Section 3: Introduction to R

Here we'll cover an "R Bootcamp", including many of the basic functions of R.

We'll focus more on the specifics of R code and RStudio. After this section, you'll have a solid foundation and exposure to the R functions we'll use throughout the course.

Pro Tip: For this section in particular, I recommend that you start to gather all of the functions that we use in the course. Keep track of them with a list or through some other method. This will help you later in the semester to easily determine which function to use.

R as a Calculator

R can be used simply as a calculator. Below is a table of operations we can use R to perform. What do you think these operations do? Record your prediction before we share the actual use.

Operation Symbol	Prediction	Actual
+		
-		
*		
/		
^		
e		

Creating Variables

Sometimes it's helpful to save the results of a calculation. Have you used the ANS button on your calculator before? That's one reason we may want to create a variable in R.

For example, consider the following code:

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

What will the output from the last line of code be?

Note: You can use either = or <- operators to name an object in R.

Hints: R is case-sensitive, which means that `d` and `D` are different to R. Naming conventions in R can also be tricky. Avoid starting a variable with a number, using spaces, using the name of a function, using the name of another object, or anything else that has meaning in R.

Creating Vectors

Vectors are lists of multiple entries that contain the same "type" of value for each entry. There are a number of methods to create vectors. Manually, you can use the `c` function, which stands for concatenate, to join or merge items together. Inside the `c` function, separate each entry with a comma.

```
a = c(2, 3)
d = a + 4
print(d)
```

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

One of the benefits of using vectors is that we can easily scale up computations; In the second line of code, we computed 2 + 4 and 3 + 4 simultaneously. We can perform the same types of simultaneous calculations with additional operations, as below.

```
d * 5
```

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

What do you think might happen with the following code?

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

What were the results shown in class? Why did this happen?

The colon is a nice shortcut if we want to create a vector with all integers between the start and end value. For example, `1:10` would result in a vector of length 10 with entries of 1, 2, 3, and so forth up to and including 10. If you want different spacing between your entries, you can alternatively use the `seq` (stands for sequence) function.

We use square brackets [ ] if we want to pull out only specific entries of a vector. Inside the square brackets, we can place number(s) or other indicators of which entries we'd like to have returned. Below are a few examples from the built-in state.name vector that includes the names of the 50 states in the United States.

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

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

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

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

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

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

Using R Functions

Now, if you are performing the same type of calculation or analysis many times, it can be helpful to create what's called a function that automates the same process for some input. We've already seen two functions that are built-in with R. There are many functions in R. For example, the sqrt function, which you may be able to anticipate what it would perform. Functions can be simple or complex with sophisticated code in the background.

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 call a function with its name.
- Function names are followed by **parentheses**. You can provide additional information inside of the parentheses, but you do need to both open and close the parentheses. The use of parentheses help tell R to implement the function; otherwise, R returns the underlying code of the function. For the built-in functions, the code likely won't make much sense at first glance.
- You can provide **arguments** or **inputs** inside of the parentheses. Not all functions require arguments, and arguments can be either required or optional. If multiple arguments are given, they should be separated by commas. Arguments have many possible settings.
  - Each argument has a name itself.
  - You assign a value to an argument using = inside the parentheses.
  - The function writer can assign a default value to an argument. If no other value is provided, then the default is used.
  - You do not need to include the argument name. I recommend the argument name in most cases as a beginner using R, as it reduces the possibility of errors occurring.
    - You can order the arguments in any order if the argument name is provided.
    - You need to order the arguments in the specified default order if you do not provide the names for each argument. You may be returned incorrect results if you provide your arguments in an incorrect order.

Knowing the anatomy of a function, can you identify any functions that we've already seen? What do you think it might do?

There are many common functions in R. A list of common R functions that we use in this course will be provided on Canvas.

Getting Help

Sometimes, you'll encounter a function that you haven't seen before. Or you'll forget exactly what a function needs as arguments and/or how it works. Maybe you received an error message, and you need a little bit of guidance or extra information. There are many reasons you may seek help with R. I regularly look at help features for functions as I work to make R do what I want. Luckily, help is pretty well developed within R.

In the Console, you can always type ? or ?? followed by the function name, package name, or dataset name for built-in data. The requested information will appear in the lower right quadrant where plots also appear in RStudio. These pages often include much more information that you need. Try to skim this page, try things out after looking over the page, and see what information you can get from here. The ? and ?? are shortcuts for going to the Help tab directly and searching from there.

Alternatively, there are many resources outside of R. Web engine searches often take you to Stack Overflow, which can be very helpful. You can also navigate directly to online discussion boards or campuswire, if needed. My recommendation: include screenshots or full examples on campuswire, so that we can help diagnose any issues and provide good guidance.

Try It! Exercises

1. We mentioned the seq function above but didn't provide an example. Here, you'll create your own example. Experiment with different options and see if you can create a sequence that goes from 0 to 1 by 0.1. Confirm that you correctly created this vector. **Challenge**: see if you can create the same vector using another method.

2. Assign the vector you just created to a letter. You can choose any letter other than c, as c is also used for the concatenate function. See if you can calculate numerical summaries of this vector. Note: Test out different functions that you think might work, but you might have to do some experimenting. Note the functions that you used below.

