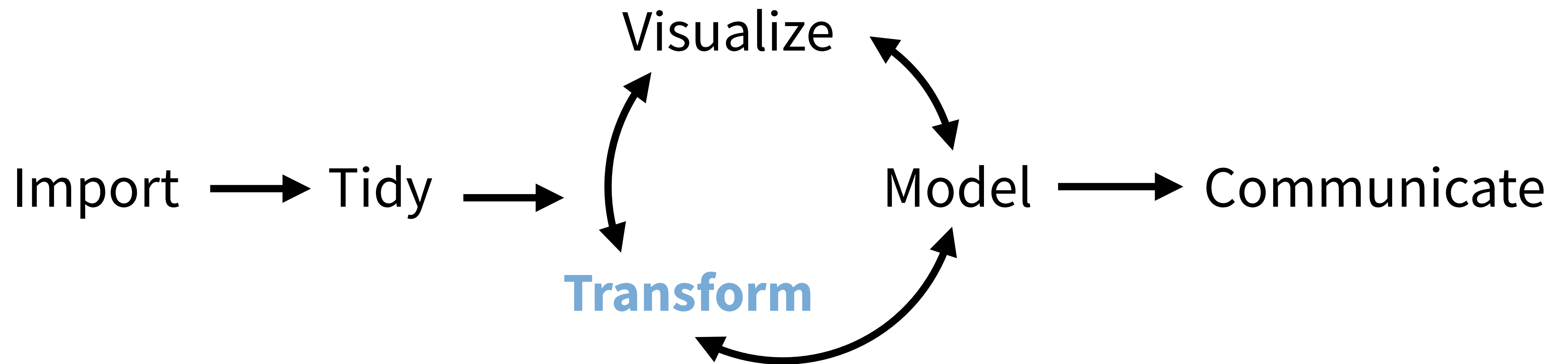


Data Transformation with



(Applied) Data Science



Program



babynames

Names of male and female babies born in the US from 1880 to 2008. 1.8M rows.



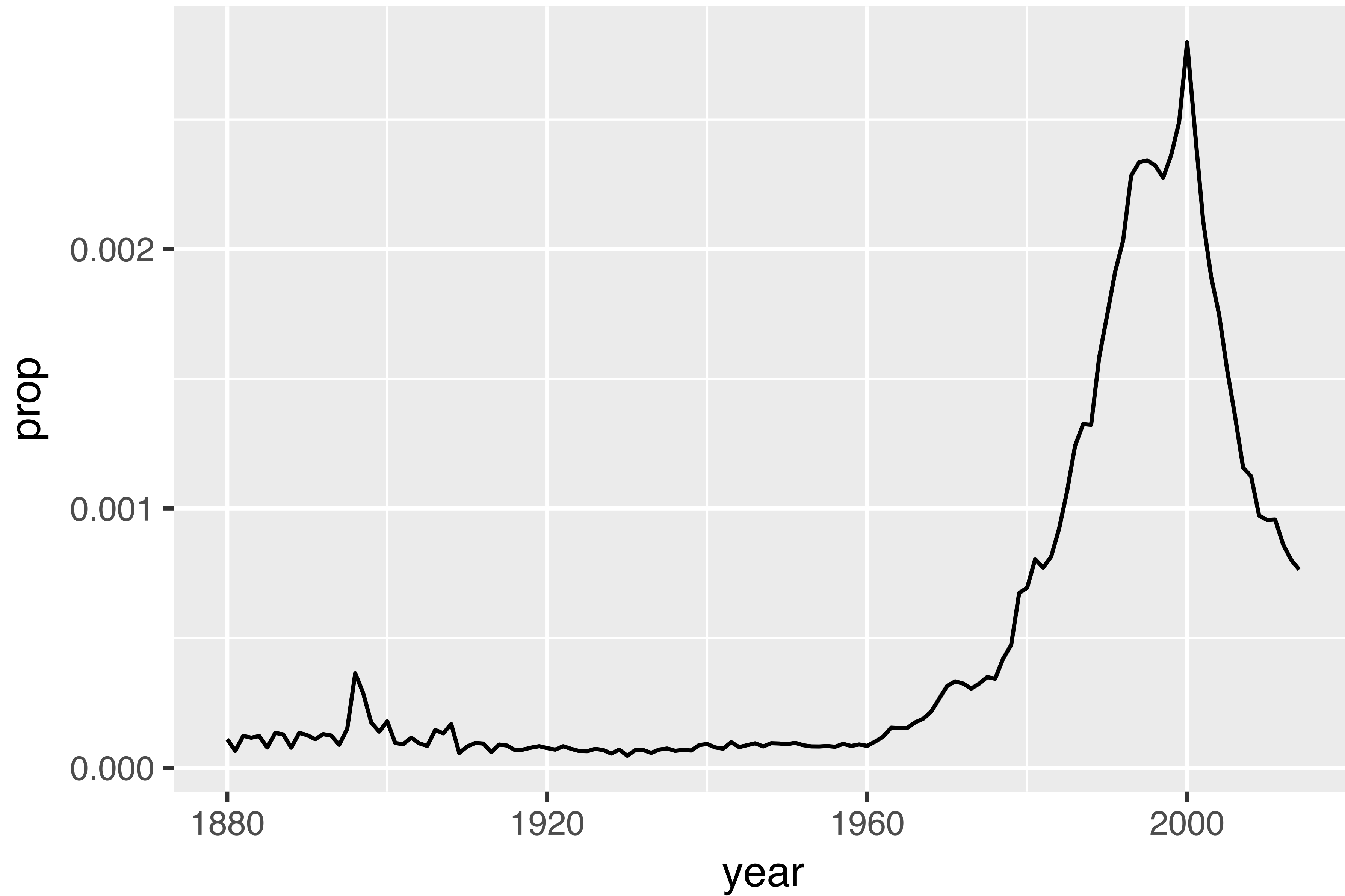
```
# install.packages("babynames")  
library(babynames)
```

```
View(babynames)
```

	year	sex	name	n	prop
1	1880	F	Mary	7065	0.0723835869
2	1880	F	Anna	2604	0.0266789611
3	1880	F	Emma	2003	0.0205214897
4	1880	F	Elizabeth	1939	0.0198657856
5	1880	F	Minnie	1746	0.0178884278
6	1880	F	Margaret	1578	0.0161672045
7	1880	F	Ida	1472	0.0150811946
8	1880	F	Alice	1414	0.0144869628
9	1880	F	Bertha	1320	0.0135238973
10	1880	F	Sarah	1288	0.0131960453
11	1880	F	Annie	1258	0.0128886840



