# Lab 6: R and SQL

Alex Ptacek

## Overview

This lab is divided into two parts. In the first part you will practice using joins for data wrangling and analysis on the `nycflights` dataset. Some of the problems come from Chapter 19 of your book. For the second part, you will download a dataset on the budgets of college sports programs and process it for storage in a relational database (I strongly recommend using `duckdb` which can be installed using `install.packages("duckdb")`- duckdb is highly performant, self-contained, and ideally suited both to learning SQL and performing data analysis). Then you will load this database and use `dbplyr` to perform an analysis. You will also practice using `forcats` to recode some of the variables as factors (which are supported by duckdb) and using `separate_wider_delim` to split columns of text data.

You will need to have installed and to the following libraries:

```
library(tidyverse)
library(DBI)
library(duckdb)
library(nycflights13)
```

## Problems

### Part I: Airline Flight Delays

For the first part of this lab exercise, we will be using the `nycflights` library, which contains several different built in datasets including `planes`, which has information on each plane that appears in the data; `flights`, which has information on individual flights; `airports`, which has information on individual airports; and `weather`, which has information on the weather at the origin airports. In order to do this set of lab exercises, you will need to use different types of joins to combine variables in each data frame.

**Problem 1**

- Use the `flights` and `planes` tables to compute the mean departure delay of each aircraft that has more than 30 recorded flights in the dataset. Hint: Make note of the fact that the variable `year` appears in both `flights` and `planes` but means different things in each before performing any joins.

```
flights |>
  left_join(planes, by = join_by(tailnum)) |>
  group_by(model) |>
  summarise(n = n(), mean_dep_delay = mean(dep_delay)) |>
  filter(n >30)
```

```
# A tibble: 83 x 3
   model          n mean_dep_delay
   <chr>      <int>          <dbl>
 1 150           54          NA
 2 172M          45          19.8
 3 172N         165          NA
 4 206B          42          NA
 5 210-5(205)    39          NA
 6 421C          36          NA
 7 550           38          NA
 8 717-200     3150          NA
 9 737-3H4      526          NA
10 737-4B7      111           2.68
# i 73 more rows
```

- Use `anti-join` to identify flights where `tailnum` does not have a match in `plane`. Determine the carriers for which this problem is the most common.

```
flights |>
  anti_join(planes, by = join_by(tailnum)) |>
  count(carrier) |>
  arrange(desc(n)) |>
  left_join(airlines) |>
  relocate(name, .before = n)
```

```
Joining with `by = join_by(carrier)`
```

```
# A tibble: 10 x 3
   carrier name                        n
   <chr>   <chr>                   <int>
```

```
 1 MQ      Envoy Air                   25397
 2 AA      American Airlines Inc.      22558
 3 UA      United Air Lines Inc.        1693
 4 9E      Endeavor Air Inc.            1044
 5 B6      JetBlue Airways               830
 6 US      US Airways Inc.               699
 7 FL      AirTran Airways Corporation   187
 8 DL      Delta Air Lines Inc.          110
 9 F9      Frontier Airlines Inc.         50
10 WN      Southwest Airlines Co.         38
```

- Find the airplane model which made the most flights in the dataset, and filter the dataset to contain only flights flown by airplanes of that model, adding a variable which corresponds to the year each those airplanes were built. Then compute the average departure delay for each year of origin and plot the data. Is there any evidence that older planes have more greater departure delays?