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

4. Apply the log function to your vector created above. What does your output look like? Describe the structure.

5. Predict what the following code will do. You can check if you were correct.

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

Tip: If you aren't sure what code will do in R, you can always print your results! Printing your results from intermediate steps helps you check that each step in your analysis is being performed as you intend.

Loading in Data

In addition to built-in data in R, you may want to analyze data located on your computer. Many times, this data will be in a csv (comma separated values).

The easiest way to read in the data is to place the data in the same folder on your computer as the Rmd document that you are working with. Then, you can use the read.csv function in R with the file name. Alternatively, you can provide the full path to the file if it is located in a different folder. The read.csv function does have additional arguments that you could specify, like whether a header with column names is present in the data. You can read more about this from the R documentation.

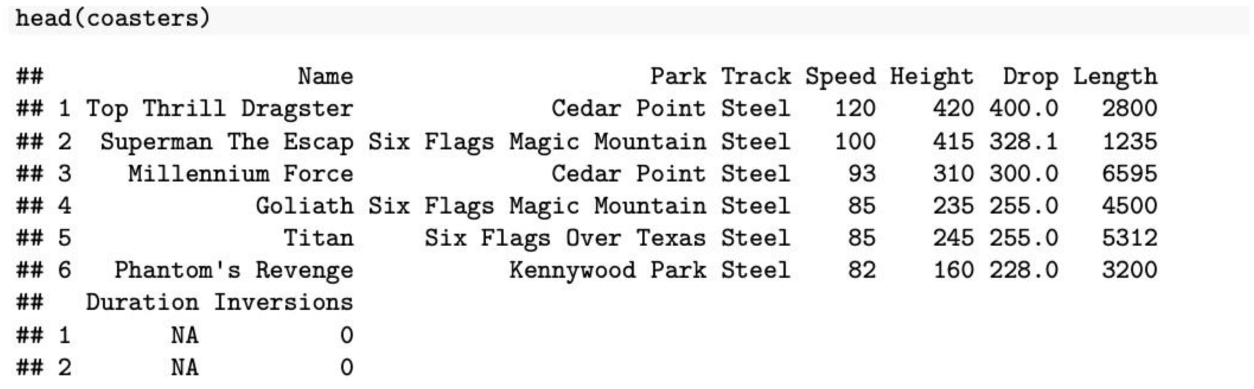
```
coasters = read.csv('coasters-clean.csv')
```

Working with Data Frames

Data Frames are a special way to save and work with data in R. They consist of columns that are vectors. The vectors are organized such that each row corresponds to the same observation, so that each row 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.

Below are some examples of code and the resulting output. See if you can guess what each of these functions are doing.



```
head(coasters)
```

```
##                 Name                      Park Track Speed Height  Drop Length
## 1 Top Thrill Dragster               Cedar Point Steel   120    420 400.0   2800
## 2  Superman The Escap Six Flags Magic Mountain Steel   100    415 328.1   1235
## 3    Millennium Force               Cedar Point Steel    93    310 300.0   6595
## 4             Goliath Six Flags Magic Mountain Steel    85    235 255.0   4500
## 5               Titan     Six Flags Over Texas Steel    85    245 255.0   5312
## 6   Phantom's Revenge            Kennywood Park Steel    82    160 228.0   3200
##   Duration Inversions
## 1       NA          0
## 2       NA          0
```

```
dim(coasters)
```

```
## [1] 241   9
```

```
ncol(coasters)
```

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

```
nrow(coasters)
```

```
## [1] 241
```

```
names(coasters)
```

```
## [1] "Name"       "Park"       "Track"      "Speed"      "Height"
## [6] "Drop"       "Length"     "Duration"   "Inversions"
```

```
colnames(coasters)
```

```
## [1] "Name"       "Park"       "Track"      "Speed"      "Height"
## [6] "Drop"       "Length"     "Duration"   "Inversions"
```

```
head(coasters$Duration)
```

```
## [1]  NA  NA 165 180 210  NA
```

```
summary(coasters$Duration)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##      28      90     120     123     150     240      84
```

```
coasters[1,8]
```

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

```
coasters[1,]
```

```
##                    Name       Park Track Speed Height Drop Length Duration
## 1 Top Thrill Dragster Cedar Point Steel   120    420  400   2800       NA
##   Inversions
## 1          0
```

```
summary(coasters[,8])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##      28      90     120     123     150     240      84
```

```
summary(coasters)
```

```
##      Name               Park               Track               Speed
##  Length:241         Length:241         Length:241         Min.   :  4.50
##  Class :character   Class :character   Class :character   1st Qu.: 45.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 55.00
##                                                           Mean   : 55.35
##                                                           3rd Qu.: 65.30
##                                                           Max.   :149.10
##
```

Data Frame Try It! Exercises

We’ll now apply some of what we’ve seen and learned to a built-in dataset in R: the swiss dataset.

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

2. Calculate the mean, standard deviation, and five number summary for the percent of the population that was educated in the provinces. Write down the functions that you used to calculate these values.

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 fairly. To calculate a z-score, subtract the mean from the observation, then divide by the standard deviation. Calculate and save the z-score for the percent of the population that was educated in each province. Then, calculate the mean, standard deviation, and five number summary for the standardized proportion of educated people across the provinces. Record the code to calculate the z-score.

