Mutating Joins

Joe Nese

University of Oregon

Fall 2021

Share

`{tidylog}`

- Provides feedback about `{dplyr}` and `{tidyr}` operations

[view link]

Homework(s) Review

Mutating Joins

Week 5

Credit Daniel Anderson for slides

Agenda

- Quick Share
- `bind_rows()`
- `*_join()`

Overall Purpose

- Understand and be able to identify keys
- Understand different types of mutating joins
  - `left_join`, `right_join`
  - one-to-one, one-to-many
- Understand some ways joins fail

A bit about joins

- Also data "merge"
- Today we'll talk about **mutating** joins
- Mutating joins add columns to a dataset
- Mutating joins are the most common, but **filtering** joins can be very powerful

What if I want to add rows?

- Not technically a join (no key involved)

Binding rows

```
g3 <- tibble(sid = 1:3,
             grade = rep(3, 3),
             score = as.integer(rnorm(3, 200,10)))
g4 <- tibble(sid = 9:11,
             grade = rep(4, 3),
             score = as.integer(rnorm(3, 200,10)))
```

```
g3
```

```
## # A tibble: 3 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     1     3   184
## 2     2     3   203
## 3     3     3   212
```

```
g4
```

```
## # A tibble: 3 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     9     4   175
## 2    10     4   204
## 3    11     4   193
```

bind_rows()

- In examples like the previous data sets, we just want to combine the data by **stacking the rows**
- Data have same (or approximately same) columns
- We can do so with `bind_rows()`

```
bind_rows(g3, g4)
```

```
## # A tibble: 6 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     1     3   184
## 2     2     3   203
## 3     3     3   212
## 4     9     4   175
## 5    10     4   204
## 6    11     4   193
```

dplyr::bind_rows()

- an efficient way to bind many data frames into one, by stacking rows
  - can bind multiple datasets

```
one <- mtcars[1:4, ]
two <- mtcars[6:10, ]
three <- mtcars[12:14, ]
bind_rows(one, two, three)
```

- like joining (merging) data frames that have the same columns
- columns don't need to match when row-binding

Optional `.id` argument

- What if we knew the grade, but didn't have a variable in each dataset already?
- Use `.id` to add an index for each dataset

```
bind_rows(select(g3, -grade), select(g4, -grade), .id = "dataset")
```

```
## # A tibble: 6 x 3
##   dataset   sid score
##   <chr>   <int> <int>
## 1 1           1   184
## 2 1           2   203
## 3 1           3   212
## 4 2           9   175
## 5 2          10   204
## 6 2          11   193
```

Recode .id column

```
bind_rows(select(g3, -grade), select(g4, -grade), .id = "dataset") %>%
  mutate(grade = ifelse(dataset == 1, 3, 4))
```

```
## # A tibble: 6 x 4
##   dataset   sid score grade
##   <chr>   <int> <int> <dbl>
## 1 1           1   184     3
## 2 1           2   203     3
## 3 1           3   212     3
## 4 2           9   175     4
## 5 2          10   204     4
## 6 2          11   193     4
```

Even better

```
bind_rows(select(g3, -grade), select(g4, -grade), .id = "grade") %>%
  mutate(grade = ifelse(grade == 1, 3, 4))
```

```
## # A tibble: 6 x 3
##   grade   sid score
##   <dbl> <int> <int>
## 1     3     1   184
## 2     3     2   203
## 3     3     3   212
## 4     4     9   175
## 5     4    10   204
## 6     4    11   193
```

What if columns don't match exactly?

Pads with NA

```
bind_rows(g3, g4[ ,-2], .id = "dataset")
```

```
## # A tibble: 6 x 4
##   dataset   sid grade score
##   <chr>   <int> <dbl> <int>
## 1 1           1     3   184
## 2 1           2     3   203
## 3 1           3     3   212
## 4 2           9    NA   175
## 5 2          10    NA   204
## 6 2          11    NA   193
```

You can also `bind_cols()`

```r
read <- tibble(sid = 1:3,
               read = as.integer(rnorm(3, 200,10)))
math <- tibble(math = as.integer(rnorm(3, 200,10)))
```

```r
read
```

```
## # A tibble: 3 x 2
##     sid  read
##   <int> <int>
## 1     1   202
## 2     2   206
## 3     3   190
```

```r
math
```

```
## # A tibble: 3 x 1
##    math
##   <int>
## 1   202
## 2   202
## 3   204
```

bind_cols()

```
bind_cols(read, math)
```

```
## # A tibble: 3 x 3
##     sid  read  math
##   <int> <int> <int>
## 1     1   202   202
## 2     2   206   202
## 3     3   190   204
```

Joins

(not to be confused with row binding)

Keys

- Uniquely identify rows in a dataset
- Variable(s) in common between two datasets to be joined
- A key can be more than one variable

Types of keys

- Small distinction that you probably won't have to worry about much, but is worth mentioning:
    - **Primary keys**: uniquely identify observations in *their* dataset
    - **Foreign keys**: uniquely identify observations in *other* datasets

What's the primary key here?