Proportion of boys with the name Garrett



tibbles

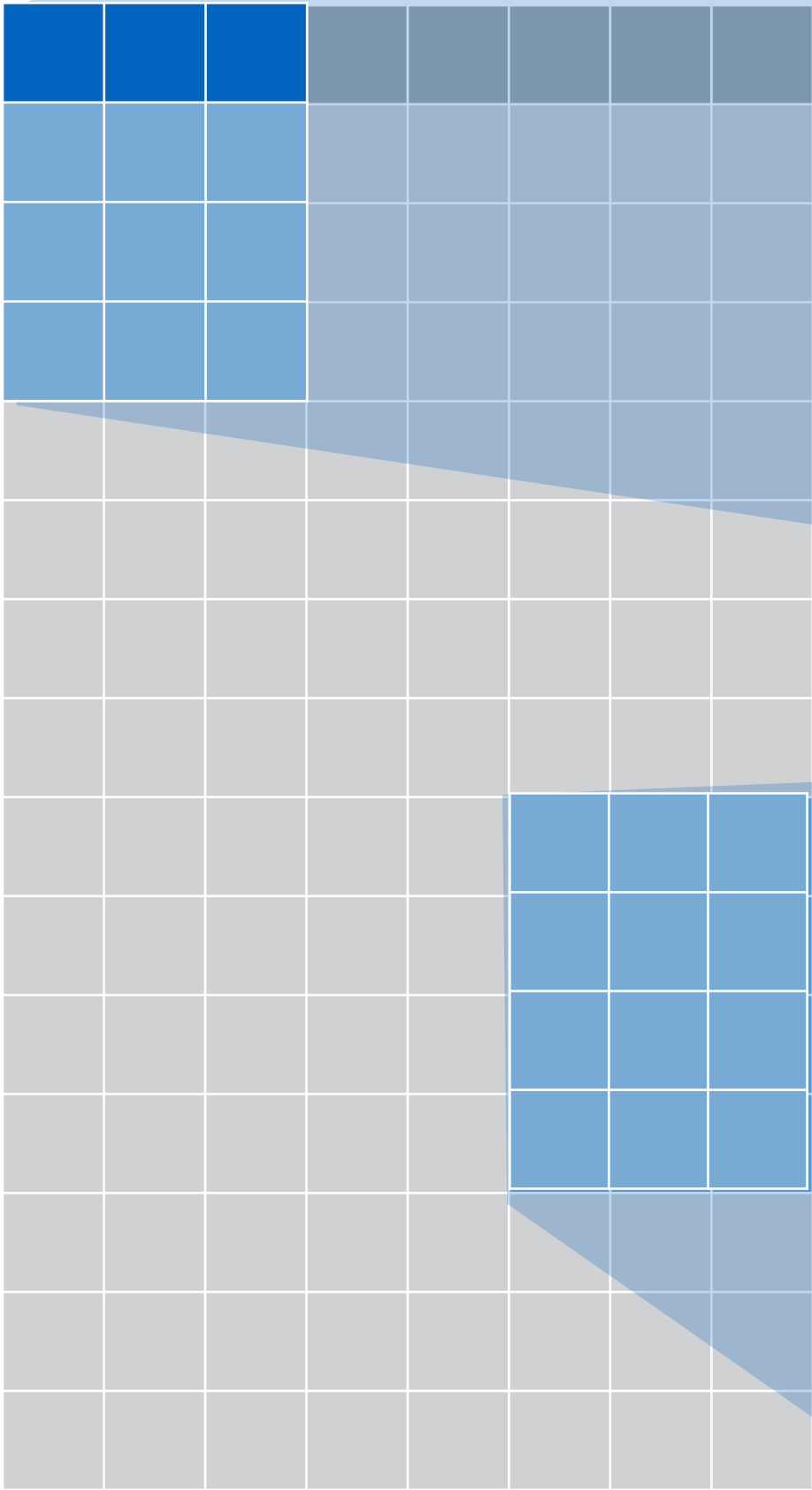


tibbles

A type of data frame common throughout tidyverse packages.
Tibbles enhance data frames in three ways:

- 1. Subsetting** - `[` always returns a new tibble, `[[` and `$` always return a new vector
- 2. No partial matching** - You must use full column names when subsetting
- 3. Display** - When you print a tibble, R provides a concise view of the data that fits on one screen





A large table to display

```
# A tibble: 234 × 6
  manufacturer      model displ
  <chr>            <chr> <dbl>
1      audi         a4    1.8
2      audi         a4    1.8
3      audi         a4    2.0
4      audi         a4    2.0
5      audi         a4    2.8
6      audi         a4    2.8
7      audi         a4    3.1
8      audi a4 quattro  1.8
9      audi a4 quattro  1.8
10     audi a4 quattro  2.0
# ... with 224 more rows, and 3
# more variables: year <int>,
# cyl <int>, trans <chr>
```

tibble display

```
156 1999      6 auto(l4)
157 1999      6 auto(l4)
158 2008      6 auto(l4)
159 2008      8 auto(s4)
160 1999      4 manual(m5)
161 1999      4 auto(l4)
162 2008      4 manual(m5)
163 2008      4 manual(m5)
164 2008      4 auto(l4)
165 2008      4 auto(l4)
166 1999      4 auto(l4)
[ reached getOption("max.print") --
omitted 68 rows ]
```

data frame display





tibble

A package with several helper functions for tibbles:

- **as_tibble()** - convert a data frame to a tibble
- **as.data.frame()** - convert a tibble to a data frame
- **tribble()** - make a tibble (transversed)

```
tribble(  
  ~x, ~y,  
  1, "a",  
  2, "b",  
  3, "c")
```

x	y
1	a
2	b
3	c



Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen.
- **Subsetting** - `[` always returns a new tibble, `[[` and `$` always return a vector.
- **No partial matching** - You must use full column names when subsetting

```
# A tibble: 234 x 6
  manufacturer    model displ
    <chr>         <chr>   <dbl>
1      audi      a4         1.8
2      audi      a4         1.8
3      audi      a4         2.0
4      audi      a4         2.0
5      audi      a4         2.8
6      audi      a4         2.8
7      audi      a4         3.1
8      audi a4 quattro  1.8
9      audi a4 quattro  1.8
10     audi a4 quattro  2.0
# ... with 224 more rows, and 3
# more variables: year <int>,
# cyl <int>, trans <chr>
```

tibble display

```
156 1999 6 auto(l4)
157 1999 6 auto(l4)
158 2008 6 auto(l4)
159 2008 8 auto(s4)
160 1999 4 manual(m5)
161 1999 4 auto(l4)
162 2008 4 manual(m5)
163 2008 4 manual(m5)
164 2008 4 auto(l4)
165 2008 4 auto(l4)
166 1999 4 auto(l4)
# reached getOption("max.print")
# omitted 68 rows
```

data frame display

A large table to display

- Control the default appearance with options:


```
options(tibble.print_max = n,
        tibble.print_min = m, tibble.width = Inf)
```
- View entire data set with **View(x, title)** or **glimpse(x, width = NULL, ...)**
- Revert to data frame with **as.data.frame()** (required for some older packages)

Construct a tibble in two ways

tibble(...)

Construct by columns.

```
tibble(x = 1:3,
       y = c("a", "b", "c"))
```

Both make this tibble

tribble(...)

Construct by rows.

```
tribble(
  ~x, ~y,
  1, "a",
  2, "b",
  3, "c")
```

```
A tibble: 3 x 2
  x     y
<int> <dbl>
1     1  a
2     2  b
3     3  c
```

as_tibble(x, ...) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Converts named vector to a tibble with a names column and a values column.

is_tibble(x) Test whether x is a tibble.

tibbles

Tibbles - an enhanced data frame

The tibble package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen.
- **Subsetting** - `[` always returns a new tibble, `[[` and `$` always return a vector.
- **No partial matching** - You must use full column names when subsetting

tibble display

```
# A tibble: 234 x 6
  manufacturer    model displ
    <chr>         <chr>   <dbl>
1      audi      a4         1.8
2      audi      a4         1.8
3      audi      a4         2.0
4      audi      a4         2.0
5      audi      a4         2.8
6      audi      a4         2.8
7      audi      a4         3.1
8      audi a4 quattro  1.8
9      audi a4 quattro  1.8
10     audi a4 quattro  2.0
# ... with 224 more rows, and 3
# more variables: year <int>,
# cyl <int>, trans <chr>
```

data frame display

```
156 1999 6 auto(l4)
157 1999 6 auto(l4)
158 2008 6 auto(l4)
159 2008 8 auto(s4)
160 1999 4 manual(m5)
161 1999 4 auto(l4)
162 2008 4 manual(m5)
163 2008 4 manual(m5)
164 2008 4 auto(l4)
165 2008 4 auto(l4)
166 1999 4 auto(l4)
# reached getOption("max.print")
# omitted 68 rows
```

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages. A table is tidy if:

- Each variable is in its own column
- Each observation, or case, is in its own row
- Makes variables easy to access as vectors
- Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout. Each uses the idea of a key column: value column pair.

gather(data, key, value, ...) `na.rm = FALSE`, `convert = FALSE`, `factor_key = FALSE`

Gather moves column names into a key column, gathering the column values into a single value column.

