Intro to Data Science - HW 6


```
# Enter your name here:  Ryan Tervo
# Course Number:         IST 687
# Assignement Name:      Homework #6
# Due Date:              21 Nov 2022
# Submitted Date:        21 Nov 2022
```

Attribution statement: (choose only one and delete the rest)

```
# 1. I did this homework by myself, with help from the book and the professor.
```

Last assignment we explored **data visualization** in R using the **ggplot2** package. This homework continues to use ggplot, but this time, with maps. In addition, we will merge datasets using the built-in **merge( )** function, which provides a similar capability to a **JOIN in SQL** (don't worry if you do not know SQL). Many analytical strategies require joining data from different sources based on a **"key"** – a field that two datasets have in common.

Step 1: Load the population data

A. Read the following JSON file, https://intro-datascience.s3.us-east-2.amazonaws.com/cities.json and store it in a variable called **pop**.

Examine the resulting pop dataframe and add comments explaining what each column contains.

```
#   LOAD LIBRARIES:
library(jsonlite)
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──
##  ggplot2 3.4.0      purrr   0.3.5
##  tibble  3.1.8      dplyr   1.0.10
##  tidyr   1.2.1      stringr 1.4.1
##  readr   2.1.3      forcats 0.5.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
##  dplyr::filter()  masks stats::filter()
##  purrr::flatten() masks jsonlite::flatten()
##  dplyr::lag()     masks stats::lag()
```

```
library(dplyr)
library(purrr)
library(maps)
```

```
##
## Attaching package: 'maps'
##
```

```
## The following object is masked from 'package:purrr':
##
##     map
```

```
#   DEFINE VARIABLES:
dataset <- url("https://intro-datascience.s3.us-east-2.amazonaws.com/cities.json")
pop <- jsonlite::fromJSON(dataset)

#   INSPECT/INVESTIGATE DATASET:
str(pop)
```

```
## 'data.frame':    1000 obs. of  7 variables:
##  $ city                    : chr  "New York" "Los Angeles" "Chicago" "Houston" ...
##  $ growth_from_2000_to_2013: chr  "4.8%" "4.8%" "-6.1%" "11.0%" ...
##  $ latitude                : num  40.7 34.1 41.9 29.8 40 ...
##  $ longitude               : num  -74 -118.2 -87.6 -95.4 -75.2 ...
##  $ population              : chr  "8405837" "3884307" "2718782" "2195914" ...
##  $ rank                    : chr  "1" "2" "3" "4" ...
##  $ state                   : chr  "New York" "California" "Illinois" "Texas" ...
```

```
print("", quote = FALSE)
```

```
## [1]
```

```
head(pop)
```

```
##           city growth_from_2000_to_2013 latitude  longitude population rank
## 1     New York                     4.8% 40.71278  -74.00594    8405837    1
## 2  Los Angeles                     4.8% 34.05223 -118.24368    3884307    2
## 3      Chicago                    -6.1% 41.87811  -87.62980    2718782    3
## 4      Houston                    11.0% 29.76043  -95.36980    2195914    4
## 5 Philadelphia                     2.6% 39.95258  -75.16522    1553165    5
## 6      Phoenix                    14.0% 33.44838 -112.07404    1513367    6
##          state
## 1     New York
## 2   California
## 3     Illinois
## 4        Texas
## 5 Pennsylvania
## 6      Arizona
```

```
################################################################################
#   COMMENTS EXPLAINING WHAT EACH COLUMN CONTAINS:
################################################################################
#   Column Name                   Description
#   ------------------------------------------------------------------------------------------
------
#   city                      |   City Name, type is string or character
#   growith from 2000 - 2013  |   Growth,  % of growth between the years 2000 - 2013. Type is
```

```
string but it shows % signs
#   latitude                |   Latitude coordinate, stored as numeric, likely center of cit
y
#   longitude               |   Longitude coordinate, stored as numeric, likely center of ci
ty
#   population              |   population.  It's unclear if it's the population in 2000, 20
13, or current, likely 2013, stored as character or string
#   rank                    |   City rank based on highest population. Stored as character o
r string.
#   state                   |   State the city is in.  Stored as string or character.
```

