Joins

Daniel Anderson
Week 3, Class 2

UNIVERSITY OF OREGON

Learning Objectives

- Understand and be able to identify keys
- Understand different types of joins
  - left, right, inner, full
  - one-to-one, one-to-many
- Understand common ways joins fail
- Understand the difference between mutating and filtering joins

Before we get started

- Today we'll talk about both mutating and filtering joins
- Mutating joins are more common, but filtering joins can be highly powerful
- Mutating joins add columns to a dataset

Before we get started

- Today we'll talk about both mutating and filtering joins
- Mutating joins are more common, but filtering joins can be highly powerful
- Mutating joins add columns to a dataset

What if I want to add rows?

- Not technically a join (no key involved, which we'll talk about momentarily)

Quick example, 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   205
## 2     2     3   197
## 3     3     3   207
```

```
g4
```

```
## # A tibble: 3 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     9     4   187
## 2    10     4   200
## 3    11     4   184
```

bind_rows

- In examples like the previous datasets, we just want to "staple" the rows together.

- We can do so with `bind_rows`.

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

```
## # A tibble: 6 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     1     3   205
## 2     2     3   197
## 3     3     3   207
## 4     9     4   187
## 5    10     4   200
## 6    11     4   184
```

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

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

```
## # A tibble: 6 x 3
##   dataset   sid score
##   <chr>   <int> <int>
## 1 1           1   205
## 2 1           2   197
## 3 1           3   207
## 4 2           9   187
## 5 2          10   200
## 6 2          11   184
```

```r
bind_rows(g3[ ,-2], g4[ ,-2], .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   205     3
## 2 1           2   197     3
## 3 1           3   207     3
## 4 2           9   187     4
## 5 2          10   200     4
## 6 2          11   184     4
```

Even better usage

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

```
## # A tibble: 6 x 3
##   grade   sid score
##   <chr> <int> <int>
## 1 g3        1   205
## 2 g3        2   197
## 3 g3        3   207
## 4 g4        9   187
## 5 g4       10   200
## 6 g4       11   184
```

What if columns don't match exactly?

- Pad 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   205
## 2 1           2     3   197
## 3 1           3     3   207
## 4 2           9    NA   187
## 5 2          10    NA   200
## 6 2          11    NA   184
```

Last note - read in a bunch of files

- We'll talk about this a lot more in the next course
- `purrr::map_df` uses `bind_rows` in the background

Last note - read in a bunch of files

- We'll talk about this a lot more in the next course
- `purrr::map_df` uses `bind_rows` in the background

```
dir.create("tmp")

mtcars %>%
  split(.$cyl) %>%
  walk2(c("tmp/cyl4.csv", "tmp/cyl6.csv", "tmp/cyl8.csv"),
        write_csv)

list.files("tmp")
```

```
## [1] "cyl4.csv" "cyl6.csv" "cyl8.csv"
```

Read in files

- Use `purrr::map_df` with the file names
- Note `fs::dir_ls` is equivalent to `list.files`, but plays nicer with `purrr::map_df`

```
new_mtcars <- map_df(fs::dir_ls("tmp"), rio::import, setclass = "tbl_df",
                     .id = "file")
new_mtcars %>%
  select(file, mpg, cyl) %>%
  slice(1:3)
```

```
## # A tibble: 3 x 3
##   file            mpg   cyl
##   <chr>         <dbl> <int>
## 1 tmp/cyl4.csv   22.8     4
## 2 tmp/cyl4.csv   24.4     4
## 3 tmp/cyl4.csv   22.8     4
```

```
unlink("tmp", recursive = TRUE)
```

Joins

(not to be confused with row binding)

Keys

- Uniquely identify rows in a dataset

Keys

- Uniquely identify rows in a dataset
- Variable(s) in common between two datasets to be joined

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

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?

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

```
## # A tibble: 984 x 33
##    child_id teacher_id school_id k_type school_type sex   ethnic famtype
##    <chr>    <chr>      <chr>     <chr>  <chr>       <chr> <chr>  <chr>
##  1 0842021C 0842T02    0842      full-… public      male  BLACK… BIOLOG…
##  2 0905002C 0905T01    0905      full-… private     male  ASIAN  BIOLOG…
##  3 0150012C 0150T01    0150      full-… private     fema… BLACK… BIOLOG…
##  4 0556009C 0556T01    0556      full-… private     fema… HISPA… BIOLOG…
##  5 0089013C 0089T04    0089      full-… public      male  WHITE… BIOLOG…
##  6 1217001C 1217T13    1217      half-… public      fema… NATIV… BIOLOG…
##  7 1092008C 1092T01    1092      half-… public      fema… HISPA… BIOLOG…
##  8 0083007C 0083T16    0083      full-… public      male  WHITE… BIOLOG…
##  9 1091005C 1091T02    1091      half-… private     male  WHITE… BIOLOG…
## 10 2006006C 2006T01    2006      full-… private     male  WHITE… BIOLOG…
## # … with 974 more rows, and 25 more variables: numsibs <dbl>,
## #   SES_cont <dbl>, SES_cat <chr>, age <dbl>, T1RSCALE <dbl>,
## #   T1MSCALE <dbl>, T1GSCALE <dbl>, T2RSCALE <dbl>, T2MSCALE <dbl>,
## #   T2GSCALE <dbl>, IRTreadgain <dbl>, IRTmathgain <dbl>, IRTgkgain <dbl>,
## #   T1ARSLIT <dbl>, T1ARSMAT <dbl>, T1ARSGEN <dbl>, T2ARSLIT <dbl>,
```

