Programming Assignment 3

In this programming assignment, you are asked to build a web app that visualizes the data.csv dataset. BPD Arrests data is a list of arrests since 2014 provided by the Baltimore Police Department. You can find the data here.

It is recommended that you consult with the materials provided in the "Week-5" folder to familiarize yourself with some examples of the functions mentioned in this assignment. Each line of code in your app.R file should be sufficiently commented such that the intent/functionality of each line of your code is clearly specified.

The goal of this assignment is to practice working with time-series data, plotly and leaflet packages, and more advanced concepts in the Shiny app. The expected result is published in the following website: https://yazdi.shinyapps.io/PAssignment3/

For parts a-e, you should create a new R script and run your code there. Once you can create the desired graphs, then you will learn how to adjust your existing code for a shiny web app and insert them into the app.R in parts f and g. app.R is an incompleted shiny app provided to you. You should only submit the completed app.R file through the Blackboard. You do not need to publish your work for this assignment.

a) Required libraries:

You need the following packages for this assignment:

- tidyverse: for data wrangling and ggplot visualization
- shiny: for building the web application visualizations
- shinydashboard and shinydashboardPlus: for a dashboard web-interface
- leaflet: for the interactive map visualization in the "Map" tab
- DT: for the interactive data table in the "Data" tab
- plotly: for making the ggplot visualization in the "Plotly" tab interactive

b) Data Wrangling:

The goal of this part is to read and clean the data.csv dataset. Follow the instructions below:

- Store the dataset on a variable named data.

- The dates of arrests are stored as characters in R. Use the following command to convert ArrestDate column from characters to date:

  ```
  data$ArrestDate <- as.Date(data$ArrestDate, format="%m/%d/%Y")
  ```

- You will use the DT package later to show the data dataframe on the Data tab. The two columns Longitude and Latitude include the coordinates of arrests up to 10^{-12} of a degree which makes it hard for users to read the table. Round both Longitude and Latitude columns to 5 digits.

c) Density graph

In this part, you will create a density graph illustrating age distribution for each of Male and Female gender (See the tab Density on the published web app). The ultimate goal is to connect this graph with the slider input widget named year so users can see the trend of density over different times. But, for now, use the following instructions to write a code which creates the visualization for only the year 2014.

- Use the following command inside the filter function to filter the data for the year 2014.

  ```
  as.numeric(format(ArrestDate,'%Y')) == 2014
  ```

- Pass the filtered data to the ggplot function.
- As you may notice, the goal of this graph is not to compare the number of crimes over different genders. The goal is to compare the density distribution within each gender and that is why geom_density will be used and not geom_histgram. Use the geom_density to create density functions. You can adjust the thickness of the lines as you want by the parameter size.
- Add the year 2014 to the background as you did in PA1.
- Set the labs, xlim, and ylim of the graph properly.
- The color legend shows the first letter of each gender (like “M”). Use scale_color_discrete function to change them to the complete format (like “Male”).
- You can decide about the theme of the graph. I found the density unit confusing for users. That is why I used the following command to remove some elements of the y-axis.

  ```
  theme(axis.text.y=element_blank(), axis.ticks.y = element_blank(),
            panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
            panel.background = element_blank())
  ```

d) Plotly graph

In this part, you practice how to convert a ggplot figure to an interactive ggplotly figure. As you can see in the Plotly tab in the published web app, the ultimate goal of this part is to create an interactive line chart showing the number of arrest records in the dataset for each day over the past 6 years so users can explore the trend and outliers in the graph. Users also should be able to show the holidays on the figure because being a holiday can affect the number of arrests.

- Read the usholidays.csv dataset, remove the first column(row index), and store it on holidays variable.
- Convert the Date column from characters to Date.
- I found the name of each holiday unnecessarily long to be shown on the figure. That is why I decided to use its abbreviations instead. Use the following code to create a new column named Abb which holds the abbreviation of Holiday column. For example, “New Year's Day” will be changed to “NYD”.

```
words=unique(holidays$Holiday)
Abb=c("NYD","MLKB","WaB", "MeD", "InD", "LaD", "CoD", "VeD", "ThD", "ChD",
"NYD","MLKB","WaB")
      holidays$Abb=holidays$Holiday
      for (i in 1:length(words)) {
         holidays$Abb=str_replace(holidays$Abb,words[i],Abb[i])
      }
```

- Find the number of crimes in each day using the following command:

```
data %>% group_by(Date=ArrestDate) %>%  summarise(N=n())
```

- Use the merge function to merge the dataframe created above with the holidays dataframe and store it on a variable named data_hol. Make sure to use parameter all.x=TRUE inside the merge function to include all days in the data_hol even if they are not holidays.
- Use the geom_line and geom_smooth to create a plot like the one in the Plotly tab and store the result on a variable named f.
- You do not need to change the theme of the figure. Use the following command to add the holiday points and texts on the figure.

```
f=f+geom_point(data= subset(data_hol, !is.na(Holiday)), color="purple")+
    geom_text(data=subset(data_hol,!is.na(Holiday)), aes(x=Date, y=N, label=Abb))
```

- Finally, you can convert your ggplot figure to ggplotly one using the following command:

```
ggplotly(f)
```

e) Map

In this part, you will create a heatmap using the leaflet package. As you can see in the Map tab in the published web app, the color of each rectangle shows the number of arrests in that area. If it is solid red, it means that the number of crimes in that rectangle has been very high since 2014 and a transparent box means the number of arrests has been very low.

