Joining Data

Max Turgeon

DATA 2010—Tools and Techniques in Data Science

Lecture Objectives

- Understand the difference between the different types of joins
- Be able to choose and select the appropriate mutating join or filtering join

Motivation

- In the last lecture, we discussed the tidy data framework.
- Recall: each observational unit should be in its own dataset.
    - For easier maintenance
- Today we will learn how to recombine each dataset for analysis/visualization etc.

Relational data i

- So far we've looked at data that fits neatly into a `data.frame`.
    - Each row is an observation, and for each observation we collected the **same** variables.
- This is not the only way to store data. Let's look at an example: university course enrollment data.
    - For every student we need to collect personal information.
    - For every course we need to collect specific information.
- Clearly these datasets should be separate; you can think of them as two different `data.frame`s.
- **Question**: How should we store information about which courses students are taking?

Relational data ii

- Should we add the name of courses to the student `data.frame` as new variables? How many variables should we create?
- Should we add the name of students to the course `data.frame` as new variables? How many variables should we create?
- **A better solution**: Create a *new* dataset, where each row corresponds to a pair (student, course).
- **Why does this work?** Each student has a **unique** identifier, and so does each course.

Relational data iii

- To create a class list:
  - Filter the (student, course) `data.frame` to only keep pairs for a given course.
  - Look up which students appear in the filtered dataset
  - Keep relevant personal information (e.g. student number, major, degree)
- The process of "looking up" is called a **mutating join**.

Example i

- This dataset is separated into two CSV files:
    - One contains a list of 2,410 US craft beers
    - The other contains data on 510 US breweries
- The beers and breweries datasets have a variable in common, called `brewery_id`.

```
library(tidyverse)

df_beers <- read_csv("beers.csv")
df_breweries <- read_csv("breweries.csv")

glimpse(df_beers)
```

Example ii

```
## Rows: 2,410
## Columns: 7
## $ abv <dbl> 0.050, 0.066, 0.071, 0.090, 0~
## $ ibu <dbl> NA, NA, NA, NA, NA, NA, NA, N~
## $ id <dbl> 1436, 2265, 2264, 2263, 2262,~
## $ name <chr> "Pub Beer", "Devil's Cup", "R~
## $ style <chr> "American Pale Lager", "Ameri~
## $ brewery_id <dbl> 408, 177, 177, 177, 177,
177,~
## $ ounces <dbl> 12, 12, 12, 12, 12, 12, 12, 1~
```

Example iii

```
glimpse(df_breweries)
```

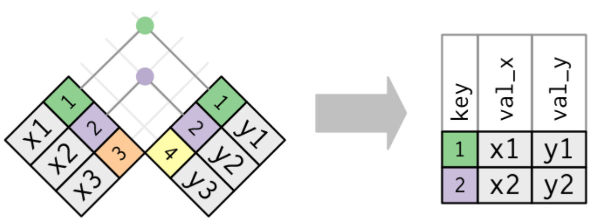
```
## Rows: 558
## Columns: 4
## $ brewery_id <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8,
9,~
## $ name <chr> ”NorthGate Brewing”, ”Against~
## $ city <chr> ”Minneapolis”, ”Louisville”, ~
## $ state <chr> ”MN”, ”KY”, ”MA”, ”CA”, ”CA”,~
```

Mutating joins

- **Mutating joins** create a new dataset by combining two datasets and respecting their relationship.
    - This relationship is encoded by a common variable (or set of variables), often a unique identifier.
- The main idea is as follows:
    - Take a row from the first dataset
    - Find a matching row in the second dataset
    - Create a new row by concatenating the two rows
- The different types of mutating joins differ in how they handle cases with no matches.

Inner join

- In inner joins, we only create a new row if we can match rows from both datasets.



