ST558 Project 1

Kamlesh Pandey

9/6/2022

Contents

1 Data Processing

This code chunk is responsible for manually pre-processing the data. This pre-processing involve storing data into a tibble from a URL using csv reader function from R package readr. *After saving data into a tibble, selecting relevant columns, renaming and pivot the data along the columns ending with 'D'. The final step in the code block is to use sub-string function to extract strings from the enrollments and convert them into Survey, values and, date feature columns.*

The most important manipulation step in this data processing code is to take sub strings from the enrollment feature and convert them into years (YYYY format). For this step, using a loop through the length of the sub-string and adding year '20' or '19' as prefix based on the data can be a helpful method, but not efficient as it require more memory storage compared to any other matrix method.

```
#Data Pre-processing
library(readr)
library(readr)
library(dplyr)
library(tidyr)

sheet1 <- readr::read_csv("https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv")
sheet1 <- sheet1 %>% select(ends_with("D"), 'Area_name', 'STCOU') %>%
                     rename(area_name = 'Area_name')%>%
                     mutate_at('STCOU', as.numeric)


# Converting data into long format
sheet1 <- sheet1 %>%
          pivot_longer(cols = ends_with('D'),
                       names_to = 'Item_id', values_to = 'Value')

# Year Extraction
last_Two_Digit      <- as.numeric(substr(sheet1$Item_id, 8,9))
sheet1$Survey       <- substr(sheet1$Item_id,1,3)
sheet1$ValueType    <- substr(sheet1$Item_id,4,7)
sheet1$Year         <- NA

# Loop to get year column in YYYY format
for (i in 1:length(last_Two_Digit)){
  if (last_Two_Digit[i] < 22){
    sheet1[i,7] <- paste0(200, last_Two_Digit[i])
  }
  else if (last_Two_Digit[i] > 22){
    sheet1[i,7] <- paste0(19, last_Two_Digit[i])
  }

}
```

1.1 Creating two data set

This code block is responsible for splitting the dataset into county and non-county datasets based on a pattern present in 'state_name' feature column in the original tibble data. A class of type county and non county also added into county and non-county type dataset for proving better flexibility in writing custom functions for later use.

```
# county data
countyData_index <- grep(pattern = ", \\w\\w", sheet1$area_name)
countyDataset <- sheet1[countyData_index,]

# Non County dataset
nonCountyDataset <- sheet1[-countyData_index,]

# adding class to both datasets
class(countyDataset) <- c("county", class(countyDataset))
class(nonCountyDataset) <- c("state", class(nonCountyDataset))
```

1.2 Adding State and divisions

This code block is responsible for adding a features columns either state's abbreviation (i.e. for Arizona, AZ) or division to each county and non county dataset respectively for better manipulation and visualization in later use. For county a new column State *added and for non county dataset a division in the county and non-county datasets respectively.*

```
# For county dataset
countyDataset$state <-  substr(countyDataset$area_name,
                               nchar(countyDataset$area_name)-2+1,
                               nchar(countyDataset$area_name))

# For non county dataset
nonCountyDataset <- within(nonCountyDataset, {
    division = "NO"
    division[area_name %in% c("UNITED STATES")] = 'ERROR'
    division[area_name %in% toupper(c('Connecticut', 'Maine', 'Massachusetts', 'New Hampshire', 'Rhode I
    division[area_name %in% toupper(c('New Jersey', 'New York', 'Pennsylvania'))] = "Mid-Atlantic"
    division[area_name %in% toupper(c('Illinois', 'Indiana', 'Michigan', 'Ohio','Wisconsin'))] = "East N
    division[area_name %in% toupper(c('Iowa', 'Kansas', 'Minnesota', 'Missouri', 'Nebraska', 'North Dako
    division[toupper(area_name) %in% toupper(c('Delaware', 'Florida', 'Georgia', 'Maryland', 'North Caro
    division[area_name %in% toupper(c('Alabama', 'Kentucky', 'Mississippi',  'Tennessee'))] = 'East Sout
    division[area_name %in% toupper(c('Arkansas', 'Louisiana', 'Oklahoma',  'Texas'))] = 'West South Cen
    division[area_name %in% toupper(c('Arizona', 'Colorado', 'Idaho', 'Montana', 'Nevada', 'New Mexico'
    division[area_name %in% toupper(c('Alaska', 'California', 'Hawaii', 'Oregon', 'Washington'))] = 'Pac
})
```

2 Requirements

This code block is responsible for writing functions to add more control in our data wrangling. This block has five main code block responsible for writing functions for all steps involved in first code block.