B. Calculate the **average population** in the dataframe. Why is using mean() directly not working? Find a way to correct the data type of this variable so you can calculate the average (and then calculate the average)

Hint: use **str(pop)** or **glimpse(pop)** to help understand the dataframe

```
#   CONVERT POPULATON STRING DATA TO NUMERIC
pop$population <- as.numeric(pop$population)

#   DETERMINE AVERAGE POP
averagePop <- mean(pop$population)

#   DISPLAY RESULTS:
printVal <- format(round(averagePop, 0), nsmall = 0, big.mark =",")
printStringB <- paste('The average population is: ', printVal, sep = "")
print(printStringB, quote = FALSE)
```

```
## [1] The average population is: 131,132
```

C. What is the population of the smallest city in the dataframe? Which state is it in?

```
#   CONVERT POPULATON STRING DATA TO NUMERIC
smallestPop  <- min(pop$population)
smallestPopPrint <- format(round(smallestPop, 0), nsmall = 0, big.mark =",")
location     <- pop[pop$population == min(pop$population), c('city', 'state')]

#   DISPLAY RESULTS
printStringC <- paste('The smallest city populaton is ', smallestPopPrint, '.  ',  location[1]
, ' in the state of ', location[2], '.', sep = "")
pringStringC2 <- 'This is a test'
print(printStringC, quote = FALSE)
```

```
## [1] The smallest city populaton is 36,877.  Panama City in the state of Florida.
```

Step 2: Merge the population data with the state name data

D. Read in the state name .csv file from the URL below into a dataframe named **abbr** (for “abbreviation”) – make sure to use the read_csv() function from the tidyverse package:
https://intro-datascience.s3.us-east-2.amazonaws.com/statesInfo.csv

```
#   DEFINE THE VARIABLES:
fileName <- 'https://intro-datascience.s3.us-east-2.amazonaws.com/statesInfo.csv'

#   READ EXCEL FILE USING WEBSITE FILE:
abbr  <- data.frame(read_csv(fileName))
```

```
## Rows: 51 Columns: 2
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (2): State, Abbreviation
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
#   INSPECT DOWNLOADED DATA
head(abbr)
```

```
##        State Abbreviation
## 1    Alabama           AL
## 2     Alaska           AK
## 3    Arizona           AZ
## 4   Arkansas           AR
## 5 California           CA
## 6   Colorado           CO
```

E. To successfully merge the dataframe **pop** with the **abbr** dataframe, we need to identify a **column they have in common** which will serve as the **"key"** to merge on. One column both dataframes have is the **state column**. The only problem is the slight column name discrepancy – in **pop**, the column is called **"state"** and in **abbr** – **"State."** These names need to be reconciled for the merge() function to work. Find a way to rename **abbr's "State"** to **match** the **state column in pop**.

```
#   UPDATE COL NAME:  from 'State' to 'state'
colnames(abbr)[colnames(abbr) == "State"] ="state"

#   VERIFY UPDATE:
print('Column names are shown below:', quote = FALSE)
```

```
## [1] Column names are shown below:
```

```
print(colnames(abbr), quote = FALSE)
```

```
## [1] state        Abbreviation
```

F. Merge the two dataframes (using the **'state' column** from both dataframes), storing the resulting dataframe in **dfNew**.

```
#   MERGE DATAFRAMES USING 'state' COLUMN:
dfNew <- merge(pop, abbr, by = "state")
```

