STAT 420/ASRM 450 R Tutorial

Charles Ancel

Fall 2023

We're back, with an official start to our R tutorials! Here, we'll explain some of the basics of R and work through some examples together. You'll also have the opportunity to practice generating your own code by solving some exercises. This should help prepare you for the first homework assignment, which should be released soon if it hasn't already been!

Features of R

You may choose to adjust some of the Appearance Features of R. You can do this by going to Tools > Global Options > Appearance.

You can also load in a dataset that's been downloaded from your computer through a number of methods.

1. Recommended method: place the dataset in the same folder location as your .Rmd file, and the file can be read in using its name only.
2. Type in the (full) appropriate file path for the dataset
3. Pointing and clicking to navigate the file path to the dataset in question. Do this by (a) selecting the Open Folder icon under the Environment tab in the upper right of RStudio, and navigate to the file, or (b) you can use File > Import Dataset > and select the appropriate option (usually From Text (base)...).

Using R as a Calculator

R is a very fancy calculator. We can perform many of the basic arithmetic operations using symbols that you might expect (`+`, `-`, `*`, and `/`). We can also perform additional operations using features like `^`, `e`, `%`, and `*%*`.

```
2+3
```

```
## [1] 5
```

```
4-1
```

```
## [1] 3
```

```
0-2
```

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

```
4*3
```

```
## [1] 12
```

```
3^3
```

```
## [1] 27
```

Often, if we want to perform many of these arithmetic operations, we will use the Console to quickly perform the calculation. Most of the time, we aren't concerned with saving the result.

Creating Variables

Sometimes, we do want to save the result of a calculation. Have you ever used the ANS button on a calculator? Or continued with a calculation using the result of a previous calculation? That's one reason you may want to create a variable in R: to save the results of a calculation for later reuse. Then, instead of typing the answer, which may or may not have infinite decimal points, you can simply use your saved results to calculate a more exact solution.

```
a = 2 * 3
d = a + 4
print(d)
```

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

Hint: R is case-sensitive. That means it matters if you use lower case or upper case letters.

Hint #2: Naming conventions in R can be tricky. Things to avoid include: starting with a number, using spaces, and using the name of a function, another object, or anything else that has meaning in R.

Here, we've seen a single value being assigned (saved) to a single letter. We can use more complex objects for both parts: the name, and what that name represents. We'll continue looking at this more below.

Vectors

Creating Vectors

There are a number of ways to create a vector in R. Perhaps the simplest is using the `c` function. `c` stands for concatenate; in other words, join or merge items together. Remember how we named `a` above, and then created a new variable `d`. Let's modify the original code now:

```
a = c(2, 3)
print(a)
```

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

```
d = a + 4
print(d)
```

```
## [1] 6 7
```

One of the nice things about vectors is that we can easily scale up computations; we did two calculations at once: 2 + 4 and 3 + 4. We could add additional computations, as well. For example, we could multiply `d` by 5:

```
d*5
```

```
## [1] 30 35
```

This `c` function is the most flexible way to generate a vector. You can type any values (they can be numbers or strings of letters), it can be of any length, and the elements just have to be separated by commas. One catch is that all elements must be of the same type.

Quick Question:

What do you think might happen if you perform c(2, "hello")?

Let's try it out:

```
c(2, "hello")
```

```
## [1] "2"     "hello"
```

What changed? What happened?

Going back to creating functions, there are two additional methods discussed above. For instances when you want to create a vector that consists of consecutive integers, you can use the `:` between your first and last desired integer. The `:` is a nice shortcut, and we'll use it often.

Finally, if you want to create a vector that consists of numbers with consistent spacing, you can use the `seq` function. We'll continue working with the `seq` function in the next two examples.

Working With Vectors

There are times that we want to gather certain information from a vector. Maybe we just want one entry, or maybe we want just a couple of entries. We can use square brackets `[]` after the vector name to pull out certain entries. Or maybe we want to know how many entries are in the vector.

We'll use the built in vector `state.name` that consists of the names of the 50 states to demonstrate some of these examples.

```
state.name[1]
```

```
## [1] "Alabama"
```

```
state.name[46:50]
```

```
## [1] "Virginia"      "Washington"    "West Virginia" "Wisconsin"
## [5] "Wyoming"
```

```
length(state.name)
```

```
## [1] 50
```

Now, let's dive a little deeper into the second line of code.

```
46:50
```

```
## [1] 46 47 48 49 50
```

```
state.name[46:50]
```

```
## [1] "Virginia"      "Washington"    "West Virginia" "Wisconsin"
## [5] "Wyoming"
```

We've embedded the 46:50 vector from the first line of code into the second line of code. Here, we've combined a two objects and an operator together in order to print the last five states, alphabetically.

We could make this even more general, with:

```
n = length(state.name)
state.name[(n-4):n]
```

```
## [1] "Virginia"      "Washington"    "West Virginia" "Wisconsin"
## [5] "Wyoming"
```

```
print(n)
```

```
## [1] 50
```

```
print((n-4):n)
```

```
## [1] 46 47 48 49 50
```

For our course, I know that there are multiple approaches and/or methods to achieve the same result. As long as you follow a logical approach and interpret output correctly, I am happy.

Using R Functions

Now, functions are another form of shortcut in R. Some functions are simple and you may be able to guess what they do based on their name. One example: `sqrt`

Other functions have complex or sophisticated code driving them. One example: `lm`.

Anatomy of a Function

Functions have a predictable anatomy (structure). Here's what is required and may be included in a function:

- Functions have a name. You start using a function with the name.
- Function names are followed by parentheses. These parentheses allow you to provide additional information. They also help indicate to R that we are using a function. If you forget the parentheses, you'll see the underlying code returned to you, which likely won't make much sense at first glance.
- Arguments are optional. Many functions do require some arguments. There are a number of ways that arguments can be used.
  - Each argument has a name itself.
  - You assign a value to an argument using `=`.
  - The function writer can assign a default value to an argument. If no other value is provided, then the default is used.
  - You may choose to provide the argument name, or you may omit it. I recommend the argument name in most cases as a beginner using R, as it can help make sure that the output matches what you expect.
    - If you provide the argument name, then you can order the arguments within the function in any order, and the output should be what you expected.
    - If you choose not to provide an argument name, then the values that you provide will be matched to the default order of the arguments. Without care, your results might not be what you expected.

Quick Question:

Where have we seen parentheses used with code in this document? Is it a function?

Some Common Functions:

- mean
- summary
- sd
- IQR
- length
- head
- tail

Getting Help

Sometimes, you encounter a function that you haven't seen before. Or, you'll forget exactly what a function needs and/or how it works. Maybe you received an error message, and you need a little bit of guidance or extra information. Luckily, help is pretty well developed within R. You can always type in the Console ? or ?? followed by the function name, package name, or dataset name (if it's a built-in dataset). You'll be taken to a page that includes information about the object of interest, often including a description, information about usage, arguments, details, and examples. Reading these pages is an art; feel free to skim, try things out, and see if you can get the information from here. The ? and ?? are a shortcut to the Help tab in the bottom right corner.

Alternatively, you may choose to go to sources outside of R if you can't figure it out. There are many resources available through a web engine search, including Stack Overflow. You can also navigate directly to online discussion boards or campuswire, if needed. My recommendation: include screenshots or a full example, so that we can help you diagnose what's going on.

Try It! Exercises

(1)

We've talked about the `seq` function, but we haven't seen an example yet. I'll have you create the example now. Playing around, see if you can create a sequence that goes from 0 to 1 by 0.1. Be sure to look at the vector you've created to confirm that you created it correctly. **Challenge:** See if you can create the same vector using a different method.

```
seq(0,1, 0.1)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
c(0,1,2,3,4,5,6,7,8,9,10)/10
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

(2)

Assign the vector you just created to a letter; you can choose any letter other than c, which is used for the concatenate function. See if you can calculate numerical summaries to represent this vector.

```
v10 = seq(0,1,0.1)
length(v10)
```

```
## [1] 11
```

```
mean(v10)
```

```
## [1] 0.5
```

```
median(v10)
```

```
## [1] 0.5
```

(3)

Look into the `log` and `exp` functions. If you use the default `log`, what base is being used? What about the `exp` function?

Answer:Both use base10.

(4)

Apply the `log` function to your original vector created in **(2)**. What does your output look like?

```
print(log(v10))
```

```
##  [1]       -Inf -2.3025851 -1.6094379 -1.2039728 -0.9162907 -0.6931472
##  [7] -0.5108256 -0.3566749 -0.2231436 -0.1053605  0.0000000
```

```
print(exp(v10))
```

```
##  [1] 1.000000 1.105171 1.221403 1.349859 1.491825 1.648721 1.822119 2.013753
##  [9] 2.225541 2.459603 2.718282
```

(5)

Predict what the following code will do. Then check if you were correct:

```
state.name[16:19]
```

```
## [1] "Kansas"    "Kentucky"  "Louisiana" "Maine"
```

```
state.name[-16]
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "Arkansas"
##  [5] "California"     "Colorado"       "Connecticut"    "Delaware"
##  [9] "Florida"        "Georgia"        "Hawaii"         "Idaho"
## [13] "Illinois"       "Indiana"        "Iowa"           "Kentucky"
## [17] "Louisiana"      "Maine"          "Maryland"       "Massachusetts"
## [21] "Michigan"       "Minnesota"      "Mississippi"    "Missouri"
## [25] "Montana"        "Nebraska"       "Nevada"         "New Hampshire"
## [29] "New Jersey"     "New Mexico"     "New York"       "North Carolina"
## [33] "North Dakota"   "Ohio"           "Oklahoma"       "Oregon"
## [37] "Pennsylvania"   "Rhode Island"   "South Carolina" "South Dakota"
## [41] "Tennessee"      "Texas"          "Utah"           "Vermont"
## [45] "Virginia"       "Washington"     "West Virginia"  "Wisconsin"
## [49] "Wyoming"
```

Answer: Return states 16 thru 19.

Reading in Data

While we've been focusing on some built-in datasets for this tutorial (and this week's homework), we'll often want to read in our data from an outside source, like the coasters dataset.

For our course, the first step is to download the data to your computer. Then, place the data file in the same directory (file folder) where our Rmd file lives, and you'll be able to read in the data using the dataset name, as we've previously seen.

There are alternative ways to read in data if you know the full path name to the data file in RMarkdown; you would need to update the file path if you ever move the file on your computer. There are a few point-and-click methods of reading in data, but these do not work with RMarkdown. Finally, there are methods that can be used to read in data directly when it is posted on the internet, although we won't focus on those methods in our course.

Working With Data Frames

Data Frames are a special way to save data. They consist of columns that are vectors. The rows also line up, so that each entry represents a specific unit.

We'll typically use data frames in our course. There are a number of nice features about data frames, including the types of functions that we can apply to the data frame.

Let's check out a few things that we can do with the coasters dataset.

