Lecture 13: Wrangling Relational Data with `dplyr`

UNIVERSITY OF
SAN FRANCISCO

Abbie M. Popa

BSDS 100 - Intro to Data Science with `R`

Relational Data in `R`

- Up until now, we have almost exclusively been wrangling data by making adjustments to a single data frame
- We have combined the occasional data frame with `rbind()` and `cbind()`, but these don't merge data in a very informed way

Example

- We have two data frames, one which lists the movie titles and genres, and one which lists the movie titles and gross profit
- We want to look at profit by genre, so we need to combine these into a single data frame
- We can use `cbind`, but what if the movies are out of order? Or worse yet, what if one movie is only in one data frame?

Example, cont

- We can make this work using various functions we've learned, but it's cumbersome and error-prone
- There is a built in function to `R` named `merge` which works fairly well
- We will focus on the functions of `dplyr` which give us more control than does `merge`

An intro to `dplyr`

- A very common place to see relational data tables is in a `SQL` database (structured query language)
- Though we will not be covering `SQL` directly, `dplyr` borrows a core idea from `SQL` called "joins"
- Joins operate on two tables, the most common thing for them to do is to combine the two tables

Joins with `dplyr`

- We join data by matching rows on some column
- If the "key" in the matching column is in both tables this is easiest to understand, so for our first example lets fix up our movie example by adding a row to the profit data
- Load the `dplyr` library with `library(dplyr)`
- Then join the two data frames with `full_join(movie_genre, movie_profit, by = "title")`

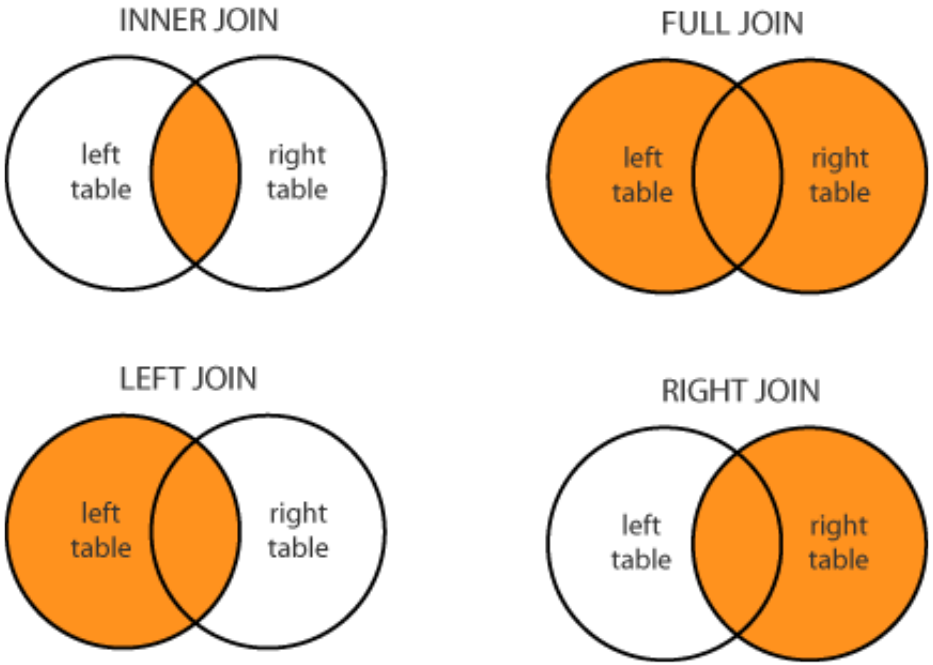
Joins with `dplyr`

- Notice how the new data frame has only one column for "title", but has columns for both "genre" and "profit_mil"
- For the sake of simplicity we did this on data frames that contained all the same movies, but `dplyr` can also accurately deal with data frames that have unmatched data

Mutating Joins

- Joins that work like this, where we preserve variables from both data frames, are called "mutating joins"
- When there is unmatched data you can choose how to deal with it by picking a specific mutating join

Types of Mutating Joins



- Inner joins only keep rows that are in **both** data sets
- Left joins keep **all** rows in the **first** (left) data set, if they aren't in the second (right) data set, that variable value will be `NA`
- Right joins keep **all** rows in the **second** (right) data, if they aren't in the first (left) data set, that variable value will be `NA`
- Full joins keep **all** rows in **both** data frames, missing values will be `NA`

Examples of Mutating Joins

- First, lets add an unmatched row to each movie data frame
- What is the output of each type of join on the movie data frame?

Original Data		
Data Frame Name	number of rows	number of columns
movie_genre	5	2
movie_profit	5	2

Mutated Data			
Type of Join	number of rows	number of columns	number of `NA`s
Inner	4	3	0
Left	5	3	1
Right	5	3	1
Full	6	3	2

Practice

Load the two tables into `R` using the following code:

```
cust_data <- read.csv("https://github.com/abbiepopa/
          BSDS100/raw/master/Data/customer_table.csv")

trans_data <- read.csv("https://github.com/abbiepopa/
          BSDS100/raw/master/Data/transaction_table.csv")
```

Join the two tables preserving **all** the **transaction** data by filling in the blanks:

```
_____join(trans_data, cust_data, by = ____)
```

Hint: check the names of the columns in each data frame to look up what should follow `by =`

Filtering Joins

- So far we have been using joins to combine two data frames, meaning we want the variables from both data frames (though we may not want the rows from both data frames)
- We may also want to use a second data frame to determine which rows in the first data frame we want to keep
- For example, perhaps I conduct an experiment, but find some participants had a health condition that means I should exclude them. How can I efficiently remove these participants?
- This is why we use filtering joins

Types of Filtering Joins

- `semi_join(a, b)` keeps only rows in `a` that match with `b` (but variables from `b` are not included)
- `anti_join(a, b)` keeps only rows in `a` that do NOT match with `b` (variables from `b` are not included)

Example

- I have data from 10 participants:
  - `d` all data
  - `consent_on_file` who do I have a consent form from?
  - `known_health_condition` who has a health condition, and thus should be excluded?
- First, keep only participants who have a consent form with a `semi_join`

```
d_consent <- semi_join(d, consent_on_file, by = "pid")
```

- Then, remove participants with a health condition using an `anti_join`

```
d_consent_healthy <- anti_join(d_consent, known_health_condition,
by = "pid")
```

Practice

If you don't still have it, reload the customer and transaction data:

```
cust_data <- read.csv("https://github.com/abbiepopa/
        BSDS100/raw/master/Data/customer_table.csv")

trans_data <- read.csv("https://github.com/abbiepopa/
        BSDS100/raw/master/Data/transaction_table.csv")
```

Keep only the customers who have shopped at the store (i.e., customers who have an entry in `trans_data`) by filling in the blanks:

```
_____join(____, ____, by = ____)
```

End of Class

- Complete the lab on github https://github.com/abbiepopa/BSDS100 and submit to canvas
- You may work together, but please each submit your own work on canvas
- Accuracy is not required, but an attempt is