Example i

```r
library(tidyverse)

dataset <- inner_join(df_beers,
                      df_breweries,
                      by = "brewery_id")
glimpse(dataset)
```

Example ii

```
## Rows: 2,410
## Columns: 10
## $ abv <dbl> 0.050, 0.066, 0.071, 0.090, 0~
## $ ibu <dbl> NA, NA, NA, NA, NA, NA, NA, N~
## $ id <dbl> 1436, 2265, 2264, 2263, 2262,~
## $ name.x <chr> "Pub Beer", "Devil's Cup", "R~
## $ style <chr> "American Pale Lager", "Ameri~
## $ brewery_id <dbl> 408, 177, 177, 177, 177,
177,~
## $ ounces <dbl> 12, 12, 12, 12, 12, 12, 12, 1~
## $ name.y <chr> "10 Barrel Brewing Company", ~
## $ city <chr> "Bend", "Gary", "Gary", "Gary~
```

Example iii

```
## $ state <chr> "OR", "IN", "IN", "IN", "IN",~
```

```
# dataset and df_beers have the same # of rows
nrow(dataset) == nrow(df_beers)
```

```
## [1] TRUE
```

```
# dataset has one less than the sum of # cols
c(ncol(dataset), ncol(df_beers), ncol(df_breweries))
```

```
## [1] 10  7  4
```

Exercise

Find the state with the highest average of alcohol by volume (`abv`) per beers.

Solution i

- Now that the datasets are joined, we can use `group_by` and `summarise`.

```
# Careful about missing values!
dataset %>%
  group_by(state) %>%
  summarise(avg_abv = mean(abv, na.rm = TRUE)) %>%
  filter(avg_abv == max(avg_abv))
```

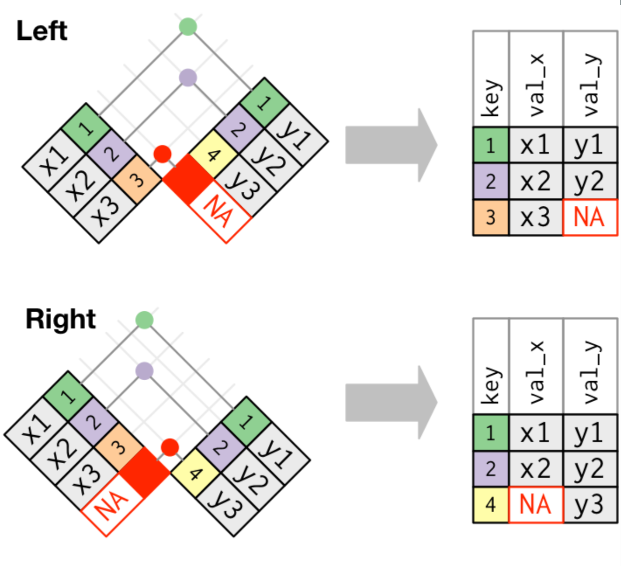
Solution ii

```
## # A tibble: 1 x 2
##   state avg_abv
##   <chr>   <dbl>
## 1 NV     0.0669
```

Left/right join i

- But what if we want to keep rows from a dataset that don't have a matching row in the other dataset?
- Left and right (outer) joins will do just that and replace the non-matching row with `NA`s.
- Left and right refer to the dataset from which we want to keep rows.
    - `left_join(x, y)` will keep rows of `x`
    - `right_join(x, y)` will keep rows of `y`

Left/right join ii



Example i

```r
library(nycflights13)

# Information about flights
glimpse(flights)
```

```
## Rows: 336,776
## Columns: 19
## $ year <int> 2013, 2013, 2013, 2013, 2~
## $ month <int> 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ day <int> 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ dep_time <int> 517, 533, 542, 544, 554, ~
```

Example ii

```
## $ sched_dep_time <int> 515, 529, 540, 545,
600, ~
## $ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, ~
## $ arr_time <int> 830, 850, 923, 1004, 812,~
## $ sched_arr_time <int> 819, 830, 850, 1022,
837,~
## $ arr_delay <dbl> 11, 20, 33, -18, -25, 12,~
## $ carrier <chr> "UA", "UA", "AA", "B6", "~
## $ flight <int> 1545, 1714, 1141, 725, 46~
## $ tailnum <chr> "N14228", "N24211", "N619~
## $ origin <chr> "EWR", "LGA", "JFK", "JFK~
## $ dest <chr> "IAH", "IAH", "MIA", "BQN~
```

Example iii

```
## $ air_time <dbl> 227, 227, 160, 183, 116, ~
## $ distance <dbl> 1400, 1416, 1089, 1576, 7~
## $ hour <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6~
## $ minute <dbl> 15, 29, 40, 45, 0, 58, 0,~
## $ time_hour <dttm> 2013-01-01 05:00:00, 201~
```

```
# Information about airplanes
glimpse(planes)
```

