Project 1

Overall Goal

Our goal is to write functions that will manipulate and process data sets that come in a certain form. We'll create a *generic* function to automatically plot the returned data. We'll write up a document via R Markdown that outputs to `.html` to describe our thought process and give examples of using the functions. **You must have a narrative throughout the document.**

Data

We'll use a number of `.csv` files that contain information from the census bureau. This data is a little older (2010). Our first goal will be to read in one of these `.csv` files and parse the data using functions we've written. Then we'll combine some parsed data and deal with it from there.

Data Processing

First Steps

First I'll just explain what you'll do to parse the data you read in. Then I'll give you requirements on how to parse it below. (I find it easier to first do all of the things below and then convert certain things into functions and what-not. Feel free to create functions as required first if you'd like.)

Read in one section of the data using the code below:

```
library(readr)
sheet1 <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")
```

1. Select only the following columns:
    - `Area_name` (rename as `area_name`)
    - `STCOU`
    - Any column that ends in "D"
2. Convert the data into long format where each row has only one enrollment value for that `Area_name`.
3. One of the new columns should now correspond to the old column names that end with a "D". (All columns in these census data files will have this similar format. The first three characters represent the survey with the next four representing the type of value you have from that survey. The last two digits prior to the "D" represent the year of the measurement. For more about the variables see the data information sheet).
    - Parse the string to pull out the year and convert the year into a **numeric** value such as `1997` or `2002`.
    - Grab the first three characters and following four digits to create a new variable representing which measurement was grabbed.
        - Hint: Check out the `substr()` function from base R.
4. Create two data sets
    - one data set that contains only non-county data
    - one data set that contains only county level data

    Note that all county measurements have the format "County Name, DD" where "DD" represents the state. This can be used to subset the data. I used the code `grep(pattern = ", \\w\\w", Area_name)` to get the indices corresponding to states.

    For the county level data, add a `class` to the tibble called `county`. Similarly, add a `class` to the non-county data called `state`. This can be done by overwriting the `class()` you see on the object:

```
class(your_county_tibble) <- c("county", class(your_county_tibble))
```

5. For the county level tibble, create a new variable that describes which state one of these county measurements corresponds to (the two digit abbreviation is fine, see `substr()`).
6. For the non-county level tibble, create a new variable called "division" corresponding to the state's classification of division here. If row corresponds to a non-state (i.e. `UNITED STATES`), return `ERROR` for the division. The code for this part will not be a ton of fun but can be made easier with the use of `%in%`.

Requirements

Now we want to repeat the above process for the 2nd component of the data set. This is available at the link below and can be read in by modifying the `read_csv()` code given earlier

- https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv

Rather than copying and pasting a bunch of stuff and changing things here and there, we want to write functions that do the above pieces and one function that we can call to do it all!

- Write one function that does steps 1 & 2 above. Give an optional argument (that is it has a default value) that allows the user to specify the name of the column representing the value (enrollment for these data sets).
- Write another function that takes in the output of step 2 and does step 3 above.
- Write a function to do step 5
- Write a function to do step 6
- Write another function that takes in the output from step 3 and creates the two tibbles in step 4, calls the above two functions (to perform steps 5 and 6), and returns two final tibbles.

Now last thing, put it all into one function call! This is called creating a wrapper function. Create a function that takes in the URL of a `.csv` file in this format and the optional argument for the variable name, calls the functions you wrote above, and then returns the two tibbles.

```
#here is the idea in case it isn't clear
my_wrapper <- function(url, default_var_name = "default of some kind"){
  #a <- read_csv_code(...)
  #b <- function_for_step_1_2(a, ...)
  #c <- function_for_step_3(b, ...)
  #d <- function_for_steps4_5_6(c, ...) (where this function calls the other functions mentioned)
  #return final result
  }
```

Call It and Combine Your Data

Call the function you made two times to read in and parse the two `.csv` files mentioned so far. Be sure to call the new `value` column the same in both function calls.

Write a single short function that takes in the results of two calls to your wrapper function. The function should combine the tibbles appropriately (that is the two county level data sets get combined and the two non-county level data sets get combined). This can easily be done using `dplyr::bind_rows()`. The function should then return two data sets as one object (in the same format as the input data sets as we will be combining this output with more calls to the wrapper function in a bit).

Call this function to combine the result of the two calls to the wrapper function.