4. Compare the summary values that you calculated in Exercises 2 & 3.

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 this new set of 23 observations, calculate the five number summary. Compare this to the values from Exercise 2.

6. **Challenge**: Calculate the proportion of those not involved in either Agriculture or Education in each province (those involved in Other pursuits). Find the minimum and maximum values for those involved in Other pursuits. Record the code and output below.

7. **Challenge**: First, predict what the following code will do. Then, run the code. Did your prediction match the output? If not, try to determine why the results turned out the way that they did.

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

Data Frame Operations

In the previous two sections, we began to explore dataframe operations. You tried to read code, predict the output, and interpret the output. Much of what you were doing was without any formal instruction on data frames, and I hope that you were able to accomplish some of these tasks. This section will introduce and explain some of those same functions applied to the swiss dataframe. Luckily, much of these tasks are made easier by informative function names and complete help files.

One of the first functions that I reach for when beginning to use a dataframe is the `dim` function, which prints the dimensions (rows first, then columns). R will refer to the rows first, and then the columns when referring to locations within a data frame.

```
dim(swiss)
```

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

While the dataframe has 6 columns in it, sometimes we want to pull out a specific row or a specific column. We can again do so using the square brackets, similar to vectors. Because we now have a two-dimensional object, we will need to specify the row or column number, separated by a comma. We can pull out all entries in a row by leaving the column indicator blank, or vice versa. We do need to include the comma in the brackets, so R knows whether we want a row or column returned to us.

We can also use a shortcut for pulling out a specific variable/column from a dataframe. The $ operator indicates to R that we should pull a specific variable from a dataframe, according to the following format: `dataframe_name$variable_name`. (Replace the dataframe_name & variable_name) with the appropriate names.)

We can then calculate the minimum and maximum values for the Catholic variable from the swiss dataset using informative function names as follows:

```
min(swiss$Catholic)

## [1] 2.15
max(swiss$Catholic)

## [1] 100
```

Now, suppose I want to calculate an additional variable – how much larger the educated proportion is compared to the proportion receiving highest marks on an army examination. I'll start by first calculating the difference:

```
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
```

It's helpful to print these values out to make sure that I have created 47 differences, one for each province. The problem is that now I have 47 values floating around. It can be helpful to attach this newly created variable to the original dataset. I can do so using a few different options, which we'll explore below.

First, let's look at the `cbind` (column bind) function. Let's see (a preview of) what happens when we use the cbind function first:

```
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
##               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
## Gruvere                   21.0                                  -5
```

Here, I've provided only a selection of the output. We can see that we did in fact add the variable to the dataset, but the entire dataset was printed out. The full dataset takes almost 2 full pages to be printed, so we generally wouldn't want that output to be included in our final report!

We can also confirm that the variable was added to our dataset:

```
dim(swiss)
```

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

Oh no! We have the same dimensions as before we ran the cbind function!

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 object (data frame) in R. We haven't actually created the object but simply printed the results out. We can adjust that with the code below. Note that no output is printed when running this code.

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

In our new swiss1 dataset, we have also given our new variable the name overeducated, which is easier to use later than the formula we see printed above.

Similar to cbind, there is also an rbind function which can be used to add a new row (observation) to an existing data frame.

We can also use the $ operator as a shortcut to extend data frames. Here's an example:

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

Finally, we can confirm that both of these calculations were performed correctly by checking dimensions or previewing our new swiss1 dataframe.

```
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
```

A couple of notes:

- I chose to save my new dataframe as swiss1. I wanted to create a new dataframe to explore, extend, and adjust without writing over the original, in case I made a mistake and needed to revert back to the original. Adding a number to the end is just one simple way to adjust the data name.
- 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 that capitalization is maintained in the variable names.
- The $ operator is helpful for dataframes, so we don't have to rely on a stable variable column number and correctly counting to identify a variable location. This is especially helpful for dataframe with hundreds (or more) columns.
- We previously saw vector operations and how we can extend these same operations to dataframes.
- I like viewing the data regularly as a way to check that I've done everything correctly.
- Building in "sanity checks" to your analysis can help catch mistakes and catch them early! Notice that we paused a number of times to make sure our dimensions were as expected or to look at the output before moving on to the next piece of the analysis.
- When submitting any documents for the course, please do NOT submit the printed data frame. Look through your homework document, and make sure that it isn't more than ~25 pages. swiss is pretty small, and it still takes about 2 pages to print. Instead of printing the full dataset, consider printing the head or some other selection.

Using R Markdown

RMarkdown is a special type of file that allows you to generate reports that include code, output, and text in a neat document. Additionally, it consists of many options to adjust a document stylistically.

We covered some of the basics of RMarkdown documents in Section 1. Some of the extra formatting options include:

- Within code chunks (the grey, highlighted components with code to be run)
    - Special options for the chunks, including chunk names
    - Whether to print the code in the report
    - Whether to show the output in the report
    - Whether to print any errors
    - Whether to save the results of the output to save time and memory when re-generating the report
- For the text (in the white components)
    - Headers (indicated using a single or multiple #'s)
    - 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 italicized text)
    - LaTeX math (indicated by either a single or double dollar sign at the beginning and the end of the desired text, especially used for math or Greek symbols)

You can run the code contained in the chunk in your current 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 quadrant of RStudio. 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 (many of these also appear in Section 1)

- **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 error messages provide is challenging for even the most experienced programmer, you might get a hint of what the issue is or where it's located. If all else fails, take a screenshot of the error message 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 with using the tools in that 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. You do not need to re-buy the book every time that you start R, although you can.