Example iv

```
## Rows: 3,322
## Columns: 9
## $ tailnum <chr> "N10156", "N102UW", "N103US~
## $ year <int> 2004, 1998, 1999, 1999, 200~
## $ type <chr> "Fixed wing multi engine", ~
## $ manufacturer <chr> "EMBRAER", "AIRBUS
INDUSTRI~
## $ model <chr> "EMB-145XR", "A320-214", "A~
## $ engines <int> 2, 2, 2, 2, 2, 2, 2, 2, 2, ~
## $ seats <int> 55, 182, 182, 182, 55, 182,~
## $ speed <int> NA, NA, NA, NA, NA, NA, NA,~
## $ engine <chr> "Turbo-fan", "Turbo-fan", "~
```

Example v

```r
# How many flights? How many planes?
c(nrow(flights), nrow(planes))
```

```
## [1] 336776   3322
```

```r
# How many flights have matching plane?
inner_join(flights, planes, by = "tailnum") %>%
  nrow
```

```
## [1] 284170
```

Example vi

```r
# With left_join, we keep all flights
left_join(flights, planes, by = "tailnum") %>%
  glimpse
```

```
## Rows: 336,776
## Columns: 27
## $ year.x <int> 2013, 2013, 2013, 2013, 2~
## $ month <int> 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ day <int> 1, 1, 1, 1, 1, 1, 1, 1, 1~
## $ dep_time <int> 517, 533, 542, 544, 554, ~
## $ sched_dep_time <int> 515, 529, 540, 545,
```

Example vii

```
600, ~
## $ dep_delay <dbl> 2, 4, 2, -1, -6, -4, -5, ~
## $ arr_time <int> 830, 850, 923, 1004, 812,~
## $ sched_arr_time <int> 819, 830, 850, 1022,
837,~
## $ arr_delay <dbl> 11, 20, 33, -18, -25, 12,~
## $ carrier <chr> "UA", "UA", "AA", "B6", "~
## $ flight <int> 1545, 1714, 1141, 725, 46~
## $ tailnum <chr> "N14228", "N24211", "N619~
## $ origin <chr> "EWR", "LGA", "JFK", "JFK~
## $ dest <chr> "IAH", "IAH", "MIA", "BQN~
## $ air_time <dbl> 227, 227, 160, 183, 116, ~
```

Example viii

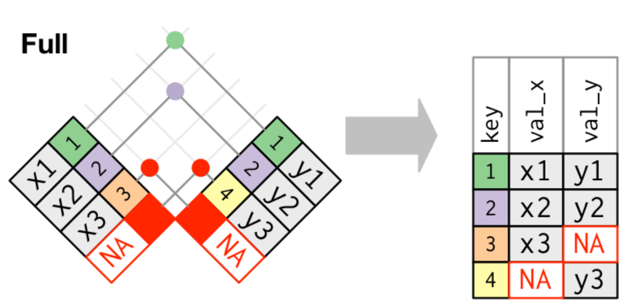
```
## $ distance <dbl> 1400, 1416, 1089, 1576, 7~
## $ hour <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6~
## $ minute <dbl> 15, 29, 40, 45, 0, 58, 0,~
## $ time_hour <dttm> 2013-01-01 05:00:00, 201~
## $ year.y <int> 1999, 1998, 1990, 2012, 1~
## $ type <chr> "Fixed wing multi engine"~
## $ manufacturer <chr> "BOEING", "BOEING",
"BOEI~
## $ model <chr> "737-824", "737-824", "75~
## $ engines <int> 2, 2, 2, 2, 2, 2, 2, 2, 2~
## $ seats <int> 149, 149, 178, 200, 178, ~
## $ speed <int> NA, NA, NA, NA, NA, NA, N~
```

Example ix

```
## $ engine <chr> "Turbo-fan", "Turbo-fan",~
```

Full join

- The full join allows us to keep unmatched rows from **both** datasets.



Exercise

The `flights` dataset contains information about departure and arrival delays (`dep_delay` and `arr_delay`). Compute the average delays for each manufacturing year (i.e. the year the plane was manufactured).