1. Function 1 is responsible taking a URL and saving the dataset in a tibble form and later performing column selection and pivoting the whole dataset.
2. Function 2 is responsible for adding Survey, Value and Year feature columns for better manipulation.
3. Function 3 is responsible for adding state feature column in case of a county dataset
4. Function 4 is responsible for adding division feature column in case of a non-county dataset
5. Function 5 is responsible for creating index for county and non county dataset from a given dataset based on a pattern and later calling function 3 and 4.
6. *Wrapper function* this function is responsible for wrapping all functions inside one function. So on calling the wrapping function, all other functions will be called eventually.

```
library(dplyr)

#WRITING FUNCTIONS-----------------------------------------------------------------

# function 1
function_for_step_1_2 <- function( col_names = c('D','Area_name', 'STCOU'), url){
  df     <- readr::read_csv(url)
  sheet2 <- df %>% select(ends_with(col_names[1]),col_names[2:3] )
  sheet2 <- sheet2 %>% rename( area_name =  col_names[2])
  sheet2 <- sheet2 %>% mutate_at(col_names[3], as.numeric)

  # converting to pivot_long

  sheet2 <- sheet2 %>% pivot_longer(cols = ends_with(col_names[1]),
                                    names_to = 'Item_Id',
                                    values_to = 'Values')

  return(sheet2)
}
```

```r
# function 2
function_for_step_2_3 <- function(outputSheet){

  # Extracting year from the dataset
  last_Two_Digit        <-  as.numeric(substr(outputSheet$Item_Id, 8,9))
  outputSheet$Survey    <-  substr(outputSheet$Item_Id,1,3)
  outputSheet$ValueType <-  substr(outputSheet$Item_Id,4,7)

  # create a empty Year feature col to store the output from concatenation
  outputSheet$Year      <-  NA

  # run through the length of a vector created by extracting two digits from the enrollment feature col
  for (i in 1:length(last_Two_Digit)){

  # if two digit less than 22 make them years of 21st centure
  if (last_Two_Digit[i] < 22){
    outputSheet[i,7] <- as.numeric(paste0(200, last_Two_Digit[i]))
  }

  # if greater then 22, make them years of 19th century
  else if (last_Two_Digit[i] > 22){
    outputSheet[i, 7] <- as.numeric(paste0(19, as.character(last_Two_Digit[i])))
    }
  }
  return(outputSheet)
}

# function 3

function_for_step_5 <- function(dataset){

  # For county tibble
    dataset$state <- substr(dataset$area_name,
                            nchar(dataset$area_name)-2+1,
                            nchar(dataset$area_name))

  return(dataset)
}

# function 4
function_for_step_6 <- function(dataset){
    # For non-county tibble

    dataset <- within(dataset,{
      division = "NO"
      division[toupper(area_name) %in% c("UNITED STATES")] = 'ERROR'
      division[toupper(area_name) %in% toupper(c('Connecticut', 'Maine', 'Massachusetts', 'New Hampshir
      division[toupper(area_name) %in% toupper(c('New Jersey', 'New York', 'Pennsylvania'))] = "Mid-Atl
      division[toupper(area_name) %in% toupper(c('Illinois', 'Indiana', 'Michigan', 'Ohio','Wisconsin'))
      division[toupper(area_name) %in% toupper(c('Iowa', 'Kansas', 'Minnesota', 'Missouri', 'Nebraska',
      division[toupper(area_name) %in% toupper(c('Delaware', 'Florida', 'Georgia', 'Maryland', 'North C
      division[toupper(area_name) %in% toupper(c('Alabama', 'Kentucky', 'Mississippi',  'Tennessee'))]
      division[toupper(area_name) %in% toupper(c('Arkansas', 'Louisiana', 'Oklahoma',  'Texas'))] = 'We
      division[toupper(area_name) %in% toupper(c('Arizona', 'Colorado', 'Idaho', 'Montana', 'Nevada', '
      division[toupper(area_name) %in% toupper(c('Alaska', 'California', 'Hawaii', 'Oregon', 'Washingto
    })
  return(dataset)
}

#function 5
function_for_step_4 <- function(dataset){

  # get the index
  countyData_index <- grep(pattern = ", \\w\\w", dataset$area_name)

  # select dataset based on the county index
  county_2    <- dataset[countyData_index,]

  # select rest of the data as non county
  nonCounty_2 <- dataset[-countyData_index,]

  finalCounty    <- function_for_step_5(dataset = county_2)

  finalnonCounty <- function_for_step_6(dataset = nonCounty_2 )

  # adding class to both datasets

  class(finalCounty)    <- c('county', class(finalCounty))
  class(finalnonCounty) <- c('state', class(finalnonCounty))

  return(list(county = finalCounty, nonCounty = finalnonCounty))
}


#Wrapper function
my_wrapper           <- function(URL, default_var_type = c('D','Area_name', 'STCOU') ){

  opt_for_step_1_2   <- function_for_step_1_2(col_names = default_var_type, url = URL)

  opt_for_step_2_3   <- function_for_step_2_3 (opt_for_step_1_2)

  opt_for_step_4_5_6 <- function_for_step_4(opt_for_step_2_3)

return (opt_for_step_4_5_6)
}
```

2.0.1 Combining both dataset

This code is responsible for combining any two dataset provided as arguments into a function. This function will take two datasets as input arguments and will combine the both as a single dataset either as a county or a non-county.

```
combined_dataset <- function ( df1, df2 ){

  df <- dplyr::bind_rows(df1,df2)
return (df)
}
```

2.0.2 A Function for Plot (using ggplot library)

This code block is responsible for modifying plot functions as per our county and non-county dataste. The class created previously will help in creating a customized graph using ggplot *for both county and non county dataset generated after combining dataset from the previous function. The plot will be customised for each county and state type of class.*

```
library(ggplot2)

#For non county dataset
plot.state <- function(df, var_name = 'Values'){
  df       <- df %>% select('Year', 'division', 'Values') %>%
                     filter(division != 'ERROR') %>%
                     dplyr::group_by(Year, division) %>%
                     summarise(Mean = mean(get('Values')))

  plot     <- ggplot(df, aes(x = df$Year, y = df$Mean, color = division)) + geom_line() +
                               xlab('Year') + ylab('Mean Value of Enrollment') +
                               ggtitle('Year vs Mean Value of Enrollment for NON COUNTY')

return (list(data = df, plot = plot))
}



# For  county dataset
plot.county <- function(df, var_name = 'Values', state_of_interest = 'IL', top_or_bottom = 'top', n = 5

  # creating a copy for final filtering
  df_copy <- df

  # To filter out the state of interest
  df <- df %>% filter(state == state_of_interest) %>%
               select('area_name', 'state', 'Values', 'Year') %>%
               group_by (area_name)  %>%
               summarise(Mean = mean(get(var_name)))

  # conditioning for the top or bottom n

  # if top, arrange the dataset in descending mean
  if (top_or_bottom == 'top') {
    df <- dplyr::arrange(df, desc(Mean))
  }

  # if bottom selected, arrange the dataset in ascending mean
  else if (top_or_bottom == 'bottom'){
    df <- df %>% arrange(Mean)
  }
  # if any other keyword provided as an input, throw an error to the user
  else {
    cat ('Error!! .........Pleae select either TOP or BOTTOM')
  }
  filter_df <- head(df,n)

  finalDf <- df_copy %>% filter(area_name == filter_df$area_name)

  plot <- ggplot(finalDf, aes(x = Year, y = get(var_name), color = area_name)) + geom_line() +
                        xlab('Year') +
                        ylab('Values of Enrollment') +
                        ggtitle('Year vs Values for COUNTY')

return (list(countyTibble = finalDf, plot = plot ))

}
```

3 Putting it All Together

The wrapper function will call all other functions created previously. The combine function will eventually add the outputs from wrapper function into either county or non county type.

- Run your data processing function on the two enrollment URLs given previously, specifying an appropriate name for the enrollment data column.