Double-checking

```
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
```

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

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

What about here?

```r
income_ineq <- import(here("data", "incomeInequality_tidy.csv"),
                      setclass = "tbl_df")
income_ineq
```

```
## # A tibble: 726 x 6
##     Year Number.thousands realGDPperCap PopulationK percentile    income
##    <int>            <int>         <dbl>       <int>      <dbl>     <dbl>
##  1  1947            37237      14117.32      144126         20    14243
##  2  1947            37237      14117.32      144126         40    22984
##  3  1947            37237      14117.32      144126         60    31166
##  4  1947            37237      14117.32      144126         80    44223
##  5  1947            37237      14117.32      144126         50    26764.14
##  6  1947            37237      14117.32      144126         90    41477
##  7  1947            37237      14117.32      144126         95    54172
##  8  1947            37237      14117.32      144126         99   134415
##  9  1947            37237      14117.32      144126         99.5 203001
## 10  1947            37237      14117.32      144126         99.9 479022
## # … with 716 more rows
```

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

```
## # A tibble: 0 x 3
## # … with 3 variables: Year <int>, 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)
```

```
flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # … with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
```

```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.

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

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

```
## # A tibble: 336,776 x 4
##    rowid  year month time_hour
##    <int> <int> <int> <dttm>
##  1     1  2013     1 2013-01-01 05:00:00
##  2     2  2013     1 2013-01-01 05:00:00
##  3     3  2013     1 2013-01-01 05:00:00
##  4     4  2013     1 2013-01-01 05:00:00
##  5     5  2013     1 2013-01-01 06:00:00
##  6     6  2013     1 2013-01-01 05:00:00
##  7     7  2013     1 2013-01-01 06:00:00
##  8     8  2013     1 2013-01-01 06:00:00
##  9     9  2013     1 2013-01-01 06:00:00
## 10    10  2013     1 2013-01-01 06:00:00
## # … with 336,766 more rows
```

Mutating joins

Mutating joins

- In *tidyverse*, we use `mutate()` to create new variables within a dataset.

Mutating joins

- In *tidyverse*, we use `mutate()` to create new variables within a dataset.
- A mutating join works similarly, in that we're adding to variables to the existing dataset through a join.

Mutating joins

- In *tidyverse*, we use `mutate()` to create new variables within a dataset.
- A mutating join works similarly, in that we're adding to variables to the existing dataset through a 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. This is also sometimes referred to as an *outer* join.

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. This is also sometimes referred to as an *outer* join.

If the keys match exactly in the two tables (datasets), all of these will result in the **exact** same result.

Using joins to recode

Say you have a dataset like this

```
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 40         193
## 2     2 50         200
## 3     3 74         190
## 4     4 96         201
## 5     5 20         193
## 6     6 96         217
```

Codes

Code	Disability
00	'Not Applicable'
10	'Mental Retardation'
20	'Hearing Impairment'
40	'Visual Impairment'
43	'Deaf-Blindness'
50	'Communication Disorder'
60	'Emotional Disturbance'
70	'Orthopedic Impairment'
74	'Traumatic Brain Injury'

Code	Disability
80	'Other Health Impairments'
82	'Autism Spectrum Disorder'
90	'Specific Learning Disability'
96	'Developmental Delay 0-2yr'
98	'Developmental Delay 3-4yr'

One method

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

Joining 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', 'Mental Retardation'
                                      'Hearing Impairment', 'Visual Impairme
                                      'Deaf-Blindness', 'Communication Disor
                                      'Emotional Disturbance', 'Orthopedic I
                                      '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       Mental Retardation
##  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 40         193 Visual Impairment
##  2     2 50         200 Communication Disorder
##  3     3 74         190 Traumatic Brain Injury
##  4     4 96         201 Developmental Delay 0-2yr
##  5     5 20         193 Hearing Impairment
##  6     6 96         217 Developmental Delay 0-2yr
##  7     7 98         207 Developmental Delay 3-4yr
##  8     8 80         209 Other Health Impairments
##  9     9 74         203 Traumatic Brain Injury
## 10    10 00         216 Not Applicable
## # … with 190 more rows
```

What if the keys don't match perfectly?

Consider the following hypothetical datasets to be merged

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

```
gender
```

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

```
sped
```

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

What will happen with a left join?

What will happen with a left join?

```
left_join(gender, sped)
```

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

What about a right join?

What about a right join?

```r
right_join(gender, sped)
```

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

Inner join?

Inner join?

```
inner_join(gender, sped)
```

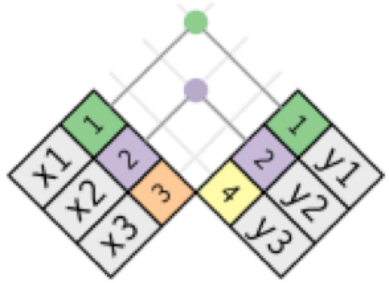
```
## # A tibble: 2 x 3
##     key  male  sped
##   <dbl> <int> <int>
## 1     1     1     1
## 2     2     1     1
```

Full join?

Full join?

