Introduction

Link to app: https://emiliaruzicka.shinyapps.io/USPS_Operations_Data/

For my final project, I made a Shiny App using data from the US Postal Service (USPS), which I also used for my midterm project. The data was originally downloaded in multiple datasets and contains annual:

- Domestic letter rates, 1792—Present
- Pieces of mail handled, 1789—Present
- Number of post offices, 1789—Present
- Income, 1789—Present
- Expenses, 1789—Present
- Stamped card/postcard rates, 1873—Present
- First class mail volume, 1926—Present
- Number of stamped cards/postcards sent via first class mail, 1926—Present
- Number of postal employees, 1926—Present

All of the data can be found through the USPS directly at: https://about.usps.com/who-we-are/postal-history/rates-historical-statistics.htm

The intent of this app is to allow users to explore the operations data of the USPS for themselves. Since the USPS was reorganized in 1970 and the data prior to that has many missing values and anomalies, only data from 1970-2019 is available to view in the app. This app has four useful outputs: a scatterplot with a trendline, a boxplot, a line graph, and a table. Each of these elements is responsive to the user input in the left sidebar.

Getting started

To begin, I clearly documented my app file and imported the necessary libraries. I also imported by data and made a vector of the names of my data’s columns so that I could easily use them later.

```r
# Emilia Ruzicka
# PHP 2560 Final Project
# 12/07/2020

# Import packages
library(shiny)
```
library(ggplot2)
library(tidyverse)
library(RColorBrewer)

# Import data
USPSData <- read.csv("USPSData.csv", stringsAsFactors=FALSE)

# Vector of column names for ease of use

UI setup

My UI setup starts with using fluidPage() so that the positioning of the elements of my app is responsive to the size of the user’s screen. I then added a title using the titlePanel() function.

# Define UI for application with side panel, scatterplot, boxplot, line graph, and table
ui <- fluidPage(
  # Application title

  Next, I set up the sidebar where users can choose which variables they would like to see in the visualizations. The “Year” variable is changed using sliderInput() because years have a predetermined order.

  # Sidebar
  sidebarLayout(sidebarPanel(
    # Slider input for years
    sliderInput("Year", "Years:", min = 1970, max = 2019, value = c(1970,2019), sep = ""),

    All of the other variables are changed using selectInput() because they are chosen from a list of available data columns. Users can change the x and y values of the scatterplot, the color of the scatterplot, the type of trendline on the scatterplot, the group of data showing in the boxplot, and the y values of the line graph.

    # Drop-down menu for scatterplot x axis
    selectInput("X", "Scatterplot X Value:", choices = data_names, selected = "Income"),

    # Drop-down menu for scatterplot y axis
    selectInput("Y", "Scatterplot Y Value:", choices = data_names, selected = "First.Class.Mil.Pieces"),

    # Drop-down menu for scatterplot color
    selectInput("color", "Color Scatterplot by:", choices = data_names, selected = "Year"),

    # Drop-down menu for scatterplot trendline
    selectInput("trend", "Scatterplot Trendline:", choices = c("Linear", "Quadratic", "Smooth"), selected = "Linear"),

    # Drop-down menu for boxplot data
    selectInput("group", "Boxplot Data:", choices = c("Financial", "Card Mail Volume", "Overall Mail Volume", "Mail Rates"), selected = "Financial"),

    # Drop-down menu for first line graph y value
    selectInput("Y2", "Line Graph Y Value (purple):", choices = data_names, selected = "Income"),

  2
The final step for setting up the UI of this app is to build the main panel with all of the visualizations in it. This is implemented by calling plotOutput() and tableOutput() within the mainPanel() function. The order that I call the *Output() functions in determines the order that the visualizations will show up in my app.

