Guide on reproducing BRFSS graphic

Xianbin Xu

8/24/2022

Introduction

This R Markdown file replicates various graphics from Behavioral Risk Factor Surveillance System data. It contains specific instructions and replicable codes.

Running Environment

This RMD (R Markdown) file had been knitted into a PDF file for easier reading. PDF file and R Markdown file(end with .rmd) are all included in the directory under the same name. To reproduce the code, open the R Markdown in RStudio, then run the whole markdown file.

To get a LaTeX file of the document, add "keep_tex: true" below the pdf_output: line, in the header section of R Markdown. Then, knit the whole markdown file.

Make sure the file is located in /Graphic_Reproduction_Folder, with directory BRFSS_Data in the same directory containing /Graphic_Reproduction_Folder.

Make sure the following files are in place at BRFSS_Data folder:

> Processed_BRFSS_1999_to_2020.csv. This can be gained by running BRFSS_Fractured_Download_Process.R under Default parameters.

To use processed BRFSS Data of a different year range, change the parameter at the head of BRFSS_Fractured_Download_Process.R. See comment of that script for specific parameters to be used.

The following packages are used:

```
library("readxl") ## To read Excel
library("dplyr") ## For better data processing
library("haven") ## To read and export DTA Files
library(tidyr) ## Tidy functions
library(cdlTools) ## To corresponde FIPS code to state names/abbreviations
library(ggplot2) ## To have easier time plotting in R
library(gridExtra) ## To place two plots by each other
```

use install.packages() to install if you haven't installed already.

And read the file:

```
Data = read.csv(file = "./../BRFSS_Data/Processed_BRFSS_1999_to_2020.csv",
                stringsAsFactors = FALSE)
```

The Data should be individual entries accumulated over time, each entry representing a respondent. It have to be processed before using later on.

Data Range Checker

This function serves an auxilary purpose. By inputting a data frame and the variable name, it tells the year-state range the data would be available. It would be useful for plotting, since averaging on year when data is NOT available would lead to great inaccuracy.

Upon processing BRFSS Data, whole state-year of data would be given -1 for a variable if they are NOT found in raw data. NA from original data would be kept. 0 was NOT seen in raw data. As such, if we drop year-state where sum of one variable is strictly greater than 0, it means that at least One entry were available, and it is not the case that all entry are NA or -1. Since -1 would be given only to whole year of entry when the variable were NOT found in raw data and NOT given if the variable were found in the year's data, this criterion should work well.

```
Data_Range_Checker = function(Input_Data, Var){
  temp = Input_Data %>% group_by(STATE_FIPS_CODE, YEAR_RESPONSE) %>%
    summarize(Temp1 = across({{ Var }}, sum, na.rm = TRUE)) %>%
    ### To sum by state-year for this variable.
    mutate(Temp1 = as.numeric(unlist(Temp1)))

  temp = temp[temp$Temp1 > 0, ]
  ## keep only those where we have over 0 in previous result---
  ## This means that there should be enough data entry.
  return(temp[, 1:2])
}
```

Calculating Sum and Averages

While our original BRFSS Data consisted individual entries, for efficient plotting it should be in state-year format. As such, we shall use sum or averages to condense the data into state-year. The following function does the job. Technical details are included in comments below. Here is a list of input to be used:

> Input_Data: The dataframe to be used. Usually just use Data, which was read from BRFSS_99_20
>
> Var1: String, variable name for the First Variable to be calculated
>
> Var2: String, variable name for Second Variable to be Calculated. Default NA. If NO input, then Only Var1 would be cauculated
>
> Var1 Condition, Var1 Denominator Values: This is used for calculation. Formula:

$$Ratio = \frac{\text{Number of Entries Fitting Var1 Condition}}{\text{Number of Entries Fitting Var1 Denominator Values}}$$

By "Fitting X", it means that the value of the entry is contained in X. Thus, These conditions can be a vector.

Note that If there's any NA in Var1 Denominator Value, includeing NA in a vector or it was not inputted.

Denominator Values are by default c(1:6, 8) since The BRFSS Dataset usually have 7 as unsure and 9 as refused to answer.

Var2 Condition, Var2 Denominator Values: Same. Default NA. NOT used if Var2 is also NA. Warning: Not specifying value of Var2 Condition with a proper Var2 input results in error.

Num or Ratio: if input is "ratio", after calculating enumerator and denominator above, It calculates the ratio (the fraction). It refers to percentage of the entries that fits the condition specified by Var1_Condition and Var2_Condition, amongst the data fitting Var1_Denominator_Value or Var2_Denominator_Value.

If there's ANY input other than "ratio", then the ratio is NOT calculated. Sums would be kept nontheless.

Rename Columns: if TRUE, then Columns will be renamed starting with values you put in Var1 and Var2 Otherwise, it will be left as Var1_Ratio, Var1_Fit_Condition_Count etc.

Years begin, years end: specify the year range. Default is '99 to '20.