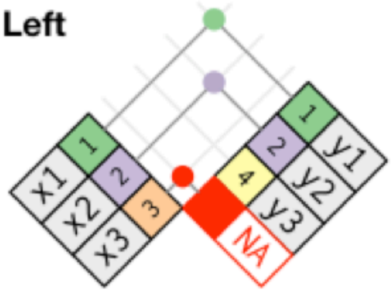
```
full_join(gender, sped)
```

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



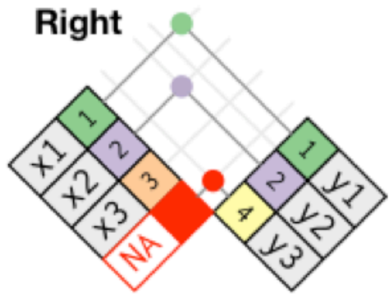
key	val_x	val_y
1	x1	y1
2	x2	y2

Left



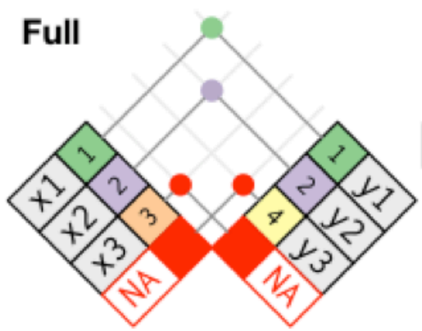
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



