MATH 3190 Homework 2

Focus: Notes 4 and 5

Due February 17, 2024

Your homework should be completed in R Markdown or Quarto and Knitted to an html document. You will 'turn in' this homework by uploading to your GitHub Math_3190_Assignment repository in the Homework directory.

Problem 1 (35 points)

Part a (30 points)

In Homework 1, you created an **R** package related to the basketball dataset with four functions in it. Now create a shiny app that relies on some of these functions.

- Let the user enter the name of the team they want information for (in a text box), and then your app should return a table that includes their wins, losses, and winning percentage (like your function you made in Problem 2j of HW 1). Look into the `knitr::kable()` function for outputting a table that looks nice.
- Have your app show a plot for something related to that entered team.
- If the user enters a team name that is not in the dataset, give a message than the user needs to enter a valid team name, and return a list of all the teams in alphabetical order.

```
library(shiny)
library(dplyr)
library(stringr)
library(collegeBasketball.bs)

cbbga24 <- read.fwf("http://kenpom.com/cbbga24.txt",
                    widths = c(10, 24, 3, 24, 3, 2, 20),
                    strip.white=TRUE) |>
  as_tibble() |>
  rename('Date' = V1,
         'Home_Team' = V2,
         'Home_Score' = V3,
         'Visiting_Team' = V4,
         'Visiting_Score' = V5,
         'Site_Type' = V6,
         'Game_Site' = V7) |>
  mutate(Margin = Home_Score - Visiting_Score ) |>
  arrange(Home_Team) |>
  select(!c(Site_Type, Game_Site))
```

```r
all_teams_df <- tibble(Team = unique(c(cbbga24$Home_Team, cbbga24$Visiting_Team)))

## College Team App build
ui <- fluidPage(
  sidebarLayout(

    sidebarPanel(
      #selectInput('team_select', 'Select Home or Visiting Team:', choices = c('Home_Team', 'Visiting_T

      uiOutput("team_search_ui"),
      actionButton("confirm_selection", "Please Click to Confirm")

    ),

    mainPanel(
      tableOutput('teamRecord'),
      plotOutput('thePlot'),

      tags$head(
        tags$style(
          '.top-text {
            position: absolute;
            top: 10%;
            left: 50%;
            transform: translateX(-50%);
          }'
        )
      ),
      div(class = "top-text", h3(textOutput('plot_Unmatched'))
      )


    )

  )
)

server <- function(input, output, session) {

  output$team_search_ui <- renderUI({
    selectizeInput('team_search', "Search Team by Typing:", choices = NULL)
  })

  observe({
    updateSelectizeInput(session, 'team_search', choices = all_teams_df$Team)
  })


  output$teamRecord <- function(){
    if (input$team_search %in% all_teams_df[['Team']]) {
    req(input$confirm_selection)
    collegeBasketball.bs::team_Record(cbbga24, input$team_search) |>
      knitr::kable(format = "html") |>
```

```
      kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
    }
  }


  output$thePlot <- renderPlot({

    if (input$team_search %in% all_teams_df[['Team']]) {
      req(input$confirm_selection)

      # wins and losses
      result <- collegeBasketball.bs::team_Record(cbbga24, input$team_search)
      wins <- result$Wins
      losses <- result$Losses

      # bar plot
      ggplot(data = data.frame(Result = c("Wins", "Losses"), Count = c(wins, losses)),
             aes(x = Result, y = Count)) +
        geom_bar(stat = "identity", fill = c("slategray3", "slategray")) +
        labs(x = "Outcome", y = "Count", title = "Wins vs Losses")+
        theme(axis.title = element_text(size = 14),
              legend.title = element_text(size = 14),
              plot.title = element_text(size = 18, face = "bold"),
              panel.background = element_rect(fill = 'ivory',
                                              color = 'navajowhite3', size=2,
                                              linetype = "solid"))
    } else {

        output$plot_Unmatched <- renderText({
          if ( !(input$team_search %in% all_teams_df[['Team']]) ) {
            paste("You have not typed an available team's name,
              please type a few letters of the teams name and select from list")
          }
        })

      }
  })


}

shinyApp(server = server, ui = ui)
```

Response: Considering the lengthy list of names and strings which may not be immediately apparent

Part b (5 points)

Add the shiny app to an `inst/` directory in the package you created in Problem 2l of HW 1 and create a function that can be used to call your Shiny app. Be sure to document it. You can see your `mypackage` package for how I did this with the correlation app. Update your **R** package on GitHub so I can install it and run your app.

Problem 2 (65 points)

Now it is time to practice using SQL. First, we will need some files that are worthy of being treated as databases. For this, you will need a little more than 1 GB of hard drive space. Alternatively, you should be able to store these files on your OneDrive. Go ahead and download all of the files from this link: https://drive.google.com/drive/folders/1X1hKpTZhoCNGz30ZZeJbets8pRXs_8JQ?usp=share_link. You can also click on the "flight data" folder name at the top of the page and click "Download" to download the entire folder. While the `airports.csv` and `airlines.csv` files are quite small, the `flights.csv` is over 565 MB in size and contains the records of 5,819,079 flights in USA in 2015. These data came from the Bureau of Transportation Statistics.

