Processing and Manipulating Data

Max Campbell and Mike Maccia

```
#|message=FALSE
#|warning=FALSE

# Load in necessary packages here!
suppressPackageStartupMessages(library(tidyverse))
```

Data Processing

First Steps: Importing Data

Our overall goal in this project is to process and manipulate data using common techniques and R packages. In this instance, we are using data from the 2010 census as an example! The first step in any data science project is to read in the data from wherever we are importing it from. In this case, we have some comma separated value (csv) files. Luckily, R makes this pretty easy with the `read.csv()` function!

```
#|message=FALSE

# Read in the first dataset using read.csv
census_1 <- read.csv(file =
                    "https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv",
                     header = TRUE)

# Convert to a tibble for ease of use
census_1 <- as_tibble(census_1)
```

1: Selecting Columns

Now that we have some data to work with, we can use the `dplyr` package to select the columns we care about and take a look to make sure that everything got read in properly.

```
# Select Area_name, STCOU, and any column ending with a D
census_1_small <- census_1 |>
  select(Area_name, STCOU, ends_with("D")) |>
    rename(area_name = Area_name) #rename variable

#View our selection!
head(census_1_small, n = 5)
```

```
# A tibble: 5 x 12
  area_name     STCOU EDU010187D EDU010188D EDU010189D EDU010190D EDU010191D
  <chr>         <int>      <int>      <int>      <int>      <int>      <int>
1 UNITED STATES     0   40024299   39967624   40317775   40737600   41385442
2 ALABAMA        1000     733735     728234     730048     728252     725541
3 Autauga, AL    1001       6829       6900       6920       6847       7008
4 Baldwin, AL    1003      16417      16465      16799      17054      17479
5 Barbour, AL    1005       5071       5098       5068       5156       5173
# i 5 more variables: EDU010192D <int>, EDU010193D <int>, EDU010194D <int>,
#   EDU010195D <int>, EDU010196D <int>
```

Looks good!

2: Pivoting Data

There is some information stored in the column names that we care about in our data, and it isn't exactly easy to analyze that data when it's stored in the name. As such, we are going to pivot this data into a longer dataframe.

```
#Pivot longer so that we can access data stored in some column names
census_1_long <- census_1_small |>
  pivot_longer(cols = 3:12,
             names_to = "survey_ID",
             values_to = "enrollment")

#View results!
head(census_1_long, n = 5)
```

```
# A tibble: 5 x 4
  area_name     STCOU survey_ID  enrollment
  <chr>         <int> <chr>           <int>
1 UNITED STATES     0 EDU010187D   40024299
```

```
2 UNITED STATES     0 EDU010188D   39967624
3 UNITED STATES     0 EDU010189D   40317775
4 UNITED STATES     0 EDU010190D   40737600
5 UNITED STATES     0 EDU010191D   41385442
```

Notice how there is only one measurement per row now!

3: Parsing Strings

Let's take a closer look at the `survey_ID` column now. According to the data information sheet provided, the first three letters and first four numbers represent the type of survey that was completed. For example, row one has the value `EDU0101` which represents school survey_ID. The next two digits represent the year the survey was taken. Using the first row as an example again, we see that those two digits are `87`, meaning that the survey was taken in 1987. We can parse this column to get some meaningful data year-over-year.

```
#Parse the survey_ID column:
#survey_ID - first three letters plus first four digits (ex. EDU0101)
#year - next two digits (ex. 87 becomes 1987)
long_updated <- census_1_long |>
  mutate(year = substr(survey_ID, 8, 9),
         survey_ID = substr(survey_ID, 1, 7)) |>
  #Add 1900 (or 2000) to the year column to get the YYYY instead of YY
  mutate(year = case_when(
    as.numeric(year) >= 87 ~ 1900 + as.numeric(year),
    .default = 2000 + as.numeric(year)
    )
  )

#View results!
head(long_updated, n = 5)
```

```
# A tibble: 5 x 5
  area_name     STCOU survey_ID enrollment  year
  <chr>         <int> <chr>          <int> <dbl>
1 UNITED STATES     0 EDU0101     40024299  1987
2 UNITED STATES     0 EDU0101     39967624  1988
3 UNITED STATES     0 EDU0101     40317775  1989
4 UNITED STATES     0 EDU0101     40737600  1990
5 UNITED STATES     0 EDU0101     41385442  1991
```

Now we have two columns to explain the survey information in a more intuitive manner.

4: Splitting the Dataset

Note that when we first imported the dataset, we saw that there was data for the country as a whole, as well as individual states as well as the counties within the states. If we wished to analyze the data, it may make more sense to be able to easily subset it by county and non-county data so we can get better insights. As such, let's split the data into a county set and a non-county set. We can use `dplyr`'s `slice()` function to accomplish this, using an expression provided that returns the indices of all rows that match a given condition!

```
#Filter tibble into two separate tibbles using county as the identifier
county_tibble <- long_updated |>
  slice(grep(pattern = ", \\w\\w", area_name))

#We can invert using the 'invert' argument in grep() as well
state_tibble <- long_updated |>
  slice(grep(pattern = ", \\w\\w", area_name, invert = TRUE))

#Add the custom classes to the tibbles for future use
class(county_tibble) <- c("county", class(county_tibble))
class(state_tibble) <- c("state", class(state_tibble))

head(county_tibble, n = 10)
```

```
# A tibble: 10 x 5
   area_name   STCOU survey_ID enrollment  year
   <chr>       <int> <chr>          <int> <dbl>
 1 Autauga, AL  1001 EDU0101         6829  1987
 2 Autauga, AL  1001 EDU0101         6900  1988
 3 Autauga, AL  1001 EDU0101         6920  1989
 4 Autauga, AL  1001 EDU0101         6847  1990
 5 Autauga, AL  1001 EDU0101         7008  1991
 6 Autauga, AL  1001 EDU0101         7137  1992
 7 Autauga, AL  1001 EDU0101         7152  1993
 8 Autauga, AL  1001 EDU0101         7381  1994
 9 Autauga, AL  1001 EDU0101         7568  1995
10 Autauga, AL  1001 EDU0101         7834  1996
```

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

```
# A tibble: 10 x 5
   area_name     STCOU survey_ID enrollment  year
   <chr>         <int> <chr>         <int> <dbl>
 1 UNITED STATES     0 EDU0101    40024299  1987
 2 UNITED STATES     0 EDU0101    39967624  1988
 3 UNITED STATES     0 EDU0101    40317775  1989
 4 UNITED STATES     0 EDU0101    40737600  1990
 5 UNITED STATES     0 EDU0101    41385442  1991
 6 UNITED STATES     0 EDU0101    42088151  1992
 7 UNITED STATES     0 EDU0101    42724710  1993
 8 UNITED STATES     0 EDU0101    43369917  1994
 9 UNITED STATES     0 EDU0101    43993459  1995
10 UNITED STATES     0 EDU0101    44715737  1996
```