```
## Windows Code File Path
## coasters = read.delim("C:/Users/jdeeke/Downloads/coasters-2015.txt")

## Mac Code File Path
## coasters = read.delim('~/Downloads/coasters-2015.txt')
```

```
# View(coasters) # View won't work with R Markdown typically,
# because the additional window isn't included in the final report.
# head(coasters)
# dim(coasters)
# ncol(coasters)
# nrow(coasters)
# names(coasters)
# colnames(coasters)
# head(coasters$Duration)
# summary(coasters$Duration)
# coasters[1,8]
# coasters[1,]
# summary(coasters[,8])
# summary(coasters)
```

What do you think each of these values represents?

My goal is that between the function name, output, and help file, you can infer what each of these functions is doing. The naming of many of these functions are in fact informative once you've gotten to know R a little better.

Hint: You can use the format dataset_name$variable_name rather than numbers that represent the location (order) of the variable in the dataset.

Data Frame Try It! Exercises

We'll apply these tutorials to the built-in `swiss` dataset.

(1)

Where can you go to find out more information about the `swiss` dataset?

Answer: Type ?swiss into the console.

(2)

Calculate the mean, standard deviation, and the five number summary for the percent of the population that was educated in the provinces.

```
mean(swiss[,4])
```

```
## [1] 10.97872
```

```
sd(swiss[,4])
```

```
## [1] 9.615407
```

```
summary(swiss[,4])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    6.00    8.00   10.98   12.00   53.00
```

(3)

A z-score is a way of standardizing variables so that values from one variable can be compared to the values from another variable. To calculate the z-score, subtract the mean from the observation, then divide by the standard deviation. Calculate the z-score for the percent of the population that was educated. Then, calculate

the mean, standard deviation, and five number summary for the standardized proportion of educated people in the provinces.

```
zscore= (swiss$Education - mean(swiss[,4]))/sd(swiss[,4])
summary(zscore)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.0378 -0.5178 -0.3098  0.0000  0.1062  4.3702
```

```
sd(zscore)
```

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

(4)

Compare the summary values from Exercises 2 & 3.

Answer:

(5)

Create a new vector that includes the even values between 1 and 47. Using that new vector, create a vector that includes only the even entries (rows) of the education variable. For the new set of 23 observations, calculate the five number summary. Compare this to the values from Exercise 2.

```
even=seq(2,47,2)
evenEd=swiss$Education[even]
summary(evenEd)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    7.00    9.00   11.09   12.00   32.00
```

```
sd(evenEd)
```

```
## [1] 7.896565
```

(6): Challenge

Calculate the proportion of those not involved in either Agriculture or Education in each province (Other pursuits). Find the minimum and maximum values for those involved in Other pursuits.

```
other = 100-swiss$Agriculture-swiss$Education
min(other)
```

```
## [1] 8.3
```

```
max(other)
```

```
## [1] 81.3
```

(7)

First, predict what the following chunk of code will do.

Prediction:

```
mysample = sample(1:nrow(swiss), 20)
mean(swiss$Education[mysample])
```

```
## [1] 8.6
```

```r
mean(swiss$Education[-mysample])
```

```
## [1] 12.74074
```

```r
sd(swiss$Education[mysample]) < sd(swiss$Education[-mysample])
```

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

Then, run the code. Did your prediction match the output? If not, think about how the results differ from your prediction. Can you determine why the results turned out this way?

Answer:

More Data Frame Operations

We briefly talked about some of the functions that you can apply to a data frame in R.

In my opinion, one of the most used functions is the `dim` function. It helps me get a quick sense of the size of a data frame. You can also use it to check that a data frame has been loaded correctly. We'll be using the built-in `swiss` data frame for most of the remaining tutorials, so let's first check out its size (dimensions).

```r
dim(swiss)
```

```
## [1] 47  6
```

There are a few other functions to review that we haven't talked about yet.

```r
min(swiss$Catholic)
```

```
## [1] 2.15
```

```r
max(swiss$Catholic)
```

```
## [1] 100
```

Can you guess what the min and max function are calculating?

Now suppose I want to calculate an additional variable and incorporate it into the original dataset – how much larger the proportion educated is compared to the proportion receiving highest marks on an army examination. I could go ahead and calculate the difference:

```r
swiss$Education - swiss$Examination
```

```
##  [1]  -3   3   0  -5  -2  -2  -9  -6  -5  -3  -8  -9  -7  -7 -17 -16  -9   2 -11
## [20] -10 -12 -11 -10 -14 -11  -6  -3  -6  -6  -7  -1  -1  -3  -6  -4   0   0   0
## [39] -14 -18  -9  -3  -8 -18  16  13   7
```

The problem is that now I have an extra 47 values floating around. It can be helpful to attach this newly created variable to the original dataset. We can do this using the function `cbind` (column bind).

Let's see what happens when we use the `cbind` function as we might expect it:

```r
cbind(swiss, swiss$Education - swiss$Examination)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
## Broye             83.8        70.2          16         7    92.85
```

```
## Glane             92.4        67.8          14         8    97.16
## Gruyere           82.4        53.3          12         7    97.67
## Sarine            82.9        45.2          16        13    91.38
## Veveyse           87.1        64.5          14         6    98.61
## Aigle             64.1        62.0          21        12     8.52
## Aubonne           66.9        67.5          14         7     2.27
## Avenches          68.9        60.7          19        12     4.43
## Cossonay          61.7        69.3          22         5     2.82
## Echallens         68.3        72.6          18         2    24.20
## Grandson          71.7        34.0          17         8     3.30
## Lausanne          55.7        19.4          26        28    12.11
## La Vallee         54.3        15.2          31        20     2.15
## Lavaux            65.1        73.0          19         9     2.84
## Morges            65.5        59.8          22        10     5.23
## Moudon            65.0        55.1          14         3     4.52
## Nyone             56.6        50.9          22        12    15.14
## Orbe              57.4        54.1          20         6     4.20
## Oron              72.5        71.2          12         1     2.40
## Payerne           74.2        58.1          14         8     5.23
## Paysd'enhaut      72.0        63.5           6         3     2.56
## Rolle             60.5        60.8          16        10     7.72
## Vevey             58.3        26.8          25        19    18.46
## Yverdon           65.4        49.5          15         8     6.10
## Conthey           75.5        85.9           3         2    99.71
## Entremont         69.3        84.9           7         6    99.68
## Herens            77.3        89.7           5         2   100.00
## Martigwy          70.5        78.2          12         6    98.96
## Monthey           79.4        64.9           7         3    98.22
## St Maurice        65.0        75.9           9         9    99.06
## Sierre            92.2        84.6           3         3    99.46
## Sion              79.3        63.1          13        13    96.83
## Boudry            70.4        38.4          26        12     5.62
## La Chauxdfnd      65.7         7.7          29        11    13.79
## Le Locle          72.7        16.7          22        13    11.22
## Neuchatel         64.4        17.6          35        32    16.92
## Val de Ruz        77.6        37.6          15         7     4.97
## ValdeTravers      67.6        18.7          25         7     8.65
## V. De Geneve      35.0         1.2          37        53    42.34
## Rive Droite       44.7        46.6          16        29    50.43
## Rive Gauche       42.8        27.7          22        29    58.33
##              Infant.Mortality swiss$Education - swiss$Examination
## Courtelary               22.2                                 -3
## Delemont                 22.2                                  3
## Franches-Mnt             20.2                                  0
## Moutier                  20.3                                 -5
## Neuveville               20.6                                 -2
## Porrentruy               26.6                                 -2
## Broye                    23.6                                 -9
## Glane                    24.9                                 -6
## Gruyere                  21.0                                 -5
## Sarine                   24.4                                 -3
## Veveyse                  24.5                                 -8
## Aigle                    16.5                                 -9
## Aubonne                  19.1                                 -7
```

```
## Avenches                22.7                                -7
## Cossonay                18.7                               -17
## Echallens               21.2                               -16
## Grandson                20.0                                -9
## Lausanne                20.2                                 2
## La Vallee               10.8                               -11
## Lavaux                  20.0                               -10
## Morges                  18.0                               -12
## Moudon                  22.4                               -11
## Nyone                   16.7                               -10
## Orbe                    15.3                               -14
## Oron                    21.0                               -11
## Payerne                 23.8                                -6
## Paysd'enhaut            18.0                                -3
## Rolle                   16.3                                -6
## Vevey                   20.9                                -6
## Yverdon                 22.5                                -7
## Conthey                 15.1                                -1
## Entremont               19.8                                -1
## Herens                  18.3                                -3
## Martigwy                19.4                                -6
## Monthey                 20.2                                -4
## St Maurice              17.8                                 0
## Sierre                  16.3                                 0
## Sion                    18.1                                 0
## Boudry                  20.3                               -14
## La Chauxdfnd            20.5                               -18
## Le Locle                18.9                                -9
## Neuchatel               23.0                                -3
## Val de Ruz              20.0                                -8
## ValdeTravers            19.5                               -18
## V. De Geneve            18.0                                16
## Rive Droite             18.2                                13
## Rive Gauche             19.3                                 7
```

Well, we added the variable to the dataset, but the values were all printed out. We can confirm that we did in fact add the value to the dataset:

```
dim(swiss)
```

```
## [1] 47  6
```

Oh no! We have the same dimensions that we did before. Note that we also printed the values in the code chunk above. Do you have any guesses as to what the problem might be?

One of the issues here is that we haven't saved the results to a new data frame. We can adjust that below.