Here is the function

```
Data_Condensor = function(Input_Data, Var1, Var2 = NA,
                          Var1_Condition, Var1_Denominator_Values = NA,
                          Var2_Condition = NA, Var2_Denominator_Values = NA,
                          Num_or_Ratio = "Ratio", Rename_Columns = FALSE,
                          year_begin = 1999, year_end = 2020){


  temp = Input_Data %>%
    filter(YEAR_RESPONSE >= year_begin & YEAR_RESPONSE <= year_end)
  ### Filter the data with year range

  year_List_1 = Data_Range_Checker(temp, Var1)
  ### This year_list_1 tells us which state-year these data would be available

  if(!is.na(Var2)){
    ### If var2 is not NA, meaning that there is an entry, also check the range
    ### for var2.
    year_List_2 = Data_Range_Checker(temp, Var2)

    ### Left-Join it with year range for Var1.
    ### Keeps only common elements, so that year-state left here is available
    ### All the year.
    year_state_List = inner_join(year_List_1, year_List_2)

  }else{
    year_state_List = year_List_1
```

```
}

temp = inner_join(temp, year_state_List)
###Keer only state-year that have got available data

if(TRUE %in% is.na(Var1_Denominator_Values)){
  # No entry is for Var1_Denominator_Values. Use Default Setting.
  # Contain everything that ain't not Don't Know, Unsure, or refused.
  # According to codebook, it shall be 1:9 without 7 and 9.
  Var1_Denominator_Values = c(1:6, 8)
}

# Sum up by state-year, drop those with too small a sample size.
First_Var_Result = mutate(temp,
                          Var1_Mask = ifelse(temp[, Var1] %in% Var1_Condition,
                                             1, 0)) %>%
  ### Gives 1 to a temporary variable if the Var1
  ### Fits in condition
  mutate(temp, Var1_Denominator_Count =
           ifelse(temp[, Var1] %in% Var1_Denominator_Values, 1, 0)) %>%
  ### Give 1 to a temporary variable if Var1 fits in
  ### Denominator Value.
  group_by(STATE_FIPS_CODE, YEAR_RESPONSE) %>%
  summarize(Var1_Fit_Condition_Sum = sum(Var1_Mask, na.rm = TRUE),
            Var1_Total = sum(Var1_Denominator_Count, na.rm = TRUE)) %>%
  ### Sum the previous temporary variables by state-year
  ### And write them in two variables for use later on.
  filter(Var1_Fit_Condition_Sum >= 100)

if(!is.na(Var2)){
  # Var2 is default NA.
  # If anything was put onto Var2, then this chunk shall run,
  # doing the same thing as above.

  if(TRUE %in% is.na(Var2_Denominator_Values)){
    Var2_Denominator_Values = c(1:6, 8)
  }
  Second_Var_Result = temp %>%
    mutate(Var2_Mask = ifelse(temp[, Var2] %in% Var2_Condition, 1, 0)) %>%
    mutate(temp, Var2_Denominator_Count =
             ifelse(temp[, Var2] %in% Var2_Denominator_Values, 1, 0)) %>%
    group_by(STATE_FIPS_CODE, YEAR_RESPONSE) %>%
    summarize(Var2_Fit_Condition_Sum = sum(Var2_Mask, na.rm = TRUE),
              Var2_Total = sum(Var2_Denominator_Count, na.rm = TRUE)) %>%
    filter(Var2_Fit_Condition_Sum >= 100)

  ### Re-write temp dataframe as inner-join of the
  ### result from calculating two variables
  temp = inner_join(First_Var_Result, Second_Var_Result)
}else{
  ### No variable 2. Use only variable 1 result.
  temp = First_Var_Result
}
```

```
  if(Num_or_Ratio == "Ratio"){
    # Find ratios for var1.
    temp = temp %>% mutate(Var1_Ratio = Var1_Fit_Condition_Sum / Var1_Total)
    # rename, if Rename_Columns was TRUE.
    if(Rename_Columns){
      temp = temp %>%
        rename_at(vars(c("Var1_Fit_Condition_Sum", "Var1_Total", "Var1_Ratio")),
                  ~ c(paste(Var1, "_Fit_Condition_Count", sep = ""),
                      paste(Var1, "_Total", sep = ""),
                      paste(Var1, "_Ratio", sep = "")))
    }
    if(!is.na(Var2)){
      # Do the same thing for Var2, if it was calculated.
      temp = temp %>%
        mutate(Var2_Ratio = Var2_Fit_Condition_Sum / Var2_Total)
      if(Rename_Columns){
        temp = temp %>%
          rename_at(vars(c("Var2_Fit_Condition_Sum",
                           "Var2_Total", "Var2_Ratio")),
                    ~ c(paste(Var2, "_Fit_Condition_Count", sep = ""),
                        paste(Var2, "_Total", sep = ""),
                        paste(Var2, "_Ratio", sep = "")))
      }
    }
  }
  else{
    ### No Ratio Calculated. Do Rename here.
    if(Rename_Columns){
      temp = temp %>%
        rename_at(vars(c("Var1_Fit_Condition_Sum", "Var1_Total")),
                  ~ c(paste(Var1, "_Fit_Condition_Count", sep = ""),
                      paste(Var1, "_Total", sep = "")))
    }
    if(!is.na(Var2)){
      temp = temp %>% mutate(Var2_Ratio = Var2_Fit_Condition_Sum / Var2_Total)
      if(Rename_Columns){
        temp = temp %>%
          rename_at(vars(c("Var2_Fit_Condition_Sum", "Var2_Total")),
                    ~ c(paste(Var2, "_Fit_Condition_Count", sep = ""),
                        paste(Var2, "_Total", sep = "")))
      }
    }
  }
  return(temp)
}
```

Here, we shall have a dataframe of these ratios and sums, calculated for year-states of good availability. See Codebook.xlsx for availability of entries under BRSFF_Data for specific availability of data.

Scatterplot, ratio-to-ratio of two variables

The following function plots two ratios, as calculated above, as a scatterplot, with each state-year entry of two ratios shown as a data-point. The color of the point depends on the year of response. I tried to create a version of using state names of color, but found it too confusing as colors near to each other usually represent drastically different state.

