Joins

Daniel Anderson

Week 3, Class 2

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 really 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

```r
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)))
```

```r
g3
```

```
## # A tibble: 3 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     1     3   215
## 2     2     3   203
## 3     3     3   200
```

```r
g4
```

```
## # A tibble: 3 x 3
##     sid grade score
##   <int> <dbl> <int>
## 1     9     4   213
## 2    10     4   191
## 3    11     4   209
```

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   215
## 2     2     3   203
## 3     3     3   200
## 4     9     4   213
## 5    10     4   191
## 6    11     4   209
```

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   215
## 2 1           2   203
## 3 1           3   200
## 4 2           9   213
## 5 2          10   191
## 6 2          11   209
```

```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   215     3
## 2 1           2   203     3
## 3 1           3   200     3
## 4 2           9   213     4
## 5 2          10   191     4
## 6 2          11   209     4
```

Even better usage

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

```
## # A tibble: 6 x 3
##   grade   sid score
##   <chr> <int> <int>
## 1 g3        1   215
## 2 g3        2   203
## 3 g3        3   200
## 4 g4        9   213
## 5 g4       10   191
## 6 g4       11   209
```

What if columns don't match exactly?

Pad with `NA`

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

```
## # A tibble: 6 x 4
##   dataset   sid grade score
##   <chr>   <int> <dbl> <int>
## 1 1           1     3   215
## 2 1           2     3   203
## 3 1           3     3   200
## 4 2           9    NA   213
## 5 2          10    NA   191
## 6 2          11    NA   209
```

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 = '
                     .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
- 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?

```r
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
##   <chr>    <chr>      <chr>     <chr>    <chr>       <chr>  <chr>
## 1 0842021C 0842T02    0842      full-day public      male   BLACK OR AFR
## 2 0905002C 0905T01    0905      full-day private     male   ASIAN
## 3 0150012C 0150T01    0150      full-day private     female BLACK OR AFR
## 4 0556009C 0556T01    0556      full-day private     female HISPANIC, RA
## 5 0089013C 0089T04    0089      full-day public      male   WHITE, NON-H
## 6 1217001C 1217T13    1217      half-day public      female NATIVE HAWAI
## # … with 978 more rows, and 26 more variables: famtype <chr>, numsibs <c
## #   SES_cat <chr>, age <dbl>, T1RSCALE <dbl>, T1MSCALE <dbl>, T1GSCALE <
## #   T2MSCALE <dbl>, T2GSCALE <dbl>, IRTreadgain <dbl>, IRTmathgain <dbl>
## #   T1ARSLIT <dbl>, T1ARSMAT <dbl>, T1ARSGEN <dbl>, T2ARSLIT <dbl>, T2AR
## #   ARSlitgain <dbl>, ARSmathgain <dbl>, ARSgkgain <dbl>, testdate1 <dat
## #   elapse <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
## # … with 978 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 <- read_csv(here("data", "incomeInequality_tidy.csv")
print(income_ineq, n = 15)
```

```
## # A tibble: 726 x 6
##     Year Number.thousands realGDPperCap PopulationK percentile     incom
##    <dbl>            <dbl>         <dbl>       <dbl>      <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      26764.1
##  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
## 11  1947            37237      14117.32      144126      99.99 1584506
## 12  1948            38624      14451.94      146631      20      13779
## 13  1948            38624      14451.94      146631      40      22655
## 14  1948            38624      14451.94      146631      60      30248
## 15  1948            38624      14451.94      146631      80      42196
## # … with 711 more rows
```

```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
##   <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
## # … with 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_
## #   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
## # … with 5 more rows
```

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
## # … with 336,770 more rows
```

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.
- 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.

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 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
00	'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'

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

```r
dis_tbl %>%
  mutate(disability = case_when(
    dis_code == "10" ~ "Intellectual Disability",
    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', '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
## # … with 8 more rows
```

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
## # … with 194 more rows
```

Imperfect key match?

Consider the following

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

```r
gender
```

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

```r
sped
```

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

left_join()?

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

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

right_join()?

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

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

inner_join()?

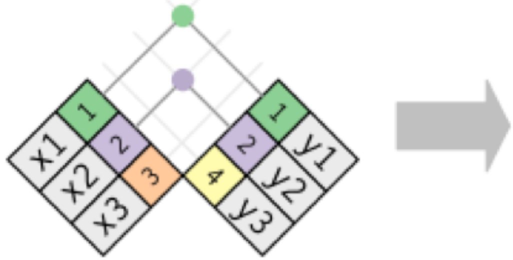
```
inner_join(gender, sped)
```

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

full_join()?