5: Splitting Strings in the County Data

We may want to subset by state in our analysis, so let's subset the county data to separate the county's name and state by using the comma as a delimiter.

```
#Split the county name and state using , as a delimiter
county_1_split <- county_tibble |>
  # separate() indicates that it is superseded per help file
  # Using separate_wider_delim() in favor of separate()
  separate_wider_delim(area_name, ",", names = c("county", "state")) |>
  # Trim the whitespace on the state column
  mutate(state = trimws(state))

#View results
head(county_1_split, n = 5)
```

```
# A tibble: 5 x 6
  county  state STCOU survey_ID enrollment  year
  <chr>   <chr> <int> <chr>          <int> <dbl>
1 Autauga AL     1001 EDU0101         6829  1987
2 Autauga AL     1001 EDU0101         6900  1988
3 Autauga AL     1001 EDU0101         6920  1989
4 Autauga AL     1001 EDU0101         6847  1990
5 Autauga AL     1001 EDU0101         7008  1991
```

6: Designating Regions in Non-County Data

Similar to what we did in the county data above, we can describe the non-county data by larger groups. For the county data, this was the state each county was located in. For the non-county data, it will be the the divisions described by the Census Bureau's designation. In other words, there are nine defined divisions that the Census Bureau that could be useful for our analysis. In order to indicate divisions we will have to use the `case_when` function in conjunction with the `%in%` operator to check and see if each entry is contained within a specific list of states belonging to a division.

```
#Create string vectors of the states for clarity in the mutate call
ne <- c("CONNECTICUT", "MAINE", "MASSACHUSETTS",
        "NEW HAMPSHIRE", "RHODE ISLAND", "VERMONT")
ma <- c("NEW JERSEY", "NEW YORK", "PENNSYLVANIA")
enc <- c("ILLINOIS", "INDIANA", "MICHIGAN", "OHIO", "WISCONSIN")
wnc <- c("IOWA", "KANSAS", "MINNESOTA",
         "MISSOURI", "NEBRASKA", "NORTH DAKOTA",
         "SOUTH DAKOTA")
sa <- c("DELAWARE", "DISTRICT OF COLUMBIA", "FLORIDA",
        "GEORGIA", "MARYLAND", "NORTH CAROLINA",
        "SOUTH CAROLINA", "VIRGINIA", "WEST VIRGINIA")
esc <- c("ALABAMA", "KENTUCKY", "MISSISSIPPI", "TENNESSEE")
wsc <- c("ARKANSAS", "LOUISIANA", "OKLAHOMA", "TEXAS")
mtn <- c("ARIZONA", "COLORADO", "IDAHO",
         "MONTANA", "NEVADA", "NEW MEXICO",
         "UTAH", "WYOMING")
pac <- c("ALASKA", "CALIFORNIA", "HAWAII", "OREGON", "WASHINGTON")

# Add the division variable to indicate Census-designated divisions
non_county_1_div <- state_tibble |>
  mutate(division = case_when(
    area_name %in% ne ~ "New England",
    area_name %in% ma ~ "Mid-Atlantic",
    area_name %in% enc ~ "East North Central",
    area_name %in% wnc ~ "West North Central",
    area_name %in% sa ~ "South Atlantic",
    area_name %in% esc ~ "East South Central",
    area_name %in% wsc ~ "West South Central",
    area_name %in% mtn ~ "Mountain",
    area_name %in% pac ~ "Pacific",
    .default = "ERROR"
  ))

#View results from the end
#since the US data is first and will return ERROR
tail(non_county_1_div, n = 5)
```

```
# A tibble: 5 x 6
  area_name STCOU survey_ID enrollment  year division
  <chr>     <int> <chr>          <int> <dbl> <chr>
1 WYOMING   56000 EDU0101       101715  1992 Mountain
2 WYOMING   56000 EDU0101       100729  1993 Mountain
3 WYOMING   56000 EDU0101       100899  1994 Mountain
4 WYOMING   56000 EDU0101       100369  1995 Mountain
5 WYOMING   56000 EDU0101        99859  1996 Mountain
```

With that, we've done everything that we'll need to do for our analysis! However, there is one key step that we need to do to make this process easily reproducible.

Generalizing into Function Calls

This census data has two key characteristics that are relevant right now: the data that is imported follows the same general format, and we want to process the data in several different ways. That is, we want to easily be able to do the same processes without having to do a bunch of copying-and-pasting work. Luckily, we can write our own functions to take care of that!

There will have to be some generalizations made. For example, what if we want to observe data for a different metric? The census takes record of many different aspects of American demographics, and we may be interested in analyzing a different survey one day. Also, the latest observations that we have in the data we read in previously are from 1996, which is almost 30 years ago! We will want to read in some more recent records as well to get a bigger picture on survey_ID data.

