Project 1 Fish McDowell

Data is everywhere. Its power is immeasurable with finding patterns, modeling relationships, and driving decisions. In order to be able to do those things, data must be handled appropriately. In this report, we will go through the motions of loading in and preprocessing some data so that its true power can be used as discussed above.

Initial Data Cleaning

Question 1: Selecting Columns

First, we will load in the appropriate data set and select only `Area_name`, `STCOU`, and any columns that end with the letter D, as this is the only information we need. We will also lower case the `Area_name` variable.

```
sec1 <- read_csv("./data/EDU01a.csv", col_names = TRUE)
```

```
Rows: 3198 Columns: 42
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (22): Area_name, STCOU, EDU010187N1, EDU010187N2, EDU010188N1, EDU010188...
dbl (20): EDU010187F, EDU010187D, EDU010188F, EDU010188D, EDU010189F, EDU010...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
sec1_new <- sec1 |>
  select(area_name = Area_name,
         STCOU,
         ends_with("D"))

head(sec1_new, n = 5)
```

```
# A tibble: 5 x 12
  area_name     STCOU EDU010187D EDU010188D EDU010189D EDU010190D EDU010191D
  <chr>         <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1 UNITED STATES 00000   40024299   39967624   40317775   40737600   41385442
2 ALABAMA       01000     733735     728234     730048     728252     725541
3 Autauga, AL   01001       6829       6900       6920       6847       7008
4 Baldwin, AL   01003      16417      16465      16799      17054      17479
5 Barbour, AL   01005       5071       5098       5068       5156       5173
# i 5 more variables: EDU010192D <dbl>, EDU010193D <dbl>, EDU010194D <dbl>,
#   EDU010195D <dbl>, EDU010196D <dbl>
```

The selected columns look to be what we hoped, with all that aren't `area_name` and `STCOU` end with "D".

Question 2: Long Formatted Data

Next, we will convert this data into long format with only one row per enrollment value for that area name. We will put the column names into a separate new variable to keep that information.

```
sec1_long <- sec1_new |>
  pivot_longer(cols = 3:12,
               names_to = "survey_type",
               values_to = "enrollment")

head(sec1_long, n = 5)
```

```
# A tibble: 5 x 4
  area_name     STCOU survey_type enrollment
  <chr>         <chr> <chr>            <dbl>
1 UNITED STATES 00000 EDU010187D    40024299
2 UNITED STATES 00000 EDU010188D    39967624
3 UNITED STATES 00000 EDU010189D    40317775
4 UNITED STATES 00000 EDU010190D    40737600
5 UNITED STATES 00000 EDU010191D    41385442
```

This looks to match the pivot we hoped to make.

Question 3: Further Splitting Data

As above, we notice that the entries of one of the new columns (labeled survey_type) corresponds to the old column names that end with "D". We know that the information in this column represents multiple pieces of information. Namely, the first 3 characters represent the survey, the next 4 represent the value type, and the last 2 digits represent the year of measurement. Knowing this information, we will now parse through those strings and create a new variable with the numeric date represented as YYYY. We will also do that with the first 3 and remaining 4 characters in the string.

```
long_updated <- sec1_long |>
  mutate(
    year = as.numeric(paste0("19", substr(sec1_long$survey_type, 8, 9))),
    measurement = substr(sec1_long$survey_type, 1, 7)
  )

head(long_updated, n = 5)
```

```
# A tibble: 5 x 6
  area_name     STCOU survey_type enrollment  year measurement
  <chr>         <chr> <chr>            <dbl> <dbl> <chr>
1 UNITED STATES 00000 EDU010187D    40024299  1987 EDU0101
2 UNITED STATES 00000 EDU010188D    39967624  1988 EDU0101
3 UNITED STATES 00000 EDU010189D    40317775  1989 EDU0101
4 UNITED STATES 00000 EDU010190D    40737600  1990 EDU0101
5 UNITED STATES 00000 EDU010191D    41385442  1991 EDU0101
```

Looking at the head of this data set, we have split the `survey_type` variable into the 3 separate pieces of information that it represents.

Question 4: Splitting Into County and Non-County Data

Next, we want to create two datasets, with one containing only non-county data, and the other containing only county data. We are able to do this based on how the `area_name` column is set up. We also want to create new variables corresponding to either the county or state based on which dataset it is placed into.

```
subset_index <- grep(pattern = ", \\w\\w", long_updated$area_name)

state_tibble <- long_updated[-subset_index, ]
county_tibble <- long_updated[subset_index, ]
```

```r
class(county_tibble) <- c("county", class(county_tibble))
class(state_tibble) <- c("state", class(state_tibble))

head(county_tibble, 10)
```

```
# A tibble: 10 x 6
   area_name   STCOU survey_type enrollment  year measurement
   <chr>       <chr> <chr>            <dbl> <dbl> <chr>
 1 Autauga, AL 01001 EDU010187D        6829  1987 EDU0101
 2 Autauga, AL 01001 EDU010188D        6900  1988 EDU0101
 3 Autauga, AL 01001 EDU010189D        6920  1989 EDU0101
 4 Autauga, AL 01001 EDU010190D        6847  1990 EDU0101
 5 Autauga, AL 01001 EDU010191D        7008  1991 EDU0101
 6 Autauga, AL 01001 EDU010192D        7137  1992 EDU0101
 7 Autauga, AL 01001 EDU010193D        7152  1993 EDU0101
 8 Autauga, AL 01001 EDU010194D        7381  1994 EDU0101
 9 Autauga, AL 01001 EDU010195D        7568  1995 EDU0101
10 Autauga, AL 01001 EDU010196D        7834  1996 EDU0101
```

```r
head(state_tibble, 10)
```

```
# A tibble: 10 x 6
   area_name     STCOU survey_type enrollment  year measurement
   <chr>         <chr> <chr>            <dbl> <dbl> <chr>
 1 UNITED STATES 00000 EDU010187D    40024299  1987 EDU0101
 2 UNITED STATES 00000 EDU010188D    39967624  1988 EDU0101
 3 UNITED STATES 00000 EDU010189D    40317775  1989 EDU0101
 4 UNITED STATES 00000 EDU010190D    40737600  1990 EDU0101
 5 UNITED STATES 00000 EDU010191D    41385442  1991 EDU0101
 6 UNITED STATES 00000 EDU010192D    42088151  1992 EDU0101
 7 UNITED STATES 00000 EDU010193D    42724710  1993 EDU0101
 8 UNITED STATES 00000 EDU010194D    43369917  1994 EDU0101
 9 UNITED STATES 00000 EDU010195D    43993459  1995 EDU0101
10 UNITED STATES 00000 EDU010196D    44715737  1996 EDU0101
```