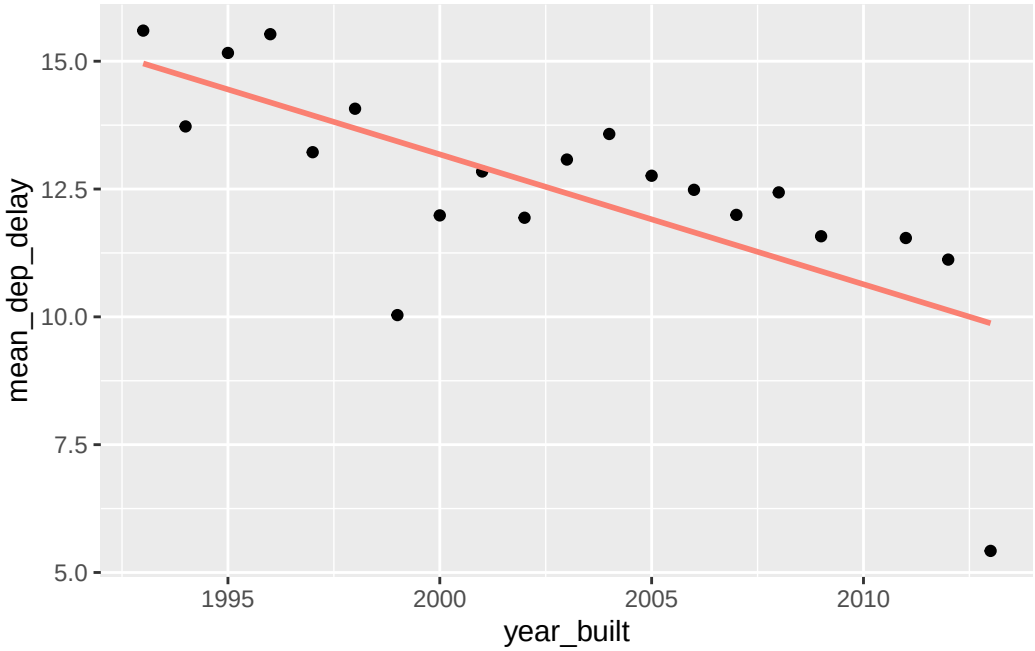
*Answer: Older manufactures of the airplane model with the most flights seem to have greater departure delays.*

```
#Find model with most flights
most_flights_model <- flights |>
  left_join(planes, by = join_by(tailnum)) |>
  count(model) |>
  slice_max(order_by = n, n = 2) |>
  filter(model != is_null(model)) #Filtering out the null value as this likely represents mul

#Find average dep_delay by year_built for this model
flights |>
  left_join(planes, by = join_by(tailnum)) |>
  rename(year_built = year.y) |>
  filter(model %in% most_flights_model$model) |>
  group_by(year_built) |>
  summarise(mean_dep_delay = mean(dep_delay, na.rm = TRUE)) |>

#Plot this table
  ggplot(aes(x = year_built, y = mean_dep_delay)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "salmon")
```

```
`geom_smooth()` using formula = 'y ~ x'
```




**Problem 2**

- Compute the average delay by destination, then join on the airports data frame so you can show the spatial distribution of delays. Here's an easy way to draw a map of the United States:

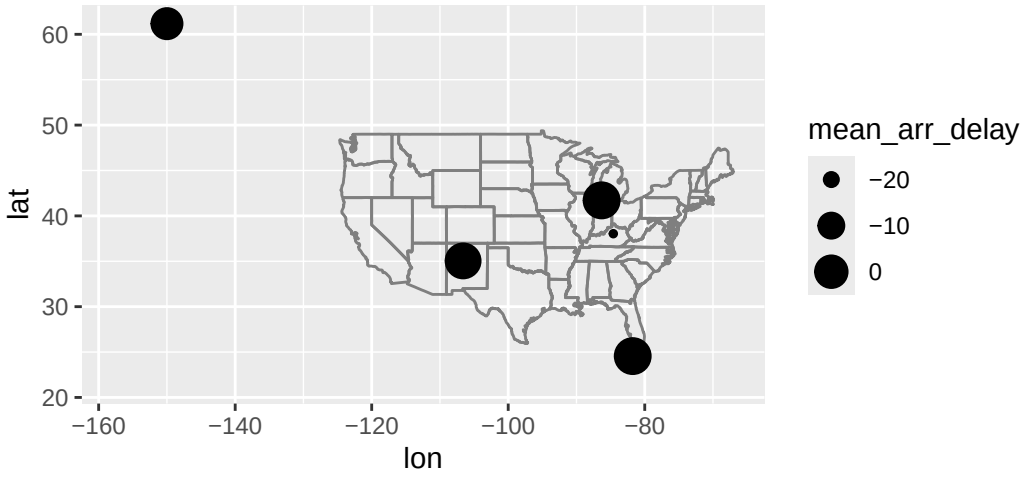
```
airports |>
  semi_join(flights, join_by(faa == dest)) |>
  ggplot(aes(x = lon, y = lat)) +
    borders("state") +
    geom_point() +
    coord_quickmap()
```

You might want to use the size or color of the points to display the average delay for each airport.

```
flights |>
  group_by(dest) |>
  summarise(mean_arr_delay = mean(arr_delay)) |>
  left_join(airports, join_by(dest == faa)) |>
  ggplot(aes(x = lon, y = lat, size = mean_arr_delay)) +
  borders("state") +
```

```
  geom_point() +
  coord_quickmap()
```

```
Warning: Removed 100 rows containing missing values or values outside the scale range
(`geom_point()`).
```