So, we will write functions to generalize the processes performed above, and we will write a function to merge datasets in order to congregate our newer and older data in one (well, two, because we still want to split the data by county status) data frame(s)!

```r
#---STATE STRINGS---
#We will need these again for one of the helper functions
ne <- c("CONNECTICUT", "MAINE", "MASSACHUSETTS",
        "NEW HAMPSHIRE", "RHODE ISLAND", "VERMONT")
ma <- c("NEW JERSEY", "NEW YORK", "PENNSYLVANIA")
enc <- c("ILLINOIS", "INDIANA", "MICHIGAN", "OHIO", "WISCONSIN")
wnc <- c("IOWA", "KANSAS", "MINNESOTA",
         "MISSOURI", "NEBRASKA", "NORTH DAKOTA",
         "SOUTH DAKOTA")
sa <- c("DELAWARE", "DISTRICT OF COLUMBIA", "FLORIDA",
        "GEORGIA", "MARYLAND", "NORTH CAROLINA",
        "SOUTH CAROLINA", "VIRGINIA", "WEST VIRGINIA")
esc <- c("ALABAMA", "KENTUCKY", "MISSISSIPPI", "TENNESSEE")
wsc <- c("ARKANSAS", "LOUISIANA", "OKLAHOMA", "TEXAS")
mtn <- c("ARIZONA", "COLORADO", "IDAHO",
         "MONTANA", "NEVADA", "NEW MEXICO",
         "UTAH", "WYOMING")
pac <- c("ALASKA", "CALIFORNIA", "HAWAII", "OREGON", "WASHINGTON")

#---HELPER FUNCTIONS---
#This is steps 1 and 2, selecting columns and pivoting data
select_and_pivot <- function(tib, name) {
  #Step 1: Select Columns
  result <- tib |>
    select(Area_name, STCOU, ends_with("D")) |>
    rename(area_name = Area_name) |> #rename variable
    pivot_longer(cols = 3:12,
                 names_to = "survey_ID",
                 values_to = name)

  return(result)
}

#This is step 3, where we parse the old column names in survey_ID
# as well as the survey_stat_name column
parse_survey_ID <- function(tib) {
  result <- tib |>
  mutate(year = substr(survey_ID, 8, 9),
         survey_ID = substr(survey_ID, 1, 7)) |>
  #Add 1900 (or 2000) to the year column to get the YYYY instead of YY
  mutate(year = case_when(
    as.numeric(year) >= 26 ~ 1900 + as.numeric(year),
    .default = 2000 + as.numeric(year)
    )
  )

  return(result)
}

#This is steps 4-6, where we split data and did some more parsing of strings
split_by_county_status <- function(tib) {
  #Take the county data and add state column
  result_c <- tib |>
```

```
    slice(grep(pattern = ", \\w\\w", area_name)) |>
    # separate() indicates that it is superseded per help file
    # Using separate_wider_delim() in favor of separate()
    separate_wider_delim(area_name, ",", names = c("county", "state")) |>
    # Trim the whitespace on the state column
    mutate(state = trimws(state))

  #Add custom county class
  class(result_c) <- c("county", class(result_c))

  #Take the non-county data and add division column
  result_nc <- tib |>
    slice(grep(pattern = ", \\w\\w", area_name, invert = TRUE)) |>
    mutate(division = case_when(
    area_name %in% ne ~ "New England",
    area_name %in% ma ~ "Mid-Atlantic",
    area_name %in% enc ~ "East North Central",
    area_name %in% wnc ~ "West North Central",
    area_name %in% sa ~ "South Atlantic",
    area_name %in% esc ~ "East South Central",
    area_name %in% wsc ~ "West South Central",
    area_name %in% mtn ~ "Mountain",
    area_name %in% pac ~ "Pacific",
    .default = "ERROR"
  ))

  #Add custom state class
  class(result_nc) <- c("state", class(result_nc))

  #Return a list of two tibbles
  return(list(result_c, result_nc))
}

#---MAIN FUNCTION CALL---
#This just calls all the helper functions in order
#and returns the final product
process_census_data <- function(url, survey_stat_name = "enrollment") {
  #Quick sanity check to make sure we have a character string
  if (!is.character(url)) {
    stop("url argument must be a character string!")
  }
```

```r
  #Run through the helper functions and return the results
  result <- as_tibble(read.csv(file = url, header = TRUE)) |>
   select_and_pivot(name = survey_stat_name) |>
   parse_survey_ID() |>
   split_by_county_status()

  #Retuns a list of 2 tibbles
  return(result)
}

#---DATA COMBINING FUNCTION---
#This will take two lists of two tibbles each and
# combine them into one list of two tibbles

combine_census_data <- function(x, y)  {
  #Sanity checks: make sure each list is a list of size 2
  if(!is_tibble(x[[1]]) || !is_tibble(x[[2]])) {
    stop("First list argument does not exclusively contain tibbles.")
  }

  if(!is_tibble(y[[1]]) || !is_tibble(y[[2]])) {
    stop("Second list argument does not exclusively contain tibbles.")
  }

  if(length(x) != length(y) || length(x) != 2) {
    stop("Lists must be of length 2. Current length: ", length(x))
  }

  #Combine the first element of each list (county data) together
  result_c <- bind_rows(x[[1]], y[[1]])
  #Combine the second element of each list (non-county data) together
  result_nc <- bind_rows(x[[2]], y[[2]])

  #Return the result datasets!
  return(list(result_c, result_nc))
}
```

Now that we've written our functions, all that's left to do is call them. If everything works properly, we should see a list of two different tibbles: one with county data from 1987-2006, and another with non-county data from 1987-2006. These datasets should be formatted exactly like the example data we used above!

```
#Call the functions we just made!
county_data_test <- process_census_data(url =
                      "https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv",
                      survey_stat_name = "enrollment_count")
state_data_test <- process_census_data(url =
                     "https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv",
                     survey_stat_name = "enrollment_count")

#Combine the two lists together
census_test <- combine_census_data(county_data_test, state_data_test)
```

Let's do a quick sort and take a look at some of the data to see if everything looks okay.

```
#View results
census_test[[1]] |>
  arrange(county, year) |>
  head(n = 20)
```

```
# A tibble: 20 x 6
   county    state STCOU survey_ID enrollment_count  year
   <chr>     <chr> <int> <chr>                <int> <dbl>
 1 Abbeville SC    45001 EDU0101               3941  1987
 2 Abbeville SC    45001 EDU0101               3934  1988
 3 Abbeville SC    45001 EDU0101               3880  1989
 4 Abbeville SC    45001 EDU0101               3834  1990
 5 Abbeville SC    45001 EDU0101               3806  1991
 6 Abbeville SC    45001 EDU0101               3780  1992
 7 Abbeville SC    45001 EDU0101               3804  1993
 8 Abbeville SC    45001 EDU0101               3750  1994
 9 Abbeville SC    45001 EDU0101               3780  1995
10 Abbeville SC    45001 EDU0101               3803  1996
11 Abbeville SC    45001 EDU0101               3821  1997
12 Abbeville SC    45001 EDU0101               3730  1998
13 Abbeville SC    45001 EDU0101               3861  1999
14 Abbeville SC    45001 EDU0102               3927  2000
15 Abbeville SC    45001 EDU0102               3871  2001
16 Abbeville SC    45001 EDU0102               3967  2002
17 Abbeville SC    45001 EDU0152               3812  2003
18 Abbeville SC    45001 EDU0152               3777  2004
19 Abbeville SC    45001 EDU0152               3680  2005
20 Abbeville SC    45001 EDU0152               3616  2006
```

Looks good on the county side. Let's check the non-counties as well.

```
#View results, from the end this time for the sake of it.
census_test[[2]] |>
  arrange(division, area_name, year) |>
  tail(n = 20)
```

```
# A tibble: 20 x 6
   area_name STCOU survey_ID enrollment_count  year division
   <chr>     <int> <chr>                <int> <dbl> <chr>
 1 TEXAS     48000 EDU0101            3209515  1987 West South Central
 2 TEXAS     48000 EDU0101            3236867  1988 West South Central
 3 TEXAS     48000 EDU0101            3282956  1989 West South Central
 4 TEXAS     48000 EDU0101            3268933  1990 West South Central
 5 TEXAS     48000 EDU0101            3382509  1991 West South Central
 6 TEXAS     48000 EDU0101            3464371  1992 West South Central
 7 TEXAS     48000 EDU0101            3535742  1993 West South Central
 8 TEXAS     48000 EDU0101            3601839  1994 West South Central
 9 TEXAS     48000 EDU0101            3670193  1995 West South Central
10 TEXAS     48000 EDU0101            3740260  1996 West South Central
11 TEXAS     48000 EDU0101            3828975  1997 West South Central
12 TEXAS     48000 EDU0101            3891877  1998 West South Central
13 TEXAS     48000 EDU0101            3945367  1999 West South Central
14 TEXAS     48000 EDU0102            3991783  2000 West South Central
15 TEXAS     48000 EDU0102            4059619  2001 West South Central
16 TEXAS     48000 EDU0102            4163447  2002 West South Central
17 TEXAS     48000 EDU0152            4331751  2003 West South Central
18 TEXAS     48000 EDU0152            4405215  2004 West South Central
19 TEXAS     48000 EDU0152            4525394  2005 West South Central
20 TEXAS     48000 EDU0152            4599509  2006 West South Central
```

Looks good here, too. Now that we have a way to easily process the data we are looking for, it's time to start thinking about how we are going to summarize and analyze it.