Question 5: Creating new variable for county tibble

Next, we want to create a new variable in our county tibble that describes which state the county-level observation corresponds to. In order to do this, we need to get the last two characters in the string area_name, and since this exact number varies based on how many

characters are in the county name, we will utilize the nchar() function to determine the starting and stopping point in the subtr() function.

```
county_tibble <- county_tibble |>
  mutate(state_name = substr(area_name, nchar(area_name) - 1,
                             nchar(area_name)))
```

Question 6: Creating new variable for state tibble

Lastly for the initial data processing part, we want to create a new variable for the state tibble corresponding to the division.

```
state_tibble <- state_tibble |>
  mutate(division = case_when(
    area_name %in% c("CONNECTICUT", "MAINE", "MASSACHUSETTS", "NEW HAMPSHIRE",
                     "RHODE ISLAND", "VERMONT") ~ "New England",
    area_name %in% c("NEW JERSEY", "NEW YORK",
                     "PENNSYLVANIA") ~ "Mid-Atlantic",
    area_name %in% c("ILLINOIS", "INDIANA", "MICHIGAN", "OHIO",
                     "WISCONSIN") ~ "East North Central",
    area_name %in% c("IOWA", "KANSAS", "MINNESOTA", "MISSOURI", "NEBRASKA",
                     "NORTH DAKOTA", "SOUTH DAKOTA") ~ "West North Central",
    area_name %in% c("DELAWARE", "District of Columbia",
                     "DISTRICT OF COLUMBIA", "FLORIDA", "GEORGIA",
                     "MARYLAND", "NORTH CAROLINA", "SOUTH CAROLINA",
                     "VIRGINIA", "WEST VIRGINIA") ~ "South Atlantic",
    area_name %in% c("ALABAMA", "KENTUCKY", "MISSISSIPPI",
                     "TENNESSEE") ~ "East South Central",
    area_name %in% c("ARKANSAS", "LOUISIANA", "OKLAHOMA",
                     "TEXAS") ~ "West South Central",
    area_name %in% c("ARIZONA", "COLORADO", "IDAHO", "MONTANA", "NEVADA",
                     "NEW MEXICO", "UTAH", "WYOMING") ~ "Mountain",
    area_name %in% c("ALASKA", "CALIFORNIA", "HAWAII", "OREGON",
                     "WASHINGTON") ~ "Pacific",
    TRUE ~ "ERROR"
  ))
```

Creating Functions

Now that we have completed the data processing for our first dataset, we want to repeat the same process for our other dataset. Rather than copying and pasting all of our original code,

it is much more efficient for us to create functions that can do the above data cleaning for this new dataset.

Function for steps 1 and 2

We want the first function that we create to select the desired columns, and to convert the data into long format with one row per enrollment value for the area names. We are setting a default name for our new values column to be called "enrollment", and this can be changed as desired if we wanted to call it something else.

```
function1and2 <- function(messydata, val_name = "enrollment"){
  long_data <- messydata |>
    select(area_name = Area_name,
         STCOU,
         ends_with("D")) |>
    pivot_longer(cols = 3:12,
                 names_to = "survey_type",
                 values_to = val_name)
  return(long_data)
}
```

Function for step 3

Next, we want to write a function that does the task of further splitting the data that we performed in step 3 where we split up the survery type into the different components of year and survey/value type.

```
function3 <- function(long_data){
  clean_data <- long_data |>
    mutate(
    year = as.numeric(paste0("19", substr(long_data$survey_type, 8, 9))),
    survey = substr(long_data$survey_type, 1, 3),
    val_type = substr(long_data$survey_type, 4, 7)
  )
  return(clean_data)
}
```