G. Review the structure of **dfNew** and explain the columns (aka attributes) in that dataframe.

```
#   DISPLAY 'dfNew'
str(dfNew)
```

```
## 'data.frame':    1000 obs. of  8 variables:
##  $ state                  : chr  "Alabama" "Alabama" "Alabama" "Alabama" ...
##  $ city                   : chr  "Auburn" "Florence" "Huntsville" "Dothan" ...
##  $ growth_from_2000_to_2013: chr  "26.4%" "10.2%" "16.3%" "16.6%" ...
##  $ latitude               : num  32.6 34.8 34.7 31.2 33.5 ...
##  $ longitude              : num  -85.5 -87.7 -86.6 -85.4 -86.8 ...
##  $ population             : num  58582 40059 186254 68001 212113 ...
##  $ rank                   : chr  "615" "922" "126" "502" ...
##  $ Abbreviation           : chr  "AL" "AL" "AL" "AL" ...
```

```
print("", quote = FALSE)
```

```
## [1]
```

```
head(dfNew)
```

```
##     state        city growth_from_2000_to_2013 latitude longitude population
## 1 Alabama      Auburn                    26.4% 32.60986 -85.48078      58582
## 2 Alabama    Florence                    10.2% 34.79981 -87.67725      40059
## 3 Alabama  Huntsville                    16.3% 34.73037 -86.58610     186254
## 4 Alabama      Dothan                    16.6% 31.22323 -85.39049      68001
## 5 Alabama  Birmingham                   -12.3% 33.52066 -86.80249     212113
## 6 Alabama Phenix City                    31.9% 32.47098 -85.00077      37498
##   rank Abbreviation
## 1  615           AL
## 2  922           AL
## 3  126           AL
## 4  502           AL
## 5  101           AL
## 6  983           AL
```

```
#############################################################
#   COMMENTS EXPLAINING WHAT EACH COLUMN CONTAINS:
#############################################################
#   Column Name                   Description
#   ------------------------------------------------------------------------------------------
------
#   state                     |   State the city is in.  Stored as string or character.
#   city                      |   City Name, type is string or character
#   growith from 2000 - 2013  |   Growth,  % of growth between the years 2000 - 2013. Type is
string but it shows % signs
#   latitude                  |   Latitude coordinate, stored as numeric, likely center of cit
y
```

```
#   longitude                 |   Longitude coordinate, stored as numeric, likely center of ci
ty
#   population                |   population.  It's unclear if it's the population in 2000, 20
13, or current, likely 2013, stored as character or string
#   rank                      |   City rank based on highest population. Stored as character o
r string.
#   Abbreviation              |   Shows 2 letter state abbreviation.  Data came from abbr df.
```

