DataSci 306, Homework 6

Max Han, maxhan

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(lubridate)
library(dbplyr)
```

```
##
## Attaching package: 'dbplyr'
##
## The following objects are masked from 'package:dplyr':
##
##     ident, sql
```

```
library(nycflights13)
library(DBI)
library(RSQLite)
library(readxl)
```

The following code loads nycflights13 into a SQLite database:

```
con <- dbConnect(SQLite(), ":memory:")
dbWriteTable(con, "flights", flights, overwrite = T)
dbWriteTable(con, "airports", airports, overwrite = T)
dbWriteTable(con, "airlines", airlines, overwrite=T)
dbWriteTable(con, "planes", planes, overwrite = T)
dbWriteTable(con, "weather", weather, overwrite = T)

q <- function(...) dbGetQuery(con, ...)
```

Problem 1 (5 pts)

Problem 1 expects you to use SQL and for Problem 2 you can use dplyr

For each question below, write an appropriate SQL query that produces the answer. For example, if the question was "How many flights departed each month", an appropriate answer would be:

```
q("SELECT * FROM flights  LIMIT 5")
```

```
##   year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
## 1 2013     1   1      517            515         2      830            819
## 2 2013     1   1      533            529         4      850            830
## 3 2013     1   1      542            540         2      923            850
## 4 2013     1   1      544            545        -1     1004           1022
## 5 2013     1   1      554            600        -6      812            837
##   arr_delay carrier flight tailnum origin dest air_time distance hour minute
## 1        11      UA   1545  N14228    EWR  IAH      227     1400    5     15
## 2        20      UA   1714  N24211    LGA  IAH      227     1416    5     29
## 3        33      AA   1141  N619AA    JFK  MIA      160     1089    5     40
## 4       -18      B6    725  N804JB    JFK  BQN      183     1576    5     45
## 5       -25      DL    461  N668DN    LGA  ATL      116      762    6      0
##    time_hour
## 1 1357034400
## 2 1357034400
## 3 1357034400
## 4 1357034400
## 5 1357038000
```

```
q("SELECT month, count(*) as total FROM flights GROUP BY month")
```

```
##    month total
## 1      1 27004
## 2      2 24951
## 3      3 28834
## 4      4 28330
## 5      5 28796
## 6      6 28243
## 7      7 29425
## 8      8 29327
## 9      9 27574
## 10    10 28889
## 11    11 27268
## 12    12 28135
```

Your answers should only use SQL – no dplyr allowed! (However, it is fine to use dplyr to check your answers.)

a) Display all United (carrier code UA) routes where the difference between maximum air_time and the minimum air_time (i.e., the range) for the same route is greater than 120 minutes. A route is nothing but a origin, dest pair. Hint: In SQL also you have `max`, `min` functions

Your table should have 3 columns: origin , dest and range. (1 pt)

```
q("SELECT origin, dest, range FROM
    (
      SELECT origin, dest, max(air_time) - min(air_time) as range FROM flights
      WHERE carrier = 'UA' AND air_time IS NOT NULL
      GROUP BY origin, dest
    )
  WHERE range > 120

")
```

```
##   origin dest range
## 1    EWR  AUS   127
```

```
## 2    EWR  DEN   124
## 3    EWR  HNL   133
## 4    EWR  IAH   123
## 5    EWR  LAS   143
## 6    EWR  LAX   124
## 7    EWR  SFO   125
## 8    EWR  SNA   131
## 9    LGA  DEN   145
```

b) Display the proportions of non-cancelled (i.e., dep_time is not NA) flights in July, for each origin, rounded to 2 decimal places. Your table should have 2 columns: origin, prop_non_cancel. Also, the rows should be arranged in descending order of prop_non_cancel. (1 pt)

```
q(
"SELECT origin, ROUND((1.00 * non_cancel) / total, 2) AS prop_non_cancel FROM
  (
    SELECT origin, COUNT(dep_time) AS non_cancel, COUNT(*) AS total FROM flights WHERE month = 7
    GROUP BY origin
  )
ORDER BY prop_non_cancel DESC
")
```

```
##   origin prop_non_cancel
## 1    JFK            0.98
## 2    EWR            0.97
## 3    LGA            0.95
```

(c) Identify planes (Recall that each plane is uniquely identified by its `tailnum`.) that flew a given route more than 4 times on any day, and list those aircraft in descending order of flight frequency. Your output should have tailnum (no NA values), origin, dest, count, month and day (1 point)

```
q("SELECT tailnum, origin, dest, count, month, day FROM
    (
      SELECT tailnum, origin, dest, COUNT() AS count, month, day FROM flights WHERE tailnum IS NOT NULL
      GROUP BY tailnum, year, month, day, origin, dest
    )
   WHERE count > 4 ORDER BY count DESC
")
```

```
##   tailnum origin dest count month day
## 1  N714US    LGA  DCA     5     6  19
## 2  N732US    LGA  DCA     5     2  15
```

```
q("
  SELECT tailnum, origin, dest, COUNT() AS count, month, day FROM flights WHERE tailnum IS NOT NULL
  GROUP BY tailnum, year, month, day, origin, dest HAVING count > 4 ORDER BY count DESC
")
```