key	val_x	val_y
1	x1	y1
2	x2	y2
4	NA	y3

Full

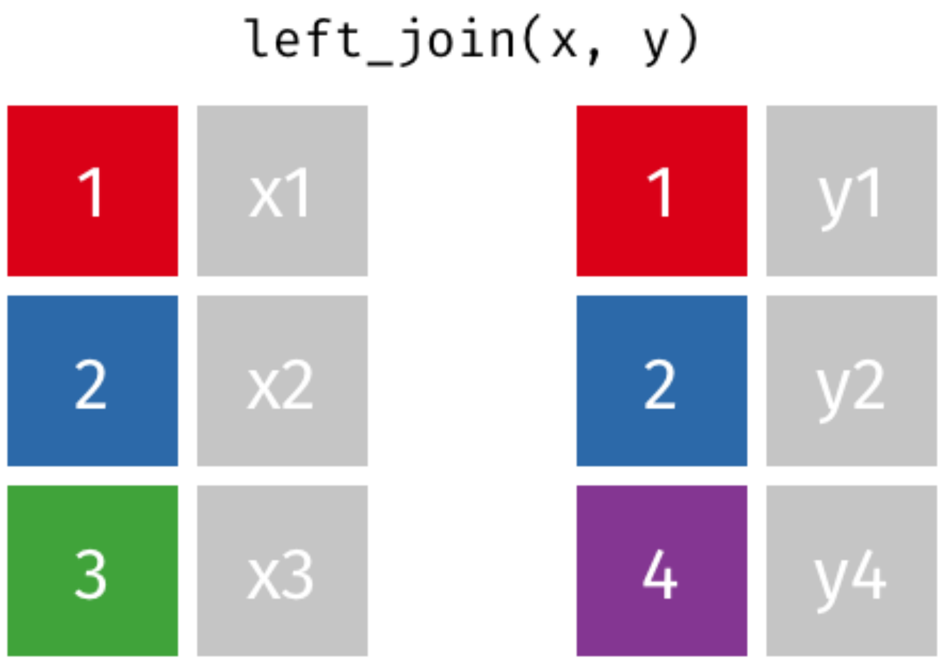


key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

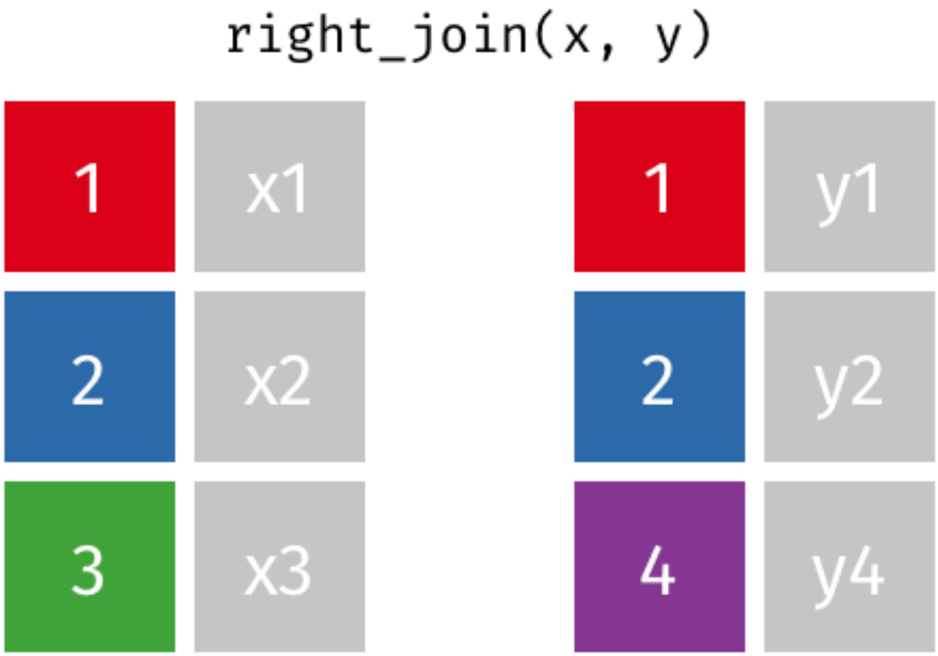
Animations

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

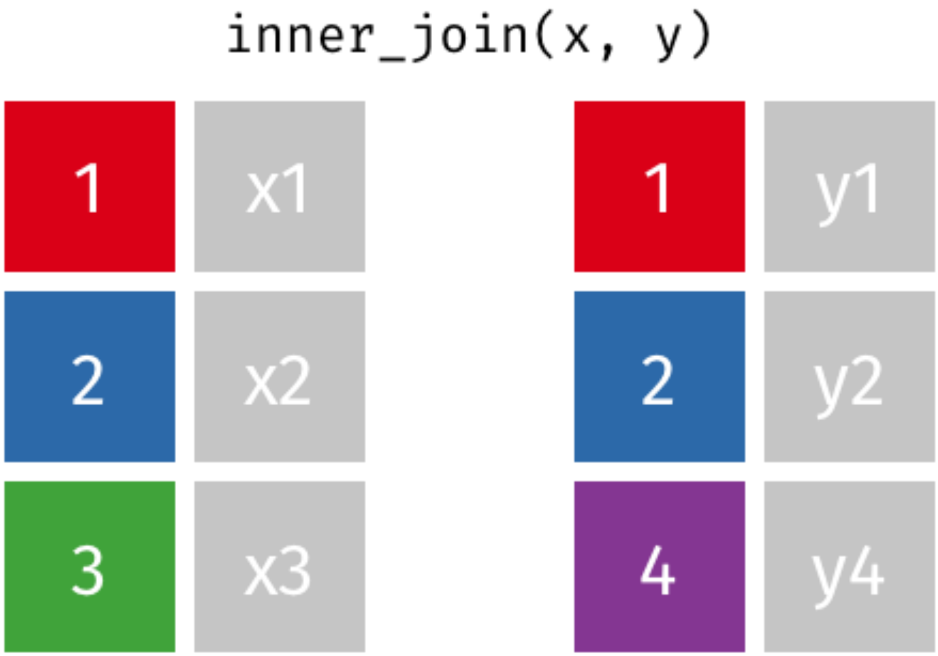
Animated `left_join`



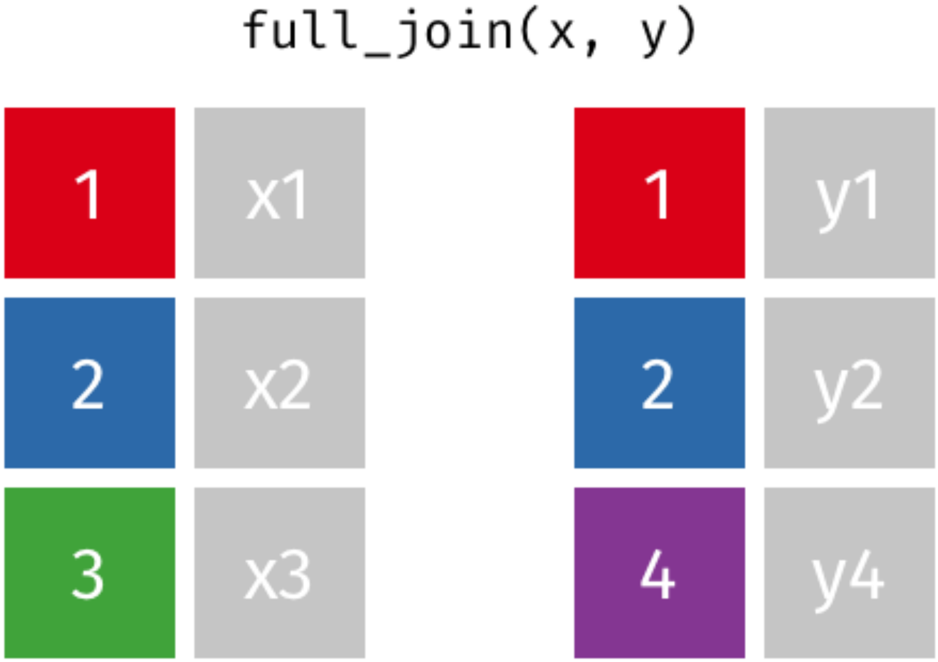
Animated `right_join`



Animated `inner_join`

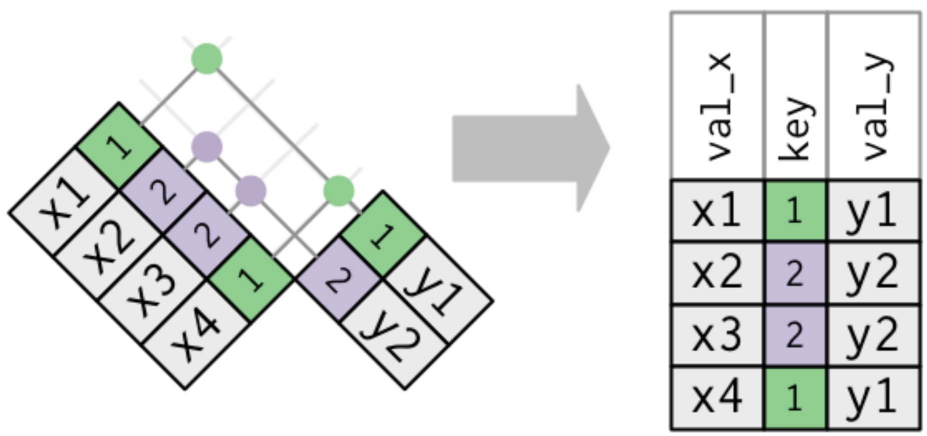


Animated `full_join`

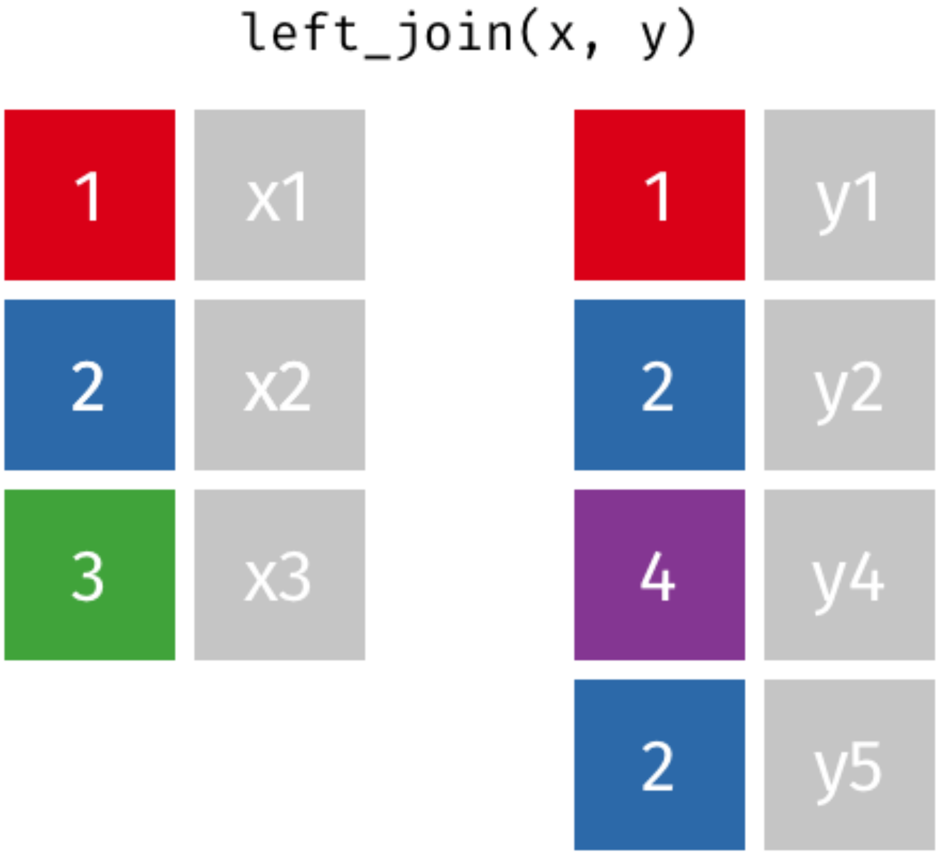


What if the key is not unique?

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



Animated one-to-many join



Example

```
stu <- tibble(
        sid = rep(1:3, each = 3),
        season = rep(c("f", "w", "s
        score = c(10, 12, 15,
                   8,  9, 11,
                  12, 15, 17)
        )
stu
```

```
## # A tibble: 9 x 3
##     sid season score
##   <int> <chr>  <dbl>
## 1     1 f         10
## 2     1 w         12
## 3     1 s         15
## 4     2 f          8
## 5     2 w          9
## 6     2 s         11
## 7     3 f         12
## 8     3 w         15
## 9     3 s         17
```

