Practicing and Extending Joins

DATA 202 21FA

Reminders

- Data Science Firepit Tonight!

Also:

- Quiz 5 closes tonight
- Homework 3 due Friday
- Discussion Forum replies due tomorrow

Preview: replication project

Gradescope trial

See Moodle.

Q&A

> Can't you rewrite any `right_join` as a flipped `left_join`?

Yes! It's mostly convenience (piped data comes in on *left*).

> Can we join more than 2 tables at once?

SQL? yes. `dplyr` makes you split it up though.

Q: Why did we need `summarize()`?

```
purchases_and_prices %>%
  summarize(total_revenue = sum(price))
```

```
# A tibble: 1 × 1
  total_revenue
          <dbl>
1          5.75
```

```
sum(price)
```

```
Error in eval(expr, envir, enclos): object 'price' not found
```

```
sum(purchases_and_prices$price)
```

```
[1] 5.75
```

Aside: vectors

So far we've always packaged vectors in data frames. But they can live on their own.

```
purchases_and_prices %>% selec
```

```
# A tibble: 5 × 1
  price
  <dbl>
1  1
2  0.8
3  0.15
4  0.8
5  3
```

```
price_vec<- purchases_and_pric
price_vec
```

```
[1] 1.00 0.80 0.15 0.80 3.00
```

```
typeof(price_vec)
```

```
[1] "double"
```

Relational Data

- Bring together data that may never have been brought together before
- Values:
  - No one source has a complete view of the world
  - We need each other ("one body, many parts")
- Helps us get a more holistic picture
  - people aren't just their transactions; they also live somewhere

Example Applications

- Looking up abbreviations
- Connecting sales data to each customer's demographics
- Combining student data from Moodle, an online textbook, GitHub activity reports, ...

What others can you think of?

Code Together: Flight Delays

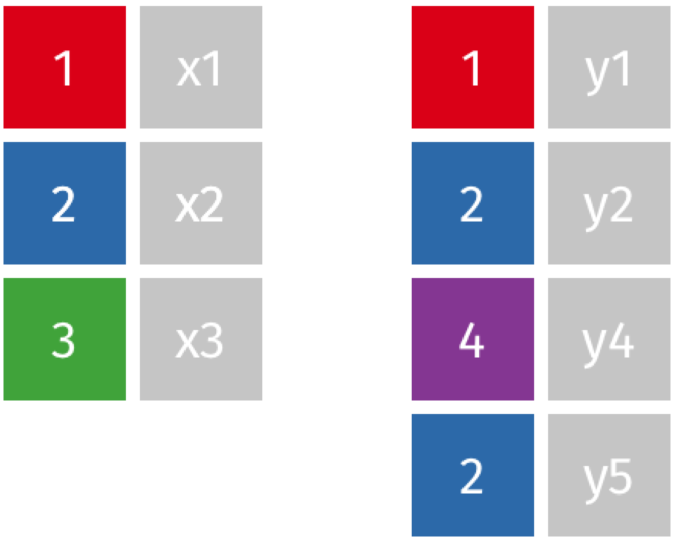
```
library(nycflights13)
```

```
flights %>%
  drop_na(arr_delay) %>%
  group_by(carrier) %>%
  summarize(avg_delay = mean(arr_delay)) %>%
  arrange(desc(avg_delay)) %>%
  left_join(airlines, by = "carrier") %>%
  select(name, avg_delay)
```

```
# A tibble: 16 × 2
  name                        avg_delay
  <chr>                           <dbl>
1 Frontier Airlines Inc.           21.9
2 AirTran Airways Corporation      20.1
3 ExpressJet Airlines Inc.         15.8
4 Mesa Airlines Inc.               15.6
5 SkyWest Airlines Inc.            11.9
6 Envoy Air                        10.8
# … with 10 more rows
```

Multiple matches

`left_join(x, y)`



Revenue by item?

`purchases`

customer_id	item
c1	bread
c1	milk
c1	banana
c2	milk
c2	toilet paper

`prices`

item	price
avocado	0.50
banana	0.15
bread	1.00
milk	0.80
toilet paper	3.00

Revenue by item?

