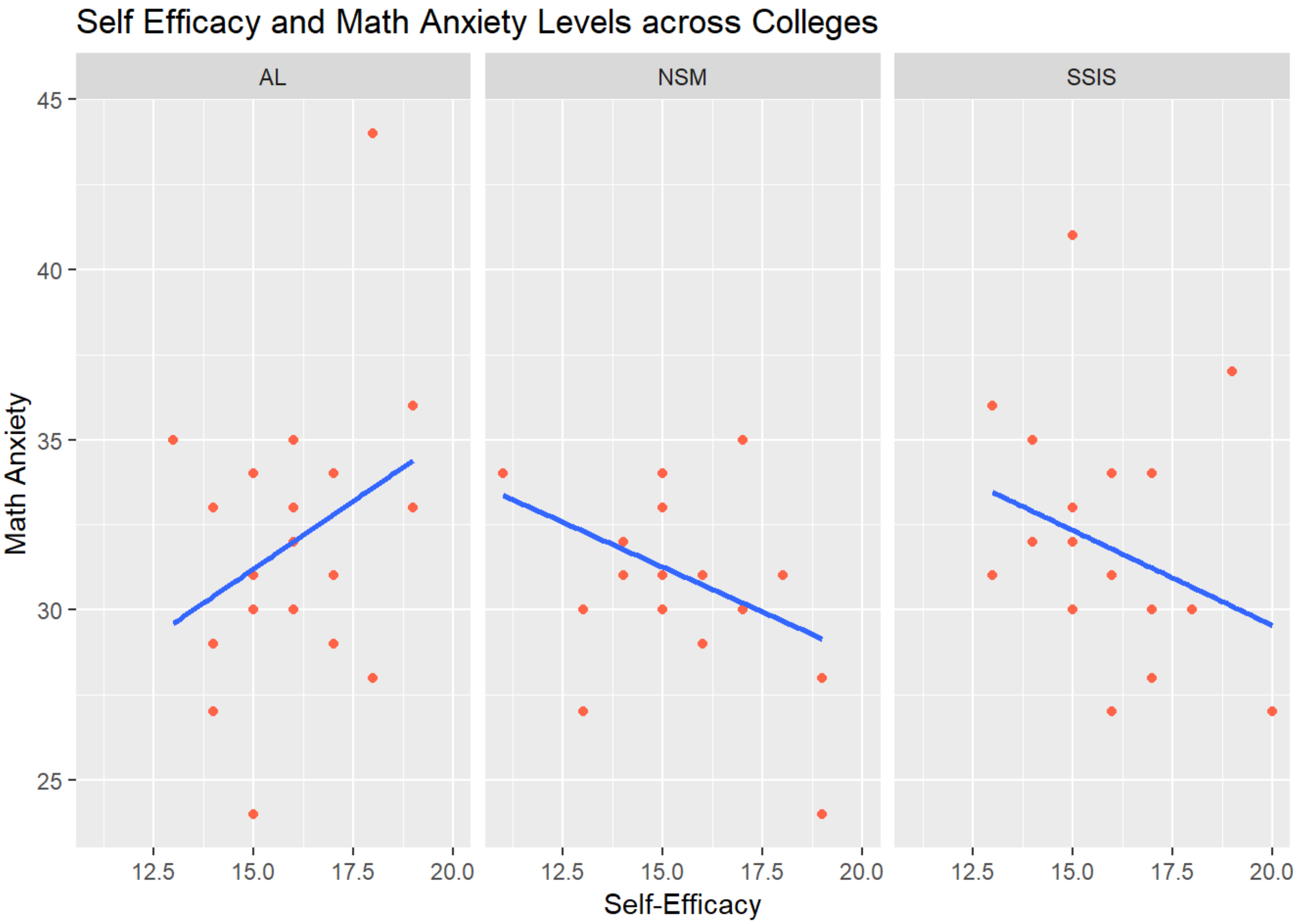
```
ggplot(data=MathDat, aes( x=MathDat$SelfEfficacy, y=MathDat$MathAnx, size=Diff)) + ge
om_point(color="tomato") + xlab("Self-Efficacy") + ylab("Math Anxiety") + ggtitle("Se
lf Efficacy and Math Anxiety Levels across Colleges") + facet_wrap(MathDat$College)
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



We could add yet another feature to the plot using `stat_smooth` with `method=lm` which will add a line-of-best-fit (using the method of least squares) to the plot space.