Using Functions to Summarize Data

We will start by plotting the data based on state as the location. Our plan is to create a function that plots the mean value of `enrollment count` across the years for each `Division`.

```
#make function as directed
plot.state <- function(tib, var_name="enrollment_count") {

#creating an object to change y axis based on var_name
y_label <- paste("Mean", gsub("_", " ", var_name))

tib |> #inputting our intended data frame
    filter(division != "ERROR") |>  #removing error values from division
    group_by(year, division) |>  #grouping by year then divison
    #getting mean of the variable from var_name
    summarise(mean_value = mean(get(var_name), na.rm=T), .groups ="drop")  |>
    ggplot(aes(x=year, y=mean_value, color=division)) +
    #aesthetics of plot, followed by a line plot
    geom_line() +
    labs(x= "Year",
         y = y_label,
         title = "Census Data by Division") +
    scale_color_discrete("Division")

}
```

Next we are going to create a plotting function for the `county` level data. The goal of this function will be for the user to be able to input a state of interest using the 2-letter abbreviated (the default value is NC), determine whether the `top` or `bottom` most counties are the target (default of `top`), and finally an instruction to determine how many of the 'top' or bottom most counties should be investigated (default of 5).

```
plot.county <- function(tib, var_name = "enrollment_count",
                        state.name="NC",
                        top_bottom = "top",
                        n_counties = 5) {

#creating an object to change y axis based on var_name
y_label <- paste("Mean", gsub("_", " ", var_name))

#creating an object to change title based on inputs
plot_title <- paste("Census Data from",
                    top_bottom,
                    n_counties,
                    "Counties in",
                    state.name)
#filter data by state
state_data <- tib |>
  filter(state == state.name)

#calculate mean of the chosen variable (default enrollment_count) for each county.
county_means <- state_data |>
  group_by(county) |>
  summarise(mean_var_name = mean(get(var_name), na.rm=TRUE), .groups="drop")

#using the mean value, sort base on if top or bottom is requested
#then slice based on number of counties requested
sorted_county_means <- if (top_bottom == "top") {
  county_means |>
    arrange(desc(mean_var_name))
      } else {
  county_means |>
    arrange(mean_var_name)
  }

#pull out counties based on chosen rank and n
selected_counties <- sorted_county_means |>
  slice_head (n = n_counties) |>
  pull (county)

#Use initial data then filter from the subset after calculating means
plot_data <- state_data |>
  filter(county %in% selected_counties)

#Plot the values
ggplot(plot_data, aes(x = year, y= get(var_name), color = county)) +
         geom_line() +
         labs(x= "Year",
              y = y_label,
              title = plot_title) +
         scale_color_discrete("County") +
         guides(color= guide_legend(ncol = 2))

}
```

Demonstrating our Functions to Put it Together

We are now going to put all of our functions together to test their use.

The first iteration will be using the initial data we used to create the functions. The first step is to process the data utilizing the `enrollment data` column. We will then combine the 2 data frames to make 1 list.

```
#processing the data using the 2 data sets with our function
county_data <- process_census_data(url =
                 "https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv",
                 survey_stat_name = "enrollment_count")
state_data <- process_census_data(url =
                "https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv",
                survey_stat_name = "enrollment_count")

#using our function to make a list with the 2 data sets
census <- combine_census_data(county_data, state_data)

#indexing to display the first 15 values of the county data
census[[1]] |>
  head(n=15)
```

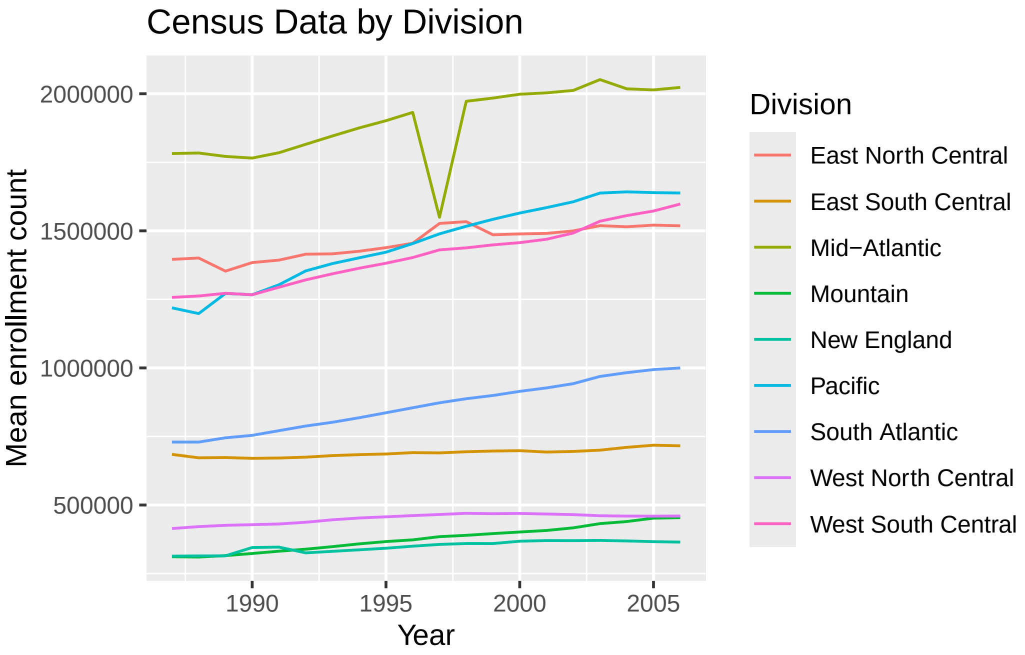
```
# A tibble: 15 x 6
   county  state STCOU survey_ID enrollment_count  year
   <chr>   <chr> <int> <chr>                <int> <dbl>
 1 Autauga AL     1001 EDU0101               6829  1987
 2 Autauga AL     1001 EDU0101               6900  1988
 3 Autauga AL     1001 EDU0101               6920  1989
 4 Autauga AL     1001 EDU0101               6847  1990
 5 Autauga AL     1001 EDU0101               7008  1991
 6 Autauga AL     1001 EDU0101               7137  1992
 7 Autauga AL     1001 EDU0101               7152  1993
 8 Autauga AL     1001 EDU0101               7381  1994
 9 Autauga AL     1001 EDU0101               7568  1995
10 Autauga AL     1001 EDU0101               7834  1996
11 Baldwin AL     1003 EDU0101              16417  1987
12 Baldwin AL     1003 EDU0101              16465  1988
13 Baldwin AL     1003 EDU0101              16799  1989
14 Baldwin AL     1003 EDU0101              17054  1990
15 Baldwin AL     1003 EDU0101              17479  1991
```

```
#indexing to display the last 15 values of the state data to avoid `ERROR` in `division`
census[[2]] |>
  tail(n=15)
```

```
# A tibble: 15 x 6
   area_name STCOU survey_ID enrollment_count  year division
   <chr>     <int> <chr>                <int> <dbl> <chr>
 1 WISCONSIN 55000 EDU0102             879361  2002 East North Central
 2 WISCONSIN 55000 EDU0152             880031  2003 East North Central
 3 WISCONSIN 55000 EDU0152             864652  2004 East North Central
 4 WISCONSIN 55000 EDU0152             875174  2005 East North Central
 5 WISCONSIN 55000 EDU0152             876700  2006 East North Central
 6 WYOMING   56000 EDU0101              99058  1997 Mountain
 7 WYOMING   56000 EDU0101              97115  1998 Mountain
 8 WYOMING   56000 EDU0101              94988  1999 Mountain
 9 WYOMING   56000 EDU0102              92283  2000 Mountain
10 WYOMING   56000 EDU0102              89940  2001 Mountain
11 WYOMING   56000 EDU0102              87897  2002 Mountain
12 WYOMING   56000 EDU0152              87462  2003 Mountain
13 WYOMING   56000 EDU0152              84733  2004 Mountain
14 WYOMING   56000 EDU0152              86420  2005 Mountain
15 WYOMING   56000 EDU0152              85193  2006 Mountain
```