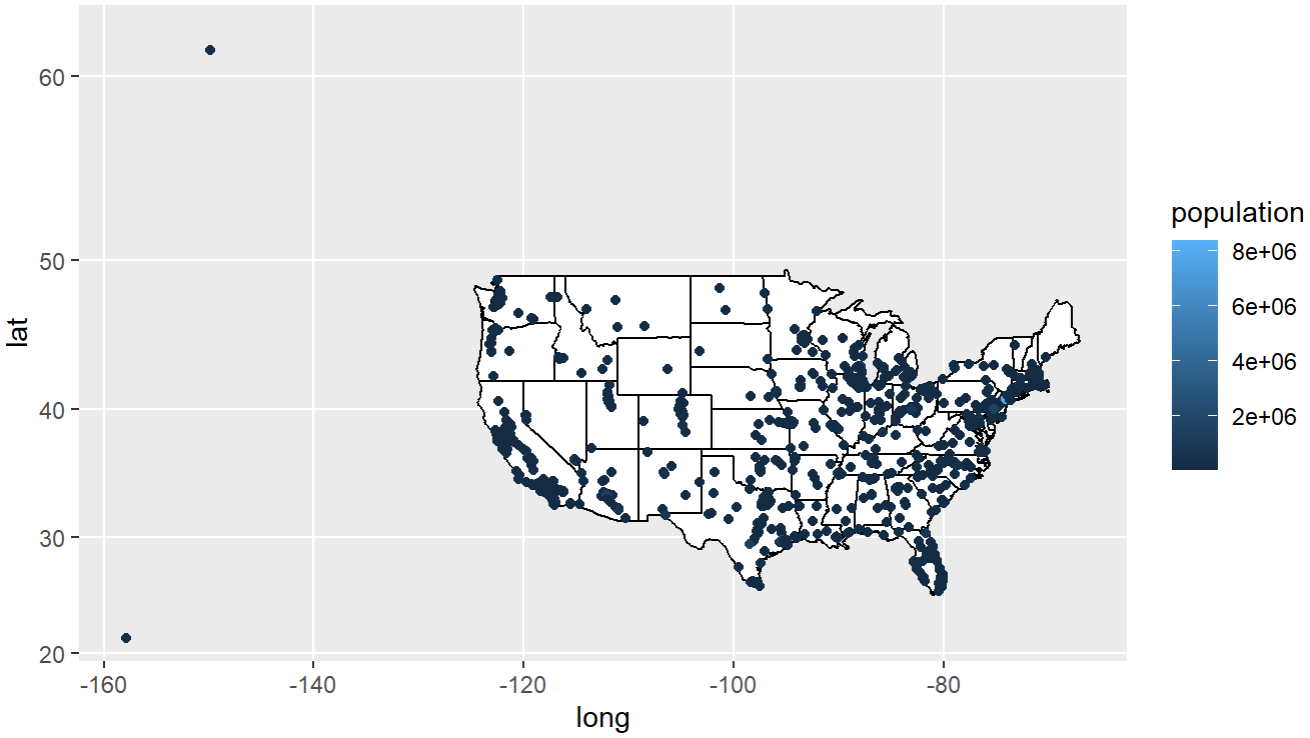
Step 3: Visualize the data

H. Plot points (on top of a map of the US) for **each city**. Have the **color** represent the **population**.

```
#   BUILD MAP:
state_geomDF <- map_data("state")
map.simple   <- ggplot(state_geomDF)
map.simple   <- map.simple + geom_polygon(color = "black", fill = "white", aes(x = long, y = l
at, group = group))
map.simple   <- map.simple + coord_map()

#  PLACE LONG/LAT OF CITIES ON THE MAP:
map.simple <- map.simple + geom_point(data = dfNew, aes(y = latitude, x = longitude, color = p
opulation))

#   DISPLAY MAP:
map.simple
```



I. Add a block comment that criticizes the resulting map. It's not very good.

```
#   1)  Hawaii and Alaska state outlines are missing but the city info is still there.
#   2)  The color code is not helpful.  Adds virtually no value.
#   3)  Map has a lot of "dead" space
```

Step 4: Group by State

J. Use group_by and summarise to make a dataframe of state-by-state population. Store the result in **dfSimple**.

```
#   Create dfSimple0 using aggregate function:
dfSimple0 <- aggregate(dfNew$population, by = list(dfNew$state), FUN = sum)

#   Create dfSimple using group_by and summarize.
dfSimple <- dfNew %>% group_by(state) %>% summarise(sum(population))

#   Display results:
head(dfSimple)
```

```
## # A tibble: 6 × 2
##   state      `sum(population)`
##   <chr>                  <dbl>
## 1 Alabama              1279813
## 2 Alaska                300950
## 3 Arizona              4691466
## 4 Arkansas              787011
## 5 California          27910620
## 6 Colorado             3012284
```

K. Name the most and least populous states in **dfSimple** and show the code you used to determine them.

```
#   CONVERT TO DATA FRAME AND RENAME COLUMN WITH '()':
dfSimple <- data.frame(dfSimple)
colnames(dfSimple)[2] = "sumPopulation"
head(dfSimple)
```

```
##        state sumPopulation
## 1    Alabama       1279813
## 2     Alaska        300950
## 3    Arizona       4691466
## 4   Arkansas        787011
## 5 California      27910620
## 6   Colorado       3012284
```

```
#   GET STATE NAMES WITH LEAST AND MOST POPULOUS:
mostPopulousState  <- dfSimple[dfSimple$sumPopulation == max(dfSimple$sumPopulation), 1]
leastPopulousState <- dfSimple[dfSimple$sumPopulation == min(dfSimple$sumPopulation), 1]

#   DISPLAY RESULTS:
```

```
printString1 = paste('The most  populous state is ', mostPopulousState, sep = '')
printString2 = paste('The least populous state is ', leastPopulousState, sep = '')

print(printString1, quote = FALSE)
```

```
## [1] The most  populous state is California
```

```
print(printString2, quote = FALSE)
```

```
## [1] The least populous state is Vermont
```