```
ggplot(data=MathDat, aes(x=MathDat$SelfEfficacy, y=MathDat$MathAnx)) + geom_point(col
or="tomato") + stat_smooth(method = "lm", se = FALSE) + xlab("Self-Efficacy") + ylab(
"Math Anxiety") + ggtitle("Self Efficacy and Math Anxiety Levels across Colleges") +
facet_wrap(MathDat$College)
```



Brief Intro to Statistical Analysis using R

Correlation

The `cor` function will calculate Pearson's R. Let's calculate the correlation betwee Math Scores (before taking the math class) and Math Anxiety. The two required arguments for `cor` are the two variables that you want to correlate.

```
cor(MathDat$MathAnx, MathDat$MathScore)
```

```
## [1] -0.318327
```

Notice that the output for `cor` only reports the Pearson's r value, and not an associated test for statistical significance. You may find that the `cor.test` function is more useful because it will include both of these pieces of information in the output.

```
cor.test(MathDat$MathAnx, MathDat$MathScore)
```

```
##
##  Pearson's product-moment correlation
##
## data:  MathDat$MathAnx and MathDat$MathScore
## t = -2.5573, df = 58, p-value = 0.01319
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.52945528 -0.07006583
## sample estimates:
##       cor
## -0.318327
```

Let’s do another. We will use `cor.test` to see if participants’ change scores on the math test are associated with Self Efficacy.

```
cor.test(MathDat$Diff, MathDat$SelfEfficacy)
```

```
##
##  Pearson's product-moment correlation
##
## data:  MathDat$Diff and MathDat$SelfEfficacy
## t = 1.7274, df = 56, p-value = 0.08962
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.0354415  0.4566786
## sample estimates:
##       cor
## 0.2249132
```

t-tests

t-tests are very common in Psychology and are easily implemented in R. Below we demonstrate a single sample t-test, an independent samples t-test, and a dependent samples t-test. Note that all of these tests can be implemented within the function `t.test`.

Single Sample t-test

Let’s use a single sample test to test the hypothesis that our sample has a different baseline level of math performance than the general population of college students (μ = 73).

```
t.test(MathDat$MathScore, mu=73)
```

```
##
##  One Sample t-test
##
## data:  MathDat$MathScore
## t = 2.9744, df = 59, p-value = 0.004248
## alternative hypothesis: true mean is not equal to 73
## 95 percent confidence interval:
##  74.29812 79.63521
## sample estimates:
## mean of x
##  76.96667
```

Notice in the output that R tells us the alternative hypothesis for the test. In this case, the alternative hypothesis is that our sample mean is *different* than the population mean (and the null hypothesis would be that our sample mean is equal to the population mean). This should indicate to you that R performed a two-tailed test. The `t.test` function will always default to a two-tailed test unless you tell it otherwise, which you can do using the argument `alternative`, as demonstrated below. We will specify ‘greater’ to test the hypothesis that our sample mean is greater than the population mean.

```
t.test(MathDat$MathScore, mu=73, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  MathDat$MathScore
## t = 2.9744, df = 59, p-value = 0.002124
## alternative hypothesis: true mean is greater than 73
## 95 percent confidence interval:
##  74.73808      Inf
## sample estimates:
## mean of x
##  76.96667
```

Challenge: Look at the help page for `t.test` to determine the syntax for running a single sample t-test in which you want to test the hypothesis that your sample mean is *less* than the population mean.

Independent Samples t-test

Suppose we need to test whether SSIS majors and AL majors have different baseline math abilities. Remember that we created a subsetted dataframe ealier that only includes SSIS and AL in the College variable. We called the new dataset SSIS_AL. This will come in handy now!

```
t.test(SSIS_AL$MathScore~SSIS_AL$College)
```

```
##
##  Welch Two Sample t-test
##
## data:  SSIS_AL$MathScore by SSIS_AL$College
## t = -1.6669, df = 30.404, p-value = 0.1058
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -8.5643623  0.8643623
## sample estimates:
##   mean in group AL mean in group SSIS
##              76.45              80.30
```

One of the neat things about R being open-source software is that people are constantly developing new features. Some of those features are called ‘wrappers’ are are just used to make an existing function more user friendly. Someone wrote a wrapper for `t.test` called `independentSamplesTTest` that produces prettier output, and also reports Cohen’s *d*, a measure of effect size. This wrapper function is in the package `lsr`.

