# Project 1

Zach Ginder and Makenna Meyer

```r
#Load in libraries
library(tidyverse)
```

## Data Reading

First, we read in the CSV file. This file contains comma delimited information from the census about education and enrollment in the US.

```r
#Reading in comma delimited data
census_2010 <- read_csv(
  "https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")
```

## Data Processing, with and without Functions

### Question 1: Column Selection and Renaming

The first step of data processing is selecting the necessary variables. For this project, we are selecting the variables corresponding to area name, STCOU, and any column ending in "D". We also rename the area name variable.

```r
#Without function
selected_columns <- census_2010 |>
  #Selecting area name, STCOU, and all columns ending in D
  select(Area_name, STCOU, ends_with("D")) |>
  #Renaming Area_name
  rename(area_name = Area_name)

head(selected_columns, n = 5L) #Returning first 5 rows
```

```
# 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>
```

### Question 2: Long Format Conversion

The next step of data processing is converting the data file to the proper form. In this case, instead of a wide tibble, we want a long tibble where each row is an enrollment value corresponding to a particular census survey and area.

```r
#Without function
long_format <- selected_columns |>
  #taking the columns ending in D (corresponding to different census surveys)
  #and creating individual rows for each
  pivot_longer(cols = ends_with("D"), names_to = "surveys")
head(long_format, n = 5L) #Returning first 5 rows of the new tibble
```

```
# A tibble: 5 x 4
  area_name     STCOU surveys       value
  <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
```

We can also create a function that performs both the column selection and renaming steps from question one, and performs the wide to long tibble conversion from step two. This function could be used on other tibbles, thereby making additional data cleaning easier.

```
#With function
#Function that does question 1 and 2
#Convert the tibble into long format
long_conversion <- function(tibble, value = "values for enrollment") {
  long_format <- tibble |>
    #Selecting appropriate columns
    select(Area_name, STCOU, ends_with("D")) |>
    #Renaming area name
    rename(area_name = Area_name) |>
    #taking the columns ending in D
    #(corresponding to different census surveys)
    #and creating individual rows for each
    pivot_longer(cols = ends_with("D"), names_to = "surveys")
  return(long_format)
}
```

### Question 3: Create Year and Measurement Columns

The next step of data processing is to collect all of the pieces of data from the surveys column. From the metadata information, we know that we can get year from the 8th and 9th characters of the surveys column and the measurement name from the 1st through 7th character of the surveys column. We can then mutate the year variable to be 4 digits be checking whether the two digits are less than or equal to 25, indicating the year is from the 2000s. Otherwise, the year is from the 1900s. We can also write a function to do this to make data processing easier for future tibbles.

```
#Without function
#Parse the survey column to create measurement and year columns
long_updated <- long_format |>
  #Pulling year from the 8th and 9th characters of the surveys column
  mutate(years = as.numeric(substr(surveys, 8, 9))) |>
  #Converting year into a 4 digit year
  mutate(years =
           ifelse(years <= 25 & years >= 0, years + 2000, years + 1900)) |>
  #Pulling the measurement name from the 1st through
  #7th character of the surveys column
  mutate(measurements = substr(surveys, 1, 7))
head(long_updated, n = 5L)
```

```
# A tibble: 5 x 6
  area_name     STCOU surveys       value years measurements
```

```
  <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
```

```
#Function that does question 3
surveys_year_measurements <- function(long_format) {
  long_updated <- long_format |>
  #Pulling year from the 8th and 9th characters of the surveys column
  mutate(years = as.numeric(substr(surveys, 8, 9))) |>
  #Converting year into a 4 digit year
  mutate(years =
           ifelse(years <= 25 & years >= 0, years + 2000, years + 1900)) |>
  #Pulling the measurement name from the 1st through
  #7th character of the surveys column
  mutate(measurements = substr(surveys, 1, 7))
  return(long_updated)
}
```