Step 5: Create a map of the U.S., with the color of the state representing the state population

L. Make sure to expand the limits correctly and that you have used **coord_map** appropriately.

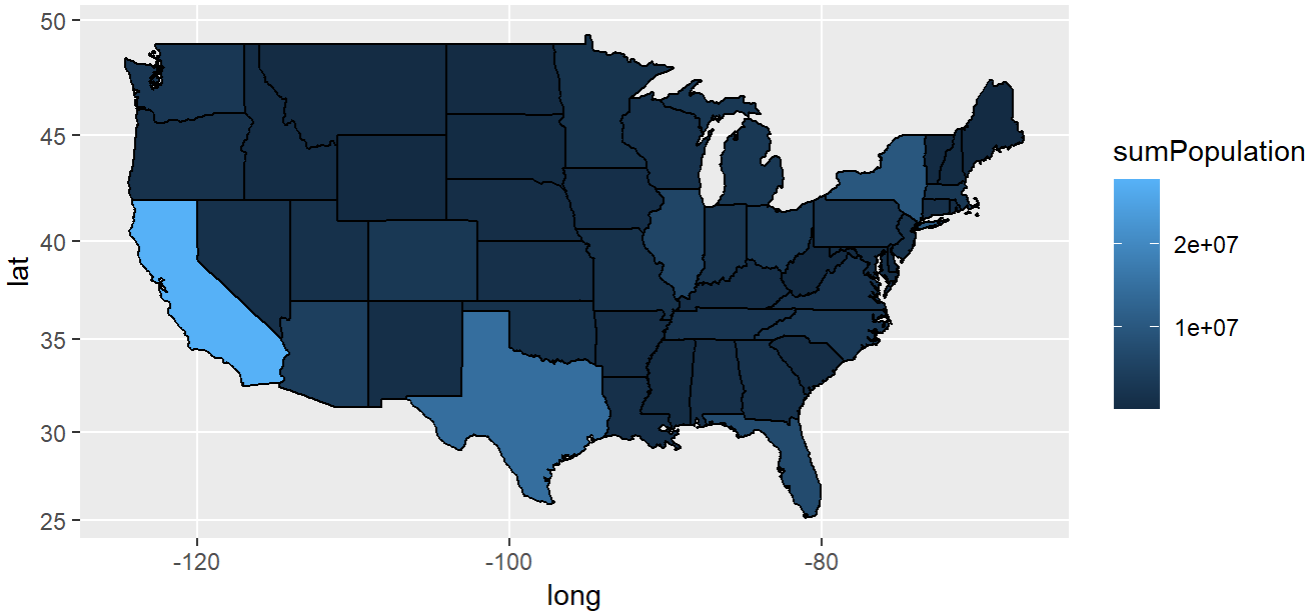
```
#   UPDATE DATAFRAMES SO THAT THE INFORMATION CAN BE MERGED:
#   Update df_simple dataframe so it can be successfully merged.
colnames(dfSimple)[1] = "region"
dfSimple$region <-  tolower(dfSimple$region)

#   Set map_data to dataframe.
map2 <- data.frame(map_data('state'))

#   Merge into dfSimple and map2 into a single dataframe
df_Last <- merge(map2, dfSimple, by ="region")

#   CREATE MAP
map.L <- ggplot(df_Last)
map.L <- map.L + geom_polygon(color = "black", aes(x = long, y = lat, group = group, fill = su
mPopulation))
map.L <- map.L + coord_map()

#   DISPLAY MAP!
map.L
```


50
45
40
35
30
25
lat
-120
-100
-80
long
sumPopulation
2e+07
1e+07