```
independentSamplesTTest(MathScore~College, SSIS_AL)
```

```
## Warning in independentSamplesTTest(MathScore ~ College, SSIS_AL): grouping
## variable has unused factor levels
```

```
##
##    Welch's independent samples t-test
##
## Outcome variable:   MathScore
## Grouping variable:  College
##
## Descriptive statistics:
##                 AL   SSIS
##    mean     76.450 80.300
##    std dev.  5.165  8.945
##
## Hypotheses:
##    null:        population means equal for both groups
##    alternative: different population means in each group
##
## Test results:
##    t-statistic:  -1.667
##    degrees of freedom:  30.404
##    p-value:  0.106
##
## Other information:
##    two-sided 95% confidence interval:  [-8.564, 0.864]
##    estimated effect size (Cohen's d):  0.527
```

Dependent (Paired) Samples t-test

We may want to test whether math scores significantly improved from the baseline test to the second assessment. Because the scores came from a single sample of people, taken at two time points, this research question calls for a dependent (or paired) samples t-test. We can modify the `t.test` syntax to include the argument `paired=TRUE` to conduct this test.

```
t.test(MathDat$MathScore, MathDat$MathAfter, paired=TRUE)
```

```
##
##  Paired t-test
##
## data:  MathDat$MathScore and MathDat$MathAfter
## t = -6.5905, df = 57, p-value = 1.527e-08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -6.609131 -3.528800
## sample estimates:
## mean of the differences
##               -5.068966
```

There is also a wrapper for the dependent test called `pairedSamplesTTest` that is accessible once you've installed the `lsr` package.

```
pairedSamplesTTest(~ MathScore + MathAfter, MathDat)
```

```
## Warning in pairedSamplesTTest(~MathScore + MathAfter, MathDat): 2 case(s)
## removed due to missingness
```

```
## 
##    Paired samples t-test 
## 
## Variables:  MathScore , MathAfter 
## 
## Descriptive statistics: 
##             MathScore MathAfter difference
##    mean        77.586    82.655     -5.069
##    std dev.     9.584     8.753      5.858
## 
## Hypotheses: 
##    null:        population means equal for both measurements
##    alternative: different population means for each measurement
## 
## Test results: 
##    t-statistic:  -6.59 
##    degrees of freedom:  57 
##    p-value:  <.001 
## 
## Other information: 
##    two-sided 95% confidence interval:  [-6.609, -3.529] 
##    estimated effect size (Cohen's d):  0.865
```

Analysis of Variance (ANOVA)

Perhaps you are interested in testing whether there are significant differences across majors in Math Score improvement. Because we have three means to compare (mean change score for SSIS, NSM, and AL), we must use an ANOVA. The function for an ANOVA in R is `aov` and is demonstraed below:

```
aov(MathDat$Diff~MathDat$College)
```

```
## Call:
##    aov(formula = MathDat$Diff ~ MathDat$College)
## 
## Terms:
##                 MathDat$College Residuals
## Sum of Squares           1.7215 1954.0026
## Deg. of Freedom               2        55
## 
## Residual standard error: 5.96048
## Estimated effects may be unbalanced
## 2 observations deleted due to missingness
```

The `aov` function doesn’t produce much output, but we can get a summary of the results for more information using `summary` or you could also tru the `Anova` function in the `car` package. We will just stick with `summary` for this example.

```
summary(aov(MathDat$Diff~MathDat$College))
```

```
##                 Df Sum Sq Mean Sq F value Pr(>F)
## MathDat$College  2    1.7    0.86   0.024  0.976
## Residuals       55 1954.0   35.53               
## 2 observations deleted due to missingness
```

Sometimes you might want to name a model, so you can get a summary of it later. such as:

```
Model1 <- aov(MathDat$Diff~MathDat$College)
summary(Model1)
```

```
##                 Df Sum Sq Mean Sq F value Pr(>F)
## MathDat$College  2    1.7    0.86   0.024  0.976
## Residuals       55 1954.0   35.53               
## 2 observations deleted due to missingness
```

This output doesn't include the group-level summary statistics, but you can easily calculate them using an `apply` function. This is a series of functions that allow you to apply another function over a set of variables or levels. We will use `tapply` here to apply the `mean` function across all levels of the College variable. We could also use the same approach to calculate group standard deviations.

