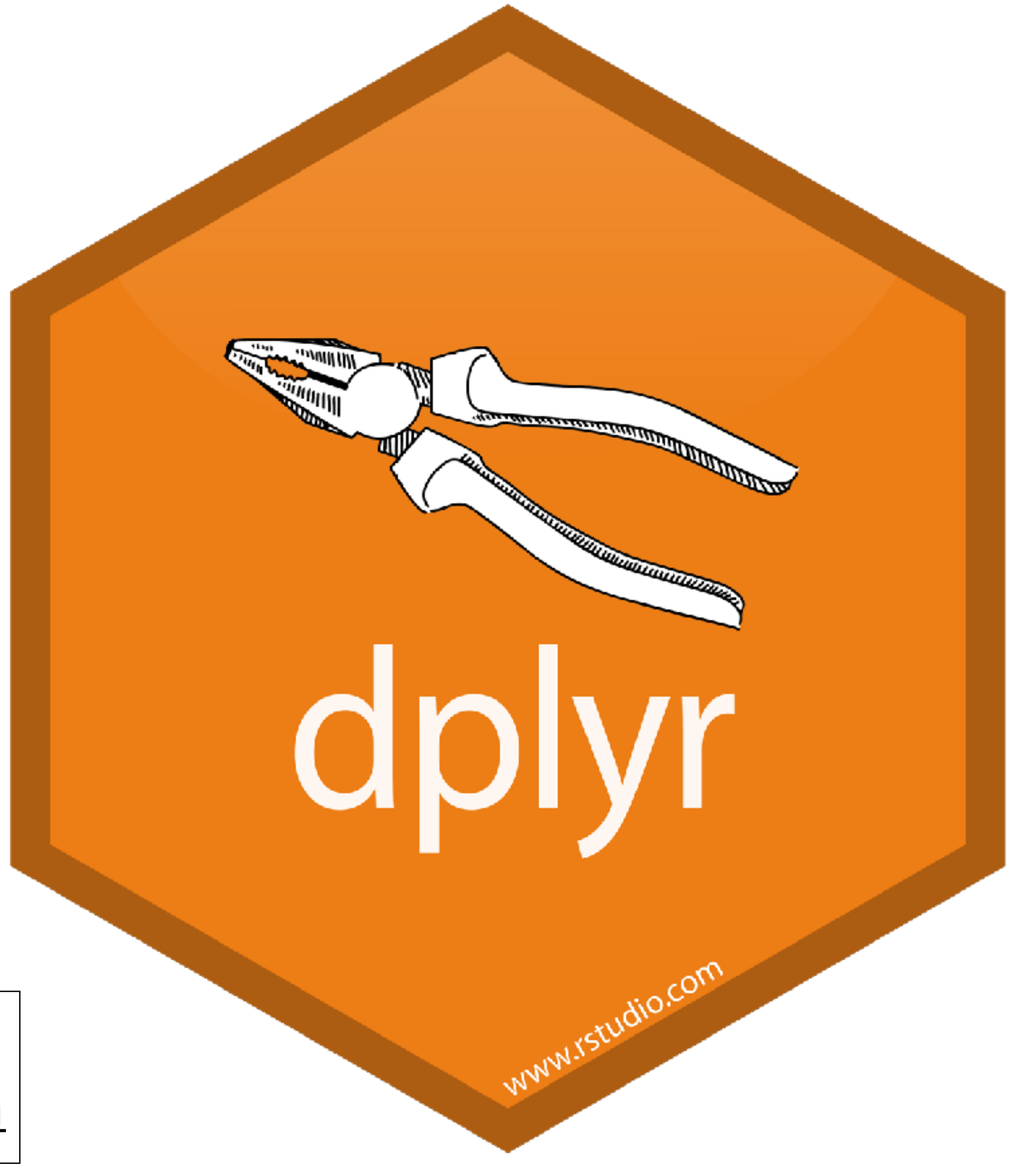
Data Transformation

Jake Thompson
wjakethompson.com
@wjakethompson



In R4DS

Data Transformation

ATLAS

Your Turn 0

- Open **02-Transform.Rmd**
- Run the setup chunk

babynames

babynames

year <dbl>	sex <chr>	name <chr>	n <int>	prop <dbl>
1880	F	Mary	7065	0.07238359
1880	F	Anna	2604	0.02667896
1880	F	Emma	2003	0.02052149
1880	F	Elizabeth	1939	0.01986579
1880	F	Minnie	1746	0.01788843
1880	F	Margaret	1578	0.01616720
1880	F	Ida	1472	0.01508119
1880	F	Alice	1414	0.01448696
1880	F	Bertha	1320	0.01352390
1880	F	Sarah	1288	0.01319605

1–10 of 1,924,665 rows

Previous 1 2 3 4 5 6 ... 100 Next

ATLAS

skim(babynames)

```
Skim summary statistics
 n obs: 1924665
 n variables: 5

── Variable type:character ─────────────────────────────────────────────────────────────────────────
 variable missing complete       n min max empty n_unique
     name       0  1924665 1924665   2  15     0    97310
      sex       0  1924665 1924665   1   1     0        2

── Variable type:integer ───────────────────────────────────────────────────────────────────────────
 variable missing complete       n   mean      sd p0 p25 p50 p75  p100     hist
        n       0  1924665 1924665 180.87 1533.34  5   7  12  32 99686 ▇▁▁▁▁▁▁▁▁
```

```
── Variable type:numeric ──────────────────────────────────────────────────────────────────────────────────────────────
 variable missing complete       n       mean      sd          p0        p25        p50        p75     p100     hist
     prop       0  1924665 1924665    0.00014  0.0012    2.3e-06    3.9e-06    7.3e-06    2.3e-05    0.082 ▇▁▁▁▁▁▁▁▁
     year       0  1924665 1924665 1974.85    34.03   1880       1951       1985       2003       2017      ▁▂▃▅▇
```

skimr output

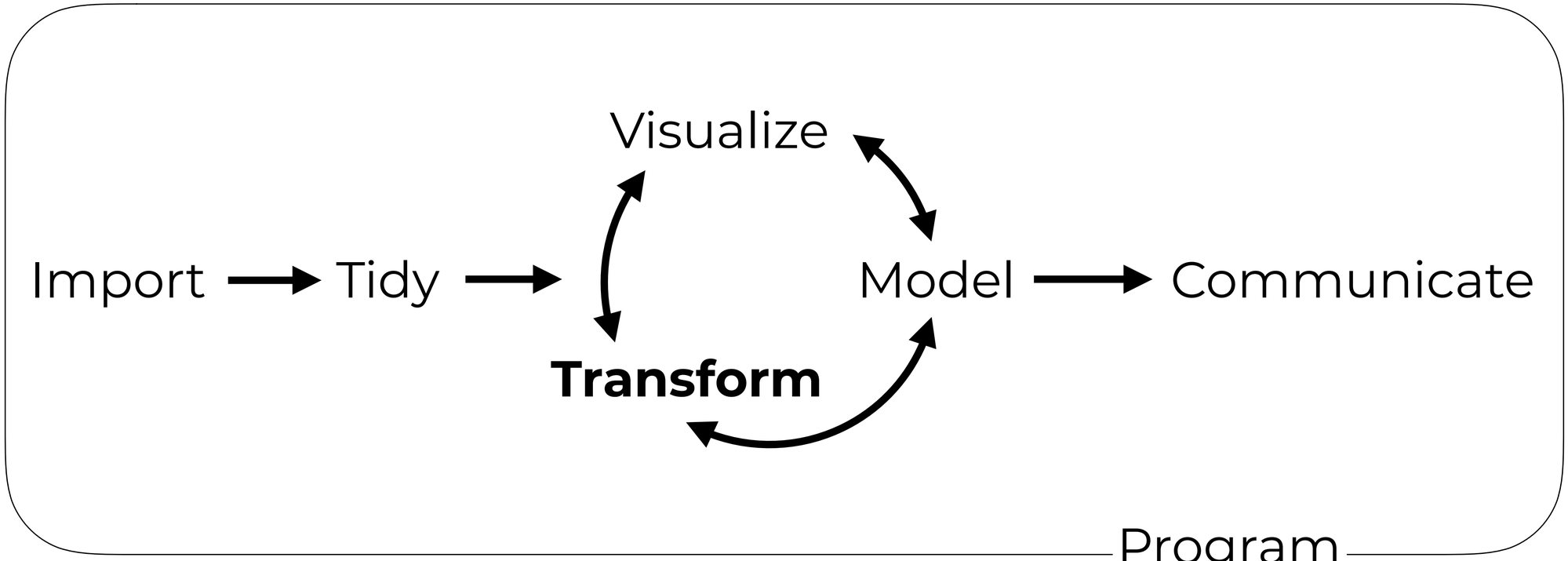
```
skim_with(integer = list(p25 = NULL, p75 = NULL))
```

```
skim_with_defaults()
```