```
swiss1 = cbind(swiss, undereducated = swiss$Education - swiss$Examination)
```

Now, similar to cbind, there does also exist a function **rbind**. This can be used to add a new row (observation) to an existing data frame.

One additional way to extend a data frame is to do so directly. Here's an example below:

```
swiss1$InfToFert = swiss1$Infant.Mortality/swiss1$Fertility
```

We can confirm that the variables were added correctly by checking out the dimensions, but one thing that we might miss is any incorrect calculations that may have occurred. We may choose instead to preview the

data frame, to make sure that everything was calculated correctly.

```
head(swiss1)
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
##              Infant.Mortality undereducated InfToFert
## Courtelary               22.2            -3 0.2768080
## Delemont                 22.2             3 0.2671480
## Franches-Mnt             20.2             0 0.2183784
## Moutier                  20.3            -5 0.2365967
## Neuveville               20.6            -2 0.2678804
## Porrentruy               26.6            -2 0.3495401
```

To answer some questions that may have come up, or to point out some things that I included below:

- I chose to save my new data frame as `swiss1`. Since `swiss` is a built-in data frame, I wanted to make sure that I didn't overwrite it. To do that, I want to create a new data frame that I can play with. There are a number of ways that you could adjust the name of the data frame, and adding a number is just one way
- I named my new variables. I **named them** and tried to pick a name that explained what the variable contains. There are no spaces in the names, although periods and underscores are allowed. You can also see capitalization maintained in each of these variables.
- The $ operator can be so helpful with data frames, so that we don't have to count which variable is in which position.
- We've also seen how vector operations can be especially helpful in these contexts.
- I do prefer to view the data regularly, just to make sure that I've done everything correctly.
- Building in some of the "sanity checks" to your analysis process can be very helpful. Notice that we paused a number of times to make sure that we actually had what we expected before moving on to the next piece of the analysis.
- When submitting your own document, please do NOT submit the data frame printed. `swiss` is pretty small, and it still takes about 2 pages of printout. Generally, we know what the data frame has. Printing the `head` is better, but even that could be excluded for submission (if the question doesn't ask for that).

Using R Markdown

R Markdown consists of a number of features. This is a special type of file that allows you to generate reports that includes code, output, and text in a neat document.

Features/anatomy of R Markdown include:

- chunks (the grey, highlighted components that represent code to be run)
  - special options for the chunks, including chunk names
  - whether to show the code itself
  - whether to show the output
  - whether to print any errors
- formatting of text, like:
  - headers (indicated using #)
  - italics (indicated by a single asterisk at the beginning and the end of the italicized text)
  - bold (indicated by a double asterisk at the beginning and the end of the bolded text)

You can either add all of these features manually, or you can use shortcuts. For example, you can select the green box with a c and a plus icon at the top of the Editor window to enter a new chunk.

You can run the code contained in the chunk in your Console (your current version of R) using the green arrow on the right side of the chunk. This allows you to preview the results of your code, and make adjustments as needed. Depending on the code, the output will appear either below the code chunk (as a preview) or in the Console. Be careful, because this can sometimes create issues when you go to look at the final report. The final report is created in a fresh workspace, so you may have errors if your code is dependent on anything that is written in your Console only but doesn't appear in the workspace of your Markdown document.

You can find additional information about formatting, keyboard shortcuts, and other options by searching for R Markdown cheat sheets on the internet.

Tips for working with R Markdown

- **Knit often**. This will help you avoid some of the common errors that occur when working with R Markdown files.
- **Don't be afraid to experiment**. Experimenting is how you figure out how things work. We all forget sometimes, and you might find something new in the process!
- **Don't be afraid of mistakes**. Everyone will make mistakes, and you won't break the code or your Markdown file. Hopefully you'll be able to learn something by working through any mistakes that come up.
- **Use the R code that's available to you as a model**. Figure out how you might be able to modify it for your specific example, based on how it's been used in its previous context.
- **Try to read any error messages that show up**. While understanding all of the details they provide is challenging for even the most experienced programmer, you might get a hint of what the issues is or where it's located. All else fails, take a screenshot of it and head to campuswire or office hours for help fixing it.

Installing Packages

Packages are special "packages" of code. These packages allow you to use specifically pre-designed and prepared functions, access certain datasets, and perform analyses more easily. The packages that are maintained through CRAN, which all that we use for this course are, also contain helpful Help files that guide you through using tools within the package.

The first time that you use a package, you'll need to run the following code, replacing the package_name with your package of interest:

install.packages("package_name").

This is equivalent to buying a book and putting it on your bookshelf. You now "own" that package, or have a copy of it available to you.

Everytime that you want to use that package, you need to run the following code in the session:

library(package_name).

This means that you'll need to include it in every R Markdown file where you want to use that package, because the R Markdown file creates its own fresh environment every time. Anytime that you close and restart R, you'll also need to run it in the console (if you are going to use a part of the package in the Console).

This is equivalent to opening the book up on your desk and allowing you to reference anything within it.

We did this on the first day with `ggplot2`. See if you can go back and identify this.

Graphs

One of the most powerful tools in data analysis is graphing. We saw a little bit of that in the first day's class. We'll now return to it, learning `ggplot` more formally.

Among the reasons that we'll use `ggplot` in this course is that the formatting is consistent, regardless of the type of graph that you want to make. At a minimum, your plot will include:

```
ggplot (data, mapping = aes(variables)) +     geom_plottype()
```

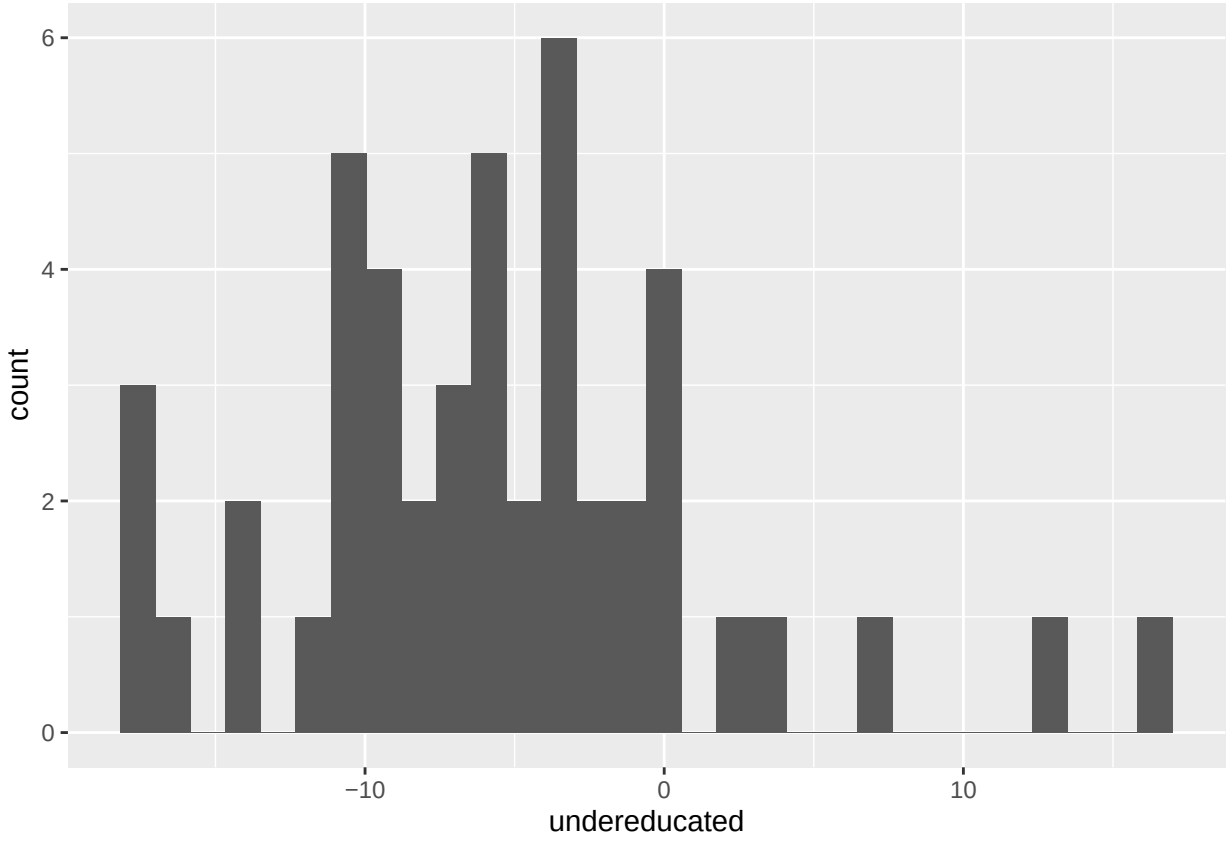
You'll need to replace the data, variables, and plottype in the code above to suit your needs. Let's start by looking at some examples. Before we can actually generate any graphs, we will need to load the `ggplot2` package into our library. Generally, best practice is to load all libraries needed for that document at the top of the document. After this tutorial, we'll switch to that style. Anytime you are generating ggplot graphs, you will need to load the `ggplot2` library in that document.