Part a (5 points)

While it is probable that your computer could read in this data set in its entirety, we are going to avoid that and instead make a database that we can access using SQL. Follow the example on slides 55-57 of Notes 5 to create a `.sqlite` file that contains the tables `flights` (from `flights.csv`), `airports` (from `airports.csv`), and `airlines` (from `airlines.csv`).

```
#library(DBI)
#library(RSQLite) #depends the type of database


csv_to_sqlite( csv_file = 'flights.csv',
               sqlite_file = 'airBTS.sqlite',
               table_name = 'flights')

csv_to_sqlite( csv_file = 'airports.csv',
               sqlite_file = 'airBTS.sqlite',
               table_name = 'airports')

csv_to_sqlite( csv_file = 'airlines.csv',
               sqlite_file = 'airBTS.sqlite',
               table_name = 'airlines')
```

Part b (2 points)

Connect to the `.sqlite` database that you created in Part a like we did on slide 52 of Notes 5. Then use the `dbListTable()` function on that connection to list all the tables in the database. There should be three tables that you added in Part a.

```
air_con <- DBI::dbConnect(RSQLite::SQLite(),
                          'airBTS.sqlite')

dbListTables(air_con) #list all database tables
```

```
## character(0)
```

Part c (1 point)

There are many different types of SQL databases: MySQL, SQLite, SQL Server, etc. We created a SQLite database in Part a. A way to see all the columns of a table in an SQLite database is with the command

`PRAGMA table_info('table_name')`. Using your connection from Part b, use the `dbGetQuery()` function in an **R** code chunk to print all the column names from the `flights` table. This is useful when we have a large file that is hard to even open to see what variables are in there.

```
DBI::dbGetQuery(air_con, 'PRAGMA table_info(flights) ') |> print()
```

Part d (24 points)

Again using the `dbGetQuery()` function (one query per part) in an **R** code chunk, use SQL statements to answer the following questions. Note that distances are in miles.

1. What is the average distance of all flights in the US in 2015?
2. What is the minimum flight distance of all flights in the US in 2015?
3. What is the maximum flight distance of all flights in the US in 2015?
4. Between what two airports did the minimum distance flight take place?
5. Between what two airports did the maximum distance flight take place? The `SELECT DISTINCT` command will be useful here.
6. You might notice that some airports are coded with numbers instead of their IATA codes. Repeat item 5, but do not include the airports that begin with a "1".
7. While you may know what these airports codes are, it would be nice if the full name of the airport was printed. The airport code information is in the `airport` table in your database in the `IATA_CODE` variable. Use the `INNER JOIN` statement to output the airport name for the two airports that have the minimum flight distance. Note that this command can take a few seconds to run.
8. Repeat item 7 with the maximum flight distances.

```
#1. average distance of all flights in the US in 2015
DBI::dbGetQuery(air_con, 'SELECT AVG(DISTANCE) AS avg_dist
            FROM flights
            WHERE YEAR = 2015') |> print()

#2. minimum flight distance of all flights in the US in 2015?
DBI::dbGetQuery(air_con, 'SELECT MIN(DISTANCE) AS min_dist
            FROM flights
            WHERE YEAR = 2015') |> print()

#3. maximum flight distance of all flights in the US in 2015?
DBI::dbGetQuery(air_con, 'SELECT MAX(DISTANCE) AS max_dist
            FROM flights
            WHERE YEAR = 2015') |> print()

#4. two airports between which the minimum distance flight took place
DBI::dbGetQuery(air_con, 'SELECT ORIGIN_AIRPORT,DESTINATION_AIRPORT AS two_AirportsMin            FROM
            ORDER BY DISTANCE ASC
            LIMIT 1') |> print()

#or rather:
DBI::dbGetQuery(air_con, 'SELECT DISTINCT ORIGIN_AIRPORT,DESTINATION_AIRPORT AS two_AirportsMin
            WHERE DISTANCE = (SELECT MIN(DISTANCE) FROM flights)') |> print()

#5. Between what two airports did the maximum distance flight take place? The `SELECT DISTINCT` command
DBI::dbGetQuery(air_con, 'SELECT DISTINCT ORIGIN_AIRPORT,DESTINATION_AIRPORT AS two_AirportsMax
            FROM flights
```

```
            WHERE DISTANCE = (SELECT MAX(DISTANCE) FROM flights)')

#6.  Repeat item 5, but do not include the airports that begin with a "1" IATA code.
DBI::dbGetQuery(air_con, 'SELECT DISTINCT ORIGIN_AIRPORT,DESTINATION_AIRPORT
            FROM flights
            WHERE DISTANCE = (SELECT MAX(DISTANCE) FROM flights) AND DESTINATION_AIRPORT NOT LIKE "1%" AI


#7. need to make an "aliased" form of `airports` table with an "alias" which then applies the dot opera
DBI::dbGetQuery(air_con, 'SELECT DISTINCT a1.AIRPORT, a2.AIRPORT
                  FROM flights
                  INNER JOIN airports a1
                  ON  flights.ORIGIN_AIRPORT = a1.IATA_CODE
                  INNER JOIN airports a2
                  ON  flights.DESTINATION_AIRPORT = a2.IATA_CODE
                  WHERE DISTANCE = (SELECT MIN(DISTANCE) FROM flights)'
)

#8.
DBI::dbGetQuery(air_con, 'SELECT DISTINCT a1.AIRPORT, a2.AIRPORT
                  FROM flights
                  INNER JOIN airports a1
                  ON  flights.ORIGIN_AIRPORT = a1.IATA_CODE
                  INNER JOIN airports a2
                  ON  flights.DESTINATION_AIRPORT = a2.IATA_CODE
                  WHERE DISTANCE = (SELECT MAX(DISTANCE) FROM flights)'
)
```