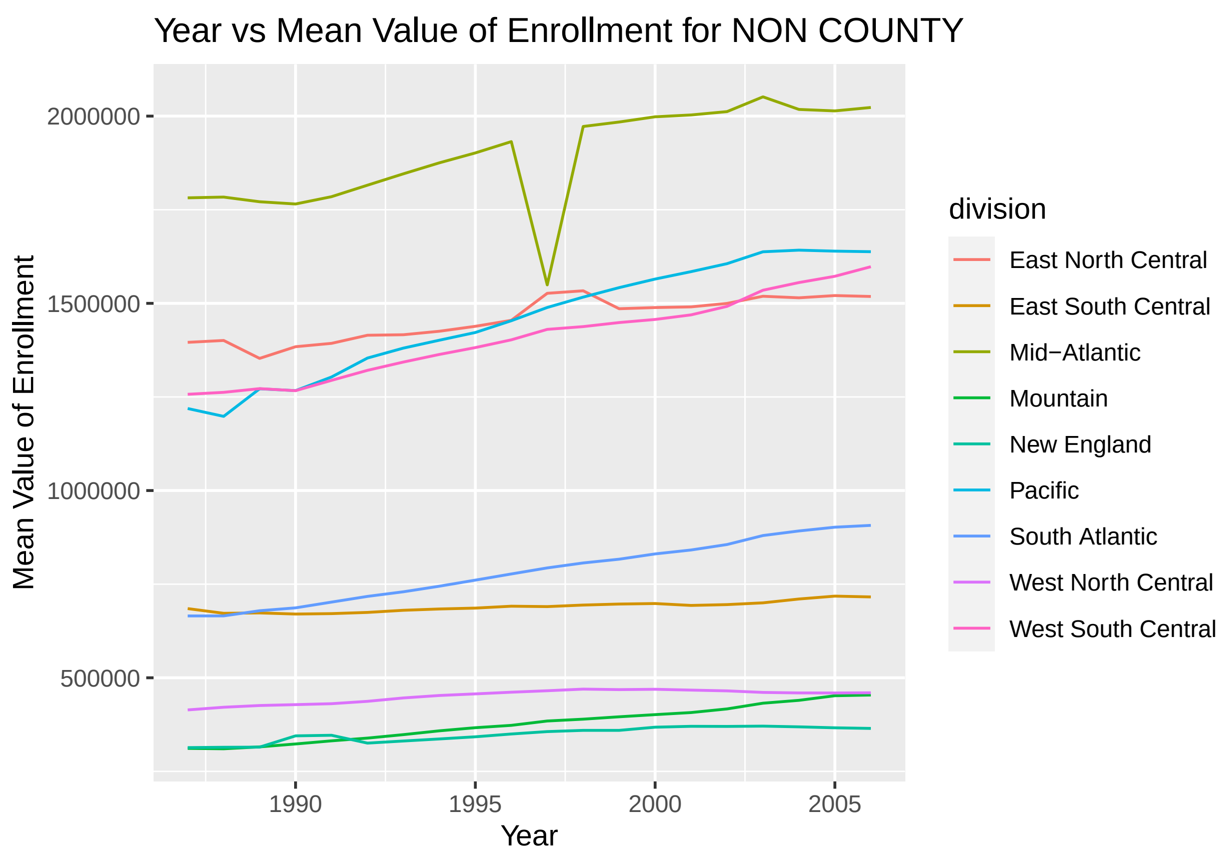
```
# Generating final  from URLs

url1 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/EDU01a.csv')
url2 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/EDU01b.csv')

# Retrieving county dataset

url1_countyData = url1$county
url2_countyData = url2$county


# Retrieving non county dataset
url1_nonCountyData = url1$nonCounty
url2_nonCountyData = url2$nonCounty
```

- Run your data combining function to put these into one object (with two data frames)

```
# combining county and non county dataset respectively
combined_county    <- combined_dataset(url1_countyData, url2_countyData)
combined_nonCounty <- combined_dataset(url1_nonCountyData, url2_nonCountyData)
```

- Use the plot function on the state data frame

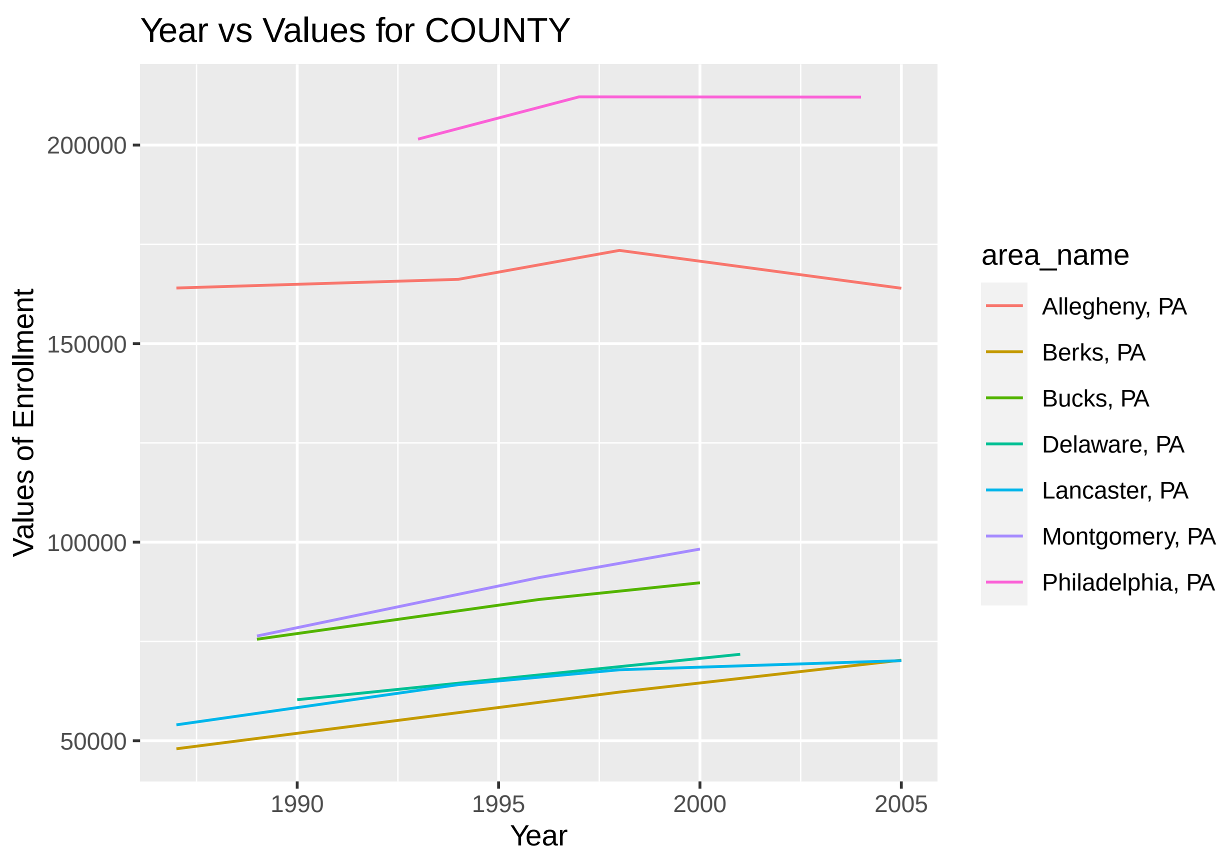
```
# applying plot function created for "STATE" class

state_plot <- plot(combined_nonCounty)
state_plot$plot
```