For each item, look up all sales data.

```
prices %>%
  left_join(purchases)
```

```
# A tibble: 6 × 3
  item         price customer_id
  <chr>        <dbl> <chr>
1 avocado       0.5  <NA>
2 banana        0.15 c1
3 bread         1    c1
4 milk          0.8  c1
5 milk          0.8  c2
6 toilet paper  3    c2
```

Notice: multiple rows for each item. Where did each one come from?

Revenue by item?

```
prices %>%
  left_join(purchases) %>%
  group_by(item) %>%
  summarize(revenue = sum(price))
```

```
# A tibble: 5 × 2
  item         revenue
  <chr>          <dbl>
1 avocado         0.5
2 banana          0.15
3 bread           1
4 milk            1.6
5 toilet paper    3
```

...but nobody bought any avocados!

inner vs left

```
prices %>%
  left_join(purchases)
```

```
# A tibble: 6 × 3
  item         price customer_i
  <chr>        <dbl> <chr>
1 avocado       0.5  <NA>
2 banana        0.15 c1
3 bread         1    c1
4 milk          0.8  c1
5 milk          0.8  c2
6 toilet paper  3    c2
```

```
prices %>%
  inner_join(purchases)
```

```
# A tibble: 5 × 3
  item         price customer_i
  <chr>        <dbl> <chr>
1 banana        0.15 c1
2 bread         1    c1
3 milk          0.8  c1
4 milk          0.8  c2
5 toilet paper  3    c2
```

Put it together

```
prices %>%
  inner_join(purchases) %>%
  group_by(item) %>%
  summarize(revenue = sum(price))
```

```
# A tibble: 4 × 2
  item         revenue
  <chr>          <dbl>
1 banana          0.15
2 bread           1   
3 milk            1.6 
4 toilet paper    3   
```

uh oh...

customer_id	item
c1	bread
c1	milk
c1	bananas
c2	milk
c2	toilet paper

item	price
AVOCADO	0.50
BANANA	0.15
BREAD	1.00
MILK	0.80
TOILET_PAPER	3.00

uh oh...

customer_id	item
c1	bread
c1	milk
c1	bananas
c2	milk
c2	toilet paper

item	price
AVOCADO	0.50
BANANA	0.15
BREAD	1.00
MILK	0.80
TOILET_PAPER	3.00

```
purchases %>%
  left_join(prices)
```

```
# A tibble: 5 × 3
  customer_id item         price
  <chr>       <chr>        <dbl>
1 c1          bread           NA
2 c1          milk            NA
3 c1          bananas         NA
4 c2          milk            NA
5 c2          toilet paper    NA
```

Specifying keys

- Keys must match *exactly*
- Can join on multiple columns (first name **and** last name)
- Default join: columns with same names
- Specify what columns to use: `left_join(x, y, by = c("first_name", "last_name"))`

Make a new key that *does* match

```
prices_fixed <- prices %>%
  mutate(item_norm = str_to_lower(item))
purchases %>%
  left_join(prices_fixed,
            by = c("item" = "item_norm"))
```

```
# A tibble: 5 × 4
  customer_id item         item.y price
  <chr>       <chr>        <chr>  <dbl>
1 c1          bread        BREAD    1
2 c1          milk         MILK     0.8
3 c1          bananas      <NA>    NA
4 c2          milk         MILK     0.8
5 c2          toilet paper <NA>    NA
```

But still some mismatches

```r
purchases %>%
  anti_join(
    prices_fixed,
    by = c("item" = "item_norm
```

```r
prices_fixed %>%
  anti_join(
    purchases,
    by = c("item_norm" = "item
```

```
# A tibble: 2 × 2
  customer_id item
  <chr>       <chr>
1 c1          bananas
2 c2          toilet paper
```

```
# A tibble: 3 × 3
  item        price item_norm
  <chr>       <dbl> <chr>
1 AVOCADO      0.5  avocado
2 BANANA       0.15 banana
3 TOILET_PAPER 3    toilet_pap
```

Manual massaging: `if_else`

```r
prices_fixed <- prices %>%
  mutate(
    item_norm = str_to_lower(item),
    item_norm = if_else(
      item_norm == "toilet_paper", "toilet paper",
                                   item_norm)
  )
purchases %>%
  left_join(prices_fixed, by = c("item" = "item_norm"))
```

```
# A tibble: 5 × 4
  customer_id item         item.y       price
  <chr>       <chr>        <chr>        <dbl>
1 c1          bread        BREAD          1
2 c1          milk         MILK           0.8
3 c1          bananas      <NA>          NA
4 c2          milk         MILK           0.8
5 c2          toilet paper TOILET_PAPER   3
```