First, let's break down the code:

```
library(rio)
library(here)
ecls <- import(here("data", "ecls-k_samp.sav"),
               setclass = "tbl_df") %>%
    characterize()
```

```
## # A tibble: 984 x 33
##    child_id teacher_id school_id k_type   school_type sex    ethnic   famty
##    <chr>    <chr>      <chr>     <chr>    <chr>       <chr>  <chr>    <chr>
##  1 0842021C 0842T02    0842      full-day public      male   BLACK O~ BIOLO
##  2 0905002C 0905T01    0905      full-day private     male   ASIAN    BIOLO
##  3 0150012C 0150T01    0150      full-day private     female BLACK O~ BIOLO
##  4 0556009C 0556T01    0556      full-day private     female HISPANI~ BIOLO
##  5 0089013C 0089T04    0089      full-day public      male   WHITE, ~ BIOLO
##  6 1217001C 1217T13    1217      half-day public      female NATIVE ~ BIOLO
##  7 1092008C 1092T01    1092      half-day public      female HISPANI~ BIOLO
##  8 0083007C 0083T16    0083      full-day public      male   WHITE, ~ BIOLO
##  9 1091005C 1091T02    1091      half-day private     male   WHITE, ~ BIOLO
## 10 2006006C 2006T01    2006      full-day private     male   WHITE, ~ BIOLO
## # ... with 974 more rows, and 21 more variables: T1RSCALE <dbl>, T1MSCALE <
```

Let's verify the key

```
ecls %>%
  count(child_id)
```

```
## # A tibble: 984 x 2
##    child_id     n
##    <chr>    <int>
##  1 0001010C     1
##  2 0002010C     1
##  3 0009005C     1
##  4 0009014C     1
##  5 0009026C     1
##  6 0013003C     1
##  7 0016004C     1
##  8 0016009C     1
##  9 0022005C     1
## 10 0022014C     1
## # ... with 974 more rows
```

Let's verify the key

```
ecls %>%
  count(child_id) %>%
  arrange(desc(n)) %>%
  slice(1:3)
```

```
## # A tibble: 3 x 2
##   child_id      n
##   <chr>     <int>
## 1 0001010C      1
## 2 0002010C      1
## 3 0009005C      1
```

OR

```
ecls %>%
  count(child_id) %>%
  filter(n > 1)
```

```
## # A tibble: 0 x 2
## # ... with 2 variables: child_id <chr>, n <int>
```

What about the key here?

```r
income_ineq <- read_csv(here("data", "incomeInequality_tidy.csv"))
head(income_ineq, n = 15)
```

```
## # A tibble: 15 x 6
##     Year Number.thousands realGDPperCap PopulationK percentile   income
##    <dbl>            <dbl>         <dbl>       <dbl>      <dbl>    <dbl>
##  1  1947            37237        14117.      144126       20      14243
##  2  1947            37237        14117.      144126       40      22984
##  3  1947            37237        14117.      144126       60      31166
##  4  1947            37237        14117.      144126       80      44223
##  5  1947            37237        14117.      144126       50      26764.
##  6  1947            37237        14117.      144126       90      41477
##  7  1947            37237        14117.      144126       95      54172
##  8  1947            37237        14117.      144126       99     134415
##  9  1947            37237        14117.      144126       99.5   203001
## 10  1947            37237        14117.      144126       99.9   479022
## 11  1947            37237        14117.      144126      100.   1584506
## 12  1948            38624        14452.      146631       20      13779
## 13  1948            38624        14452.      146631       40      22655
## 14  1948            38624        14452.      146631       60      30248
## 15  1948            38624        14452.      146631       80      42196
```

```r
income_ineq %>%
    count(Year, percentile) %>%
    filter(n > 1)
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: Year <dbl>, percentile <dbl>, n <int>
```

Sometimes there is no key

These tables have an *implicit* id - the row numbers. For example:

```
install.packages("nycflights13")
library(nycflights13)
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_ti
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <in
## 1  2013     1     1      517            515         2      830           81
## 2  2013     1     1      533            529         4      850           83
## 3  2013     1     1      542            540         2      923           85
## 4  2013     1     1      544            545        -1     1004          102
## 5  2013     1     1      554            600        -6      812           83
## 6  2013     1     1      554            558        -4      740           72
## # ... with 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_t
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
flights %>%
  count(year, month, day, flight, tailnum) %>%
  filter(n > 1)
```

```
## # A tibble: 11 x 6
##     year month   day flight tailnum     n
##    <int> <int> <int>  <int> <chr>   <int>
##  1  2013     2     9    303 <NA>        2
##  2  2013     2     9    655 <NA>        2
##  3  2013     2     9   1623 <NA>        2
##  4  2013     6     8   2269 N487WN      2
##  5  2013     6    15   2269 N230WN      2
##  6  2013     6    22   2269 N440LV      2
##  7  2013     6    29   2269 N707SA      2
##  8  2013     7     6   2269 N259WN      2
##  9  2013     8     3   2269 N446WN      2
## 10  2013     8    10   2269 N478WN      2
## 11  2013    12    15    398 <NA>        2
```

Create a key

If there is no key, it's often helpful to add one

These are called *surrogate* keys

```
flights2 <- flights %>%
  rowid_to_column()

flights2 %>%
  select(1:3, ncol(flights))
```

```
## # A tibble: 336,776 x 4
##    rowid  year month minute
##    <int> <int> <int>  <dbl>
##  1     1  2013     1     15
##  2     2  2013     1     29
##  3     3  2013     1     40
##  4     4  2013     1     45
##  5     5  2013     1      0
##  6     6  2013     1     58
##  7     7  2013     1      0
##  8     8  2013     1      0
##  9     9  2013     1      0
```

Mutating joins

Mutating `*_joins()`

- In `{tidyverse}`, we use `mutate()` to create new variables within a dataset
- A mutating join works similarly, in that we're adding new variables to the existing dataset through a join
- **Join**: Two tables of data joined by a common key

Four types of joins

- `left_join`: keep all the data in the **left** dataset, drop any non-matching cases from the right dataset

- `right_join`: keep all the data in the **right** dataset, drop any non-matching cases from the left dataset

- `inner_join`: keep **only** data that matches in **both** datasets

- `full_join`: keep **all** the data in **both** datasets (also sometimes referred to as an *outer* join)