Next we are going to test out our plotting function for the state data frame.

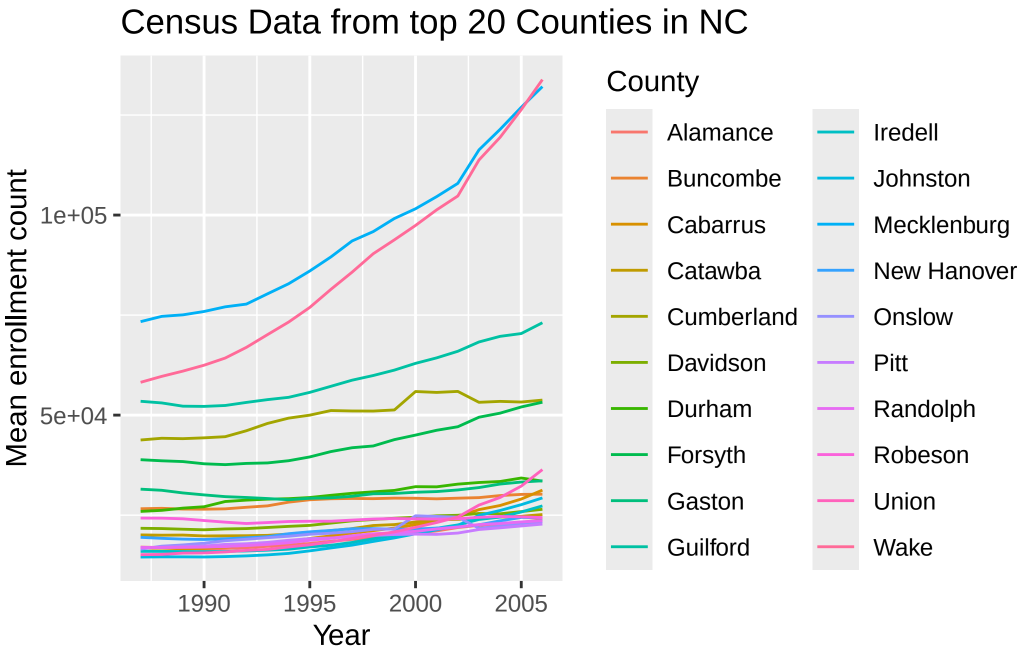
```
plot.state(census[[2]],var_name = "enrollment_count")
```



Finally, let’s test out multiple inputs using our county data.

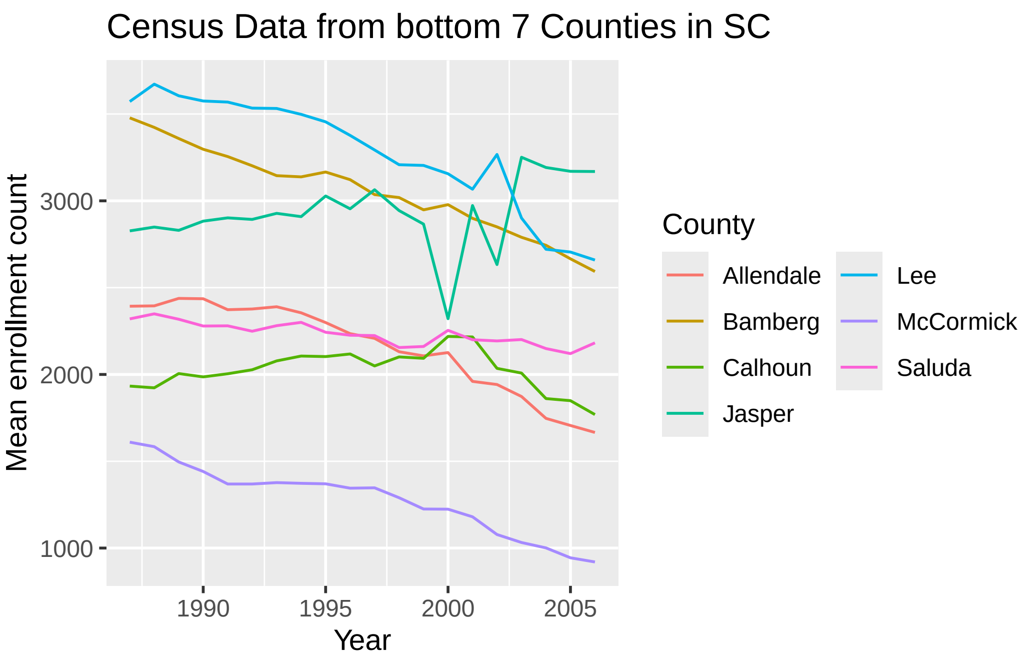
1. Let’s look at the state of “NC” and the top 20 counties for enrollment data

```
plot.county(census[[1]],var_name = "enrollment_count",
                        state.name="NC",
                        top_bottom = "top",
                        n_counties = 20)
```