```
library(ggplot2)
```

Let's start by looking at some of the graphs for one variable. We'll look at our newly created undereducated variable.

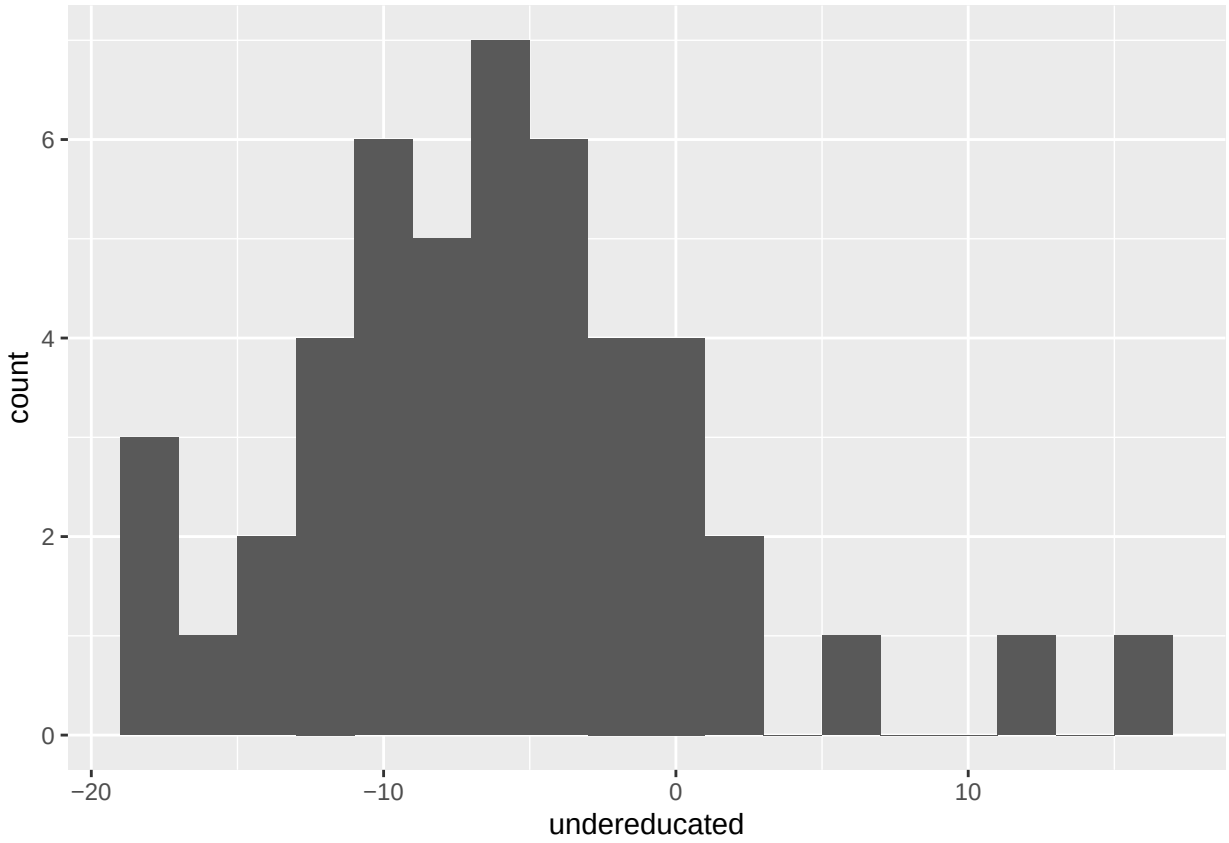
```
ggplot(swiss1, aes(x = undereducated)) +
  geom_histogram()
```

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



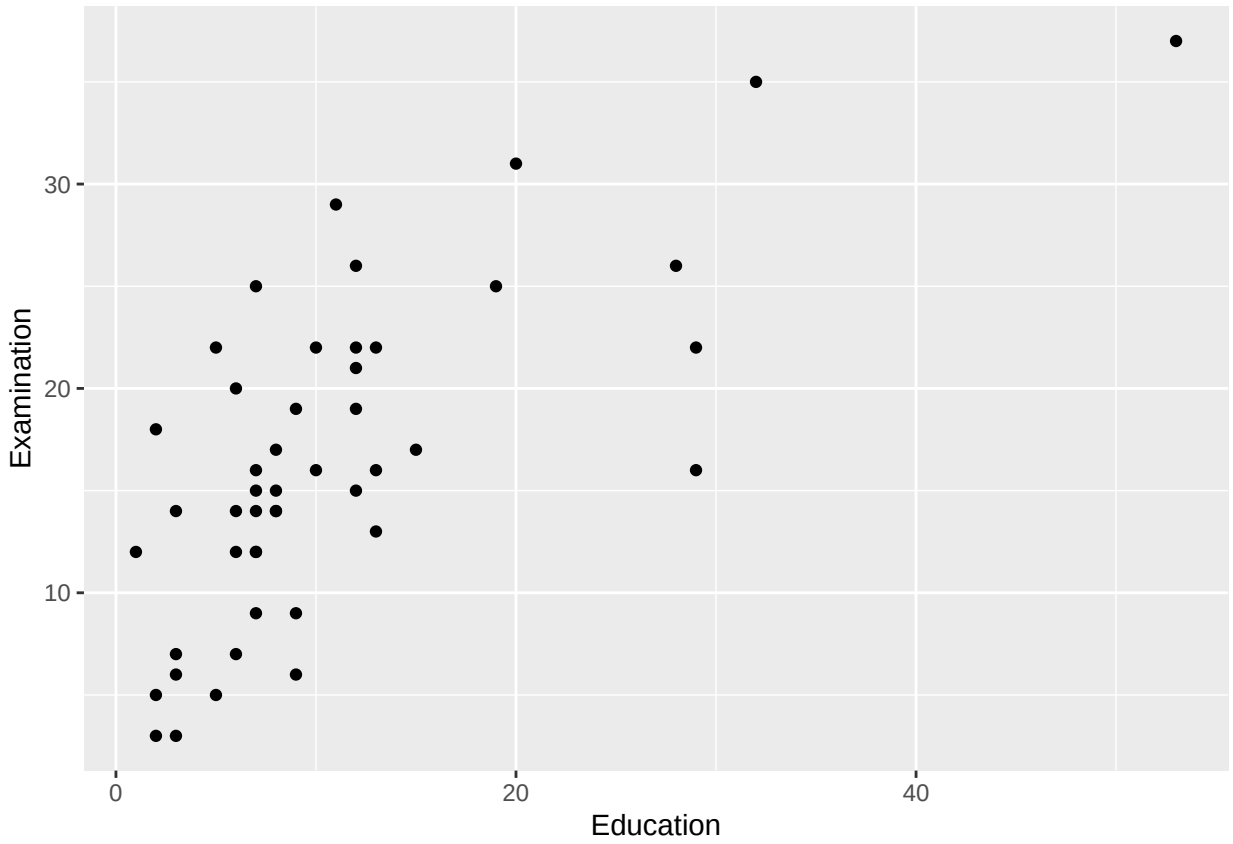
Above you see one basic graph for a single variable. We are given a warning message (not an error message). Because a graph was created, this is fine. If we wanted to bypass the error message, we could specify the binwidth (the length of the bar), as shown below.

```
ggplot(swiss1, aes(x = undereducated)) +
  geom_histogram(binwidth = 2)
```



What if we have multiple variables that we are interested in? We can change our graph type based on the type and number of variables to include. Below, we'll look at a scatterplot:

```
ggplot(swiss1, aes(x = Education, y = Examination)) +
  geom_point()
```



Now that we've seen some graphs in action, let's try them out.

First Graphs Try It! Exercises

(1)

Make a histogram of the InfToFert variable. What can you learn from this histogram?

```
# Use this code chunk to compute your solution.
```

Answer:

(2)

Create a scatterplot of the Infant Mortality and the Fertility Measure for the provinces in Switzerland.

```
# Use this code chunk to compute your solution.
```

(3)

What else would you want to include in these graphs? Stylistically, is there anything you would change or adjust?

Answer:

Customizing Graphs

Before we start this section, I want to add one more variable to the `swiss1` data frame that measures whether a majority of the province is Catholic.

```
swiss1$MostCatholic = swiss1$Catholic > 50
```

We'll return to this above line of code later.

There are a number of ways to customize graphs. We saw a few of them last week. For example, we can add color, we can add lines, and we can adjust titles. Check out the ggplot Cheat Sheet that you can find online (through R) for just a small number of these options.

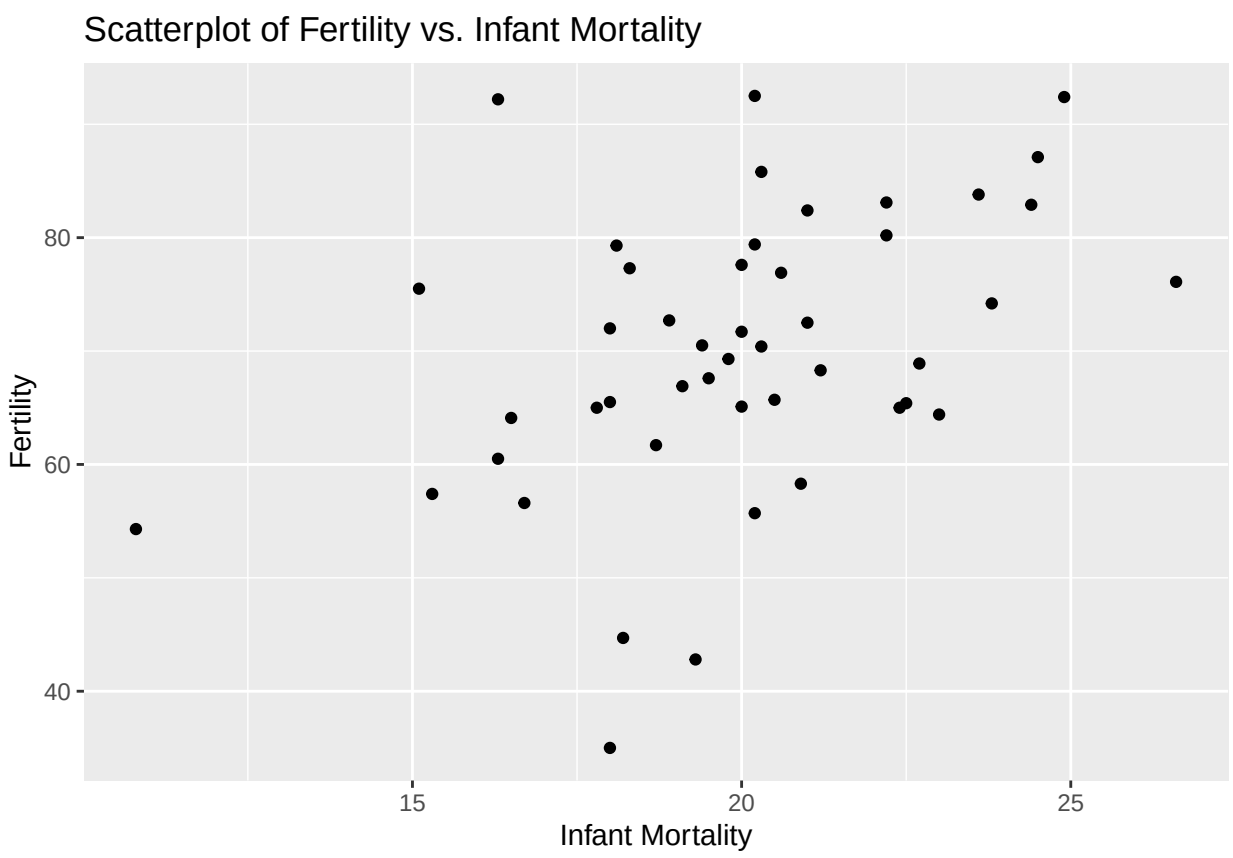
We'll focus on customizing graphs in a few ways in this class.

Labels

First, we can add better labels to a graph with the `labs` layer.

Let's apply this layer to the last plot that we created.