Four types of joins

Mutating joins

- `left_join`: keep all the data in the **left** dataset, drop any non-matching cases from the right dataset

- `right_join`: keep all the data in the **right** dataset, drop any non-matching cases from the left dataset

Filtering joins

- `inner_join`: keep **only** data that matches in **both** datasets

- `full_join`: keep **all** the data in **both** datasets (also sometimes referred to as an *outer* join)

Four types of joins

Mutating joins

- `left_join`: keep all the data in the **left** dataset, drop any non-matching cases from the right dataset
- `right_join`: keep all the data in the **right** dataset, drop any non-matching cases from the left dataset

Filtering joins

- `inner_join`: keep **only** data that matches in **both** datasets
- `full_join`: keep **all** the data in **both** datasets (also sometimes referred to as an *outer* join)

Using joins to recode

Say you have a dataset like this

```r
set.seed(1)
disab_codes <- c("00", "10", "20", "40", "43", "50", "60",
                 "70", "74", "80", "82", "90", "96", "98")
dis_tbl <- tibble(
  sid = 1:200,
  dis_code = sample(disab_codes, 200, replace = TRUE),
  score = as.integer(rnorm(200, 200, 10))
  )
head(dis_tbl)
```

```
## # A tibble: 6 x 3
##     sid dis_code score
##   <int> <chr>    <int>
## 1     1 74         190
## 2     2 40         200
## 3     3 60         200
## 4     4 00         183
## 5     5 10         210
## 6     6 96         188
```

Codes

Code	Disability
0	'Not Applicable'
10	'Intellectual Disability'
20	'Hearing Impairment'
40	'Visual Impairment'
43	'Deaf-Blindness'
50	'Communication Disorder'
60	'Emotional Disturbance'
70	'Orthopedic Impairment'
74	'Traumatic Brain Injury'
80	'Other Health Impairments'
82	'Autism Spectrum Disorder'
90	'Specific Learning Disability'
96	'Developmental Delay 0-2yr'
98	'Developmental Delay 3-4yr'

Recode method

Using `case_when()`

```
dis_tbl %>%
  mutate(disability = case_when(
    dis_code == "10" ~ "Intellectual Disability",
    dis_code == "20" ~ 'Hearing Impairment',
    ...,
    TRUE ~ "Not Applicable"
    )
  )
```

Join method

```
dis_code_tbl <- tibble(
  dis_code = c(
    "00", "10", "20", "40", "43", "50", "60",
    "70", "74", "80", "82", "90", "96", "98"
    ),
  disability = c(
    'Not Applicable', 'Intellectual Disability',
    'Hearing Impairment', 'Visual Impairment',
    'Deaf-Blindness', 'Communication Disorder',
    'Emotional Disturbance', 'Orthopedic Impairment',
    'Traumatic Brain Injury', 'Other Health Impairments',
    'Autism Spectrum Disorder', 'Specific Learning Disability',
    'Developmental Delay 0-2yr', 'Developmental Delay 3-4yr'
    )
  )
```

```r
dis_code_tbl
```

```
## # A tibble: 14 x 2
##    dis_code disability
##    <chr>    <chr>
##  1 00       Not Applicable
##  2 10       Intellectual Disability
##  3 20       Hearing Impairment
##  4 40       Visual Impairment
##  5 43       Deaf-Blindness
##  6 50       Communication Disorder
##  7 60       Emotional Disturbance
##  8 70       Orthopedic Impairment
##  9 74       Traumatic Brain Injury
## 10 80       Other Health Impairments
## 11 82       Autism Spectrum Disorder
## 12 90       Specific Learning Disability
## 13 96       Developmental Delay 0-2yr
## 14 98       Developmental Delay 3-4yr
```

Join the tables

```
left_join(dis_tbl, dis_code_tbl)
```

```
## Joining, by = "dis_code"

## # A tibble: 200 x 4
##      sid dis_code score disability
##    <int> <chr>    <int> <chr>
##  1     1 74         190 Traumatic Brain Injury
##  2     2 40         200 Visual Impairment
##  3     3 60         200 Emotional Disturbance
##  4     4 00         183 Not Applicable
##  5     5 10         210 Intellectual Disability
##  6     6 96         188 Developmental Delay 0-2yr
##  7     7 60         203 Emotional Disturbance
##  8     8 82         204 Autism Spectrum Disorder
##  9     9 98         201 Developmental Delay 3-4yr
## 10    10 10         198 Intellectual Disability
## # ... with 190 more rows
```

Imperfect key match?

Consider the following

```
frl <- tibble(key = 1:3, frl = rbinom(3, 1, .5))

sped <- tibble(key = c(1, 2, 4), sped = rbinom(3, 1, .5))
```

```
frl
```

```
## # A tibble: 3 x 2
##     key   frl
##   <int> <int>
## 1     1     0
## 2     2     1
## 3     3     0
```

```
sped
```

```
## # A tibble: 3 x 2
##     key  sped
##   <dbl> <int>
## 1     1     0
## 2     2     1
## 3     4     0
```

Consider the following

`left_join()`?

