STAT 380 –Data Wrangling Review Part 2 (Lecture 3)

IDEA: Data analysis commonly involves more than one dataset (OR table OR dataframe). As part of the process of preparing the data for analysis, we will often have to combine datasets.

Relational Databases

NOTE: Storing data in separate tables can be beneficial even when the data are coming from the same source:

- My experiences: Insurance Companies, Zillow
- Organizations can (and do) restrict access to tables with sensitive information, while permitting wider access with non-sensitive data tables.
    - HIPPA & patient identifiers in healthcare industry
    - FERPA & student records at educational institutions
- Often the kinds of analysis that will be done are not specifically anticipated when data are collected and additional data is collected at future times.

IDEA: We want to join related tables as needed rather than smash all available data into one big table.

Joins

NOTE: We will examine several methods for joining data frames/tables using dplyr functions (part of tidyverse). A join is a way of connecting each row in table (dataset) X to zero, one, or more rows in a second table Y. Joins establish a correspondence — i.e. match — between each row in the left table and zero or more rows in the right table. The join function takes the form:

```
xxxx_join(X, Y)
```

Several examples include:

- `left_join(X, Y)`: keeps all observations (rows) from X (X is the 1st dataset listed)
- `right_join(X, Y)`: keeps all observations (rows) from Y (Y is the 2nd dataset listed). Does the same thing as left_join(Y,X)
- `inner_join(X, Y)`: keeps all rows from X where there are matching values in Y, return all combination of multiple matches in the case of multiple matches. The most important property of an inner join is that unmatched rows are not included in the result.
- `full_join(X, Y)`: keeps all observations (rows) from both X and Y
- `semi_join(X, Y)`: keeps all rows from X where there are matching values in Y, keeping just columns from X. This is an example of a join that can be used for filtering.
- `anti_join(X, Y)`: returns all rows from S where there are not matching values in Y, never duplicate rows of Y. This is an example of a join that can be used for filtering.

NOTE: A match between a row in the left table and a row in the right table is made based on the values in pairs of corresponding variables. Some notes:

- You specify which pair(s) to use.
- A pair is a variable from the left table and a variable from the right table.
- Fine to specify several variables for matching
    - For instance we might use `by = c("A" = "var1", "B" = "var2")` which matches variables A and B from left table to variables var1 and var2 from right table
- If you don't specify the variables directly, the default value of by = is all variables with the same names in both tables
    - This is not reliable in general unless you've checked
- Cases must have **exactly equal** values in the left variable and right variable for a match to be made.

EXAMPLE 1: Helpful tutorials may be found at https://statisticsglobe.com/r-dplyr-join-inner-left-right-full-semi-anti and https://datasciencebox.org/course-materials/_slides/u2-d08-multi-df/u2-d08-multi-df.html#7 (through the slide ending in .html#21). If using the second resource, consider the following:

Table Name:	X		Y	
Variable Names:	ID	xVar	ID	yVar
	1	x1	1	y1
	2	x2	2	y2
	3	x3	4	y4

IDEA: `left_join(X, Y)`: joins matching rows from the right table (Y) to the left table (X)

CODE: `left_join(X, Y)` OR
`X %>% left_join(Y)`

RESULT:

1	x1	y1
2	x2	y2
3	x3	

IDEA: `inner_join(X, Y)`: only retains rows for which a match exists

CODE: `inner_join(X, Y)` OR
`X %>% inner_join(Y)`

RESULT:

1	x1	y1
2	x2	y2

NOTE: The difference between these two joins lies in what we do when there is no match.

NOTE: Different joins have different answers to these questions.

- What to do when there is **no match** between a left case and any right case?
- What to do when there are **multiple matching cases** in the right table for a case in the left table?

IF no right cases match the left case…

- `left_join()`: Keep the left case and fill in the new variables (from the right table) with `NA`
- `inner_join()`: Discard the left case.