```
ggplot(swiss1, mapping = aes(y = Fertility, x = Infant.Mortality)) +
  geom_point() +
  labs(title = 'Scatterplot of Fertility vs. Infant Mortality', x = 'Infant Mortality', y = 'Fertility')
```

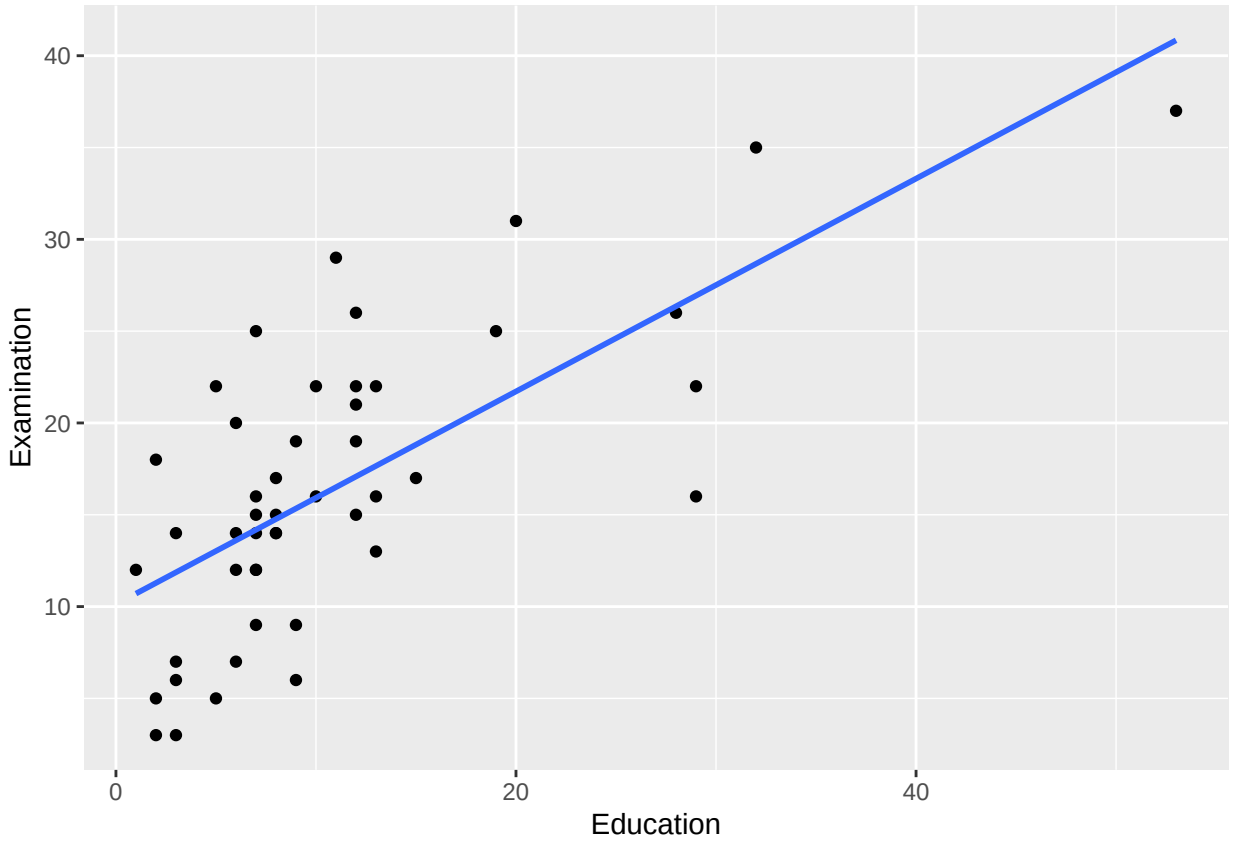


We were able to remove the period in the middle of the Infant Mortality title. We didn't *need* to supply the y-axis label, as "Fertility" was already used by default. It didn't hurt to supply it either.

Adding Lines to Scatterplots

R has the ability to include summary lines with a scatterplot. Below is an example:

```
ggplot(swiss1, aes(x = Education, y = Examination)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, formula = y~x)
```



We'll use only these arguments for the `geom_smooth` function, at least for the first part of the semester.

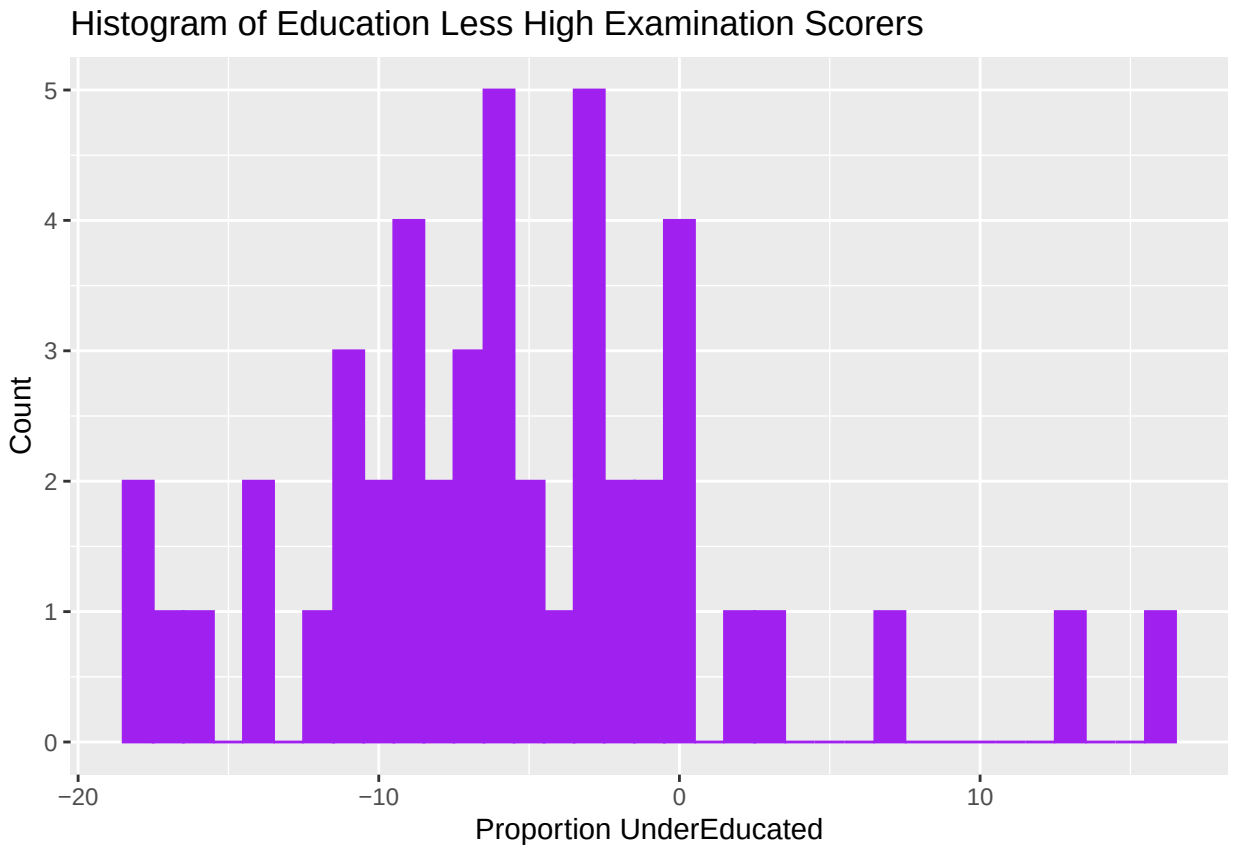
The `geom_smooth` function is another layer that we've added to the graph and is responsible for adding the line to the visualization.

Colors and Shapes

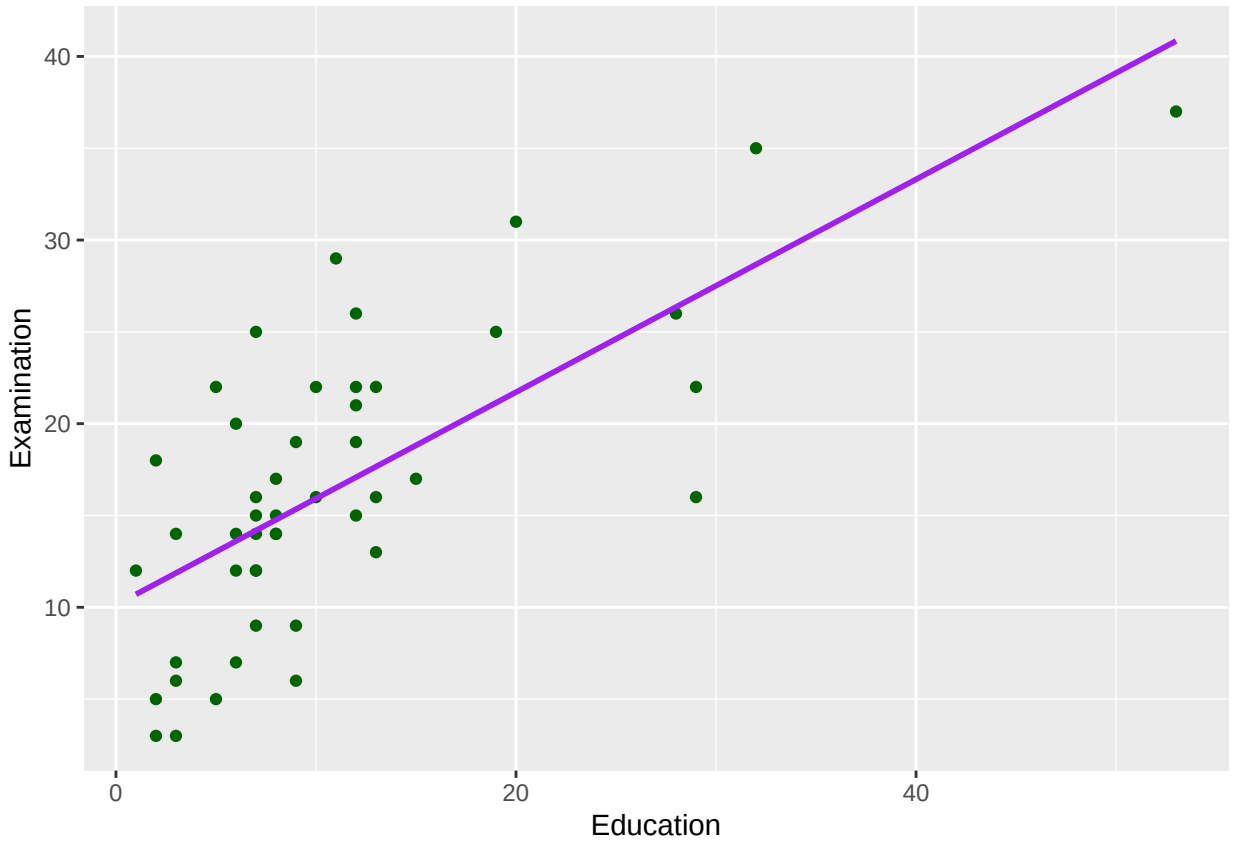
For Visual Appeal

Another customization is adding colors and/or shapes. First, we may be interested in adding some color to a graph, to make it more visually appealing. If it doesn't have to do with any variable, we may put the `color` argument either into the overall `ggplot` information or in the specific layer where we would like the color to appear if we don't want the color used throughout all areas of the plot.