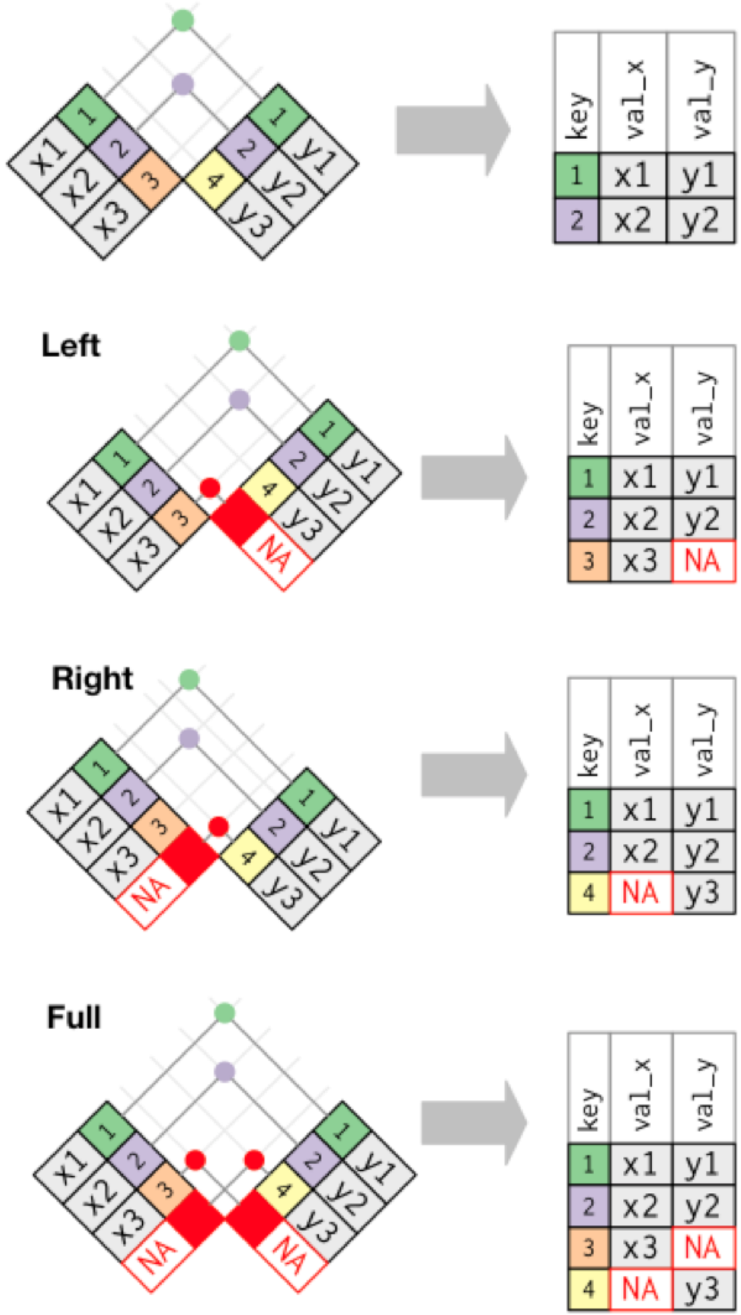
```
left_join(frl, sped)
```

```
## # A tibble: 3 x 3
##     key   frl  sped
##   <dbl> <int> <int>
## 1     1     0     0
## 2     2     1     1
## 3     3     0    NA
```

`right_join()`?

```
right_join(frl, sped)
```

```
## # A tibble: 3 x 3
##     key   frl  sped
##   <dbl> <int> <int>
## 1     1     0     0
## 2     2     1     1
## 3     4    NA     0
```