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, set = NULL)

Spread moves the unique values of a key column into the column names, spreading the values of a value column across the new columns that result.

Split and Combine Cells

Use these functions to split or combine cells into individual, atomic values.

separate(data, col_name, sep = "[^a-zA-Z]", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn")

Separate each cell in a column to make several columns.

unite(data, col, ... sep = "", remove = TRUE)

Collapse cells across several columns to make a single column.

Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in ... columns.

fill(data, ... direction = c("down", "up"), fill = NA)
Fill in NA's in ... columns with most recent non-NA values.

replace_na(data, replace = list(), ...)
Replace NA's by column.

Expand Tables - quickly create tables with combinations of values

complete(data, ... fill = list())
Adds to the data missing combinations of the values of the variables listed in ...

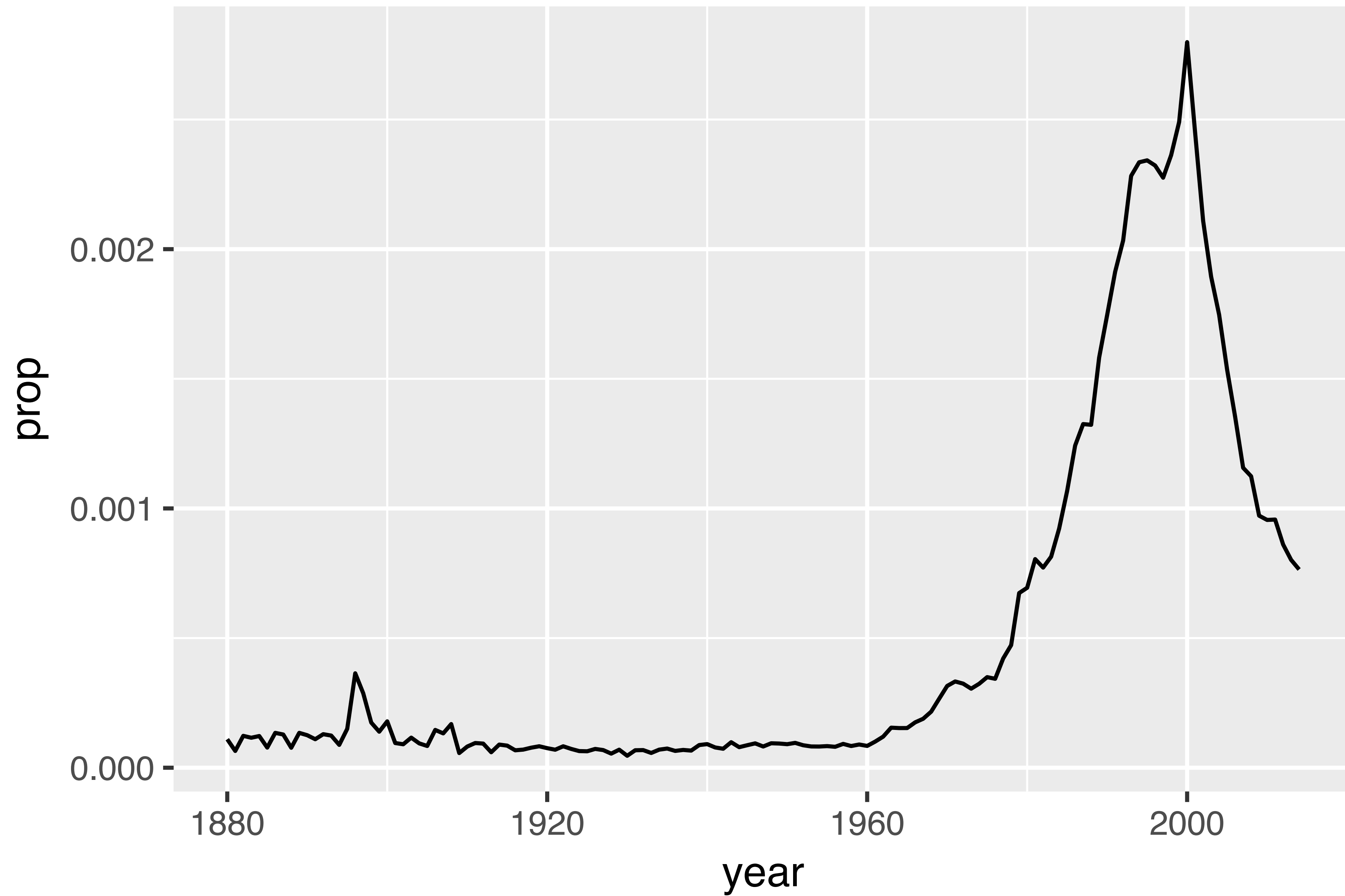
expand_grid(...)
Create new tibble with all possible combinations of the values of the variables listed in ...

