Minilab 3b Worksheet

Joining Multiple Datasets

When working with real-world data, you will often find that the data is stored across *multiple* tables or files. For example, if you had a dataset containing information on a fundraising campaign that tracked donations (date, amount), you would likely store information about each donor (email, phone number) in a separate data file (and thus data table).

- Rather than duplicating information about each donor every time that person makes a donation, you can store that information a single time. This will reduce the amount of space your data takes up.
- If you need to update information about a donor, you can make that change in a *single location*.

This separation and organisation of data is a core concern in the design of *relational databases*.

1. Queries and Joins

In a previous minilab, we looked at the `flights` dataset in the "nycflights13" R package.

```
library(tidyverse)
install.packages("nycflights13")    # run this only once
library(nycflights13)
```

This R package actually contains five datasets (tibbles), as follows. For further information about these datasets see https://www.rdocumentation.org/packages/nycflights13/

- `flights`: all flights that departed from New York city in 2013
- `weather`: hourly meterological data for each airport
- `planes`: construction information about each plane

- `airports`: airport names and locations
- `airlines`: translation between two letter carrier codes and names

Remember that you can easily view a dataset (tibble) in full in a spreadsheet-like viewer (here "View" has a capital V). The airlines tibble has two columns, namely *carrier* (a two letter code, e.g., AA) and *name* (which gives a longer name for the airline, e.g., American Airlines Inc.).

```
View(airlines)
```

At some point, you will want to access information from two datasets, and thus need a way to reference values from both datasets at once, in effect to *combine* the datasets. In SQL we use JOIN to combine datasets using a common key. To illustrate some analyses involving joins we will try to answer the following challenges.

Challenge 1. Which **airline** has the highest number of delayed departures?

Firstly, we use filter to find flights (rows) with a delayed departure. We then form one group for each airline (carrier column in the tibble) and count how many rows for each group.

```
delayed_flights = flights %>%       # start with the flights
  filter(dep_delay>0) %>%           # find only the delays
  group_by(carrier) %>%             # group by airline (carrier)
  summarise(num_delay=n())          # count the observations
delayed_flights
```

Notice that we now have a (summary) tibble consisting of two columns: carrier (a two character code), num_delay. Information about each airline is stored in a separate tibble called airlines.

Notice that the summary tibble delayed_flights and the airlines tibble share a common column named carrier. We say that *carrier* is a **common key**. Therefore, to combine these two tibbles we use a join.

```
left_join(delayed_flights, airlines, by="carrier")
```

It remains to find the maximum of the num_delay column and select the particular airline (row) that gives that maximum value.

```
left_join(delayed_flights, airlines, by="carrier") %>%
  filter(num_delay==max(num_delay)) %>% # find most delayed
  select(name)                          # select the airline
```

It is possible to complete the whole sequence as one dplyr pipe.

```
flights %>%                               # start with the flights
  filter(dep_delay>0) %>%                 # find only the delays
  group_by(carrier) %>%                   # group by airline (carrier)
  summarise(num_delay=n()) %>%            # count the observations
  left_join(airlines, by="carrier") %>%   # join by common key
  filter(num_delay==max(num_delay)) %>%   # find most delayed
  select(name)                            # select the airline
```

Challenge 2. On average, to which **airport** do flights arrive most early?

In the flights tibble, a plane arrives at a destination airport (dest) and earliness is measured by negative values of arr_delay.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_delay=mean(arr_delay))
```

It is ALWAYS a good idea to check your work as you perform each step of an analysis. Don't write a long sequence of manipulations and hope that you get the right answer.

Looking at the output of the code so far, hopefully you notice a lot of NA values. These are "not available" or "missing" values, because there are a lot of entries in arr_delay that are also missing values (NA). It would be a good idea to remove these entries from the calculation of a mean.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_delay=mean(arr_delay, na.rm=TRUE))
```