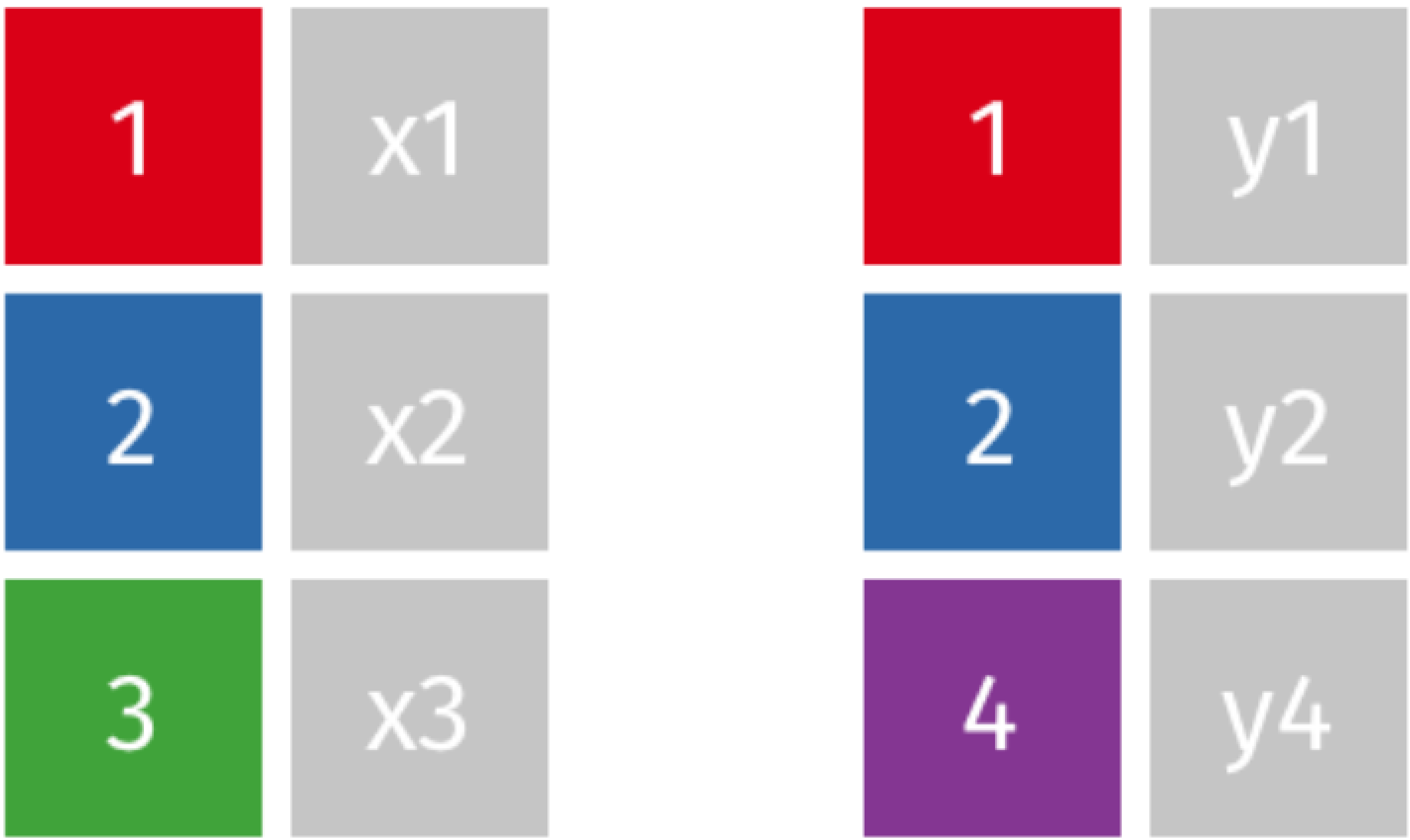


From r4ds

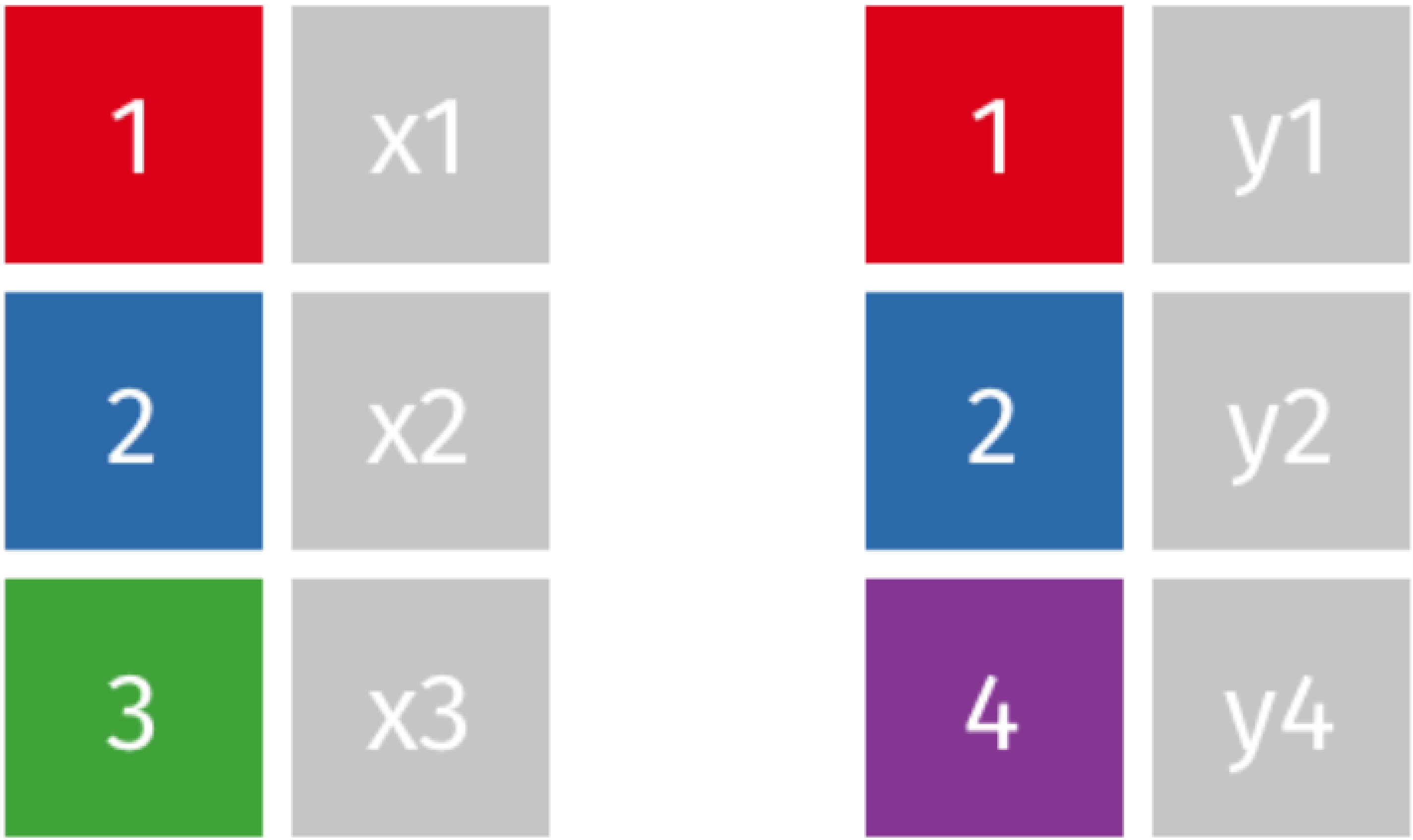
Animations

All of the following animations were created by Garrick Aden-Buie and can be found here

```
left_join(x, y)
```

