

# Transform Data with



# Exercise: What do you already know?

- Split into groups of 2
- Ask your partner:
  - *What do you already know about dplyr?*
  - *What type of "data transformations" do you normally do?*
- 5 minutes



dplyr

## Main Ideas

Notes

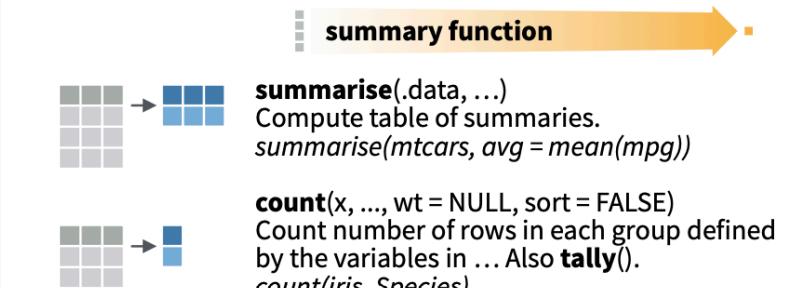
# Data Transformation with dplyr :: **CHEAT SHEET**

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



## 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).



## VARIATIONS

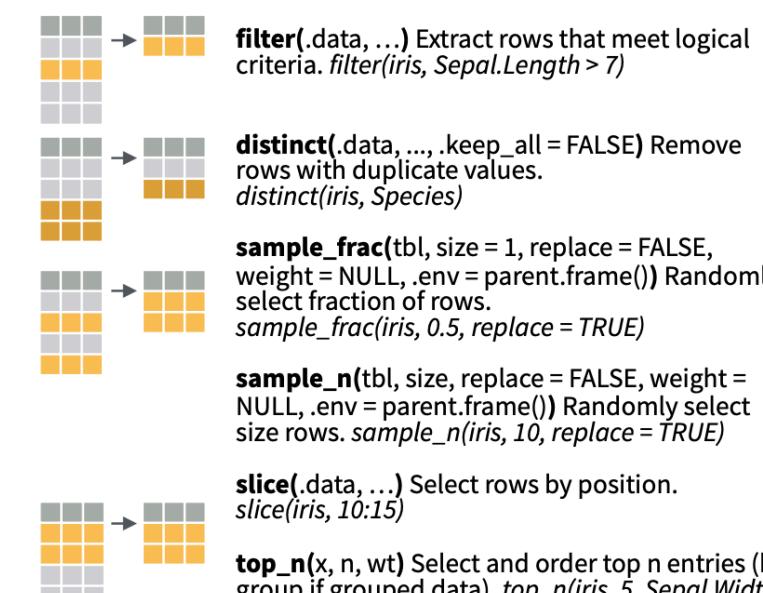
**summarise\_all()** - Apply funs to every column.  
**summarise\_at()** - Apply funs to specific columns.  
**summarise\_if()** - Apply funs to all cols of one type.

## Group Cases

## Manipulate Cases

## **EXTRACT CASES**

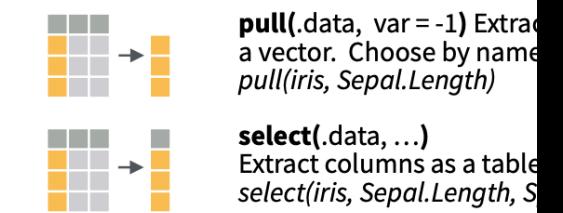
Row functions return a subset of rows as a new table.



## Manipulate Variables

## EXTRACT VARIABLES

Column functions return a set of columns



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

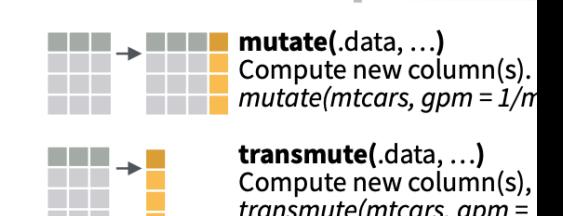
**contains(match)**    **num\_range(prefix, r)**  
**ends\_with(match)**    **one\_of(...)**

**matches(match)**      **starts\_with(match)**

.....

## MAKE NEW VARIABLES

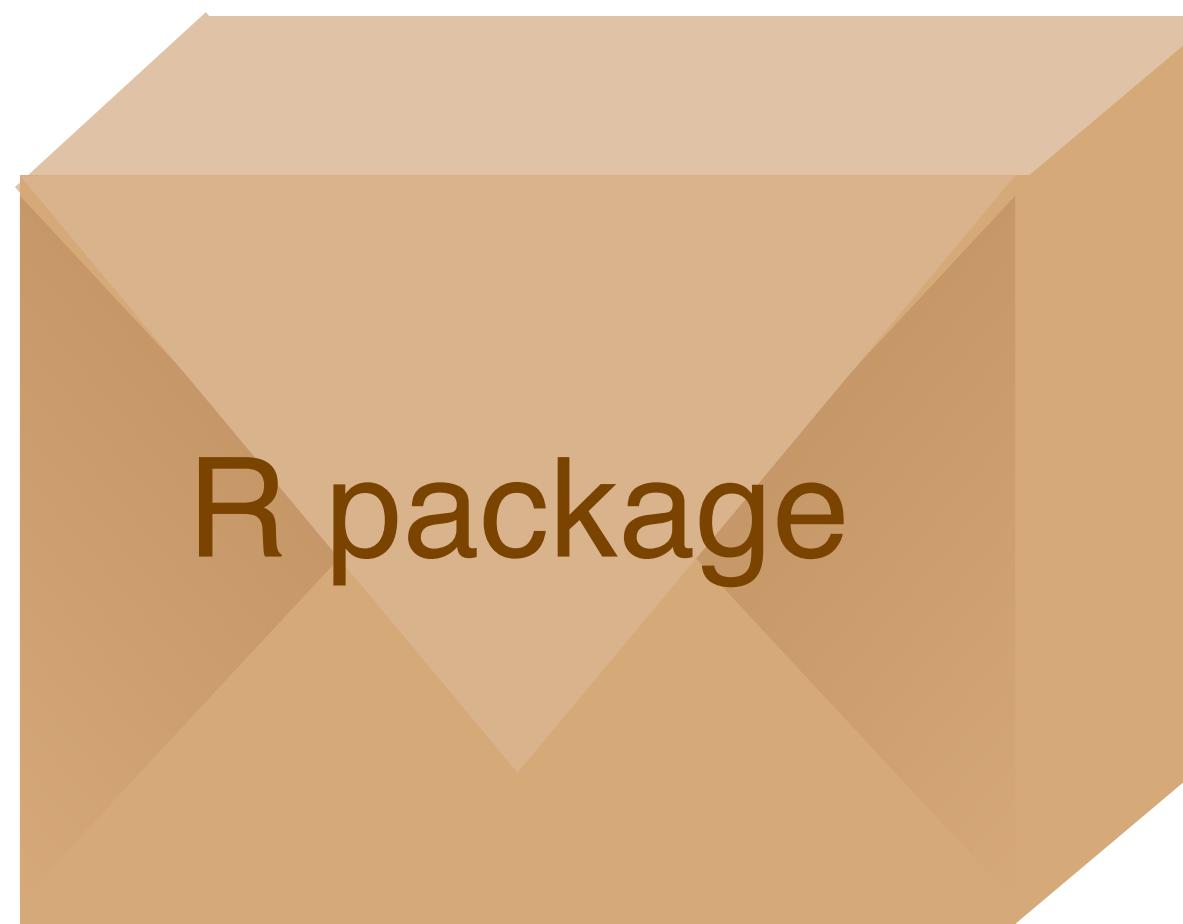
These apply **vectorized functions** to column vectors as input and return vectors of the same length (see back).



# Notes form & Cheat Sheet

# Please take out

# babynames



R package

Names of male and female babies born  
in the US from 1880 to 2015. 1.8M rows.

```
# install.packages("babynames")
library(babynames)
```



dplyr

# babynames

year	sex	name	n	prop
<dbl>	<chr>	<chr>	<int>	<dbl>
1880	F	Mary	7065	7.238433e-02
1880	F	Anna	2604	2.667923e-02
1880	F	Emma	2003	2.052170e-02
1880	F	Elizabeth	1939	1.986599e-02
1880	F	Minnie	1746	1.788861e-02
1880	F	Margaret	1578	1.616737e-02
1880	F	Ida	1472	1.508135e-02
1880	F	Alice	1414	1.448711e-02
1880	F	Bertha	1320	1.352404e-02
1880	F	Sarah	1288	1.319618e-02

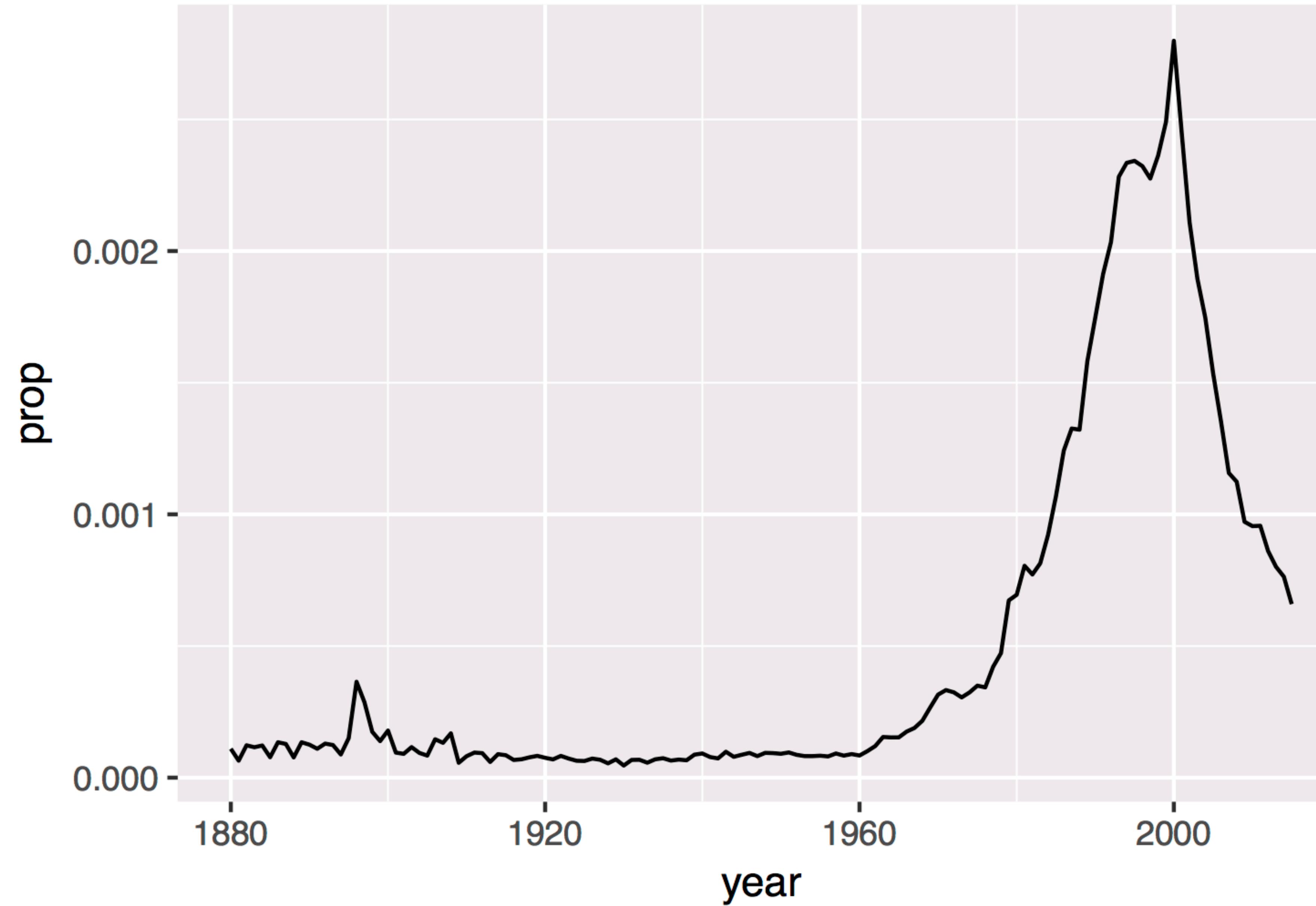
1-10 of 1,858,689 rows

Previous

1 2 3 4 5 6 ... 100 Next



# Proportion of boys with the name Garrett



# How to isolate?

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081
1881	M	William	8524	0.0787
1881	M	James	5442	0.0503
1881	M	Charles	4664	0.0431
1881	M	Garrett	7	0.0001
1881	M	Gideon	7	0.0001



year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
...	...	Garrett	...	...



# dplyr



# dplyr



A package that transforms data.

dplyr implements a *grammar* for transforming tabular data.



# Isolating data

**select()** - extract **variables (columns)**

**filter()** - extract **cases (rows)**

**arrange()** - reorder **cases**



# select()



# 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)
```

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

name	prop
John	0.0815
William	0.0805
James	0.0501
Charles	0.0451
Garrett	0.0001
John	0.081



# Your Turn 1

Open up **02-dplyr-exercises.Rmd**

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

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



```
select(babynames, n)
```

```
#       n  
# <int>  
# 1 7065  
# 2 2604  
# 3 2003  
# 4 1939  
# 5 1746  
# ... ...
```



```
> name  
Error: object 'name' not found  
> select(babynames, name)  
# A tibble: 1,924,665 x 1  
  name  
  <chr>  
1 Mary  
2 Anna  
3 Emma  
4 Elizabeth
```

# Non Standard Evaluation (NSE)

Using Column Names as Parameters

# select() helpers

:- Select range of columns

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

-- Select every column but

```
select(storms, -c(storm, 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

**Data Transformation with dplyr Cheat Sheet**

R Studio

**Manipulate Cases**

**Extract Cases**

Row functions return a subset of rows as a new table. Use a variant that ends in `_` for non-standard evaluation friendly code.

- `filter(data, ...)` Extract rows that meet logical criteria. Also `filter_()`. `filter(iris, Sepal.Length > 7)`
- `distinct(data, ..., keep_all = FALSE)` Remove rows with duplicate values. Also `distinct_()`. `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. Also `slice_()`. `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()`

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

**Manipulate Variables**

**Extract Variables**

Column functions return a set of columns as a new table. Use a variant that ends in `_` for non-standard evaluation friendly code.

- `select(data, ...)` Extract columns by name. Also `select_if()`. `select(iris, Sepal.Length, Species)`
- `use these helpers with select(), e.g. select(iris, starts_with("Sepal"))`

**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).

- `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()`. `mutate_all(faithful, funs(log(), log2(.)))`
- `mutate_at(tbl, .cols, funs, ...)` Apply funs to specific columns. Use with `funs()` and the helper functions for `select()`. `mutate_at(iris, -Species, funs(log(.)))`
- `mutate_if(tbl, predicate, funs, ...)` Apply funs to all columns of one type. Use with `funs()`. `mutate_if(iris, is.numeric, funs(log(.)))`
- `add_column(data, ..., before = NULL, after = NULL)` Add new column(s). `add_column(mtcars, new = 1:32)`
- `rename(data, ...)` Rename columns. `rename(iris, Length = Sepal.Length)`

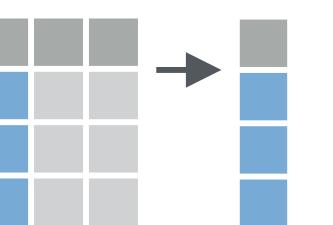
Learn more with `browseVignettes(package = c("dplyr", "tibble"))` • dplyr 0.5.0 • tibble 1.2.0 • Updated: 11/16

## Extract Variables

Column functions return a set of columns as a new table. Use a variant that ends in `_` for non-standard evaluation friendly code.

### select(.data, ...)

Extract columns by name. Also `select_if()`, `select(iris, Sepal.Length, Species)`



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

`contains(match)`  
`ends_with(match)`  
`matches(match)`

`num_range(prefix, range)` ::, e.g. `mpg:cyl`  
`one_of(...)` -, e.g., `-Species`  
`starts_with(match)`



# Quiz

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"))`

# Quiz

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()



# filter()

Extract rows that meet logical criteria.

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

**data frame to transform**

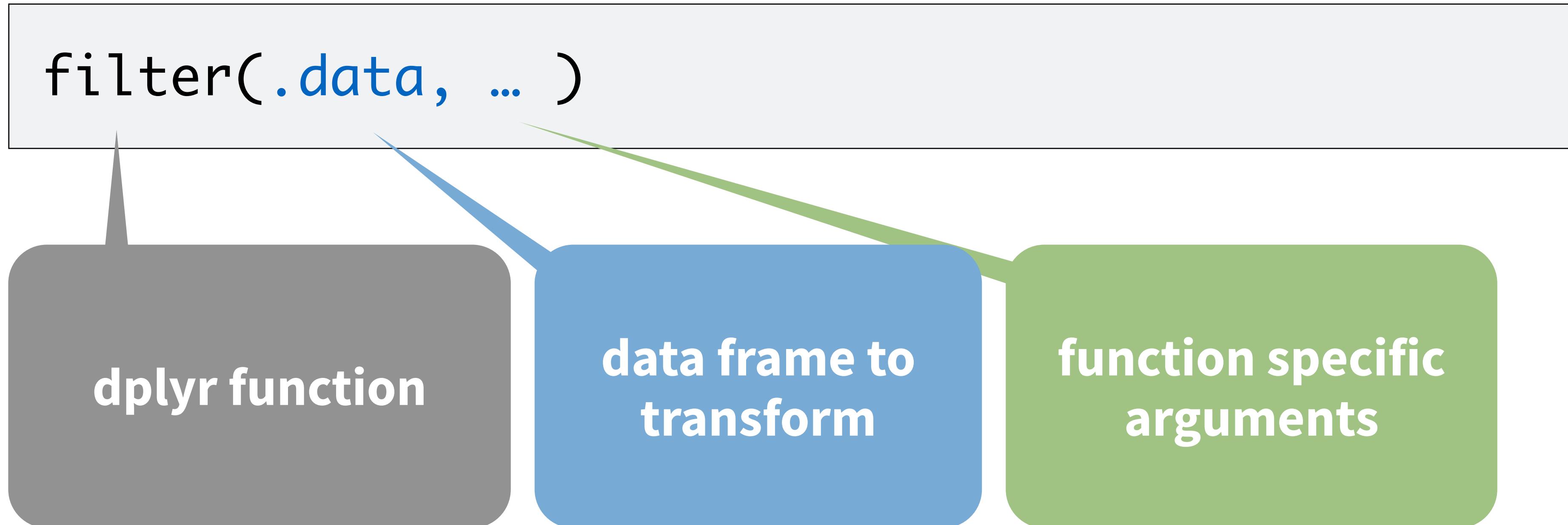
**one or more logical tests**  
(filter returns each row for which the test is TRUE)



# common syntax

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

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



# filter()

Extract rows that meet logical criteria.

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

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
...	...	Garrett	...	...



# filter()

Extract rows that meet logical criteria.

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

babynames					
year	sex	name	n	prop	
1880	M	John	9655	0.0815	
1880	M	William	9532	0.0805	
1880	M	James	5927	0.0501	
1880	M	Charles	5348	0.0451	
1880	M	Garrett	13	0.0001	
1881	M	John	8769	0.081	

= sets  
(returns nothing)  
== tests if equal  
(returns TRUE or FALSE)



# Logical tests

## ?Comparison

<code>x &lt; y</code>	Less than
<code>x &gt; y</code>	Greater than
<code>x == y</code>	Equal to
<code>x &lt;= y</code>	Less than or equal to
<code>x &gt;= y</code>	Greater than or equal to
<code>x != y</code>	Not equal to
<code>x %in% y</code>	Group membership
<code>is.na(x)</code>	Is NA
<code>!is.na(x)</code>	Is not NA



# Your Turn 2

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 “Sea”
- All of the names that have a missing value for **n**  
(Hint: this should return an empty data set).



```
filter(babynames, prop >= 0.08)
```

```
#   year sex name    n      prop
# 1 1880 M  John 9655 0.08154630
# 2 1880 M William 9531 0.08049899
# 3 1881 M  John 8769 0.08098299
```

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

```
#   year sex name    n      prop
# 1 1982 F  Sea     5 2.756771e-06
# 2 1985 M  Sea     6 3.119547e-06
# 3 1986 M  Sea     5 2.603512e-06
# 4 1998 F  Sea     5 2.580377e-06
```

```
filter(babynames, is.na(n))
```

```
# 0 rows
```

# Two common mistakes

## 1. Using `=` instead of `==`

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

## 2. Forgetting quotes

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



# filter()

Extract rows that meet *every* logical criteria.

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

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001



# 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)



# filter()

Extract rows that meet *every* logical criteria.

```
filter(babynames, name == "Garrett" & year == 1880)
```

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop
1880	M	Garrett	13	0.0001



# Your Turn 3

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

- Girls named 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 == "Sea" | name == "Anemone")
```



```
filter(babynames, name == "Sea", sex == "F")
```

```
#   year  sex  name    n      prop
# 1 1982    F  Sea     5 2.756771e-06
# 2 1998    F  Sea     5 2.580377e-06
```

```
filter(babynames, n == 5 | n == 6, year == 1880)
```

```
#   year  sex  name    n      prop
# 1 1880    F  Abby    6 6.147289e-05
# 2 1880    F  Aileen  6 6.147289e-05
# ...    ...  ...    ...  ...  ...
```

```
filter(babynames, name == "Acura" | name == "Lexus" | name == "Yugo"))
```

```
#   year  sex  name    n      prop
# 1 1990    F  Lexus  36 1.752932e-05
# 2 1990    M  Lexus  12 5.579156e-06
# ...    ...  ...    ...  ...  ...
```

# Two more common mistakes

## 3. Collapsing multiple tests into one

```
filter(babynames, 10 < n < 20)  
filter(babynames, 10 < n, n < 20)
```

## 4. Stringing together many tests (when you could use %in%)

```
filter(babynames, n == 5 | n == 6 | n == 7 | n == 8)  
filter(babynames, n %in% c(5, 6, 7, 8))
```



```
filter(babynames, name == "Sea", sex == "F")  
#> #>   year  sex  name    n      prop  
#> # 1 1982  F  Sea    5 2.756771e-06  
#> # 2 1998  F  Sea    5 2.580377e-06
```

```
filter(babynames, n == 5 | n == 6, year == 1880)  
#> #>   year  sex  name    n      prop  
#> # 1 1880  F  Abby   6 6.147289e-05  
#> # 2 1880  F  Aileen 6 6.147289e-05  
#> # ...  ...  ...    ...  ...  ...
```

```
filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))  
#> #>   year  sex  name    n      prop  
#> # 1 1990  F  Lexus  36 1.752932e-05  
#> # 2 1990  M  Lexus  12 5.579156e-06  
#> # ...  ...  ...    ...  ...  ...
```

# arrange()



# 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)
```

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1880	M	Charles	5348	0.0451
1880	M	James	5927	0.0501
1881	M	John	8769	0.081
1880	M	William	9532	0.0805
1880	M	John	9655	0.0815



# Your Turn 4

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?



## arrange(babynames, n, prop)

```
#   year   sex      name    n      prop
# 1 2007     M     Aaban  5 2.259872e-06
# 2 2007     M     Aareon  5 2.259872e-06
# 3 2007     M     Aaris  5 2.259872e-06
# 4 2007     M       Abd  5 2.259872e-06
# 5 2007     M  Abdulazeez 5 2.259872e-06
# 6 2007     M  Abdulhadi 5 2.259872e-06
# 7 2007     M  Abdulhamid 5 2.259872e-06
# 8 2007     M  Abdulkadir 5 2.259872e-06
# 9 2007     M  Abdulraheem 5 2.259872e-06
# 10 2007     M  Abdulrahim 5 2.259872e-06
# ... with 1,858,679 more rows
```



# desc()

Changes ordering to largest to smallest.

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

babynames				
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081

→

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1881	M	John	8769	0.081
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001



# Your Turn 5

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

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



```
arrange(babynames, desc(prop))
```

```
# #   year   sex   name    n      prop
# 1 1880     M   John  9655 0.08154630
# 2 1881     M   John  8769 0.08098299
# 3 1880     M William 9531 0.08049899
# 4 1883     M   John  8894 0.07907324
# 5 1881     M William 8524 0.07872038
# 6 1882     M   John  9557 0.07831617
# 7 1884     M   John  9388 0.07648751
# 8 1882     M William 9298 0.07619375
# 9 1886     M   John  9026 0.07582198
# 10 1885    M   John  8756 0.07551791
# ... with 1,858,679 more rows
```

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

```
# #   year   sex   name    n      prop
# 1 1947     F Linda 99680 0.05483609
# 2 1948     F Linda 96211 0.05521159
# 3 1947     M James 94763 0.05102057
# 4 1957     M Michael 92726 0.04238659
# 5 1947     M Robert 91646 0.04934237
# 6 1949     F Linda 91010 0.05184281
# 7 1956     M Michael 90623 0.04225479
# 8 1958     M Michael 90517 0.04203881
# 9 1948     M James 88588 0.04969679
# 10 1954    M Michael 88493 0.04279403
# ... with 1,858,679 more rows
```



%>%

R

# 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 the results so the *n* column is descending

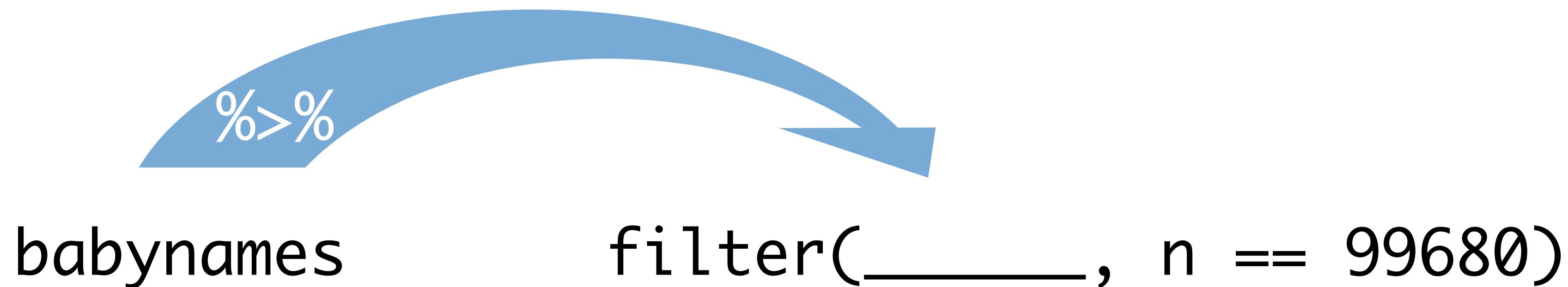
# Steps

```
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)
```



# Pipes

```
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)
```

# Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)



# Your Turn 6

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.



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

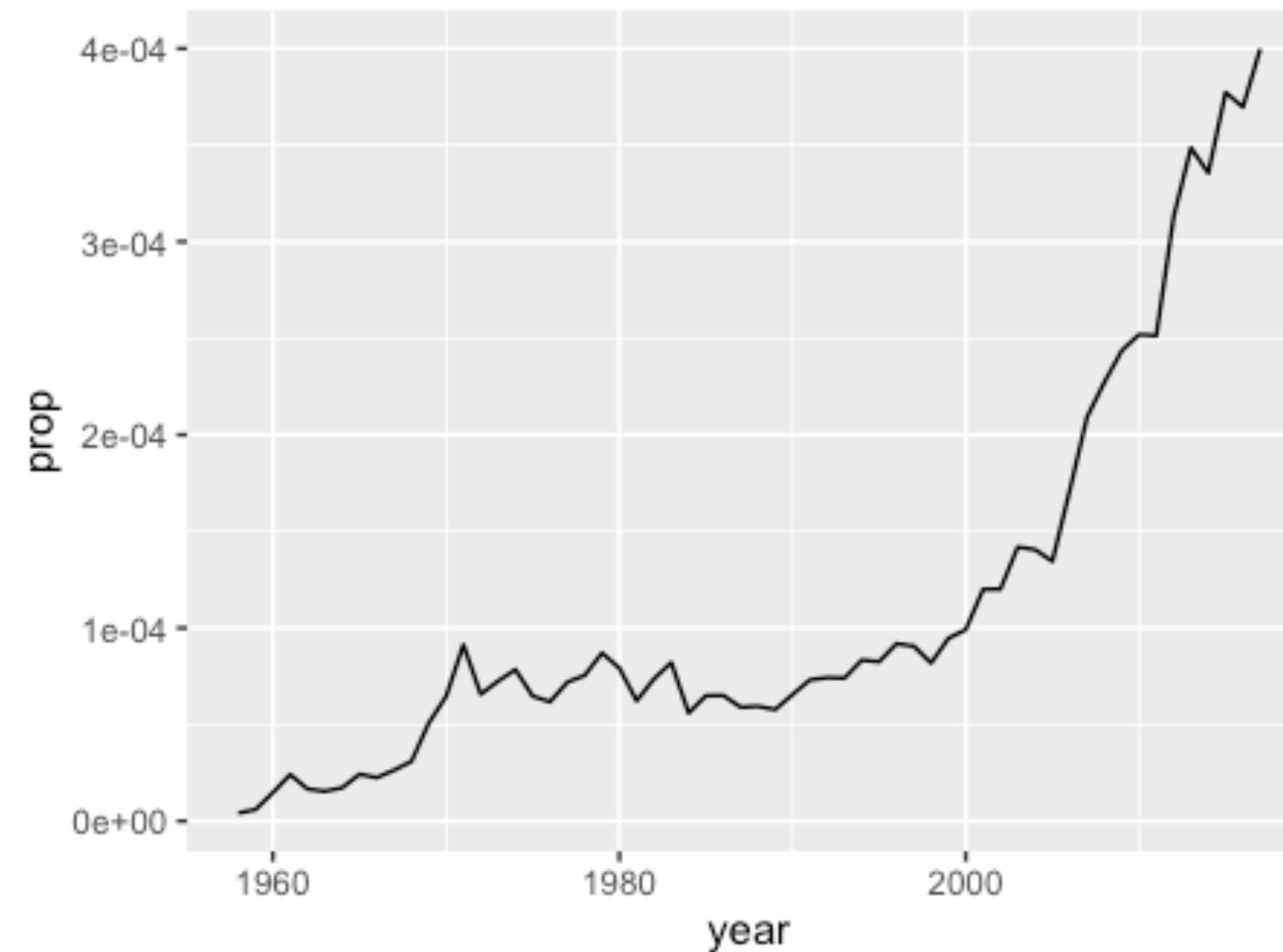
```
#          name     n  
# 1      Emma 20355  
# 2      Olivia 19553  
# 3      Sophia 17327  
# 4        Ava 16286  
# 5    Isabella 15504  
# 6        Mia 14820  
# 7    Abigail 12311  
# 8        Emily 11727  
# 9   Charlotte 11332  
# 10       Harper 10241  
# ... with 18,983 more rows
```

# Exam

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



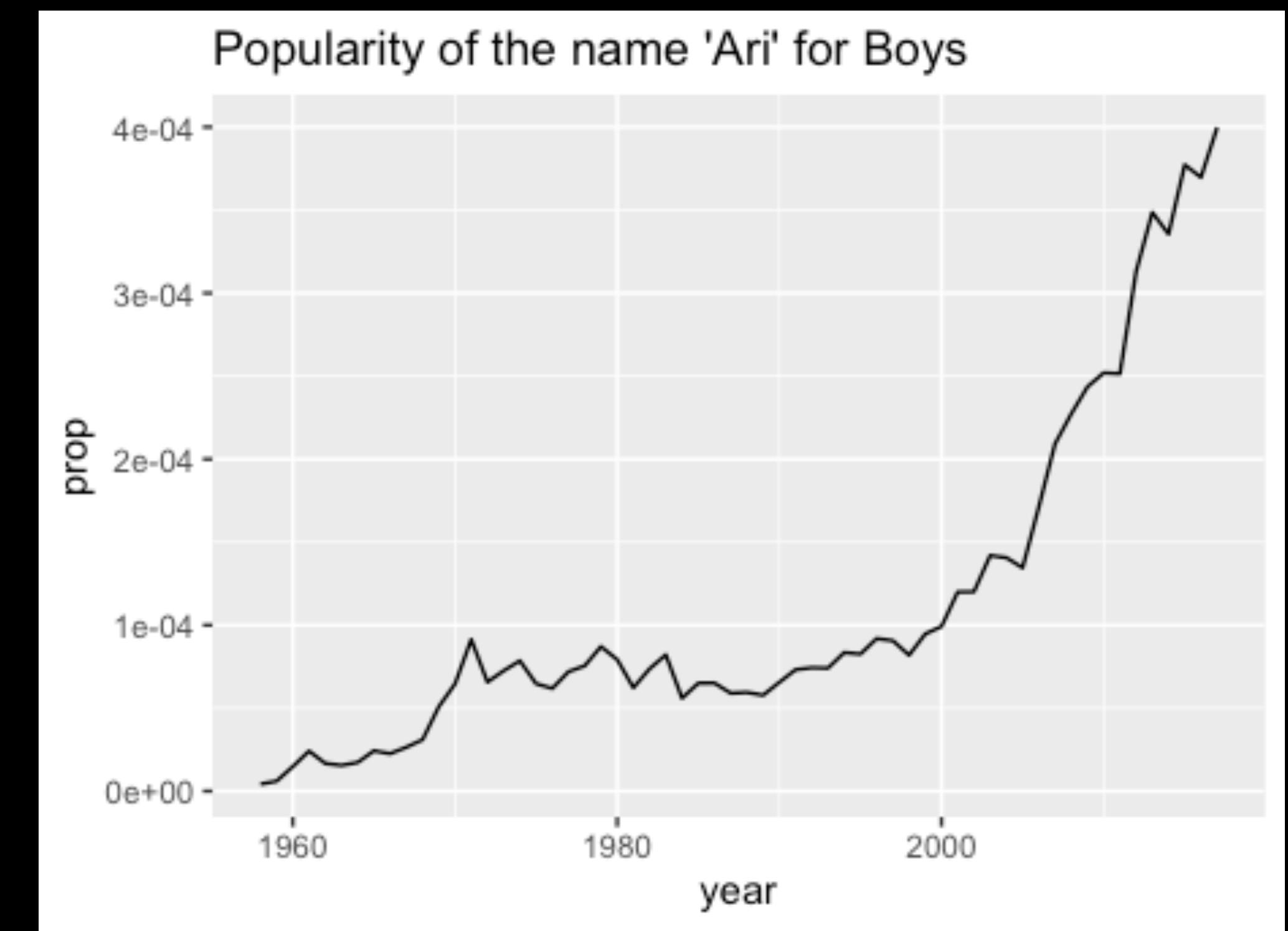
```
babynames %>%  
  filter(name == "Ari", sex == "M") %>%  
  select(year, prop) %>%  
  ggplot() +  
  geom_line(mapping = aes(year, prop))
```



# Adding a Title

- Google "*How to add a title in ggplot2*"

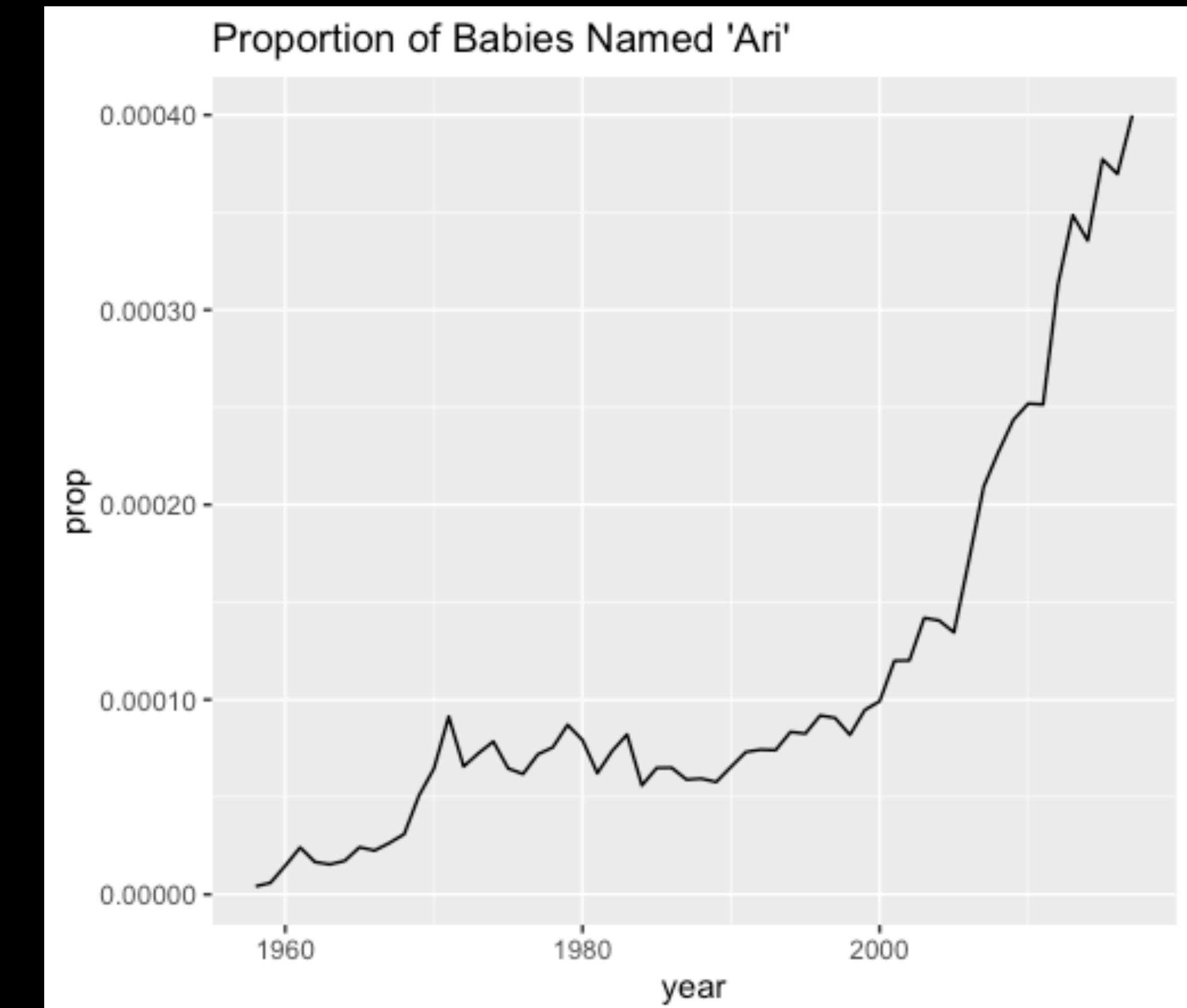
```
p + ggtitle("Popularity of ...")
```



# Removing Scientific Notation

- Google "*ggplot2 remove scientific notation from y axis*"

```
library(scales)  
p +  
  scale_y_continuous(labels=comma)
```



A screenshot of a Google search results page. The search query is "ggplot2 remove scientific notation from y axis". The results show a top result from Stack Overflow about forcing R to stop plotting abbreviated axis labels. Below it are links to other Stack Overflow questions and a statisticsglobe.com article. A large blue arrow points from the search results towards the Stack Overflow answer.

ggplot2 remove scientific notation from y axis

All Images Videos News Shopping More Settings To

About 90,600 results (0.48 seconds)

stackoverflow.com › questions › force-r-to-stop-plottin... ▾

Force R to stop plotting abbreviated axis labels - e.g. 1e+00 in ...

Jan 28, 2013 – As a more general solution, you can use scales::format\_format to remove the scientific notation. This also gives you lots of control around how exactly you want your labels to be displayed, as opposed to scales::comma which only does comma separations of the orders of magnitude.

7 answers

How to force axis values to scientific notation in ggplot ... 1 answer Apr 29, 2017

How do I change the formatting of numbers on an axis ... 5 answers Jun 16, 2014

how to remove scientific notation for plot() - Stack ... 5 answers Mar 9, 2015

avoid scientific notation x axis ggplot - Stack Overflow 1 answer Oct 11, 2018

More results from stackoverflow.com

statisticsglobe.com › change-formatting-of-numbers-of... ▾

Change Formatting of Numbers of ggplot2 Plot Axis in R ...

ggp <- ggplot(data, aes(x, y)) + # Create basic ggplot2 plot geom\_point() ggp. Figure 2: Disable

A screenshot of a Stack Overflow answer. The title is "7 Answers". It features a large blue arrow pointing from the Google search results towards this answer. The answer itself is by user "Dave Jan" and has 140 upvotes. It provides two code snippets: one for scientific notation and one for comma separators.

7 Answers

I think you are looking for this:

140

require(ggplot2)  
df <- data.frame(x=seq(1, 1e9, length.out=10)  
# displays x-axis in scientific notation  
p <- ggplot(data = df, aes(x=x, y=y)) + geo  
p

# displays as you require  
require(scales)  
p + scale\_x\_continuous(labels = comma)

share improve this answer follow edited Jan 30 '19

Dave Jan 27.8k • 3

# The "Google -> Stack Overflow" Loop

*"ggplot2 remove scientific notation from y axis"*

What are the most  
popular names?

# Deriving information

**summarise()** - summarise **variables**

**group\_by()** - group **cases**

**mutate()** - create new **variables**



# summarise()

A faint watermark of the R logo is visible in the bottom right corner, consisting of a circular arrow with the letter 'R' inside.

# summarise()

Compute table of summaries.

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

babynames

year	sex	name	n	prop	
1880	M	John	9655	0.0815	
1880	M	William	9532	0.0805	
1880	M	James	5927	0.0501	
1880	M	Charles	5348	0.0451	
1880	M	Garrett	13	0.0001	
1881	M	John	8769	0.081	

→

total	max
348120517	99686



# 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.

 **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()` - mean absolute deviation  
`sd()` - standard deviation  
`var()` - variance

### Vectorized 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.

 **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

**Misc**

`dplyr::between()` -  $x > \text{right} \& x < \text{left}$   
`dplyr::case_when()` - multi-case if\_else()  
`dplyr::coalesce()` - first non-NA values by element across a set of vectors  
`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.

 **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**

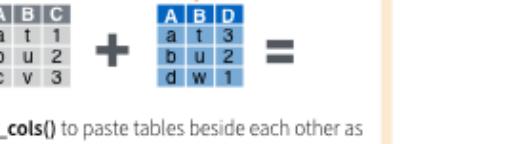
`quantile()` - nth quantile  
`min()` - minimum value  
`max()` - maximum value

**Spread**

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

### Combine Tables

#### Combine Variables

 **x** + **y** =

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

**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

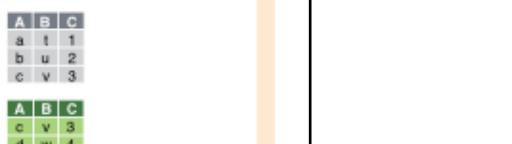
**Rank**

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

**Spread**

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

#### Combine Cases

 **x** + **y** =

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

**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()` - mean absolute deviation  
`sd()` - standard deviation  
`var()` - variance

#### Extract Rows

 **x** + **y** =

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

**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()` - mean absolute deviation  
`sd()` - standard deviation  
`var()` - variance



# Your Turn 7

Use `summarise()` 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



```
babynames %>%  
  summarise(first = min(year),  
            last = max(year),  
            total = sum(n))
```

```
#     first  last      total  
# 1  1880  2015 340851912
```



# Your Turn 8

Extract the rows where **name == "Khaleesi"**. Then use `summarise()` and a summary function to find:

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



```
babynames %>%  
  filter(name == "Khaleesi") %>%  
  summarise(total = sum(n), first = min(year))  
#   total first  
# 1  1127  2011
```



# n()

The number of rows in a dataset/group

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

babynames

year	sex	name	n	prop	
1880	M	John	9655	0.0815	
1880	M	William	9532	0.0805	
1880	M	James	5927	0.0501	
1880	M	Charles	5348	0.0451	
1880	M	Garrett	13	0.0001	
1881	M	John	8769	0.081	

→

n
1924665



# n\_distinct()

The number of distinct values in a variable

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

babynames

year	sex	name	n	prop	n	nname
1880	M	John	9655	0.0815	1924665	97310
1880	M	William	9532	0.0805		
1880	M	James	5927	0.0501		
1880	M	Charles	5348	0.0451		
1880	M	Garrett	13	0.0001		
1881	M	John	8769	0.081		



# Grouping cases



# group\_by()

Groups cases by common values of one or more columns.

```
babynames %>%  
  group_by(sex)
```

Source: local data frame [1,825,433 x 5]

Groups: sex [2]

	year	sex	name	n	prop
	<dbl>	<chr>	<chr>	<int>	<dbl>
1	1880	F	Mary	7065	0.07238359



# group\_by()

Groups cases by common values.

```
babynames %>%  
  group_by(sex) %>%  
  summarise(total = sum(n))
```

sex	total
F	167070477
M	170064949



# Transform Data Notebook

```
1 ---  
2 title: "Transform Data"  
3 output: html_notebook  
4 ---  
5  
6 ```{r setup}  
7 library(tidyverse)  
8 library(babynames)  
9  
10 # Toy datasets to use  
11  
12 pollution <- tribble(  
13   ~city, ~size, ~amount,  
14   "New York", "large", 23,  
15   "New York", "small", 14,  
16   "London", "large", 22,  
17   "London", "small", 16,  
18   "Beijing", "large", 121,  
19   "Beijing", "small", 56  
20 )  
21  
22 band <- tribble(  
23   ~name, ~band,  
24   "Mick", "Stones",  
25   "John", "Beatles",  
26   "Paul", "Beatles"  
27 )  
28  
29 instrument <- tribble(  
30   ~name, ~plays,  
31   "John", "guitar",  
32   "Paul", "bass",  
33   "Keith", "guitar"  
34 )  
35  
36  
37 ## Your Turn 1  
38
```

34:2 | Chunk 1: setup | R Markdown

```
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
```

Toy data sets to practice with



## pollution

```
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	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
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

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

city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
New York	large	23
New York	small	14



mean	sum	n
18.5	37	2

London	large	22
London	small	16



19.0	38	2
------	----	---

Beijing	large	121
Beijing	small	56



88.5	177	2
------	-----	---

group\_by() + summarise()



# group\_by()

city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
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

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

# group\_by()

city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount ( $\mu\text{g}/\text{m}^3$ )
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	mean	sum	n
New York	large	23	23	1
New York	small	14	14	1
London	large	22	22	1
London	small	16	16	1
Beijing	large	121	121	1
Beijing	small	56	56	1

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

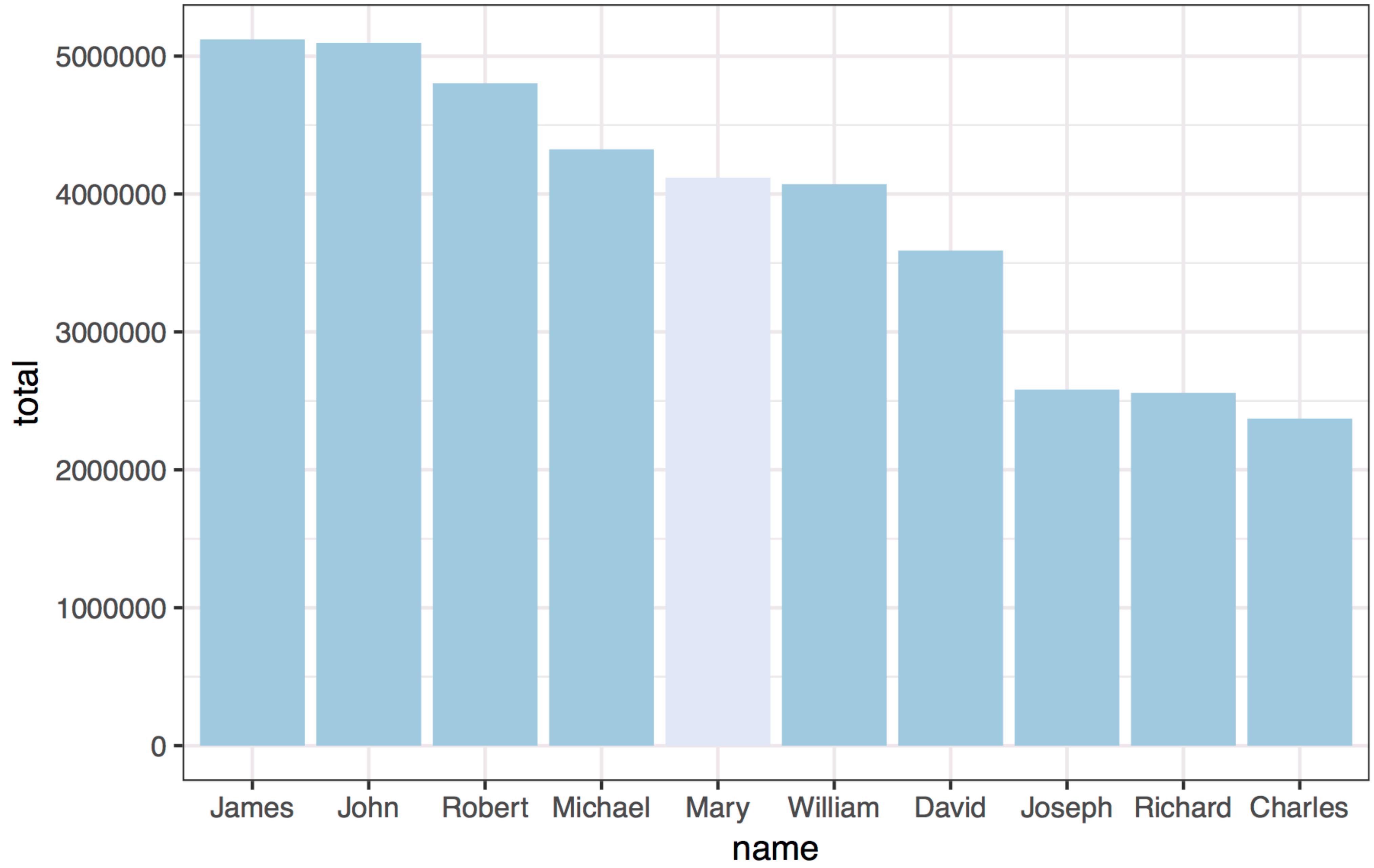
# Your Turn 9

Use **group\_by()**, **summarise()**, and **arrange()** to display the ten most popular names. Compute popularity as the total number of children of a single gender given a name.



```
babynames %>%  
  group_by(name, sex) %>%  
  summarise(total = sum(n)) %>%  
  arrange(desc(total))
```

```
#          name sex  total  
# 1      James   M 5120990  
# 2      John   M 5095674  
# 3    Robert   M 4803068  
# 4  Michael   M 4323928  
# 5      Mary   F 4118058  
# 6  William   M 4071645  
# 7      David   M 3589754  
# 8    Joseph   M 2581785  
# 9  Richard   M 2558165  
# 10  Charles   M 2371621  
# ... with 105,376 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 10

Use grouping to calculate number of children born each year.

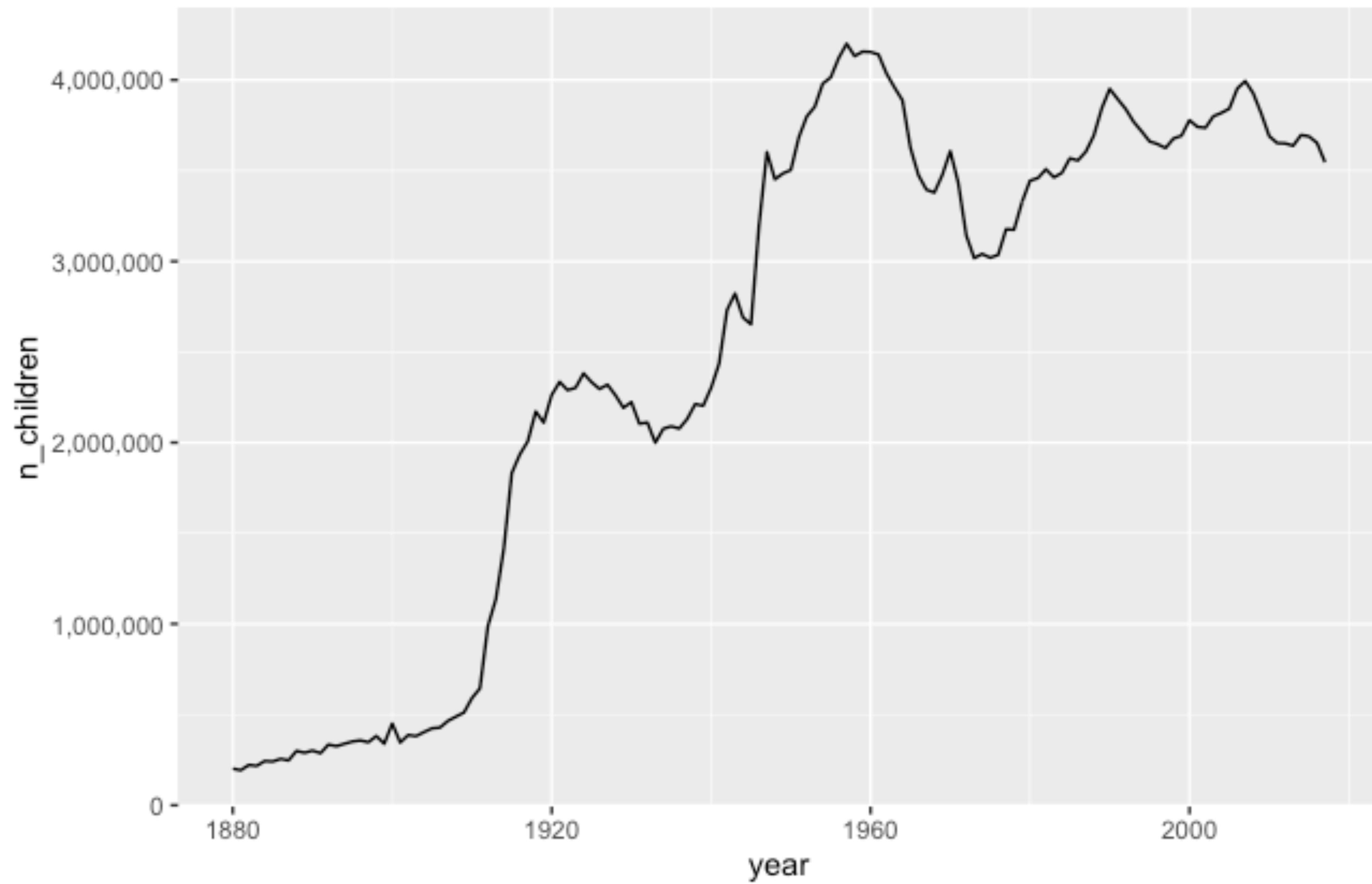
Plot the results as a line graph.



```
library(scales)

babynames %>%
  group_by(year) %>%
  summarise(n_children = sum(n)) %>%
  ggplot() +
  geom_line(mapping = aes(x = year, y = n_children)) +
  scale_y_continuous(label=comma) +
  ggtitle("Number of Children Born Each Year")
```

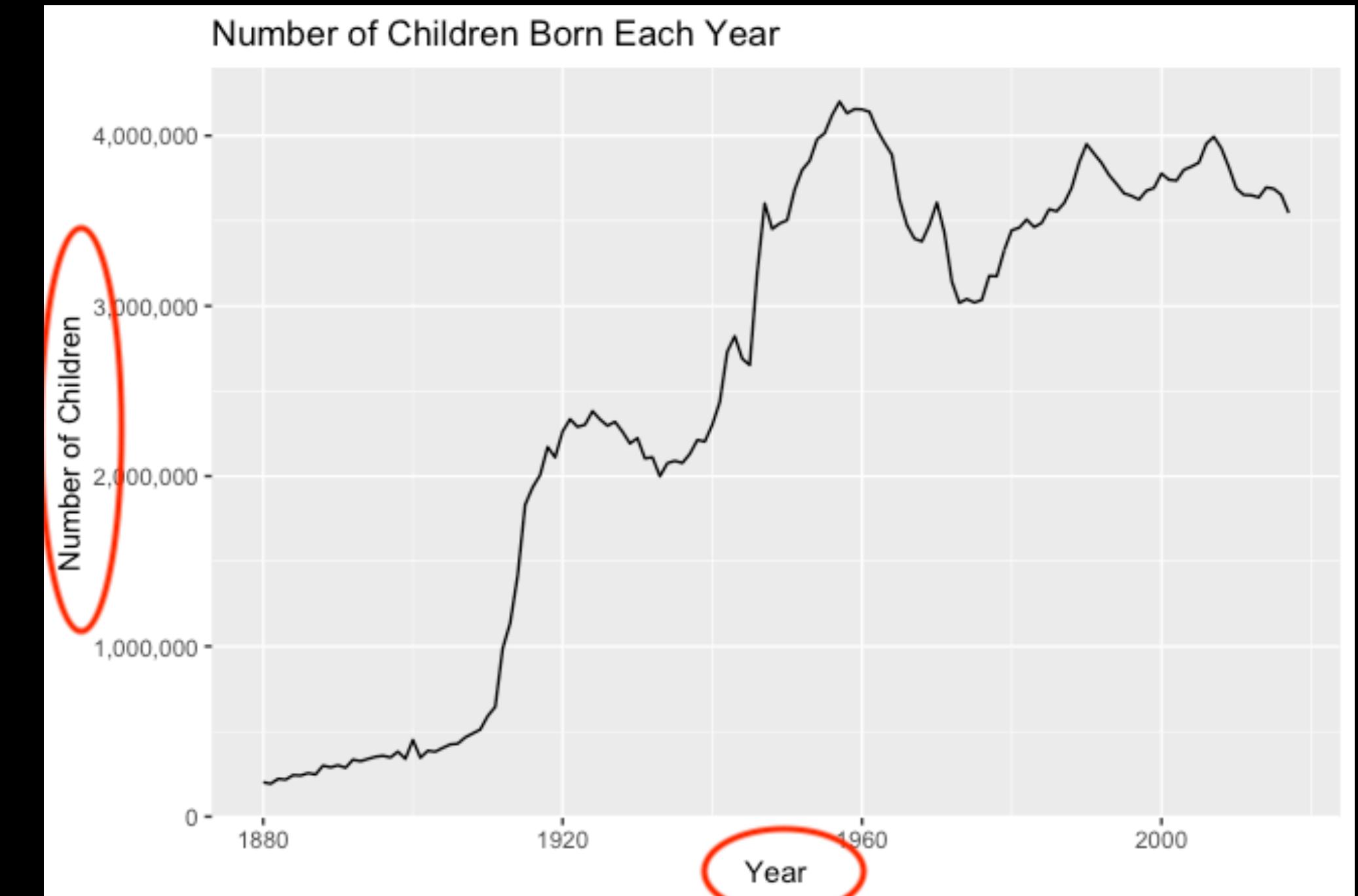
## Number of Children Born Each Year



# Custom Scale Labels

- Google "*ggplot2 change axis labels*"

```
p +  
  labs(x = "Year", y = "Number of Children")
```



# ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%  
  group_by(name, sex) %>%  
  summarise(total = sum(n)) %>%  
  arrange(desc(total))
```

#		name	sex	total
#	1	James	M	5120990
#	2	John	M	5095674
#	3	Robert	M	4803068
#	4	Michael	M	4323928
#	5	Mary	F	4118058



# ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%  
  group_by(name, sex) %>%  
  ungroup() %>%  
  summarise(total = sum(n)) %>%  
  arrange(desc(total))  
  
#       total  
# 1 340851912
```



# mutate()



# mutate()

Create new columns.

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

year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop	percent
1880	M	John	9655	0.0815	8.15
1880	M	William	9532	0.0805	8.05
1880	M	James	5927	0.0501	5.01
1880	M	Charles	5348	0.0451	4.51
1880	M	Garrett	13	0.0001	0.01
1881	M	John	8769	0.081	8.1



# mutate()

Create new columns.

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

babynames

year	sex	name	n	prop	percent	nper
1880	M	John	9655	0.0815	8.15	8
1880	M	William	9532	0.0805	8.05	8
1880	M	James	5927	0.0501	5.01	5
1880	M	Charles	5348	0.0451	4.51	5
1880	M	Garrett	13	0.0001	0.01	0
1881	M	John	8769	0.081	8.1	8

→

## Vectorized 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.



#### 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 <= 1  
`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

#### Misc

`dplyr::between()` -  $x >$  right &  $x <$  left  
`dplyr::case_when()` - multi-case if\_else()  
`dplyr::coalesce()` - first non-NA values by element across a set of vectors  
`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

# Vectorized functions

Take a vector as input.  
Return a vector of the same length as output.

The grid contains the following sections:

- Vectorized Functions**: to use with `mutate()`. Describes `mutate()` and `transmute()` applying vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.
- Summary Functions**: to use with `summarise()`. Describes `summarise()` applying summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.
- Combine Tables**:
  - Combine Variables**: Shows tables `x` and `y` being combined with `bind_cols()`.
  - Combine Cases**: Shows tables `x`, `y`, and `z` being combined with `bind_rows()`.
- Extract Rows**: Shows tables `x` and `y` being filtered with `semi_join()` and `anti_join()`.

Each card includes examples and descriptions of the functions used.

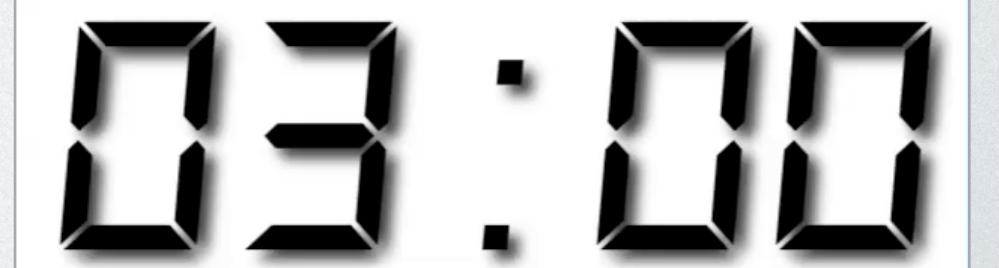


# Your Turn 11

Dataframe **weights** lists the name and weight (in pounds) of professional martial artists. Add a column **kgs** that lists their weight to kilograms.

Hint: multiply the **lbs** column by .45 to get the number of kilograms

```
> weights
# A tibble: 5 × 2
  name     lbs
  <chr>   <dbl>
1 Stipe    255
2 Jan      205
3 T        185
4 S        220
5 C        210
```



```
weights %>%  
  mutate(kgs = lbs *.45)
```

```
> weights %>%  
+   mutate(kgs = lbs *.45)  
# A tibble: 5 x 3  
  name    lbs    kgs  
  <chr>  <dbl>  <dbl>  
1 Stipe    255  115.  
2 Jan      205  92.2  
3 Israel   185  83.2  
4 Karamu   170  76.5  
5 Khabib   150  67.5
```

# Recap: Single table verbs



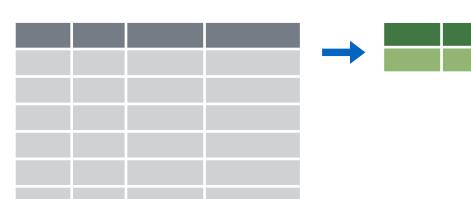
Extract variables / columns with **select()**



Extract cases / rows with **filter()**



Arrange rows, with **arrange()**.



Make tables of summaries with **summarise()**.



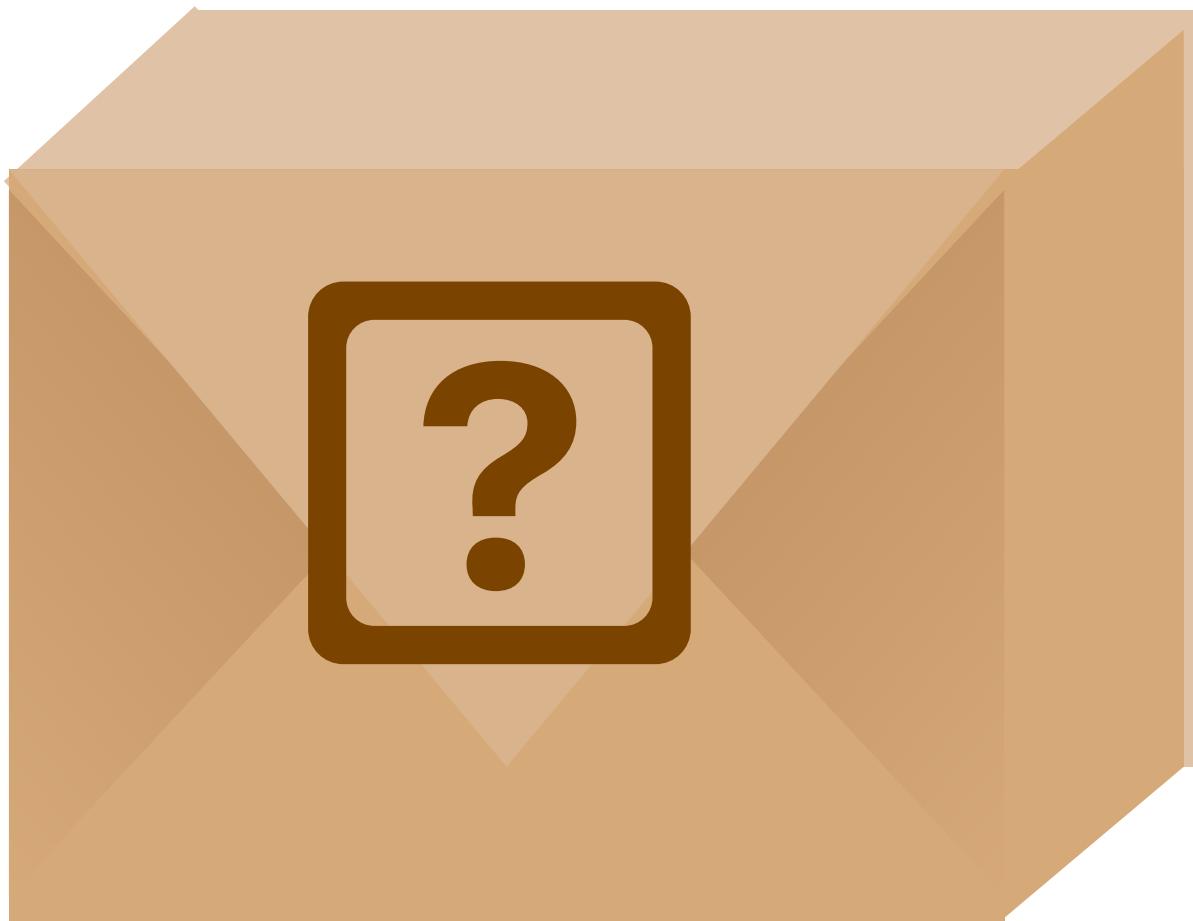
Make new variables, with **mutate()**.



# Joining Datasets

R

# nycflights13

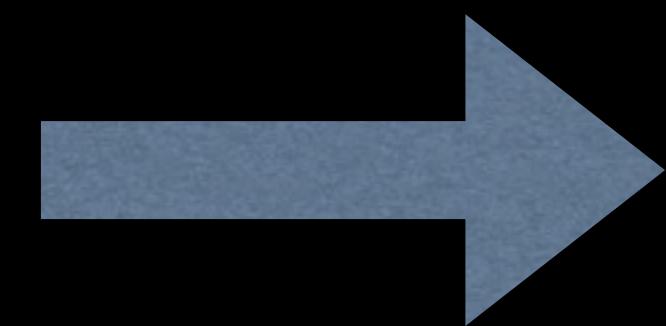


Data about every flight that departed La  
Guardia, JFK, or Newark airports in 2013

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



<input type="checkbox"/>	<a href="#">modelr</a>	Modelling Functions that Work with the Pipe	0.1.8	
<input type="checkbox"/>	<a href="#">munsell</a>	Utilities for Using Munsell Colours	0.5.0	
<input type="checkbox"/>	<a href="#">nlme</a>	Linear and Nonlinear Mixed Effects Models	3.1-150	
<input type="checkbox"/>	<a href="#">nnet</a>	Feed-Forward Neural Networks and Multinomial Log-Linear Models	7.3-14	
<input type="checkbox"/>	<a href="#">numDeriv</a>	Accurate Numerical Derivatives	2016.8-1.1	
<input checked="" type="checkbox"/>	<a href="#">nycflights13</a>	Flights that Departed NYC in 2013	1.0.1	
<input type="checkbox"/>	<a href="#">openssl</a>	Toolkit for Encryption, Signatures and Certificates Based on OpenSSL	1.4.3	
<input type="checkbox"/>	<a href="#">packrat</a>	A Dependency Management System for Projects and their R Package Dependencies	0.5.0	
<input type="checkbox"/>	<a href="#">parallel</a>	Support for Parallel computation in R	4.0.3	
<input type="checkbox"/>	<a href="#">readr</a>	Read data from text files	1.3.0	



## Documentation for package 'nycflights13' version 1.0.1

- [DESCRIPTION file.](#)

### Help Pages

<a href="#">airlines</a>	Airline names.
<a href="#">airports</a>	Airport metadata
<a href="#">flights</a>	Flights data
<a href="#">planes</a>	Plane metadata.
<a href="#">weather</a>	Hourly weather data

# Package Description

5 tables

# Flights

[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
2013	1	1	517	515	2	830	819	11	UA	1545	N14228	EWR	IAH	227	1400	5	15	2013-01-01 05:00:00
2013	1	1	533	529	4	850	830	20	UA	1714	N24211	LGA	IAH	227	1416	5	29	2013-01-01 05:00:00
2013	1	1	542	540	2	923	850	33	AA	1141	N619AA	JFK	MIA	160	1089	5	40	2013-01-01 05:00:00
2013	1	1	544	545	-1	1004	1022	-18	B6	725	N804JB	JFK	BQN	183	1576	5	45	2013-01-01 05:00:00
2013	1	1	554	600	-6	812	837	-25	DL	461	N668DN	LGA	ATL	116	762	6	0	2013-01-01 06:00:00
2013	1	1	554	558	-4	740	728	12	UA	1696	N39463	EWR	ORD	150	719	5	58	2013-01-01 05:00:00
2013	1	1	555	600	-5	913	854	19	B6	507	N516JB	EWR	FLL	158	1065	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	709	723	-14	EV	5708	N829AS	LGA	IAD	53	229	6	0	2013-01-01 06:00:00
2013	1	1	557	600	-3	838	846	-8	B6	79	N593JB	JFK	MCO	140	944	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	753	745	8	AA	301	N3ALAA	LGA	ORD	138	733	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	849	851	-2	B6	49	N793JB	JFK	PBI	149	1028	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	853	856	-3	B6	71	N657JB	JFK	TPA	158	1005	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	924	917	7	UA	194	N29129	JFK	LAX	345	2475	6	0	2013-01-01 06:00:00
2013	1	1	558	600	-2	923	937	-14	UA	1124	N53441	EWR	SFO	361	2565	6	0	2013-01-01 06:00:00
2013	1	1	559	600	-1	941	910	31	AA	707	N3DUAA	LGA	DFW	257	1389	6	0	2013-01-01 06:00:00
2013	1	1	559	559	0	702	706	-4	B6	1806	N708JB	JFK	BOS	44	187	5	59	2013-01-01 05:00:00
2013	1	1	559	600	-1	854	902	-8	UA	1187	N76515	EWR	LAS	337	2227	6	0	2013-01-01 06:00:00
2013	1	1	600	600	0	851	858	-7	B6	371	N595JB	LGA	FLL	152	1076	6	0	2013-01-01 06:00:00

## Details



⚠ The class of service you searched may not be available on one or more flights

BNA - ORD

Flight 1 of 2

ORD - YVR

Flight 2 of 2

Nashville, TN to Chicago, IL

Thursday, July 26, 2018

4:10 PM → 6:03 PM

AA 3246 ■ CRJ-900 RJ 700  
Operated by SkyWest Airlines As American Eagle

### Travel info

Travel time: 1h 53m  
Connection time: 2h 33m

### Performance

On time: 52%  
Late: 43%

## Performance\*

**On time: 52%\*\***  
**Late: 43%**

### Main Cabin

Meals: Beverage service  
Booking code: V  
Class: Economy

### Business

Meals: Beverage service  
Booking code: I  
Class: First

\* This is based on information from the month of May 2018

\*\* The on-time arrival percentage for the selected flight is based on arrival within 14 minutes after

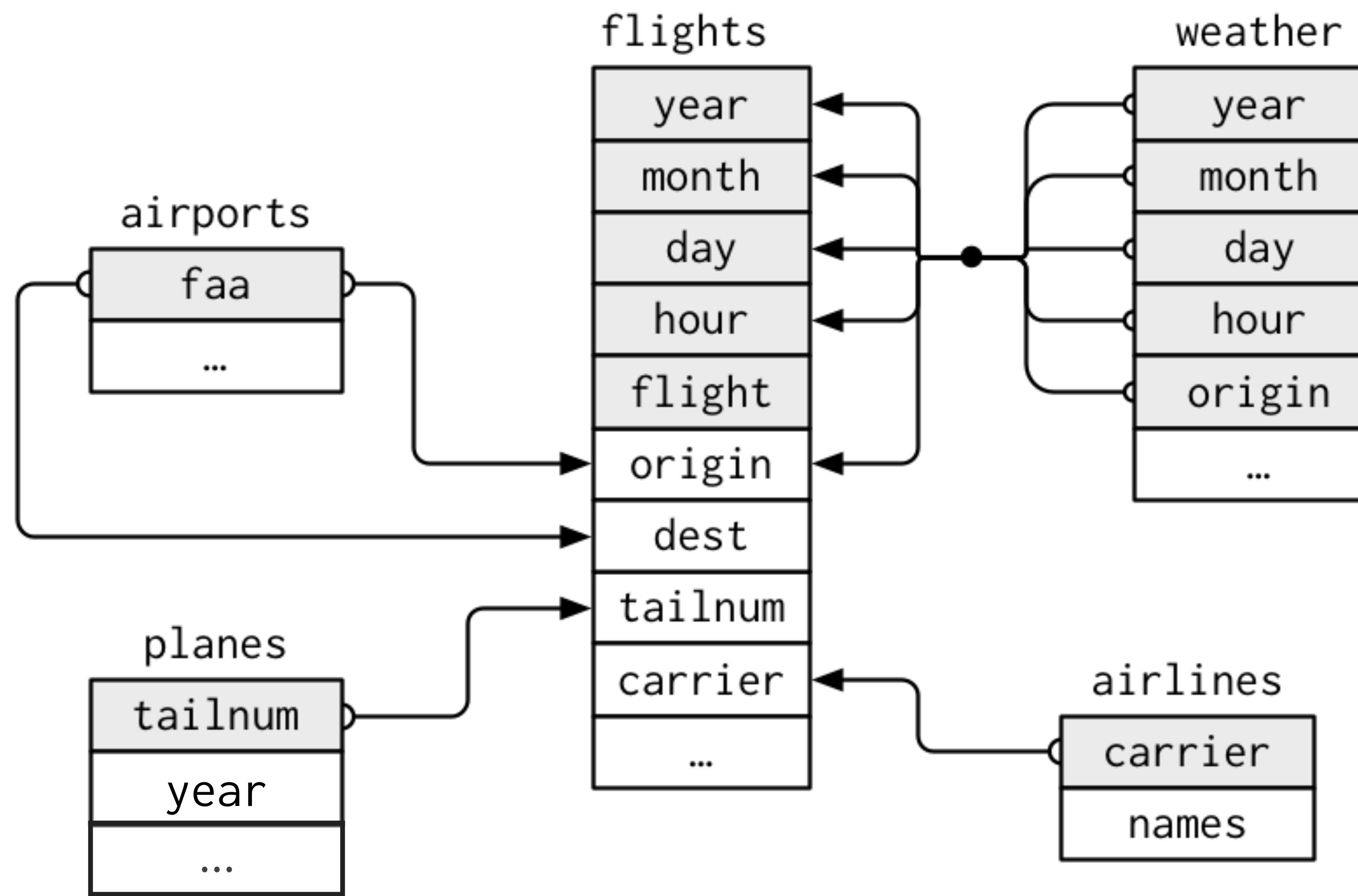
**\*\* The on-time arrival percentage for the selected flight is based on arrival within 14 minutes after the scheduled arrival as reported monthly to the U.S. Department of Transportation.**

# Carriers (airlines)

[View\(airlines\)](#)

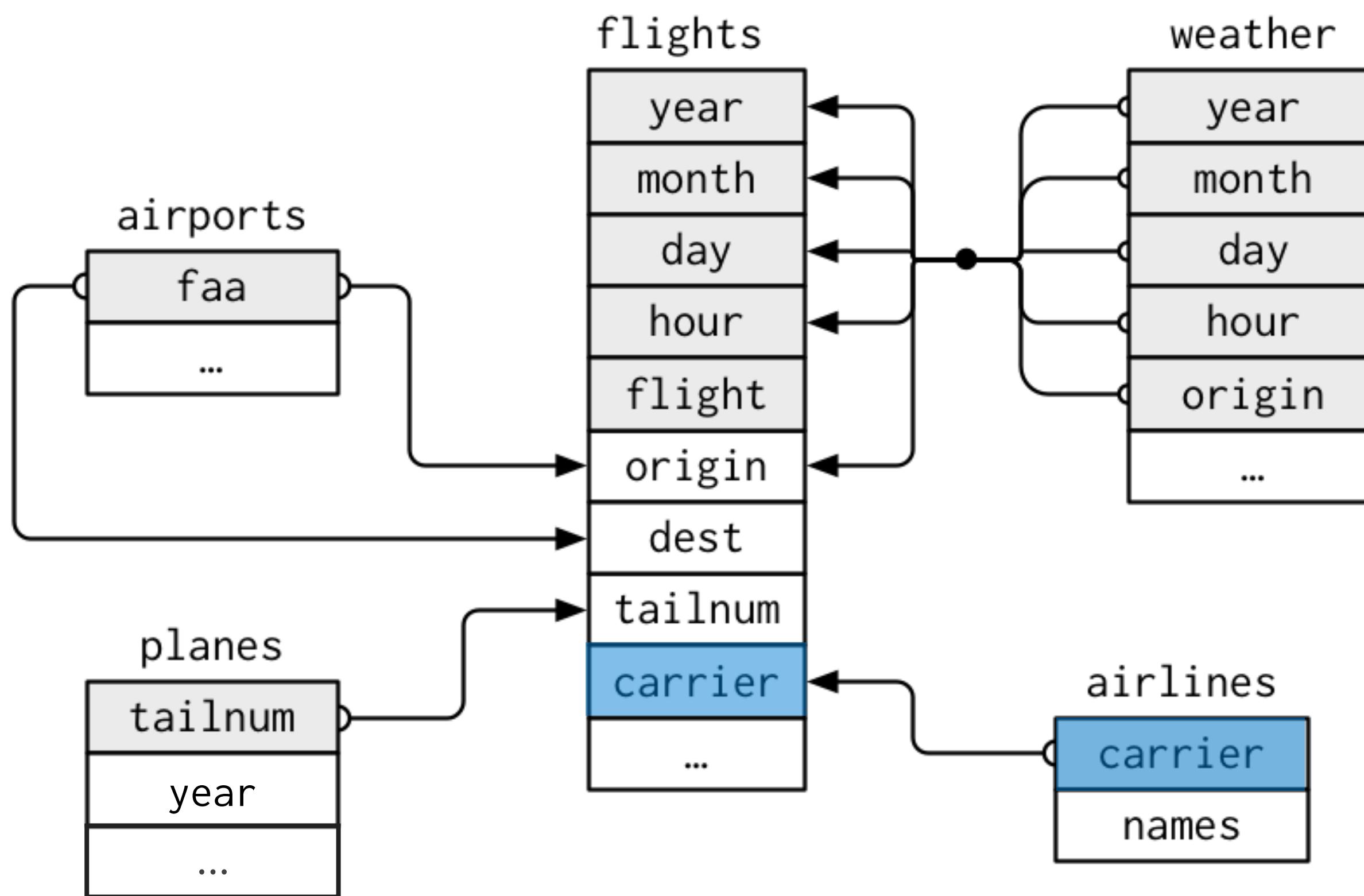
	<b>carrier</b>	<b>name</b>
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways

# nycflights13



# nycflights13

What airline had the longest delays?



# Types of joins

R

# common syntax

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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

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

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

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



# Toy data

band		instrument	
name	band	name	plays
Mick	Stones	John	guitar
John	Beatles	Paul	bass
Paul	Beatles	Keith	guitar



# inner

```
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



# left

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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass

=

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



# right

```
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

```
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



# Your Turn 12

Which airline had the most flights?

flights %>%

group\_by(\_\_\_\_\_) %>%

summarize(\_\_\_\_\_) %>%

arrange(\_\_\_\_\_) %>%

left\_join(\_\_\_\_)

1. Start with a normal "group by / summarize / arrange"

2. Then do a "left\_join" to get the carrier"



```
flights %>%  
  group_by(carrier) %>%  
  summarize(n=n()) %>%  
  arrange(desc(n)) %>%  
  left_join(airlines)
```

```
carrier      n name  
<chr>    <int> <chr>  
1 UA        58665 United Air Lines Inc.  
2 B6        54635 JetBlue Airways  
3 EV        54173 ExpressJet Airlines Inc.  
4 DL        48110 Delta Air Lines Inc.  
5 AA        32729 American Airlines Inc.
```



# Your Turn 13

Which airlines had the largest arrival delays?

```
flights %>%
```

```
drop_na(arr_delay) %>%
```

```
...
```

1. Drop rows with NA delays

2. Calculate average delay  
with `group_by / summarize /  
arrange`

3. Use `left_join` to get airline  
name

06 : 00

```
flights %>%  
  drop_na(arr_delay) %>%  
  group_by(carrier) %>%  
  summarize(avg_arr_delay = mean(arr_delay)) %>%  
  arrange(desc(avg_arr_delay)) %>%  
  left_join(airlines)
```

	carrier	avg_arr_delay	name
	<chr>	<dbl>	<chr>
1	F9	21.9	Frontier Airlines Inc.
2	FL	20.1	AirTran Airways Corporation
3	EV	15.8	ExpressJet Airlines Inc.
4	YV	15.6	Mesa Airlines Inc.
5	00	11.9	SkyWest Airlines Inc.



# Toy data

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

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

instrument2

artist	plays
John	guitar
Paul	bass
Keith	guitar

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



# What if the names do not match?

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

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

A named vector

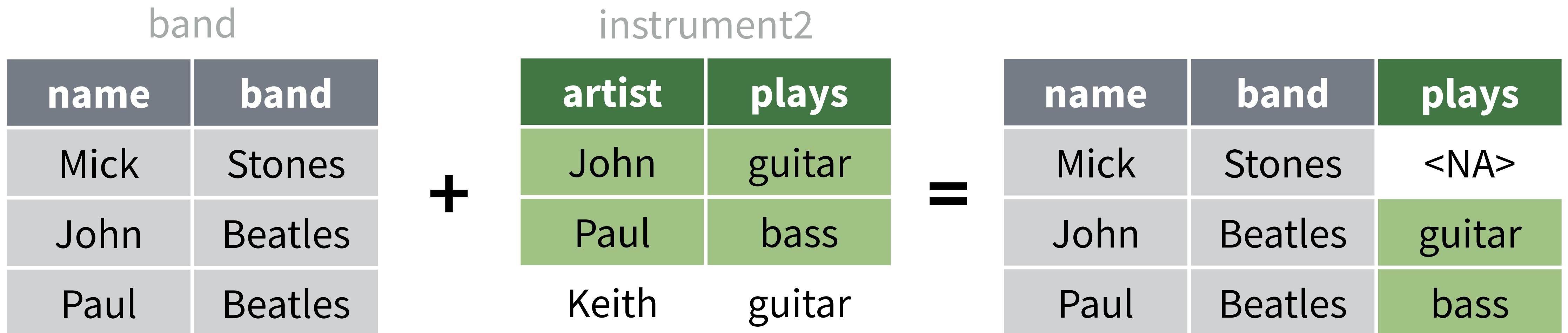
The name of the  
element = the column  
name in the first data  
set

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



# common syntax - matching names

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



# Airport names

`View(flights["dest"])`

	dest
1	IAH
2	IAH
3	MIA
4	BQN
5	ATL
6	ORD
7	FIA

`View(airports)`

faa	name
04G	Lansdowne Airport
06A	Moton Field Municipal Airport
06C	Schaumburg Regional
06N	Randall Airport
09J	Jekyll Island Airport
0A9	Elizabethton Municipal Airport
0CC	Willow Creek Airport

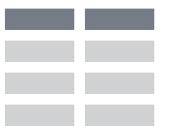
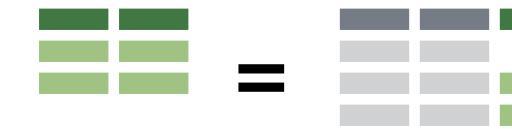
# common syntax - matching names

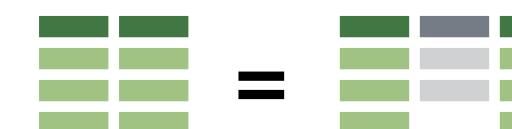
```
airports %>% left_join(flights, by = c("faa" = "dest"))
```

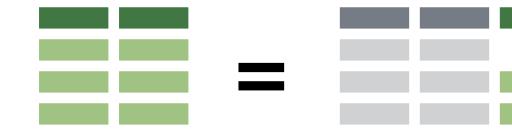
	faa <chr>	name <chr>		dest <chr>	air_time <dbl>
04G	Lansdowne Airport			IAH	227
06A	Moton Field Municipal Airport			IAH	227
06C	Schaumburg Regional			MIA	160
06N	Randall Airport			BQN	183
09J	Jekyll Island Airport			ATL	116
0A9	Elizabethton Municipal Airport			ORD	150
0G6	Williams County Airport			FLL	158
0G7	Finger Lakes Regional Airport			IAD	53

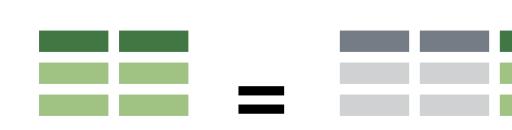


# Recap: Two table verbs

 +  =  **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 only cases in **both** data sets



# Two table verbs

**Vectorized 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.

 **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

**Misc**

`dplyr::between()` - x > right & x < left  
`dplyr::case_when()` - multi-case if\_else()  
`dplyr::coalesce()` - first non-NA values by element across a set of vectors  
`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

**Row names**

`dplyr::between()` - x > right & x < left  
`dplyr::case_when()` - multi-case if\_else()  
`dplyr::coalesce()` - first non-NA values by element across a set of vectors  
`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

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

**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.

 **Counts**  
`dplyr::n()` - number of values/rows  
`dplyr::n_distinct()` - # of uniqueness  
`sum(is.na())` - # of non-NAs

**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()` - mean absolute deviation  
`sd()` - standard deviation  
`var()` - variance

**Extract Rows**

`dplyr::between()` - Use `by = c("col1", "col2")` to specify the column(s) to match on.  
`dplyr::left_join()` - Use `by = "A"`  
`dplyr::semi_join()` - Use a named vector, `by = c("col1" = "col2")`, to match on columns with different names in each data set.  
`dplyr::anti_join()` - Use `suffix` to specify suffix to give to duplicate column names.  
`dplyr::left_join()` - Use `by = c("C" = "D")`, `suffix = c("1", "2")`

Learn more with `browseVignettes(package = c("dplyr", "tibble"))` • dplyr 0.5.0 • tibble 1.2.0 • Updated: 12/16

**Combine Tables**

**Combine Variables**

x	A	B	C
	a	t	1
	b	u	2
	c	v	3

**+**

y	A	B	D
	a	t	3
	b	u	2
	d	w	1

**=**

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

A	B	C	A	B	D
a	t	1	a	t	3
b	u	2	b	u	2
c	v	3	d	w	1

**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.

A	B	C	D
a	t	1	3
b	u	2	2
c	v	3	NA

**left\_join(x, y, by = NULL, copy = FALSE, suffix = c("x", "y"), ...)**

Join matching values from y to x.

A	B	C	D
x	a	t	1
x	b	u	2
x	c	v	3
x	d	w	4

**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)

DF	A	B	C
x	a	t	1
x	b	u	2
x	c	v	3
z	c	v	3
z	d	w	4

**intersect(x, y, ...)**

Rows that appear in both x and z.



**setdiff(x, y, ...)**

Rows that appear in both x but not z.



**union(x, y, ...)**

Rows that appear in x or z. (Duplicates removed). `union_all()` retains duplicates.



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

**Combine Cases**

x	A	B	C
	a	t	1
	b	u	2
	c	v	3

**+**

y	A	B	C
	c	v	3
	d	w	4

**=**

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

A	B	C
a	t	1
b	u	2
c	v	3
d	w	4

**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)

**left\_join(x, y, by = "A")**

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

A	B.x	C	B.y	D
a	t	1	1	3
b	u	2	2	2
c	v	3	NA	NA

**semi\_join(x, y, by = NULL, ...)**

Use a named vector, `by = c("col1" = "col2")`, to match on columns with different names in each data set.

A	B	C	A.y	B.y
a	t	1	d	w
b	u	2	b	u
c	v	3	a	t

**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.

A	B	C	A2	B2
a	t	1	d	w
b	u	2	b	u
c	v	3	a	t

**left\_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))**

Use `suffix` to specify suffix to give to duplicate column names.

A	B	C
a	t	1
b	u	2
c	v	3

**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.



# Tidy tools

R

## Vignettes and other documentation

---



### Vignettes from package 'tidyverse'

[tidyverse::manifesto](#) The tidy tools manifesto  
[tidyverse::paper](#) Welcome to the tidyverse

There are four basic principles to a tidy API:

1. Reuse existing data structures.
2. Compose simple functions with the pipe.
3. Embrace functional programming.
4. Design for humans.

# The Tidy Tools Manifesto

## Really

# **Designed for Humans:**

*Functions do one thing, and they do it well*

**filter()** - extract **cases**

**arrange()** - reorder **cases**

**group\_by()** - group **cases**

**select()** - extract **variables**

**mutate()** - create new **variables**

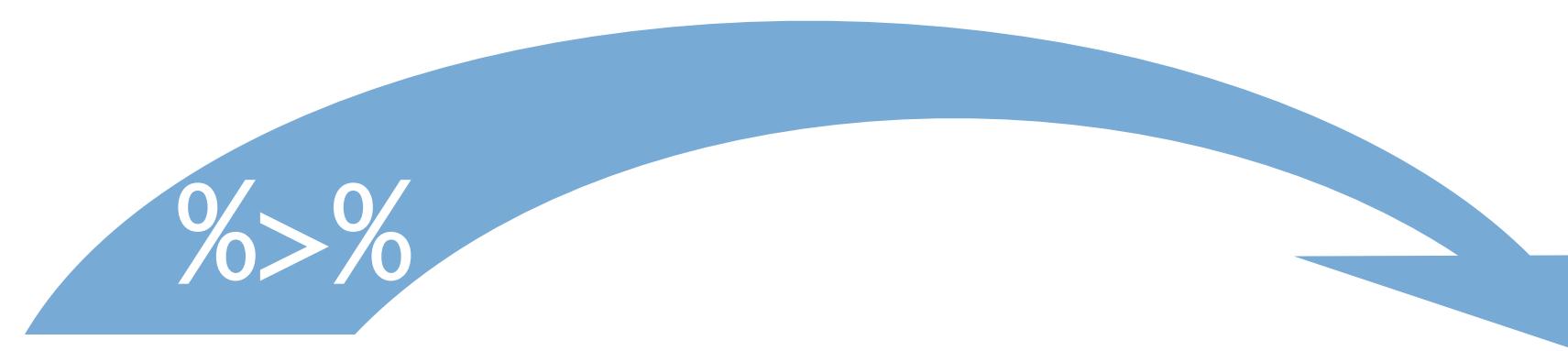
**summarise()** - summarise **variables** / create **cases**

The screenshot shows a web browser window with the title "Welcome | The tidyverse style" and the URL "style.tidyverse.org". The main content is titled "Welcome" and discusses the importance of good coding style. It mentions that the style guide is derived from Google's original R Style Guide. Below this, there is a section about style guides being fundamentally opinionated and providing consistency. A sidebar on the left contains a "Table of contents" with numbered sections from 1 to 11, including "Welcome", "Analyses", "Files", "Syntax", "Functions", "Pipes", "ggplot2", "Packages", "Files", "Documentation", "Tests", "Error messages", "News", and "Git/GitHub". At the bottom of the sidebar is a link "View book source". On the right side, there is a "On this page" sidebar with links to "Welcome", "View source", and "Edit this page". A small screenshot of a software interface showing the "STYLER" add-in is also visible.

# Following Conventions

## Tidyverse Style Guide

# Compose simple functions with the pipe



`babynames`      `mutate(_____, percent = prop * 100)`

Each dplyr function takes a data frame as its first argument and returns a data frame. As a result, you can directly pipe the output of one function into the next.

# Closing Exercise

- Fill out the "Summary" section
  - Fill out the "How I can apply to my work"
  - Share with your neighbor!

03.00

# dplyr

Main Ideas	Notes _____ _____ _____ _____ _____ _____ _____ _____ _____
Summary	
How I can apply this to my work	

# Any Questions?