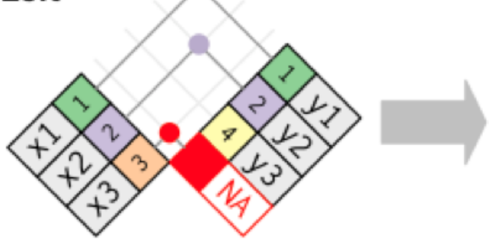
```
full_join(gender, sped)
```

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



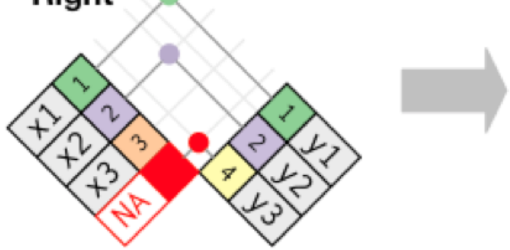
key	val_x	val_y
1	x1	y1
2	x2	y2

Left



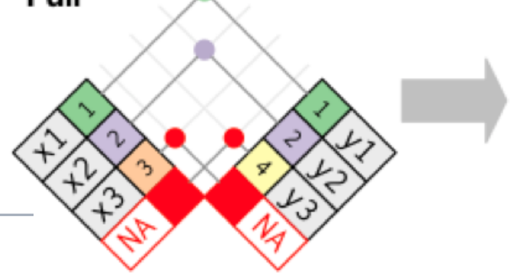
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA

Right



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

Full



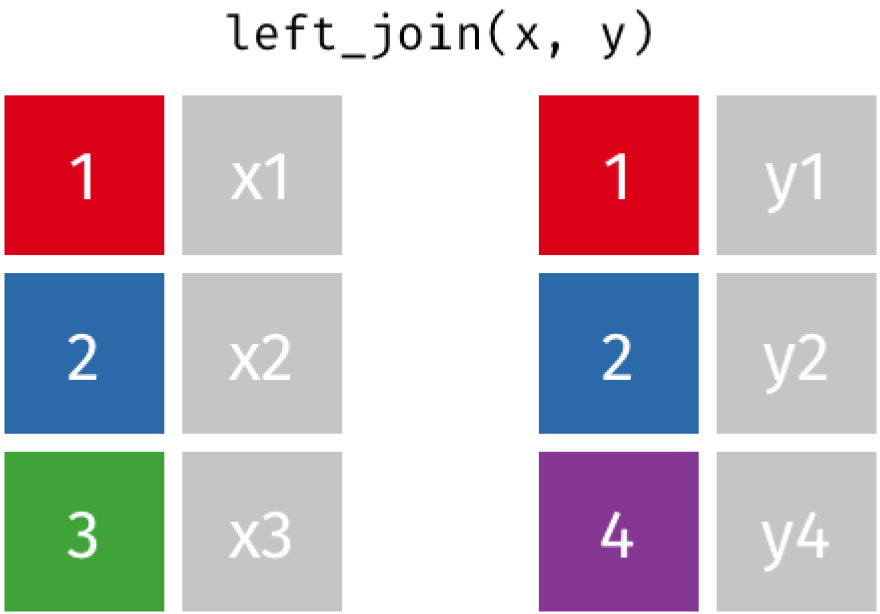
key	val_x	val_y
1	x1	y1
2	x2	y2
3	x3	NA
4	NA	y3