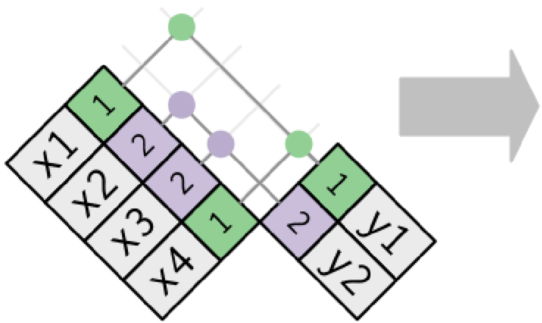


`right_join(x, y)`



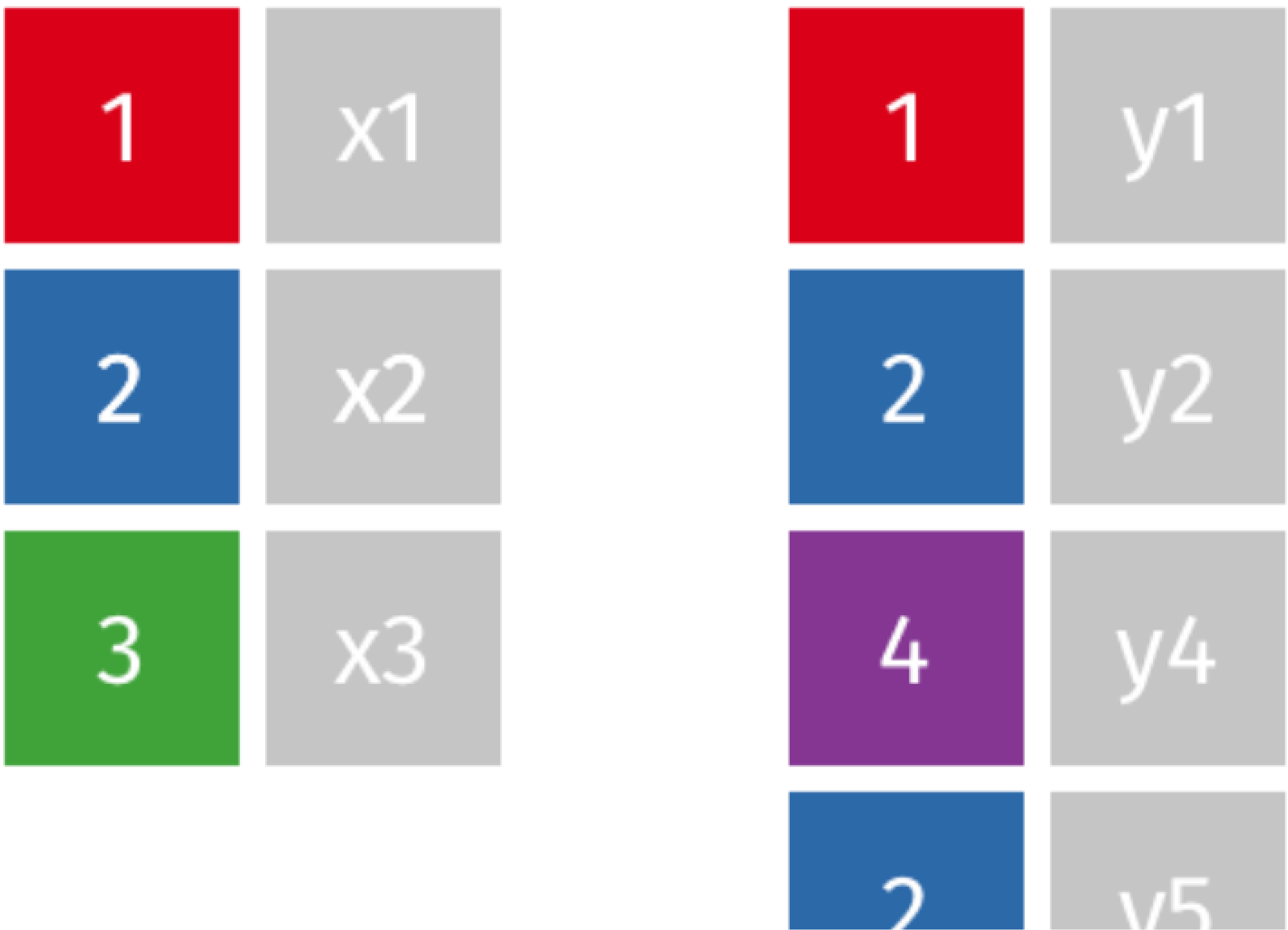
What if the key is not unique?

- Not an issue, as long as they are unique in one of the tables
    - In this case, it's called a one-to-many join



val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

`left_join(x, y)`



Example

Student-level data

```
(stu <- tibble(
  sid = 1:9,
  scid = c(1, 1, 1, 1, 2, 2, 3, 3
  score = c(10, 12, 15, 8,  9, 11
```

```
## # A tibble: 9 x 3
##     sid  scid score
##   <int> <dbl> <dbl>
## 1     1     1    10
## 2     2     1    12
## 3     3     1    15
## 4     4     1     8
## 5     5     2     9
## 6     6     2    11
## 7     7     3    12
## 8     8     3    15
```

School-level data

```
(schl <- tibble(
  scid = 1:3,
  stu_tch_ratio = c(22.05, 31.14,
  per_pupil_spending = c(15741.08
  )
)
```

```
## # A tibble: 3 x 3
##    scid stu_tch_ratio per_pupil_spend
##   <int>         <dbl>              <d
## 1     1          22.0              157
## 2     2          31.1              117
## 3     3          24.9              130
```

One to many

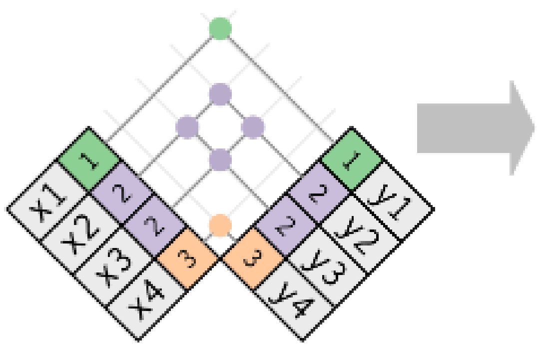
```
left_join(stu, schl)
```

```
## # A tibble: 9 x 5
##     sid  scid score stu_tch_ratio per_pupil_spending
##   <int> <dbl> <dbl>         <dbl>              <dbl>
## 1     1     1    10          22.0             15741.
## 2     2     1    12          22.0             15741.
## 3     3     1    15          22.0             15741.
## 4     4     1     8          22.0             15741.
## 5     5     2     9          31.1             11732.
## 6     6     2    11          31.1             11732.
## 7     7     3    12          24.9             13028.
## 8     8     3    15          24.9             13028.
## 9     9     3    17          24.9             13028.
```

What if key is not unique to either table?