- Use the plot function on the county data frame

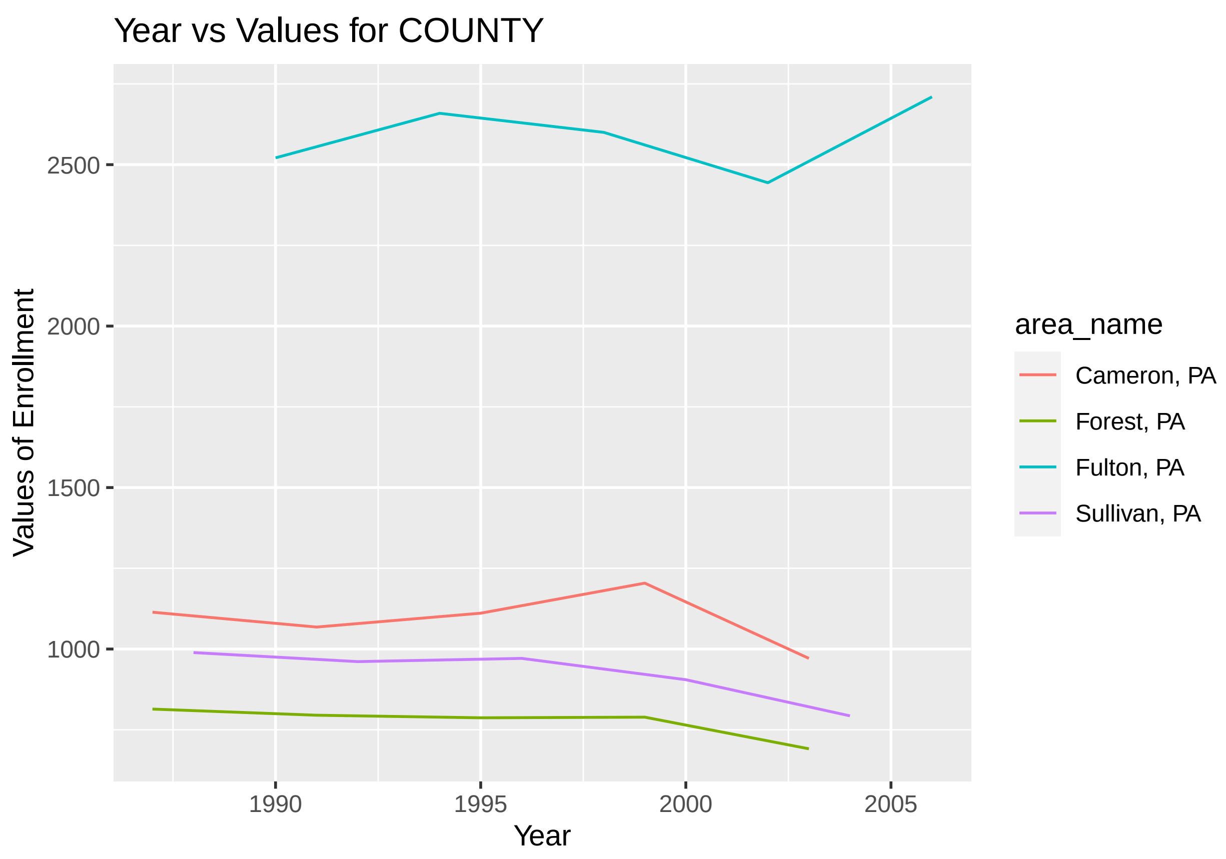
1. Once specifying the state to be "PA", the group being the top, the number looked at being 7

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or
county_plot <- plot(df = combined_county, var_name = 'Values', state_of_interest = 'PA', top_or_bottom =
county_plot$plot
```



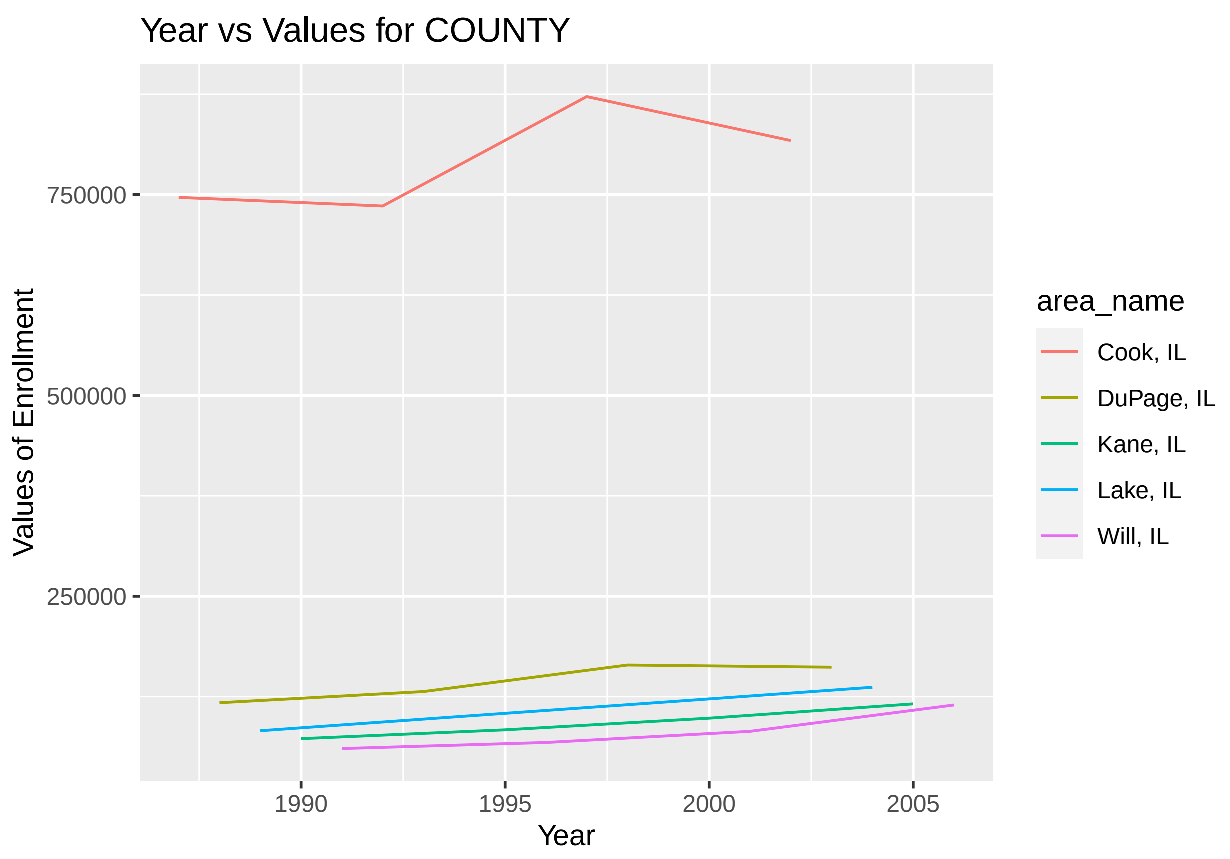
2. Once specifying the state to be "PA", the group being the bottom, the number looked at being 4

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

county_plot <- plot(df = combined_county, var_name = 'Values', state_of_interest = 'PA', top_or_bottom =

county_plot$plot
```