From r4ds

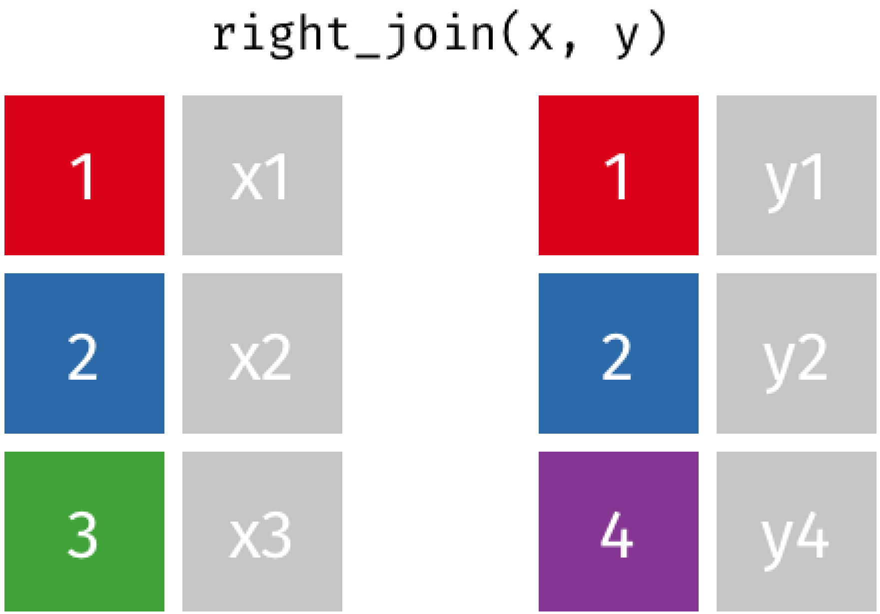
Animations

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

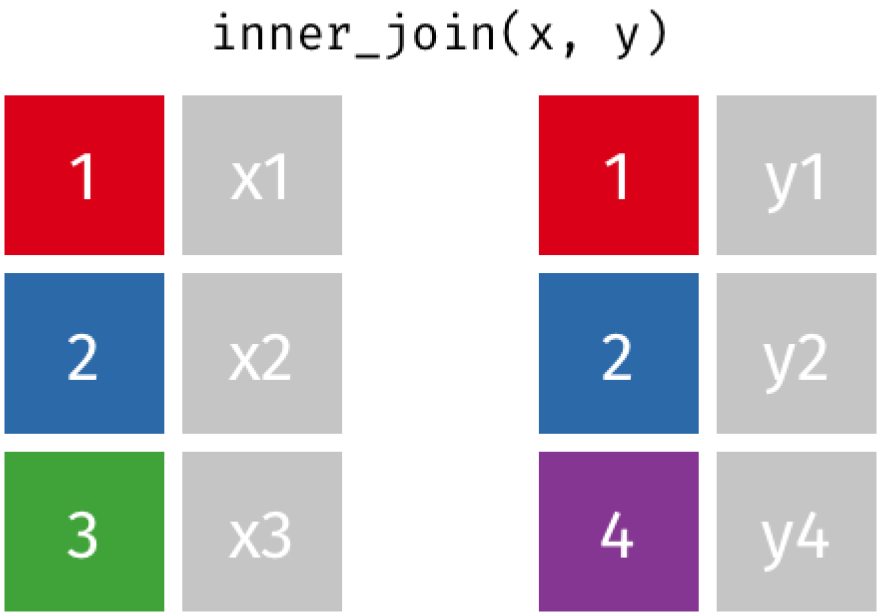
Animated `left_join()`



Animated `right_join`

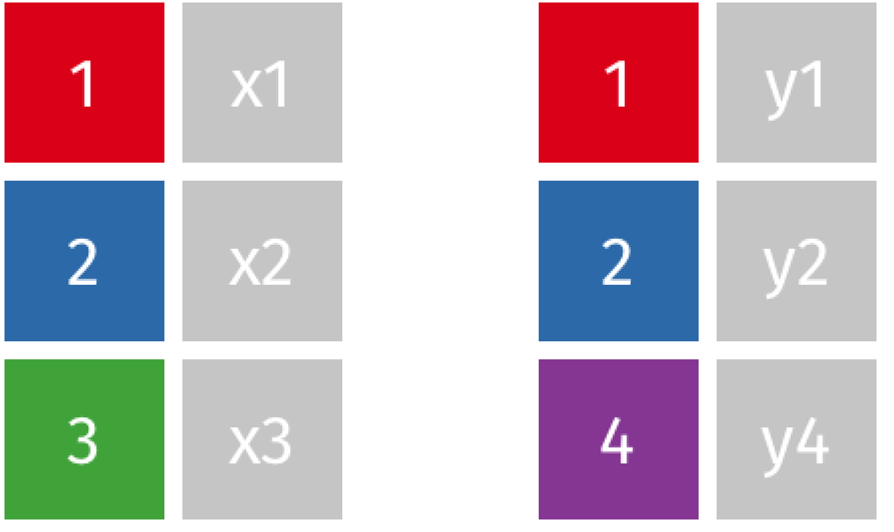


Animated `inner_join`



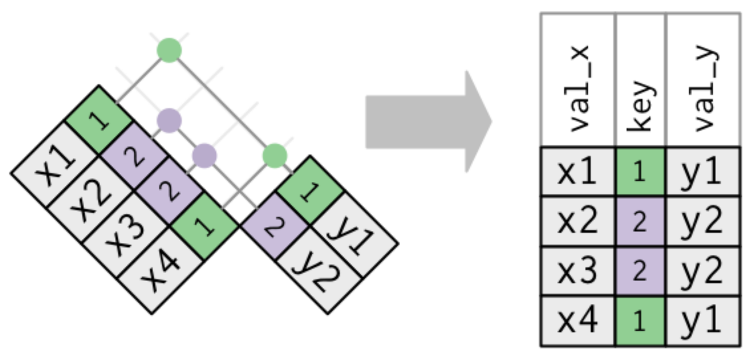
Animated `full_join`

```
full_join(x, y)
```



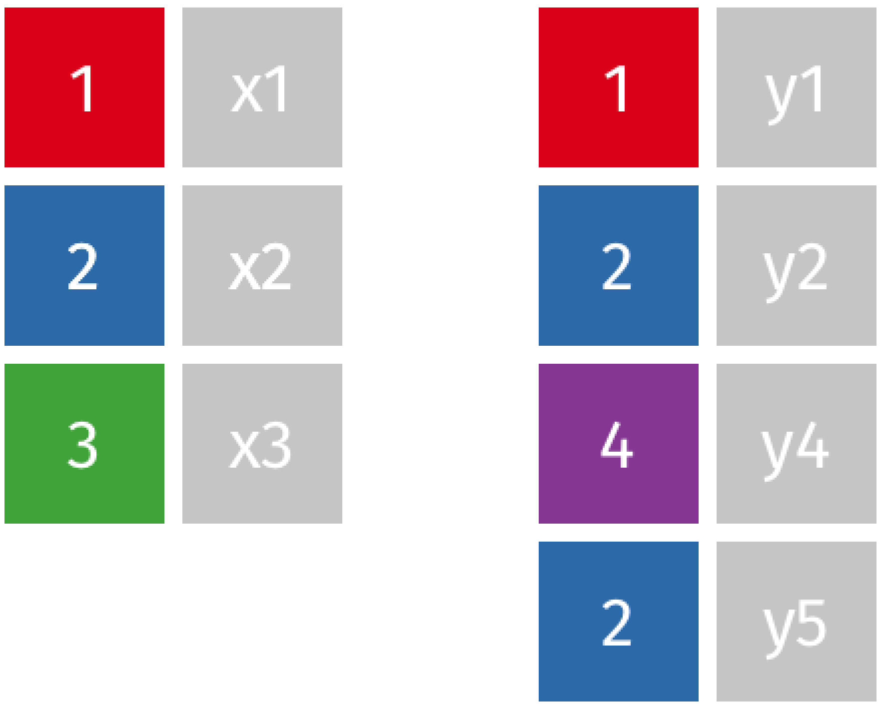
What if the key is not unique?