We want the dest with the most negative value of arr_delay, so we need to find the row of this summary table with minimum value. It is possible that in taking the mean but removing all the NA values that there are no values to take a mean of, i.e., **all** of the values in the column arr_delay were NA for one or more dest values. Therefore it is recommended to also use na.rm=TRUE when using the min() or max() functions.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_delay=mean(arr_delay, na.rm=TRUE)) %>%
  filter(mean_delay==min(mean_delay, na.rm=TRUE))
```

The result so far is a tibble with a single row. It might have been that there was a tie for the minimum mean_delay in which case we could have more than one row in our tibble produced so far.

In this resulting tibble, the destination is in the column labelled *dest*. However, in the *airports* dataset, these equivalent airport codes are in a column labelled *faa* (since these are the Federal Aviation Administration airport codes). To apply a join between our results and the *airports* dataset, we need to specify that *dest* and *faa* are to be treated as the **common key**.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_delay=mean(arr_delay, na.rm=TRUE)) %>%
  filter(mean_delay==min(mean_delay, na.rm=TRUE)) %>%
  left_join(airports,by=c("dest"="faa"))
```

Finally we just want the full name of the airport.

```
flights %>%
  group_by(dest) %>%
  summarise(mean_delay=mean(arr_delay, na.rm=TRUE)) %>%
  filter(mean_delay==min(mean_delay, na.rm=TRUE)) %>%
  left_join(airports,by=c("dest"="faa")) %>%
  select(name)
```

For the R code above, make sure you add one line at a time (without the %>%) to make sure you can see how the analysis builds.

Notice that there is a common method emerging for this type of data wrangling:

- **filter** out the rows that meet the conditions
- create a summary by group to produce a summary tibble (**group_by** and **summarise**)
- **filter** out the row you want from the summary tibble (the "solution" to the problem)
- **join** with another dataset that includes more information about the solution, and
- **select** the information you want to provide as the final answer.

Exercise. Use the nycflights13 package datasets to answer the following.

(a) Considering only flights from JFK to SEA, what was the average, min, and max air time of those flights?

(b) Which city was flown to with the highest average speed?

Exercise. The "diamonds" dataset contains data about over 50000 diamonds including their sale *price* and other attributes known to influence their price including the 4 Cs (carat, cut, color, and clarity). See https://ggplot2.tidyverse.org/reference/diamonds.html for a description of the variables in the dataset.

```
library(tidyverse)
diamonds
```

(a) Write a dplyr pipe that produces a summary table showing for each *cut* of diamond the number of diamonds of that cut, along with the minimum, median and maximum price of diamonds of that cut.

(b) Write a dplyr pipe to determine what *color* of diamond has the highest mean of price per carat among diamonds with Ideal cut.

Bonus. Just to convince you that dplyr pipes are fantastic, we will have a look at how average arrival delay varies by month. Note that `month.name` is a vector containing the names of the months that is built in to R.

```
flights %>%
  group_by(month) %>%
  summarise(mean_delay=mean(arr_delay, na.rm=TRUE)) %>%
  ggplot(aes(x=mean_delay, y=month)) +
    geom_col() +
    scale_y_discrete(limits=month.name)
```

2. Advanced Joins (optional)

In dplyr, there are many different types of joins.

- Have look at https://stat545.com/join-cheatsheet.html for some great examples (featuring superheroes and publishers datasets) of the different types of joins.
- Chapter 13 Relational Data from *R for Data Science* has lots more information and some helpful diagrams: https://r4ds.had.co.nz/relational-data.html

Mutating joins	
`left_join(A,B)`	All rows from A are returned with extra columns added with information from B. Rows from A without a match will have NA in the extra columns.
`right_join(A,B)`	All rows from B are returned with exta columns added with information from A. Rows from B without a match will have NA in the extra columns.
`inner_join(A,B)`	Only rows in both data frames are returned. Observations in A that have no match in B or observations in B that have no match in A, will not be returned at all.

`full_join(A,B)`	All rows from both tables are returned. Observations without a match will have NA in the columns from the other table.
Filtering joins	
`semi_join(A,B)`	Keeps all observations in A that have a match in B.
`anti_join(A,B)`	Drops all observations in A that have a match in B.

Look at the "Data Transformation with dplyr" Cheat Sheet in the third column of the second page headed up "Combine Tables" | "Combine Variables". Here we see the different types of *mutating joins*. For the *filtering joins*, see the bottom right hand corner.

Summary

In this minilab, we have seen that using a join (just like in SQL) allows us to combine information in two tibbles via a *common key*.

We have also established a pattern for common forms of data queries:

- **filter** out the rows that meet the condition
- create a summary by group to produce a summary tibble (**group_by** and **summarise**)
- **filter** out the row you want from the summary tibble (the "solution" to the problem)
- **join** with another dataset that includes more information about the solution, and
- **select** the information you want to provide as the final answer.

The "Data Transformation with dplyr" Cheat Sheet contains a lot of useful information.