Solution i

```r
# First we combine the two datasets
# and create tot_delay variable
# Note: Could also use inner_join
dataset <- left_join(flights,
                     planes,
                     by = "tailnum") %>%
  mutate(tot_delay = dep_delay + arr_delay)
```

Solution ii

```
# Next group by year and summarise
data_avg <- dataset %>%
  group_by(year) %>%
  summarise(avg_delay = mean(tot_delay, na.rm = TRUE))
```

```
## Error: Must group by variables found in `.data`.
## * Column `year` is not found.
```

```
# What happened?
# Both flights and planes have a variable year
# year.y refers to the one from planes
names(dataset)
```

Solution iii

```
## [1] "year.x" "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" "year.y"
## [21] "type" "manufacturer" "model" "engines"
## [25] "seats" "speed" "engine" "tot_delay"
```

Solution iv

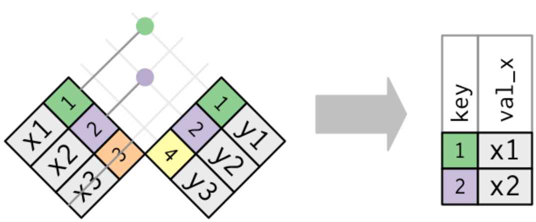
```
# Try again
data_avg <- dataset %>%
  group_by(year.y) %>%
  summarise(avg_delay = mean(tot_delay, na.rm = TRUE))
```

Filtering joins

- The starting point is still the same:
    - We have two `data.frame`s `x` and `y`
    - They have a variable in common that allows us to match rows across
- In filtering joins, we want to filter the rows of `x` based on their relationship with the rows of `y`.
    - In particular, the output of a filtering join is a *subset* of `x`.

Semijoin

- In a **semijoin**, we only keep the rows of x with a corresponding match in y



Example i

```r
library(tidyverse)

df_beers <- read_csv("beers.csv")
df_breweries <- read_csv("breweries.csv")
```

```r
# Top 5 states for # breweries
state_top5 <- df_breweries %>%
  count(state) %>%
  top_n(5)
```

Example ii

```
state_top5
```

```
## # A tibble: 5 x 2
##   state     n
##   <chr> <int>
## 1 CA       39
## 2 CO       47
## 3 MI       32
## 4 OR       29
## 5 TX       28
```

Example iii

```
breweries_top5 <- semi_join(df_breweries,
                            state_top5)
```

```
breweries_top5
```

```
## # A tibble: 175 x 4
## brewery_id name city state
## <dbl> <chr> <chr> <chr>
## 1 3 Mike Hess Brewing Company San Diego CA
## 2 4 Fort Point Beer Company San Francisco CA
## 3 6 Great Divide Brewing Company Denver CO
```

Example iv

```
## 4 7 Tapistry Brewing Bridgman MI
## 5 8 Big Lake Brewing Holland MI
## 6 9 The Mitten Brewing Company Grand Rapids MI
## 7 10 Brewery Vivant Grand Rapids MI
## 8 11 Petoskey Brewing Petoskey MI
## 9 12 Blackrocks Brewery Marquette MI
## 10 13 Perrin Brewing Company Comstock Park MI
## # ... with 165 more rows
```

Example v

```
# Only keep beers from these states
semi_join(df_beers,
          breweries_top5,
          by = "brewery_id") %>%
  count(style, sort = TRUE)
```

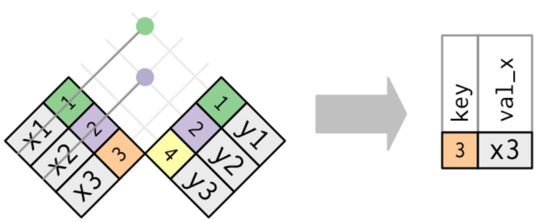
```
## # A tibble: 86 x 2
##    style                                n
##    <chr>                            <int>
##  1 American IPA                       141
##  2 American Pale Ale (APA)             90
```

Example vi

```
##  3 American Amber / Red Ale            57
##  4 American Double / Imperial IPA      43
##  5 American Blonde Ale                 38
##  6 American Pale Wheat Ale             38
##  7 Saison / Farmhouse Ale              24
##  8 American Brown Ale                  21
##  9 Cider                               21
## 10 American Stout                      20
## # ... with 76 more rows
```