Writing a Generic Function for Summarizing

First steps

We briefly discussed the ideas around object oriented programming and method dispatch. That is, we discussed why `plot(iris)` and `plot(exp)` give different types of plots. They look at the `class` of `iris` and the `class` of `exp` and the appropriate plotting function is called.

```
#Run these in your console
plot.function #what is used for a class = function
getS3method("plot","data.frame")  #what is used for a class = data frame
```

The generic function is `plot()`. Notice that it calls `UseMethod("plot")` and that is how the appropriate plotting function is determined.

```
plot
```

```
## function (x, y, ...)
## UseMethod("plot")
## <bytecode: 0x0000025a940c7a90>
## <environment: namespace:base>
```

Great, well we have our own classes now (`county` and `state`). We can write our own custom `plot` function for these! We simply write our function as follows:

```
plot.state <- function(...){...}
```

For the `state` plotting method, let's write a function that plots the mean value of the statistic (enrollment for this data set) across the years for each `Division`. That is, on the x-axis we want the numeric year value, on the y-axis we want the mean of the statistic for each `Division` and numeric year. Also, we want to remove observations at the `ERROR` setting of `Division`. Let me help you out since we haven't covered plotting (make sure you have loaded the `tidyverse`).

```
plot.state <- function(df, var_name = "_your_default_value_"){
  #code to find the means for each division and year (think tidyverse)
  #Use get(var_name) to reference the var_name in your call to mean(): mean(get(var_name))
  #For this, think group_by along with summarize (we did mutate in class, summarize is very similar)

  #remove the "ERROR" setting

  ggplot(new_df, aes(x = _your_year_variable_, y = _your_mean_variable_, color = Division)) +
    geom_line()
}
```

Test out this function by running `plot(_class_state_df_here_, var_name = ...)`. (This doesn't need to go into the report here, just make sure it is working!)

For the class `county` we'll do a similar plotting function but with more flexibility. This function should allow the user to: - specify the state of interest, giving a default value if not specified - determine whether the 'top' or 'bottom' most counties should be looked at with a default for 'top' - instruct how many of the 'top' or 'bottom' will be investigated with a default value of 5

Within your plot function you should:

- filter the data to only include data from the state specified
- find the overall mean of the statistic (use `get(var_name)` here as well) for each `Area_name` and sort those values from largest to smallest if 'top' is specified or smallest to largest if 'bottom' is specified
- obtain the top or bottom `x` number of `Area_name`s from the previous step where `x` is given by the user or the default

- filter the data for this state to only include the `Area_name`'s from the previous part (this is the data we'll use to plot)
- use the plot code here to create a line plot:

```
ggplot(_filtered_df_, aes(x = _your_year_variable_, y = get(var_name), color = Area_name)) +
  geom_line()
```

Notice we aren't plotting the means here, the actual statistic's value. Test out this function by running `plot(_class_county_df_here_)`. Run it a few more times specifying different input arguments. (This doesn't need to go into the report here, just make sure it is working!)

Put it Together

The end of your report should have a section where you do the following:

- Run your data processing function on the two enrollment URLs given previously, specifying an appropriate name for the enrollment data column.
- Run your data combining function to put these into one object (with two data frames)
- Use the plot function on the `state` data frame
- Use the plot function on the `county` data frame
  - Once specifying the state to be "NC", the group being the top, the number looked at being 10
  - Once specifying the state to be "AZ", the group being the bottom, the number looked at being 6
  - Once without specifying anything (defaults used)
  - Once specifying the state to be "OH", the group being the top, the number looked at being 8

Lastly, read in another couple similar data sets and apply your functions!

- Run your data processing function on the four data sets at URLs given below:
  - https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv
  - https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv
- Run your data combining function (probably three times) to put these into one object (with two data frames)
- Use the plot function on the `state` data frame
- Use the plot function on the `county` data frame
  - Once specifying the state to be "PA", the group being the top, the number looked at being 5
  - Once specifying the state to be "TX", the group being the bottom, the number looked at being 12
  - Once without specifying anything (defaults used)
  - Once specifying the state to be "NY", the group being the top, the number looked at being 6

Submission

For the submission link, you should upload your `.Rmd` file and your `.html` file.

Rubric for Grading (total = 100 points)

Item	Points	Notes
Data Processing Functions	40	Worth either 0, 5, 10, ... 40
Combining Data Functions	10	Worth either 0, 5 or 10
Generic Functions	35	Worth either 0, 5, ..., 35
Putting it all together	15	Worth either 0, 5, 10, or 15

Notes on grading:

- For each item in the rubric, your grade will be lowered one level for each each error (syntax, logical, or other) in the code and for each required item that is missing or lacking a description.
- You should use Good Programming Practices when coding (see wolfware). If you do not follow GPP you can lose up to 50 points on the project.