### Question 4: Creating County and State Tibbles

For this project, we need to divide the data into two tibbles: one with county and state level information and one with only state level information. We can do this by creating a list of indices where state initials are provided in the form ", XX", then using those indices to create the county tibble. Indices that do not follow that pattern only have state information, so the rest of the data comprise the state tibble. We also add "county" as a class to the county tibble and "state" as a class to the state tibble. This will allow us to make and utilize class specific functions later on in the project.

```
#Without function
#Finding the indices corresponding to the counties
indices <- grep(pattern = ", \\w\\w", long_updated$area_name)

#Creating a county tibble that contains the county indices
county_tibble <- long_updated[indices,]
#Adding "county" as a class to the county tibble
class(county_tibble) <- c("county", class(county_tibble))

#Creating a state tibble that does not contain the county indices
```

```
state_tibble <- long_updated[-c(indices),]
#Adding "state" as a class to the state tibble
class(state_tibble) <- c("state", class(state_tibble))

#Displaying 10 rows of each tibble
head(county_tibble, n=10L)
```

```
# A tibble: 10 x 6
   area_name   STCOU surveys    value years measurements
   <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
```

```
head(state_tibble, n=10L)
```

```
# A tibble: 10 x 6
   area_name     STCOU surveys       value years measurements
   <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 State Variable for County Tibble

Next, we add a state variable to the county tibble by taking the last and second to last characters of area name. We create a function to do this so we can more easily do this again on other tibbles .

```
#Without function
county_q5 <- county_tibble |>
  #Creating a state variable from the last and second to last
  #characters of area name
  mutate(state = substr(area_name, (nchar(area_name) - 1), nchar(area_name)))
```

```
#With function
#Function to perform step 5
adding_state_to_county <- function(county_tibble){
  county_w_state <- county_tibble |>
    #Creating a state variable from the last and second to last characters
    #of area name
    mutate(state = substr(area_name, (nchar(area_name) - 1),
                          nchar(area_name)))
  return(county_w_state)
}
```

### Question 6: Creating Division Variable for Non-County Tibble

For the state tibble, we can create a division variable based on the region that the state is located in. This allows us to assess regional trends and patterns later in the project. We also created a function to do this to make this processing step easier with future tibbles.

```
#Without function
non_county_q6 <- state_tibble |>
  mutate(division =
           #Creating a division variable by matching the state name
           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("KENTUCKY", "TENNESSEE", "MISSISSIPPI",
                                 "ALABAMA")
                  ~ "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"))
```

```
#With function

#Function to perform step 6
adding_division_to_noncounty <- function(state_tibble){
  noncounty_w_division <- state_tibble |>
    mutate(division =
             #Creating a division variable by matching the state name
             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("KENTUCKY", "TENNESSEE", "MISSISSIPPI",
                                  "ALABAMA")
                     ~ "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(noncounty_w_division)
}
```

Now that we have finished the 6 data processing steps, we can create a function that performs all of the steps. This function takes in the long format tibble that we created, divides it into a county tibble and a state tibble, adds the state and division variables to those two tibbles respectively, and then returns a list of the final county tibble and the final non-county (state) tibble.

```
#Writing function that uses Step 3 output and performs Steps 4, 5, and 6

#This function takes in the long dataset created by Step 3
creating2tibbles_addingstateordivision <- function(long_updated){
  #Separating into county and state tibbles (Step 4)
  indices <- grep(pattern = ", \\w\\w", long_updated$area_name)
```

```r
  county_tibble <- long_updated[indices,]
  class(county_tibble) <- c("county", class(county_tibble))
  state_tibble <- long_updated[-c(indices),]
  class(state_tibble) <- c("state", class(state_tibble))

  #Adding the state variable to the county tibble using the Step 5 function
  county_state_final <- adding_state_to_county(county_tibble)

  #Adding the division variable to the state tibble using the Step 6 function
  noncounty_division_final <- adding_division_to_noncounty(state_tibble)

  #Returning a list of the two tibbles
  return(list("county_final" = county_state_final,
              "noncounty_final" = noncounty_division_final))
}
```

## Combining Data Functions

So far, we have read in data and processed it using functions. We are now going to create a wrapper function that both reads in the data and processes it using the functions that we have already written.

### Creating a Wrapper Function

```r
#Creating a wrapper function to read in data and perform data
#processing steps 1-6
wrapper_function <- function(url, value = "values for enrollment") {
  #Reading in the data using read_csv and the provided url
  tibbles <- read_csv(url) |>
    #Using the Step 1 and 2 function to convert the tibble from wide to long
    long_conversion(value = value) |>
    #Using the Step 3 data to create additional variables
    surveys_year_measurements() |>
    #Using the Steps 4-6 function to divide into two tibbles and
    #add variables and class
    creating2tibbles_addingstateordivision()
  return(tibbles)
}
```

### Creating a Single Short Function to Combine Tibbles From Wrapper Iterations

In case we are reading in data from multiple sources, we also want to write a function that appropriately combines the tibbles generated by the wrapper function. That way, we have larger tibbles of the appropriate classes that we can use in future class specific data presentations.