Generally this is an error

Result is probably not going to be what you want



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Example

```
seasonal_means <- tibble(
  scid = rep(1:3, each = 3),
  season = rep(c("fall", "winter", "spring"), 3),
  mean = rnorm(3*3)
)
seasonal_means
```

```
## # A tibble: 9 x 3
##    scid season   mean
##   <int> <chr>   <dbl>
## 1     1 fall    0.345
## 2     1 winter  1.54
## 3     1 spring -0.330
## 4     2 fall    0.948
## 5     2 winter -0.479
## 6     2 spring -1.51
## 7     3 fall    0.435
## 8     3 winter -0.520
## 9     3 spring -0.835
```

```r
left_join(stu, seasonal_means)
```

```
## # A tibble: 27 x 5
##      sid  scid score season   mean
##    <int> <dbl> <dbl> <chr>   <dbl>
##  1     1     1    10 fall    0.345
##  2     1     1    10 winter  1.54
##  3     1     1    10 spring -0.330
##  4     2     1    12 fall    0.345
##  5     2     1    12 winter  1.54
##  6     2     1    12 spring -0.330
##  7     3     1    15 fall    0.345
##  8     3     1    15 winter  1.54
##  9     3     1    15 spring -0.330
## 10     4     1     8 fall    0.345
## # ... with 17 more rows
```

How do we fix this?

In some cases, the solution is obvious, in others it is not

But **you must have at least one unique key** to join the datasets

In this case

Move the dataset to wide before joining

Move to wide

```
seasonal_means_wide <- seasonal_means %>%
  pivot_wider(names_from = "season",
              values_from = "mean")
seasonal_means_wide
```

```
## # A tibble: 3 x 4
##    scid  fall winter spring
##   <int> <dbl>  <dbl>  <dbl>
## 1     1 0.345  1.54  -0.330
## 2     2 0.948 -0.479 -1.51
## 3     3 0.435 -0.520 -0.835
```

We will cover this in Week 8

Join

One to many join

```
left_join(stu, seasonal_means_wide)
```

```
## # A tibble: 9 x 6
##     sid  scid score  fall winter spring
##   <int> <dbl> <dbl> <dbl>  <dbl>  <dbl>
## 1     1     1    10 0.345  1.54  -0.330
## 2     2     1    12 0.345  1.54  -0.330
## 3     3     1    15 0.345  1.54  -0.330
## 4     4     1     8 0.345  1.54  -0.330
## 5     5     2     9 0.948 -0.479 -1.51
## 6     6     2    11 0.948 -0.479 -1.51
## 7     7     3    12 0.435 -0.520 -0.835
## 8     8     3    15 0.435 -0.520 -0.835
## 9     9     3    17 0.435 -0.520 -0.835
```

Another example

- Often you want to add summary info to your dataset
- You can do this easily with by piping arguments

ECLS-K reminder

```
ecls
```

```
## # A tibble: 984 x 33
##    child_id teacher_id school_id k_type   school_type sex    ethnic   famty
##    <chr>    <chr>      <chr>     <chr>    <chr>       <chr>  <chr>    <chr>
##  1 0842021C 0842T02    0842      full-day public      male   BLACK O~ BIOLO
##  2 0905002C 0905T01    0905      full-day private     male   ASIAN    BIOLO
##  3 0150012C 0150T01    0150      full-day private     female BLACK O~ BIOLO
##  4 0556009C 0556T01    0556      full-day private     female HISPANI~ BIOLO
##  5 0089013C 0089T04    0089      full-day public      male   WHITE, ~ BIOLO
##  6 1217001C 1217T13    1217      half-day public      female NATIVE ~ BIOLO
##  7 1092008C 1092T01    1092      half-day public      female HISPANI~ BIOLO
##  8 0083007C 0083T16    0083      full-day public      male   WHITE, ~ BIOLO
##  9 1091005C 1091T02    1091      half-day private     male   WHITE, ~ BIOLO
## 10 2006006C 2006T01    2006      full-day private     male   WHITE, ~ BIOLO
```

Compute group means

```
ecls %>%
  group_by(school_id) %>%
  summarize(sch_pre_math = mean(T1MSCALE))
```

```
## # A tibble: 515 x 2
##    school_id sch_pre_math
##    <chr>            <dbl>
##  1 0001              20.5
##  2 0002              15.0
##  3 0009              18.8
##  4 0013              42.3
##  5 0016              17.6
##  6 0022              17.8
##  7 0023              15.5
##  8 0025              19.4
##  9 0026              16.9
## 10 0028              14.4
## # ... with 505 more rows
```