Learn more at

tidyverse.com

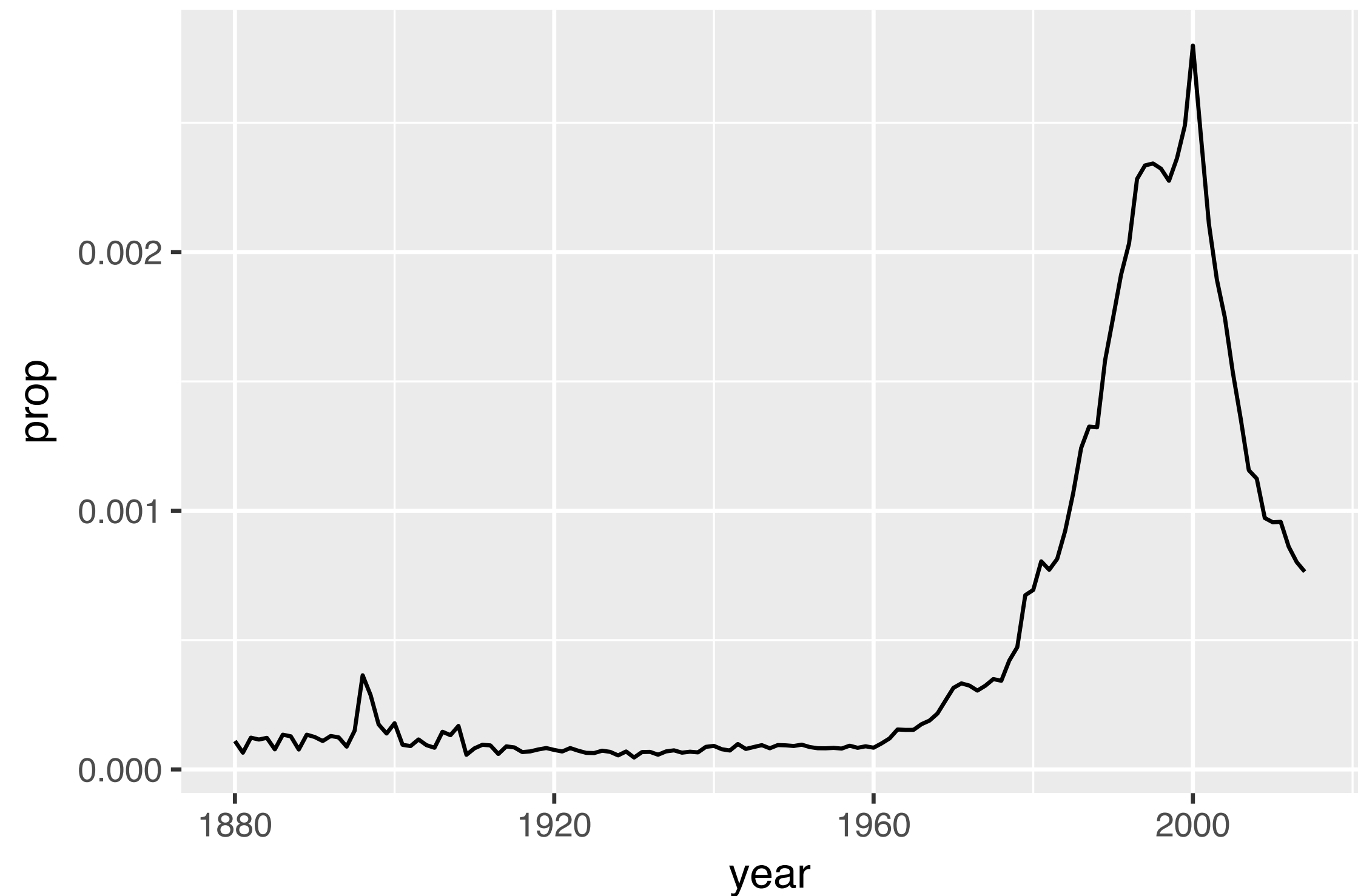


Proportion of boys with the name Garrett



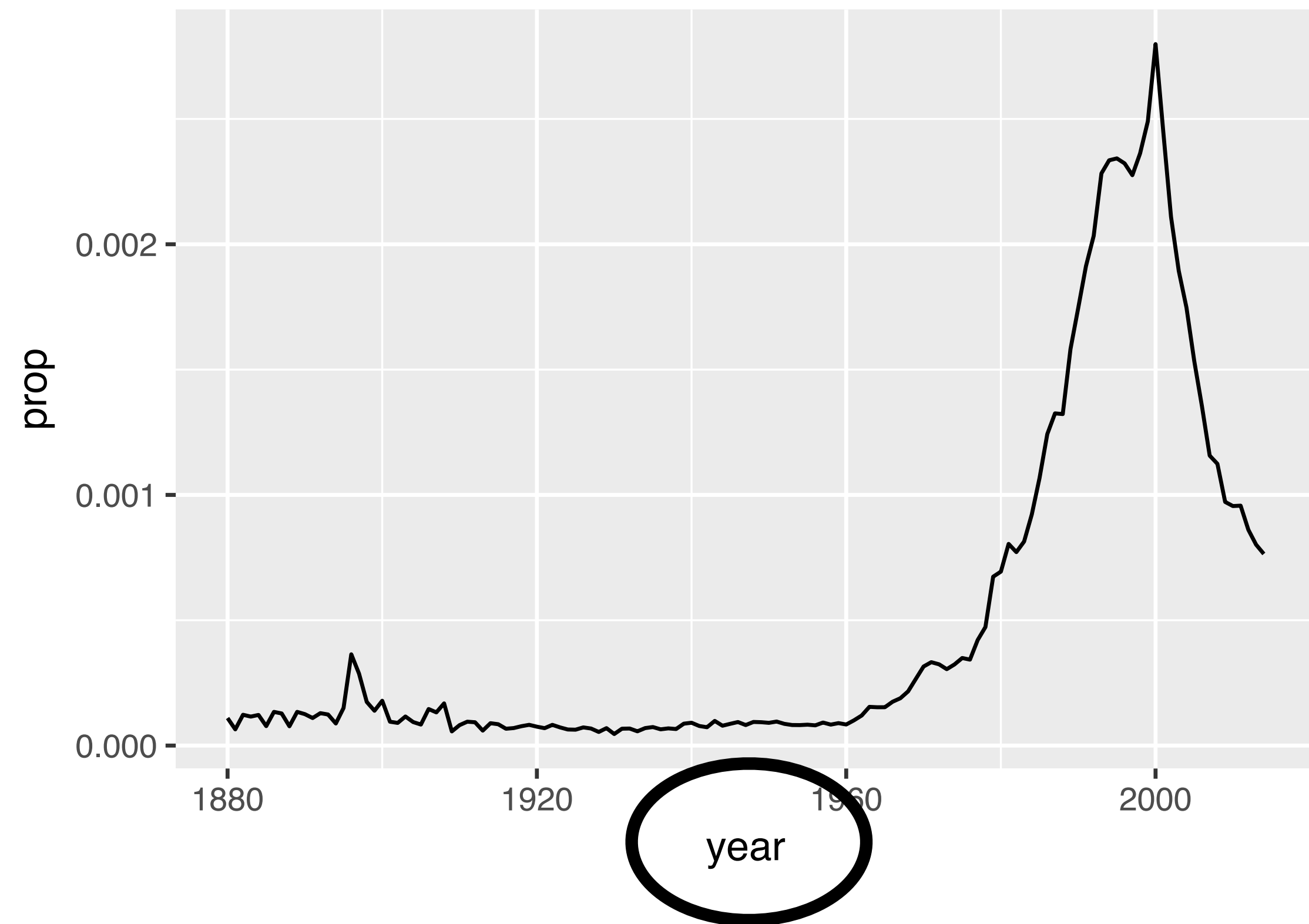
What variables do we need?

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



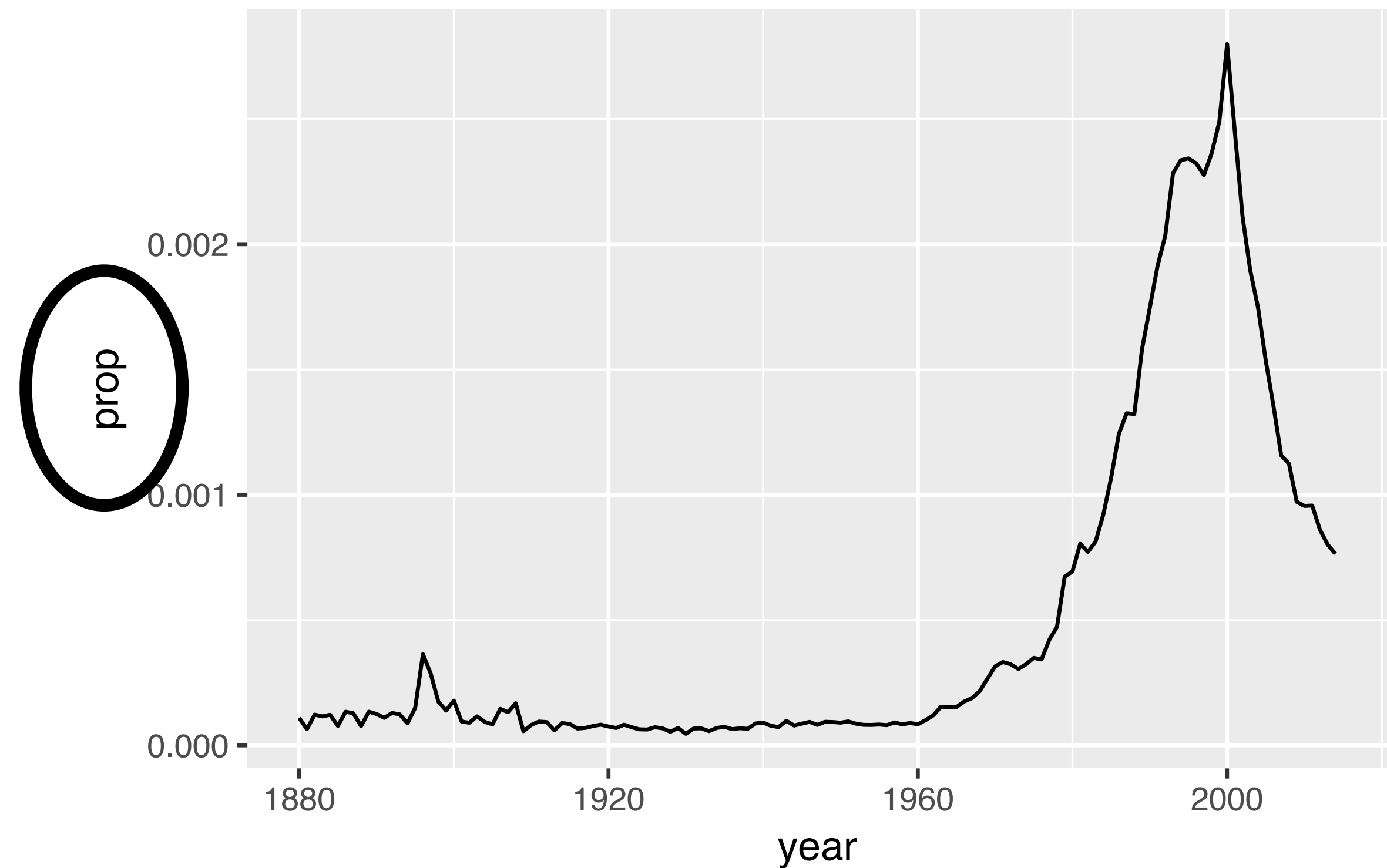
What variables do we need?