Function for steps 5 and 6

Next, we want to write a function that does both the tasks in 5 and 6, where we altered the tibbles for the county and non county level data.

```r
function5 <- function(county_tibble){
  county_tibble <- county_tibble |>
    mutate(state_name = substr(area_name, nchar(area_name) - 1,
                               nchar(area_name)))
  return(county_tibble)
}

function6 <- function(state_tibble){
  state_tibble <- state_tibble |>
      mutate(division = case_when(
    area_name %in% c("CONNECTICUT", "MAINE", "MASSACHUSETTS", "NEW HAMPSHIRE",
                     "RHODE ISLAND", "VERMONT") ~ "New England",
    area_name %in% c("NEW JERSEY", "NEW YORK",
                     "PENNSYLVANIA") ~ "Mid-Atlantic",
    area_name %in% c("ILLINOIS", "INDIANA", "MICHIGAN", "OHIO",
                     "WISCONSIN") ~ "East North Central",
    area_name %in% c("IOWA", "KANSAS", "MINNESOTA", "MISSOURI", "NEBRASKA",
                     "NORTH DAKOTA", "SOUTH DAKOTA") ~ "West North Central",
    area_name %in% c("DELAWARE", "District of Columbia",
                     "DISTRICT OF COLUMBIA", "FLORIDA", "GEORGIA",
                     "MARYLAND", "NORTH CAROLINA", "SOUTH CAROLINA",
                     "VIRGINIA", "WEST VIRGINIA") ~ "South Atlantic",
    area_name %in% c("ALABAMA", "KENTUCKY", "MISSISSIPPI",
                     "TENNESSEE") ~ "East South Central",
    area_name %in% c("ARKANSAS", "LOUISIANA", "OKLAHOMA",
                     "TEXAS") ~ "West South Central",
    area_name %in% c("ARIZONA", "COLORADO", "IDAHO", "MONTANA", "NEVADA",
                     "NEW MEXICO", "UTAH", "WYOMING") ~ "Mountain",
    area_name %in% c("ALASKA", "CALIFORNIA", "HAWAII", "OREGON",
                     "WASHINGTON") ~ "Pacific",
    TRUE ~ "ERROR"
  ))
  return(state_tibble)
}
```

Function for step 4

Finally, we write a function that completes step 4 and creates the 2 separate datasets for county and state level data while using functions 5 and 6.

```r
function4 <- function(clean_data){
  subset_index <- grep(pattern = ", \\w\\w", clean_data$area_name)
  state_tibble <- clean_data[-subset_index, ]
  county_tibble <- clean_data[subset_index, ]
  class(county_tibble) <- c("county", class(county_tibble))
  class(state_tibble) <- c("state", class(state_tibble))
  county <- function5(county_tibble)
  state <- function6(state_tibble)
  return(list(county = county, state = state))
}
```

Putting it all into one function

Now that we have created functions that do all of the data cleaning we want, we want to combine everything into one big function that does everything for us.

```r
my_wrapper <- function(url, default_var_name = "enrollment"){
  final <- read_csv(url, col_names = TRUE) |>
    function1and2() |>
    function3() |>
    function4()
  return(final)
}
```

Now we can call this new function for both of our datasets and combine them.

```r
data1 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")
```

```
Rows: 3198 Columns: 42
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (22): Area_name, STCOU, EDU010187N1, EDU010187N2, EDU010188N1, EDU010188...
dbl (20): EDU010187F, EDU010187D, EDU010188F, EDU010188D, EDU010189F, EDU010...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
data2 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv")
```

```
Rows: 3198 Columns: 42
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (22): Area_name, STCOU, EDU010197N1, EDU010197N2, EDU010198N1, EDU010198...
dbl (20): EDU010197F, EDU010197D, EDU010198F, EDU010198D, EDU010199F, EDU010...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
combined_data <- function(data1, data2){
  combined_county <- dplyr::bind_rows(data1$county, data2$county)
  combined_state <- dplyr::bind_rows(data1$state, data2$state)
  return(list(county = combined_county, state = combined_state))
}

combined_data(data1, data2)
```

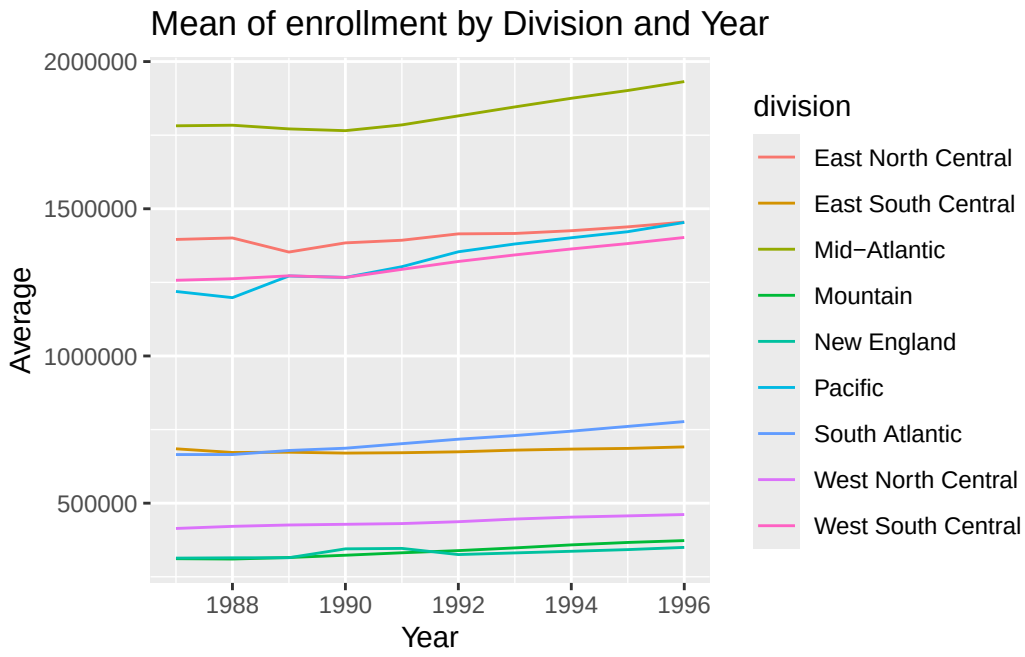
```
$county
# A tibble: 62,900 x 8
   area_name   STCOU survey_type enrollment  year survey val_type state_name
   <chr>       <chr> <chr>            <dbl> <dbl> <chr>  <chr>    <chr>
 1 Autauga, AL 01001 EDU010187D        6829  1987 EDU    0101     AL
 2 Autauga, AL 01001 EDU010188D        6900  1988 EDU    0101     AL
 3 Autauga, AL 01001 EDU010189D        6920  1989 EDU    0101     AL
 4 Autauga, AL 01001 EDU010190D        6847  1990 EDU    0101     AL
 5 Autauga, AL 01001 EDU010191D        7008  1991 EDU    0101     AL
 6 Autauga, AL 01001 EDU010192D        7137  1992 EDU    0101     AL
 7 Autauga, AL 01001 EDU010193D        7152  1993 EDU    0101     AL
 8 Autauga, AL 01001 EDU010194D        7381  1994 EDU    0101     AL
 9 Autauga, AL 01001 EDU010195D        7568  1995 EDU    0101     AL
10 Autauga, AL 01001 EDU010196D        7834  1996 EDU    0101     AL
# i 62,890 more rows

$state
# A tibble: 1,060 x 8
   area_name     STCOU survey_type enrollment  year survey val_type division
   <chr>         <chr> <chr>            <dbl> <dbl> <chr>  <chr>    <chr>
 1 UNITED STATES 00000 EDU010187D    40024299  1987 EDU    0101     ERROR
 2 UNITED STATES 00000 EDU010188D    39967624  1988 EDU    0101     ERROR
 3 UNITED STATES 00000 EDU010189D    40317775  1989 EDU    0101     ERROR
 4 UNITED STATES 00000 EDU010190D    40737600  1990 EDU    0101     ERROR
 5 UNITED STATES 00000 EDU010191D    41385442  1991 EDU    0101     ERROR
```

```
 6 UNITED STATES 00000 EDU010192D    42088151  1992 EDU    0101     ERROR
 7 UNITED STATES 00000 EDU010193D    42724710  1993 EDU    0101     ERROR
 8 UNITED STATES 00000 EDU010194D    43369917  1994 EDU    0101     ERROR
 9 UNITED STATES 00000 EDU010195D    43993459  1995 EDU    0101     ERROR
10 UNITED STATES 00000 EDU010196D    44715737  1996 EDU    0101     ERROR
# i 1,050 more rows
```