Server setup

The server for this app takes in the input from the user collected using the sidebar and outputs the proper visualizations, which are defined below. First is the scatterplot, which is shown using renderPlot(). Initially, the data is narrowed so that it only encompasses the years determined by the user. Then the formula and model for the trendline are determined using an if statement and the user’s choice between linear, quadratic, and smooth. Finally, a ggplot() call draws the scatterplot itself, taking in the x, y, and color values from the input data that the user chose. The function stat_smooth() draws the trendline and the calls to labs(), scale_color_continuous(), and theme() format the visualization properly.

```r
# Define server logic required to draw a scatterplot, boxplot, line graph, and table
server <- function(input, output) {
  output$scatter <- renderPlot({
    # Use only specified years based on user input
    USPSData_small <- USPSData %>% filter(Year %in% seq(from = input$Year[1], to = input$Year[2]))

    # Choose correct formula based on user input
    f <- "y ~ x"
    m <- "lm"
    if(input$trend == "Quadratic") {
      f <- "y ~ x + I(x^2)"
      m <- "lm"
    } else if (input$trend == "Linear") {
      f <- "y ~ x"
      m <- "lm"
    } else {
      f <- NULL
      m <- NULL
    }

    # Draw scatterplot
    ggplot(USPSData_small, aes(x = USPSData_small[, input$X], y = USPSData_small[, input$Y],
                                color = USPSData_small[, input$color])) + geom_point() +
    stat_smooth(method = m, formula = f, size = 1) +
    labs(title = paste(input$X, " vs. ", input$Y), x = paste(input$X), y = paste(input$Y)) +
    scale_color_continuous(type = "viridis", name = paste(input$color)) +
    theme(text = element_text(size = 14))
  })
}
```
The boxplot is the next visualization in the app. Again, renderPlot() is called and the years of data are narrowed by the slider input. Next, I wrote a function to reconfigure the data into a long format, which is more suited to boxplots in ggplot2. The function get_long() takes in a group name given by the user and returns a long dataframe that includes the year, type of data/variable, and the value for that variable in that year. For instance, get_long() changes a dataframe with columns for year, income, expenses, and profit, into a dataframe with columns for year (numerical years), type (a factor that has the value “income”, “expenses”, or “profit”), and value (the value for the type of data in the given year). Using this function, I can easily get the longform data and use it in my ggplot() call, which draws the boxplot. I also added specific datapoints using geom_scatter() so that users could see what years had high and low vaues for the variables. Finally, scale_color_continuous(), lab(), and theme() help with the formatting and design again.

```r
output$box <- renderPlot({
    # Use only specified years based on user input
    USPSData_small <- USPSData %>% filter(Year %in% seq(from = input$Year[1], to = input$Year[2]))
    # Write function to reconfigure data to longform
    get_long <- function(group) {
        #' Function to make USPS data longform
        #' @param group string, type of data displayed according to user input
        #' @return long dataframe, reconfigured data according to user-chosen group
        # Define empty vectors to fill according to group
        yr <- c()
        typ <- c()
        val <- c()

        # Define variable n to determine number of repeats for each type of variable
        if(input$Year[1] == 1970) {
            n <- input$Year[2] - input$Year[1]
        } else {
            n <- input$Year[2] - input$Year[1] + 1
        }

        # Reconfigure data according to group
        # yr is the "Year" vector repeated the same number of times as
        # number of variables being reconfigured.
        # typ determines which variable the value came from. Each typ name is repeated n times.
        # val is the value of the typ for the designated yr.
        if(group == "Financial") {
            yr <- rep(USPSData_small$Year, 3)
            typ <- rep(c("Income", "Expenses", "Profit"), each = n)
            val <- c(USPSData_small$Income, USPSData_small$Expenses, USPSData_small$Profit)
        } else if (group == "Card Mail Volume") {
            yr <- rep(USPSData_small$Year, 2)
            typ <- rep(c("First.Class.Cards", "First.Class.Presort"), each = n)
            val <- c(USPSData_small$First.Class.Cards, USPSData_small$First.Class.Presort)
        } else if (group == "Overall Mail Volume") {
            yr <- rep(USPSData_small$Year, 2)
            typ <- rep(c("First.Class.Cards", "First.Class.Presort"), each = n)
            val <- c(USPSData_small$First.Class.Cards, USPSData_small$First.Class.Presort)
        }
    }
})
```
yr <- rep(USPSData_small$Year, 2)
typ <- rep(c("Mail.Handled", "First.Class.Total"), each = n)
val <- c(USPSData_small$Mail.Handled, USPSData_small$First.Class.Mil.Pieces)
}
else
{
yr <- rep(USPSData_small$Year, 2)
}
# Make dataframe of yr, typ, and val
long <- data.frame(yr, typ, val)
# Return long dataframe
return(long)

# Get longform data according to user-chosen group
USPSData_long <- get_long(input$group)

# Draw scatterplot
ggplot(USPSData_long, aes(typ, val)) + geom_boxplot() + geom_point(aes(color = yr, alpha = .4))
    theme_text_color_continuous(type = "viridis", name = "Year") +
    labs(title = paste("USPS", input$group, "Data"), x = "", y = "")
)

The third visualization is the line graph. This graph lets users compare the trends of two variables over time. As with the other visualizations, we start by calling renderPlot() and only using the subset of the data for the years that the user specifies. Next, calls to ggplot() and geom_line() draw the line graph, using the two y variables chosen by the user in the sidebar. These lines are colored differently and labeled in the title of the plot so that it is easy for users to tell which line corresponds with which variable.

```r
output$line <- renderPlot(

    # Use only specified years based on user input
    USPSData_small <- USPSData %>% filter(Year %in% seq(from = input$Year[1], to = input$Year[2]))

    # Draw line graph according to user-chosen variables
    ggpolygon(USPSData_small, aes(x = Year)) +
        geom_line(aes(y = USPSData_small[, input$Y2]), color = "purple") +
        geom_line(aes(y = USPSData_small[, input$Y3]), color = "green") +
        labs(title = paste("Year vs.", input$Y2, " (purple) and", input$Y3, " (green)") , x = "Year", y = "")
)
```

Lastly, a table of all of the variables for the specified years is used. Instead of renderPlot(), renderTable() is used so that the output is a table. We subset the USPS dataset using the which() function so that only the specified years of data are shown, corresponding to all of the other visualizations in the app. Setting “striped” equal to “TRUE” makes the table easier to read.

```r
output$table <- renderTable(USPSData[which(USPSData$Year %in% seq(from = input$Year[1], to = input$Year[2])),]
    striped = TRUE, hover = FALSE, bordered = FALSE)
```
Run the app

After defining global variables, the UI, and the server, we can finally run the app. This is done using a simple call to shinyApp(), which takes in the ui and server definitions that we made above.

```r
# Run the application
shinyApp(ui = ui, server = server)
```