Chapter 5: Data Wrangling on Multiple Tables

- In Chapter 4, we learned about five key verbs used to alter, adjust, or manipulate a single data table.
- In real data science, we often work with multiple data tables which contain values on related or identical individuals.
- Think about the baseball example from last chapter:
- Suppose we had one table with batting data, another table with pitching data, another table with fielding data, tables with attendance data, financial data, etc.
- Different people may collect and update such different data sets, and it may be inefficient or impossible to store it in one large table.

Joining Tables

- To merge the *information* in different tables, we can use table *joins*.
- Various functions in the `dplyr` package do various types of joins: `inner_join`, `left_join`, `right_join`, `full_join`
- We need some *key* column (such as an ID column) that uniquely identifies individuals across the data sets.
- Note that the `merge` function in `base` R does similar operations using a bit different syntax.

Different Types of Joins

- We could perform an *Inner Join*, a *Left Join*, a *Right Join*, or a *Full (outer) Join*.
- The inner join results in a table containing the cases that appear in both tables that are being joined.
- The left join results in a table containing the cases that are in both tables or just in the first table.
- The right join results in a table containing the cases that are in both tables or just in the second table.
- The full join results in a table containing the cases that are in *either* table (or in both tables).

Simple Examples of Joins

- See the simple example on the Math class and Reading class tables.
- We see that some columns in the left, right, and full joins will necessarily contain missing values, for the cases that did not appear in one of the original tables.
- Note that when the name of the key column is different between the first and second table, the name of the corresponding column in the joined table is taken from the *first* table.

The NYC Flights tables

- As a more realistic example, consider the `flights` table in the `nycflights13` package.
- It contains $336,776$ rows: Each row represents one commercial flight from New York City (there are three airports in NYC).
- The `carrier` column contains the airline name, but only in an abbreviated form.
- In the `airlines` data table, we have the carrier abbreviation, plus the full airline name.

Example with the `inner_join` function

- We can create a table of flights that contains both the abbreviated carrier name and a full carrier name by doing an inner join with: `by = c("carrier" = "carrier")`
- Since the key column has the same name in both tables, could do: `by = join_by(carrier)`
- Note in this example, the resulting table has the same number of rows as the original `flights` table.

Example with the `left_join` function

- We can create an `airports_pt` data frame that contains information about the airports that are in the Pacific time zone.
- An inner join of `flights` with `airports_pt` results in a table with only the flights going to airports in the Pacific time zone.
- A left join of `flights` with `airports_pt` results in a table with all the NYC flights, but ...
- ... it contains airport information on only some of these cases (only the flights going to airports in the Pacific time zone).
- Based on this, we can easily get counts of the number of flights to the Pacific time zone and the number going elsewhere, using `summarize`.

Checking the Results of Joins

- It's always a good idea to check the dimensions (especially the number of rows of your original table and of your joined table).
- Check for any cases of a row in one table matched to more than one row in the other table.
- This can happen if the key column is a nonunique identifier, such as a Name that could be shared by multiple individuals.
- You can join multiple tables in a similar way – can do it efficiently using the pipe operators.

Extended Example

- See the extended example in Section 5.3 on the baseball data in the `Lahman` package.
- This combines some tools learned in Chapter 4 for wrangling data in one table with some of the Chapter 5 tools for wrangling data in multiple tables.