Functions for plotting

Now, we will make a function for plotting the objects that comes out of these functions we have written above. We will start with the `state` tibble. Our goal is to plot the average value of a specified numeric variable within the data frame input into the function. These average values will also be computed by geographic division and year.

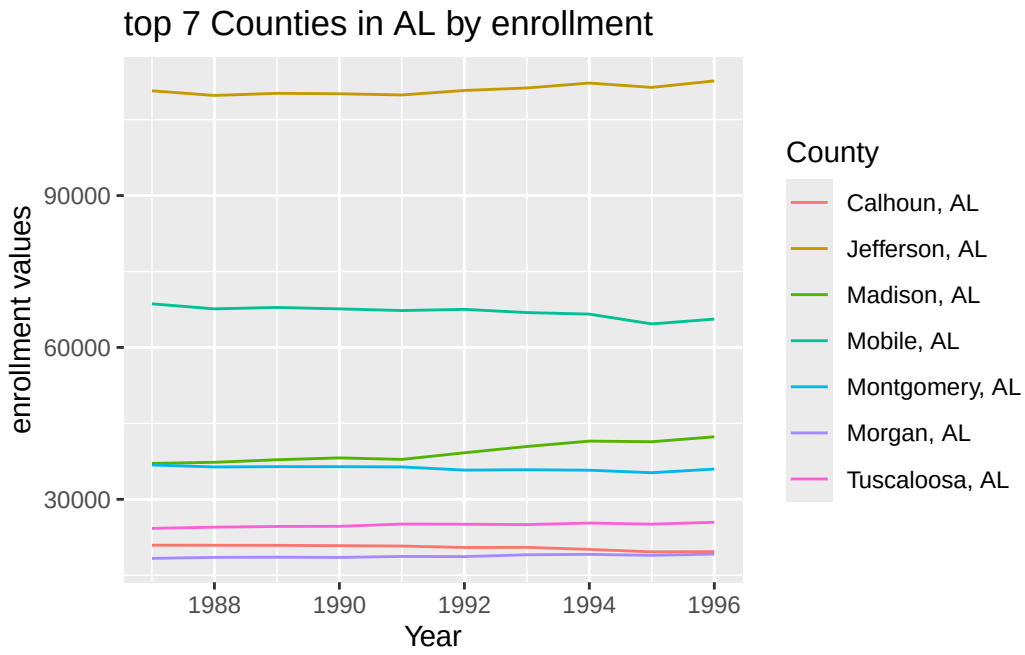
```
plot.state <- function(state_data, var_name = "enrollment"){

  #get rid of errors
  state_sum <- state_data |>
    filter(division != "ERROR") |>
    group_by(division, year) |>
    mutate(avg_vals = mean(get(var_name), na.rm = TRUE))

  #use ggplot to plot
  state_sum |>
    ggplot(aes(x = year, y = avg_vals, color = division)) +
    geom_line() +
    labs(title = paste("Mean of", var_name, "by Division and Year"),
         x = "Year",
         y = paste("Average"), var_name)
}
plot.state(data.frame(state_tibble))
```