The function only need two input:

> Input_Data: Use a processed (Using Data_Condensor function) version of Data (the data frame we read from CSV file near the top). Note that for proper function—since this is a plotting of ratios—Num_or_Ratio settings of Data_Condensor function must be set at "Ratio", the default setting.
>
> reverse_xy_axis: reverse variable of X-Y axis if needed. Default FALSE

It needs not any variable names, for it automatically retrieves the variables whose name ends with "_Ratio" and use them for plotting.

It is advised to add explanations below the graph for clearer understanding.

```
Ratio_Scatter_Plotter_1 = function(Input_Data, reverse_xy_axis = FALSE){
  ### Retrieve the list of variables needed for plotting.
  List_of_Plotting_Variables =
    colnames(Input_Data)[which(grepl("_Ratio", colnames(Input_Data)))]

  ### Record the variable names
  Var1 = List_of_Plotting_Variables[1]
  Var2 = List_of_Plotting_Variables[2]

  if(reverse_xy_axis){
    ### Reverse X-Y axis if needed
    Var1 = List_of_Plotting_Variables[2]
    Var2 = List_of_Plotting_Variables[1]
  }

  ### Plotting.
  ggplot(data = Input_Data,
                aes_string(x = Var1,
                           y = Var2,
                           color = "YEAR_RESPONSE")) +
    geom_point() +
    ggtitle("Comparing two ratios within BRFSS Dataset
            \n Averaged Over Years for All States") +
    geom_smooth(method='lm')
  ### Make a regression at the end.

}
```

The Following is an example. The Data_Condensor function were used within the Ratio_Scatter_Plotter_1 function. Here is a list of explanations to some of the variables used:

> Var1: HEALTH_PLAN_STATUS. A variable asking if the respondent have any health insurance plan.

Var1_Condition: Set to 1, meaning that the respondent have some kind of coverage.

Var1_Denominator_Value: Set to default, which was c(1:6, 8). Included all those who indeed gave an response.

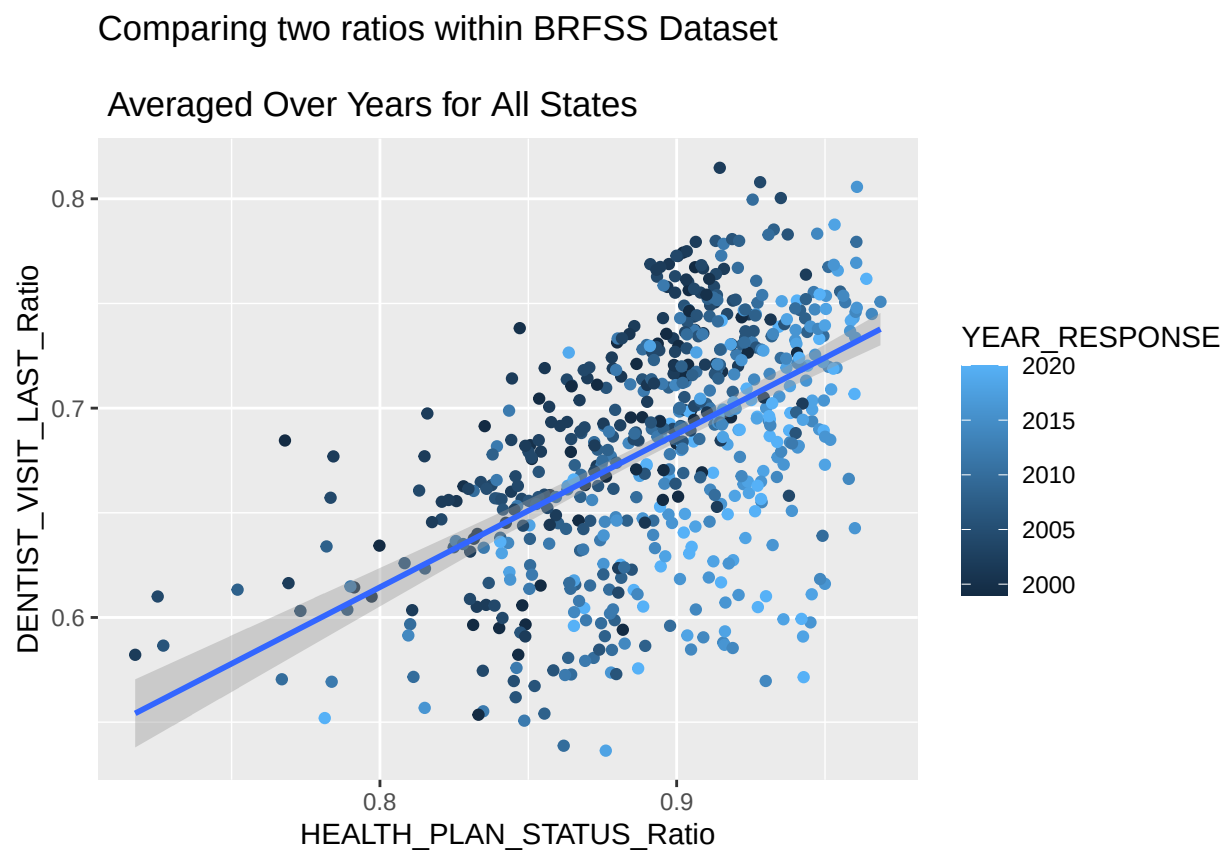
Var2: DENTIST_VISIT_LAST, A variable asking if the respondent has visited a dentist in the past year.

Var2_Condition: Set to 1, meaning that the respondent have visited dentist in the past year.

Of course, the data here would be calculated into ratios. Thus, for each data point in the plot, the Y-axis shows the percentage of people who have visited dentist last year, while the X-axis shows the percentage of people who have insurance plans. Each data point represents a state-year, with the year shown by different intensity of blue color.