```
tapply(MathDat$Diff, MathDat$College, mean, na.rm=TRUE)
```

```
##       AL      NSM     SSIS 
## 4.950000 5.315789 4.947368
```

```
tapply(MathDat$Diff, MathDat$College, sd, na.rm=TRUE)
```

```
##       AL      NSM     SSIS 
## 5.520059 7.095011 5.104178
```

Linear Regression

Simple Linear Regression

Simple linear regression allows you to test the effect/influence of a single predictor on a single outcome variable. Say we want to predict MathAfter from Age. We will use the function `lm` (for "linear model") to test our hypothesized model. We will also name our models.

```
model2 <- lm(MathAfter~MathScore, data=MathDat)
```

You will notice that when you run the line of syntax above, no results appear in the console. That is because they are stored inside an object called `model2`. Type `model2` into the console and press Enter to see some results.

Let's look at a summary of the model to get parameter estimates, model significance, and the R^2 value for the model:

```
summary(model2)
```

```
##
## Call:
## lm(formula = MathAfter ~ MathScore, data = MathDat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.306  -3.120  -1.148   2.546  14.582
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.99659    5.72833   4.538 3.05e-05 ***
## MathScore    0.73027    0.07328   9.965 5.19e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.303 on 56 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.6394, Adjusted R-squared:  0.633
## F-statistic:  99.3 on 1 and 56 DF,  p-value: 5.194e-14
```

Multiple Linear Regression

More often than not, we would like to predict an outcome from multiple predictors rather than a single predictor. To predict an outcome from multiple predictor variables, you might use multiple regression. We will use the same function (`lm`) to fit this model, adding additional predictors using the + symbol.

```
model3 <- lm(MathAfter ~ MathScore + Age + College, data=MathDat)
summary(model3)
```

```
##
## Call:
## lm(formula = MathAfter ~ MathScore + Age + College, data = MathDat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.538  -3.507  -1.224   2.736  13.983
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 33.34558   12.44282   2.680  0.00979 **
## MathScore    0.72133    0.07694   9.376 7.63e-13 ***
## Age         -0.37322    0.59602  -0.626  0.53389
## CollegeNSM   0.30953    1.75100   0.177  0.86036
## CollegeSSIS  1.17732    1.76169   0.668  0.50685
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.404 on 53 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.6456, Adjusted R-squared:  0.6188
## F-statistic: 24.13 on 4 and 53 DF,  p-value: 2.094e-11
```

Note: College is a categorical predictor. R will automatically create dummy vairables for the analysis and will choose a reference group based on which level of the variable comes first alphabetically. In this case, the college of Arts and Letters (AL) is the reference group. You can tell because no regression coefficient is estimated for that group.

You can add interaction terms to the model by adding a predictor term that is the product of the two (or more) predictors for which you want to test the interaction.

```
model4 <- lm(MathAfter ~ MathScore + Age + College + MathScore*Age, data=MathDat)
summary(model4)
```

```
##
## Call:
## lm(formula = MathAfter ~ MathScore + Age + College + MathScore *
##     Age, data = MathDat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -10.371  -3.369  -1.089   2.736  13.951
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   67.19126   92.75788   0.724    0.472
## MathScore      0.30500    1.13317   0.269    0.789
## Age           -2.14160    4.83937  -0.443    0.660
## CollegeNSM     0.29193    1.76610   0.165    0.869
## CollegeSSIS    1.19554    1.77692   0.673    0.504
## MathScore:Age  0.02174    0.05905   0.368    0.714
##
## Residual standard error: 5.449 on 52 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.6465, Adjusted R-squared:  0.6125
## F-statistic: 19.02 on 5 and 52 DF,  p-value: 1.04e-10
```

And that is all that! See below for a list of resources that may be helpful as you continue your jouRney.

Resources

Websites and online texts

R for Psychological Science by Danielle Navarro (http://compcogscisydney.org/psyr/getting-started.html)

Learning Statistics with R by Danielle Navarro (https://bookdown.org/ekothe/navarro2/)

Cookbook for R (http://www.cookbook-r.com/)

Books

Discovering Statistics Using R by Andy Field (https://us.sagepub.com/en-us/nam/product/discovering%20statistics%20using%20r)

Online Coding Tutorials for R

DataCamp (https://www.datacamp.com/)

Social Media

There is a large R community using Twitter. Follow the #rstats hashtag to keep up with current conversations!