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



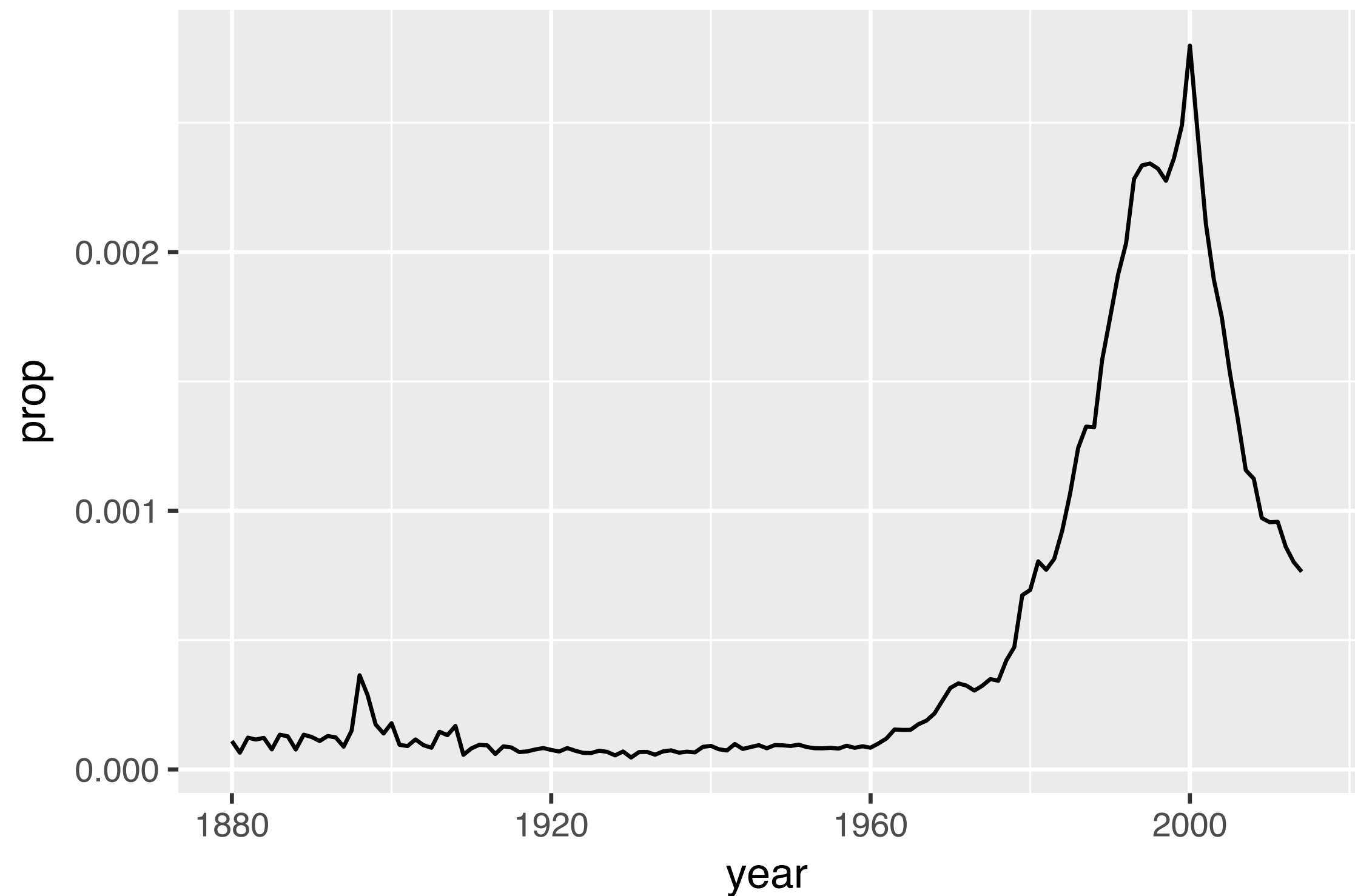
What variables do we need?

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



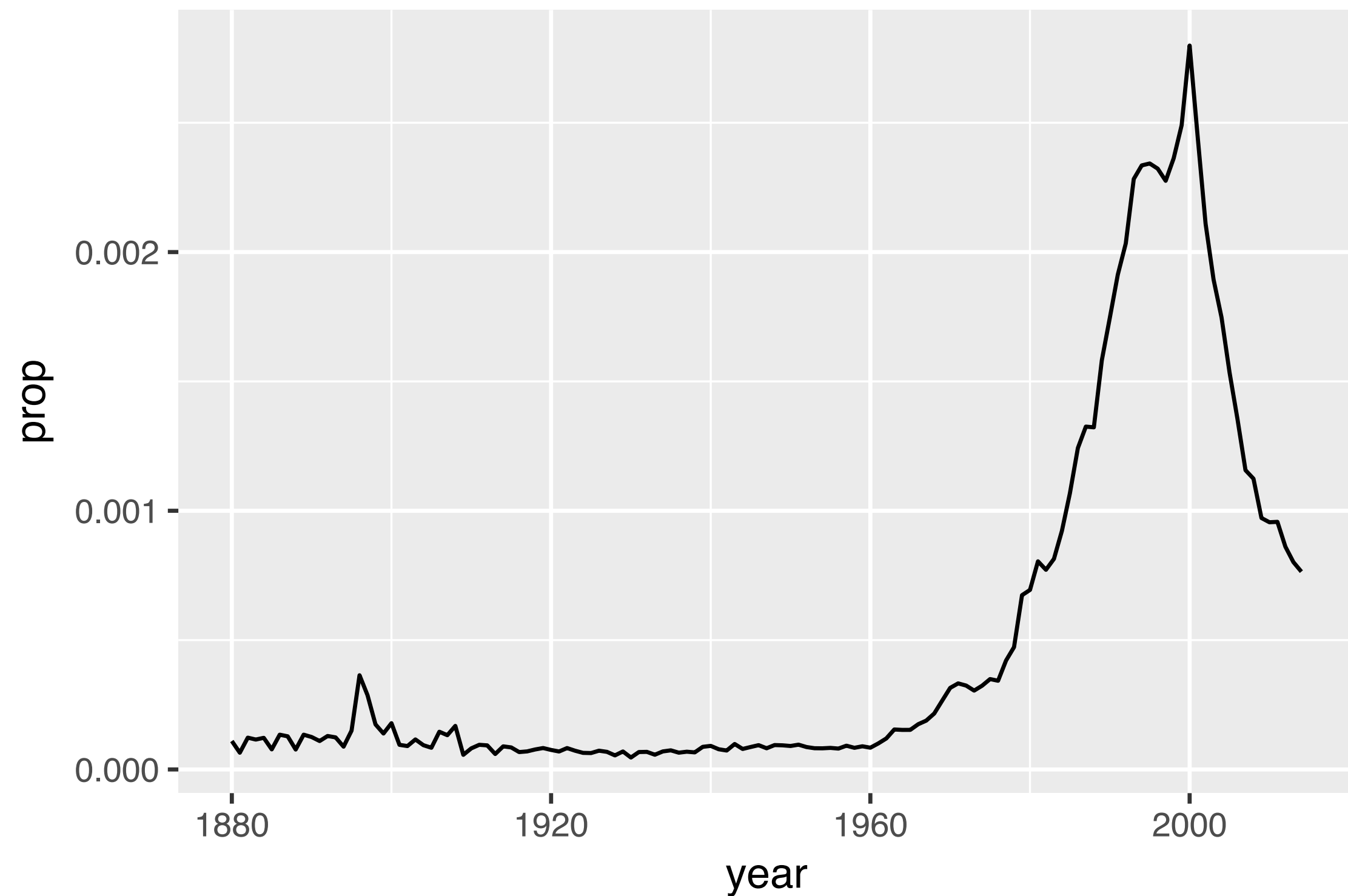
What cases do we need?