Your Turn 1

- Run the `skim_with()` command.
- Skim `babynames` again to see the difference

01:00



Isolating data

Extract variables with **`select()`**

Extract cases with **`filter()`**

Arrange cases with **`arrange()`**

select()

Extract columns by name.

```
select(.data, ...)
```

Data frame to transform

name(s) of columns to extract (or a select helper function)

select()

Extract columns by name.

```
select(babynames, name, prop)
```

year	sex	name	n	prop
1880	F	Mary	7065	0.072
1880	F	Anna	2604	0.027
1880	F	Emma	2003	0.021
1880	F	Elizabeth	1939	0.020
1880	F	Minnie	1746	0.018
1880	F	Margaret	1578	0.016
1880	F	Ida	1472	0.015

name	prop
Mary	0.072
Anna	0.027
Emma	0.021
Elizabeth	0.020
Minnie	0.018
Margaret	0.016
Ida	0.015

Your Turn 2

- Alter the code to select just the **n** column

```
select(babynames, name, prop)
```

select() helpers

: - Select range of columns

```
select(storms, name:pressure)
```

- - Select every column but

```
select(storms, -c(name, pressure))
```

starts_with() - Select columns that start with...

```
select(storms, starts_with("w"))
```

ends_with() - Select columns that end with...

```
select(storms, ends_with("e"))
```

select() helpers

contains() - Select columns whose names contain...

```
select(storms, contains("d"))
```

matches() - Select columns whose names match a regular expression

```
select(storms, matches("^.{4}$"))
```

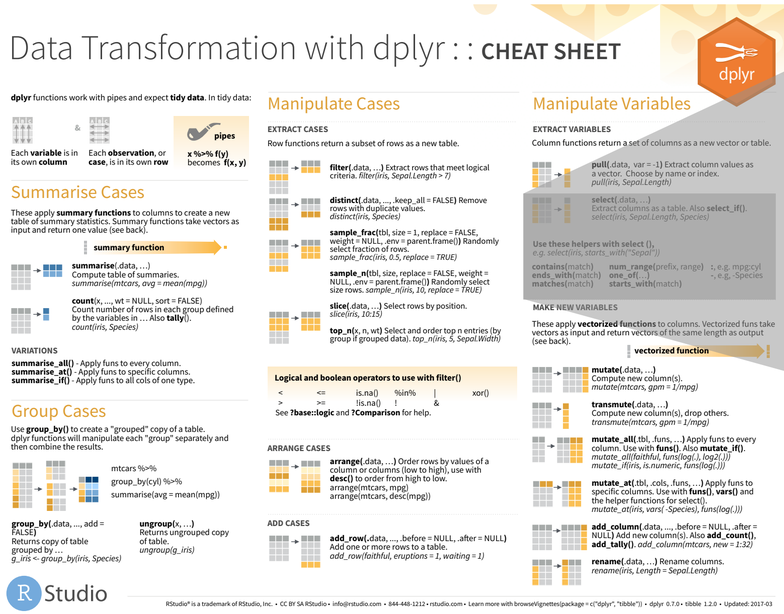
one_of() - Select columns whose names are one of a set

```
select(storms, one_of(c("name", "names", "Name")))
```

num_range() - Select columns named in prefix, number style

```
select(storms, num_range("x", 1:5))
```

select() helpers



Data Transformation with dplyr : : **CHEAT SHEET**

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, …, wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
distinct(iris, Species)

sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …) Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt) Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%	| xor()
>	>=	!is.na()	!	&

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL) Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …) Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match) **num_range**(prefix, range) **:**, e.g. mpg:cyl
ends_with(match) **one_of**(…) **-**, e.g, -Species
matches(match) **starts_with**(match)

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …)
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …)
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …) Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs(), vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, …, .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)** **num_range(**prefix, range**)** **:**, e.g. mpg:cyl
ends_with(match**)** **one_of(**…**)** **-**, e.g, -Species
matches(match**)** **starts_with(**match**)**

Consider

- Which of these is NOT a way to select the **name** and **n** columns together?

```
select(babynames, -c(year, sex, prop))

select(babynames, name:n)

select(babynames, starts_with("n"))

select(babynames, ends_with("n"))
```

Consider

- Which of these is NOT a way to select the **name** and **n** columns together?

```
select(babynames, -c(year, sex, prop))

select(babynames, name:n)

select(babynames, starts_with("n"))

select(babynames, ends_with("n"))
```

filter()

Extract columns that meet logical criteria

```
filter(.data, ...)
```

Data frame to transform

One or more logical tests (filter returns each row for which the test is TRUE)

Common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.

```
filter(.data, ...)
```

dplyr function

data frame to transform

function specific arguments

filter()

Extract columns that meet logical criteria

```
filter(babynames, name == "Jake")
```

year	sex	name	n	prop
1880	F	Mary	7065	0.072
1880	F	Anna	2604	0.027
1880	F	Emma	2003	0.021
1880	F	Elizabeth	1939	0.020
1880	F	Minnie	1746	0.018
1880	M	Jake	96	0.001
1880	F	Ida	1472	0.015

year	sex	name	n	prop
1880	M	Jake	96	0.001
1881	M	Jake	92	0.001
1882	M	Jake	97	0.001
1883	M	Jake	96	0.001
1884	M	Jake	92	0.001
1885	M	Jake	100	0.001
1886	M	Jake	95	0.001

filter()

Extract columns that meet logical criteria

```
filter(babynames, name == "Jake")
```

year	sex	name	n	prop
1880	F	Mary	7065	0.072
1880	F	Anna	2604	0.027
1880	F	Emma	2003	0.021
1880	F	Elizabeth	1939	0.020
1880	F	Minnie	1746	0.018
1880	M	Jake	96	0.001
1880	F	Ida	1472	0.015

= sets
(returns nothing)

== test if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership
`is.na(x)`	Is NA
`!is.na(x)`	Is not NA

Your Turn 3

See if you can use the logical operators to manipulate our code below to show:

- All of the names where **prop** is greater than or equal to 0.08
- All of the children named "Daenerys"
- All of the names that have a missing value for **n**
  (Hint: this should return an empty data set)

04 : 00

```
filter(babynames, prop >= 0.08)
#    year   sex      name      n         prop
# 1  1880     M      John   9655  0.08154561
# 2  1880     M   William   9531  0.08050676
# 3  1881     M      John   8769  0.08098299
```

```
filter(babynames, name == "Daenerys")
#    year   sex      name     n         prop
# 1  2012     F  Daenerys    21  1.085e-05
# 2  2013     F  Daenerys    68  3.535e-05
# 3  2014     F  Daenerys    86  4.407e-05
# 4  2015     F  Daenerys    82  4.215e-05
# 5  2016     F  Daenerys   101  5.237e-05
# 6  2017     F  Daenerys   110  5.867e-05
```

```
filter(babynames, is.na(n))
#  0 rows
```

filter()

Extract columns that meet *every* logical criteria

```
filter(babynames, name == "Jake", year == 1880)
```

year	sex	name	n	prop
1880	F	Mary	7065	0.072
1880	F	Anna	2604	0.027
1880	F	Emma	2003	0.021
1880	F	Elizabeth	1939	0.020
1880	F	Minnie	1746	0.018
1880	M	Jake	96	0.001
1880	F	Ida	1472	0.015

year	sex	name	n	prop
1880	M	Jake	96	0.001

Boolean operators

?base::Logic

`a & b`	and
`a | b`	or
`xor(a, b)`	exactly or
`!a`	not
`a %in% c(a, b)`	one of (in)

Your Turn 4

Use Boolean operators to alter the code below to return only the rows that contain:

- Girls name Sea
- Names that were used by exactly 5 or 6 children in 1880
- Names that are one of Acura, Lexus, or Yugo

```
filter(babynames, name == "Daenerys" | name == "Anemone")
```

arrange()

Order rows from smallest to largest values

```
arrange(.data, ...)
```

Data frame to transform

One or more columns to order by (additional columns will be used as tie breakers)

arrange()

Order rows from smallest to largest values

```
arrange(babynames, n)
```

year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020
1880	F	Ida	1472	0.015

year	sex	name	n	prop
1880	M	Jake	96	0.001
1880	F	Ida	1472	0.015
1880	F	Minnie	1746	0.018
1880	F	Elizabeth	1939	0.020
1880	F	Emma	2003	0.021
1880	F	Anna	2604	0.027
1880	F	Mary	7065	0.072

Your Turn 5

Arrange babynames by **n**. Add **prop** as a second (tie breaking) variable to arrange by.