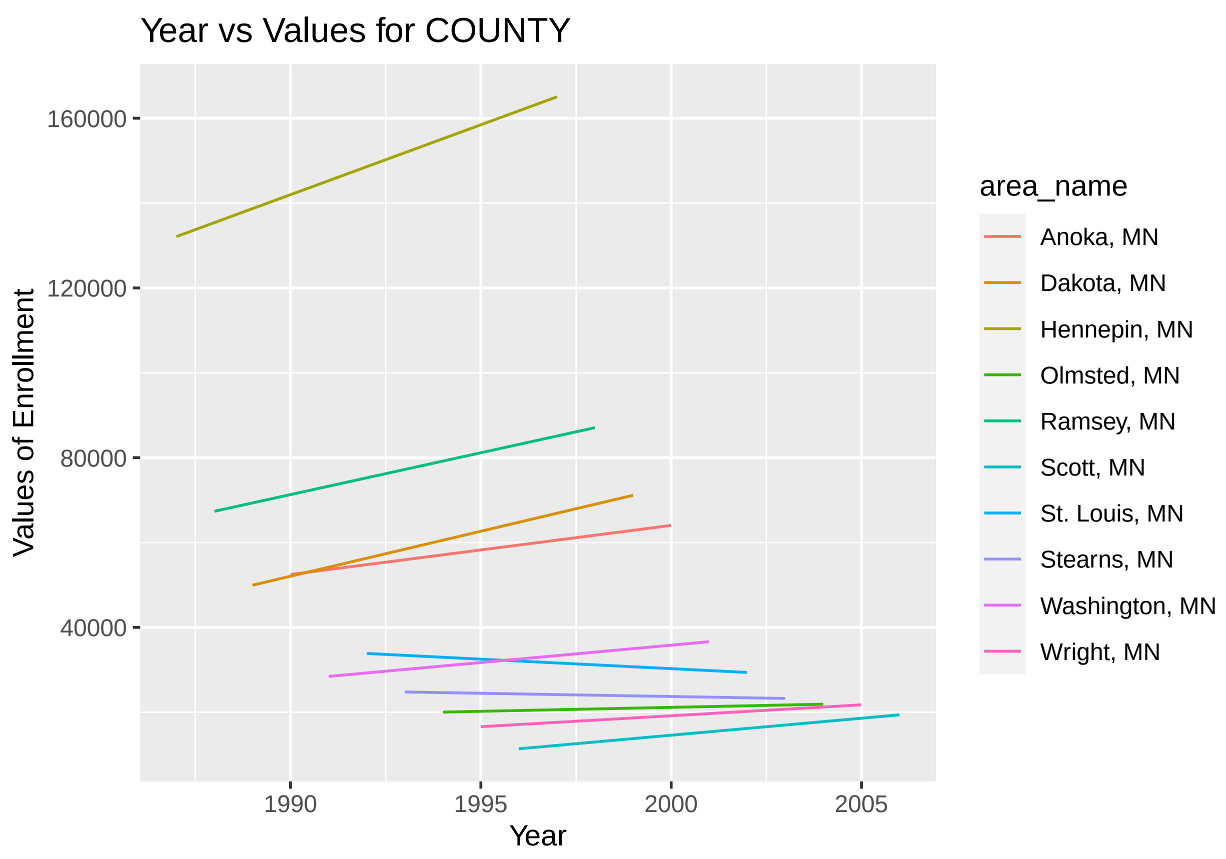
3. Once without specifying anything (defaults used)

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

county_plot <- plot( df = combined_county)

county_plot$plot
```



4. Once specifying the state to be "MN", the group being the top, the number looked at being 10

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

county_plot <- plot( df = combined_county, var_name = 'Values', state_of_interest = 'MN', top_or_bottom

county_plot$plot
```



Adding multiple dataset from the URLs provided in the project, and using wrapper and combine function to generate two separate county and non-county dataset.