- 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

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



Example

A dataset with school IDs

```r
stu <- tibble(
  sid = 1:9,
  scid = c(1, 1, 1, 1, 2, 2, 3, 3, 3),
  score = c(10, 12, 15, 8,  9, 11, 12, 15, 17)
  )
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
## 5     5     2     9
## 6     6     2    11
## # … with 3 more rows
```

A school–level dataset

```r
schl <- tibble(
  scid = 1:3,
  stu_tch_ratio = c(22.05, 31.14, 24.87),
  per_pupil_spending = c(15741.08, 11732.24, 13027.88)
)
schl
```

```
## # A tibble: 3 x 3
##    scid stu_tch_ratio per_pupil_spending
##   <int>         <dbl>              <dbl>
## 1     1         22.05           15741.08
## 2     2         31.14           11732.24
## 3     3         24.87           13027.88
```

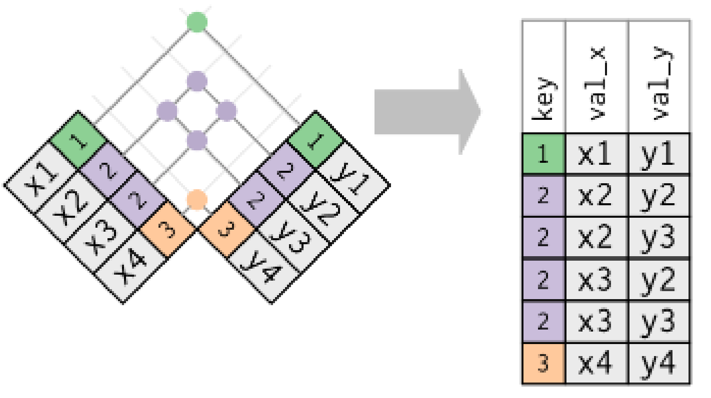
One to many

```r
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         22.05           15741.08
## 2     2     1    12         22.05           15741.08
## 3     3     1    15         22.05           15741.08
## 4     4     1     8         22.05           15741.08
## 5     5     2     9         31.14           11732.24
## 6     6     2    11         31.14           11732.24
## # … with 3 more rows
```

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).



Example

```r
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    0.3447951
## 2     1 winter  1.539648
## 3     1 spring -0.3295142
## 4     2 fall    0.9483894
## 5     2 winter -0.4792556
## 6     2 spring -1.514887
## # … with 3 more rows
```

```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    0.3447951
## 2     1     1    10 winter  1.539648 
## 3     1     1    10 spring -0.3295142
## 4     2     1    12 fall    0.3447951
## 5     2     1    12 winter  1.539648 
## 6     2     1    12 spring -0.3295142
## # … with 21 more rows
```

How do we fix this?

In some cases, the solution is obvious. In others, it's 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

```r
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 0.3447951  1.539648  -0.3295142
## 2     2 0.9483894 -0.4792556 -1.514887
## 3     3 0.4345367 -0.5195367 -0.8345590
```

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 0.3447951  1.539648  -0.3295142
## 2     2     1    12 0.3447951  1.539648  -0.3295142
## 3     3     1    15 0.3447951  1.539648  -0.3295142
## 4     4     1     8 0.3447951  1.539648  -0.3295142
## 5     5     2     9 0.9483894 -0.4792556 -1.514887
## 6     6     2    11 0.9483894 -0.4792556 -1.514887
## # … with 3 more rows
```

Move longer again?

If we did, we'd be exactly where we were with the first join.

You could make the argument it *might* make sense here

I'd still argue for *this* approach, not the cartesian product approach

More systematic, more predictable, and ultimately less error prone

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
##   <chr>    <chr>      <chr>     <chr>    <chr>       <chr>  <chr>
## 1 0842021C 0842T02    0842      full-day public      male   BLACK OR AFR
## 2 0905002C 0905T01    0905      full-day private     male   ASIAN
## 3 0150012C 0150T01    0150      full-day private     female BLACK OR AFR
## 4 0556009C 0556T01    0556      full-day private     female HISPANIC, RA
## 5 0089013C 0089T04    0089      full-day public      male   WHITE, NON-H
## 6 1217001C 1217T13    1217      half-day public      female NATIVE HAWAI
## # … with 978 more rows, and 26 more variables: famtype <chr>, numsibs <c
## #   SES_cat <chr>, age <dbl>, T1RSCALE <dbl>, T1MSCALE <dbl>, T1GSCALE <
## #   T2MSCALE <dbl>, T2GSCALE <dbl>, IRTreadgain <dbl>, IRTmathgain <dbl>
## #   T1ARSLIT <dbl>, T1ARSMAT <dbl>, T1ARSGEN <dbl>, T2ARSLIT <dbl>, T2AR
## #   ARSlitgain <dbl>, ARSmathgain <dbl>, ARSgkgain <dbl>, testdate1 <dat
## #   elapse <dbl>
```