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



What cases do we need?

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



What cases do we need?

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.



single table verbs

filter() - extract **cases**

arrange() - reorder **cases**

group_by() - group **cases**

select() - extract **variables**

mutate() - create new **variables**

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



two table verbs

bind_rows() - adds one table to another as new **cases**

union(), **intersect()**, **setdiff()** - set operations for **cases**

semi_join(), **anti_join()** - filters **cases** in one table against another

bind_cols() - adds one table to another as new **variables**

left_join(), **right_join()**, **full_join()**, **inner_join()** - mutates one table by matching values from another as new **variables**



Toy data

```
storms <- tribble(
  ~storm, ~wind, ~pressure, ~date,
  "Alberto", 110, 1007, "2000-08-12",
  "Alex", 45, 1009, "1998-07-30",
  "Allison", 65, 1005, "1995-06-04",
  "Ana", 40, 1013, "1997-07-01",
  "Arlene", 50, 1010, "1999-06-13",
  "Arthur", 45, 1010, "1996-06-21"
)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

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)



filter()

Extract rows that meet logical criteria.

```
filter(storms, wind == 45)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

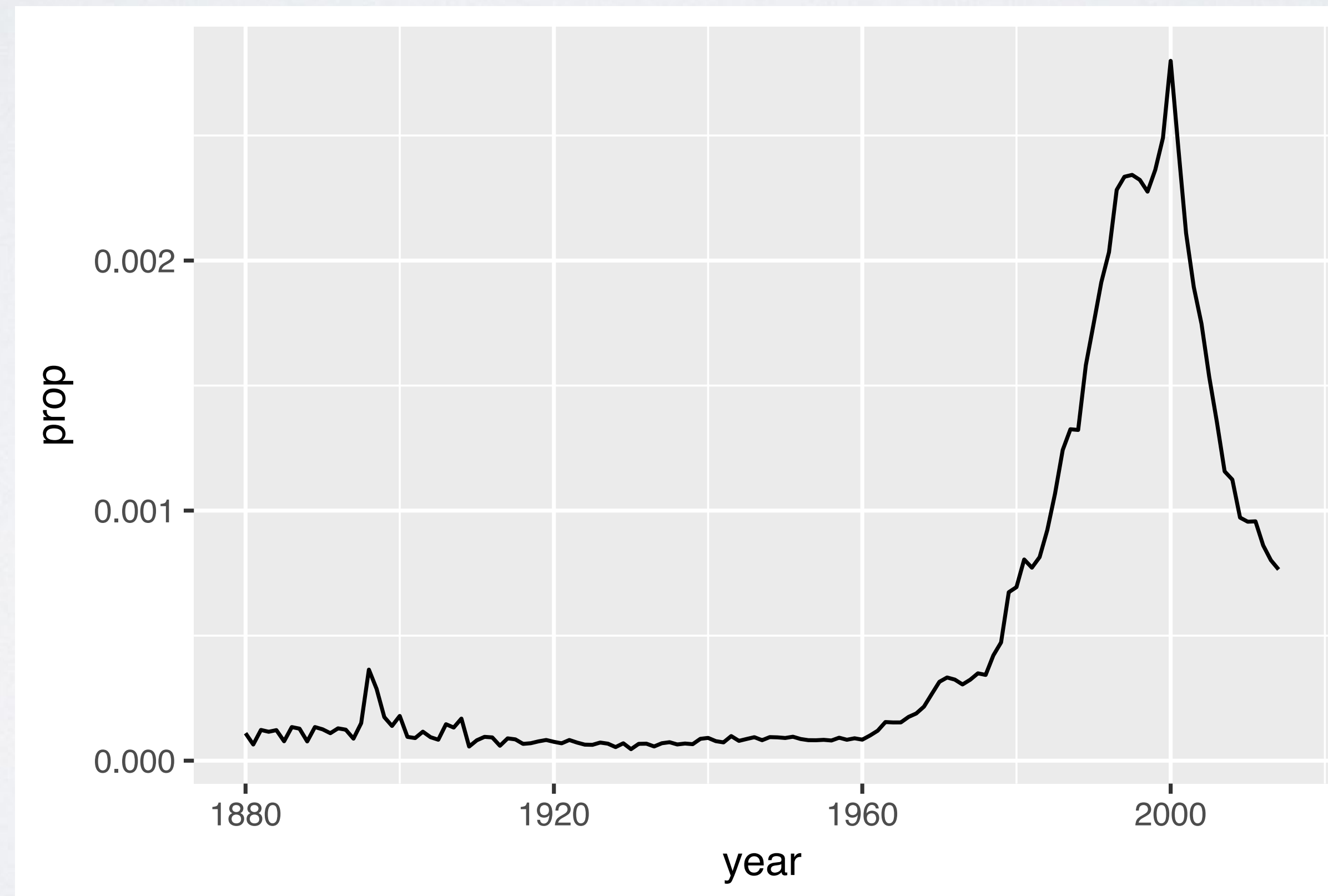
storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