We will now make a function to plot the county data as well. This function will allow the user to specify the state that they want to plot (with a default to NC), whether they want to plot the counties with the highest or lowest average enrollment values, and how many of those counties they wish to incorporate.

```
plot.county <- function(county_data, var_name = "enrollment", state = "NC",
                        side = "top", amount = 5){
  county_sum <- county_data |>
    filter(state_name == state) |>
    group_by(area_name) |>
    summarize(avg_vals = mean(get(var_name), na.rm = TRUE))

    if (side == "top"){
      which_counties <- county_sum |>
        arrange(desc(avg_vals)) |>
        slice_head(n = amount) |>
        pull(area_name)
    } else if (side == "bottom"){
      which_counties <- county_sum |>
        arrange(desc(avg_vals)) |>
        slice_head(n = amount) |>
        pull(area_name)
    }
```

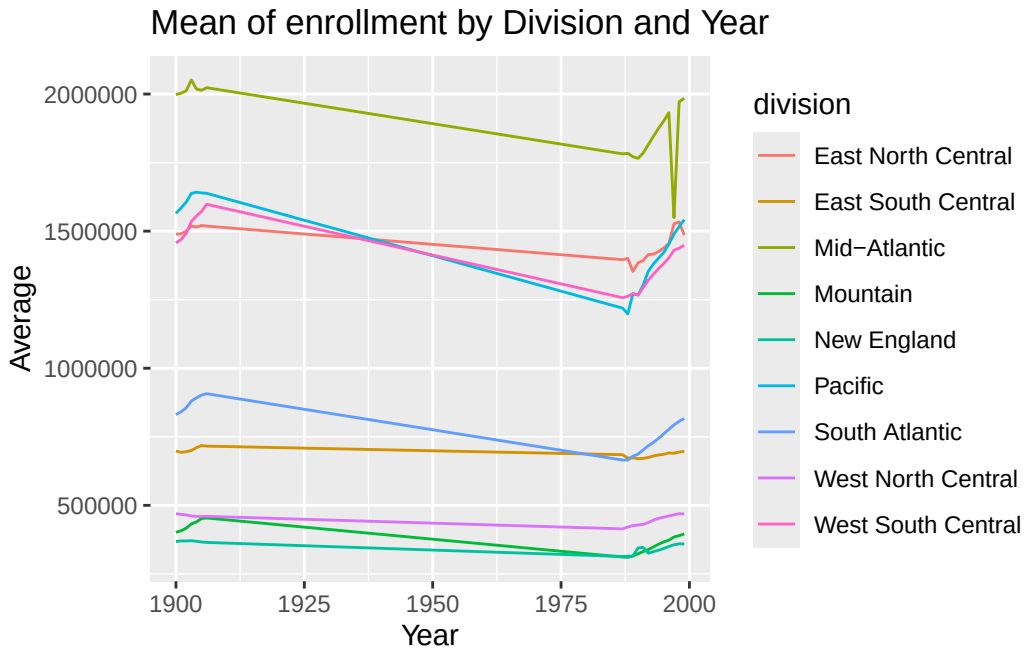
```r
  plot_data <- county_data |>
    filter(area_name %in% which_counties)


  ggplot(plot_data, aes(x = year, y = get(var_name), color = area_name)) +
    geom_line() +
    labs(title = paste(side, amount, "Counties in", state, "by", var_name),
      x = "Year",
      y = paste(var_name, "values"),
      color = "County"
    )
}

plot.county(county_tibble, state = "AL", amount = 7)
```



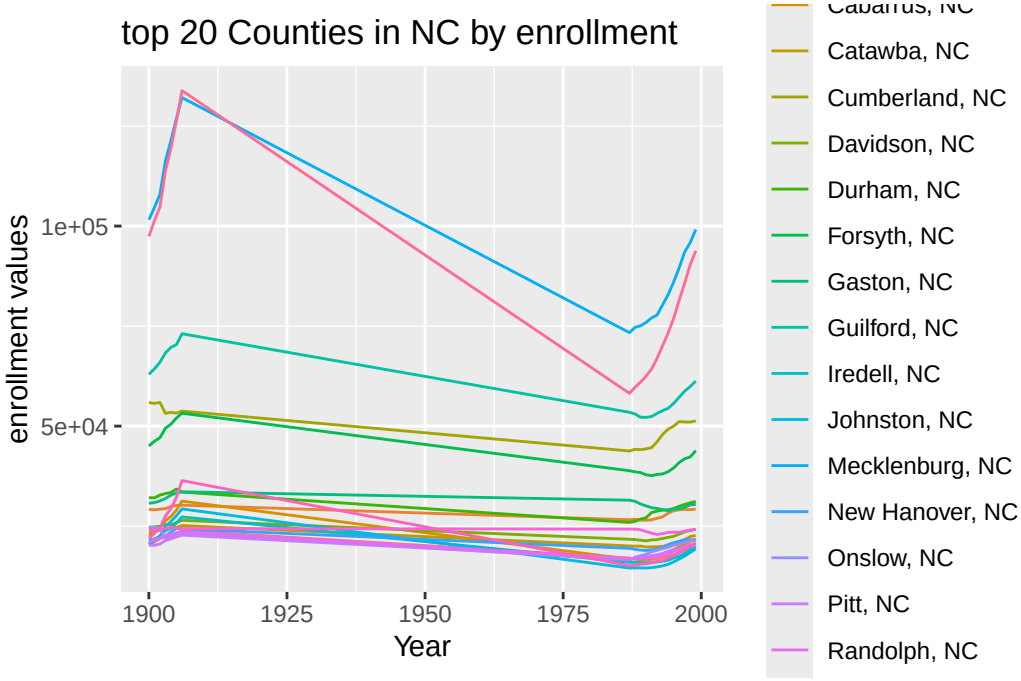
Putting it all together

We will now utilize all of the functions we have written to take in raw data, preprocess it, split it into two tibbles, combine multiple of those split tibbles, and plot them accordingly. Some examples are below.