Note: Do not run the install.packages within an RMarkdown file, as it will try to connect with the internet to download the package and return an error message.

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

```
library(package_name)
```

This is like taking it off the bookshelf and onto your desk, so you can reference this.

Anytime that you create a new environment and want to use a package, you’ll need to run the library function. This means that you’ll need to include it in every R Markdown file that uses the package and everytime that you restart R.

We did this in Section 2 with ggplot2. See if you can go back and identify this.

Graphs

One of the most powerful analytical tools is data visualization (or graphing). We saw a little bit of that in Section 2 (the first day’s class). We’ll now turn to learning parts of ggplot more formally.

ggplot uses a consistent coding syntax (structure), which is one of the reasons that we’ll focus on using ggplot in our course. (If you are familiar with R, you can also use base R graphics.) The general format of ggplot code is:

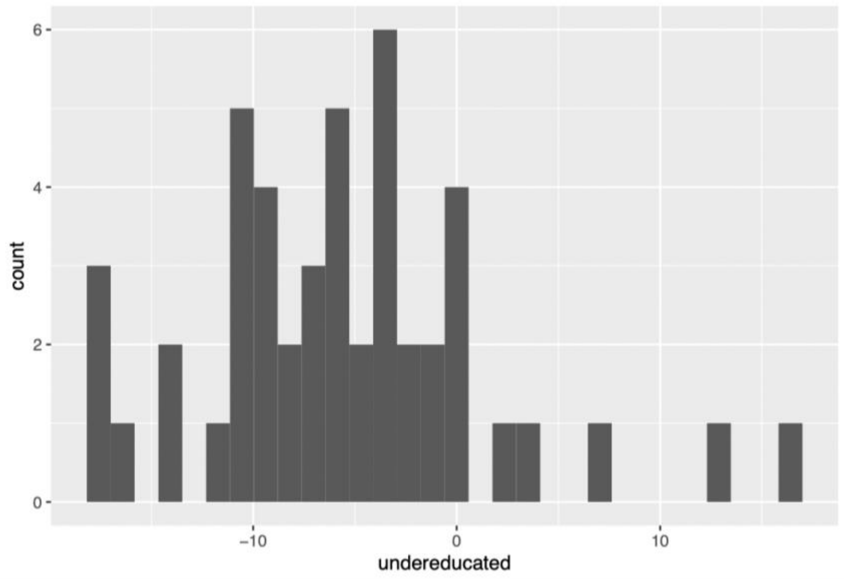
```
ggplot (data, mapping = aes(variables)) +     geom_plottype()
```

You'll need to replace the data, variables, and plottype in the code above for your specific instance. We'll start by looking at some examples. And remember that we'll need to load ggplot2 before we call ggplot for the first time in a document.

```
library(ggplot2)
```

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

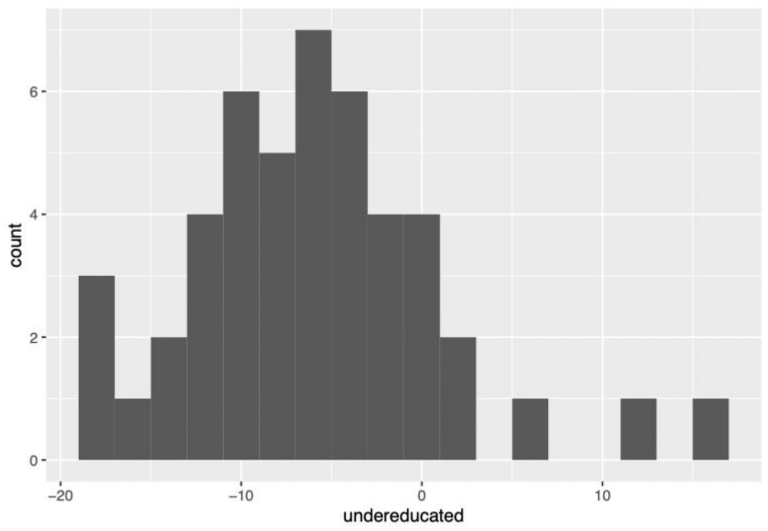
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Above, we created a histogram that represents the distribution of the values in the overeducated variable.

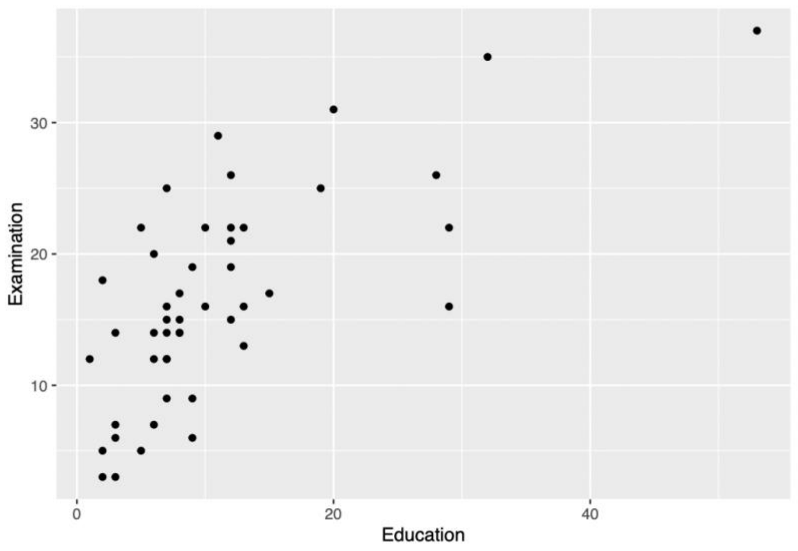
Note that a warning message was printed below this graph. The R package selected the length of bars by default, but this may not be ideal for all graphs. For the purposes of our course, you will not need to adjust the code to remove warning messages provided the graph is present and generated correctly. It is still a good idea to review any warning messages that appear. If we wanted to remove this warning message, we could specify the length of the binwidth in the code, as below.