Antijoin

- In an **antijoin**, we only keep the rows of x *without* a corresponding match in y



Example i

```
# Let's look at the other states
breweries_nottop5 <- anti_join(df_breweries,
                               state_top5)
```

```
breweries_nottop5
```

```
## # A tibble: 383 x 4
## brewery_id name city state
## <dbl> <chr> <chr> <chr>
## 1 0 NorthGate Brewing Minneapolis MN
## 2 1 Against the Grain Brewery Louisville KY
```

Example ii

```
## 3 2 Jack's Abby Craft Lagers Framingham MA
## 4 5 COAST Brewing Company Charleston SC
## 5 16 Flat 12 Bierwerks Indianapolis IN
## 6 17 Tin Man Brewing Company Evansville IN
## 7 18 Black Acre Brewing Co.  Indianapolis IN
## 8 19 Brew Link Brewing Plainfield IN
## 9 20 Bare Hands Brewery Granger IN
## 10 21 Three Pints Brewing Martinsville IN
## # ... with 373 more rows
```

Example iii

```r
# Only keep beers from these states
semi_join(df_beers,
          breweries_nottop5,
          by = "brewery_id") %>%
  count(style, sort = TRUE)
```

```
## # A tibble: 92 x 2
##    style                              n
##    <chr>                          <int>
##  1 American IPA                     283
##  2 American Pale Ale (APA)          155
```

Example iv

```
##  3 American Amber / Red Ale           76
##  4 American Blonde Ale                70
##  5 American Double / Imperial IPA     62
##  6 American Pale Wheat Ale            59
##  7 American Brown Ale                 49
##  8 American Porter                    49
##  9 Fruit / Vegetable Beer             36
## 10 Witbier                            31
## # ... with 82 more rows
```

Exercise

Filter the dataset `flights` from the `nycflights13` package to only show flights with planes that have flown at least 100 flights.

Solution i

```r
library(nycflights13)

planes100 <- flights %>%
  count(tailnum) %>%
  filter(n >= 100)
```

```r
flights100 <- semi_join(flights,
                        planes100)
```

Solution ii

```
# Do we get flights with missing
# tail number?
flights100 %>%
  filter(is.na(tailnum)) %>%
  nrow
```

```
## [1] 2512
```

Solution iii

```
# We can remove these NAs from planes100
planes100 <- filter(planes100,
                    !is.na(tailnum))
# Or we can remove them from flights100
flights100 <- filter(flights100,
                     !is.na(tailnum))
```

Some tips about joins

- You can join using more than one variable:

```
inner_join(x, y, by = c("var1", "var2"))
```

- You can join even when the same variable is named differently:

```
inner_join(x, y, by = c("name1" = "name2"))
```

Set operations i

- Here, the setup is slightly different.
  - We still have two `data.frame`s `x` and `y`.
  - But we assume they have **exactly** the same variables.
- We want to create a new dataset `z` that will also have the same variables as `x` and `y`.
- There are three different set operations:
  - **Union**: `z` has the unique observations from `x` and `y`.
  - **Intersection**: `z` has the observations common between `x` and `y`.
  - **Set difference**: `z` has the observations from `x` that are not in `y`.

Set operations ii

```r
library(tidyverse)
df1 <- tibble(
  x = c(1, 2),
  y = c(1, 1)
)
df2 <- tibble(
  x = c(1, 1),
  y = c(1, 2)
)
```

Set operations iii

```
# Note that we get 3 rows, not 4
# because of duplicates
union(df1, df2)
```

```
## # A tibble: 3 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
## 2     2     1
## 3     1     2
```

Set operations iv

```
intersect(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
```

Set operations v

```r
setdiff(df1, df2)
```

```
## # A tibble: 1 x 2
##         x     y
##     <dbl> <dbl>
## 1       2     1
```

Set operations vi

```r
# The order is important!
setdiff(df2, df1)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     2
```

Summary

- Not all data is neatly packaged into CSV files.
- Often the data we need is spread over multiple datasets.
- If these datasets have a matching variable, we can create a new dataset with matching rows using **mutating joins**.
- Choosing between an inner join, left/right join or full join depends on what we want to do with unmatched rows.
    - Do we keep all of them? Only those from one of the two datasets?