- Can you tell what the smallest value of **n** is?

02 : 00

```
arrange(babynames, n, prop)
#      year sex   name              n          prop
#   1  2007 M     Aaban             5  0.00000226
#   2  2007 M     Aareon            5  0.00000226
#   3  2007 M     Aaris             5  0.00000226
#   4  2007 M     Abd               5  0.00000226
#   5  2007 M     Abdulazeez        5  0.00000226
#   6  2007 M     Abdulhadi         5  0.00000226
#   7  2007 M     Abdulhamid        5  0.00000226
#   8  2007 M     Abdulkadir        5  0.00000226
#   9  2007 M     Abdulraheem       5  0.00000226
#  10  2007 M     Abdulrahim        5  0.00000226
#  ... with 1,924,655 more rows
```

desc()

Changes ordering to largest to smallest

```
arrange(babynames, desc(n))
```

year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020
1880	F	Ida	1472	0.015

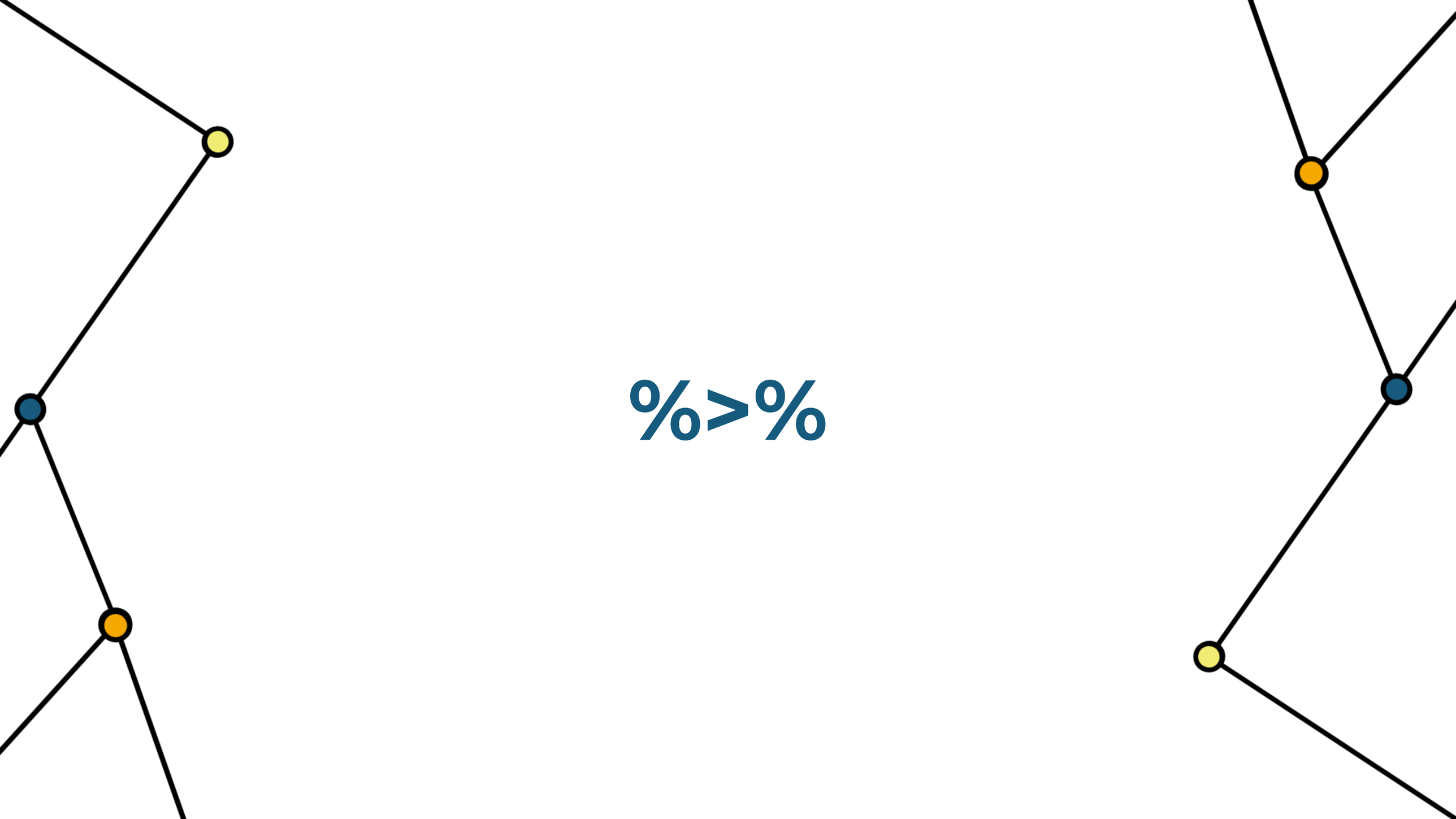
year	sex	name	n	prop
1880	F	Mary	7065	0.072
1880	F	Anna	2604	0.027
1880	F	Emma	2003	0.021
1880	F	Elizabeth	1939	0.020
1880	F	Minnie	1746	0.018
1880	F	Ida	1472	0.015
1880	M	Jake	96	0.001

Your Turn 6

Use **desc()** to find the names with the highest **prop**

Then, use **desc()** to find the names with the highest **n**

02 : 00



Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

1. Filter babynames to just boys born in 2015
2. Select the name and n columns from the result
3. Arrange those columns so that the most popular names appear near the top.

Steps

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

Steps

```
arrange(select(filter(babynames, year == 2015,
  sex == "M"), name, n), desc(n))
```

The pipe operator %>%

Passes result on left into first argument of function on right.
So, for example, these do the same thing. Try it.

```
filter(babynames, n == 99680)
babynames %>% filter(n == 99680)
```

Steps

```
babynames
boys_2015 <- filter(babynames, year == 2015, sex == "M")
boys_2015 <- select(boys_2015, name, n)
boys_2015 <- arrange(boys_2015, desc(n))
boys_2015
```

```
babynames %>%
  filter(year == 2015, sex == "M") %>%
  select(name, n) %>%
  arrange(desc(n))
```