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



We can also create scatterplots for multiple variables, like we did in the first day demo (Section 2 of the notes).

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



First Graphing Try It! Exercises

1. Make a histogram of the InfToFert variable we created (a measure of the ratio of Infant Mortality to Fertility Measure). What can you learn from this histogram?

2. Create a scatterplot of the Infant Mortality and the Fertility rates for the provinces in Switzerland. Write the code and sketch the graph below.

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

Customizing Graphs

Before starting the tutorials, I'll add one more variable to the swiss1 dataframe recording whether a majority of the province is Catholic. We'll return to this line of code shortly.

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

We saw that we can create graphs using the default settings of the graphs above. However, sometimes we want to add more to the graph. Maybe we want to add color, add lines, or adjust titles. You can find a ggplot Cheat Sheet online for just a small number of these customization options.

We'll look at a few ways to customize graphs here.

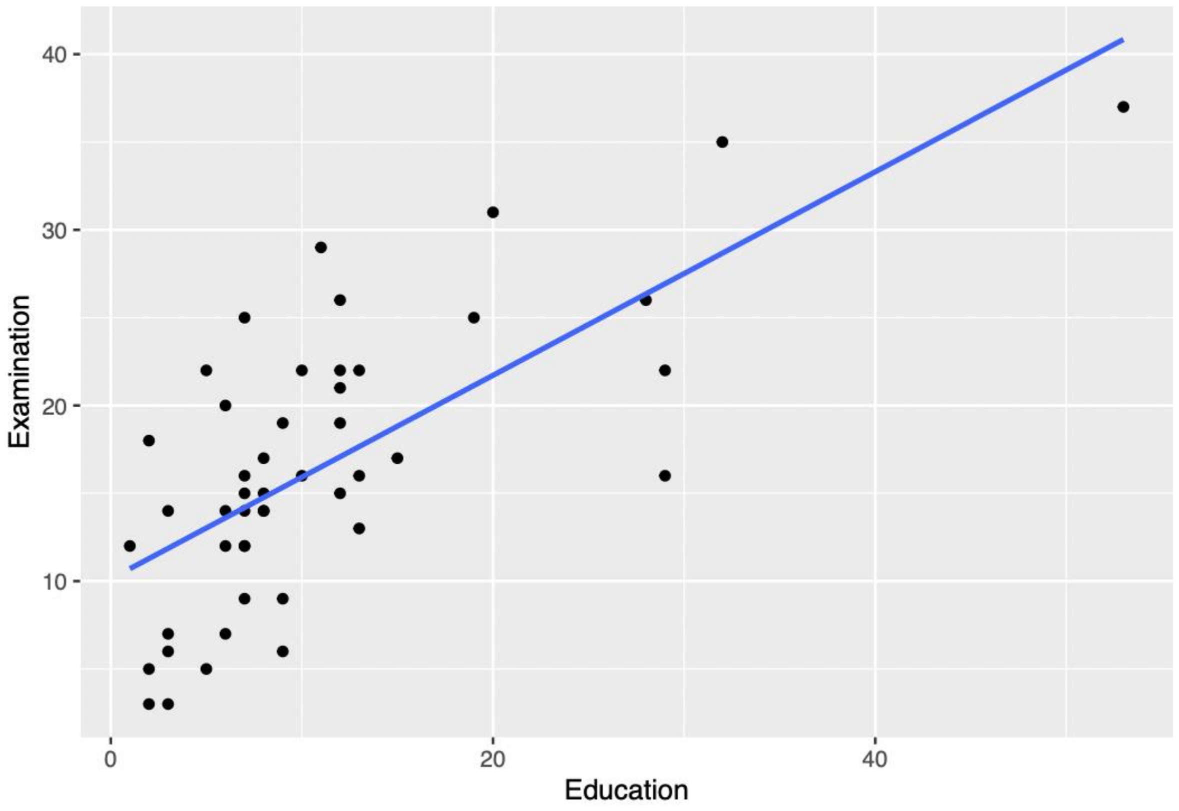
Labels

Labels can be added to graphs using the labs layer. Arguments include title, x, and y to specify the labels for the main title, x-axis title, and y-axis title respectively. This can make the graph look cleaner!