For more specific meaning of the data entries, refer to Codebook, sheet 1 in BRFSS_Data directory.

```
Ratio_Scatter_Plotter_1(Input_Data = Data_Condensor(
  Input_Data = Data, Var1 = "HEALTH_PLAN_STATUS",
  Var1_Condition = 1,
  Var2 = "DENTIST_VISIT_LAST",
  Var2_Condition = 1, Rename_Columns = TRUE),
  reverse_xy_axis = FALSE)
```

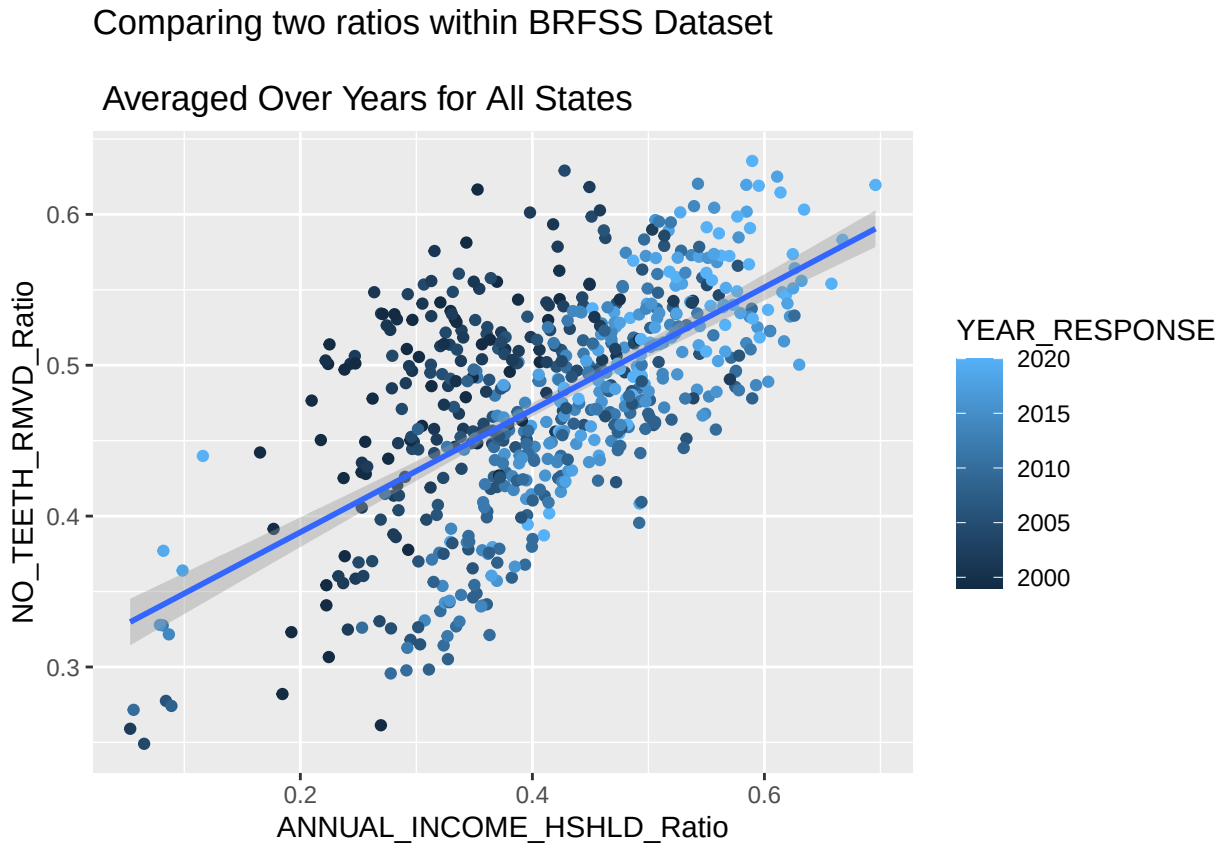


There is a list of X and Y variable that possibly be of interest. Note that you will need to check codebook for their meanings before producing plots.

```
XVar_List = c("ANNUAL_INCOME_HSHLD", "HEALTH_PLAN_STATUS",
              "MED_COST", "CHECKUP_MOST_RECENT")
YVar_List = c("DENTIST_VISIT_LAST", "NO_TEETH_RMVD", "DENTIS_CLEAN_LAST")
```

The following also gives another example plot. The First variable, one on X-axis, refers to annual household income, specifically the portion of respondent who have household income of $\$50,000$ or more. The Y-axis is Number of Teeth Removed. Var2_Condition $= 8$ means that not a single teeth of the respondent was removed.

```
Ratio_Scatter_Plotter_1(Data_Condensor(Input_Data = Data,
                                       Var1 = XVar_List[1],
                                       Var2 = YVar_List[2],
                                       Var1_Condition = c(7:8),
                                       Var1_Denominator_Values = c(1:8),
                                       Var2_Condition = 8,
                                       Var2_Denominator_Values = c(1:6, 8),
                                       Rename_Columns = TRUE),
                        reverse_xy_axis = FALSE)
```



It is advised to take notes on the meaning of the ratios upon making plots. The limited space in X-Y axis means that we cannot put the whole formula there, and the difference between variables used means difficulty in providing a coherent formula that works for each variables.

If you want to make customized entries, As long as you don't have reverse_xy_axis = TRUE the Var1 is always on X-axis and Var2 on Y-axis. There is a function in ggplot, labs(), that allows customized label for x and y axis. Add them to the GGPLot section in the function and it shall show customized labels.

Plotting Averages

The following function shall plot averages of ratios over state and/or year, using also processed data. It shall plot a scatterplot but with averaged ratios on X and Y axis respectively, with a regression line.