```
#Creating a single short function to combine tibbles
combine_tibbles <- function(tibble1,tibble2) {
  #Selecting the county tibbles to combine
  county_combined_tibble <- bind_rows(tibble1[["county_final"]],
                                      tibble2[["county_final"]])
  #Selecting the noncounty (state) tibbles to combine
  state_combined_tibble <- bind_rows(tibble1[["noncounty_final"]],
                                     tibble2[["noncounty_final"]])
  return(list("county_combined" = county_combined_tibble,
              "state_combined" = state_combined_tibble))
}
```

## Generic Functions

### Writing Generic Functions for Summarizing

The great utility of having class defined objects, is that we can create general functions that will handle those class objects appropriately according to their features.

Here, we create a plotting function based on the "state" class. Since we know state tibbles have division information, we can create a function that will generate plots that will display and compare division trends.

```
#Create plot.state function
plot.state <- function(state_tibble, var_name="value") {
  mean_tibble <- state_tibble |>
    #Grouping by division and year
    group_by(division, years) |>
    #Excluding all rows without an assigned division
    filter(!division %in% c("ERROR")) |>
    #Creating a mean enrollment values across the years for each division
    summarise(mean_enrollment = mean(get(var_name), na.rm = TRUE))

  #Returning a plot that displays mean enrollment over the years
  #for each division
```

```r
  return(ggplot(mean_tibble,
                aes(x = years, y = mean_enrollment, group = division,
                    color = division))
         + geom_line()) #Using a line plot to show trends over time
}
```

We can also create a plotting function based on the "county" class. Here, we have state level information rather than regional division information. Therefore, we can compare across the different counties in the state. Often we are particularly interested in the extremes (the tops and bottoms) and investigate specific number. Thus, our function will also allow the user to customize those features.

```r
#Create plot.county function

plot.county <- function(county_data, State = "KY", top_or_bottom = "top",
                      number_investigated = 5,var_name = "value") {
  mean_tibble <- county_data |>
    #Selecting only rows with a state value matching the user
    #provided value or default
    filter(state %in% (State)) |>
    #Grouping by the specific county
    group_by(area_name) |>
    #Calculating the overall mean for each county
    summarise(mean_enrollment = mean(get(var_name), na.rm = TRUE))

  #Checking the user provided value for top or bottom extreme
  if(top_or_bottom == "top") {
    final_tibble <- mean_tibble |>
      #Arranging in descending order so the first n are the top extreme
      arrange(desc(mean_enrollment)) |>
      #Selecting the user provided or default number to investigate
      head(n = number_investigated) |>
      select(area_name)
  } else {
    final_tibble <- mean_tibble |>
      #arranging in ascending order so the first n are the bottom extreme
      arrange(mean_enrollment) |>
      #Selecting the user provided or default number to investigate
      head(n = number_investigated) |>
      select(area_name)
  }
```

```r
return(final_tibble)
}
```

## Putting It All Together

Now that all of the function writing is complete, we can test our data processing and plotting functions on real data.

### Testing the functions on the initial two datasets

First, we use the wrapper and short combining functions to read in and process the data from two URLs then combine them.

