Lab 1 - Math 58B: Introduction to Data

Write Your Name Here

due Wednesday Feb 3, 2021

Some define statistics as the field that focuses on turning information into knowledge. The first step in that process is to summarize and describe the raw information – the data. In this lab we explore flights, specifically a random sample of domestic flights that departed from the three major New York City airport in 2013. We will generate simple graphical and numerical summaries of data on these flights and explore delay times. As this is a large data set, along the way you'll also learn the indispensable skills of data processing and subsetting.

The goals for today include be able to use the following commands:

- the pipe command `%>%`
- data wrangling verbs: `filter`, `summarize`, `group_by`, `mutate`, `arrange`
- `ifelse`

Advice for turning in the assignment

- knit early and often. In fact, go ahead and knit your .Rmd file right now. Maybe set a timer so that you knit every 5 minutes. Do **not** wait until you are done with the assignment to knit.
- The **ASSIGNMENT** part of the lab is **ONLY** the last four questions at the very bottom. However, you will need many of the commands in the top half, otherwise the commands at the bottom won't run!

Getting started

Load packages

In this lab we will explore the data using functions that can be found in the `tidyverse` package. The data can be found in the package `nycflights`.

Let's load the packages.

```
library(tidyverse)
library(nycflights13)
```

The data

The Bureau of Transportation Statistics (BTS) is a statistical agency that is a part of the Research and Innovative Technology Administration (RITA). As its name implies, BTS collects and makes available transportation data, such as the flights data we will be working with in this lab.

We begin by loading the `flights` data frame. Run the following command by clicking on the green triangle:

```
data(flights)
```

The data set `flights` that shows up in your workspace is a *data matrix*, with each row representing an *observation* and each column representing a *variable*. R calls this data format a **data frame**, which is a term that will be used throughout the labs. For this data set, each *observation* is a single flight.

To view the names of the variables, run the command

```
names(flights)
```

```
##  [1] "year"           "month"          "day"            "dep_time"
##  [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
##  [9] "arr_delay"      "carrier"        "flight"         "tailnum"
## [13] "origin"         "dest"           "air_time"       "distance"
## [17] "hour"           "minute"         "time_hour"
```

This returns the names of the variables in this data frame. The **codebook** (description of the variables) can be accessed by pulling up the help file (run this line with the green triangle and then look at the box on the bottom right of the RStudio screen):

```
?flights
```

One of the variables refers to the carrier (i.e. airline) of the flight, which is coded according to the following system.

- `carrier`: Two letter carrier abbreviation.
  - `9E`: Endeavor Air Inc.
  - `AA`: American Airlines Inc.
  - `AS`: Alaska Airlines Inc.
  - `B6`: JetBlue Airways
  - `DL`: Delta Air Lines Inc.
  - `EV`: ExpressJet Airlines Inc.
  - `F9`: Frontier Airlines Inc.
  - `FL`: AirTran Airways Corporation
  - `HA`: Hawaiian Airlines Inc.
  - `MQ`: Envoy Air
  - `OO`: SkyWest Airlines Inc.
  - `UA`: United Air Lines Inc.
  - `US`: US Airways Inc.
  - `VX`: Virgin America
  - `WN`: Southwest Airlines Co.
  - `YV`: Mesa Airlines Inc.

A very useful function for taking a quick peek at your data frame and viewing its dimensions and data types is `glimpse`.

```
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013...
## $ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 55...
## $ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 60...
## $ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2,...
## $ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 8...
## $ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 8...
## $ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7,...
## $ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6"...
## $ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301...
## $ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N...
## $ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LG...
## $ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IA...
```

```
## $ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149...
## $ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 73...
## $ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6...
## $ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59...
## $ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-0...
```

The `flights` data frame is a massive trove of information (336,776 observations!!!). Notice also that the `glimpse` function let's you know how the variables are stored: integer, double (a fancy way to say decimal number), character string, date/time, etc. Let's think about some questions we might want to answer with these data:

- How delayed were flights that were headed to Los Angeles?
- How do departure delays vary over months?
- Which of the three major NYC airports has a better on time percentage for departing flights?

Tidy Structure of Data

For plotting, analyses, model building, etc., the data should be structured according to certain principles.

- *Tidy Data*: rows (cases/observational units) and columns (variables).
  The key is that *every* row is a case and *every* column is a variable.
  No exceptions.
- Creating tidy data is often not trivial.