Compute group means

```r
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          20.45800
## 2 0002          14.977
## 3 0009          18.82
## 4 0013          42.321
## 5 0016          17.55100
## 6 0022          17.8465
## # … with 509 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          20.45800 0001010C 0001T01    full-day
## 2 0002          14.977   0002010C 0002T01    half-day
## 3 0009          18.82    0009026C 0009T01    half-day
## 4 0009          18.82    0009014C 0009T02    half-day
## 5 0009          18.82    0009005C 0009T01    half-day
## 6 0013          42.321   0013003C 0013T01    full-day
## # … with 978 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 w
##   <chr>  <int> <int> <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 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 humic
##   <int> <int> <int> <dbl> <chr>  <chr> <chr>   <chr>   <dbl> <dbl> <dbl>
## 1  2013     1     1     5 EWR    IAH   N14228  UA      39.02 28.04 64.43
## 2  2013     1     1     5 LGA    IAH   N24211  UA      39.92 24.98 54.81
## 3  2013     1     1     5 JFK    MIA   N619AA  AA      39.02 26.96 61.63
## 4  2013     1     1     5 JFK    BQN   N804JB  B6      39.02 26.96 61.63
## 5  2013     1     1     6 LGA    ATL   N668DN  DL      39.92 24.98 54.81
## 6  2013     1     1     5 EWR    ORD   N39463  UA      39.02 28.04 64.43
## # … with 336,770 more rows, and 4 more variables: precip <dbl>, pressure
## #   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      engin
##   <chr>   <int> <chr>                         <chr>            <chr>        <ir
## 1 N10156   2004 Fixed wing multi engine EMBRAER          EMB-145XR
## 2 N102UW   1998 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 3 N103US   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 4 N104UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
## 5 N10575   2002 Fixed wing multi engine EMBRAER          EMB-145LR
## 6 N105UW   1999 Fixed wing multi engine AIRBUS INDUSTRIE A320-214
```

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 Fixed wir
## 2    2013     1     1     5 LGA    IAH   N24211  UA        1998 Fixed wir
## 3    2013     1     1     5 JFK    MIA   N619AA  AA        1990 Fixed wir
## 4    2013     1     1     5 JFK    BQN   N804JB  B6        2012 Fixed wir
## 5    2013     1     1     6 LGA    ATL   N668DN  DL        1991 Fixed wir
## 6    2013     1     1     5 EWR    ORD   N39463  UA        2012 Fixed wir
## # … with 336,770 more rows, and 5 more variables: model <chr>, engines <
## #   speed <int>, engine <chr>
```

Mismatched 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         22.05           15741.08
## 2         2         31.14           11732.24
## 3         3         24.87           13027.88
```

```
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
## 5     5     2     9
## 6     6     2    11
## # … with 3 more rows
```

Join w/mismatched names

```r
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         22.05           15741.08
## 2     2     1    12         22.05           15741.08
## 3     3     1    15         22.05           15741.08
## 4     4     1     8         22.05           15741.08
## 5     5     2     9         31.14           11732.24
## 6     6     2    11         31.14           11732.24
## # … with 3 more rows
```

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

```r
av_pre_mth <- ecls %>%
    mutate(cut_high = mean(T1MSCALE) + 3*sd(T1MSCALE)) %>%
    group_by(teacher_id, k_type) %>%
    summarize(av_pre_mth = mean(T1MSCALE),
              cut_high = unique(cut_high))
av_pre_mth
```

```
## # A tibble: 707 x 4
## # Groups:   teacher_id [707]
##   teacher_id k_type   av_pre_mth cut_high
##   <chr>      <chr>         <dbl>    <dbl>
## 1 0001T01    full-day   20.45800 42.62333
## 2 0002T01    half-day   14.977   42.62333
## 3 0009T01    half-day   17.6475  42.62333
## 4 0009T02    half-day   21.165   42.62333
## 5 0013T01    full-day   42.321   42.62333
## 6 0016T01    half-day   17.55100 42.62333
## # … with 701 more rows
```