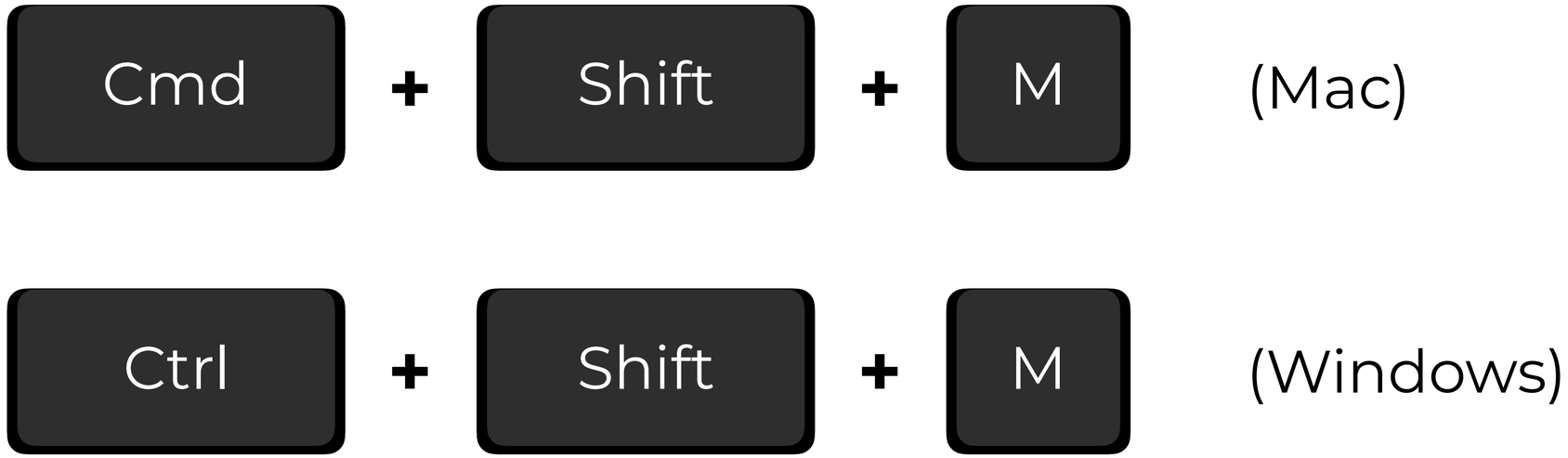
```
foo_foo <- little_bunny()
```

```
foo_foo %>%
  hop_through(forest) %>%
  scoop_up(field_mouse) %>%
  bop_on(head)
```

vs.

```
foo_foo2 <- hop_through(foo_foo, forest)
foo_foo3 <- scoop_up(foo_foo2, field_mouse)
bop_on(foo_foo3, head)
```

Keyboard shortcut



Your Turn 7

Use **%>%** to write a sequence of functions that:

1. Filter babynames to just the girls that were born in 2015
2. Select the **name** and **n** columns
3. Arrange the results so that the most popular names are near the top

05 : 00

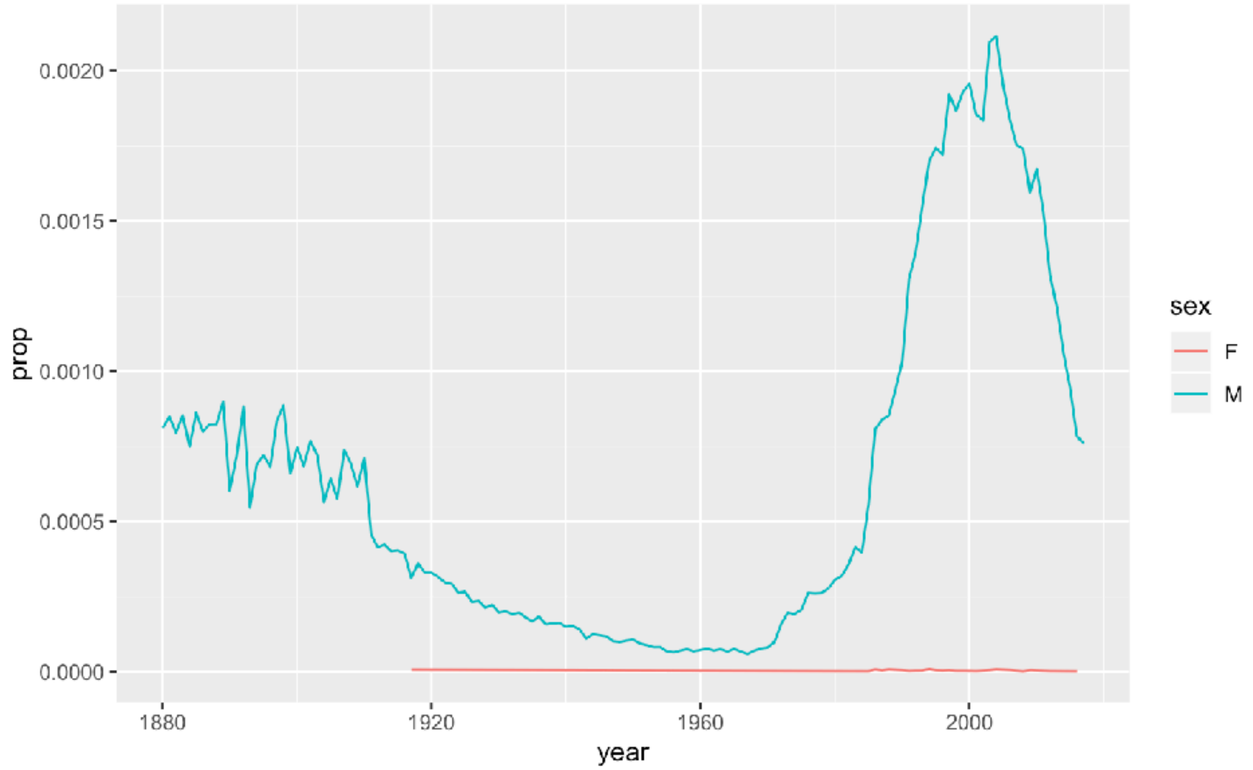
```
babynames %>%
  filter(year == 2015, sex == "F") %>%
  select(name, n) %>%
  arrange(desc(n))
#    name           n
#  1 Emma       20435
#  2 Olivia     19669
#  3 Sophia     17402
#  4 Ava        16361
#  5 Isabella   15594
#  6 Mia        14892
#  7 Abigail    12390
#  8 Emily      11780
#  9 Charlotte  11390
# 10 Harper     10291
# ... with 19,064 more rows
```

Your Turn 8

1. Trim babynames to just the rows that contain your **name**
2. Trim the result to just the columns that twill appear in your graph (not strictly necessary, but useful in practice)
3. Plot the results as a line graph with **year** on the x-axis and **prop** on the y-axis, colored by **sex**

05 : 00

```
babynames %>%
  filter(name == "Jake") %>%
  select(year, prop, sex) %>%
  ggplot(mapping = aes(x = year, y = prop)) +
    geom_line(mapping = aes(color = sex))
```



What are the most popular names?

What is popularity?

A name is popular if:

1. **Sums** - a large number of children have the name when you sum across years
2. **Ranks** - it consistently ranks among the top names from year to year.

Consider

Do we have enough information to:

1. Calculate the total number of children with each name?
2. Rank the names within each year?

Deriving information

Make table of summaries with **`summarize()`**

Make new variables with **`mutate()`**

American and British spellings accepted!

`summarize()` / `summarise()`

summarize()

Compute table of summaries

```
babynames %>% summarize(total = sum(n), max = max(n))
```

year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020
1880	F	Ida	1472	0.015

total	max
348120517	99686

Your Turn 9

Use summarize() to compute three statistics *about the data*:

1. The first (minimum) year in the dataset
2. The last (maximum) year in the dataset
3. The total number of children represented in the data

03 : 00

```
babynames %>%
  summarize(first = min(year),
            last = max(year),
            total = sum(n))
#    first  last      total
#  1  1880  2017  348120517
```

Your Turn 10

Extract the rows where **name == "Khaleesi"**. Then use summarize() to find:

1. The total number of children named Khaleesi
2. The first **year** Khaleesi appeared in the data

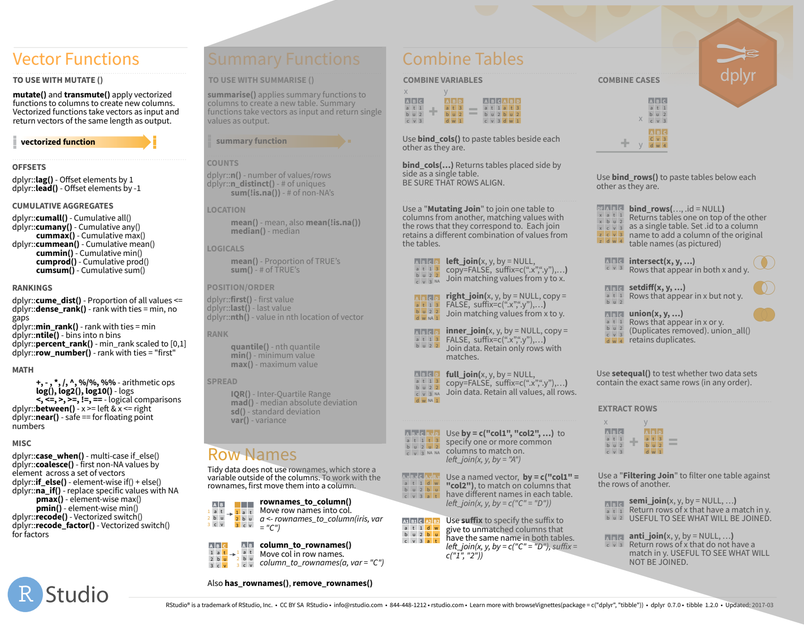
03 : 00

```
babynames %>%
  filter(name == "Khaleesi") %>%
  summarize(total = sum(n), first = min(year))
#    total  first
#  1  1964   2011
```

Summary functions

Take a vector as input.