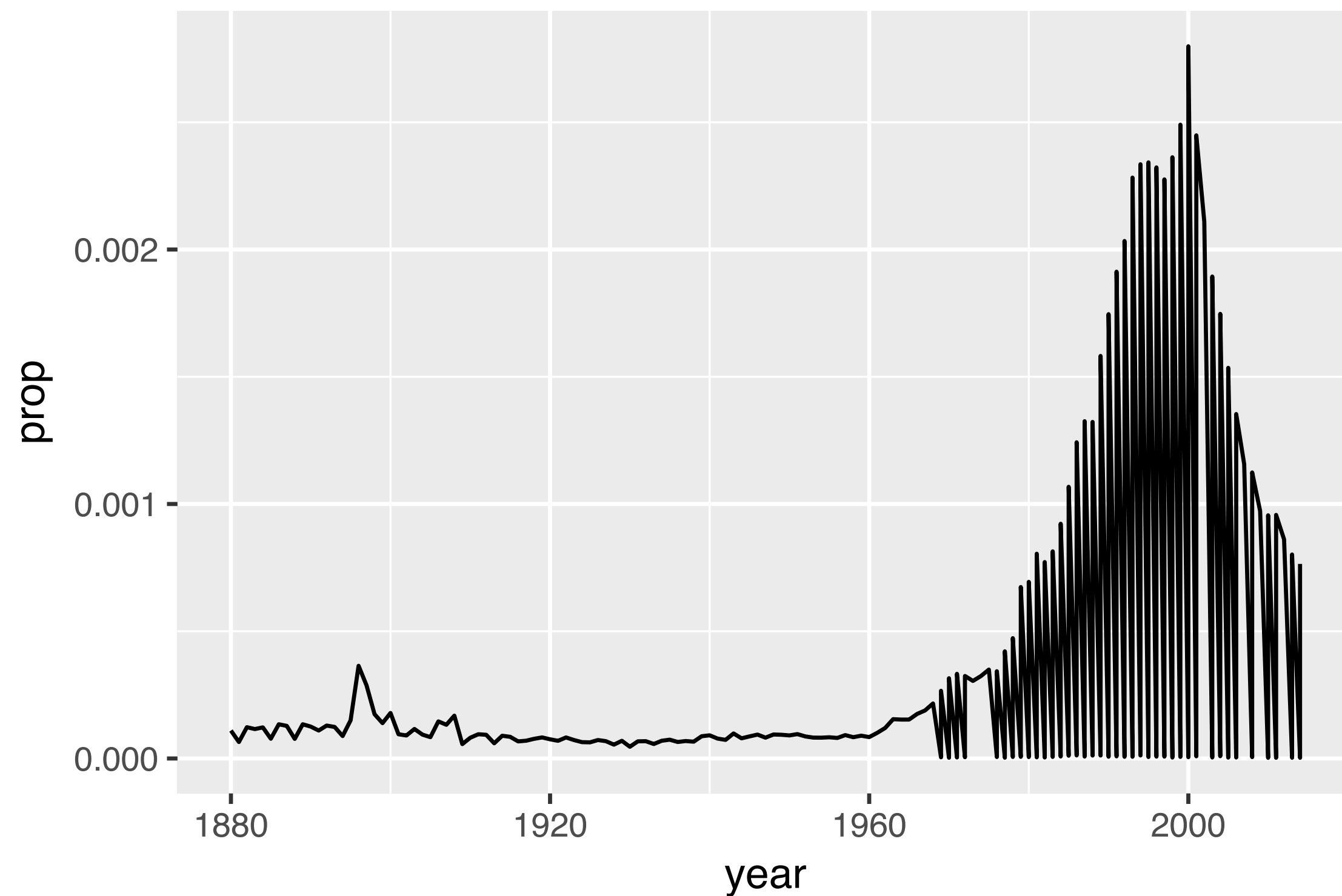
Your Turn

Use `filter()` to extract all of the rows in `babynames` that feature your name. Then use the result to recreate the plot below for your name:

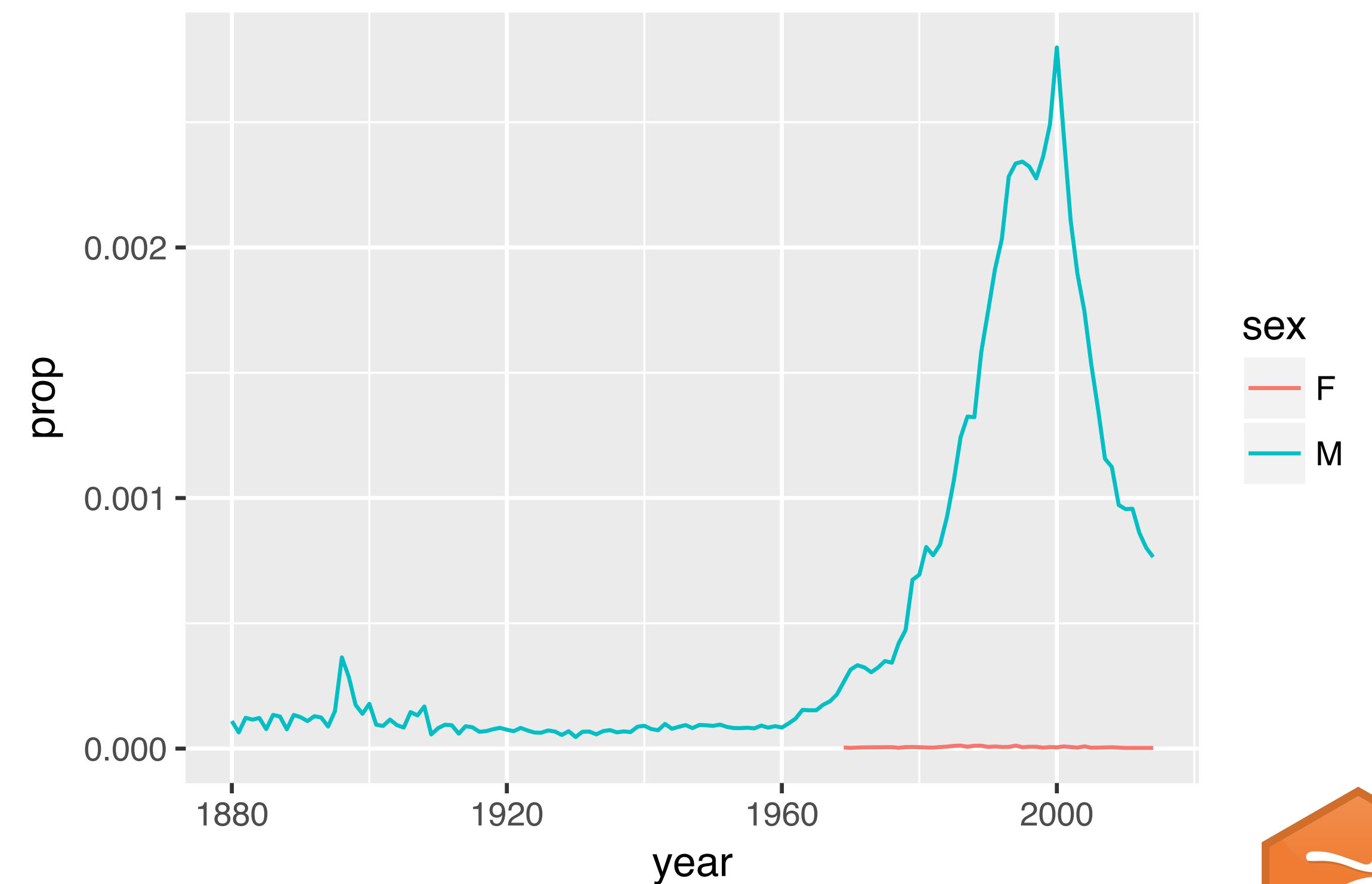
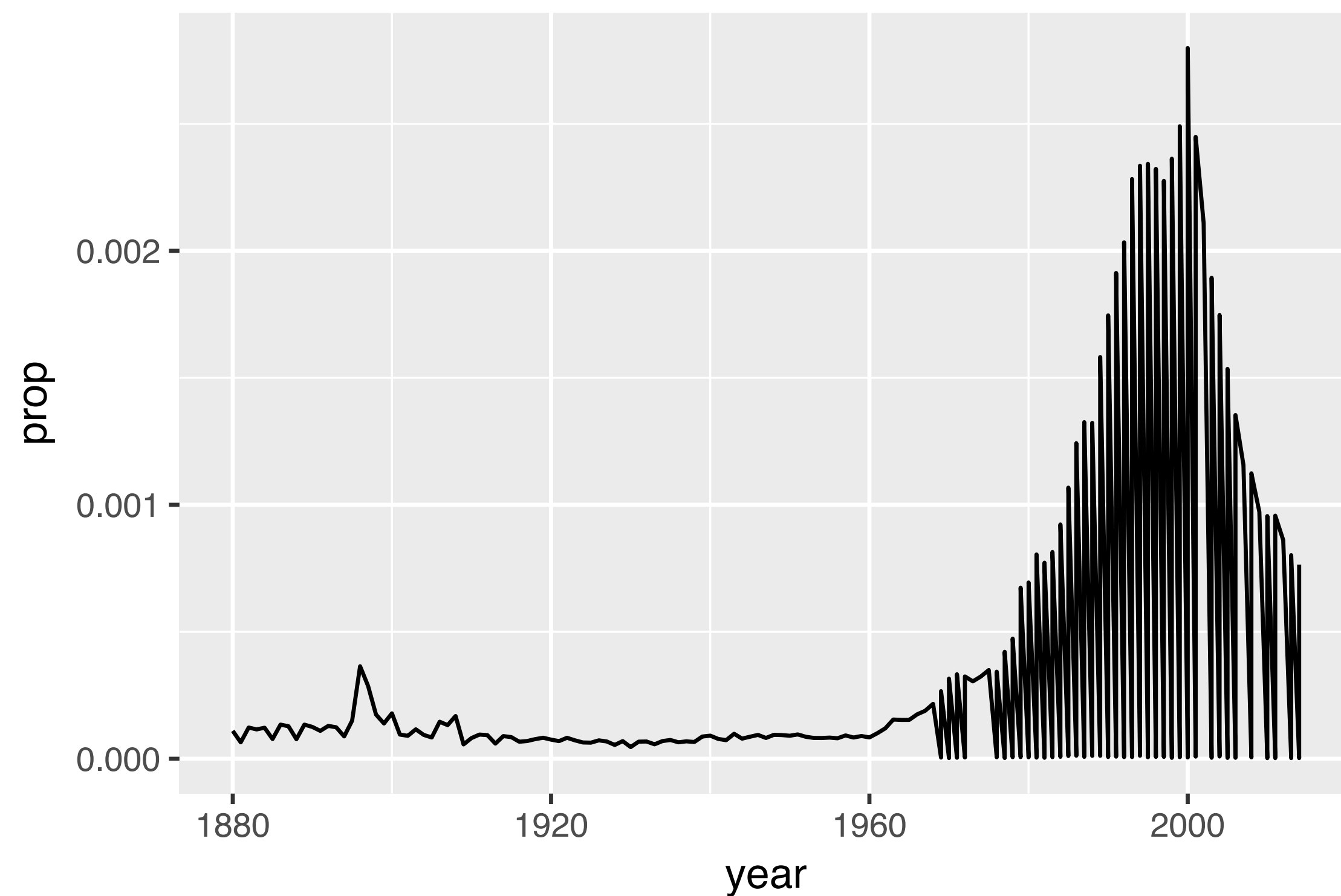