Here is the list of variables used:

> Input_Data: Use a processed dataframe just like the ones above.
>
> return_average_of: use average of year, state, or both. Options: "year", "state", "both". Default "year"
>
> return_average_of = "year" means that for each year, you get the averages (of states). = "state" means that for each state, return averages over year. if "both" were used, then there would be two plots.
>
> same Scale: if you used "both" in return_average_of, then same_scale = TRUE means that two resulted plots would be using same x and y limits, which would be designated immediately following this variable. If anything else, the scales would not be the same but automatically decided from each graph. This was used so that it will be an easier time comparing two plots. Default TRUE
>
> xlim, ylim: both pairs consists a vector of two numbers that limits X and Y axis. Default is .2 to 1 for both.
>
> Again, the X and Y variables were automatically designated with the variables ended with "Ratio". It can be changed within the function if you need otherwise.
>
> vertical_or_horizontal_arrange: default "horizontal. Choose from"horizontal" or "vertical". tells how the two graphics plotted would be arranged. Useful only when you put "both" in return_average_of.

Technical details are included in the comments in the function below.

```
Ratio_Scatter_Plotter_2 = function(Input_Data,
                                   return_average_of = "year",
                                   reverse_xy_axis = FALSE,
                                   same_scale = TRUE,
                                   xlim = c(.2, 1), ylim = c(.2, 1),
                                   vertical_or_horizontal_arrange = "horizontal"){

  List_of_Plotting_Variables =
    colnames(Input_Data)[which(grepl("_Ratio", colnames(Input_Data)))]
  ## Find the list of variables to be plotted

  Var1 = List_of_Plotting_Variables[1]
  Var2 = List_of_Plotting_Variables[2]

  if(reverse_xy_axis){
    Var1 = List_of_Plotting_Variables[2]
    Var2 = List_of_Plotting_Variables[1]
  }
```

```
### Designate the variables for X and Y axis respectively,
### Which are first and second variables ended in "_Ratio".
### Reversed if reverse_xy_axis == TRUE.

### Get the year range of data.
### Since input should be data cleaned by Data_Condensor,
### Every year should see data fully available.
Year_Range = unique(Input_Data$YEAR_RESPONSE)

# Do averages
if(return_average_of == "state"){
  # For state, find average over years.

  Input_Data =  Input_Data %>% group_by(STATE_FIPS_CODE) %>%
    summarize("Var1_Ave_Over_Years" = across({{ Var1 }}, mean),
              "Var2_Ave_Over_Years" = across({{ Var2 }}, mean)) %>%
    ### Find mean over years, for each state.

    mutate(Var1_Ave_Over_Years = as.numeric(unlist(Var1_Ave_Over_Years)),
           Var2_Ave_Over_Years = as.numeric(unlist(Var2_Ave_Over_Years))) %>%
    ### Use numeric to prevent formatting problems

    rename_at(vars(c("Var1_Ave_Over_Years", "Var2_Ave_Over_Years")),
              ~ c(paste(Var1, "_Ave_Over_Years", sep = ""),
                  paste(Var2, "_Ave_Over_Years", sep = ""))) %>%
    ### Rename the variables with character Var1 and Var2 carries
    ### Prevent confusion later on

    mutate(State_Abb = fips(STATE_FIPS_CODE, to =  "Abbreviation")) %>%
    ### Change State FIPS code to abbreviations.

    filter(STATE_FIPS_CODE <= 56)
  # No more Puerto Rico, Virgin Islands, and Guam

  ### Conduct the plotting.
  ggplot(data = Input_Data,
         aes_string(x = paste(Var1, "_Ave_Over_Years", sep = ""),
             y = paste(Var2, "_Ave_Over_Years", sep = "") )) +
    geom_point() +
    geom_text(label = Input_Data$State_Abb,
              nudge_x = -.002, nudge_y = .001) +
    ggtitle("Comparing two ratios within BRFSS Dataset
            \n Averaged Over Years for All States") +
    geom_smooth(method='lm')

}else if(return_average_of == "year"){
  # Average by year, over states.
  Input_Data =  Input_Data %>%
    filter(STATE_FIPS_CODE <= 56) %>%
    # No more Puerto Rico, Virgin Islands, and Guam

    group_by(YEAR_RESPONSE) %>%
    summarize("Var1_Ave_Over_States" = across({{ Var1 }}, mean),
```

```
            "Var2_Ave_Over_States" = across({{ Var2 }}, mean)) %>%
    ## Make Average over states for each year

    mutate(Var1_Ave_Over_States = as.numeric(unlist(Var1_Ave_Over_States)),
           Var2_Ave_Over_States = as.numeric(unlist(Var2_Ave_Over_States))) %>%

    rename_at(vars(c("Var1_Ave_Over_States", "Var2_Ave_Over_States")),
              ~ c(paste(Var1, "_Ave_Over_States", sep = ""),
                  paste(Var2, "_Ave_Over_States", sep = "")))

  ## Plot with a regression line
  ggplot(data = Input_Data,
         aes_string(x = paste(Var1, "_Ave_Over_States", sep = ""),
                    y = paste(Var2, "_Ave_Over_States", sep = "") )) +
    geom_point() +
    geom_text(label = Input_Data$YEAR_RESPONSE,
              nudge_x = -.002, nudge_y = .001) +
    ggtitle("Comparing two ratios within BRFSS Dataset
            \n Averaged Over State for All Years") +
    geom_smooth(method='lm')
}else if(return_average_of == "both"){
  # Plot both stuff above side-by-side.

  Part1 = Input_Data
  Part2 = Input_Data

  # Average over States. Generally the same as above.
  Part1 = Part1 %>% group_by(STATE_FIPS_CODE) %>%
    summarize("Var1_Ave_Over_Years" = across({{ Var1 }}, mean),
              "Var2_Ave_Over_Years" = across({{ Var2 }}, mean)) %>%
    mutate(Var1_Ave_Over_Years = as.numeric(unlist(Var1_Ave_Over_Years)),
           Var2_Ave_Over_Years = as.numeric(unlist(Var2_Ave_Over_Years))) %>%
    rename_at(vars(c("Var1_Ave_Over_Years", "Var2_Ave_Over_Years")),
              ~ c(paste(Var1, "_Ave_Over_Years", sep = ""),
                  paste(Var2, "_Ave_Over_Years", sep = ""))) %>%
    mutate(State_Abb = fips(STATE_FIPS_CODE, to =  "Abbreviation")) %>%
    filter(STATE_FIPS_CODE <= 56) # No more Puerto Rico, Virgin Islands, and Guam

  # Average over years. Also generally the same as above
  Part2 = Part2 %>%
    filter(STATE_FIPS_CODE <= 56) %>%
    # No more Puerto Rico, Virgin Islands, and Guam

    group_by(YEAR_RESPONSE) %>%
    summarize("Var1_Ave_Over_States" = across({{ Var1 }}, mean),
              "Var2_Ave_Over_States" = across({{ Var2 }}, mean)) %>%
    mutate(Var1_Ave_Over_States = as.numeric(unlist(Var1_Ave_Over_States)),
           Var2_Ave_Over_States = as.numeric(unlist(Var2_Ave_Over_States))) %>%
    rename_at(vars(c("Var1_Ave_Over_States", "Var2_Ave_Over_States")),
              ~ c(paste(Var1, "_Ave_Over_States", sep = ""),
                  paste(Var2, "_Ave_Over_States", sep = "")))

  Part1_Plot = ggplot(data = Part1,
```

```
                        aes_string(x = paste(Var1, "_Ave_Over_Years", sep = ""),
                            y = paste(Var2, "_Ave_Over_Years", sep = "") )) +
      geom_point() +
      geom_text(label = Part1$State_Abb,
                nudge_x = -.002, nudge_y = .001) +
      ggtitle("Comparing two ratios within BRFSS Dataset
              \n Averaged Over Years for All States") +
      geom_smooth(method='lm')


    Part2_Plot = ggplot(data = Part2,
           aes_string(x = paste(Var1, "_Ave_Over_States", sep = ""),
                      y = paste(Var2, "_Ave_Over_States", sep = "") )) +
      geom_point() +
      geom_text(label = Part2$YEAR_RESPONSE,
                nudge_x = -.002, nudge_y = .001) +
      ggtitle("Comparing two ratios within BRFSS Dataset
              \n Averaged Over State for All Years") +
      geom_smooth(method='lm')

    if(same_scale){
      ### If using same scales, rearrange the limits to same scales
      Part1_Plot = Part1_Plot + xlim(xlim[1], xlim[2]) + ylim(ylim[1], ylim[2])
      Part2_Plot = Part2_Plot + xlim(xlim[1], xlim[2]) + ylim(ylim[1], ylim[2])
    }
    if(vertical_or_horizontal_arrange == "horizontal"){
      grid.arrange(Part1_Plot, Part2_Plot, ncol = 2)

    }else if(vertical_or_horizontal_arrange == "vertical"){
      grid.arrange(Part1_Plot, Part2_Plot, nrow = 2)
    }

  }
}
```