```
#Using the wrapper function to read in and process the data
tibble1 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv",
                          value = value)
tibble2 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv",
                          value = value)

#Using the short combining function to combine the
#two tibbles produced by the wrapper
combined <- combine_tibbles(tibble1, tibble2)
combined
```

```
$county_combined
# A tibble: 62,900 x 7
   area_name   STCOU surveys    value years measurements state
   <chr>       <chr> <chr>      <dbl> <dbl> <chr>        <chr>
 1 Autauga, AL 01001 EDU010187D  6829  1987 EDU0101      AL
 2 Autauga, AL 01001 EDU010188D  6900  1988 EDU0101      AL
 3 Autauga, AL 01001 EDU010189D  6920  1989 EDU0101      AL
 4 Autauga, AL 01001 EDU010190D  6847  1990 EDU0101      AL
 5 Autauga, AL 01001 EDU010191D  7008  1991 EDU0101      AL
 6 Autauga, AL 01001 EDU010192D  7137  1992 EDU0101      AL
 7 Autauga, AL 01001 EDU010193D  7152  1993 EDU0101      AL
 8 Autauga, AL 01001 EDU010194D  7381  1994 EDU0101      AL
 9 Autauga, AL 01001 EDU010195D  7568  1995 EDU0101      AL
10 Autauga, AL 01001 EDU010196D  7834  1996 EDU0101      AL
# i 62,890 more rows

$state_combined
# A tibble: 1,060 x 7
```

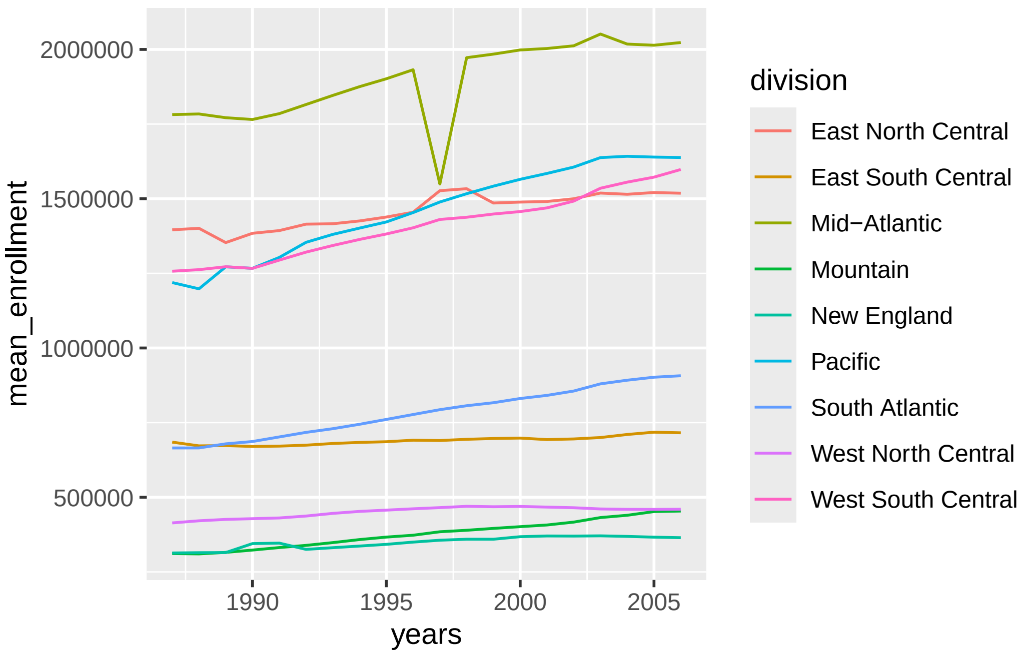
```
   area_name     STCOU surveys       value years measurements division
   <chr>         <chr> <chr>         <dbl> <dbl> <chr>        <chr>
 1 UNITED STATES 00000 EDU010187D 40024299  1987 EDU0101      ERROR
 2 UNITED STATES 00000 EDU010188D 39967624  1988 EDU0101      ERROR
 3 UNITED STATES 00000 EDU010189D 40317775  1989 EDU0101      ERROR
 4 UNITED STATES 00000 EDU010190D 40737600  1990 EDU0101      ERROR
 5 UNITED STATES 00000 EDU010191D 41385442  1991 EDU0101      ERROR
 6 UNITED STATES 00000 EDU010192D 42088151  1992 EDU0101      ERROR
 7 UNITED STATES 00000 EDU010193D 42724710  1993 EDU0101      ERROR
 8 UNITED STATES 00000 EDU010194D 43369917  1994 EDU0101      ERROR
 9 UNITED STATES 00000 EDU010195D 43993459  1995 EDU0101      ERROR
10 UNITED STATES 00000 EDU010196D 44715737  1996 EDU0101      ERROR
# i 1,050 more rows
```

### Using the Plot Function on the State Tibble

Next, we call the state plotting function on the combined state tibble.