Adding Lines to Scatterplots

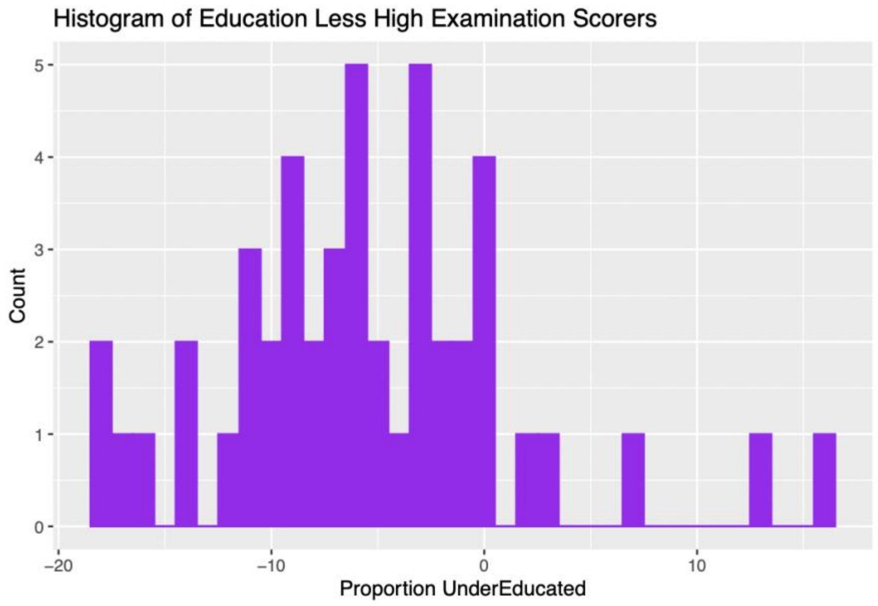
First, we can add the best fit lines to a scatterplot using the geom_smooth layer. We previously saw this in Section 2 of the Lecture Notes. We'll use different arguments in the geom_smooth layer later in the semester.

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



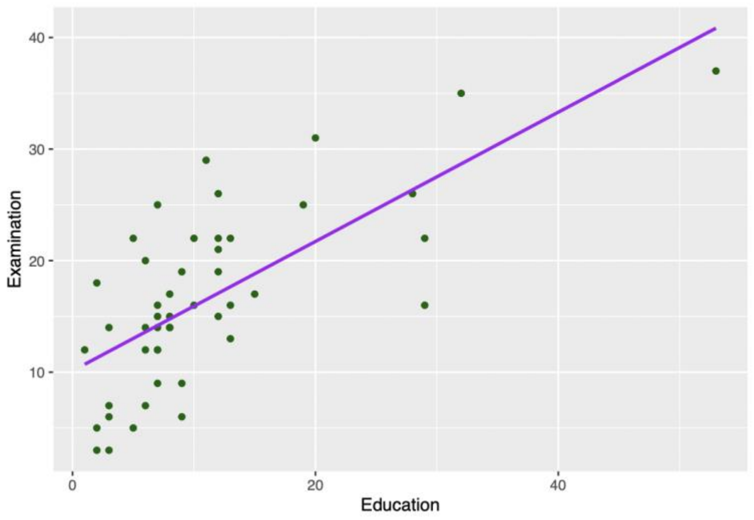
Colors for Visual Appeal

```
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 Examination Scorers")
```



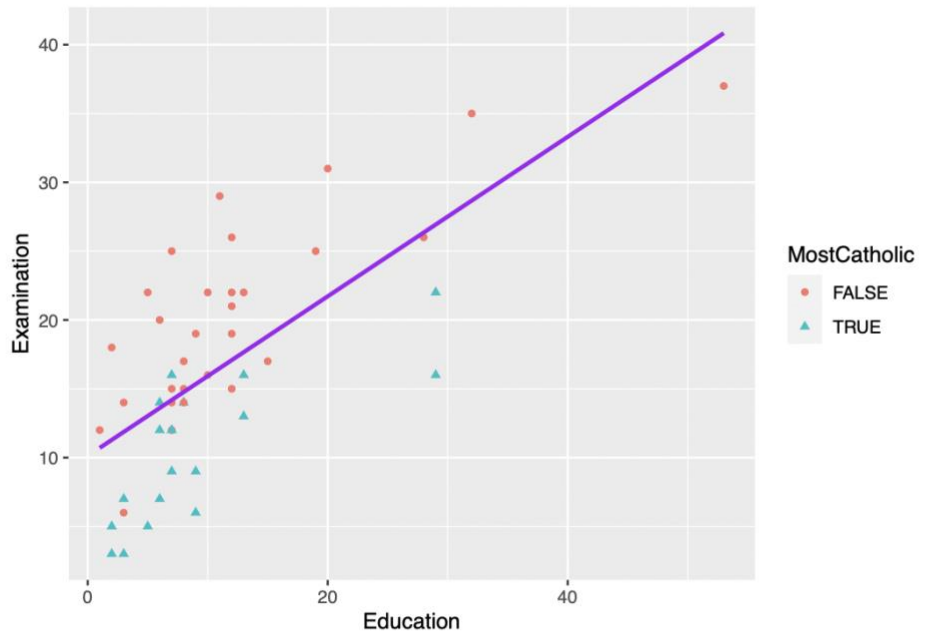
We can add color to the histogram for visual effect. We can do so similarly for a scatterplot.

