JOHNS HOPKINS
CAREY BUSINESS SCHOOL

Individual Programming Assignment 2

Goal

The goal of this assignment is to practice transforming data using dplyr package and also refining ggplots using mode advanced functions of ggplot2. The expected result is published on the following website: https://myazdi.shinyapps.io/PAssignment-2/.

Note: You do not have to publish your app on the web.

Description

In this programming assignment, you will build a web app that provides a visual summary of global education using the data provided by the World Bank.
It is recommended that you consult with the course materials for Modules 1–4 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 your graphs into the **app.R** in part f; **app.R** is an incomplete Shiny app provided to you. You should submit only the completed **app.R** file through Blackboard.

Supplemental Materials: “avgYearEdu.csv”, **”education.csv”**, **“GDP.csv”**, and **app.R**

Submission: Your completed **app.R**

a. Required libraries

You need the following packages for this assignment:

- **tidyverse** for data manipulation and visualization
- **ggthemes** for visually appealing built-in themes
- **scales** for the dollar format of the numbers on the *y*-axis

b. Reading data

- Use the **read_csv** function from the **tidyverse** package to read all three CSV files mentioned above.
- Select all columns from the education dataset except for the second, fourth, fifth, sixth, and seventh columns. You can do this using the **select** function.
- Use the following command to change the names of the remaining columns of the education dataset to **"country"**, **"year"**, **"female"**, and **"male"**:
  colnames(avgYearEdu) <- c("country","year","avgYearEdu")
- Select all columns from the **avgYearEdu** dataset except for the second column.
- Change the names of the remaining columns of the **avgYearEdu** dataset to **"country"**, **"year"**, and **"avgYearEdu"**.
- Select all columns from the GDP dataset except for the second column.
- Change the names of the remaining columns of the GDP dataset to **"country"**, **"year"**, **"GDPperCapita"**, and **"population"**.

c. Data manipulation

- Use the **inner_join** function and both the **"country"** and **"year"** columns to merge all three datasets. For example, the following command merges datasets **education** and **avgYearEdu**:
  education %>% inner_join(avgYearEdu, by = c("country","year"))

- Save the merged dataset on a new variable called **data**.
- Use the **na.omit** function to remove all the NAs from **data**.
- As you can see in the legend of the plot in the website, the **population** should be shown in millions. Use the **mutate** function to divide the **population** by 1,000,000 and overwrite it on the **population** column.
- Another observation is that data is color coded based on whether the number of out-of-school children is higher among females or males. There is no such column in the dataset. You should add a new column, such as **gender**, and set it equal to **female** for rows in which the value of **female** is greater than the value of **male**, and set it to **male** otherwise. You can do it this way:
  mutate(gender=case_when(female>male~"Female", female <= male~"Male"))

d. Visualization

- At this time, you are only interested in visualizing the data for the year 2000. Create a variable named **thisYear** and set its value to 2000.
- Filter the data with the condition **year==thisYear** and save the result on the **filteredData** variable.
- Now you are ready to run the **ggplot** and **geom_point** functions on the **filteredData** variable. Use aesthetics **x**, **y**, **color**, and **size** to properly plot the points.
- Use the **annotate** function to add the value of **thisYear** to the center of the picture. There are different shades of gray in R. I used **"grey80"** for the color of the text. After some trial and error, I came up with the following parameters for this annotation: **size=20**, **x=7.5**, and **y=35000**.
- Add the **theme_wsj()** theme to the ggplot to give your plot an appealing appearance.
- Save the ggplot result on a variable named **p** and show **p**. You should have a plot similar to Figure 1 by now.