```
means <- stu %>%
    group_by(sid) %>%
    summarize(mean_score = mean(sco
means
```

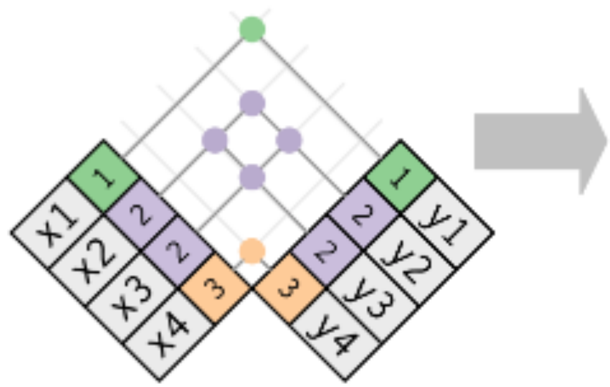
```
## # A tibble: 3 x 2
##     sid mean_score
##   <int>      <dbl>
## 1     1   12.33333
## 2     2    9.333333
## 3     3   14.66667
```

```r
left_join(stu, means)
```

```
## # A tibble: 9 x 4
##     sid season score mean_score
##   <int> <chr>  <dbl>      <dbl>
## 1     1 f         10  12.33333
## 2     1 w         12  12.33333
## 3     1 s         15  12.33333
## 4     2 f          8   9.333333
## 5     2 w          9   9.333333
## 6     2 s         11   9.333333
## 7     3 f         12  14.66667
## 8     3 w         15  14.66667
## 9     3 s         17  14.66667
```

What if key is not unique to either table?

- Generally this is an error
- Result is probably not going to be what you want (cartesian product).