Here is two examples of the two plots above recreated, but averaged for both state and year.

For this plot, again X-axis are percentage of people with health insurance, this time averaged, and Y-axis percentage of people who visited dentist last year, also averaged. While usually I use horizontal arrangement, since this is to be used in a PDF document vertical arrangement should be clearer.

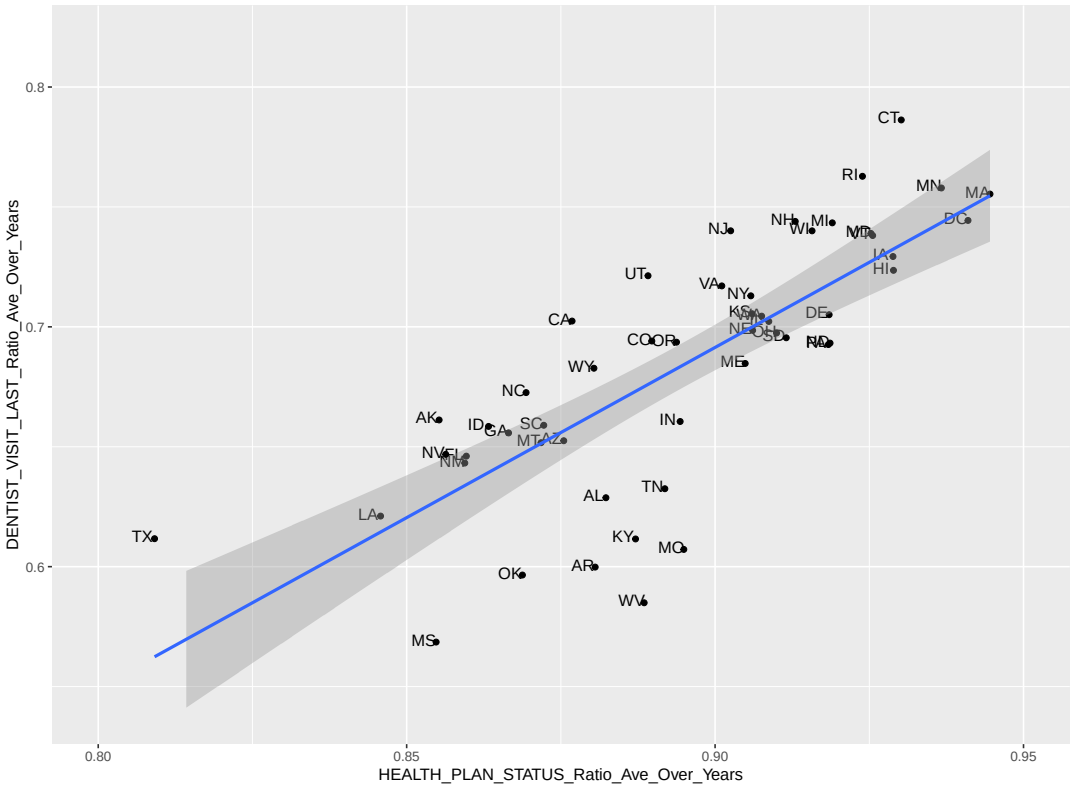
Note that for R Markdown, add fig.height = 16, fig.width=10 in the top of coding chunk (Like {r fig.height = 16, fig.width=10}) when using vertical plotting of the Ratio_Scatter_Plotter_2 function. Otherwise the graphic would be compressed too short.

```
Ratio_Scatter_Plotter_2(Input_Data = Data_Condensor(
  Input_Data = Data, Var1 = "HEALTH_PLAN_STATUS",
  Var1_Condition = 1,
  Var2 = "DENTIST_VISIT_LAST",
  Var2_Condition = 1, Rename_Columns = TRUE),
  return_average_of = "both",
  reverse_xy_axis = FALSE,
  same_scale = TRUE,
  xlim = c(.8, .95),
```

```
  ylim = c(.54, .82),
  vertical_or_horizontal_arrange = "vertical")
```