Within R (really within any type of computing language, Python, SQL, Java, etc.), it is important to understand how to build data using the patterns of the language.
Some things to consider:

- `object_name <- anything` is a way of assigning `anything` to the new `object_name`.
- `object_name <- function_name(data_frame, arguments)` is a way of using a function to create a new object.
- `object_name <- data_frame %>% function_name(arguments)` uses chaining syntax as an extension of the ideas of functions.
  In chaining, the value on the left side of `%>%` becomes the *first argument* to the function on the right side.

```
object_name <- data_frame %>%
                    function_name(arguments) %>%
                    another_function_name(other_arguments)
```

is extended chaining. `%>%` is never at the front of the line, it is always connecting one idea with the continuation of that idea on the next line. * In R, all functions take arguments in round parentheses (as opposed to subsetting observations or variables from data objects which happen with square parentheses). Additionally, the spot to the left of `%>%` is always a data table. * The pipe syntax should be read as *then*, `%>%`.

Using the pipe to chain

The pipe syntax (`%>%`) takes a data frame (or data table) and sends it to the argument of a function. The mapping goes to the first available argument in the function.
For example:

`x %>% f(y)` is the same as `f(x, y)`

`y %>% f(x, ., z)` is the same as `f(x,y,z)`

Pipes are used commonly with functions in the `dplyr` package and they allow us to sequentially build data wrangling operations. We'll start with short pipes and throughout the course build up to longer pipes that perform multiple operations.

Analysis

Departure delays

Let's start by examining the distribution of departure delays of all flights using the `summary` function. The first item (on the left) is the data set. Subsequently, two functions are applied, (1) `select` to get only the `dep_delay` variables, and (2) `summary` to produce the numerical summary.

```
flights %>%
  dplyr::select(dep_delay) %>%
  summary()
```

```
##    dep_delay
##  Min.   : -43.00
##  1st Qu.:  -5.00
##  Median :  -2.00
##  Mean   :  12.64
##  3rd Qu.:  11.00
##  Max.   :1301.00
##  NA's   :8255
```

`filter` (create a smaller dataset) If we want to focus only on departure delays of flights headed to Los Angeles, we need to first `filter` the data for flights with that destination (`dest == "LAX"`). The departure delay for the LAX flights can then be summarized.

```
lax_flights <- flights %>%
  dplyr::filter(dest == "LAX")

lax_flights %>%
  dplyr::select(dep_delay) %>%
  summary()
```

```
##    dep_delay
##  Min.   :-16.000
##  1st Qu.: -4.000
##  Median : -1.000
##  Mean   :  9.401
##  3rd Qu.:  7.000
##  Max.   :800.000
##  NA's   :98
```

Let's decipher these two commands (It's common to add a break to a new line after `%>%` to help readability).

- Command 1: Take the `flights` data frame, `filter` for flights headed to LAX, and save the result as a new data frame called `lax_flights`.
  - `==` means "if it's equal to". (notice that there are TWO equals signs)
  - `LAX` is in quotation marks since it is a character string.
- Command 2: Basically the same call for summarizing the departure delay.

Notice that if we only want the summary of `dep_delay` for the LAX flights (and we don't need to keep a copy of the dataset), we can perform the above tasks by combining them into fewer steps:

```
flights %>%
  filter(dest == "LAX") %>%
  dplyr::select(dep_delay) %>%
  summary()
```

```
##    dep_delay
```

```
##  Min.   :-16.000
##  1st Qu.: -4.000
##  Median : -1.000
##  Mean   :  9.401
##  3rd Qu.:  7.000
##  Max.   :800.000
##  NA's   :98
```

Logical operators: Filtering for certain observations (e.g. flights from a particular airport) is often of interest in data frames where we might want to examine observations with certain characteristics separately from the rest of the data. To do so we use the `filter` function and a series of **logical operators**. The most commonly used logical operators for data analysis are as follows:

- `==` means "equal to"
- `!=` means "not equal to"
- `>` or `<` means "greater than" or "less than"
- `>=` or `<=` means "greater than or equal to" or "less than or equal to"

`summarize` (calculate statistics) We can also obtain numerical summaries for these flights:

```
lax_flights %>%
  summarize(mean_dd = mean(dep_delay, na.rm=TRUE),
            median_dd = median(dep_delay, na.rm=TRUE), n_dd = n())
```

```
## # A tibble: 1 x 3
##   mean_dd median_dd  n_dd
##     <dbl>     <dbl> <int>
## 1    9.40        -1 16174
```

Note that in the `summarize` function we created a list of three different numerical summaries that we were interested in. The names of these elements are user defined, like `mean_dd`, `median_dd`, `n_dd`, and you could customize these names as you like (just don't use spaces in your names). Calculating these summary statistics also require that you know the function calls. Note that `n()` reports the sample size.