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

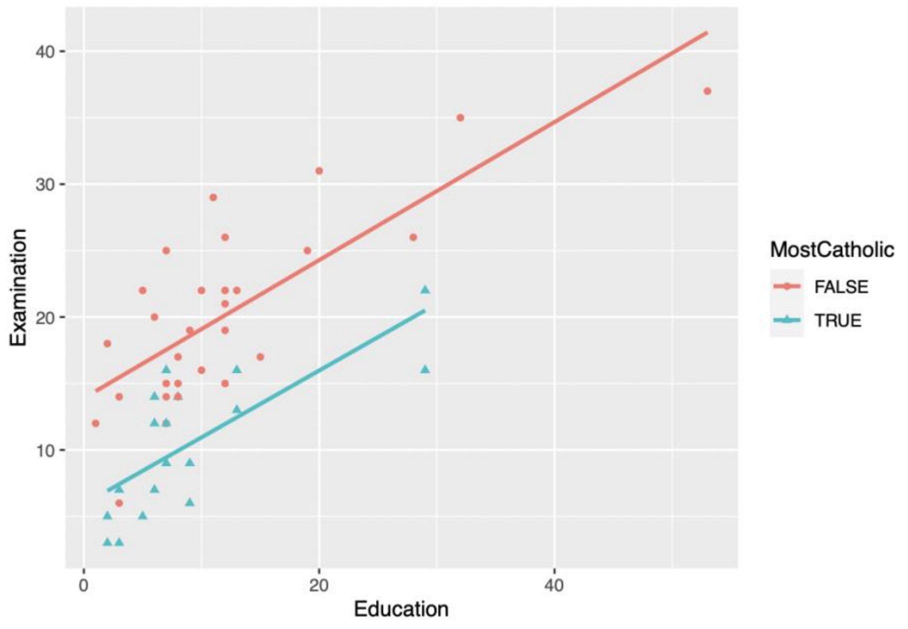


Informative Colors for a Variable

```r
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')
```

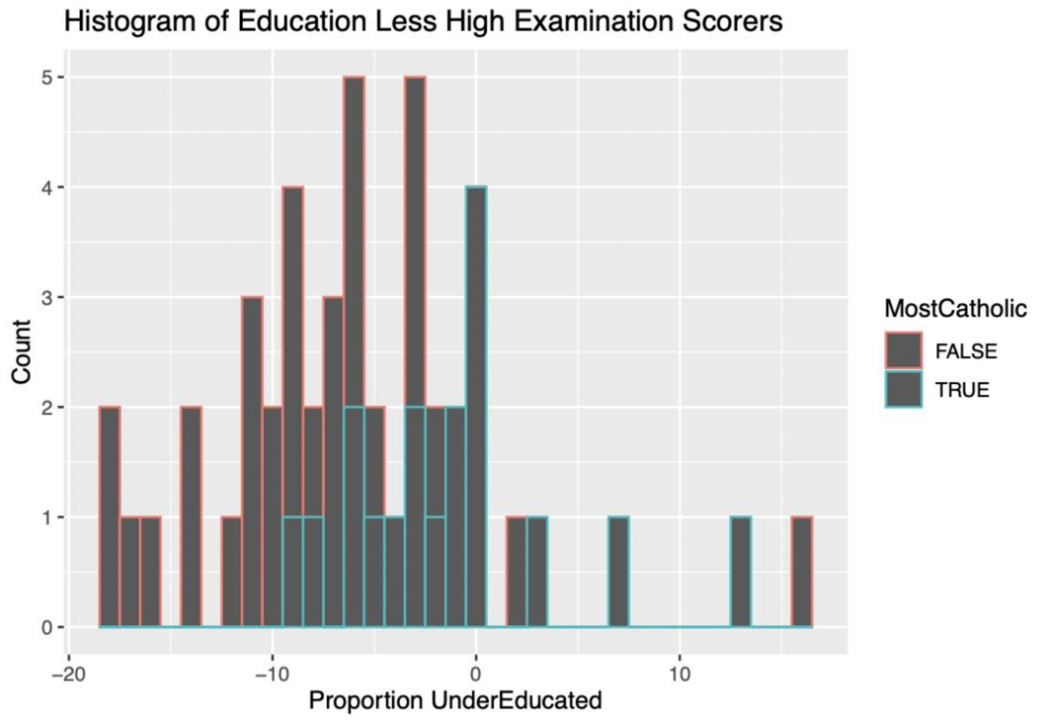


```r
ggplot(swiss1, aes(x = Education, y = Examination, color = MostCatholic, shape = MostCatholic)) +
  geom_point() +
  geom_smooth(method = 'lm', se = F, formula = 'y ~ x')
```

