JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Individual Programming Assignment 3

Goal

Practice working with time-series data, and **plotly** and **leaflet** packages, as well as more advanced concepts in the **Shiny** app. The expected result is published on the following website: https://myazdi.shinyapps.io/PAssignment3/

Description

In this programming assignment, you will build a web app that visualizes the **data.csv** dataset. The BPD arrests data is a list of arrests since 2014 provided by the Baltimore Police Department.

It is recommended that you consult with the materials provided in Modules 1–6 to familiarize yourself with some examples of the functions mentioned in this assignment.

For parts a–e, you should create a new R script and run your code there. Once you can create the desired graphs, you will learn how to adjust your existing code for a Shiny web app and insert that code into the **app.R** in parts f and g; **app.R** is an incomplete Shiny app provided to you. Your final result should be published on https://www.shinyapps.io/, and your **app.R** file should be submitted to Blackboard.

Supplemental Materials: **data.csv**, **usholidays.csv**, and **app.R**

Submission: Your completed **app.R** and the **URL** of your web app.

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** 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 **as.Date** function to convert the **ArrestDate** column from characters to dates.
- You will use the **DT** package later to show the **data** data frame on the **Data** tab. The 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 the **Longitude** and **Latitude** columns to five digits.

c) Density graph

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

- Use the following command inside the **filter** function to filter the data for 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 by gender. 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 **geom_density** to create density functions. You can adjust the thickness of the lines by using the parameter **size**.
- Add the text annotation "2014" to the background as you did in Programming Assignment 2.
- Set the **labs**, **xlim**, and **ylim** of the graph properly.
- The color legend shows the first letter of each gender (for example, **"M"**). Use the **scale_color_discrete** function to change each one to the spelled-out format (that is, **"Male"**).
- You can decide the theme of your graph. I found the density unit confusing for users, so 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

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

- Read the **usholidays.csv** dataset, remove the first column (row index), and store it on the **holidays** variable.
- Convert the **Date** column from characters to dates.
- I found the name of each holiday too long to be shown in the figure, so I decided to use abbreviations instead. Use the following code to create a new column named **Abb** to hold the abbreviated **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 for each day using the following command:

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

- Use the **full_join** function to merge the data frame created above with the **holidays** data frame and store the combined frames on a variable named **data_hol**.
- Use the **geom_line** and **geom_smooth** functions 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 a **ggplotly** one using the following command:

```
ggplotly(f)
```

e) Map

Now you will create a heat map 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 the rectangle 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 which rectangle each arrest record will be assigned to by rounding latitude and longitude to three digits. For example, loc1: -76.6352, 39.3103 and loc2: -76.6349, 39.3101 will both be part of the same rectangle, which has a 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. Using the **mutate** function, add four columns (**latL**, **latH**, **lngL**, and **lngH**) to **loc_data** to show the range of latitude and longitude for each box. In other words, **latL=lat-0.0005**, **latH=lat+0.0005**, **lngL=lng-0.0005**, and **lngH=lng+0.0005**.
- As in ggplot, we use the **%>%** operator to add different layers to 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-and-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 Open Street Map (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 in the linked content and use **addRectangles** to create the rectangles. Because the **loc_data** data frame has already been passed to the leaflet, you can use **~** to define a column. For example, **lng1=~lngL** means that the parameter **lng1**of the **addRectangles** function is set to column **lngL** of the **loc_data** data frame.
- Once you create the rectangles, you may want to adjust them using the following parameters: **fillOpacity = ~N/150, opacity = 0, fillColor = "red", label = ~N**

fillOpacity = ~N/150, **fillColor = "red"** will handle the red color of the rectangles; **opacity=0** will remove the outlines of the rectangles; and **label = ~N** shows the number of observations when users hover over a rectangle.

Now that you have completed this part of the assignment, we can move to **app.R** and try to complete both the **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** file, you should see the **Plotly**, **Density**, and **Map** tabs in your application. You will now complete the **ui**. Please note that because the **server** part of the application is almost blank, you will not yet see the outputs, including the graphs, the map, and the table.

- Change the title of the Shiny app from **Shiny Title** to **BPD Arrest**.
- Add the fourth tab, named **Data**, to the application with your choice of icon. As with the rest of the tabs, you should add the tab once 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 to show the **datatable**:

```
dataTableOutput("myTable")
```

- Inside the fourth **tabItem**, add a link to the data source. Use the **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 it into the line after **server <- function(input, output, session) {** in the **app.R** file.
- Cut the code for the first five bullets in part d (until creating the **data_hol** data frame) and paste it after the code from part b.

- Copy the code for part c and paste it inside the **render** function for the **plot1** output. Your graph is currently only working for the year **2014**. A slider named **year** in the **ui** has a value that can be set by users. Change any **2014** in your code to **input$year** so that your figure works for any year value entered by a user.
- Run your **app.R**. Your application should run successfully. The figure in the **Density** tab should change to show the value of the slider when you change the slider. Also, clicking the **play** button under the slider will show an animation of how density will change over different years.
- Copy the rest of the code for part d and paste it inside the **render** function for the **plot2** output.
- A checkbox named **holiday** is in the **Plotly** tab. Users should be able to show/hide the holiday points by checking/unchecking this checkbox. In other words, the **geom_point** and **geom_text** code that we wrote before should be added to part f only when **input$holiday==TRUE**. Replace the **geom_point** and **geom_text** code with the following lines of code:

```
if(input$holiday==TRUE){
f=f+
geom_point(data= subset(data3, !is.na(Holiday)), color="purple")+
geom_text(data=subset(data3,!is.na(Holiday)), aes(x=ArrestDate, y=N, label=Abb), size=3)
}
```

- 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 the **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 to write a proper **render** function for the **datatable** shown in the **Data** tab. Place your **render** function after the code for the **myMap** output. Put the following command inside your **render** function:

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

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

h) Publish your work on the shinyapps.io server

The steps to publish a Shiny app are as follows:

1. Run your **app.R**.
2. Click the **Publish** button at the top right of the page.
3. Click the **Add new account** link at the top right of the page.
4. Click https://www.shinyapps.io/.
5. There is a box for pasting your tokens. Follow the instructions on **Module 4 Dashboard Templates in Shiny** to copy the tokens.
6. Name your application properly and click the **Publish** button.
7. Wait for RStudio to publish your app and keep the link to your web app. Do not forget to include the **URL** of your web app in the comments section when you submit your **app.R** file on Blackboard.