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

```
extr_high <- av_pre_mth %>%
    ungroup() %>%
    filter(av_pre_mth > cut_high)
extr_high
```

```
## # A tibble: 3 x 4
##   teacher_id k_type   av_pre_mth cut_high
##   <chr>      <chr>         <dbl>    <dbl>
## 1 0078T04    half-day    45.75   42.62333
## 2 0663T01    full-day    42.8455 42.62333
## 3 0944T03    half-day    45.371  42.62333
```

Finally, use `semi_join` to show the full data for these cases

```
semi_join(ecls, extr_high)
```

```
## # A tibble: 4 x 33
##   child_id teacher_id school_id k_type   school_type sex    ethnic
##   <chr>    <chr>      <chr>     <chr>    <chr>       <chr>  <chr>
## 1 0944017C 0944T03    0944      half-day private     female WHITE, NON-H
## 2 0663006C 0663T01    0663      full-day private     male   WHITE, NON-H
## 3 0663012C 0663T01    0663      full-day private     female WHITE, NON-H
## 4 0078020C 0078T04    0078      half-day public      female WHITE, NON-H
## # … with 26 more variables: famtype <chr>, numsibs <dbl>, SES_cont <dbl>
## #   T1RSCALE <dbl>, T1MSCALE <dbl>, T1GSCALE <dbl>, T2RSCALE <dbl>, T2MS
## #   IRTreadgain <dbl>, IRTmathgain <dbl>, IRTgkgain <dbl>, T1ARSLIT <dbl
## #   T1ARSGEN <dbl>, T2ARSLIT <dbl>, T2ARSMAT <dbl>, T2ARSGEN <dbl>, ARSl
## #   ARSmathgain <dbl>, ARSgkgain <dbl>, testdate1 <date>, testdate2 <dat
```

Filtering joins

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

```
nrow(ecls)
```

```
## [1] 984
```

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

```
## [1] 980
```

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.

Stop words

One more quick example

This one is probs more realistic

Jane Austen Books

```r
# 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
## # … with 73,416 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
## # … with 725,049 more rows
```

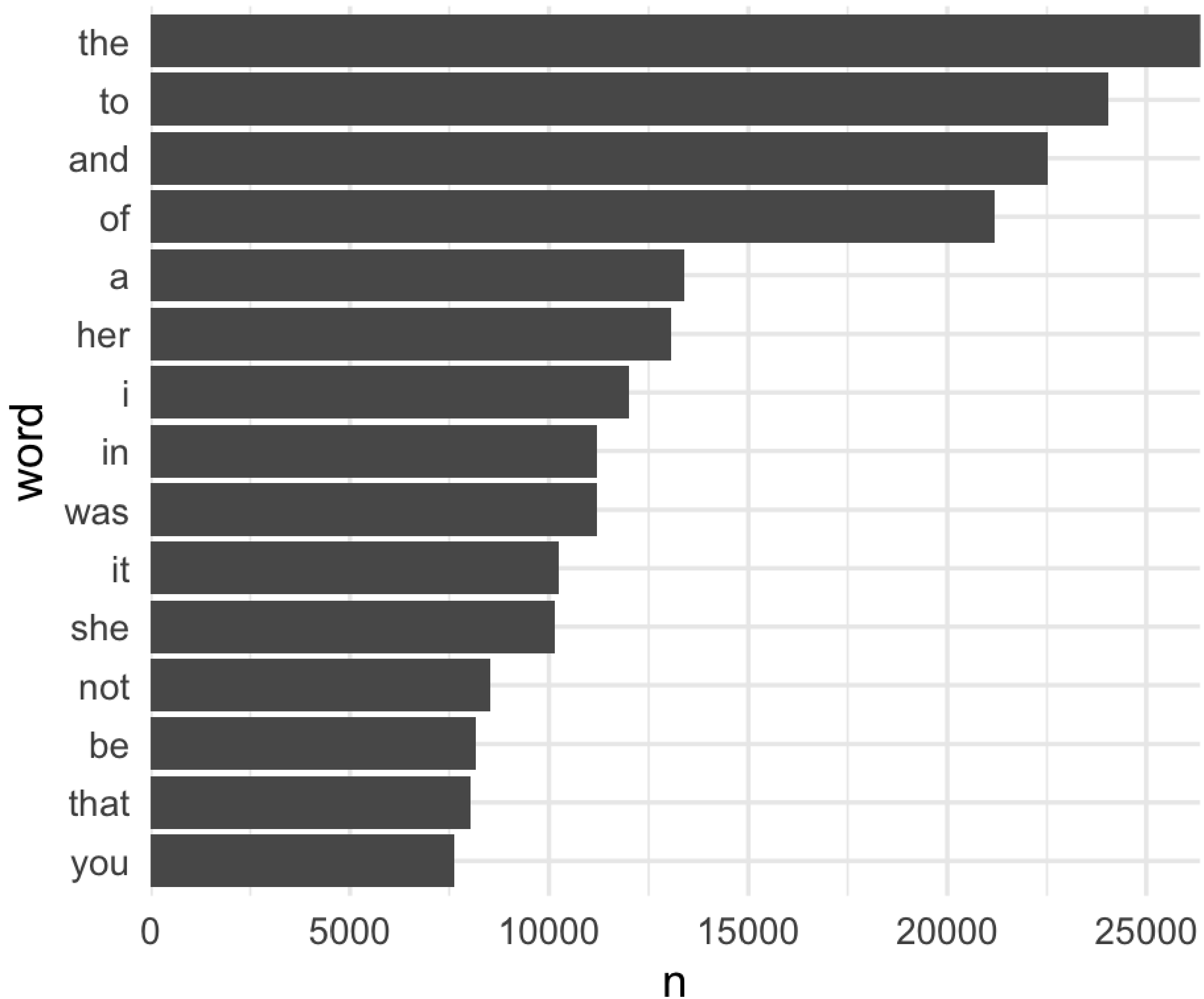
Count words

```r
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
## # … with 14,514 more rows
```