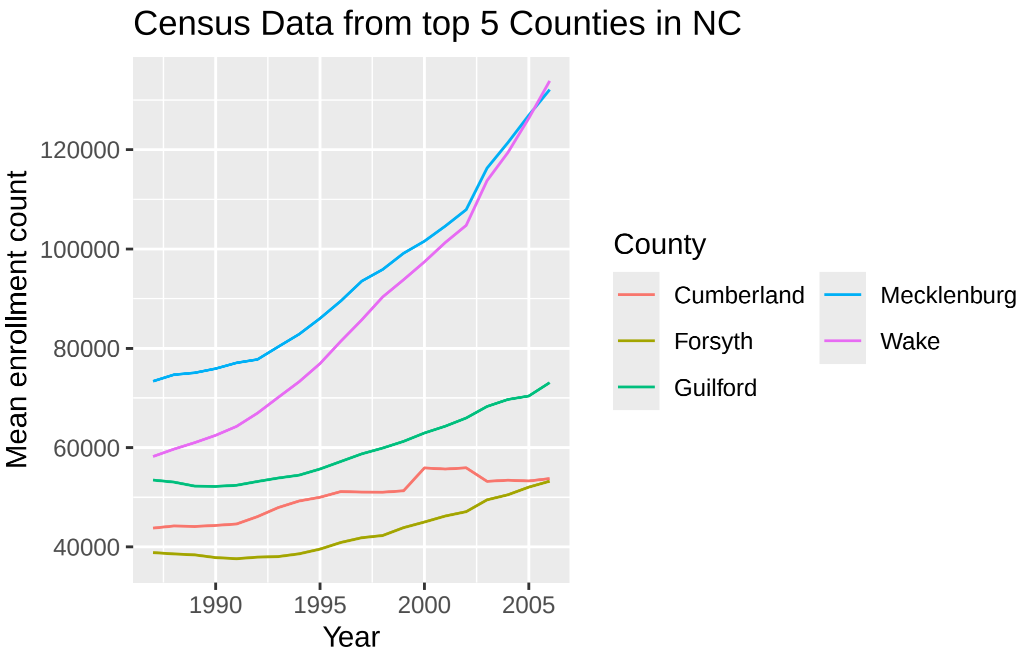
2. Let's look at the state of "SC" and the bottom 7 counties for enrollment data

```
plot.county(census[[1]],var_name = "enrollment_count",
                        state.name="SC",
                        top_bottom = "bottom",
                        n_counties = 7)
```



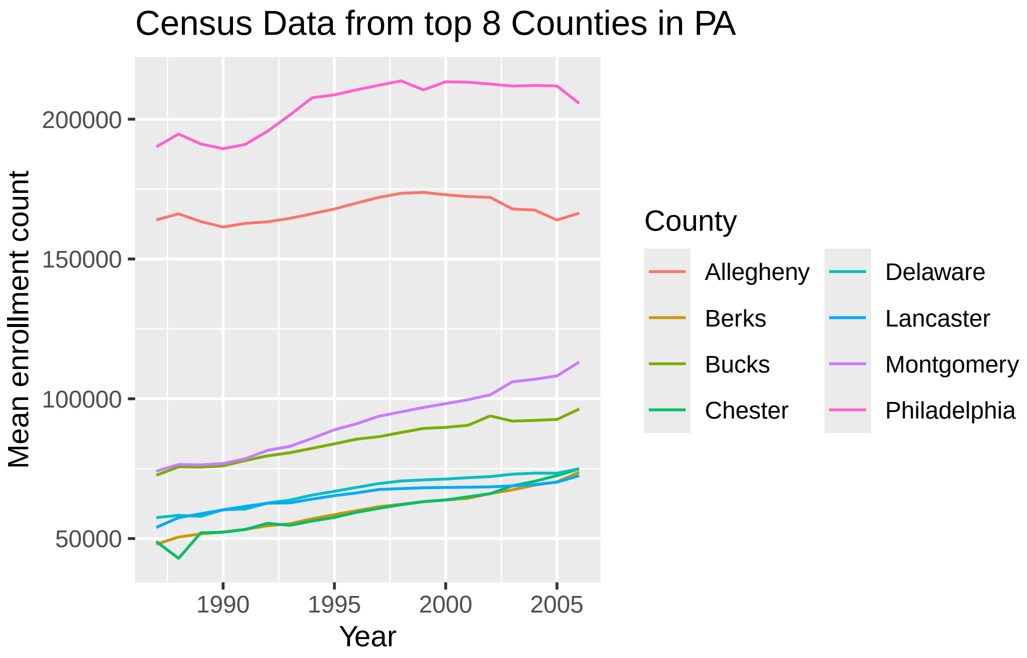
3. Let's now look at whether our defaults are working. To reference back, we chose a default state of "NC" with the top 5 counties

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



4. Finally, let's look at the state of "PA" and the top 8 counties for enrollment data

```
plot.county(census[[1]],var_name = "enrollment_count",
                        state.name="PA",
                        top_bottom = "top",
                        n_counties = 8)
```



Looks like so far our functions are performing as desired!

Now let's utilize different data sets to apply our functions.

```
#run the data processing function for each dataset
data_pst01a <- process_census_data(url =
                 "http://www4.stat.ncsu.edu/~online/datasets/PST01a.csv",
                 survey_stat_name = "population")

data_pst01b <- process_census_data(url =
                 "http://www4.stat.ncsu.edu/~online/datasets/PST01b.csv",
                 survey_stat_name = "population")

data_pst01c <- process_census_data(url =
                 "http://www4.stat.ncsu.edu/~online/datasets/PST01c.csv",
                 survey_stat_name = "population")
```

```r
data_pst01d <- process_census_data(url =
                 "http://www4.stat.ncsu.edu/~online/datasets/PST01d.csv",
                 survey_stat_name = "population")

#now we need to run our data combining function 3 times to get our desired list that contains
#tibbles, 1 for county data and 1 for state data

census_pst_ab <- combine_census_data(data_pst01a, data_pst01b)

census_pst_abc <- combine_census_data(census_pst_ab, data_pst01c)

census_pst_abcd <- combine_census_data(census_pst_abc, data_pst01d)

#indexing to display the first 15 values of the county data
census_pst_abcd[[1]] |>
  head(n=15)
```

```
# A tibble: 15 x 6
   county  state STCOU survey_ID population  year
   <chr>   <chr> <int> <chr>          <int> <dbl>
 1 Autauga AL     1001 PST0151        25508  1971
 2 Autauga AL     1001 PST0151        27166  1972
 3 Autauga AL     1001 PST0151        28463  1973
 4 Autauga AL     1001 PST0151        29266  1974
 5 Autauga AL     1001 PST0151        29718  1975
 6 Autauga AL     1001 PST0151        29896  1976
 7 Autauga AL     1001 PST0151        30462  1977
 8 Autauga AL     1001 PST0151        30882  1978
 9 Autauga AL     1001 PST0151        32055  1979
10 Autauga AL     1001 PST0251        31985  1981
11 Baldwin AL     1003 PST0151        60141  1971
12 Baldwin AL     1003 PST0151        62435  1972
13 Baldwin AL     1003 PST0151        64195  1973
14 Baldwin AL     1003 PST0151        66071  1974
15 Baldwin AL     1003 PST0151        67860  1975
```

```r
#indexing to display the last 15 values of the state data to avoid `ERROR` in `division`
census_pst_abcd[[2]] |>
  tail(n=15)
```