```
##   tailnum origin dest count month day
## 1  N714US    LGA  DCA     5     6  19
## 2  N732US    LGA  DCA     5     2  15
```

(d) Are some planes used by multiple carriers? Display such `tailnum` and the total number of carriers using them (1 pt)

```
q("
  SELECT tailnum, count FROM
    (
```

```
      SELECT tailnum, COUNT(DISTINCT carrier) AS count FROM flights WHERE tailnum IS NOT NULL
      GROUP BY tailnum
    )
   WHERE count > 1
")
```

```
##    tailnum count
## 1   N146PQ     2
## 2   N153PQ     2
## 3   N176PQ     2
## 4   N181PQ     2
## 5   N197PQ     2
## 6   N200PQ     2
## 7   N228PQ     2
## 8   N232PQ     2
## 9   N933AT     2
## 10  N935AT     2
## 11  N977AT     2
## 12  N978AT     2
## 13  N979AT     2
## 14  N981AT     2
## 15  N989AT     2
## 16  N990AT     2
## 17  N994AT     2
```

```
q("
    SELECT tailnum,
    COUNT(DISTINCT carrier) AS num_carriers
    FROM flights WHERE tailnum IS NOT NULL
    GROUP BY tailnum
    HAVING num_carriers > 1;
")
```

```
##    tailnum num_carriers
## 1   N146PQ            2
## 2   N153PQ            2
## 3   N176PQ            2
## 4   N181PQ            2
## 5   N197PQ            2
## 6   N200PQ            2
## 7   N228PQ            2
## 8   N232PQ            2
## 9   N933AT            2
## 10  N935AT            2
## 11  N977AT            2
## 12  N978AT            2
## 13  N979AT            2
## 14  N981AT            2
## 15  N989AT            2
## 16  N990AT            2
## 17  N994AT            2
```

(e) Joins

Show the five most frequent destinations and their counts. Use the full airport `name` from the airports table. The output should include columns 'name' and 'count'. (1 pt)

```
q("
  SELECT airports.name, COUNT() as count FROM flights
    LEFT JOIN airports ON flights.dest = airports.faa
    GROUP BY dest ORDER BY count DESC LIMIT 5
")
```

```
##                                name count
## 1                 Chicago Ohare Intl 17283
## 2    Hartsfield Jackson Atlanta Intl 17215
## 3                   Los Angeles Intl 16174
## 4 General Edward Lawrence Logan Intl 15508
## 5                       Orlando Intl 14082
```

Problem 2 (5 pts)

From this problem onward you can use dplyr functions instead of SQL. Refer the lab (if required) for reading excel files.

Formula One Race Data.

Formula One (more commonly known as Formula 1 or F1) is the highest class of international racing for open-wheel single-seater formula racing cars sanctioned by the Fédération Internationale de l'Automobile (FIA). The FIA Formula One World Championship has been one of the premier forms of racing around the world since its inaugural season in 1950. The dataset F1_Race_Data.xlsx in the `data` folder contains data from 1950 all the way through the 2017 season. We will use three tables for this problem: • drivers: Containing personal information of F1 driver, with primary key driverId • races: Containing the location and time for each grand prix, with primary key raceId. • results: Containing the results in each race, with primary key resultId, foreign keys are driverId (to drivers) and raceId (to races)

(a) Import the dataset into three Tibbles and name them drivers, races, and results, respectively. Load the first 10 columns only for the results table (Hint: use the range keyword argument in the read_excel function to define the columns needed). Print the column names of three tables. (1 pt)

```
# Load necessary libraries
library(dplyr)
library(readxl)

file_path <- "data/F1_Race_Data.xlsx"

drivers <- read_excel(file_path, sheet = "drivers")
races <- read_excel(file_path, sheet = "races")
results <- read_excel(file_path, sheet = "results", range = cell_cols(1:10))


cat("Drivers table columns:\n")
```

```
## Drivers table columns:
```

```
print(colnames(drivers))
```

```
## [1] "driverId"    "driverRef"   "number"      "code"        "forename"
## [6] "surname"     "dob"         "nationality" "url"
```

```
cat("\nRaces table columns:\n")
```

```
##
## Races table columns:
```

```
print(colnames(races))
```

```
## [1] "raceId"    "year"      "round"     "circuitId" "name"      "date"
## [7] "time"      "url"
```

```
cat("\nResults table columns (first 10 columns):\n")
```

```
##
## Results table columns (first 10 columns):
```

```
print(colnames(results))
```

```
##  [1] "resultId"      "raceId"        "driverId"      "constructorId"
##  [5] "number"        "grid"          "position"      "positionText"
##  [9] "positionOrder" "points"
```

(b) Find drivers who won first place (position = 1) in 2016. The output table should include: race_name, race_date, forename, surname, start_position and points. (2 pt) Hint: You can use **rename** function to rename the column names to the required output