```
ggplot(swiss1, aes(x = undereducated)) +
  geom_histogram(color = "purple", fill = "purple", binwidth = 1) +
  labs(x = "Proportion UnderEducated", y = "Count", title = "Histogram of Education Less High Examinati
```



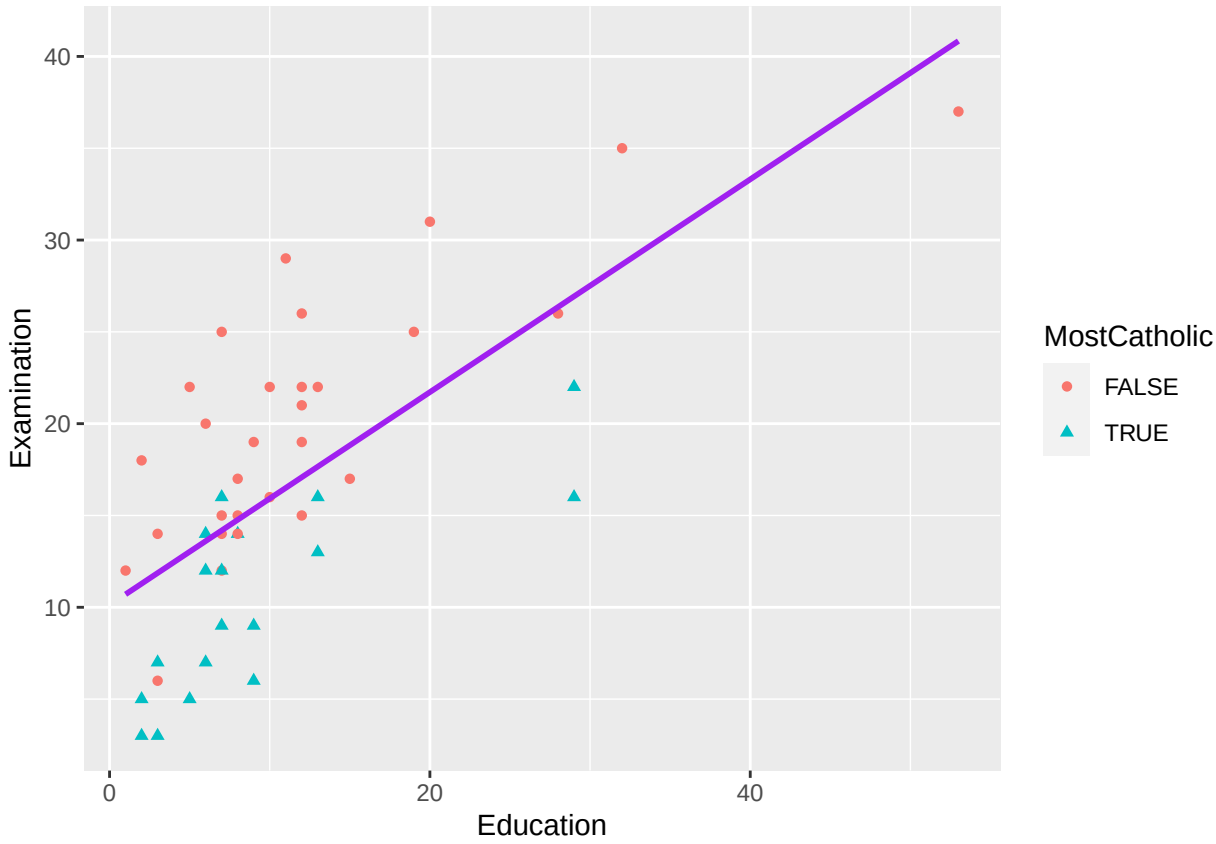