Plot top 15 words

```
austen_books() %>%
  unnest_tokens(word, text) %>%
  count(word, sort = TRUE) %>%
  mutate(word = fct_reorder(word, n)) %>%
  slice(1:15) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip()
```


the
to
and
of
a
her
i
in
was
it
she
not
be
that
you
word
0
5000
10000
15000
20000
25000
n

Stop words

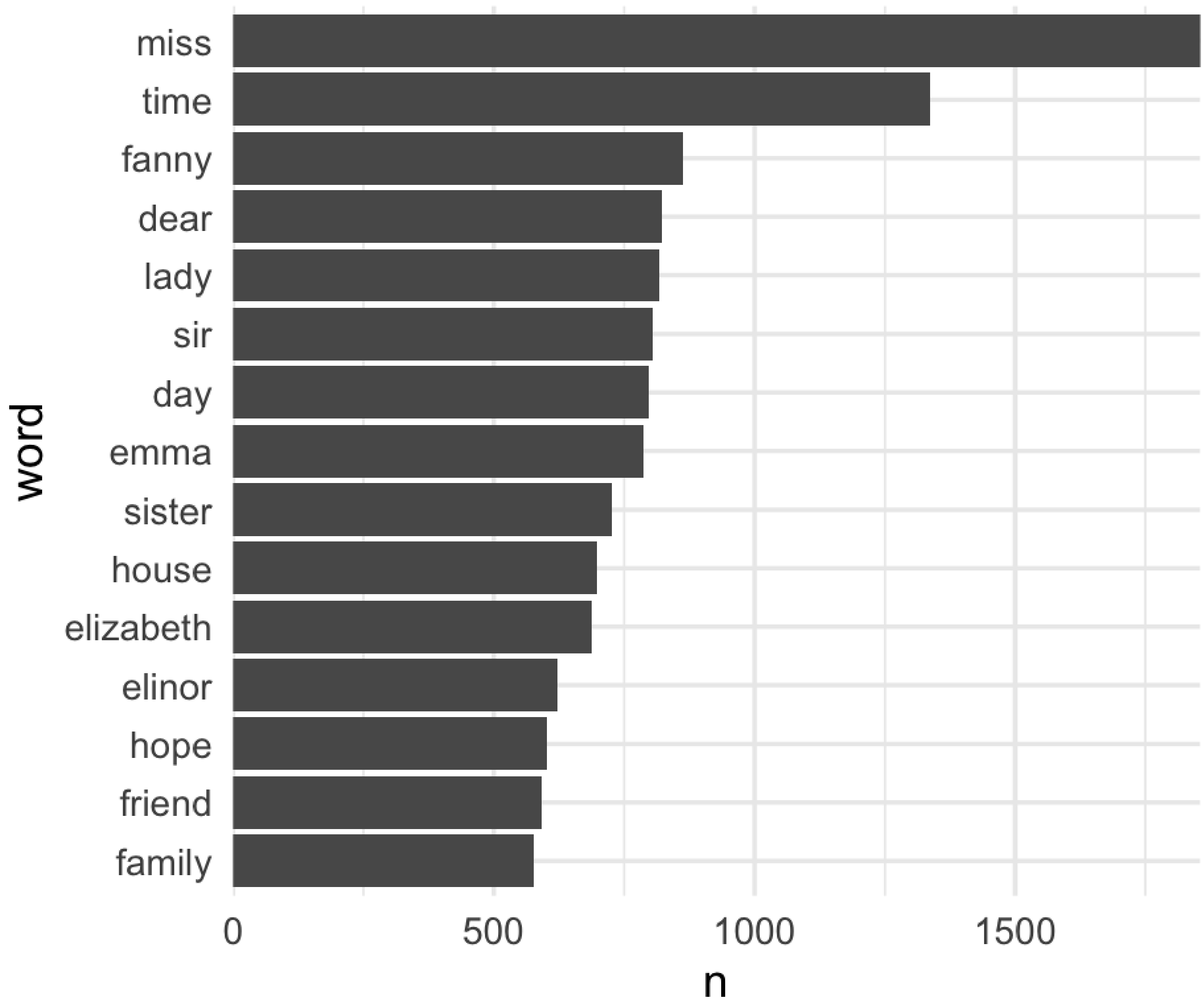
```
stop_words
```

```
## # 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
## # … with 1,143 more rows
```