Summary statistics: Some useful function calls for summary statistics for a single numerical variable are as follows:

- `mean`
- `median`
- `sd`
- `IQR`
- `min`
- `max`

Note that each of these functions take a single vector as an argument, and returns a single value.

Functions you may not be familiar with (and that we will see in more detail in coming weeks) include:

$$\text{sd} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}$$

$$\text{IQR} = 75\% - 25\%$$

We can also filter based on multiple criteria. Suppose we are interested in flights headed to San Francisco (SFO) in February:

```
sfo_feb_flights <- flights %>%
  filter(dest == "SFO", month == 2)
```

Note that we can separate the conditions using commas if we want flights that are both headed to SFO **and** in February. If we are interested in either flights headed to SFO **or** in February we can use the `|` instead of the comma.

1. Create a new data frame that includes flights headed to SFO in February, and save this data frame as `sfo_feb_flights`. How many flights meet these criteria?

```
sfo_feb_flights %>%
  summarize(n())
```

```
## # A tibble: 1 x 1
##   `n()`
##   <int>
## 1   791
```

2. Describe the distribution of the **arrival** delays of these flights using summary and/or appropriate summary statistics.

```
sfo_feb_flights %>%
  summarize(arrdelmean = mean(arr_delay, na.rm=TRUE),
            arrdelsd = sd(arr_delay, na.rm=TRUE),
            arrdelmed = median(arr_delay, na.rm=TRUE),
            arrdeliqr = IQR(arr_delay, na.rm=TRUE))
```

```
## # A tibble: 1 x 4
##   arrdelmean arrdelsd arrdelmed arrdeliqr
##        <dbl>    <dbl>     <dbl>     <dbl>
## 1      -9.14     31.4       -13        29
```

`group_by` (group before summarizing) Another useful technique is quickly calculating summary statistics for various groups in your data frame. For example, we can modify the above command using the `group_by` function to get the same summary stats for each origin airport:

```
sfo_feb_flights %>%
  group_by(origin) %>%
  summarize(median_dd = median(dep_delay, na.rm=TRUE),
            iqr_dd = IQR(dep_delay, na.rm=TRUE), n_flights = n())
```

```
## # A tibble: 2 x 4
##   origin median_dd iqr_dd n_flights
##   <chr>      <dbl>  <dbl>     <int>
## 1 EWR            0      9       194
## 2 JFK           -2      8       597
```

Here, we first grouped the data by `origin`, and then calculated the summary statistics.

3. Calculate the median and interquartile range for `arr_delay` of flights in in the `sfo_feb_flights` data frame, grouped by carrier. Which carrier has the most variable arrival delays (as measured by `IQR`)?

```
sfo_feb_flights %>%
  group_by(carrier) %>%
  summarize(arrdelmed = median(arr_delay, na.rm=TRUE),
            arrdeliqr = IQR(arr_delay, na.rm=TRUE)) %>%
  arrange(desc(arrdeliqr))
```

```
## # A tibble: 5 x 3
##   carrier arrdelmed arrdeliqr
##   <chr>       <dbl>     <dbl>
## 1 AA             -7      35
```

```
## 2 DL            -24      27.5
## 3 UA             -9      27
## 4 B6            -11      25.5
## 5 VX            -20      23
```

arrange departure delays over months

Which month would you expect to have the highest average delay departing from an NYC airport?

Let's think about how we would answer this question:

- First, calculate monthly averages for departure delays. With the new language we are learning, we need to
  - `group_by` months, then
  - `summarize` mean departure delays.
- Then, we need to `arrange` these average delays in `desc`ending order

```
flights %>%
  group_by(month) %>%
  summarize(mean_dd = mean(dep_delay, na.rm=TRUE)) %>%
  arrange(desc(mean_dd))
```

```
## # A tibble: 12 x 2
##    month mean_dd
##    <int>   <dbl>
##  1     7   21.7
##  2     6   20.8
##  3    12   16.6
##  4     4   13.9
##  5     3   13.2
##  6     5   13.0
##  7     8   12.6
##  8     2   10.8
##  9     1   10.0
## 10     9    6.72
## 11    10    6.24
## 12    11    5.44
```

On time departure rate for NYC airports

Suppose you will be flying out of NYC and want to know which of the three major NYC airports has the best on time departure rate of departing flights. Suppose also that for you a flight that is delayed for less than 5 minutes is basically "on time". You consider any flight delayed for 5 minutes of more to be "delayed".