```
ggplot(swiss1, aes(x = Education, y = Examination)) +
  geom_point(color = 'dark green') +
  geom_smooth(method = 'lm', se = F, color = 'purple', formula = 'y ~ x')
```



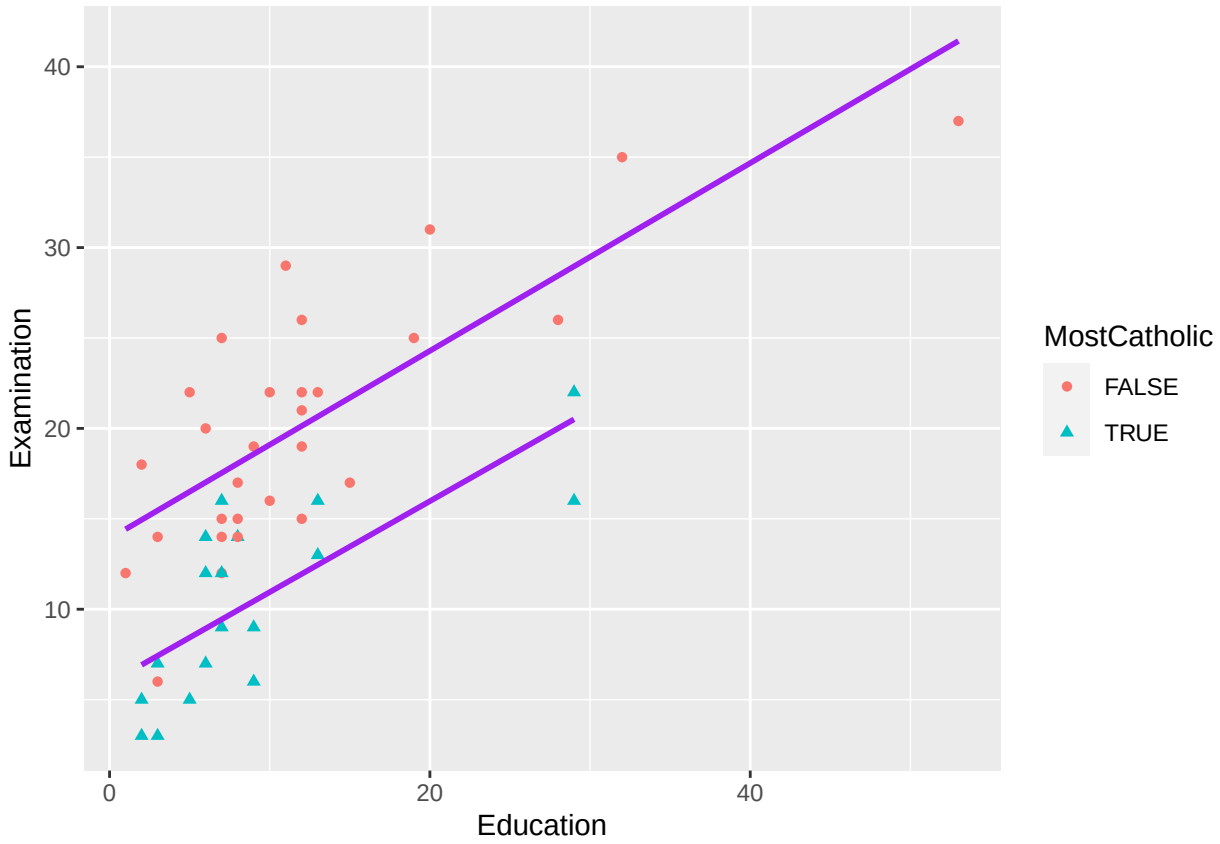
For a Variable in a Scatterplot

If we want the colors to correspond to values of a variable, we'll want to indicate this by specifying the color within an `aes` mapping function. That is, we will use the argument `color = variable_name` within the `aes` function to indicate the graph should be colored based on the variable indicated. If we want the coloring to apply to all layers throughout the whole plot, we'll include it in the original `aes` mapping in the `ggplot` function. If we want the coloring only to appear in one layer, we can include it in that specific layer's `geom` function, with the `aes` argument. Below are a few examples:

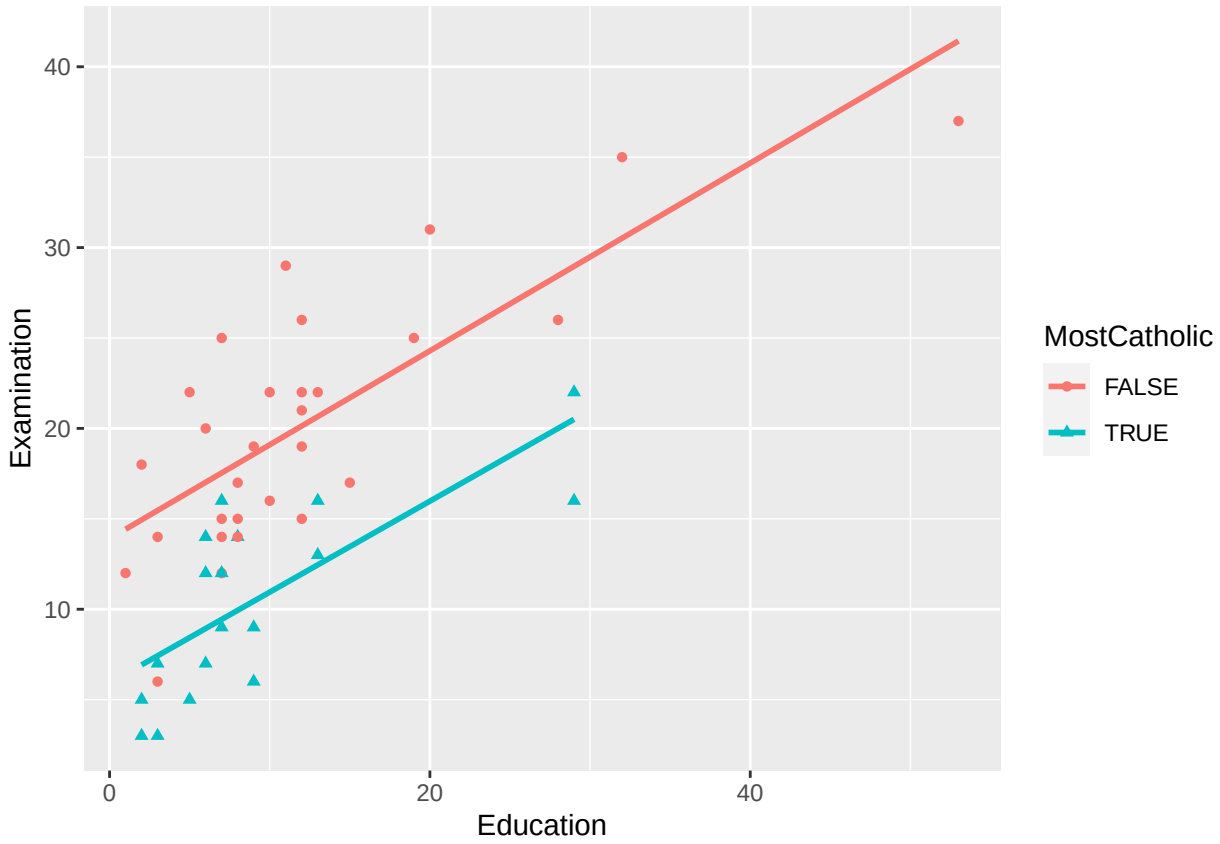
```
ggplot(swiss1, aes(x = Education, y = Examination)) +
  geom_point(aes(color = MostCatholic, shape = MostCatholic)) +
  geom_smooth(method = 'lm', se = F, color = 'purple', formula = y ~ x)
```




```
ggplot(swiss1, aes(x = Education, y = Examination, color = MostCatholic, shape = MostCatholic)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, color = 'purple', formula = y ~ x)
```



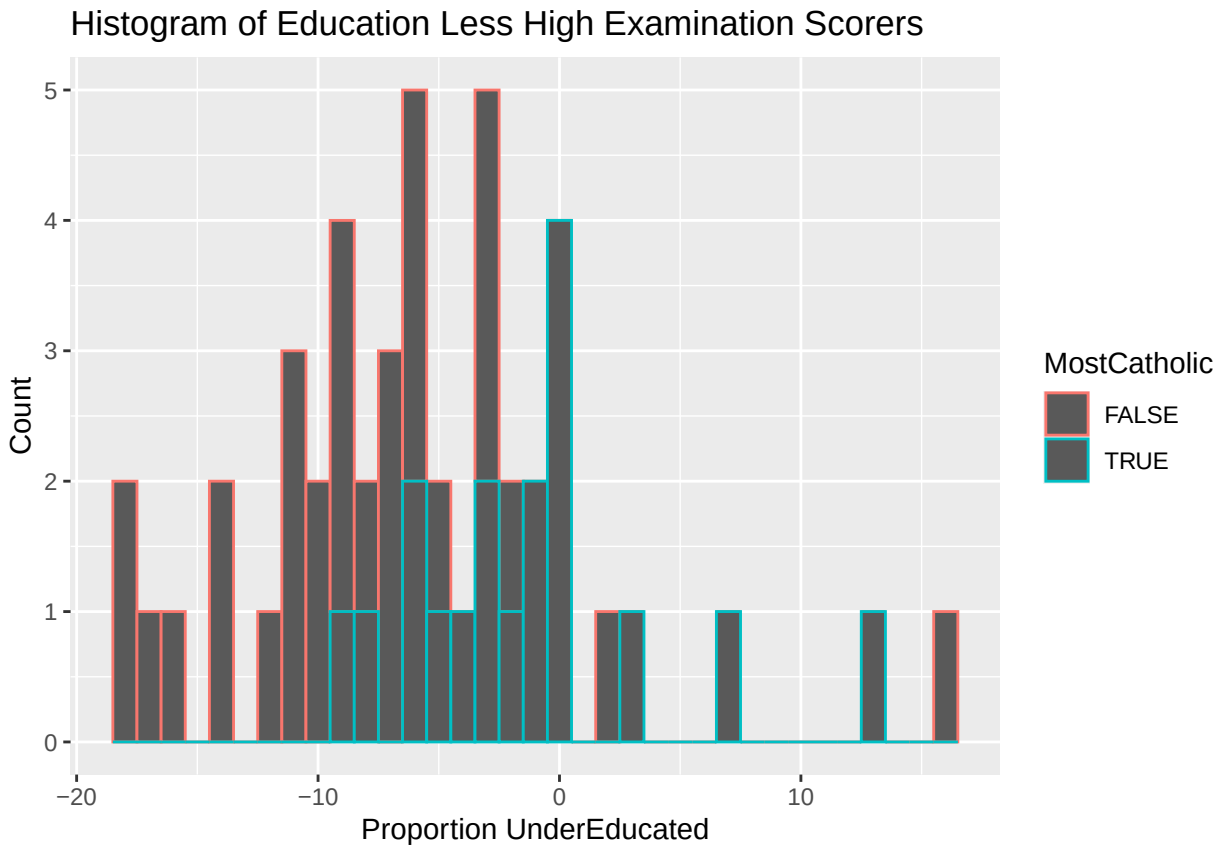