Return a single value as output.



Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

n()

The number of rows in a dataset/group

```
babynames %>% summarize(n = n())
```

year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020
1880	F	Ida	1472	0.015

n
1924665

n_distinct()

The number of distinct values in a variable

```
babynames %>% summarize(n = n(), nname = n_distinct(name))
```

year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020
1880	F	Ida	1472	0.015

n	nname
1924665	97310

group_by()

Groups cases by common values of one or more columns

```
babynames %>%
  group_by(sex)
# A tibble: 1,924,665 x 5
# Groups:   sex [2]
```

group_by()

Groups cases by common values

```
babynames %>%
  group_by(sex) %>%
  summarize(total = sum(n))
#    sex       total
# 1  F    172371079
# 2  M    175749438
```

Toy Data for transforming

```
pollution <- tribble(
       ~city,   ~size, ~amount,
  "New York", "large",      23,
  "New York", "small",      14,
    "London", "large",      22,
    "London", "small",      16,
   "Beijing", "large",     121,
   "Beijing", "small",      56
)
```

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

```
pollution %>%
  summarize(mean = mean(amount), sum = sum(amount), n = n())
```

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

group_by() + summarize()

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

```
pollution %>%
  group_by(city) %>%
  summarize(mean = mean(amount), sum = sum(amount), n = n())
```

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	size	amount
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

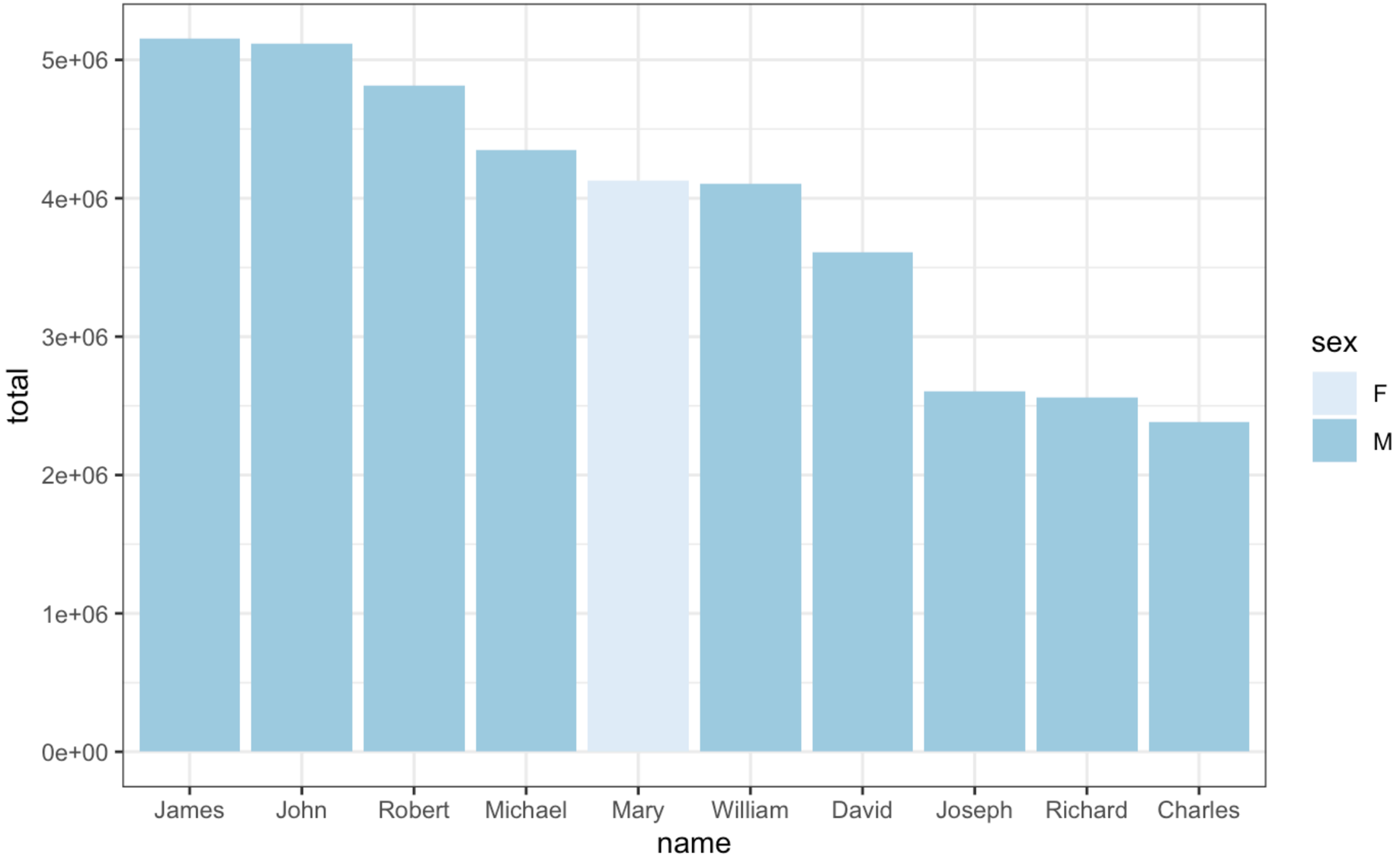
city	mean	sum	n
New York	18.5	37	2
London	19.0	38	2
Beijing	88.5	177	2

Your Turn 11

Use **group_by()**, **summarize()**, and **arrange()** to display the ten most popular baby names. Compute popularity as the total number of children of a single gender, given a name.

05 : 00

```
babynames %>%
  group_by(name, sex) %>%
  summarize(total = sum(n)) %>%
  arrange(desc(total))
#    name    sex      total
#  1 James   M      5150472
#  2 John    M      5115466
#  3 Robert  M      4814815
#  4 Michael M      4350824
#  5 Mary    F      4123200
#  6 William M      4102604
#  7 David   M      3611329
#  8 Joseph  M      2603445
#  9 Richard M      2563082
# 10 Charles M      2386048
# ... with 107,963 more rows
```



```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total)) %>%
  ungroup() %>%
  slice(1:10) %>%
  ggplot() +
    geom_col(mapping = aes(x = fct_reorder(name, desc(total)), y = total,
                           fill = sex)) +
    theme_bw() +
    scale_fill_brewer() +
    labs(x = "name")
```