```
#Use the plot function on the state tibble
plot(combined[[2]])
```

```
`summarise()` has grouped output by 'division'. You can override using the
`.groups` argument.
```



### Plotting the County Tibble

Finally, we call the county plotting function on different combinations of states, top or bottom extremes, and number of counties investigated.

### State is "NC", group is "top", and looking at 20

```
#Use the plot function on the county tibble
#Specify state to be NC, group is top, number is 20
plot(combined[[1]], State = "NC", top_or_bottom = "top",
     number_investigated = 20)
```

```
# A tibble: 20 x 1
   area_name
   <chr>
 1 Mecklenburg, NC
 2 Wake, NC
 3 Guilford, NC
 4 Cumberland, NC
 5 Forsyth, NC
 6 Gaston, NC
 7 Durham, NC
 8 Buncombe, NC
 9 Robeson, NC
10 Davidson, NC
11 Catawba, NC
12 Cabarrus, NC
13 New Hanover, NC
14 Union, NC
15 Onslow, NC
16 Randolph, NC
17 Pitt, NC
18 Iredell, NC
19 Alamance, NC
20 Johnston, NC
```

#### State is “SC”, group is “bottom”, and looking at 7

```r
#Use the plot function on the county tibble
#Specify state to be SC, group is bottom, number is 7
plot(combined[[1]], State = "SC", top_or_bottom = "bottom",
     number_investigated = 7)
```

```
# A tibble: 7 x 1
  area_name
  <chr>
1 McCormick, SC
2 Calhoun, SC
3 Allendale, SC
4 Saluda, SC
5 Jasper, SC
6 Bamberg, SC
7 Lee, SC
```

#### Default values

```r
#Use the plot function with defaults (state is KY, group is top, number is 5)
plot(combined[[1]])
```

```
# A tibble: 5 x 1
  area_name
  <chr>
1 Jefferson, KY
2 Fayette, KY
3 Kenton, KY
4 Hardin, KY
5 Daviess, KY
```

#### State is “PA”, group is “top”, and looking at 8

```r
#Use the plot function on the county tibble
#Specify state to be PA, group is top, number is 8
plot(combined[[1]], State = "PA", top_or_bottom = "top",
     number_investigated = 8)
```

```
# A tibble: 8 x 1
  area_name
  <chr>
1 Philadelphia, PA
2 Allegheny, PA
3 Montgomery, PA
4 Bucks, PA
5 Delaware, PA
6 Lancaster, PA
7 Berks, PA
8 Chester, PA
```

### Testing functions on four additional datasets

We can also test our functions on more than two URLs. Here, we run our functions on four URLs.

#### Running the Data Processing (Wrapping) Functions on Each of the Four URLs

First, we use the wrapper and short combining functions to read in and process the data from the four URLs.