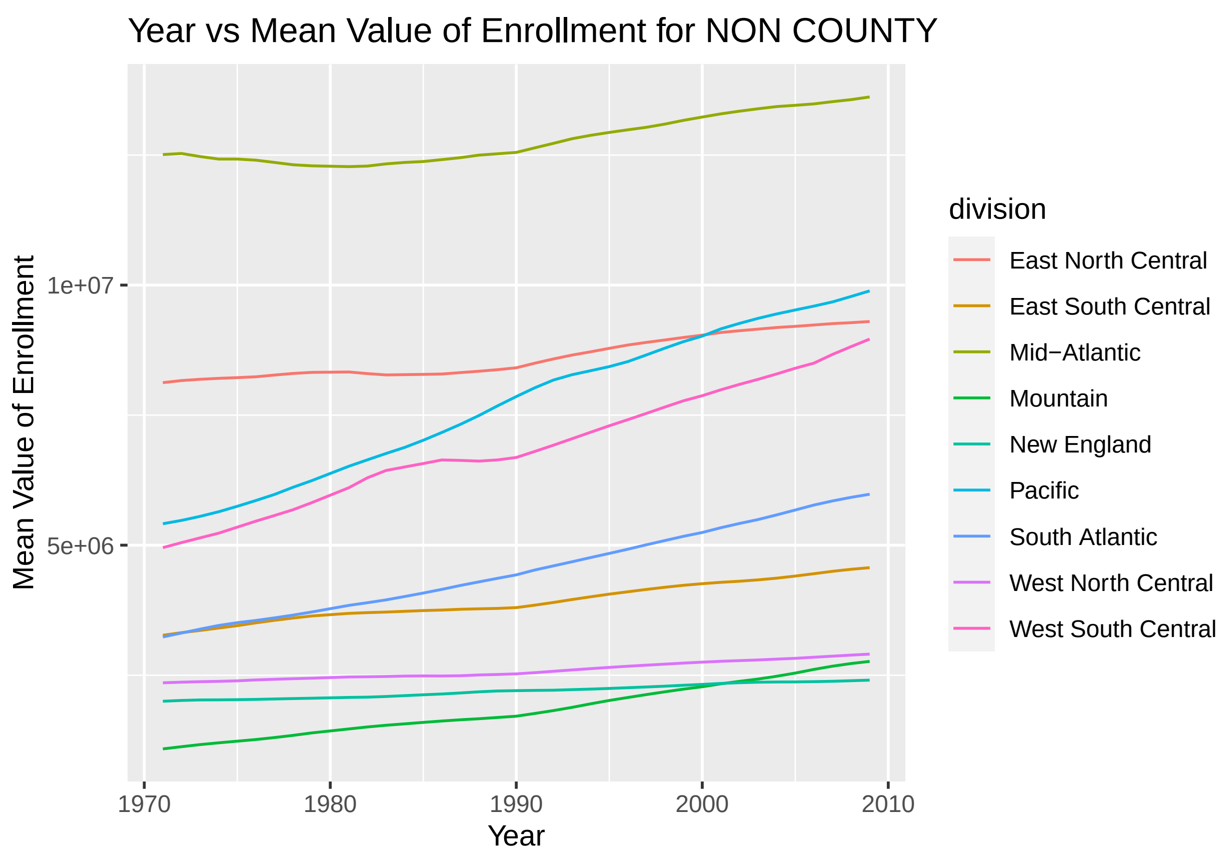
```
url3 <- 'https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv'
url4 <- 'https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv'
url5 <- 'https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv'
url6 <-  'https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv'


url3 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/PST01a.csv')
url4 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/PST01b.csv')
url5 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/PST01c.csv')
url6 <-  my_wrapper(URL = 'https://www4.stat.ncsu.edu/~online/datasets/PST01d.csv')

# Retrieving county dataset

url3_countyData = url3$county
url4_countyData = url4$county
url5_countyData = url5$county
url6_countyData = url6$county

# Retrieving non county dataset
url3_nonCountyData = url3$nonCounty
url4_nonCountyData = url4$nonCounty
url5_nonCountyData = url5$nonCounty
url6_nonCountyData = url6$nonCounty
```

```r
# COMBINED DATASET OF 4 SHEETS FROM URL
combined_county_1     <- combined_dataset(url3_countyData, url4_countyData)
combined_county_2     <- combined_dataset(url5_countyData, url6_countyData)
combined_county_final <- combined_dataset(combined_county_1, combined_county_2)

# COMBINED NON - DATASET OF 4 SHEETS FROM URL
combined_nonCounty_1     <- combined_dataset(url3_nonCountyData, url4_nonCountyData)
combined_nonCounty_2     <- combined_dataset(url5_nonCountyData, url6_nonCountyData)
combined_nonCounty_final <- combined_dataset(combined_nonCounty_1, combined_nonCounty_2)
```

- Use the plot function on the state data frame

All functions generated earlier are used in the subsequent questions

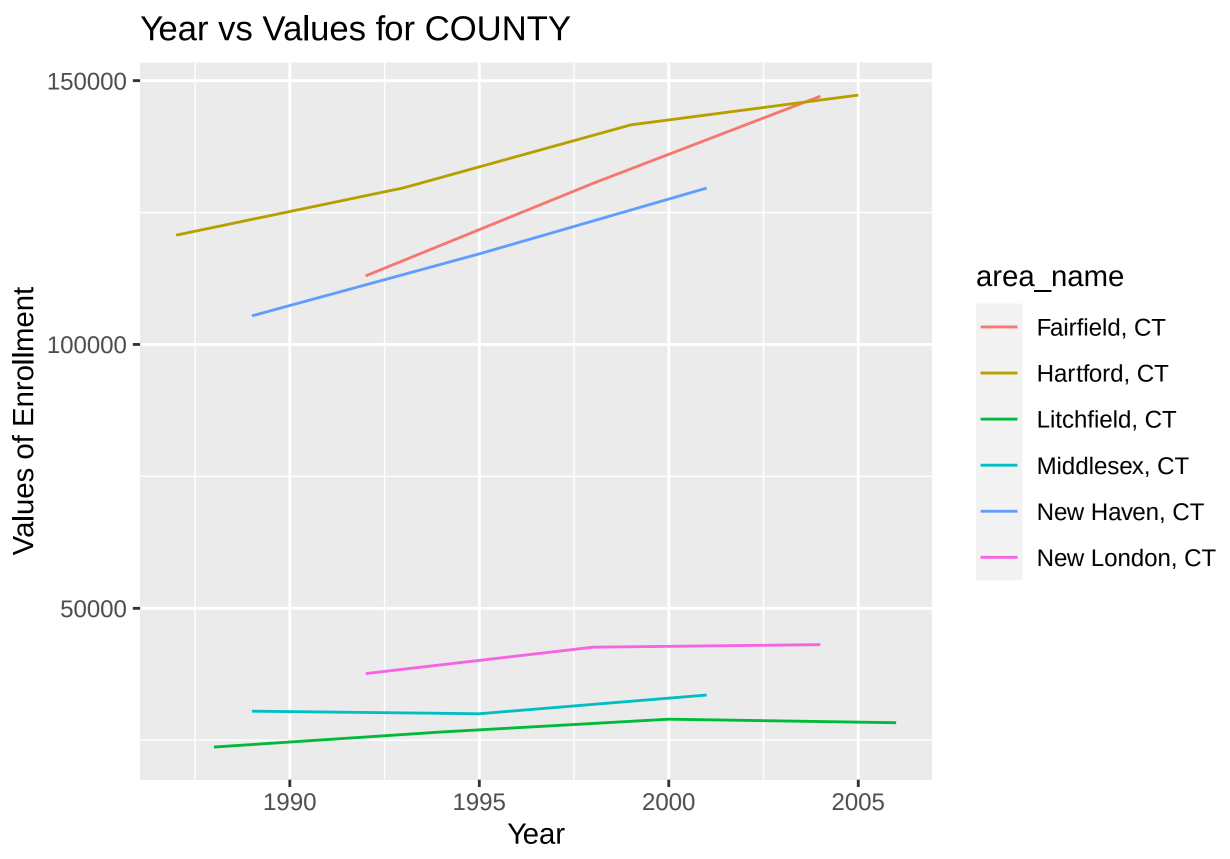
```r
# applying plot function created for "STATE" class

combined_plot <- plot(combined_nonCounty_final)

combined_plot$plot
```