Join right within pipeline

```
ecls %>%
  group_by(school_id) %>%
  summarize(sch_pre_math = mean(T1MSCALE)) %>%
  left_join(ecls) %>%
  select(school_id:k_type) # Just for space
```

```
## # A tibble: 984 x 5
##    school_id sch_pre_math child_id teacher_id k_type
##    <chr>            <dbl> <chr>    <chr>      <chr>
##  1 0001              20.5 0001010C 0001T01    full-day
##  2 0002              15.0 0002010C 0002T01    half-day
##  3 0009              18.8 0009026C 0009T01    half-day
##  4 0009              18.8 0009014C 0009T02    half-day
##  5 0009              18.8 0009005C 0009T01    half-day
##  6 0013              42.3 0013003C 0013T01    full-day
##  7 0016              17.6 0016004C 0016T01    half-day
##  8 0016              17.6 0016009C 0016T01    half-day
##  9 0022              17.8 0022005C 0022T01    half-day
## 10 0022              17.8 0022014C 0022T03    half-day
## # ... with 974 more rows
```

Default join behavior

By default, the `_join` functions will use all columns with common names as keys

```
flights2 <- flights %>%
  select(year:day, hour, origin, dest, tailnum, carrier)
flights2[1:2, ]
```

```
## # A tibble: 2 x 8
##    year month   day  hour origin dest  tailnum carrier
##   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>
## 1  2013     1     1     5 EWR    IAH   N14228  UA
## 2  2013     1     1     5 LGA    IAH   N24211  UA
```

```
weather[1:2, ]
```

```
## # A tibble: 2 x 15
##   origin  year month   day  hour  temp  dewp humid wind_dir wind_speed wind_
##   <chr>  <int> <int> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
## 1 EWR     2013     1     1     1  39.0  26.1  59.4      270      10.4
## 2 EWR     2013     1     1     2  39.0  27.0  61.6      250       8.06
## # ... with 1 more variable: time_hour <dttm>
```

```r
left_join(flights2, weather)
```

```
## Joining, by = c("year", "month", "day", "hour", "origin")

## # A tibble: 336,776 x 18
##     year month   day  hour origin dest  tailnum carrier  temp  dewp humid w
##    <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl> <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA       39.0  28.0  64.4
##  2  2013     1     1     5 LGA    IAH   N24211  UA       39.9  25.0  54.8
##  3  2013     1     1     5 JFK    MIA   N619AA  AA       39.0  27.0  61.6
##  4  2013     1     1     5 JFK    BQN   N804JB  B6       39.0  27.0  61.6
##  5  2013     1     1     6 LGA    ATL   N668DN  DL       39.9  25.0  54.8
##  6  2013     1     1     5 EWR    ORD   N39463  UA       39.0  28.0  64.4
##  7  2013     1     1     6 EWR    FLL   N516JB  B6       37.9  28.0  67.2
##  8  2013     1     1     6 LGA    IAD   N829AS  EV       39.9  25.0  54.8
##  9  2013     1     1     6 JFK    MCO   N593JB  B6       37.9  27.0  64.3
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA       39.9  25.0  54.8
## # ... with 336,766 more rows, and 4 more variables: precip <dbl>, pressure
## #   time_hour <dttm>
```

Use only certain keys?

If we were joining *flights2* and *planes*, we would not want to use the `year` variable in the join, because **it means different things in each dataset**

```
head(planes)
```

```
## # A tibble: 6 x 9
##   tailnum  year type                    manufacturer     model     engines 
##   <chr>   <int> <chr>                   <chr>            <chr>       <int> 
## 1 N10156   2004 Fixed wing multi engine EMBRAER          EMB-145XR       2
## 2 N102UW   1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214        2
## 3 N103US   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214        2
## 4 N104UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214        2
## 5 N10575   2002 Fixed wing multi engine EMBRAER          EMB-145LR       2
## 6 N105UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214        2
```

Specify `*_join()` keys

Specify the variables with `by`

```
left_join(flights2, planes, by = "tailnum")
```

```
## # A tibble: 336,776 x 16
##    year.x month   day  hour origin dest  tailnum carrier year.y type   manu
##     <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>  <chr
##  1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixed~ BOEI
##  2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixed~ BOEI
##  3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixed~ BOEI
##  4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixed~ AIRB
##  5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixed~ BOEI
##  6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixed~ BOEI
##  7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixed~ AIRB
##  8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixed~ CANA
##  9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixed~ AIRB
## 10   2013     1     1     6 LGA    ORD   N3ALAA  AA          NA <NA>   <NA>
## # ... with 336,766 more rows, and 1 more variable: engine <chr>
```

Specify `*_join()` keys

I like to *always* specify the `by` vars

Makes intent explicit

Helps me review my own code

Mismatched key names

What if you had data to merge like this?

```
names(schl)[1] <- "school_id"
schl
```

```
## # A tibble: 3 x 3
##   school_id stu_tch_ratio per_pupil_spending
##       <int>         <dbl>              <dbl>
## 1         1          22.0             15741.
## 2         2          31.1             11732.
## 3         3          24.9             13028.
```

```
stu
```

```
## # A tibble: 9 x 3
##     sid  scid score
##   <int> <dbl> <dbl>
## 1     1     1    10
## 2     2     1    12
## 3     3     1    15
## 4     4     1     8
```

Join with mismatched key names

```
left_join(stu, schl, by = c("scid" = "school_id"))
```

```
## # A tibble: 9 x 5
##     sid  scid score stu_tch_ratio per_pupil_spending
##   <int> <dbl> <dbl>         <dbl>              <dbl>
## 1     1     1    10          22.0             15741.
## 2     2     1    12          22.0             15741.
## 3     3     1    15          22.0             15741.
## 4     4     1     8          22.0             15741.
## 5     5     2     9          31.1             11732.
## 6     6     2    11          31.1             11732.
## 7     7     3    12          24.9             13028.
## 8     8     3    15          24.9             13028.
## 9     9     3    17          24.9             13028.
```

Next time

Before next class

- Homework
  - **Homework 5**
- Reading
  - R4DS 29
- Complete
  - Markdown Tutorial

Homework 5