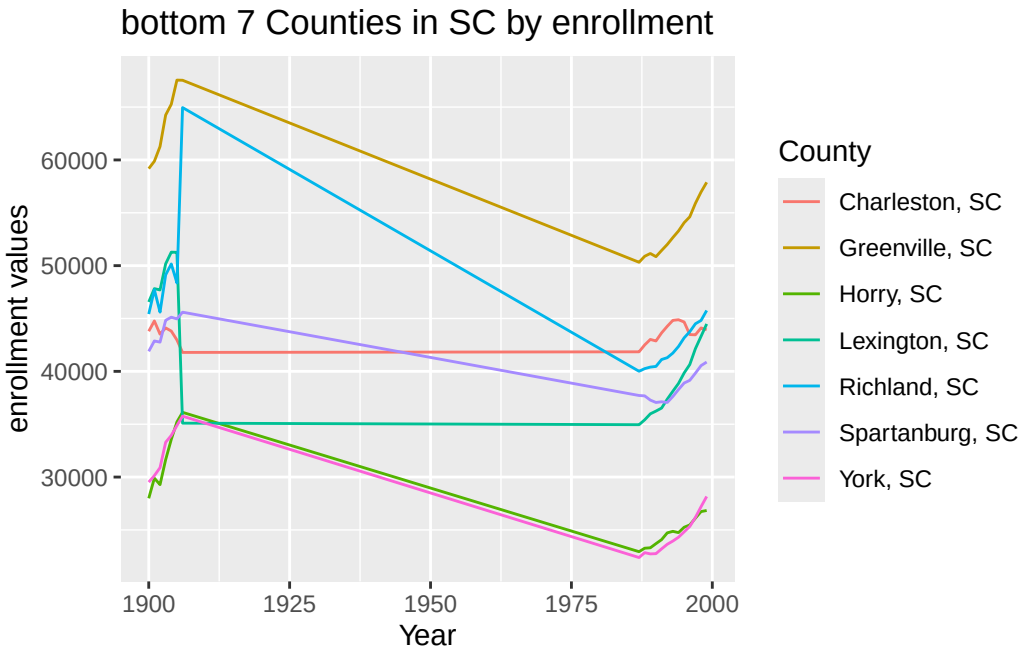
```
# run data processing function on the two enrollment URLs given previously
data1 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")
data2 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv")

# run the data combining function
data12 <- combined_data(data1, data2)

# plot the state data frame
plot.state(data12[[2]])
```



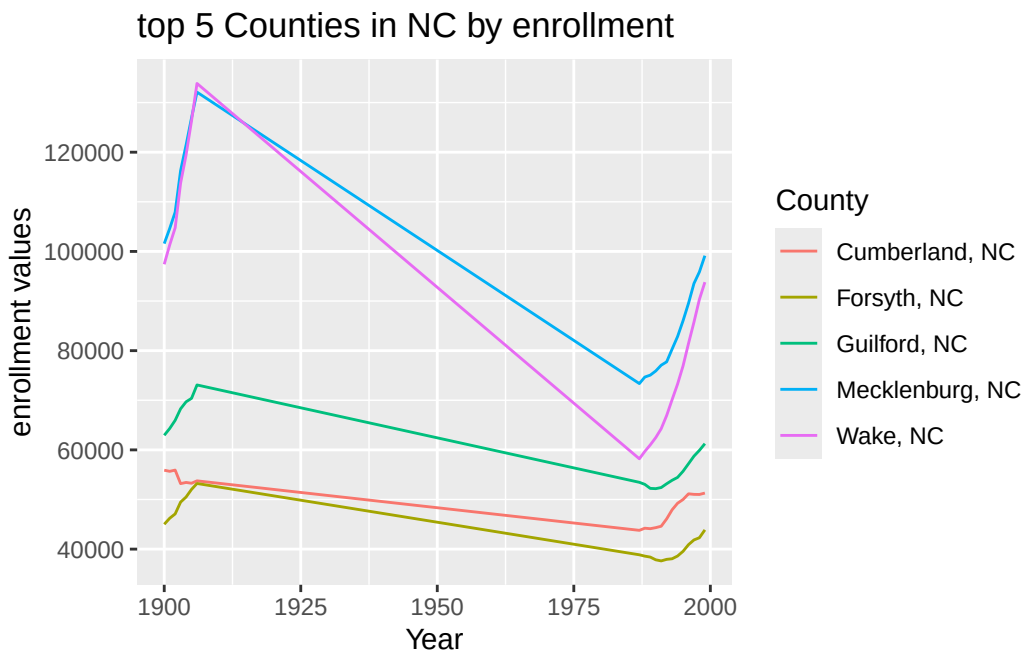

```
plot.county(data12[[1]], state = "NC", side = "top", amount = 20)
```