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

Example

```r
dems <- tibble(sid = rep(1:3, each = 3),
               sped = c(rep("no", 6), rep("yes", 3)))
dems
```

```
## # A tibble: 9 x 2
##     sid sped
##   <int> <chr>
## 1     1 no
## 2     1 no
## 3     1 no
## 4     2 no
## 5     2 no
## 6     2 no
## 7     3 yes
## 8     3 yes
## 9     3 yes
```

```r
left_join(stu, dems)
```

```
## # A tibble: 27 x 4
##      sid season score sped
##    <int> <chr>  <dbl> <chr>
##  1     1 f         10 no
##  2     1 f         10 no
##  3     1 f         10 no
##  4     1 w         12 no
##  5     1 w         12 no
##  6     1 w         12 no
##  7     1 s         15 no
##  8     1 s         15 no
##  9     1 s         15 no
## 10     2 f          8 no
## # … with 17 more rows
```

How do we fix this?

In this case it's pretty simple: select for distinct cases in the demo file.

How do we fix this?

In this case it's pretty simple: select for distinct cases in the demo file.

In others it's not so straight forward. But the important thing to remember is that you need to work toward making sure at least one of the keys is unique.

```
dems <- dems %>%
    distinct(sid, .keep_all = TRUE)
dems
```

```
## # A tibble: 3 x 2
##     sid sped
##   <int> <chr>
## 1     1 no
## 2     2 no
## 3     3 yes
```

```r
left_join(stu, dems)
```

```
## # A tibble: 9 x 4
##     sid season score sped
##   <int> <chr>  <dbl> <chr>
## 1     1 f         10 no
## 2     1 w         12 no
## 3     1 s         15 no
## 4     2 f          8 no
## 5     2 w          9 no
## 6     2 s         11 no
## 7     3 f         12 yes
## 8     3 w         15 yes
## 9     3 s         17 yes
```

Another example

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

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

```r
ecls
```

```
## # A tibble: 984 x 34
##    school_id sch_pre_math child_id teacher_id k_type school_type sex
##    <chr>            <dbl> <chr>    <chr>      <chr>  <chr>       <chr>
##  1 0001         20.45800 0001010C 0001T01    full-… public      male
##  2 0002         14.977   0002010C 0002T01    half-… public      fema…
##  3 0009         18.82    0009026C 0009T01    half-… public      male
##  4 0009         18.82    0009014C 0009T02    half-… public      male
##  5 0009         18.82    0009005C 0009T01    half-… public      male
##  6 0013         42.321   0013003C 0013T01    full-… private     fema…
##  7 0016         17.55100 0016004C 0016T01    half-… public      male
##  8 0016         17.55100 0016009C 0016T01    half-… public      fema…
##  9 0022         17.8465  0022005C 0022T01    half-… public      male
## 10 0022         17.8465  0022014C 0022T03    half-… public      fema…
## # … with 974 more rows, and 27 more variables: ethnic <chr>,
## #   famtype <chr>, numsibs <dbl>, SES_cont <dbl>, SES_cat <chr>,
## #   age <dbl>, T1RSCALE <dbl>, T1MSCALE <dbl>, T1GSCALE <dbl>,
## #   T2RSCALE <dbl>, T2MSCALE <dbl>, T2GSCALE <dbl>, IRTreadgain <dbl>,
## #   IRTmathgain <dbl>, IRTgkgain <dbl>, T1ARSLIT <dbl>, T1ARSMAT <dbl>,
## #   T1ARSGEN <dbl>, T2ARSLIT <dbl>, T2ARSMAT <dbl>, T2ARSGEN <dbl>,
## #   ARSlitgain <dbl>, ARSmathgain <dbl>, ARSgkgain <dbl>,
## #   testdate1 <date>, testdate2 <date>, elapse <dbl>
```

Default behavior & changing it

- 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
##   <chr>  <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
## 1 EWR     2013     1     1     1 39.02 26.06 59.37      270   10.35702
## 2 EWR     2013     1     1     2 39.02 26.96 61.63      250    8.05546
## # … with 5 more variables: wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour <dttm>
```

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

```
## # A tibble: 336,776 x 18
##     year month   day  hour origin dest  tailnum carrier  temp  dewp
##    <dbl> <dbl> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl>
##  1  2013     1     1     5 EWR    IAH   N14228  UA      39.02 28.04
##  2  2013     1     1     5 LGA    IAH   N24211  UA      39.92 24.98
##  3  2013     1     1     5 JFK    MIA   N619AA  AA      39.02 26.96
##  4  2013     1     1     5 JFK    BQN   N804JB  B6      39.02 26.96
##  5  2013     1     1     6 LGA    ATL   N668DN  DL      39.92 24.98
##  6  2013     1     1     5 EWR    ORD   N39463  UA      39.02 28.04
##  7  2013     1     1     6 EWR    FLL   N516JB  B6      37.94 28.04
##  8  2013     1     1     6 LGA    IAD   N829AS  EV      39.92 24.98
##  9  2013     1     1     6 JFK    MCO   N593JB  B6      37.94 26.96
## 10  2013     1     1     6 LGA    ORD   N3ALAA  AA      39.92 24.98
## # … with 336,766 more rows, and 8 more variables: humid <dbl>,
## #   wind_dir <dbl>, wind_speed <dbl>, wind_gust <dbl>, precip <dbl>,
## #   pressure <dbl>, visib <dbl>, time_hour <dttm>
```

Use only some vars?

- 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 seats speed engine
##   <chr>   <int> <chr>       <chr>           <chr>    <int> <int> <int> <chr>
## 1 N10156   2004 Fixed win… EMBRAER          EMB-1…       2    55    NA Turbo…
## 2 N102UW   1998 Fixed win… AIRBUS INDUST… A320-…       2   182    NA Turbo…
## 3 N103US   1999 Fixed win… AIRBUS INDUST… A320-…       2   182    NA Turbo…
## 4 N104UW   1999 Fixed win… AIRBUS INDUST… A320-…       2   182    NA Turbo…
## 5 N10575   2002 Fixed win… EMBRAER          EMB-1…       2    55    NA Turbo…
## 6 N105UW   1999 Fixed win… AIRBUS INDUST… A320-…       2   182    NA Turbo…
```

How?