- Use the plot function on the county data frame

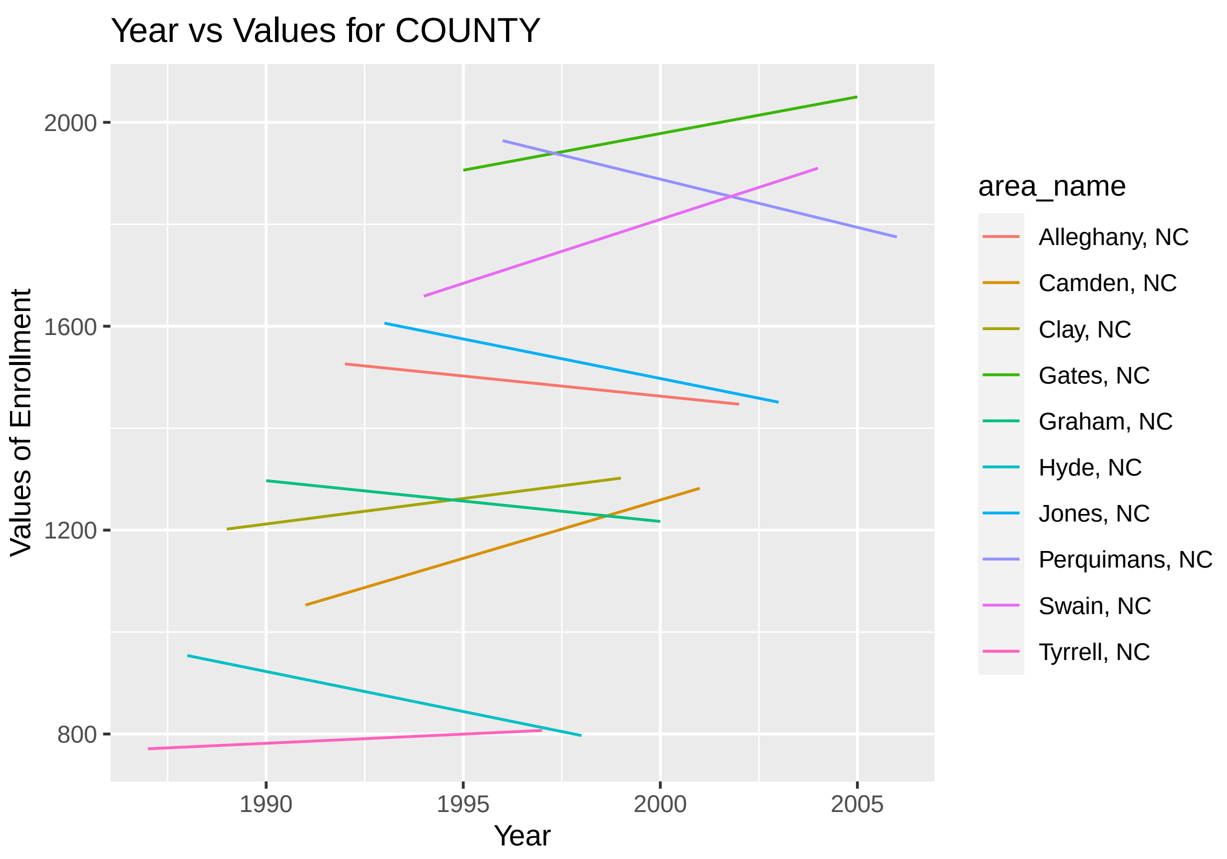
1. Once specifying the state to be "CT", the group being the top, the number looked at being 6

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

combined_plot <- plot( df = combined_county, var_name = 'Values', state_of_interest = 'CT', top_or_bott