IF multiple right cases match the left case…

`left_join()` and `inner_join()` do the same thing:

- `left_join()`: Keep **all combinations**.
- `inner_join()`: Keep **all combinations**.

EXAMPLE 2: In order to illustrate some properties of left and inner joins, we will consider two datasets related to superhero comic book characters. The 'superheroes' table consists of 7 observations and 4 variables. The variables represent the name of the superhero, their alignment, their gender, and the publisher of the comic. The 'publishers' table consists of 3 observations and 2 variables. The variables represent the publisher and the year they were founded.

superheroes

	name	alignment	gender	publisher
1	Magneto	bad	male	Marvel
2	Storm	good	female	Marvel
3	Mystique	bad	female	Marvel
4	Batman	good	male	DC
5	Joker	bad	male	DC
6	Catwoman	bad	female	DC
7	Hellboy	good	male	Dark Horse Comics

publishers

	publisher	yr_founded
1	DC	1934
2	Marvel	1939
3	Image	1992

Three Questions to Consider:
1. How do we establish a match?
2. What if there is no match?
3. What if there are multiple matches?

Use these tables to answer the questions that follow.

a. If no pairs are specified, which variable(s) is(are) used for performing joins?

b. How many observations and variables (or rows and columns) would be in the dataset that results from: `inner_join(publishers, superheroes)`? Sketch the dataset below.

superheroes

	name	alignment	gender	publisher
1	Magneto	bad	male	Marvel
2	Storm	good	female	Marvel
3	Mystique	bad	female	Marvel
4	Batman	good	male	DC
5	Joker	bad	male	DC
6	Catwoman	bad	female	DC
7	Hellboy	good	male	Dark Horse Comics

publishers

	publisher	yr_founded
1	DC	1934
2	Marvel	1939
3	Image	1992

Three Questions to Consider:
1. How do we establish a match?
2. What if there is no match?
3. What if there are multiple matches?

c. Download the file STAT380_L3.Rmd from Canvas. One way to know whether your answers are correct is to create the datasets and run the commands. The .Rmd file contains one way of creating the tables for superheroes and publishers. Run the code and verify the results for Part b.

```
> inner_join(publishers, superheroes)
Joining, by = "publisher"
# A tibble: 6 × 5
  publisher yr_founded name     alignment gender
  <chr>          <dbl> <chr>    <chr>     <chr>
1 DC              1934 Batman   good      male
2 DC              1934 Joker    bad       male
3 DC              1934 Catwoman bad       female
4 Marvel          1939 Magneto  bad       male
5 Marvel          1939 Storm    good      female
6 Marvel          1939 Mystique bad       female
```

d. For each of the following, predict the size of the resulting table. You should also consider the order of the columns, the amount of NA's, and write code in R to check your answer.

- `left_join(superheroes, publishers, by = c("publisher" = "publisher"))`
- `publishers %>% left_join(superheroes, by = c("publisher" = "publisher")(`
- `inner_join(superheroes, publishers)`

EXAMPLE 3: This example uses the several datasets (tables) found in the `nycflights13` package. The primary dataset is the `flights` dataset which contains information for all flights departing from New York City (NYC) airport (i.e. airports JFK, LGA or EWR) in 2013. The dataset contains information about the date of the flight, carrier (2 character abbreviation), origin of the flight (3 character FAA code), the destination of the flight (3 character FAA code), distance, departure delays, arrival delays, etc. You can view additional details about `flights` by accessing the help file (?flights). The package `nycflights13` also contains 4 additional datasets that are helpful for analyzing the `flights` data:

- *airlines* provides the full carrier name from its abbreviated code
- *airports* provides information about each airport, identified by the faa airport code
- *planes* gives information about each plane, identified by its tailnum
- *weather* gives the weather at each NYC airport for each hour

a. Write code that allows us to find out which carrier had the longest average (mean) arrival delays for flights departing from NYC. Your code should allow you to find the carrier with the longest mean arrival delays without manually having to search through a table.

```
## # A tibble: 16 × 3
##    carrier TotalFlights MeanArrivalDelay
##    <chr>          <int>            <dbl>
##  1 F9               685           21.9
##  2 FL              3260           20.1
##  3 EV             54173           15.8
##  4 YV               601           15.6
##  5 OO                32           11.9
##  6 MQ             26397           10.8
##  7 WN             12275            9.65
##  8 B6             54635            9.46
##  9 9E             18460            7.38
## 10 UA             58665            3.56
## 11 US             20536            2.13
## 12 VX              5162            1.76
## 13 DL             48110            1.64
## 14 AA             32729            0.364
## 15 HA               342           -6.92
## 16 AS               714           -9.93
```

b. What is the name of the carrier that had the second longest average (mean) arrival delays when departing from NYC? Unfortunately, the flights dataset only has a brief abbreviation for the carrier and it is not obvious what the abbreviations represent. Fortunately, the `airlines` dataset, also from the nycflights13 package, has the information we need. This is a case where we want to perform a join so that we can combine data from two tables (datasets). Here, we want to bring the full name of the carrier into the dataset.

Instead of the table shown in Part a., we want to add the name of the airline to our resulting dataset:

```
## # A tibble: 16 × 4
##    carrier TotalFlights MeanArrivalDelay name
##    <chr>          <int>            <dbl> <chr>
##  1 F9               685            21.9  Frontier Airlines Inc.
##  2 FL              3260            20.1  AirTran Airways Corporation
##  3 EV             54173            15.8  ExpressJet Airlines Inc.
##  4 YV               601            15.6  Mesa Airlines Inc.
##  5 OO                32            11.9  SkyWest Airlines Inc.
##  6 MQ             26397            10.8  Envoy Air
##  7 WN             12275             9.65 Southwest Airlines Co.
##  8 B6             54635             9.46 JetBlue Airways
##  9 9E             18460             7.38 Endeavor Air Inc.
## 10 UA             58665             3.56 United Air Lines Inc.
## 11 US             20536             2.13 US Airways Inc.
## 12 VX              5162             1.76 Virgin America
## 13 DL             48110             1.64 Delta Air Lines Inc.
## 14 AA             32729             0.364 American Airlines Inc.
## 15 HA               342            -6.92 Hawaiian Airlines Inc.
## 16 AS               714            -9.93 Alaska Airlines Inc.
```

c. In class, multiple students asked about making nicer tables. One useful resource is Chapter 10 of the RMarkdown Cookbook: https://bookdown.org/yihui/rmarkdown-cookbook/tables.html. We can use the kable function from the knitr library.