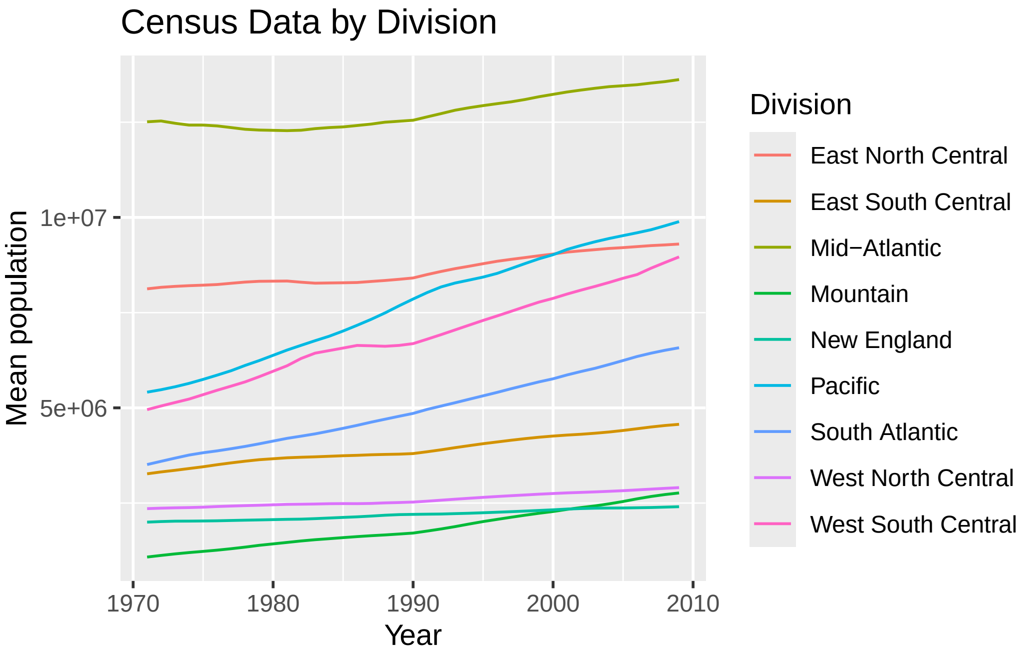
```
# A tibble: 15 x 6
```

```
   area_name STCOU survey_ID population  year division
   <chr>     <int> <chr>          <int> <dbl> <chr>
 1 WISCONSIN 55000 PST0452      5541443  2005 East North Central
 2 WISCONSIN 55000 PST0452      5571680  2006 East North Central
 3 WISCONSIN 55000 PST0452      5601571  2007 East North Central
 4 WISCONSIN 55000 PST0452      5627610  2008 East North Central
 5 WISCONSIN 55000 PST0452      5654774  2009 East North Central
 6 WYOMING   56000 PST0452       493958  2000 Mountain
 7 WYOMING   56000 PST0452       492982  2001 Mountain
 8 WYOMING   56000 PST0452       497069  2002 Mountain
 9 WYOMING   56000 PST0452       499189  2003 Mountain
10 WYOMING   56000 PST0452       502988  2004 Mountain
11 WYOMING   56000 PST0452       506242  2005 Mountain
12 WYOMING   56000 PST0452       512841  2006 Mountain
13 WYOMING   56000 PST0452       523414  2007 Mountain
14 WYOMING   56000 PST0452       532981  2008 Mountain
15 WYOMING   56000 PST0452       544270  2009 Mountain
```

It looks like this data is older than our previous data. The year of the surveys ranges from 1971 to 2009. Additionally, this is from a different survey, with a prefix of PST. Now that we read in the 4 data sets and combined them, separating out the `county` and `state` data, let's make some plots.

Our first plot will contain our data from our `state` data.

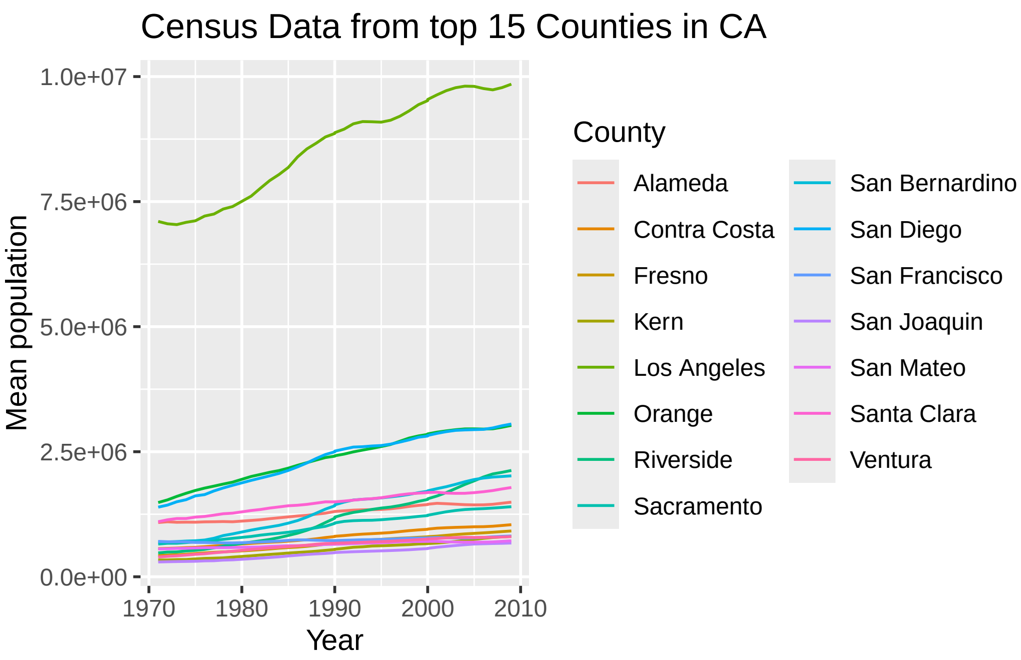
```
plot.state(census_pst_abcd[[2]],var_name = "population")
```