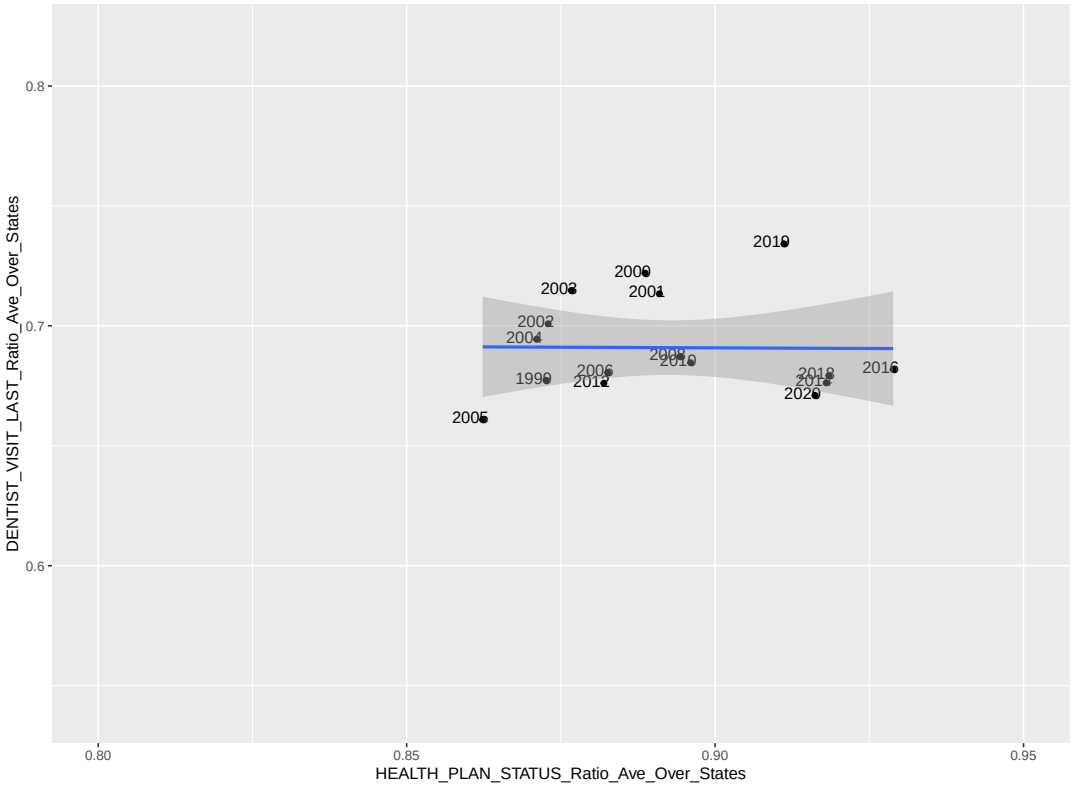
Comparing two ratios within BRFSS Dataset