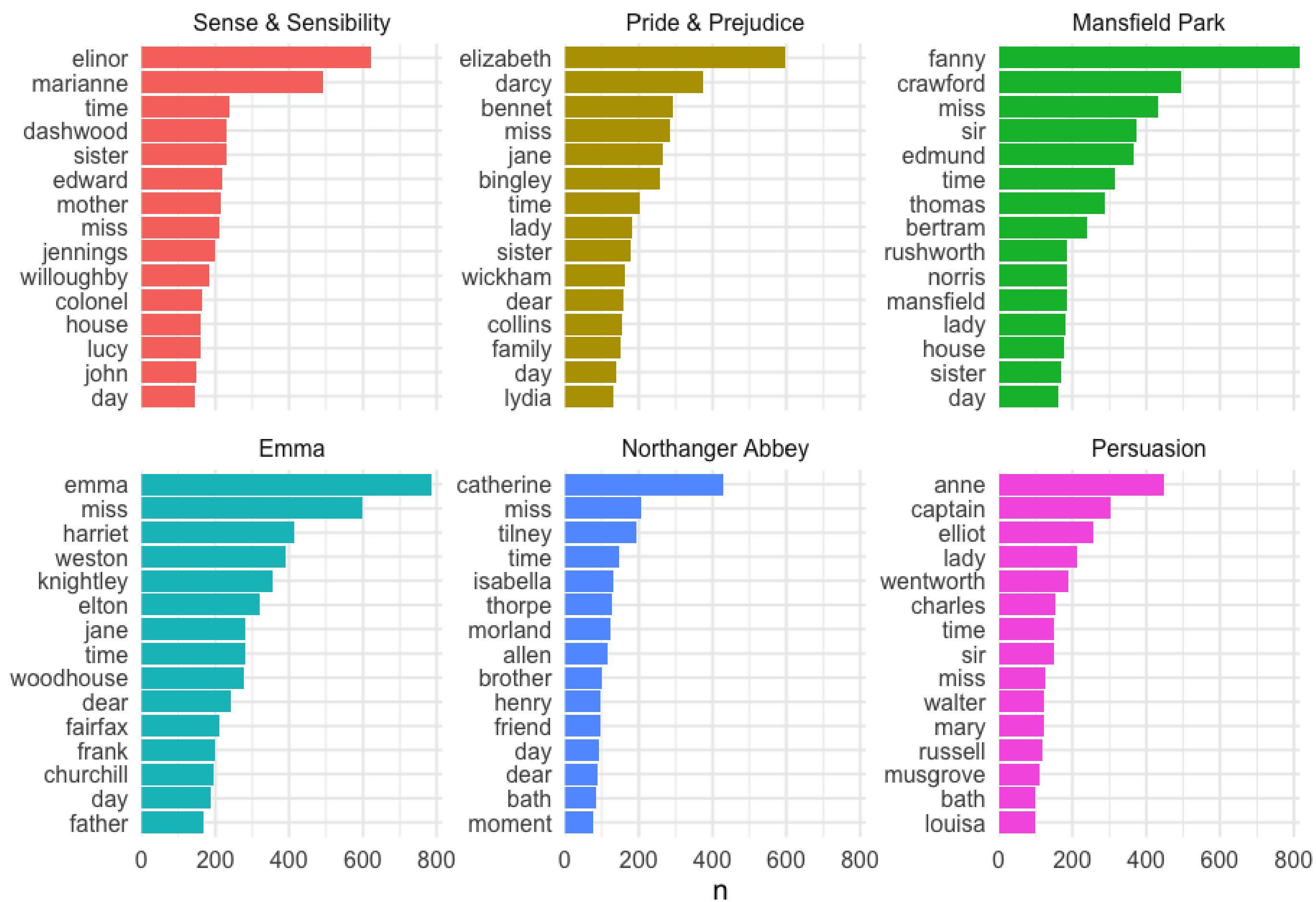
Remove stop words

```
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
## # … with 13,908 more rows
```


miss
time
fanny
dear
lady
sir
day
emma
sister
house
elizabeth
elinor
hope
friend
family
word
0
500
1000
1500
n

By book



Sense & Sensibility
elinor
marianne
time
dashwood
sister
edward
mother
miss
jennings
willoughby
colonel
house
lucy
john
day
Pride & Prejudice
elizabeth
darcy
bennet
miss
jane
bingley
time
lady
sister
wickham
dear
collins
family
day
lydia
Mansfield Park
fanny
crawford
miss
sir
edmund
time
thomas
bertram
rushworth
norris
mansfield
lady
house
sister
day
Emma
emma
miss
harriet
weston
knightley
elton
jane
time
woodhouse
dear
fairfax
frank
churchill
day
father
Northanger Abbey
catherine
miss
tilney
time
isabella
thorpe
morland
allen
brother
henry
friend
day
dear
bath
moment
Persuasion
anne
captain
elliot
lady
wentworth
charles
time
sir
miss
walter
mary
russell
musgrove
bath
louisa
0
200
400
600
800
n

Wrapping up

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