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 
##     <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>    <int> <chr>
##  1   2013     1     1     5 EWR    IAH   N14228  UA        1999 Fixe…
##  2   2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixe…
##  3   2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixe…
##  4   2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixe…
##  5   2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixe…
##  6   2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixe…
##  7   2013     1     1     6 EWR    FLL   N516JB  B6        2000 Fixe…
##  8   2013     1     1     6 LGA    IAD   N829AS  EV        1998 Fixe…
##  9   2013     1     1     6 JFK    MCO   N593JB  B6        2004 Fixe…
## 10   2013     1     1     6 LGA    ORD   N3ALAA  AA          NA <NA> 
## # … with 336,766 more rows, and 6 more variables: manufacturer <chr>,
## #   model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>
```

Mismatched names?

- What if you had data to merge like this?

```r
stu
```

```
## # A tibble: 9 x 3
##     sid season score
##   <int> <chr>  <dbl>
## 1     1 f         10
## 2     1 w         12
## 3     1 s         15
## 4     2 f          8
## 5     2 w          9
## 6     2 s         11
## 7     3 f         12
## 8     3 w         15
## 9     3 s         17
```

```r
names(dems)[1] <- "stu_id"
dems
```

```
## # A tibble: 3 x 2
##   stu_id sped
##    <int> <chr>
## 1      1 no
## 2      2 no
## 3      3 yes
```

Join w/mismatched names

```
left_join(stu, dems, by = c("sid" = "stu_id"))
```

```
## # A tibble: 9 x 4
##     sid season score sped
##   <int> <chr>  <dbl> <chr>
## 1     1 f         10 no
## 2     1 w         12 no
## 3     1 s         15 no
## 4     2 f          8 no
## 5     2 w          9 no
## 6     2 s         11 no
## 7     3 f         12 yes
## 8     3 w         15 yes
## 9     3 s         17 yes
```

filtering joins

Filtering joins

- `semi_join()` works just like `left_join` or `inner_join` but you don't actually add the variables.

- Let's filter classrooms with extremely high math pretest average scores.

First, calculate averages

```
av_pre_mth <- ecls %>%
    group_by(teacher_id, k_type) %>%
    summarize(av_pre_mth = mean(T1MSCALE))
av_pre_mth
```

```
## # A tibble: 707 x 3
## # Groups:   teacher_id [?]
##    teacher_id k_type   av_pre_mth
##    <chr>      <chr>         <dbl>
##  1 0001T01    full-day   20.45800
##  2 0002T01    half-day   14.977
##  3 0009T01    half-day   17.6475
##  4 0009T02    half-day   21.165
##  5 0013T01    full-day   42.321
##  6 0016T01    half-day   17.55100
##  7 0022T01    half-day   20.368
##  8 0022T03    half-day   15.325
##  9 0023T01    half-day   10.988
## 10 0023T04    half-day   20.02200
## # … with 697 more rows
```

Next, filter for means 3 standard deviations above the mean.

```
extr_high <- av_pre_mth %>%
    ungroup() %>%
    filter(av_pre_mth > (mean(av_pre_mth) + 3*sd(av_pre_mth)))
extr_high
```

```
## # A tibble: 8 x 3
##   teacher_id k_type   av_pre_mth
##   <chr>      <chr>         <dbl>
## 1 0013T01    full-day   42.321
## 2 0078T04    half-day   45.75
## 3 0162T02    half-day   42.318
## 4 0360T01    full-day   41.42200
## 5 0384T03    full-day   41.29
## 6 0663T01    full-day   42.8455
## 7 0944T03    half-day   45.371
## 8 1045T02    full-day   40.734
```

Finally, use `semi_join` to filter.

```
extr_high_ecls <- semi_join(ecls, extr_high)
extr_high_ecls
```

```
## # A tibble: 10 x 34
##    school_id sch_pre_math child_id teacher_id k_type school_type sex
##    <chr>            <dbl> <chr>    <chr>      <chr>  <chr>       <chr>
##  1 0013          42.321   0013003C 0013T01    full-… private     fema…
##  2 0078          25.64    0078020C 0078T04    half-… public      fema…
##  3 0162          30.52425 0162009C 0162T02    half-… public      fema…
##  4 0360          41.42200 0360014C 0360T01    full-… public      fema…
##  5 0384          30.4     0384014C 0384T03    full-… public      fema…
##  6 0663          42.8455  0663006C 0663T01    full-… private     male
##  7 0663          42.8455  0663012C 0663T01    full-… private     fema…
##  8 0944          45.371   0944017C 0944T03    half-… private     fema…
##  9 1045          35.45325 1045015C 1045T02    full-… private     male
## 10 1045          35.45325 1045020C 1045T02    full-… private     fema…
## # … with 27 more variables: ethnic <chr>, famtype <chr>, numsibs <dbl>,
## #   SES_cont <dbl>, SES_cat <chr>, age <dbl>, T1RSCALE <dbl>,
## #   T1MSCALE <dbl>, T1GSCALE <dbl>, T2RSCALE <dbl>, T2MSCALE <dbl>,
## #   T2GSCALE <dbl>, IRTreadgain <dbl>, IRTmathgain <dbl>, IRTgkgain <dbl>,
## #   T1ARSLIT <dbl>, T1ARSMAT <dbl>, T1ARSGEN <dbl>, T2ARSLIT <dbl>,
## #   T2ARSMAT <dbl>, T2ARSGEN <dbl>, ARSlitgain <dbl>, ARSmathgain <dbl>,
## #   ARSgkgain <dbl>, testdate1 <date>, testdate2 <date>, elapse <dbl>
```

Filtering joins

`anti_join()` does the opposite of `semi_join`, keeping any rows that do **not** match.