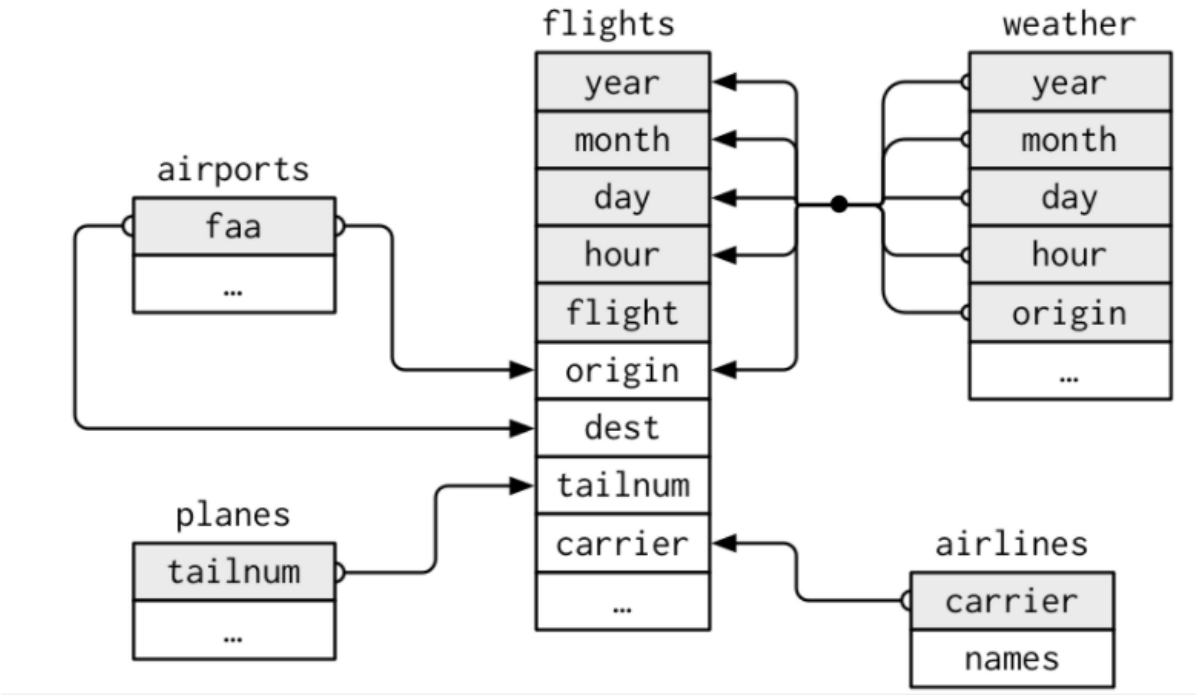
```
flights %>%
  group_by(carrier) %>%
  summarize(TotalFlights = n(),
            MeanArrivalDelay = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(desc(MeanArrivalDelay)) %>%
  left_join(airlines, by = c("carrier" = "carrier")) %>%
  select(name, carrier, TotalFlights, MeanArrivalDelay) %>%
  knitr::kable(digits = 1, align = "lccc" ,
               col.names = c("Name", "Carrier", "Number of Flights", "Mean Arrival Delay"))
```

Name	Carrier	Number of Flights	Mean Arrival Delay
Frontier Airlines Inc.	F9	685	21.9
AirTran Airways Corporation	FL	3260	20.1
ExpressJet Airlines Inc.	EV	54173	15.8
Mesa Airlines Inc.	YV	601	15.6
SkyWest Airlines Inc.	OO	32	11.9
Envoy Air	MQ	26397	10.8
Southwest Airlines Co.	WN	12275	9.6
JetBlue Airways	B6	54635	9.5
Endeavor Air Inc.	9E	18460	7.4
United Air Lines Inc.	UA	58665	3.6
US Airways Inc.	US	20536	2.1
Virgin America	VX	5162	1.8
Delta Air Lines Inc.	DL	48110	1.6
American Airlines Inc.	AA	32729	0.4
Hawaiian Airlines Inc.	HA	342	-6.9
Alaska Airlines Inc.	AS	714	-9.9

IDEA: In order to combine/join these datasets, you have to know something about how the datasets are connected and how we can establish a match. This involves knowing the variable (or set of variables) that uniquely identifies an observation, which we often call the key.

NOTE: In some (nice) cases, you will be given information about the relation between the datasets.

EXAMPLE 4: Based on nycflights13 package, we have the following:



The key to understanding diagrams like this is to remember each relation always concerns a pair of tables. You don't need to understand the whole thing; you just need to understand the chain of relations between the tables that you are interested in.

For the nycflights13 tables (datasets),

- flights connects to planes via a single variable, tailnum.
- flights connects to airlines through the carrier variable.
- flights connects to weather via origin (the location), and year, month, day and hour (the time).
- flights connects to airports in two ways: via the origin and dest variables in flights and the faa variable in airiports

NOTE: The best way to become familiar with the data wrangling commands from Lectures 2 and 3 is to see/use more examples. The file DataWrangSupp.Rmd may be found on Canvas. It is a document that your instructor created with additional examples and more advanced ideas related to joins (joining on multiple variables, importance of using by = , and joining when datasets have different names.) You should download the file and read through the examples.