```
ggplot(swiss1, aes(x = Education, y = Examination, color = MostCatholic, shape = MostCatholic)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, formula = y~x)
```



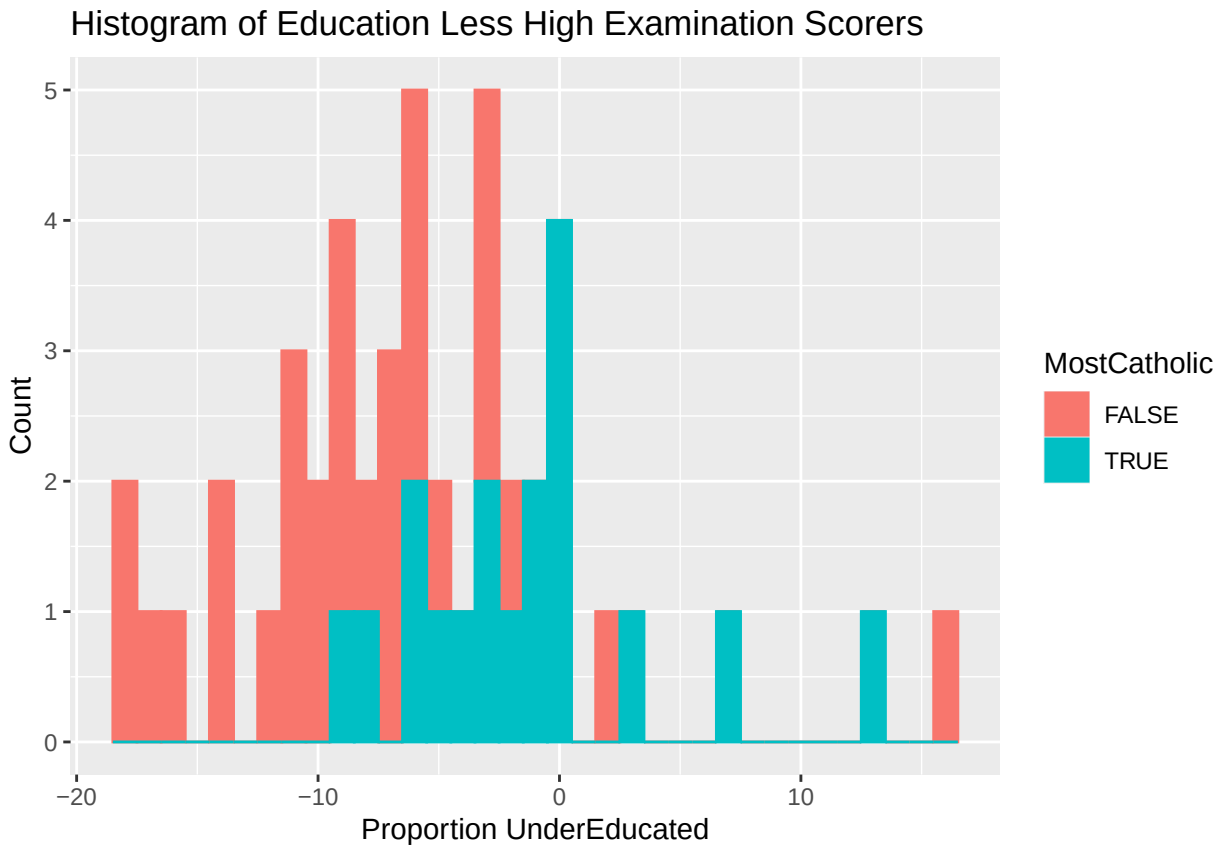
How do these three graphs differ? Why do you think they are different?

For a Variable in a Histogram

```
ggplot(swiss1, aes(x = undereducated, color = MostCatholic)) +
  geom_histogram(binwidth = 1) +
  labs(x = "Proportion UnderEducated", y = "Count",
       title = "Histogram of Education Less High Examination Scorers")
```




```
ggplot(swiss1, aes(x = undereducated, color = MostCatholic, fill = MostCatholic)) +
  geom_histogram(binwidth = 1) +
  labs(x = "Proportion UnderEducated", y = "Count",
       title = "Histogram of Education Less High Examination Scorers")
```



As seen above, the same principles for colors and shapes can be applied to both histograms and scatterplots.

Graphing Customization Try It! Exercise

Try adding the variable MostCatholic to the scatterplot you created in the last set of exercises. Include new labels for this plot, as well.

```
# Use this code chunk to compute your solution.
```

Logical Statements

A simple way to perform complex analyses is to use logical statements. These are statements that evaluate to either `True` or `False`.

Note: R recognizes `True`, `TRUE`, and `T` as representing true, and similar formats for false.

There are many operators that produce logical statements. Here are a few of the most common:

- `>` for greater than
- `<` for less than
- `>=` for greater than or equal to
- `<=` for less than or equal to
- `==` for exactly equal to
- `!` for not, as in != (not equal to) or !x (not x, where x could be a number, name, or other format)
- `|` for or, as in x | y
- `&` for and, as in x & y
- `isTRUE(x)` for evaluating if x is T

We did see this above:

```
swiss1$MostCatholic = swiss1$Catholic > 50
```

Once we have the logical statement created, we can apply them to either analysis or to data organization (as we'll see in the next section). Let's focus on creating and analyzing logical statements now.

Suppose we want to know more about the undereducated variable from before.

```
swiss1$undereducated > 0
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

Alternatively, undereducated was calculated as the proportion education minus the proportion who received highest marks. We could look at those variables directly.

```
swiss1$Education > swiss1$Examination
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

If the proportion who is educated is higher than the proportion who receive highest marks, then that is a sign that those who are educated have not been sufficiently educated. The population in general then seems to be undereducated as measured by the examination results.

It looks like the proportion educated is generally less than the proportion who scored well on exams. But maybe this is due to a large number of 0's. Let's look at if the two are equal to each other:

```
swiss1$Education == swiss1$Examination
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [37]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

For a dataset of this size, we can easily look at the logical statement values by hand. Even for this dataset and definitely for larger datasets, we may want to summarize the logical variable. For the purposes of analysis, R will replace `True` with the value 1 and `False` with the value 0. This allows for some very elegant simplifications and analysis.

Let's check it out:

```
sum(swiss1$undereducated > 0)
```

```
## [1] 5
```

```
sum(swiss1$undereducated == 0)
```

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

```
sum(swiss1$undereducated < 0)
```

```
## [1] 38
```

```
mean(swiss1$undereducated > 0)
```

```
## [1] 0.106383
```

Finally, `which` can also be a useful function. `which` will return the element (or location) that have `True` results for a logical statement.

```
which(swiss1$undereducated > 0)
```

```
## [1]  2 18 45 46 47
```

Subsetting Data

Previously, we talked about picking out only certain elements of a vector. We can perform similar analyses for a data frame. There are many ways to do so.

First, we could use the same square brackets that we used for vectors, like so:

```
swiss1[2, 4]
```

```
## [1] 9
```

We might want to pull out more than one entry, so we could use a vector of locations instead:

```
swiss1[2:4, 3:5]
```

```
##              Examination Education Catholic
## Delemont               6         9    84.84
## Franches-Mnt           5         5    93.40
## Moutier               12         7    33.77
```

Or we could pull out an entire row or an entire column:

```
swiss1[2,]
```

```
##          Fertility Agriculture Examination Education Catholic Infant.Mortality
## Delemont      83.1        45.1           6         9    84.84             22.2
##          undereducated InfToFert MostCatholic
## Delemont             3  0.267148         TRUE
```

```
swiss1[,4]
```

```
##  [1] 12  9  5  7 15  7  7  8  7 13  6 12  7 12  5  2  8 28 20  9 10  3 12  6  1
## [26]  8  3 10 19  8  2  6  2  6  3  9  3 13 12 11 13 32  7  7 53 29 29
```

These all work fine, but sometimes it can be helpful to pull out only specific rows. Say we're interested in those provinces that are undereducated. We can pull out just those provinces:

```
swiss1[which(swiss1$undereducated > 0),]
```

```
##              Fertility Agriculture Examination Education Catholic
## Delemont          83.1        45.1           6         9    84.84
## Lausanne          55.7        19.4          26        28    12.11
## V. De Geneve      35.0         1.2          37        53    42.34
## Rive Droite       44.7        46.6          16        29    50.43
## Rive Gauche       42.8        27.7          22        29    58.33
##              Infant.Mortality undereducated InfToFert MostCatholic
## Delemont                 22.2             3 0.2671480         TRUE
## Lausanne                 20.2             2 0.3626571        FALSE
## V. De Geneve             18.0            16 0.5142857        FALSE
## Rive Droite              18.2            13 0.4071588         TRUE
## Rive Gauche              19.3             7 0.4509346         TRUE
```

While we know that the `which` returns specific locations, we could also do this without the `which`, since R does know how to interpret True and False elements of a vector.

```
swiss1[swiss1$undereducated > 0,]
```

```
##              Fertility Agriculture Examination Education Catholic
## Delemont          83.1        45.1           6         9    84.84
## Lausanne          55.7        19.4          26        28    12.11
## V. De Geneve      35.0         1.2          37        53    42.34
## Rive Droite       44.7        46.6          16        29    50.43
## Rive Gauche       42.8        27.7          22        29    58.33
##              Infant.Mortality undereducated InfToFert MostCatholic
## Delemont                 22.2             3 0.2671480         TRUE
## Lausanne                 20.2             2 0.3626571        FALSE
## V. De Geneve             18.0            16 0.5142857        FALSE
## Rive Droite              18.2            13 0.4071588         TRUE
## Rive Gauche              19.3             7 0.4509346         TRUE
```

Finally, we can use a `subset` function that allows us to keep only specific rows or columns. This is sometimes easier that pulling out only specific entries of the dataframe.

```
swiss2 = subset(swiss1, swiss1$undereducated >= 0, select = c('Catholic', 'Fertility', 'InfToFert'))
dim(swiss2)
```

```
## [1] 9 3
```

```
swiss2
```

```
##              Catholic Fertility InfToFert
## Delemont        84.84      83.1 0.2671480
## Franches-Mnt    93.40      92.5 0.2183784
## Lausanne        12.11      55.7 0.3626571
## St Maurice      99.06      65.0 0.2738462
## Sierre          99.46      92.2 0.1767896
## Sion            96.83      79.3 0.2282472
## V. De Geneve    42.34      35.0 0.5142857
## Rive Droite     50.43      44.7 0.4071588
## Rive Gauche     58.33      42.8 0.4509346
```

Logical Statement Try It! Exercises

(1)

What proportion of provinces have more than half of the males involved in Agriculture?

```
# Use this code chunk to compute your solution.
```

(2)

How many provinces have both more than half of the males involved in Agriculture and more than half of the country considered Catholic? More than half of the males involved in Agriculture or more than half of the country considered Catholic?

```
# Use this code chunk to compute your solution.
```

(3)

Return to the Data Frame Try It! Exercise 6, where you calculated the proportion of males engaged in Other pursuits. How many provinces have more than 1/3 of the males engaged in Other pursuits? What proportion of provinces?

```
# Use this code chunk to compute your solution.
```

(4)

For provinces in the top half of fertility measures, what's the median proportion of Catholics? For provinces in the bottom half of fertility measures, what's the median proportion of Catholics?

```
# Use this code chunk to compute your solution.
```