Averaged Over Years for All States



Comparing two ratios within BRFSS Dataset

Averaged Over State for All Years

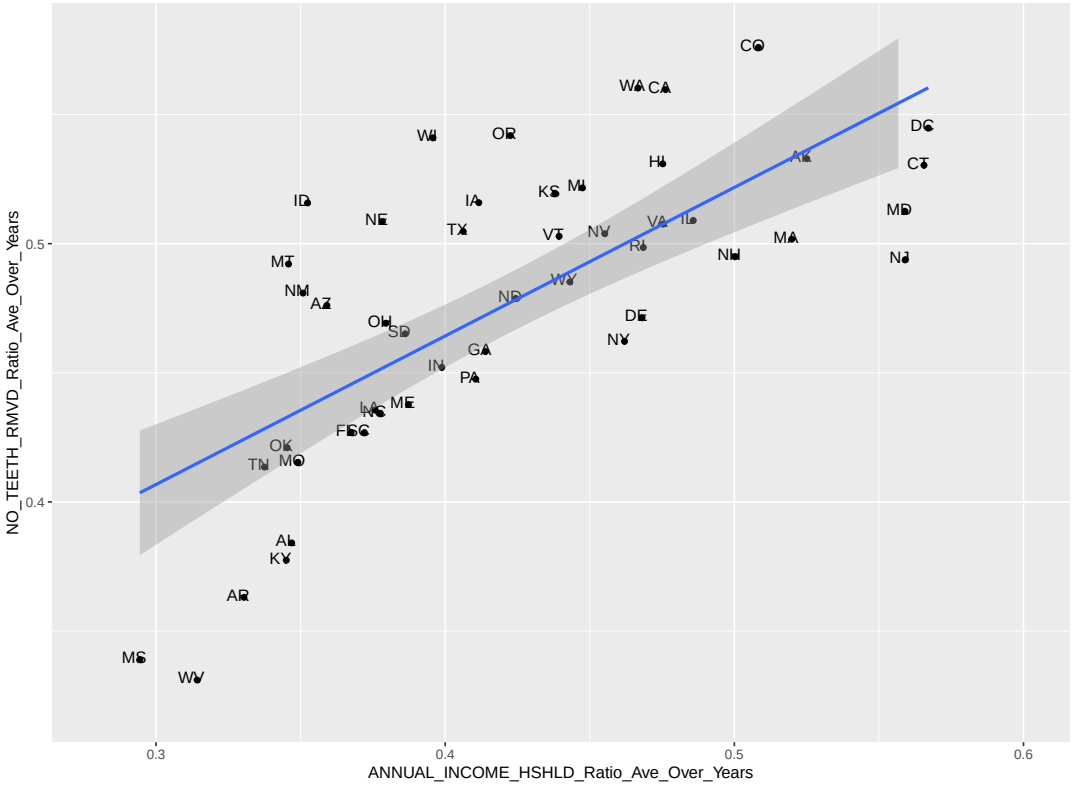


For this plot, we have X-axis for people with over $\$50,000$ annual income, and Y-axis people who have not a single teeth removed.

```
Ratio_Scatter_Plotter_2( Data_Condensor(Input_Data = Data,
                                        Var1 = XVar_List[1],
                                        Var2 = YVar_List[2],
                                        Var1_Condition = c(7:8),
                                        Var1_Denominator_Values = c(1:8),
                                        Var2_Condition = 8,
                                        Var2_Denominator_Values = c(1:6, 8),
                                        Rename_Columns = TRUE),
                         return_average_of = "both",
                         xlim = c(.28, .6),
                         ylim = c(.32, .58),
                         vertical_or_horizontal_arrange = "vertical")
```

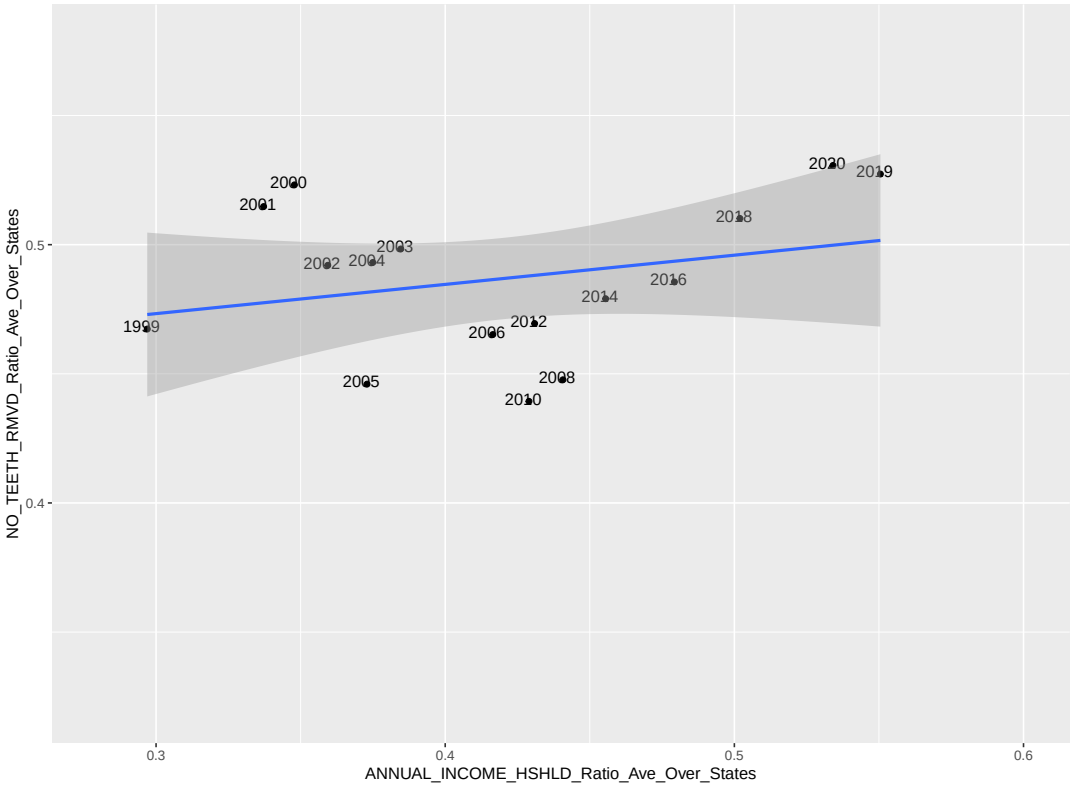
Comparing two ratios within BRFSS Dataset

Averaged Over Years for All States



Comparing two ratios within BRFSS Dataset

Averaged Over State for All Years



Feel free to use Codebook and the function to create even more plots to be used!