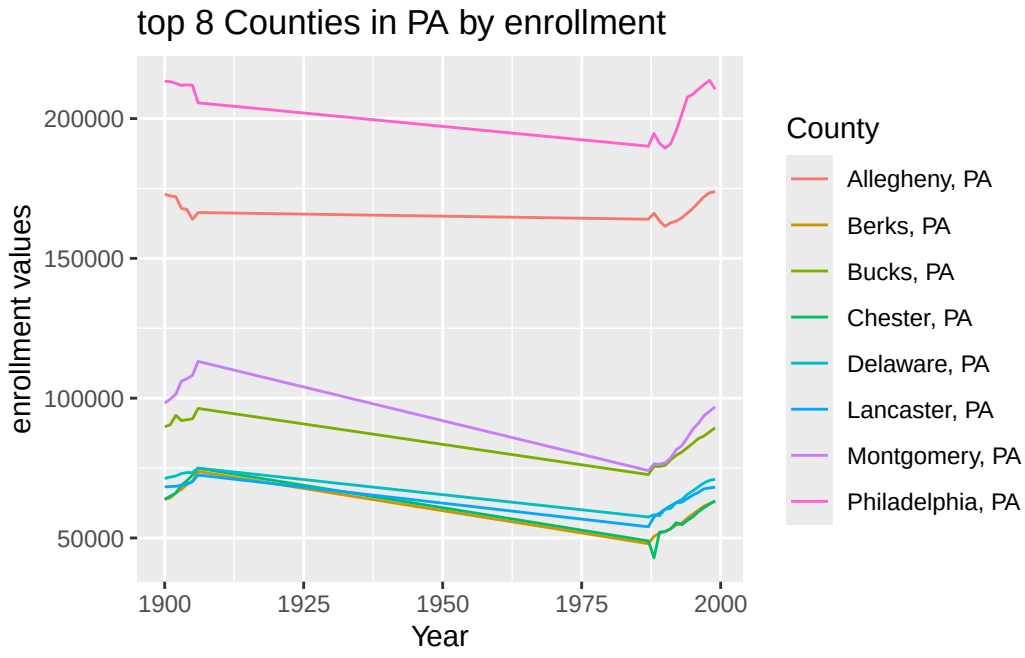

```
plot.county(data12[[1]], state = "SC", side = "bottom", amount = 7)
```



```
plot.county(data12[[1]])
```

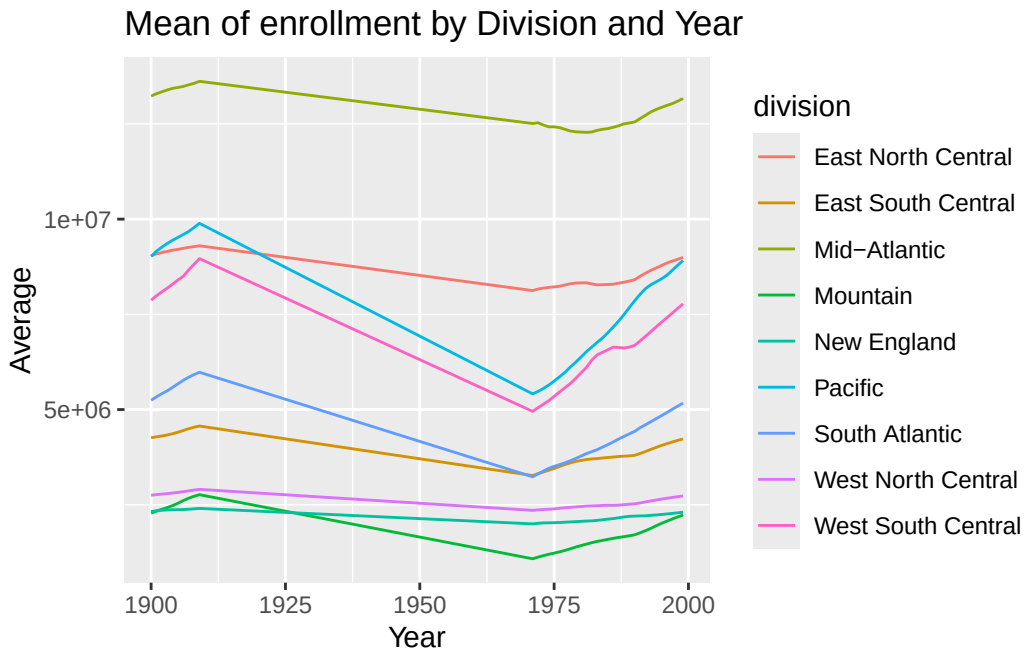



```
plot.county(data12[[1]], state = "PA", side = "top", amount = 8)
```

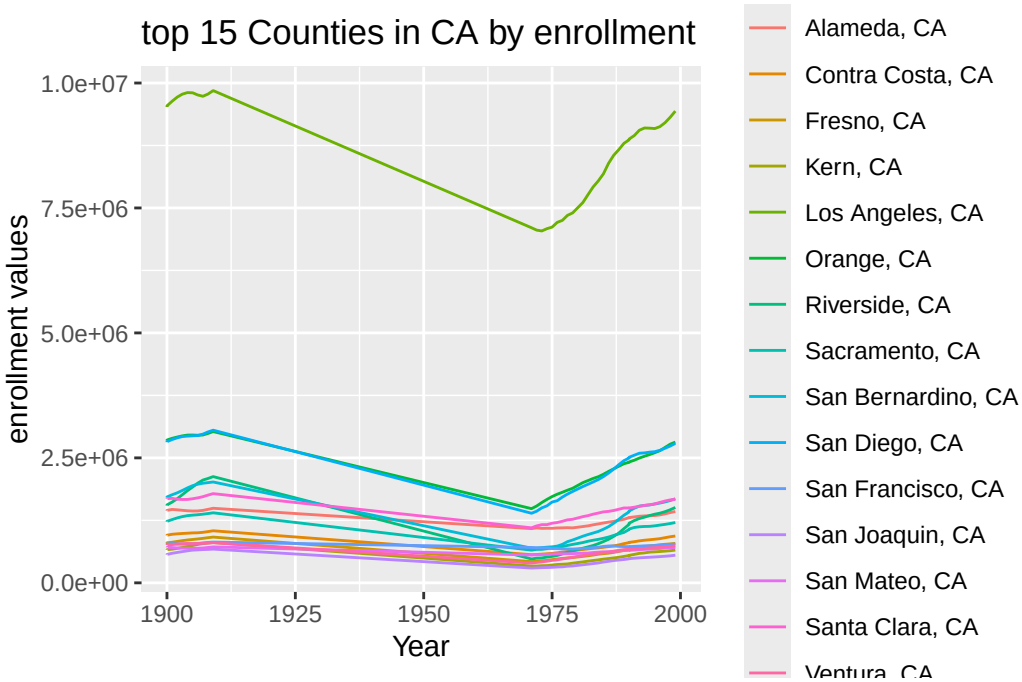


Lastly, we will start with 4 new URLs and go all the way through this process again.