combined_plot$plot
```



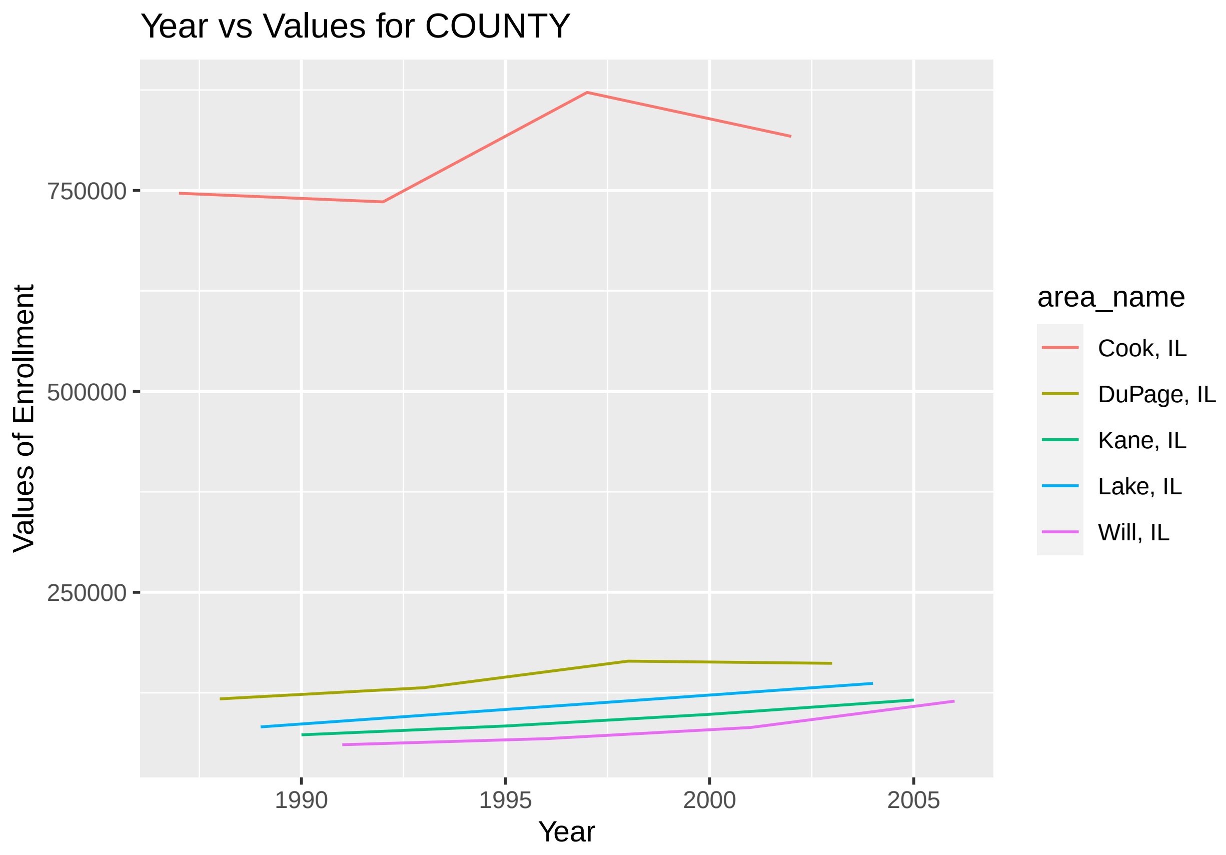
2. Once specifying the state to be "NC", the group being the bottom, the number looked at being 10

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

combined_plot <- plot( df = combined_county, var_name = 'Values', state_of_interest = 'NC', top_or_bott

combined_plot$plot
```



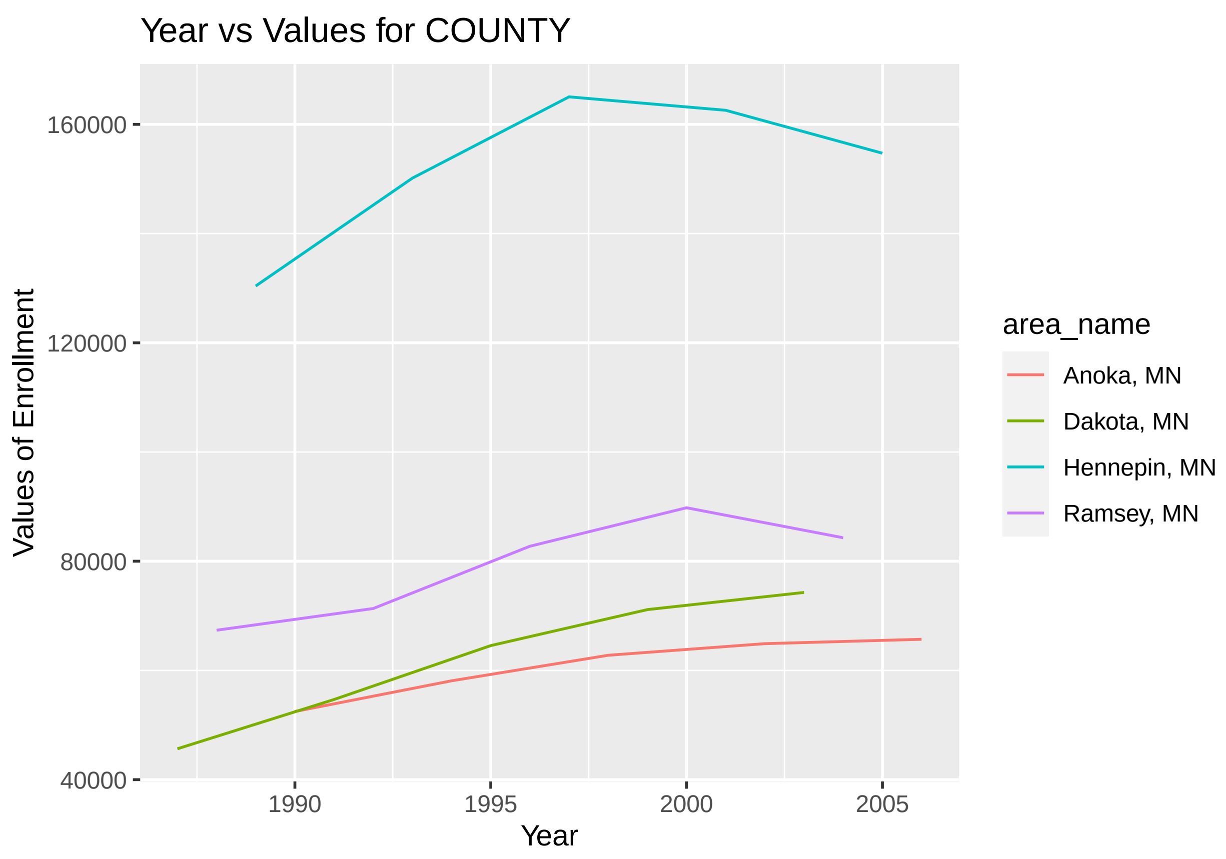
3. Once without specifying anything (defaults used)

```
# applying plot function created for "STATE" class based on the default input arguments

combined_plot <- plot( df = combined_county)

combined_plot$plot
```



4. Once specifying the state to be "MN", the group being the top, the number looked at being 4

```
# applying plot function created for "COUNTY" class based on the users choice of showing either top or

combined_plot <- plot( df = combined_county, var_name = 'Values', state_of_interest = 'MN', top_or_bott

combined_plot$plot
```


Year vs Values for COUNTY
Values of Enrollment
160000
120000
80000
40000
1990
1995
2000
2005
Year
area_name
Anoka, MN
Dakota, MN
Hennepin, MN
Ramsey, MN