### Part II: Creating and Accessing a Database

In this exercise we will begin with a flat file which contains data on college sports programs throughout the country. The source of the data is a government run database called Equity in Athletics Data Analysis, though we are working with just a small subset here. You can download this file by clicking here: sports_program_costs.csv. I have also included a data dictionary which gives a quick description of the dataset, which can be downloaded from here: sports_program_data_dictionary.qmd. This file contains information on two types of entities: sports teams and universities, however the information on both entities is combined into a single table, creating substantial redundancies. This exercise has several goals:

1. Load this data into R, split the dataframe into two dataframes, one corresponding to colleges and another corresponding to sports teams, related to each other by common keys. Many databases are stored according to normalization rules, which are designed to limit redundancy and to make it easier to both work with the data and make changes to it. By splitting the data frame we will partially normalize it (but won't go too far).
2. Create a relational database using `duckdb` which contains these two tables.

3. Read this database into R and a/an SQL query/queries to perform an analysis.

**Problem 3:**

- `sports_program_data.csv` contains variables which either describe properties of a sports team or a college. Split `sports_programs_data` into two data frames, one called `colleges` and another called `teams`. How can you tell which variables describe colleges and which describe teams? Use the data dictionary and observations of how the values vary as you move from college to college to help make the decision easier. Make sure there are primary keys for both the colleges and teams data frames (verify with `count`)- what are the primary keys in each case and are they simple keys (one variable) or compound keys (require multiple variables). One of these data-sets should contain a foreign key- which one has it and what variables comprise it?

```
sports_program_costs <- read_csv("https://raw.githubusercontent.com/georgehagstrom/DATA607/r
```

```
Rows: 132327 Columns: 28
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (8): institution_name, city_txt, state_cd, zip_text, classification_nam...
dbl (20): year, unitid, classification_code, ef_male_count, ef_female_count,...

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.
```

*Answer: I could tell which data belonged to* ***colleges*** *and which belonged to* ***teams*** *based on the data dictionary. The* ***colleges*** *data has variables that only identify aspects of the school. This includes the 3 sports division columns, such as classification code, because each school can only be apart of 1 sports division. It does not include the granularity of individual sport-level data, such as sport, gender makeup, and revenue. The* ***teams*** *data includes individual sport-level data (i.e. the remaining columns), plus the school code (****unitid****) to identify the team, and the* ***year*** *because observations are yearly. The primary key for* ***colleges*** *is* ***year*** *and* ***unitid****, and the primary key for* ***teams*** *is* ***year****,* ***unitid****, and* ***sportscode****. These are both compound keys. The foreign key is in* ***teams*** *and it is* ***year*** *and* ***unitid****.*

```
#Create colleges table
colleges <- sports_program_costs |>
  select(year:sector_name) |>
  group_by(across(year:classification_other), sector_cd, sector_name) |>
  summarise(ef_male_count = sum(ef_male_count),
            ef_female_count = sum(ef_female_count),
            ef_total_count = sum(ef_total_count))
```

```
`summarise()` has grouped output by 'year', 'unitid', 'institution_name',
'city_txt', 'state_cd', 'zip_text', 'classification_code',
'classification_name', 'classification_other', 'sector_cd'. You can override
using the `.groups` argument.
```

```
#Check colleges primary key
colleges |>
  count(year, unitid) |>
  filter(n > 1)
```

```
# A tibble: 0 x 11
# Groups:   year, unitid, institution_name, city_txt, state_cd, zip_text,
#   classification_code, classification_name, classification_other, sector_cd
#   [0]
# i 11 variables: year <dbl>, unitid <dbl>, institution_name <chr>,
#   city_txt <chr>, state_cd <chr>, zip_text <chr>, classification_code <dbl>,
#   classification_name <chr>, classification_other <chr>, sector_cd <dbl>,
#   n <int>
```

```
#Create teams table
teams <- sports_program_costs |>
  select(year, unitid, sportscode:sports)

#Check teams primary key
teams |> count(year, unitid, sportscode) |>
  filter(n >1)
```

```
# A tibble: 0 x 4
# i 4 variables: year <dbl>, unitid <dbl>, sportscode <dbl>, n <int>
```