```
races_2016 <- races |> filter(year == 2016)

first_place_2016 <- results |>
  inner_join(races_2016, by = "raceId") |>
  inner_join(drivers, by = "driverId") |>
  filter(position == 1) |>
  select(
    race_name = name,
    race_date = date,
    forename,
    surname,
    start_position = grid,
    points
  )

print(first_place_2016)
```

```
## # A tibble: 21 x 6
##    race_name          race_date           forename surname start_position points
##    <chr>              <dttm>              <chr>    <chr>            <dbl>  <dbl>
##  1 Australian Grand ~ 2016-03-20 00:00:00 Nico     Rosberg              2     25
##  2 Bahrain Grand Prix 2016-04-03 00:00:00 Nico     Rosberg              2     25
##  3 Chinese Grand Prix 2016-04-17 00:00:00 Nico     Rosberg              1     25
##  4 Russian Grand Prix 2016-05-01 00:00:00 Nico     Rosberg              1     25
##  5 Spanish Grand Prix 2016-05-15 00:00:00 Max      Versta~              4     25
##  6 Monaco Grand Prix  2016-05-29 00:00:00 Lewis    Hamilt~              3     25
##  7 Canadian Grand Pr~ 2016-06-12 00:00:00 Lewis    Hamilt~              1     25
##  8 European Grand Pr~ 2016-06-19 00:00:00 Nico     Rosberg              1     25
##  9 Austrian Grand Pr~ 2016-07-03 00:00:00 Lewis    Hamilt~              1     25
## 10 British Grand Prix 2016-07-10 00:00:00 Lewis    Hamilt~              1     25
## # i 11 more rows
```

(c) Challenge - Generate a suitable plot of the cumulative points vs round for the top 5 drivers (drivers with the most cumsum points across all rounds) for the same year 2016 (2 pt) Hint: You may want to use cumsum() function.

```r
# Filter the races table for the year 2016
races_2016 <- races |>
  filter(year == 2016) |>
  select(raceId, round)
races_2016
```

```
## # A tibble: 21 x 2
##    raceId round
##     <dbl> <dbl>
##  1    948     1
##  2    949     2
##  3    950     3
##  4    951     4
##  5    952     5
##  6    953     6
##  7    954     7
##  8    955     8
##  9    956     9
## 10    957    10
## # i 11 more rows
```

```r
# Join results with races to get rounds, and filter by year 2016
results_2016 <- results |>
  inner_join(races_2016, by = "raceId") |>
  select(driverId, points, round)
results_2016
```

```
## # A tibble: 462 x 3
##    driverId points round
##       <dbl>  <dbl> <dbl>
##  1        3     25     1
##  2        1     18     1
##  3       20     15     1
##  4      817     12     1
##  5       13     10     1
##  6      154      8     1
##  7      807      6     1
##  8      822      4     1
##  9      832      2     1
## 10      830      1     1
## # i 452 more rows
```

```r
# Calculate cumulative points for each driver across rounds in 2016
cumulative_points <- results_2016 |>
  group_by(driverId) |>
  arrange(round) |>
  mutate(cumsum_points = cumsum(points)) |>
  ungroup()
cumulative_points
```

```
## # A tibble: 462 x 4
##    driverId points round cumsum_points
##       <dbl>  <dbl> <dbl>         <dbl>
##  1        3     25     1            25
##  2        1     18     1            18
```

```
##  3       20     15     1            15
##  4      817     12     1            12
##  5       13     10     1            10
##  6      154      8     1             8
##  7      807      6     1             6
##  8      822      4     1             4
##  9      832      2     1             2
## 10      830      1     1             1
## # i 452 more rows
```

```
# Identify the top 5 drivers with the highest cumulative points
top_drivers <- cumulative_points |>
  group_by(driverId) |>
  summarize(total_points = max(cumsum_points)) |>
  arrange(desc(total_points)) |>
  slice_head(n = 5) |>
  pull(driverId)
top_drivers
```

```
## [1]   3   1 817  20 830
```

```
# Filter cumulative_points to include only the top 5 drivers
top_cumulative_points <- cumulative_points |>
  filter(driverId %in% top_drivers) |>
  inner_join(drivers, by = "driverId") |>  # Add driver names
  select(driverId, forename, surname, round, cumsum_points)
top_cumulative_points
```

```
## # A tibble: 105 x 5
##    driverId forename  surname    round cumsum_points
##       <dbl> <chr>     <chr>      <dbl>         <dbl>
##  1        3 Nico      Rosberg        1            25
##  2        1 Lewis     Hamilton       1            18
##  3       20 Sebastian Vettel         1            15
##  4      817 Daniel    Ricciardo      1            12
##  5      830 Max       Verstappen     1             1
##  6        3 Nico      Rosberg        2            50
##  7        1 Lewis     Hamilton       2            33
##  8      817 Daniel    Ricciardo      2            24
##  9      830 Max       Verstappen     2             9
## 10       20 Sebastian Vettel         2            15
## # i 95 more rows
```

```
# Generate the plot
ggplot(top_cumulative_points, aes(x = round, y = cumsum_points, color = surname, group = driverId)) +
  geom_line(size = 1) +
  geom_point() +
  labs(
    title = "Cumulative Points vs Round for Top 5 Drivers in 2016",
    x = "Round",
    y = "Cumulative Points",
    color = "Driver"
  ) +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
```

```
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```