Now we are going to run some plots utilizing our `county` data

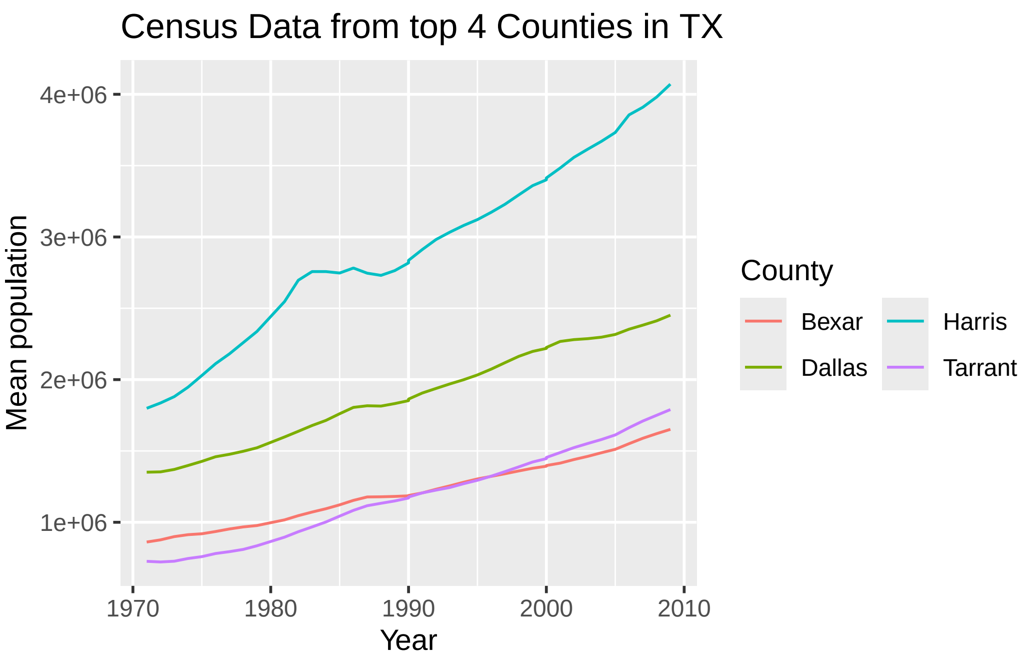
1. Here is the data from California with the top 15 counties of enrollment means

```
plot.county(census_pst_abcd[[1]],var_name = "population",
                        state.name="CA",
                        top_bottom = "top",
                        n_counties = 15)
```



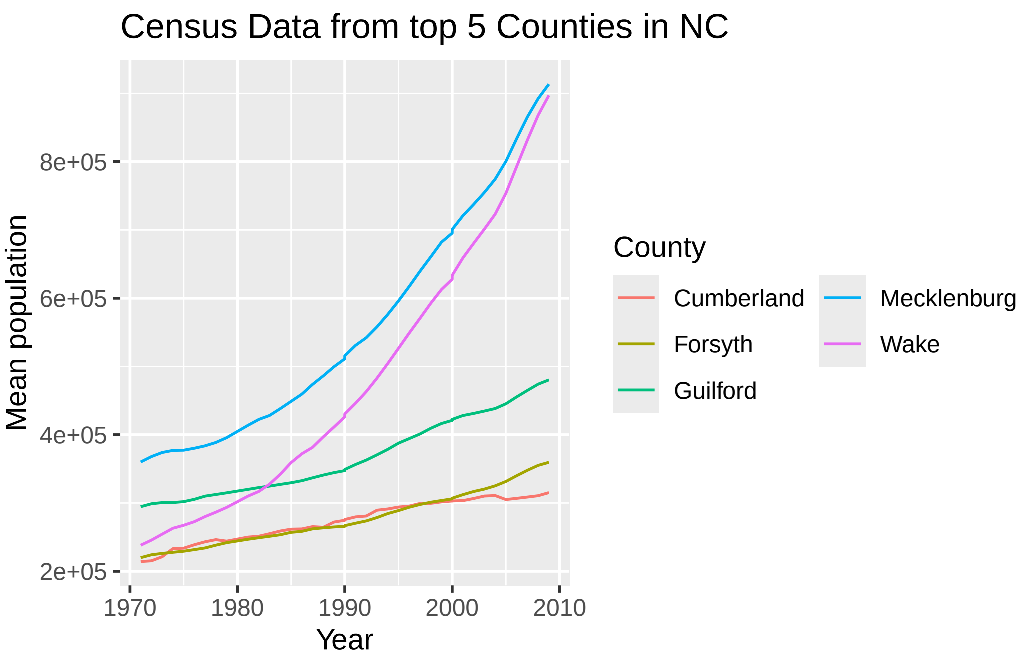
2. Next, let's look at the top 4 counties from Texas

```
plot.county(census_pst_abcd[[1]],var_name = "population",
                        state.name="TX",
                        top_bottom = "top",
                        n_counties = 4)
```



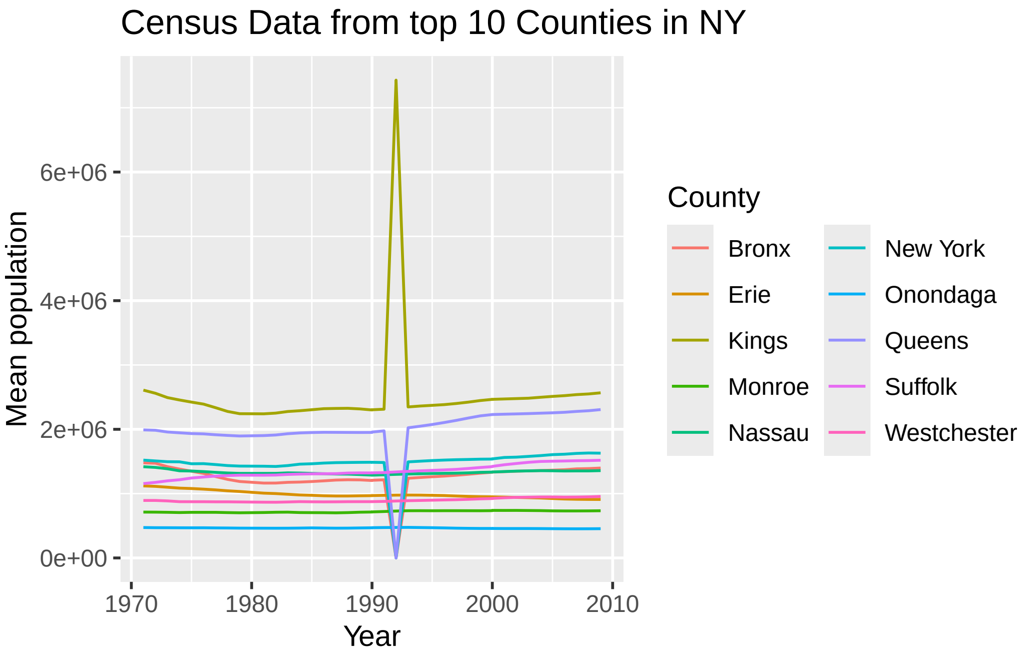
3. Here is the plot using our defaults in our plotting function. To reference back, we chose a default state of "NC" with the top 5 counties

```r
plot.county(census_pst_abcd[[1]], var_name = "population")
```



4. Finally, we are going to look at the top 10 counties, based on enrollment mean, from New York

```
plot.county(census_pst_abcd[[1]],var_name = "population",
                            state.name="NY",
                            top_bottom = "top",
                            n_counties = 10)
```


Census Data from top 10 Counties in NY
Mean population
6e+06
4e+06
2e+06
0e+00
Year
County
Bronx
Erie
Kings
Monroe
Nassau
New York
Onondaga
Queens
Suffolk
Westchester