- The variable `sector_name` contains information about whether a college is public, private, non-profit, for-profit, a 2-year college, or a 4-year + college. Split this variable (using `separate_wider_delim`) into two variables, one of which describes whether the college is a Public, Private nonprofit, or private for-profit, and another which describes how many years the college programs run.

```
#Examine sector_name variable
view_sect_names <- colleges |>
  group_by(sector_name) |>
  count(sector_name)
view_sect_names |> head(n = 5)
```

```
# A tibble: 5 x 2
# Groups:   sector_name [5]
  sector_name                              n
  <chr>                                <int>
1 Private for-profit, 2-year               9
2 Private for-profit, 4-year or above     91
3 Private nonprofit, 2-year               65
4 Private nonprofit, 4-year or above    4424
5 Public, 2-year                        2799
```

```
#Separate sector_name by comma as delim
colleges <- colleges |>
  separate_wider_delim(cols = sector_name, delim = ", ",
                       names = c("sector_type", "degree_type"),
                       cols_remove = FALSE)

#Confirm changes worked
view_sect_names2 <- colleges |>
  group_by(sector_type, degree_type) |>
  count(sector_type, degree_type)
view_sect_names2 |> head(n = 5)
```

```
# A tibble: 5 x 3
# Groups:   sector_type, degree_type [5]
  sector_type        degree_type         n
  <chr>              <chr>           <int>
1 Private for-profit 2-year              9
2 Private for-profit 4-year or above    91
3 Private nonprofit  2-year             65
4 Private nonprofit  4-year or above  4424
5 Public             2-year           2799
```

- Several variables are candidates to be recoded as factors, for example `state_cd`, `zip_text`, `classification_name`, `sports`, and the `sector` variables you just created for the previous part. Recode these variables as categorical variables. For the `classification` variable, use the `classification_code` to order the factors according to the numeric code.

```
#Create state_order as levels for state_cd
state_order <- append(state.abb, "DC", after = 7)
state_order <- append(state_order, "PR", after = 38)
state_order <- append(state_order, "VI", after = 47)
```

```r
#Mutate state_cd into factor
colleges <- colleges |>
  mutate(state_cd = fct(state_cd, levels = state_order))


#Create sorted_zips as levels for zip_text
sorted_zips <- fct_reorder(colleges$zip_text, as.integer(colleges$zip_text), .na_rm = FALSE)

#Mutate zip_text into factor
colleges <- colleges |>
  mutate(zip_text = fct(zip_text, levels = levels(sorted_zips)))


#Create sorted_class as levels for classification_name
class_order <- fct_reorder(colleges$classification_name, colleges$classification_code)

#Mutate classification_name into factor
colleges <- colleges |>
  mutate(classification_name = fct(classification_name, levels = levels(class_order)))


#Create sports_order as levels for sports
sports_order <- fct_reorder(teams$sports, teams$sportscode)

#Mutate sports into factor
teams <- teams |>
  mutate(sports = fct(sports, levels = levels(sports_order)))


#Create sector_order as levels for sector_name
sector_order <- fct_reorder(sports_program_costs$sector_name, sports_program_costs$sector_cd

#Mutate sector_name into factor
colleges <- colleges |>
  mutate(sector_name = fct(sector_name, levels = levels(sector_order)))


#Observe changes
glimpse(colleges)
```

```
Rows: 10,364
```

```
Columns: 16
Groups: year, unitid, institution_name, city_txt, state_cd, zip_text, classification_code, cl
$ year                 <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2~
$ unitid               <dbl> 100654, 100663, 100706, 100724, 100751, 100760, 1~
$ institution_name     <chr> "Alabama A & M University", "University of Alabam~
$ city_txt             <chr> "Normal", "Birmingham", "Huntsville", "Montgomery~
$ state_cd             <fct> AL, AL, AL, AL, AL, AL, AL, AL, AL, AL, AL, AL, A~
$ zip_text             <fct> 35762, 352940110, 35899, 361040271, 354870166, 35~
$ classification_code  <dbl> 2, 1, 5, 2, 1, 12, 5, 1, 6, 12, 20, 12, 12, 9, 12~
$ classification_name  <fct> NCAA Division I-FCS, NCAA Division I-FBS, NCAA Di~
$ classification_other <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ sector_cd            <dbl> 1, 1, 1, 1, 1, 4, 1, 1, 2, 4, 2, 4, 4, 2, 4, 4, 4~
$ sector_type          <chr> "Public", "Public", "Public", "Public", "Public",~
$ degree_type          <chr> "4-year or above", "4-year or above", "4-year or ~
$ sector_name          <fct> "Public, 4-year or above", "Public, 4-year or abo~
$ ef_male_count        <dbl> 19230, 33970, 23796, 16790, 155148, 1744, 6480, 1~
$ ef_female_count      <dbl> 23000, 48620, 18639, 26980, 186216, 2124, 11988, ~
$ ef_total_count       <dbl> 42230, 82590, 42435, 43770, 341364, 3868, 18468, ~
```

```
glimpse(teams)
```

```
Rows: 132,327
Columns: 16
$ year               <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 201~
$ unitid             <dbl> 100654, 100654, 100654, 100654, 100654, 100654, 100~
$ sportscode         <dbl> 1, 2, 3, 7, 8, 15, 16, 22, 26, 33, 1, 2, 3, 8, 12, ~
$ partic_men         <dbl> 31, 19, 61, 99, 9, NA, NA, 7, NA, NA, 32, 13, NA, 1~
$ partic_women       <dbl> NA, 16, 46, NA, NA, 21, 25, 10, 16, 9, NA, 20, 68, ~
$ partic_coed_men    <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ partic_coed_women  <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ sum_partic_men     <dbl> 31, 19, 61, 99, 9, 0, 0, 7, 0, 0, 32, 13, 0, 10, 2,~
$ sum_partic_women   <dbl> 0, 16, 46, 0, 0, 21, 25, 10, 16, 9, 0, 20, 68, 7, 1~
$ rev_men            <dbl> 345592, 1211095, 183333, 2808949, 78270, NA, NA, 78~
$ rev_women          <dbl> NA, 748833, 315574, NA, NA, 410717, 298164, 131145,~
$ total_rev_menwomen <dbl> 345592, 1959928, 498907, 2808949, 78270, 410717, 29~
$ exp_men            <dbl> 397818, 817868, 246949, 3059353, 83913, NA, NA, 996~
$ exp_women          <dbl> NA, 742460, 251184, NA, NA, 432648, 340259, 113886,~
$ total_exp_menwomen <dbl> 397818, 1560328, 498133, 3059353, 83913, 432648, 34~
$ sports             <fct> "Baseball", "Basketball", "All Track Combined", "Fo~
```