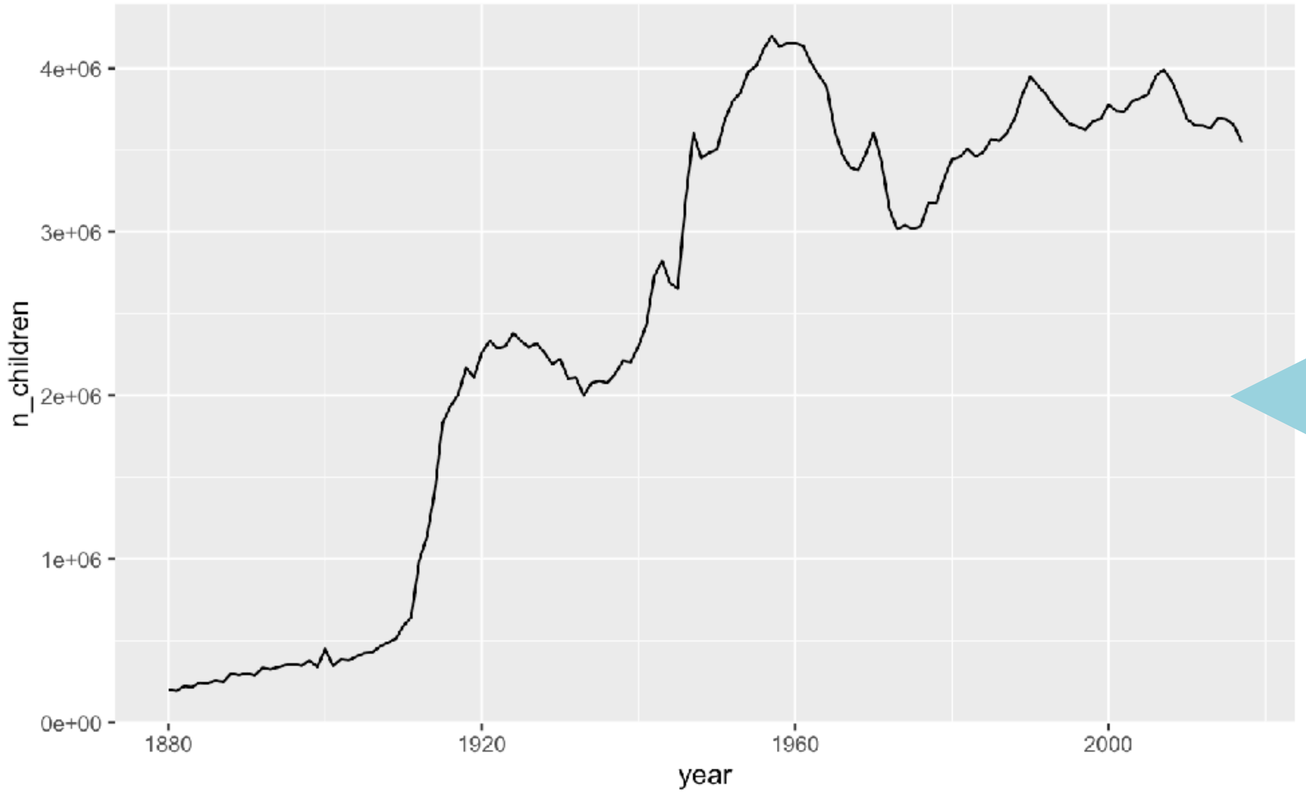
Your Turn 12

Use grouping to calculate the **number of children born each year**.

Plot the results as a line graph.

05 : 00

```
babynames %>%
  group_by(year) %>%
  summarize(n_children = sum(n)) %>%
  ggplot() +
    geom_line(mapping = aes(x = year, y = n_children))
```



How does this affect our measure of popularity?

mutate()

Create new columns

```
babynames %>%
  mutate(percent = round(prop * 100, 2))
```

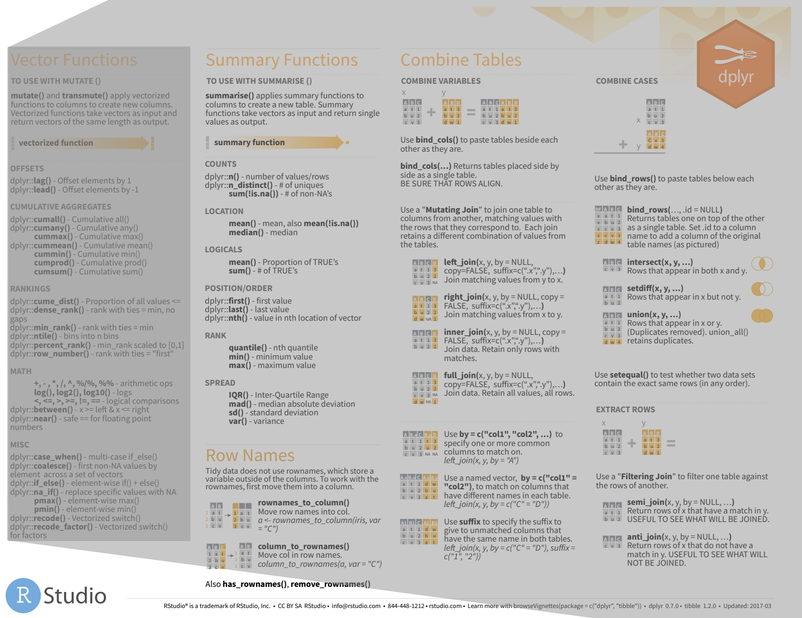
year	sex	name	n	prop
1880	F	Minnie	1746	0.018
1880	F	Mary	7065	0.072
1880	F	Emma	2003	0.021
1880	M	Jake	96	0.001
1880	F	Anna	2604	0.027
1880	F	Elizabeth	1939	0.020

year	sex	name	n	prop	percent
1880	F	Minnie	1746	0.018	1.79
1880	F	Mary	7065	0.072	7.24
1880	F	Emma	2003	0.021	2.05
1880	M	Jake	96	0.001	0.08
1880	F	Anna	2604	0.027	2.67
1880	F	Elizabeth	1939	0.020	1.99

Vector functions

Take a vector as input.

Return a vector of the same length as output.



Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

min_rank()

A go to ranking function (ties share the lowest rank)

```
min_rank(c(50, 100, 1000))
# [1] 1 2 3
```

```
min_rank(desc(c(50, 100, 1000)))
# [1] 3 2 1
```

Your Turn 13

Use **min_rank()** and **mutate()** to rank each row in babynames from largest **prop** to lowest **prop**.

```
babynames %>%
  mutate(rank = min_rank(desc(prop)))
#      year sex   name           n   prop  rank
#   1  1880 F     Mary        7065 0.0724    14
#   2  1880 F     Anna        2604 0.0267   709
#   3  1880 F     Emma        2003 0.0205  1131
#   4  1880 F     Elizabeth   1939 0.0199  1192
#   5  1880 F     Minnie      1746 0.0179  1427
#   6  1880 F     Margaret    1578 0.0162  1683
#   7  1880 F     Ida         1472 0.0151  1897
#   8  1880 F     Alice       1414 0.0145  2039
#   9  1880 F     Bertha      1320 0.0135  2279
# 10  1880 F     Sarah       1288 0.0132  2387
# ... with 1,924,655 more rows
```

```
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop)))
#      year sex   name           n   prop  rank
#   1  1880 F     Mary        7065 0.0724     1
#   2  1880 F     Anna        2604 0.0267     2
#   3  1880 F     Emma        2003 0.0205     3
#   4  1880 F     Elizabeth   1939 0.0199     4
#   5  1880 F     Minnie      1746 0.017      5
#   6  1880 F     Margaret    1578 0.0162     6
#   7  1880 F     Ida         1472 0.0151     7
#   8  1880 F     Alice       1414 0.0145     8
#   9  1880 F     Bertha      1320 0.0135     9
# 10  1880 F     Sarah       1288 0.0132    10
# ... with 1,924,655 more rows
```

Your Turn 14

What was each name's median rank?

i.e., compute each name's rank *within its year and sex*.

Then compute the median rank for each combination of *name* and *sex*, and arrange the results from highest median rank to lowest.

05 : 00

```
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  group_by(name, sex) %>%
  summarize(score = median(rank)) %>%
  arrange(score)
#    name      sex   score
#  1 Mary      F       1
#  2 James     M       3
#  3 John      M       3
#  4 William   M       4
#  5 Robert    M       6
#  6 Michael   M       7.5
#  7 Charles   M       9
#  8 Elizabeth F      10
#  9 Joseph    M      10
# 10 Thomas    M      11
# ... with 107,963 more rows
```