```
extr_low_ecls <- anti_join(ecls, extr_high)
extr_low_ecls
```

```
## # A tibble: 974 x 34
##    school_id sch_pre_math child_id teacher_id k_type school_type sex
##    <chr>            <dbl> <chr>    <chr>      <chr>  <chr>       <chr>
##  1 0001          20.45800 0001010C 0001T01    full-… public      male
##  2 0002          14.977   0002010C 0002T01    half-… public      fema…
##  3 0009          18.82    0009026C 0009T01    half-… public      male
##  4 0009          18.82    0009014C 0009T02    half-… public      male
##  5 0009          18.82    0009005C 0009T01    half-… public      male
##  6 0016          17.55100 0016004C 0016T01    half-… public      male
##  7 0016          17.55100 0016009C 0016T01    half-… public      fema…
##  8 0022          17.8465  0022005C 0022T01    half-… public      male
##  9 0022          17.8465  0022014C 0022T03    half-… public      fema…
## 10 0023          15.5050  0023017C 0023T04    half-… public      male
## # … with 964 more rows, and 27 more variables: ethnic <chr>,
## #   famtype <chr>, numsibs <dbl>, SES_cont <dbl>, SES_cat <chr>,
## #   age <dbl>, T1RSCALE <dbl>, T1MSCALE <dbl>, T1GSCALE <dbl>,
## #   T2RSCALE <dbl>, T2MSCALE <dbl>, T2GSCALE <dbl>, IRTreadgain <dbl>,
```

Why is this so beneficial?

- Sometimes the boolean logic for `filter` can be overly complicated.

Why is this so beneficial?

- Sometimes the boolean logic for `filter` can be overly complicated.
- Instead, create a data frame that has only the groups you want, and `semi_join` it with your original data

Why is this so beneficial?

- Sometimes the boolean logic for `filter` can be overly complicated.
- Instead, create a data frame that has only the groups you want, and `semi_join` it with your original data
- Alternatively, create a data frame that has all **but** the values you want.

One more quick example

Stop words

```
# install.packages(c("tidytext", "janeaustenr"))
library(tidytext)
library(janeaustenr)

austen_books()
```

```
## # A tibble: 73,422 x 2
##    text                  book
##  * <chr>                 <fct>
##  1 SENSE AND SENSIBILITY Sense & Sensibility
##  2 ""                    Sense & Sensibility
##  3 by Jane Austen        Sense & Sensibility
##  4 ""                    Sense & Sensibility
##  5 (1811)                Sense & Sensibility
##  6 ""                    Sense & Sensibility
##  7 ""                    Sense & Sensibility
##  8 ""                    Sense & Sensibility
##  9 ""                    Sense & Sensibility
## 10 CHAPTER 1             Sense & Sensibility
## # … with 73,412 more rows
```

Get words

```
austen_books() %>%
  unnest_tokens(word, text)
```

```
## # A tibble: 725,055 x 2
##    book                word
##    <fct>               <chr>
##  1 Sense & Sensibility sense
##  2 Sense & Sensibility and
##  3 Sense & Sensibility sensibility
##  4 Sense & Sensibility by
##  5 Sense & Sensibility jane
##  6 Sense & Sensibility austen
##  7 Sense & Sensibility 1811
##  8 Sense & Sensibility chapter
##  9 Sense & Sensibility 1
## 10 Sense & Sensibility the
## # … with 725,045 more rows
```

Count words

```
austen_books() %>%
  unnest_tokens(word, text) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 14,520 x 2
##    word      n
##    <chr> <int>
##  1 the   26351
##  2 to    24044
##  3 and   22515
##  4 of    21178
##  5 a     13408
##  6 her   13055
##  7 i     12006
##  8 in    11217
##  9 was   11204
## 10 it    10234
## # … with 14,510 more rows
```

Stop words

```
stop_words
```

```
## # A tibble: 1,149 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 a           SMART
##  2 a's         SMART
##  3 able        SMART
##  4 about       SMART
##  5 above       SMART
##  6 according   SMART
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
## # … with 1,139 more rows
```

Remove stop words

```
austen_books() %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 13,914 x 2
##    word        n
##    <chr>   <int>
##  1 miss     1855
##  2 time     1337
##  3 fanny     862
##  4 dear      822
##  5 lady      817
##  6 sir       806
##  7 day       797
##  8 emma      787
##  9 sister    727
## 10 house     699
## # … with 13,904 more rows
```

Wrapping up

- Homework 1 assigned today
  - Be careful about keys. Likely to be rather tricky.
- Next time: Visual perception