Part e (11 points)

Many of these queries are generating outputs that are fairly small. That is, they are easily able to be stored as variables in **R**. A couple hundred thousand rows is fine to work with since that only takes a couple MB of RAM.

1. Using the `dbGetQuery()` function again, obtain the arrival delay times and the name of the airline (by joining with the airlines table) for all the data entries that have a delay time more than 60 minutes. Save this output as a tibble in **R**.
2. Once you have this, make side-by-side boxplots in `ggplot()` for the arrival delay time split by the airline. This plot won't look too great unless the y-axis is scaled. Go ahead and use the log10 *scale* like we did back in Lab 3. Then look up how to angle the axis labels so they do not overlap and incorporate that in your plot.

```
library(viridis)
library(ggplot2)

ggplot(arrivalDelay, aes(x=airline_name, y=ARRIVAL_DELAY, fill=airline_name))+
  geom_boxplot()+
  scale_y_log10()+ #scaling variables
  labs(x = "Airlines",
       y = "log_10(Arrival Delay)")+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))+
```

```
  scale_fill_viridis(discrete = TRUE)+
  theme(legend.position = "none")
```

Part f (10 points)

1. Now, let's NOT use the `dbGetQuery()` function. Instead, insert an SQL code chunk for this part of the problem that selects the **average** arrival delay (for some reason, there is not a median function built in to SQL) and the airline's full name for each airline. So, this output should only have 14 rows.

```
SELECT AVG(ARRIVAL_DELAY) AS avg_delay, a.AIRLINE AS airline_name
FROM flights
INNER JOIN airlines a
ON flights.AIRLINE = a.IATA_CODE
GROUP BY a.AIRLINE;
```

2. Then pass this output back to **R** and create a barplot with the airlines on the x-axis and the average arrival delay on the y-axis. Again, angle the x-axis labels so they do not overlap.

```
avgDelayAirlines <- tibble(avgDelayAirlines)
head(avgDelayAirlines, 3)

ggplot(avgDelayAirlines, aes(x=airline_name, y=avg_delay ) )+
  geom_bar(stat = 'identity', color = 'gray25', bins = 10,
                fill = c("orange","sienna1","sienna2",
                         "sienna3", "sienna","sienna4",
                         "orange3", "orange2", "orange1",
                         "orange","sienna1","sienna2",
                         "sienna3", "sienna")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Part g (12 points)

1. Now let's use the `dbplyr` add-on to the `dplyr` package to work with an SQL database directly. You may have to install the `dbplyr` package. Use the `tbl()` function to create an object you can work with directly.

```
library(dbplyr)

flights <- tbl(air_con, 'flights')
flights

airlines <- tbl(air_con, 'airlines') |>
  rename(airline_name = AIRLINE)
airlines
```

2. With that object, using **R** commands from the tidyverse, find the average and standard deviation of the elapsed time of the flights by airline and arrange it by the SD (with the largest at the top). Make sure to use the `inner_join()` function so that the name of the airline, not just its code, is given. Note that to save this object in **R**'s memory, you will need to do something like pipe it into the `collect()` function.

```
flights <- inner_join(flights, airlines, by = c("AIRLINE" = "IATA_CODE"))

airline_elapsedTime <- flights |>
  group_by(airline_name) |>
  summarize(
    avg_time = mean(ELAPSED_TIME),
    std_dev = sd(ELAPSED_TIME)
  ) |>
  arrange(desc(std_dev)) |>
  collect()
```

3. Finally, make a ggplot bar graph like the one you made in Part f, but use the standard deviation as the heights on the y-axis. Why do you suppose the airline with the largest standard deviation in elapsed time has such variable flight times?

```
ggplot(airline_elapsedTime, aes(x=airline_name, y=std_dev ) )+
  geom_bar(stat = 'identity', color = 'gray25', bins = 10,
                fill = c("orange","sienna1","sienna2",
                         "sienna3", "sienna","sienna4",
                         "orange3", "orange2", "orange1",
                         "orange","sienna1","sienna2",
                         "sienna3", "sienna")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

RESPONSE: Hawaiian Airlines potentially transits over the Pacific with long carrying distances subject to both ocean current and continental airflows.

Finally, when you are done with this assignment, it is good practice to close the connection to the flights database.

```
DBI::dbDisconnect(air_con)
```