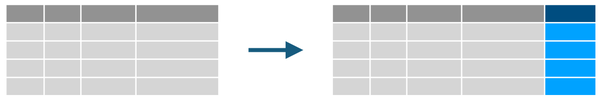
Recap: Single table verbs

Extract variables with **`select()`**

Extract cases with **`filter()`**

Arrange cases with **`arrange()`**

Make table of summaries with **`summarize()`**



Make new variables with **`mutate()`**

Joining Datasets

In R4DS

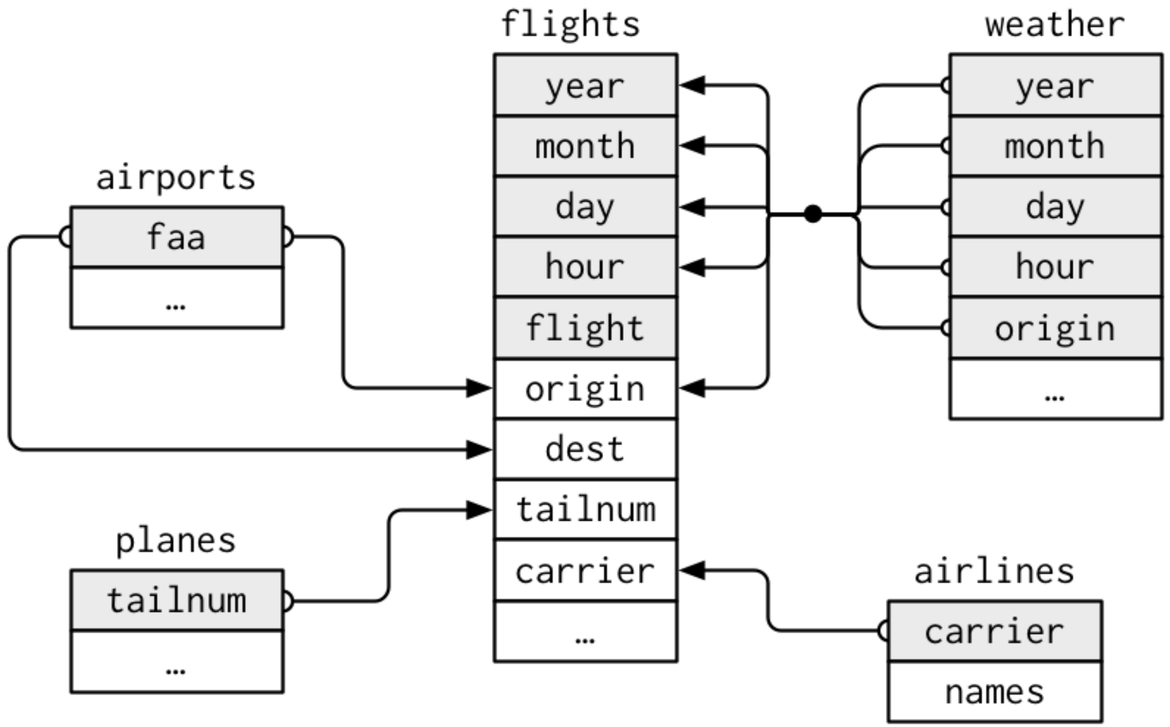
Relational Data

nycflights13

```
# install.packages("nycflights13")
library(nycflights13)
View(flights)
```

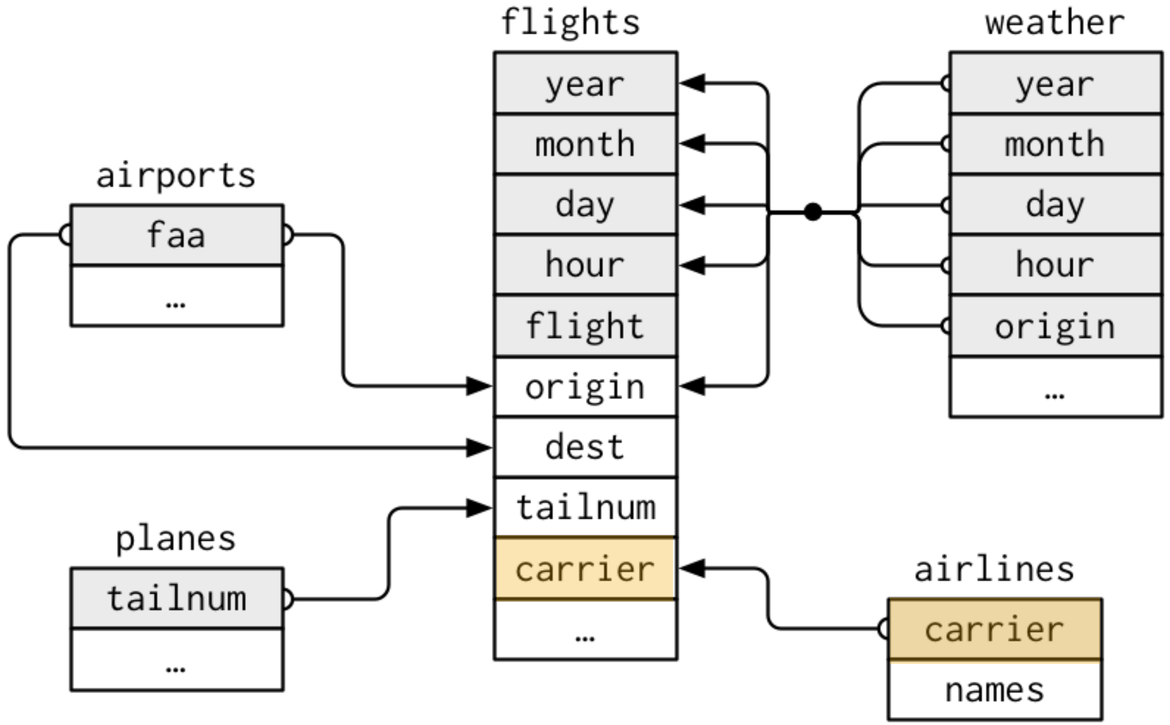
	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance	hour	minute	time_hour
1	2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5	15	2013-01-01 05:00:00
2	2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5	29	2013-01-01 05:00:00
3	2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5	40	2013-01-01 05:00:00
4	2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5	45	2013-01-01 05:00:00
5	2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6	0	2013-01-01 06:00:00
6	2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5	58	2013-01-01 05:00:00
7	2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6	0	2013-01-01 06:00:00
8	2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6	0	2013-01-01 06:00:00
9	2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6	0	2013-01-01 06:00:00
10	2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6	0	2013-01-01 06:00:00
11	2013	1	1	558	600	-2	849	851	-2	B6	49	N793JB	JFK	PBI	149	1028	6	0	2013-01-01 06:00:00
12	2013	1	1	558	600	-2	853	856	-3	B6	71	N657JB	JFK	TPA	158	1005	6	0	2013-01-01 06:00:00
13	2013	1	1	558	600	-2	924	917	7	UA	194	N29129	JFK	LAX	345	2475	6	0	2013-01-01 06:00:00
14	2013	1	1	558	600	-2	923	937	-14	UA	1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00
15	2013	1	1	559	600	-1	941	910	31	AA	707	N3DUAA	LGA	DFW	257	1389	6	0	2013-01-01 06:00:00
16	2013	1	1	559	559	0	702	706	-4	B6	1806	N708JB	JFK	BOS	44	187	5	59	2013-01-01 05:00:00
17	2013	1	1	559	600	-1	854	902	-8	UA	1187	N76515	EWR	LAS	337	2227	6	0	2013-01-01 06:00:00

nycflights13



nycflights13

Which airline had the longest delays?



Airline names

```
View(flights["carrier"])
```

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6
8	EV
9	B6
10	AA

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.
8	FL	AirTran Airways Corporation
9	HA	Hawaiian Airlines Inc.
10	MQ	Envoy Air

Types of joins

Mutating joins use information from one data set **to add variables** to another data set (like **mutate()**)

Filtering joins use information from one data set **to extract cases** from another data set (like **filter()**)

Mutating joins

Each join function returns a data frame / tibble.

```
left_join(x, y, by = NULL, ...)
```

join function

data frames to join

names of columns to join on

Toy Data for practice

```
band <- tribble(
   ~name,     ~band,
  "Mick",  "Stones",
  "John", "Beatles",
  "Paul", "Beatles"
)
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

```
instrument <- tribble(
   ~name,    ~plays,
   "John", "guitar",
   "Paul",   "bass",
  "Keith", "guitar"
)
```

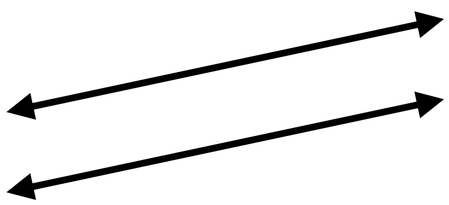
instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

Toy Data for practice

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles



instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

left_join

```
band %>% left_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

right_join

```
band %>% right_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

full_join

```
band %>% full_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

inner_join

```
band %>% inner_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

Airline names

```
View(flights["carrier"])
```

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6
8	EV
9	B6
10	AA

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.
8	FL	AirTran Airways Corporation
9	HA	Hawaiian Airlines Inc.
10	MQ	Envoy Air

Your Turn 15

Which airlines had the largest arrival delays? Complete the code below.