```
tibble1 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv",
                            value = value)
tibble2 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv",
                            value = value)
tibble3 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv",
                            value = value)
tibble4 <- wrapper_function(
  url="https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv",
                            value = value)
```

#### Creating a Singular Object Using the Short Data Combining Function

Then, we use the short data combining function three times (combining 2 tibbles each time), to produce a list containing one large county tibble and one large state tibble for all of our four data files.

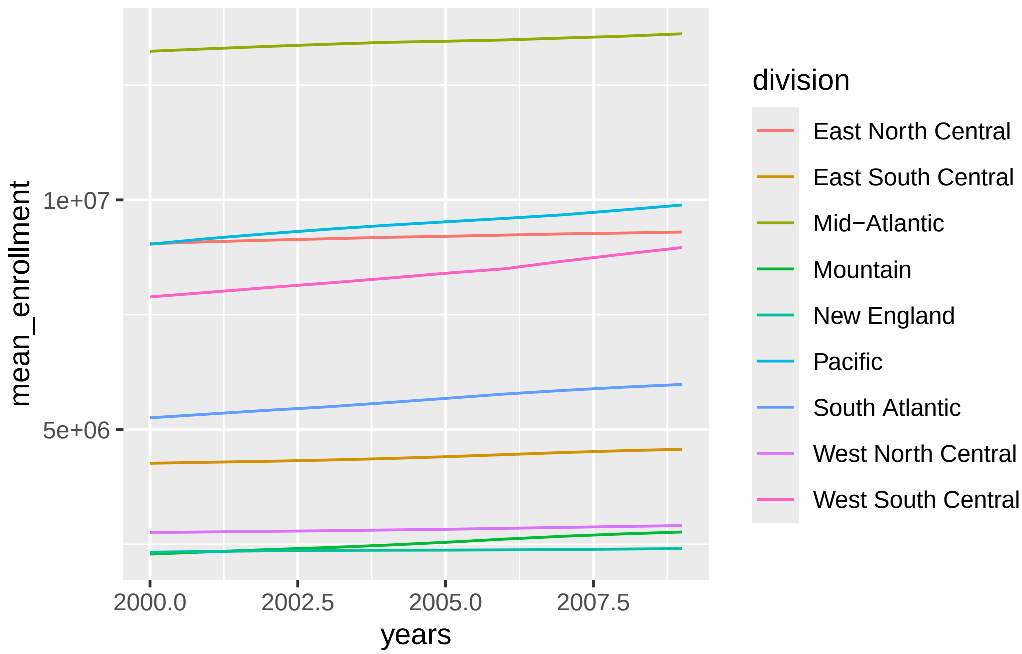
```
#combining URL1 and URL2
combined12 <- combine_tibbles(tibble1, tibble2)
#combining the combined URL1 and 2 with URL3
combined123 <- combine_tibbles(combined12, tibble3)
#combining the combined URL1 and 2 and 3 with URL4
combined1234 <- combine_tibbles(combined123, tibble4)
```

#### Using the Plot Function on the State Tibble

Next, we call the state plotting function on the combined state tibble.

```
#Use the plot function on the state tibble
plot(combined1234[[2]])
```

```
`summarise()` has grouped output by 'division'. You can override using the
`.groups` argument.
```



### Using the Plot Function on the County Tibble

Finally, we call the county plotting function on different combinations of states, top or bottom extremes, and number of counties investigated.

#### State is "CA", group is "top", and looking at 15

```
#Use the plot function on the county tibble
#Specify state to be CA, group is top, number 15
plot(combined1234[[1]], State = "CA", top_or_bottom = "top",
     number_investigated = 15)
```

```
# A tibble: 15 x 1
   area_name
   <chr>
 1 Los Angeles, CA
 2 Orange, CA
 3 San Diego, CA
 4 San Bernardino, CA
 5 Riverside, CA
 6 Santa Clara, CA
 7 Alameda, CA
 8 Sacramento, CA
 9 Contra Costa, CA
10 Fresno, CA
11 San Francisco, CA
12 Ventura, CA
13 Kern, CA
14 San Mateo, CA
15 San Joaquin, CA
```

#### State is "TX", group is "top", and looking at 4

```
#Use the plot function on the county tibble
#Specify state to be TX, group is top, number 4
plot(combined1234[[1]], State = "TX", top_or_bottom = "top",
     number_investigated = 4)
```

```
# A tibble: 4 x 1
  area_name
```

```
  <chr>
1 Harris, TX
2 Dallas, TX
3 Tarrant, TX
4 Bexar, TX
```

### Default values

```r
#Use the plot function with defaults (state is KY, group is top, number is 5)
plot(combined1234[[1]])
```

```
# A tibble: 5 x 1
  area_name
  <chr>
1 Jefferson, KY
2 Fayette, KY
3 Kenton, KY
4 Boone, KY
5 Warren, KY
```

### State is "NY", group is "top", and looking at 10

```r
#Use the plot function on the county tibble
#Specify state to be NY, group is top, number 10
plot(combined1234[[1]], State = "NY", top_or_bottom = "top",
     number_investigated = 10)
```

```
# A tibble: 10 x 1
   area_name
   <chr>
 1 Kings, NY
 2 Queens, NY
 3 New York, NY
 4 Suffolk, NY
 5 Bronx, NY
 6 Nassau, NY
 7 Westchester, NY
 8 Erie, NY
 9 Monroe, NY
10 Richmond, NY
```