Manual massaging: case_when

```
prices_fixed <- prices %>%
  mutate(
    item_norm = str_to_lower(item),
    item_norm = case_when(
      item_norm == "toilet_paper" ~ "toilet paper",
      TRUE                        ~ item_norm)
  )
purchases %>%
  left_join(prices_fixed, by = c("item" = "item_norm"))
```

```
# A tibble: 5 × 4
  customer_id item         item.y       price
  <chr>       <chr>        <chr>        <dbl>
1 c1          bread        BREAD          1
2 c1          milk         MILK           0.8
3 c1          bananas      <NA>          NA
4 c2          milk         MILK           0.8
5 c2          toilet paper TOILET_PAPER   3
```

Easy to add cases

```r
prices_fixed <- prices %>%
  mutate(
    item_norm = str_to_lower(item),
    item_norm = case_when(
      item_norm == "toilet_paper" ~ "toilet paper",
      item_norm == "banana"       ~ "bananas",
      TRUE                        ~ item_norm)
  )
purchases %>%
  left_join(prices_fixed, by = c("item" = "item_norm"))
```

```
# A tibble: 5 × 4
  customer_id item         item.y       price
  <chr>       <chr>        <chr>        <dbl>
1 c1          bread        BREAD         1
2 c1          milk         MILK          0.8
3 c1          bananas      BANANA        0.15
4 c2          milk         MILK          0.8
5 c2          toilet paper TOILET_PAPER  3
```

Aside: Could you use a `join` for this instead?

Think about this on your own.

`case_when`

`if-elif` version (Python):

```
if age < 0:
    return "invalid"
elif age < 18:
    return "child"
else:
    return "adult"
```

`case_when` version:

```
age <- 18
case_when(
  age < 0    ~ "invalid",
  age < 18   ~ "child",
  TRUE       ~ "adult"
)
```

```
[1] "adult"
```

- first to `True` wins in both versions
- `TRUE` corresponds to `else` (the default)

case_when vectorizes

Like many R functions, it actually applies to all elements of a vector.

```
age <- c(-1, 0, 17, 18) # a vector
case_when(
  age < 0    ~ "invalid",
  age < 18   ~ "child",
  TRUE       ~ "adult"
)
```

```
[1] "invalid" "child"   "child"   "adult"
```

case_when vs if_else

You can write the same thing either way. Which do you prefer?

if_else:

```
if_else(
  age < 0, "invald",
  if_else(
    age < 18, "child",
    "other"))
```

```
[1] "invald" "child"  "child"
```

case_when:

```
case_when(
  age < 0   ~ "invalid",
  age < 18  ~ "child",
  TRUE      ~ "adult"
)
```

```
[1] "invalid" "child"   "child'
```

case_when in a data frame

```
people <- tribble(
  ~name, ~age,
  "Allen Linford", -1,
  "Seb Dodds", 0,
  "Charleen Lockwood", 17,
  "Ridley Burgin", 18,
)
people %>% mutate(
  adult = case_when(
    age < 0   ~ "invalid",
    age < 18  ~ "child",
    TRUE      ~ "adult"
  )
)
```

```
# A tibble: 4 × 3
  name                age adult
  <chr>             <dbl> <chr>
1 Allen Linford        -1 invalid
2 Seb Dodds             0 child
3 Charleen Lockwood    17 child
4 Ridley Burgin        18 adult
```

The recoding pattern

```
population <- read_csv("../../data/worldbank_sp_pop_totl.csv")
population %>% mutate(
  country = case_when(
    country == "United States" ~ "USA",
    iso3c == "GBR"             ~ "UK",    # LHS conditions may us
    TRUE                       ~ country # so can RHS
  )
) %>% filter(str_starts(country, "U")) # Just to see the results
```

More `case_when` tricks

See `?case_when` for how to:

- Deal with inconsistent data types
- Efficiently encode complicated conditionals
- Reuse `case_when` expressions by making a function

and more!