The first step is to calculate the number of arrest records in each rectangle. In the map, the size of each rectangle is 0.001 degrees of latitude and 0.001 degrees of longitude. You can find what rectangle each arrest record will be assigned to by rounding latitude and longitude to 3 digits. For example, loc1: -76.6352, 39.3103 and loc2: -76.6349, 39.3101 will be both part of the same rectangle with the center of -76.635, 39.310.

- Use the following command to find the frequency of each rectangle.

  ```
  loc_data= data %>%
          group_by(lng=round(Longitude,3),lat=round(Latitude,3)) %>%
          summarise(N=n())
  ```

- The lng and lat represent the center of each column. Add four columns (latL, latH, lngL, and lngH) to loc_data to show the range of latitude and longitude of each box. In other words, latL=lat-0.0005, latH=lat+0.0005, lngL=lng-0.0005, and lngH=lng+0.0005.

- Like ggplot, we use %>% operator to add different layers on the leaflet map. Use the command lines below to create a map.

  ```
  m=loc_data %>% leaflet() %>% addTiles() %>%
          setView(-76.6,39.31, zoom=12) %>%
          addProviderTiles(providers$Stamen.Toner, group = "Toner")%>%
          addLayersControl(baseGroups = c("Toner", "OSM"),
          options = layersControlOptions(collapsed = FALSE))
  ```

The first line creates a simple map. The second line sets the zoom level and the center of the map. The third line adds a black-white tile so users can see the heat map better. The fourth line adds the legend to the map so users can switch to the default tile (OSM) if they want to.

- In the leaflet, you can add different elements and shapes to the map. You can see many examples here. Read the description on that page and use addRectangles to create the rectangles. Since the loc_data dataframe has been already passed to the leaflet, you can use ~ to define a column. For example, lng1=~lngL means that the parameter lng1of addRectangles function is set to column lngL of loc_data dataframe.

- Once you created the rectangles, you may want to adjust the rectangles using the following parameters: fillOpacity = ~N/150, opacity = 0, fillColor = "red", label = ~N

Now that you have completed this part, we can move to app.R and try to complete both ui and server parts.

f) Completing the user interface(UI):

The user interface of the application is provided to you in the app.R file. Put app.R, data.csv, and usholidays.csv in an empty folder. Once you run the app.R, you should see the Plotly, Density, and Map tabs in your application. In this part, you are asked to complete the ui. Please note that since the server part of the application is almost blank, you will not see the outputs like the graphs, the map, and the table, yet.

- Change the title of the shiny app from Shiny Title to BPD Arrest.
- Add the fourth tab named "Data" with your choice of icon to the application. As the rest of the tabs, you should once add the tab inside the sidebarMenu function and once inside the tabItems function. Please note that when you add a new parameter (element) to a function in the UI part, you need to separate your newly added parameter from other parameters with a comma.
- Inside the fourth tabItem, use the command below to add a place for the datatable to be shown.
  dataTableOutput("myTable")
- Inside the rightSidebar function, add a link to the data source. Use the tags$a function with parameters href="https://data.baltimorecity.gov/Public-Safety/BPD-Arrests/3i3v-ibrt" and target="_blank". The target parameter opens the dataset link in a new tab.
- Run your app.R to make sure that your app successfully shows the new tab Data and the Data Source link.

g) Completing the server in the app.R:

- Copy your code from part b and paste into the line after "server <- function(input, output, session) {" in the app.R file.

- Cut the code for the first 5 bullets in part d (until creating data_hol dataframe) and paste it after the code from part b.
- Copy the code for part c and paste it inside the render function for plot1 output. Your graph is currently only working for the year 2014. There is a slider named year in the ui whose value can be set by users. Change any 2014 in your code to input$year so that your figure works for any year value given by users.
- Run your app.R. Your application should run successfully. The figure in tab Density should change and show the value of the slider when you change the slider. properly. Also, you can click on the play button under the slider to show an animation of how density will change over different years.
- Copy the rest of the code for part d and paste it inside the render function for plot2 output. There is a checkbox named holiday in the Plotly tab. Users should be able to show/hide the holiday points by checking/unchecking this checkbox. In other words, the following code that we had written before, should be added to f only when input$holiday==TRUE .

  ```
  f=f+geom_point(data= subset(data_hol, !is.na(Holiday)), color="purple")+
      geom_text(data=subset(data_hol,!is.na(Holiday)), aes(x=Date, y=N, label=Abb))
  ```

  Write a proper if statement inside the render function to do that.
- Run app.R and you should be able to see the time-series figure inside the Plotly tab. You should also be able to hide or show the holidays on the figure.
- Copy the code for part e and paste it inside the render function for myMap output. Run app.R and you should be able to see the map inside the Map tab.
- Learn from the structure of the render functions that are provided in the code and also in slide 30 of “Week5.pptx” to write a proper render function for the datatable shown in the Data tab. Place your render function after the code for myMap output. Put the following command inside your render function.

  ```
  return(datatable(data, rownames= FALSE))
  ```

- Run app.R and you should see a datatable format of data in the Data tab.