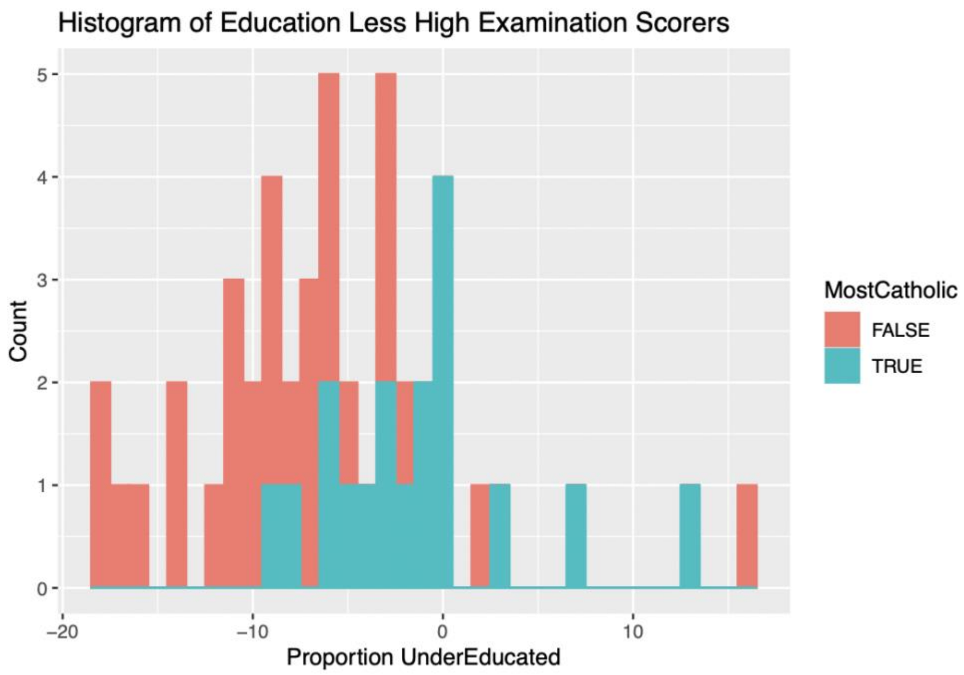


Quick Question: What is different between these two graphs? Look at the underlying code. Can you identify where the differences originate from?

```r
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")
```



```r
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")
```



In these two histograms, the coloring is associated with the variable MostCatholic. Because the coloring was associated with a variable, we placed it inside the aes function. We also added the fill argument to the second graph, so that the background color was colored by the MostCatholic variable rather than the gray of the first graph.

In the scatterplots, we saw examples where we want some customization to only appear in one layer of the graph. Note that where we introduced the MostCatholic variable in the code affected how many lines were present and how they were colored. You could continue to experiment with different combinations and locations for these customizations. If you know what features you'd like your graph to include, there's nothing wrong with trying out different combinations until your graph incorporates all of your desired features.

Graphing Customization Try It! Exercise

1. Try adding the variable MostCatholic to the scatterplot you created in the last set of graphing tutorials. Include new labels for this plot as well. Write down the code that you used to generate your graph below.

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 operators:

- > for greater than
- < for less than
- >= for greater than or equal to
- <= for less than or equal to
- == for exactly equal to (two equals signs)
- ! for not, as in != (not equal to) or !x (not x, where x could be a number or other object)
- | for or, as in x | y
- & for and, as in x & y
- isTRUE(x) for evaluating if x is True

We saw an example of a logical statement in the first graphing tutorial above:

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

Suppose we want to know more about the undereducated variable we defined earlier.

```
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 educated minus the proportion who received highest marks on an exam. This gives us a sense of what proportion of the population needed more education (was undereducated). We could look at the Education and Examination variables directly to learn the same thing as above.

```
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
```

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 provinces that had the same proportion for these two variables? Let's look into that possibility:

```
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
```

From the above vectors, identify the row numbers for the provinces where the proportion educated is greater than the proportion who scored high on an exam.

For a dataset of this size, we can easily look at the logical statement values by hand. Especially for larger datasets, we may want to summarize the logical variable. When performing calculations with logical statements, R will replace True with the value 1 and False with the value 0. This allows for elegant simplifications when performing analyses.

For example:

```
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
```

Quick Question: Explain what each of these outputs indicates about the dataset.

Finally, the `which` function can also be useful. `which` returns the location(s) that have True results for some logical statement. This is easier than the identification by eye that you recently performed and is especially useful for larger datasets.

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

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

Subsetting Data

We talked about picking out only certain elements of a vector when we introduced vectors. We can perform similar operations in relation to a dataframe in many ways.

First, we could use the same square bracket operators that we used for vectors. We also saw this in the working with dataframes section before. We can pull out a single entry

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

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

or we could pull out a larger portion of entries

```
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 entire rows or entire columns.

```
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
```

Remember that R places the row index first and the column index second, separated by a comma. We can see this above by looking at the placement of the comma in relation to the index number.

These work fine when we know the exact index number for a given row or column. However, sometimes rows or columns change locations within a dataset or a dataset is too large to be able to identify the row and column numbers by number. In these cases, we may want a more general approach to pulling out specific entries in a dataset.

Say we are interested in the 5 provinces that are undereducated. We can pull out those provinces using the logical statement & which function from the previous section:

```
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
```

In this case, the which statement is not required, as R does know how to evaluate the True and False entries in a vector, as follows:

```
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 may also opt to use the `subset` function, which allows us to keep specified rows and/or columns of a dataframe. This may be easier or faster than relying on which statements.

```
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
```

Tutorials

1. What proportion of provinces have more than half of the males involved in Agriculture? Record the logic and solution to this question.

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 (or both)? Record the logic for each of these statements.

3. Return to 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?

4. We'll consider the half of the provinces with the highest fertility measures and the half of the provinces with the lowest fertility measures separately. In other words, divide the provinces by their fertility measures. What is the median proportion of Catholics in these two halves of the population?

Additional Notes: