

Transform Data with



babynames



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

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



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

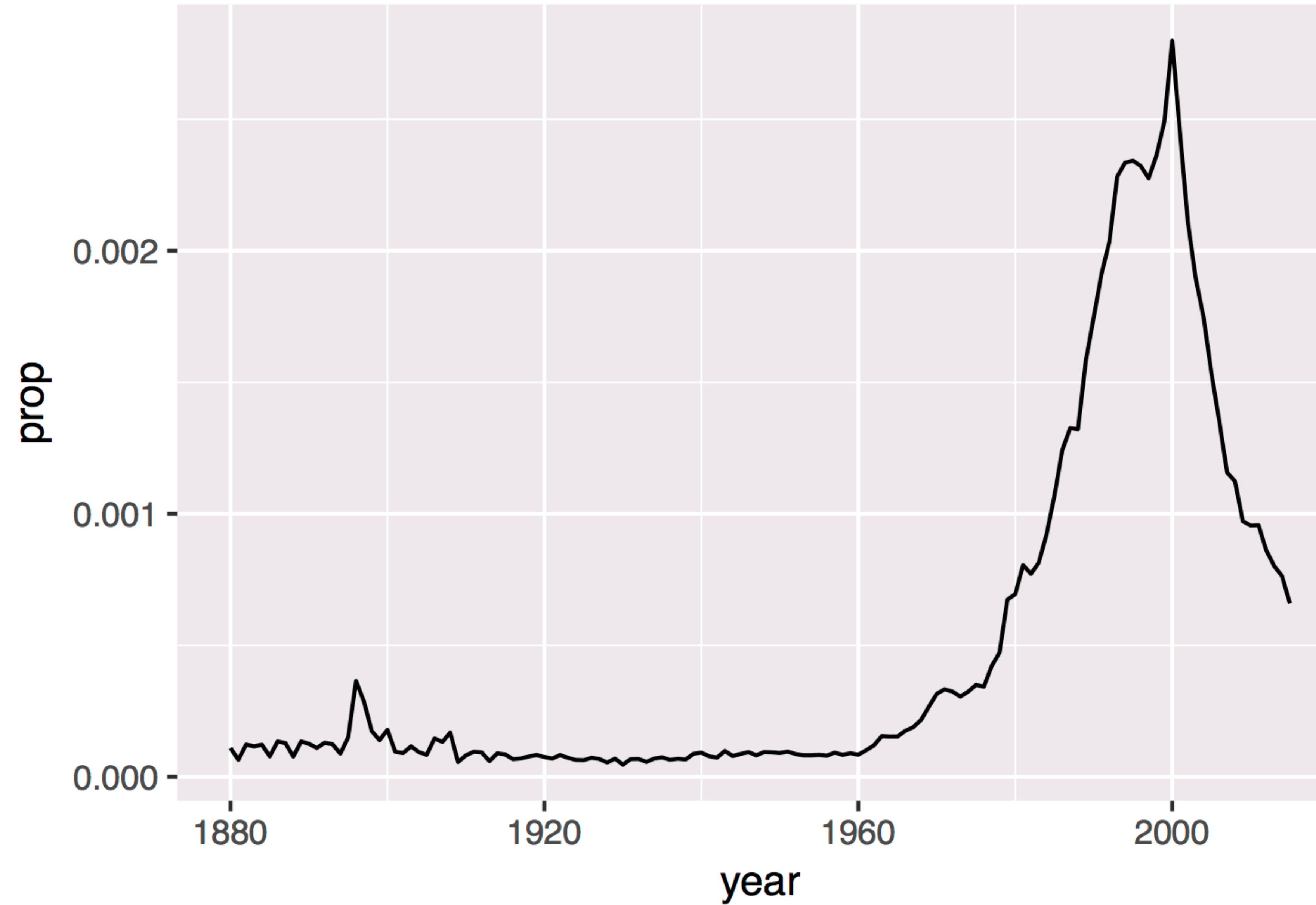
...

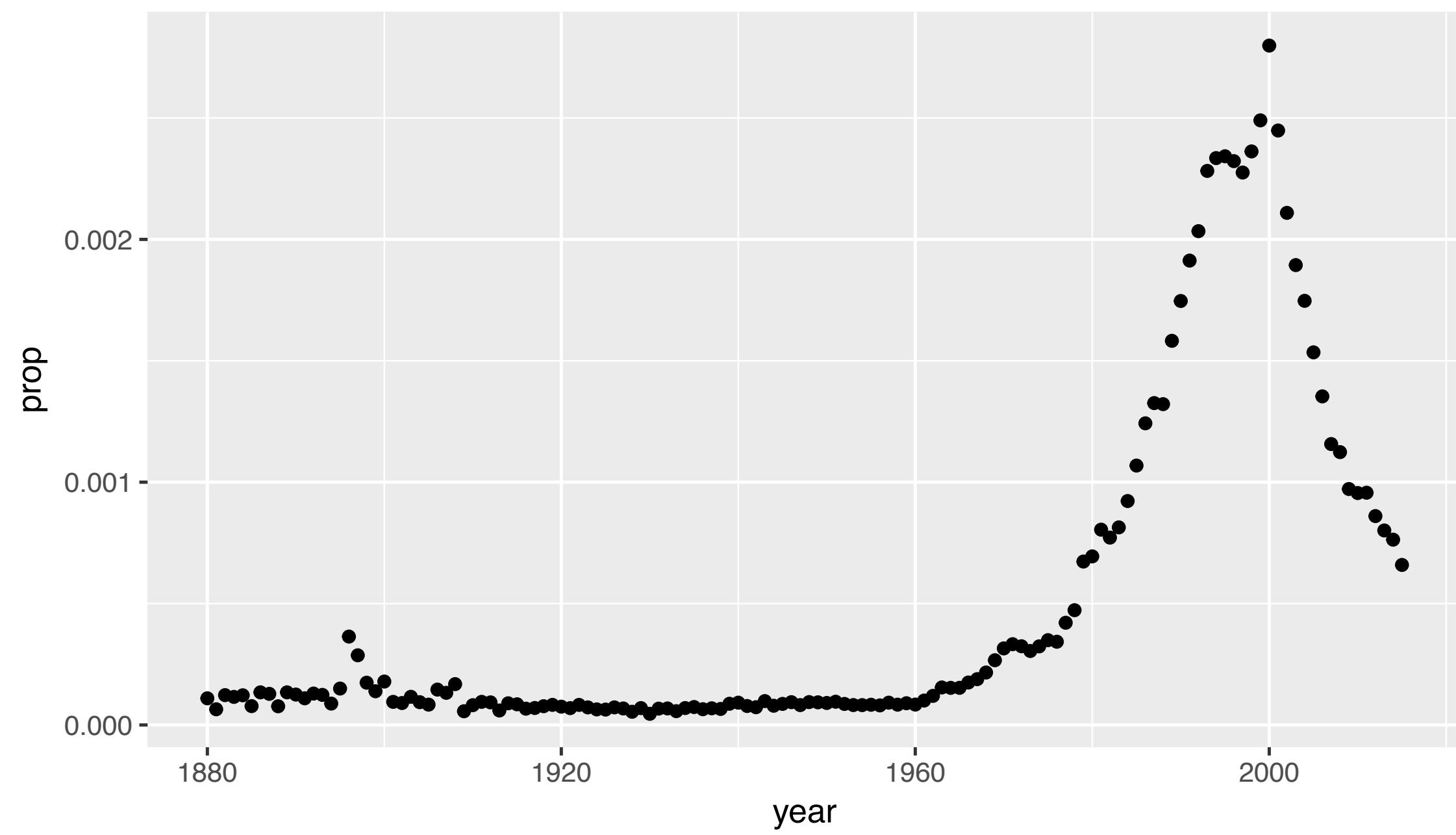
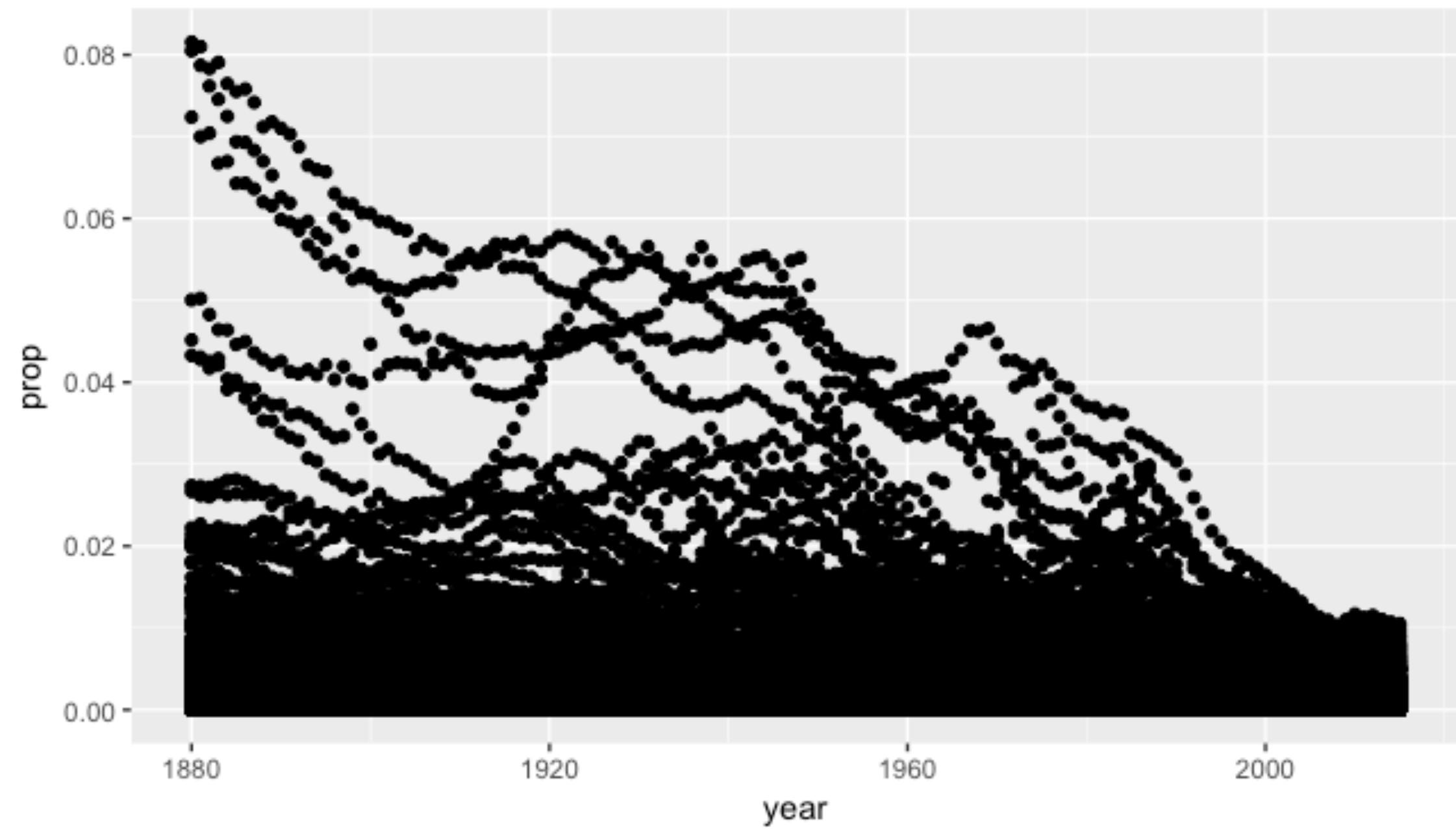
100

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**

filter() - extract **cases**

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

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  
# ... ...
```



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

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

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

matches() - Select columns whose names match regular expression

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

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

```
select(storms, one_of(c("storm", "storms", "Storm")))
```

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

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



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()`

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"))`

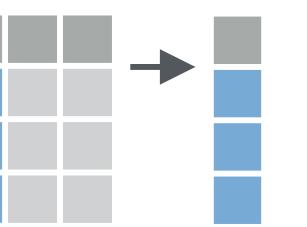
- `contains(match)`
- `ends_with(match)`
- `matches(match)`
- `num_range(prefix, range)` `::`, e.g. `mpg:cyl`
- `one_of(...)` `-`, e.g., `-Species`
- `starts_with(match)`

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

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, ...)
```

dplyr function

data frame to transform

function specific arguments



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

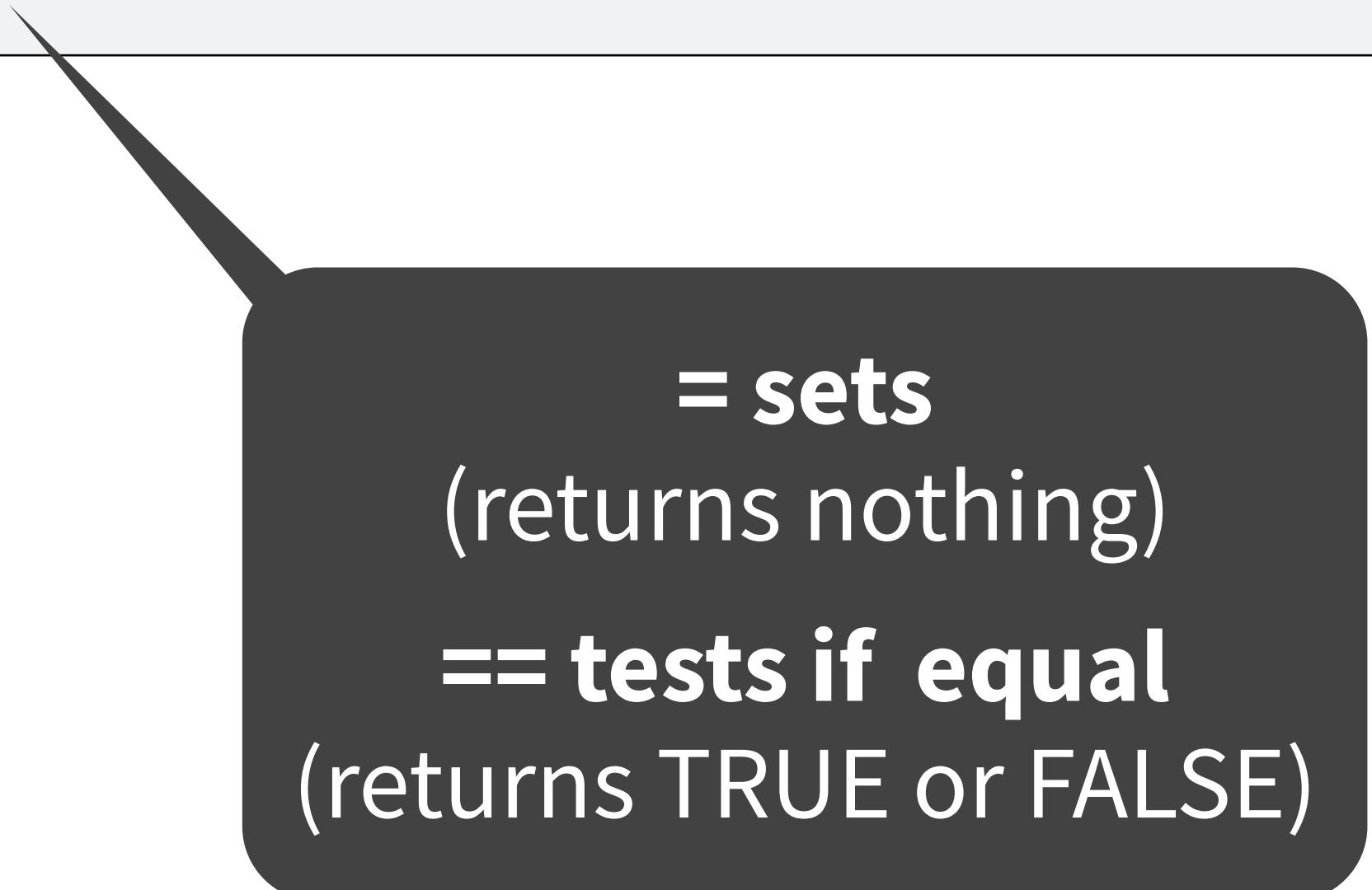


filter()

Extract rows that meet logical criteria.

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

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 < y</code>	Less than
<code>x > y</code>	Greater than
<code>x == y</code>	Equal to
<code>x <= y</code>	Less than or equal to
<code>x >= 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 those columns so that the most popular names appear near the top.

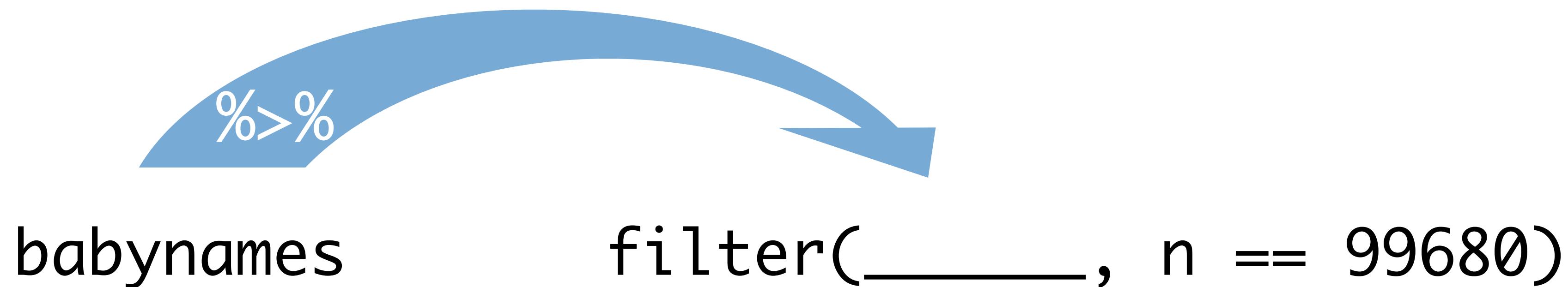
Steps

```
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)
```



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))
```

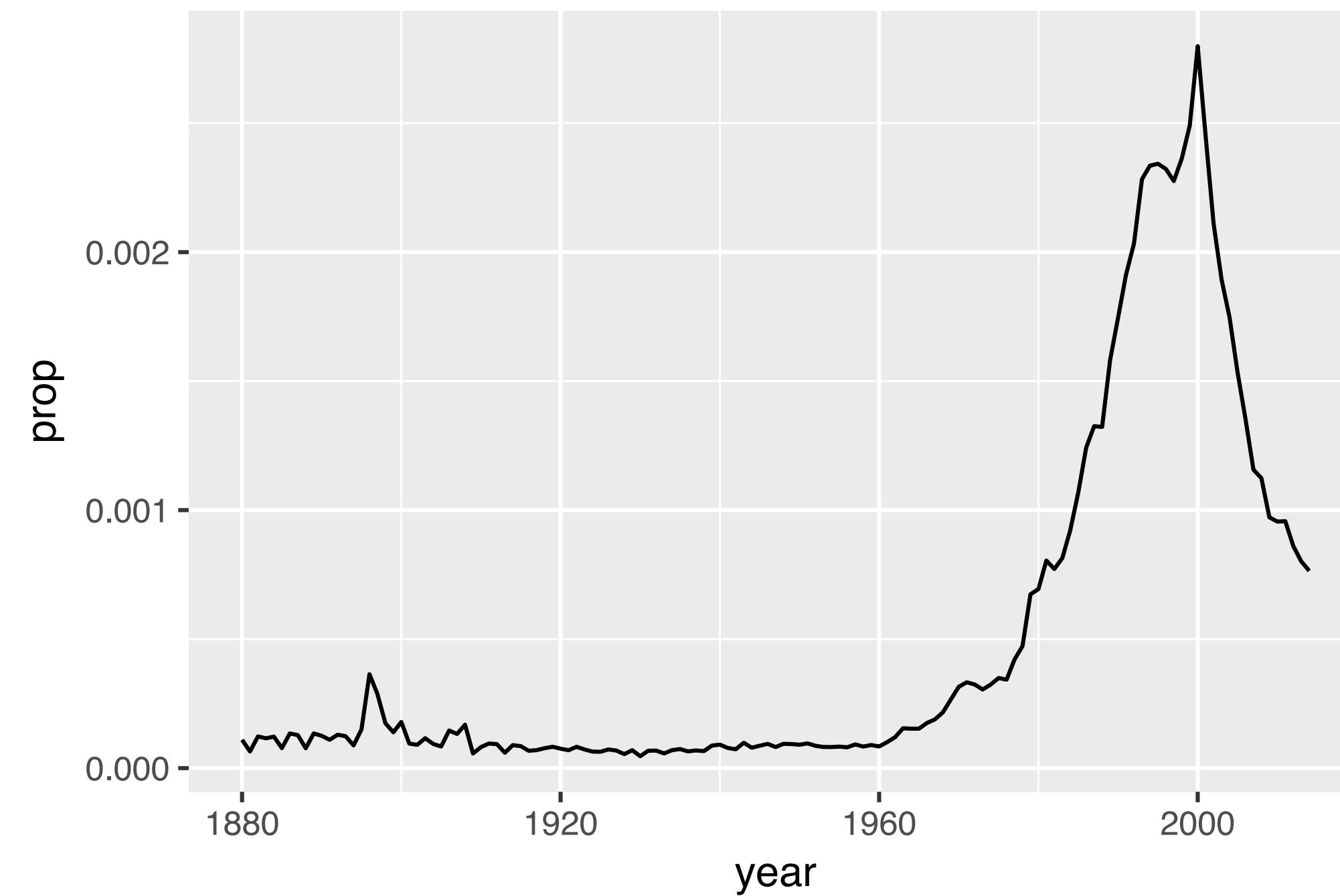
```
#          name     n  
# 1      Emma 20355  
# 2      Olivia 19553  
# 3      Sophia 17327  
# 4        Ava 16286  
# 5    Isabella 15504  
# 6        Mia 14820  
# 7    Abigail 12311  
# 8        Emily 11727  
# 9   Charlotte 11332  
# 10       Harper 10241  
# ... with 18,983 more rows
```

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 == "Garrett", sex == "M") %>%  
  select(year, prop) %>%  
  ggplot() +  
  geom_line(mapping = aes(year, prop))
```



what are the most
popular names?

How should we define popularity?

A name is popular if:



How should we define popularity?

A name is popular if:

1. **Sums** - a large number of children have the name when you sum across years



How should we define 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.



Quiz

Do we have enough information to:

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

Deriving information

summarise() - summarise **variables**

group_by() - group **cases**

mutate() - create new **variables**



summarise()

The R logo, consisting of a large, bold, italicized letter 'R' inside a circular background.

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
127538	99680



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
```



HELLO
my name is

Carl



@cdhowe

Your Turn 8

Extract the rows where **name == "Khaleesi"**. Then use `summarise()` and a summary functions 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
```



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

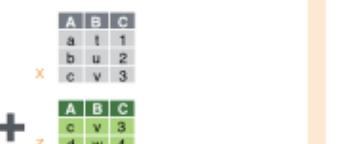
Combine Tables

Combine Variables

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

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

Combine Cases

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

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

Extract Rows

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

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

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



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
1858689



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		1858689	95025
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
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6



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



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())
```

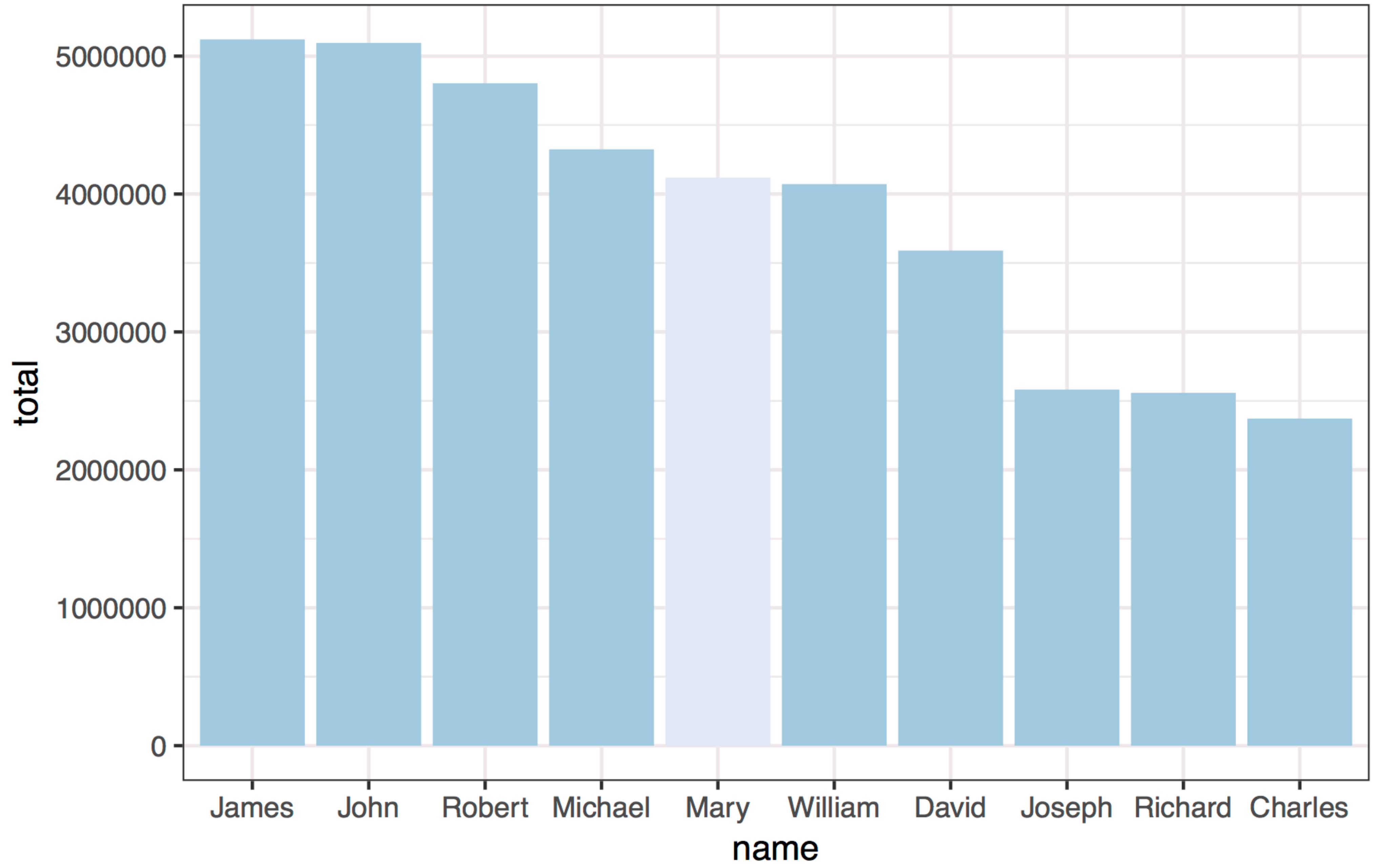
Your Turn 9

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")
```

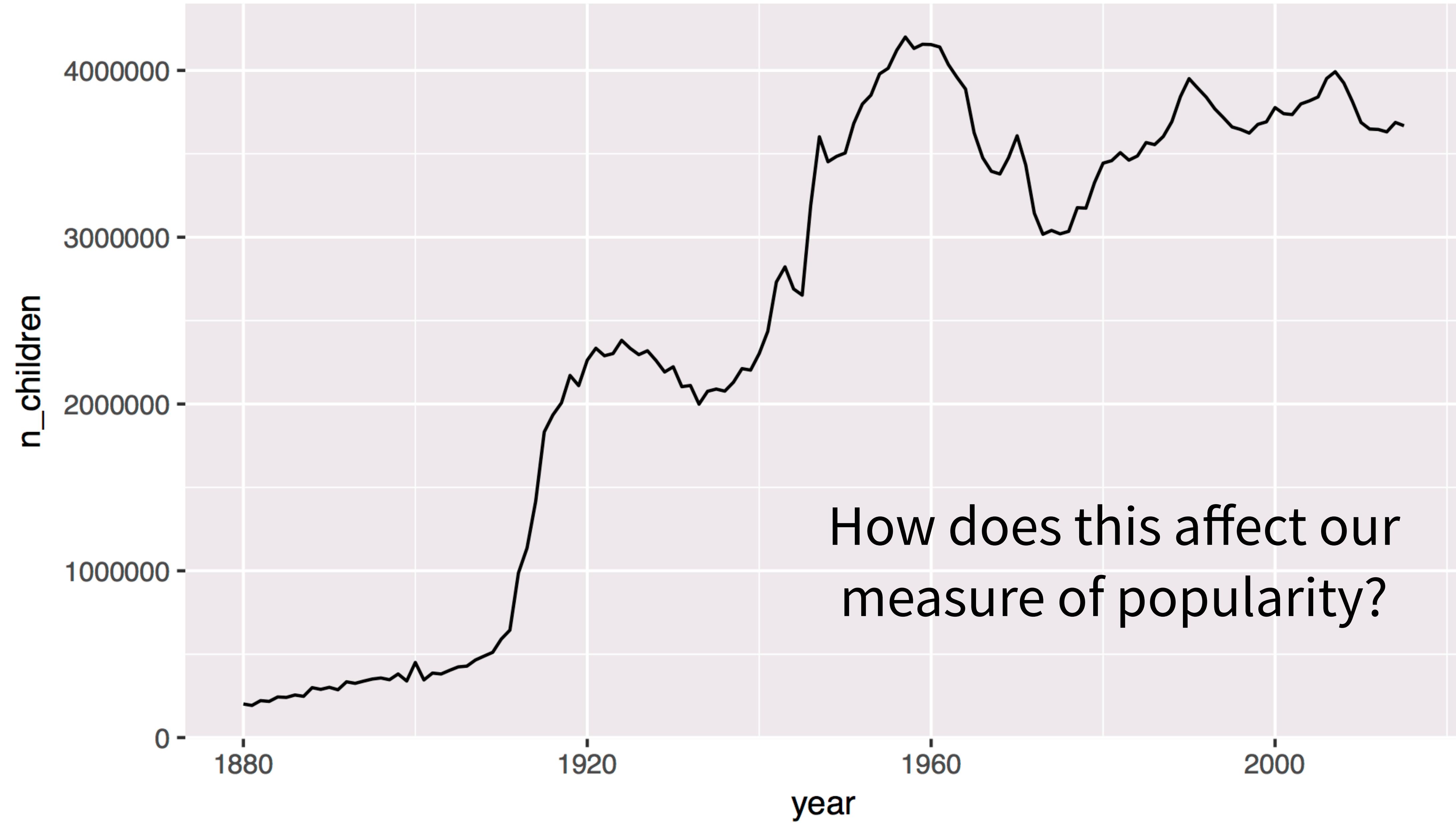
Your Turn 10

Use grouping to calculate number of children born each year.

Plot the results as a line graph.



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



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 diagram is a comprehensive guide to dplyr functions, organized into three main sections:

- Vectorized Functions**: Focuses on `mutate()` and `transmute()`. It includes sections for Offsets, Cumulative Aggregates, Rankings, Math, and Misc. A key concept is "vectorized function" (indicated by a blue arrow).
- Summary Functions**: Focuses on `summarise()`. It includes sections for Offsets, Cumulative Aggregates, Location, Logicals, Position/Order, Rank, Spread, Row names, and Misc. A key concept is "summary function" (indicated by a blue arrow).
- Combine Tables**: Focuses on joining tables. It includes sections for Combine Variables, Combine Cases, and Extract Rows. It shows how to use `bind_cols()` and `bind_rows()` for side-by-side and below-each-other joins respectively, and various join types like Mutating Join, Intersect, Setdiff, Union, and Setequal.

Each section contains examples of R code and small diagrams illustrating the data transformations.



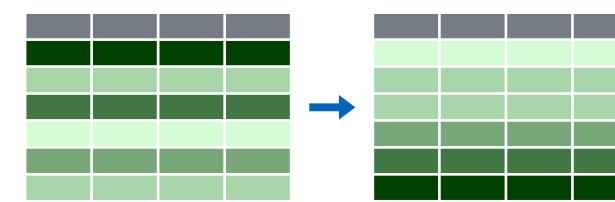
Recap: Single table verbs



Extract variables with **select()**



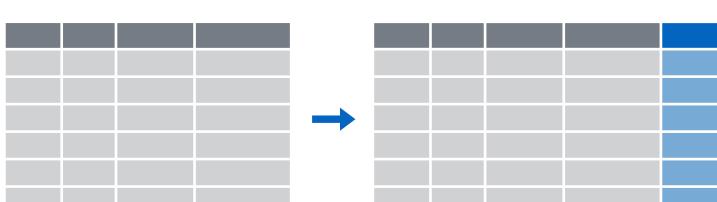
Extract cases with **filter()**



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



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



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



Transform Data with