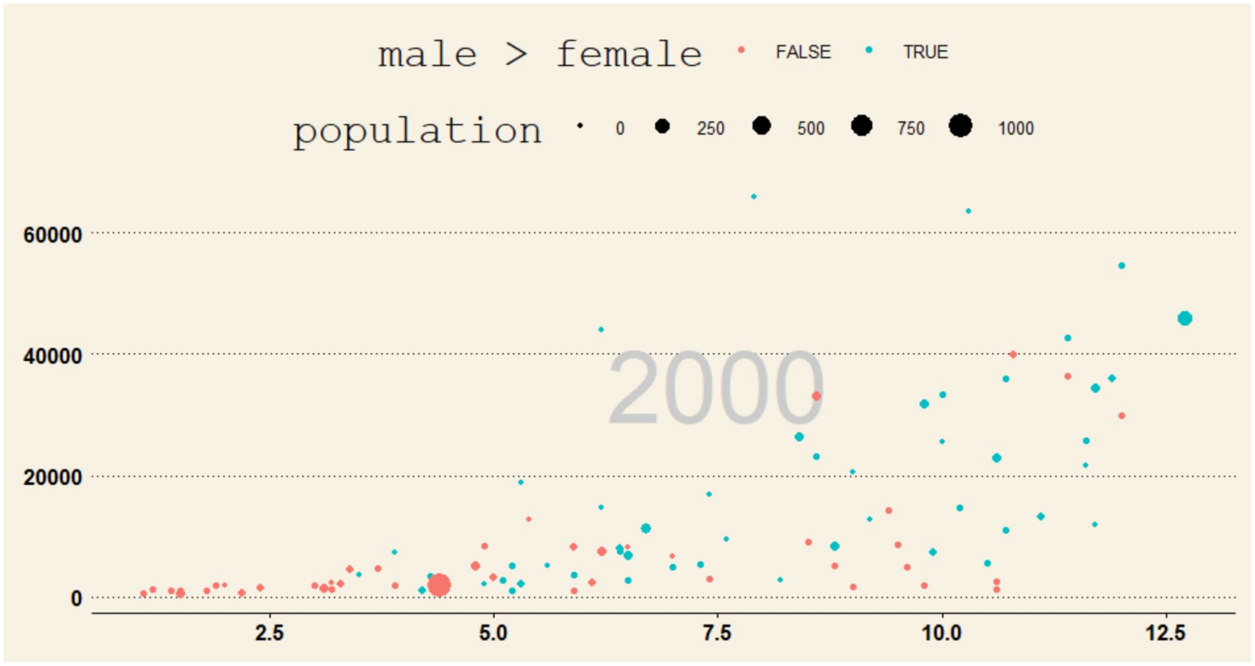

Figure 1. A snapshot of an incomplete version of the plot.

- Note: As you can see in Figure 1, points are shown on top of the label "2000." You should change the sequence of your code if your plot does not look like that.
- The legends on top look large. In addition, the plot and axis titles are missing. The reason is that the theme you have used, **theme_wsj()**, has these default settings. Add the following to the end of your code to bring these titles back to the figure in the correct size:

```
theme(plot.title = element_text(size = rel(0.6)),
        legend.title = element_text(size = rel(0.5)),
        axis.title=element_text(face = "bold", size = rel(0.6)))
```

- Now you can add the function **labs()** to the end of your code for plot **p** to rename the labels as shown on the plot in the website.
- The package scale provides several text formats. Add the following command to your code to give your *y*-axis a dollar format:

```
scale_y_continuous(labels = dollar)
```

- At this stage, expect to see a plot like the one in Figure 2.