In order to determine which airport has the best on time departure rate, we need to

- first classify each flight as "on time" or "delayed",
- then group flights by origin airport,
- then calculate on time departure rates for each origin airport,
- and finally arrange the airports in descending order for on time departure percentage.

`mutate` (create a new variable) Let's start with classifying each flight as "on time" or "delayed" by creating a new variable with the `mutate` function.

```
flights <- flights %>%
  mutate(dep_type = ifelse(dep_delay < 5, "on time", "delayed"))
```

The first argument in the `mutate` function is the name of the new variable we want to create, in this case `dep_type`. Then if `dep_delay < 5` we classify the flight as `"on time"` and `"delayed"` if not, i.e. if the flight is delayed for 5 or more minutes.

Note that we are also overwriting the `flights` data frame with the new version of this data frame that includes the new `dep_type` variable.

We can handle all the remaining steps in one code chunk:

```
flights %>%
  group_by(origin) %>%
  summarize(ot_dep_rate = mean(dep_type == "on time", na.rm=TRUE)) %>%
  arrange(desc(ot_dep_rate))
```

```
## # A tibble: 3 x 2
##   origin ot_dep_rate
##   <chr>        <dbl>
## 1 LGA          0.728
## 2 JFK          0.691
## 3 EWR          0.639
```

4. If you were selecting an airport (of the three NYC airports in the dataset) simply based on on time departure percentage, which NYC airport would you choose to fly out of? (How did you define "on time"? 0 min? 5 min? Something else?)

LGA seems to have the highest on time departure percentage, so that's the airport I would choose.

To Turn In

5. Mutate the data frame so that it includes a new variable that contains the average speed, `avg_speed` traveled by the plane for each flight (in mph). **Hint:** Average speed can be calculated as distance divided by number of hours of travel, and note that `air_time` is given in minutes.

6. Another useful `dplyr` filtering helper function is `between`. What does it do? Use it to find flights that arrived between 0 and 60 minutes late. How many such flights are there?

7. Suppose you really dislike departure delays, and you want to schedule your travel in a month that minimizes your potential departure delay leaving NYC. One option is to choose the month with the lowest mean departure delay. Another option is to choose the month with the lowest median departure delay. What are the pros and cons of these two choices? Which month do you choose?

8. Which month has the highest average arrival delay from an NYC airport? What about the highest median arrival delay? Which of these measures is more reliable for deciding which month(s) to avoid flying if you really dislike delayed flights.

HW & Lab assignments will be graded out of 5 points, which are based on a combination of accuracy and effort. Below are rough guidelines for grading.

Score & Description

5 points: All problems completed with detailed solutions provided and 75% or more of the problems are fully correct.

4 points: All problems completed with detailed solutions and 50-75% correct; OR close to all problems completed and 75%-100% correct. An assignment will earn a 4 if there is superfluous information printed out on the assignment.

3 points: Close to all problems completed with less than 75% correct

2 points: More than half but fewer than all problems completed and $> 75\%$ correct

1 point: More than half but fewer than all problems completed and $< 75\%$ correct; OR less than half of problems completed

0 points: No work submitted, OR half or less than half of the problems submitted and without any detail/work shown to explain the solutions.

General notes on homework assignments (also see syllabus for policies and suggestions):

- please be neat and organized, this will help me, the grader, and you (in the future) to follow your work.
- be sure to include your name on the assignment
- please include at least the number of the problem, or a summary of this question (this will also be helpful to you in the future to prepare for exams).
- for R problems, it is required to use R Markdown. You can write out other problems with pencil and combine pdf as appropriate.
- please do not print errors, messages, warnings, or anything else that makes your homework unwieldy. You will be graded down for superfluous printouts.
- in case of questions, or if you get stuck please don't hesitate to email me or DM on Discord! The sooner (and more often) questions get asked, the better for everyone.