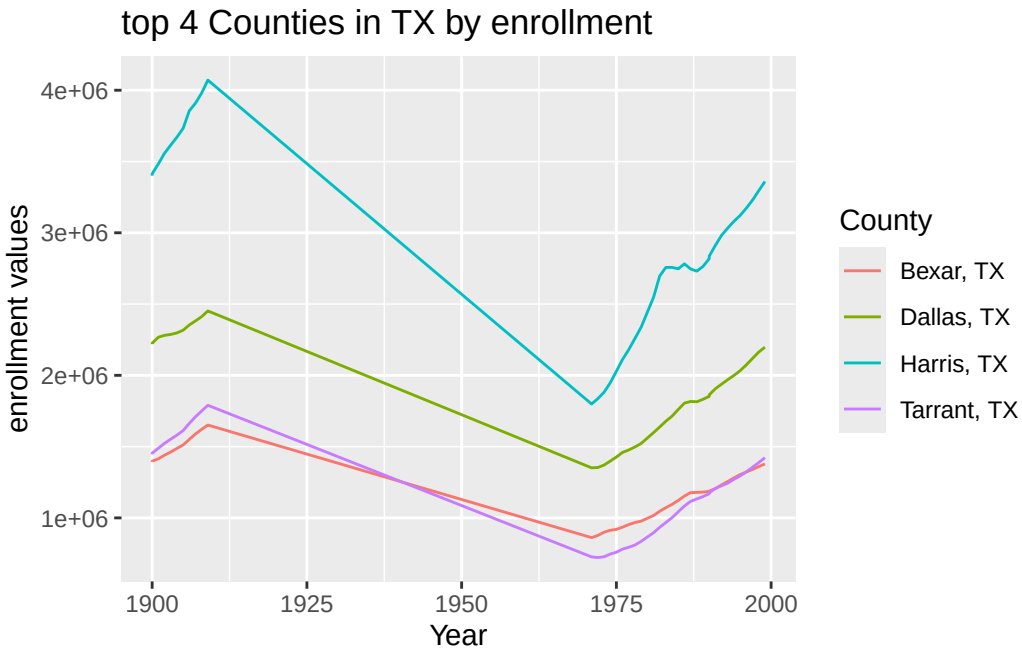
```
data3 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv")
data4 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv")
data5 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv")
data6 <- my_wrapper("https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv")

data34 <- combined_data(data3, data4)
data345 <- combined_data(data34, data5)
data_all <- combined_data(data345, data6)

plot.state(data_all[[2]])
```



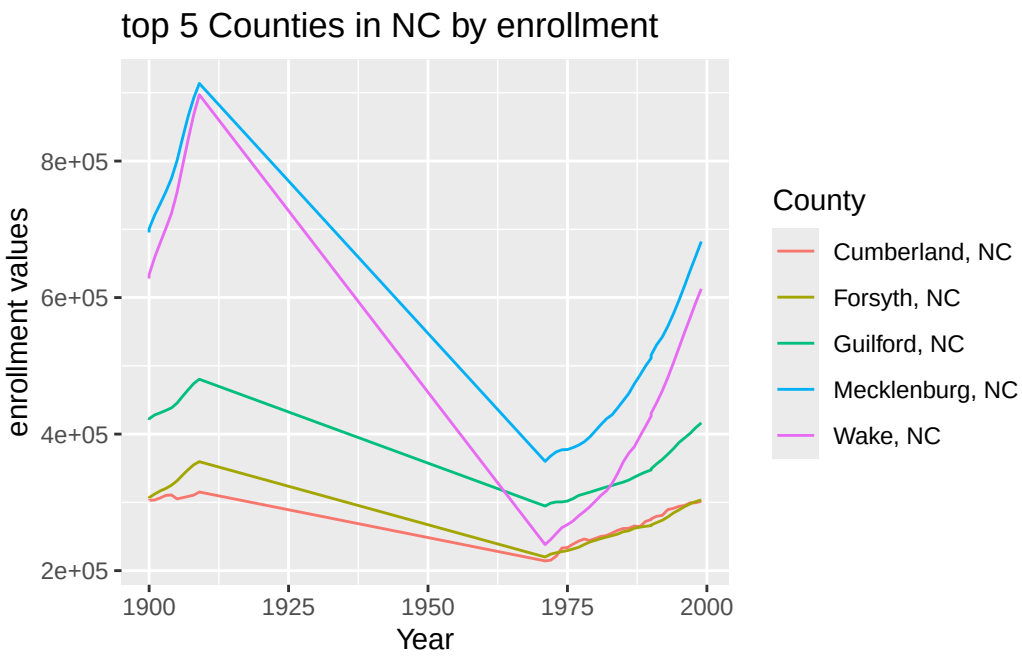

```
plot.county(data_all[[1]], state = "CA", side = "top", amount = 15)
```




```
plot.county(data_all[[1]], state = "TX", side = "top", amount = 4)
```



```
plot.county(data_all[[1]])
```




```
plot.county(data_all[[1]], state = "NY", side = "top", amount = 10)
```