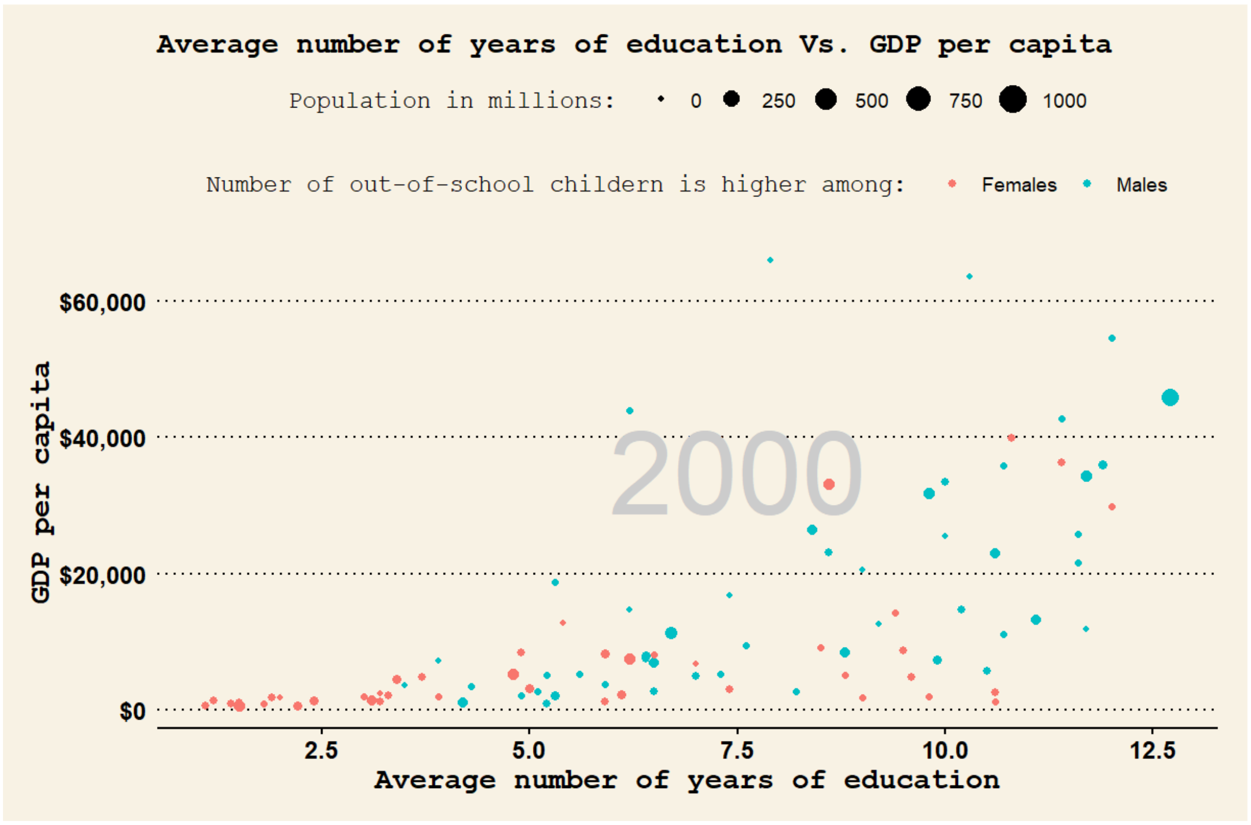

Figure 2. A snapshot of an incomplete version of the plot.

- The last step is to add the country names to the plot. There is not enough space to add each country's name. We will plot only a few of the countries with the largest populations. The following two lines of code store the names of the top five countries in terms of population size on the variable **countriesToShow**. The code explanation is as follows: **arrange(-population)** sorts the **filteredData** in descending order, while **pull(country)** takes out the values in the country column as a vector, and **head(cnum)** will filter only five numbers from the top. Include the codes below in the data manipulation part:

  cnum=5

  countriesToShow= filteredData%>% arrange(-population) %>%

pull(country) %>% head(cnum)

- Now you can easily provide the **geom_text** function with a subset of **filteredData** containing the information for these five countries. Include the command below to the end of your code:

  ```
  p= p + geom_text(data= filteredData %>% filter(country %in% countriesToShow),
                   mapping = aes(label = country))+
  ```
- You are creating a subset of **filteredData** that is related to one of the five countries in the **countriesToShow** variable. You can find more examples of **gemo_text** here.
- Note: The size of the countries' labels shown in the picture should not be dependent on the population. In other words, the size of all the countries' labels should be the same. If this is not the case for you, you probably located the **aes size** in the wrong place. You may want to change the location of **aes** to make sure that **aes size** only applies to points and not the texts.
- Display your plot (**p**) and you should see a plot similar to the plot in the website.

e. Scale your work

- In this part, you will build a Shiny application that lets users interact with the graph created in parts a–d.
- You will adjust your code slightly to be able to replicate your work based on users' input. The options for interaction are as follows:
  1. Users should be able to request a year from among the years 1970–2013 (variable **thisYear**).
  2. Users should be able to set the number of country names to be shown (variable **cnum**). Note: This item is different from what you see in the original project.
- Put **app.R** and all three CSV files in an empty folder.
- Copy and paste your code for parts a–d inside the existing **app.R**. Read the comments in the **app.R** for the exact locations.

- Remove **thisYear=2000** and replace it with **thisYear=input$year**, and replace **cnum=5** with **cnum=input$num**. By doing that, you are telling your application to filter data based on the year provided by users. Also, you are letting users define how many countries to show.
- Now your application should work properly. Ideally, users should be able to compare the plots from different years. Without manually setting the *x*-axis, *y*-axis, and size ranges, each plot would have a different range for *x*, *y*, and size. It would be very difficult to observe the trend over the years with different scales. Replace the **scale_y_continuous(labels = dollar)** in your code with the following command:

  ```
  scale_size_continuous(limits = c(0, 1000))+
  scale_y_continuous(labels = dollar, limits = c(0,70000))+
  scale_x_continuous(limits = c(0,15))
  ```

Q&A:

The following is a list of some of the common issues that you may face in this assignment. The solutions are provided. These are not big issues, and they do not require that you change your code. No points will be deducted for any of the following issues.

- Q: My legend order is different from the one on the website. How can I change this?
  A: ggplot will pick the order based on algorithms. However, you can manually set the order with the following code:

  ```
  p + guides(size = guide_legend(order = 1),
             color = guide_legend(order=2))
  ```

- Q: I would like to avoid text overlap with the point. How can I remove this overlap?
  A: You can install and load the package **ggrepel** and use the following command:

  ```
  p + geom_text_repel( )
  ```