### Problem 4

- Using `DBI`, `duckdb`, and `dbplyr`, create a relational database with two tables, writing the `sports` data frame you created in problem 3 to one and the `colleges` data frame (also from problem 3) to the other. Write this database to disk. How does the size of the database file compare to the original csv?

*Answer: The original file was 24.4MB. The new relational database is 4.5MB.*

```
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = "duckdb")

dbWriteTable(con, "colleges", colleges)
dbWriteTable(con, "teams", teams)

dbListTables(con)
```

```
[1] "colleges" "teams"
```

- Use `dbplyr` to write a query to this database that calculates the top 10 colleges ranked by the average profit (defined as revenue - expenses) of their american football team over the years of data. Print the SQL query that results from your R pipeline using `show_query()` and then use `collect()` to show the results of this query.

```
#Load dbplyr tables
colleges_tbl <- tbl(con, "colleges")
teams_tbl <- tbl(con, "teams")

teams_tbl <-  teams_tbl |>
  mutate(profit = total_rev_menwomen - total_exp_menwomen)

colleges_tbl |>
  left_join(teams_tbl, by = join_by(year, unitid)) |>
  filter(sportscode == 7) |>
  group_by(institution_name) |>
  summarise(mean_profit = mean(profit)) |>
  arrange(desc(mean_profit)) |>
  head(10) |>
  show_query() |>
  collect()
```

```
<SQL>
SELECT institution_name, AVG(profit) AS mean_profit
FROM (
  SELECT q01.*
```

```
  FROM (
    SELECT
      colleges.*,
      sportscode,
      partic_men,
      partic_women,
      partic_coed_men,
      partic_coed_women,
      sum_partic_men,
      sum_partic_women,
      rev_men,
      rev_women,
      total_rev_menwomen,
      exp_men,
      exp_women,
      total_exp_menwomen,
      sports,
      profit
    FROM colleges
    LEFT JOIN (
      SELECT teams.*, total_rev_menwomen - total_exp_menwomen AS profit
      FROM teams
    ) RHS
      ON (colleges."year" = RHS."year" AND colleges.unitid = RHS.unitid)
  ) q01
  WHERE (sportscode = 7.0)
) q01
GROUP BY institution_name
ORDER BY mean_profit DESC
LIMIT 10
```

```
# A tibble: 10 x 2
   institution_name                                              mean_profit
   <chr>                                                               <dbl>
 1 The University of Texas at Austin                           102949317.
 2 University of Michigan-Ann Arbor                             71516420
 3 University of Georgia                                        69923628.
 4 The University of Tennessee-Knoxville                        62983384.
 5 University of Notre Dame                                     59923466.
 6 University of Oklahoma-Norman Campus                         58268012.
 7 Ohio State University-Main Campus                            56567973
 8 Louisiana State University and Agricultural & Mechanical College   55336698.
```

```
 9 University of Wisconsin-Madison                                  49896698.
10 Auburn University                                                49689829
```