```
flights %>%
  drop_na(arr_delay) %>%
  _____________________ %>%
  group_by(___________) %>%
  _____________________ %>%
  arrange(_______)
```

1. Join airlines to flights

2. Compute and order the average arrival delays by airline. Display full names, no codes.

06 : 00

```
flights %>%
  drop_na(arr_delay) %>%
  left_join(airlines, by = "carrier") %>%
  group_by(name) %>%
  summarize(delay = mean(arr_delay)) %>%
  arrange(delay)
#   name                          delay
# 1 Alaska Airlines Inc.         -9.93
# 2 Hawaiian Airlines Inc.       -6.92
# 3 American Airlines Inc.        0.364
# 4 Delta Air Lines Inc.          1.64
# 5 Virgin America                1.76
# 6 US Airways Inc.               2.13
# 7 United Air Lines Inc.         3.56
```

Toy Data for practice

```
band <- tribble(
   ~name,     ~band,
  "Mick",  "Stones",
  "John", "Beatles",
  "Paul", "Beatles"
)
```

```
instrument2 <- tribble(
  ~artist,    ~plays,
   "John", "guitar",
   "Paul",   "bass",
  "Keith", "guitar"
)
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

instrument2

artist	plays
John	guitar
Paul	bass
Keith	guitar

Non-matching names

Use a named vector to match on variables with different names.

```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

A named vector

The name of the element = the column name in the first data set

The value of the element = the column name in the second data set

Matching names

```
band %>% left_join(instrument2, by = c("name" = "artist"))
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument2

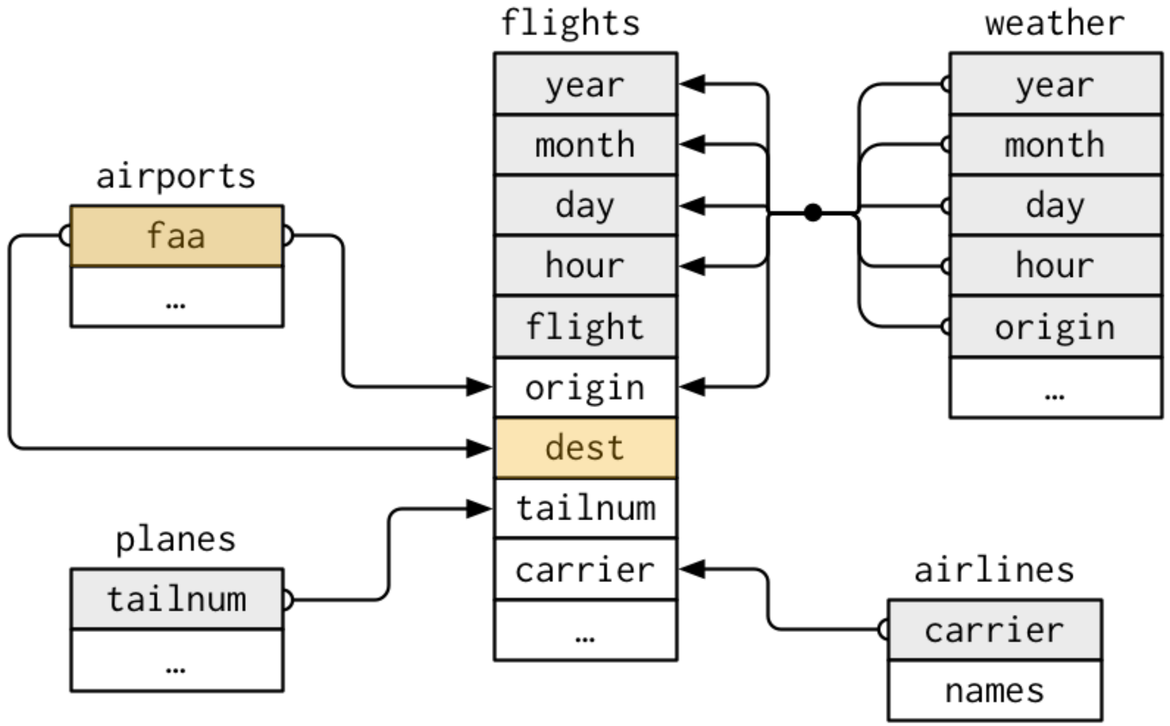
artist	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

Airport names

```
airports %>% left_join(flights, by = c("faa" = "dest"))
```



Filtering joins

Mutating joins use information from one data set **to add variables** to another data set (like **mutate()**)

Filtering joins use information from one data set **to extract cases** from another data set (like **filter()**)

semi_join

```
band %>% semi_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band
John	Beatles
Paul	Beatles

anti_join

```
band %>% anti_join(instrument, by = "name")
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band
Mick	Stones

Your Turn 16

How many airports in **airports** are serviced by flights originating in New York (i.e., flights in our dataset)?

Notice that the column to join on is named **faa** in the **airports** dataset and **dest** in the **flights** dataset.

05 : 00

```
airports %>%
  semi_join(flights, by = c("faa" = "dest")) %>%
  distinct(faa)
#    faa
#  1 ABQ
#  2 ACK
#  3 ALB
#  4 ANC
#  5 ATL
#  6 AUS
#  7 AVL
#  8 BDL
#  9 BGR
# 10 BHM
# ... with 91 more rows
```

distinct()

Removes rows with duplicate values (in a column).

```
distinct(instrument, plays)
```

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

plays
guitar
bass

Recap: Two table verbs

`left_join()` retains all cases in **left** data set

`right_join()` retains all cases in **right** data set

`full_join()` retains all cases in **either** data set

`inner_join()` retains all cases in **both** data set

`semi_join()` extracts cases that **have a match**

`anti_join()` extracts cases that **do not have a match**

Two table verbs

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , * , /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

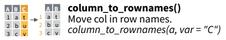
quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.



rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

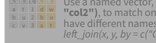
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.

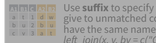
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")



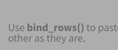
Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))



Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

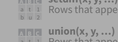
COMBINE CASES

Use **bind_rows()** to paste tables below each other as they are.



bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.



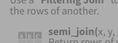
setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

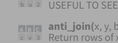
Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

Use a "**Filtering Join**" to filter one table against the rows of another.



semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.



anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio

Combine Tables

dplyr

COMBINE VARIABLES

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

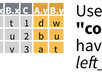
left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join matching values from x to y.

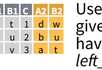
inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…)
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…)
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

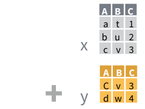


Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

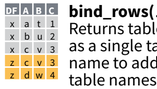


Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

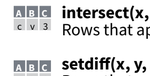
COMBINE CASES



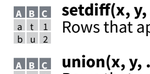
Use **bind_rows()** to paste tables below each other as they are.



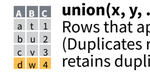
bind_rows(…, .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)



intersect(x, y, …)
Rows that appear in both x and y.



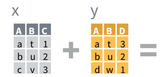
setdiff(x, y, …)
Rows that appear in x but not y.



union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

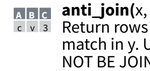
Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

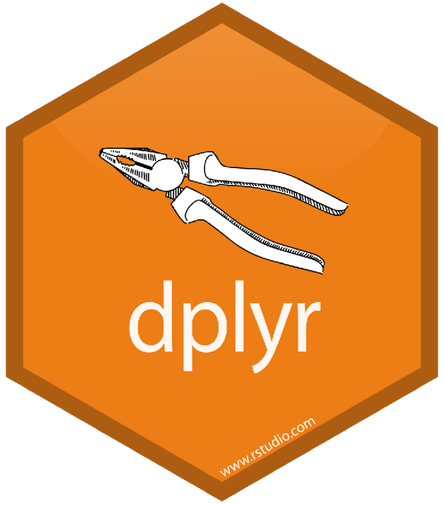
semi_join(x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.



anti_join(x, y, by = NULL, …)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

dplyr

Data Transformation



ATLAS
Accessible Teaching, Learning, and Assessment Systems

wjakethompson.com wjakethompson@ku.edu @wjakethompson @wjakethompson