05:00

```
garrett <- filter(babynames, name == "Garrett")  
ggplot(data = garrett, mapping = aes(year, prop)) +  
  geom_line()
```



"Whipsawing" suggests that you are trying to plot two groups with a single line.



filter()

Extract rows that meet logical criteria.

```
filter(storms, wind == 45)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21

filter()

Extract rows that meet logical criteria.

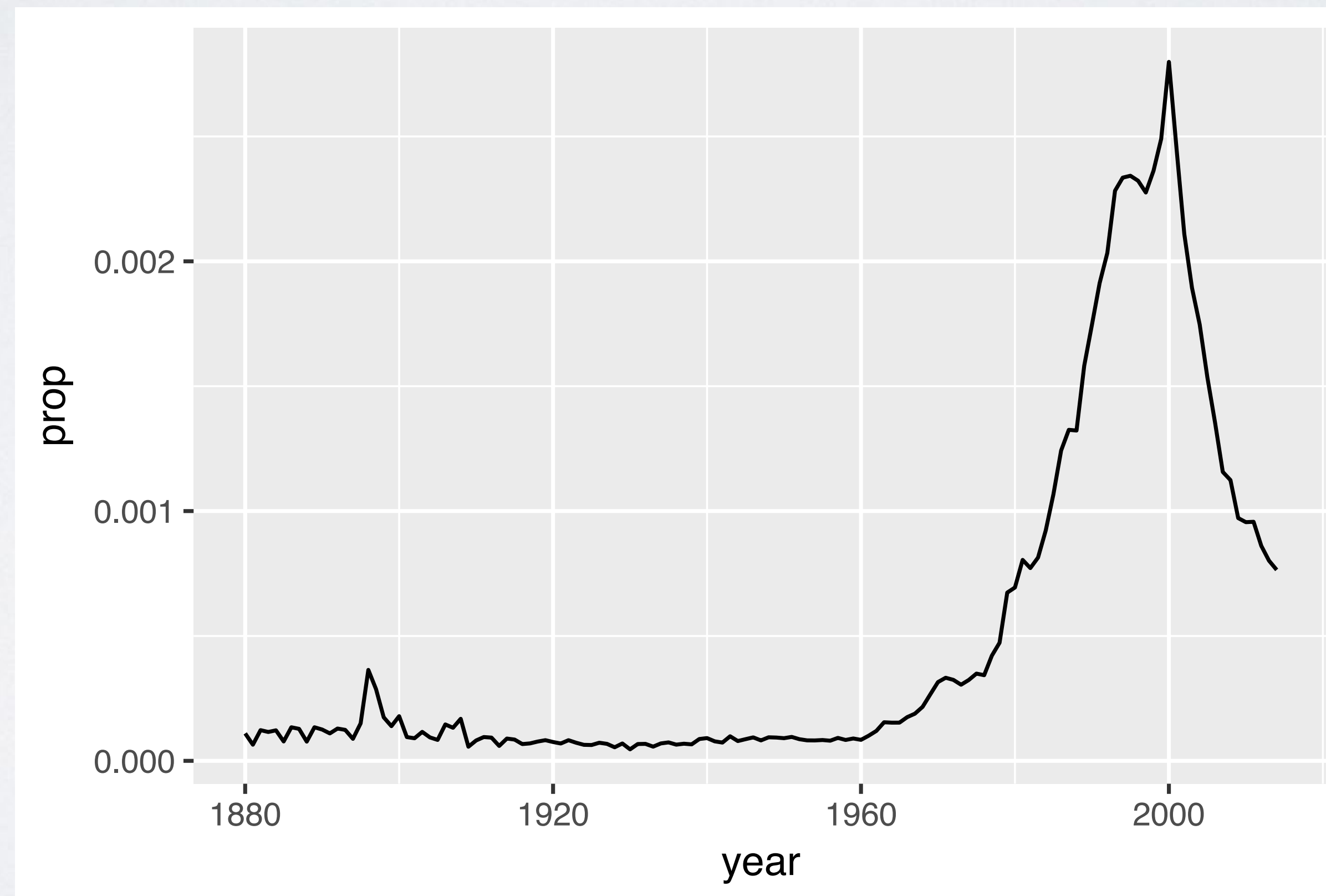
```
filter(storms, wind == 45, pressure == 1009)
```

storms

storm	wind	pressure	date		storm	wind	pressure	date
Alberto	110	1007	2000-08-12	→	Alex	45	1009	1998-07-30
Alex	45	1009	1998-07-30					
Allison	65	1005	1995-06-04					
Ana	40	1013	1997-07-01					
Arlene	50	1010	1999-06-13					
Arthur	45	1010	1996-06-21					

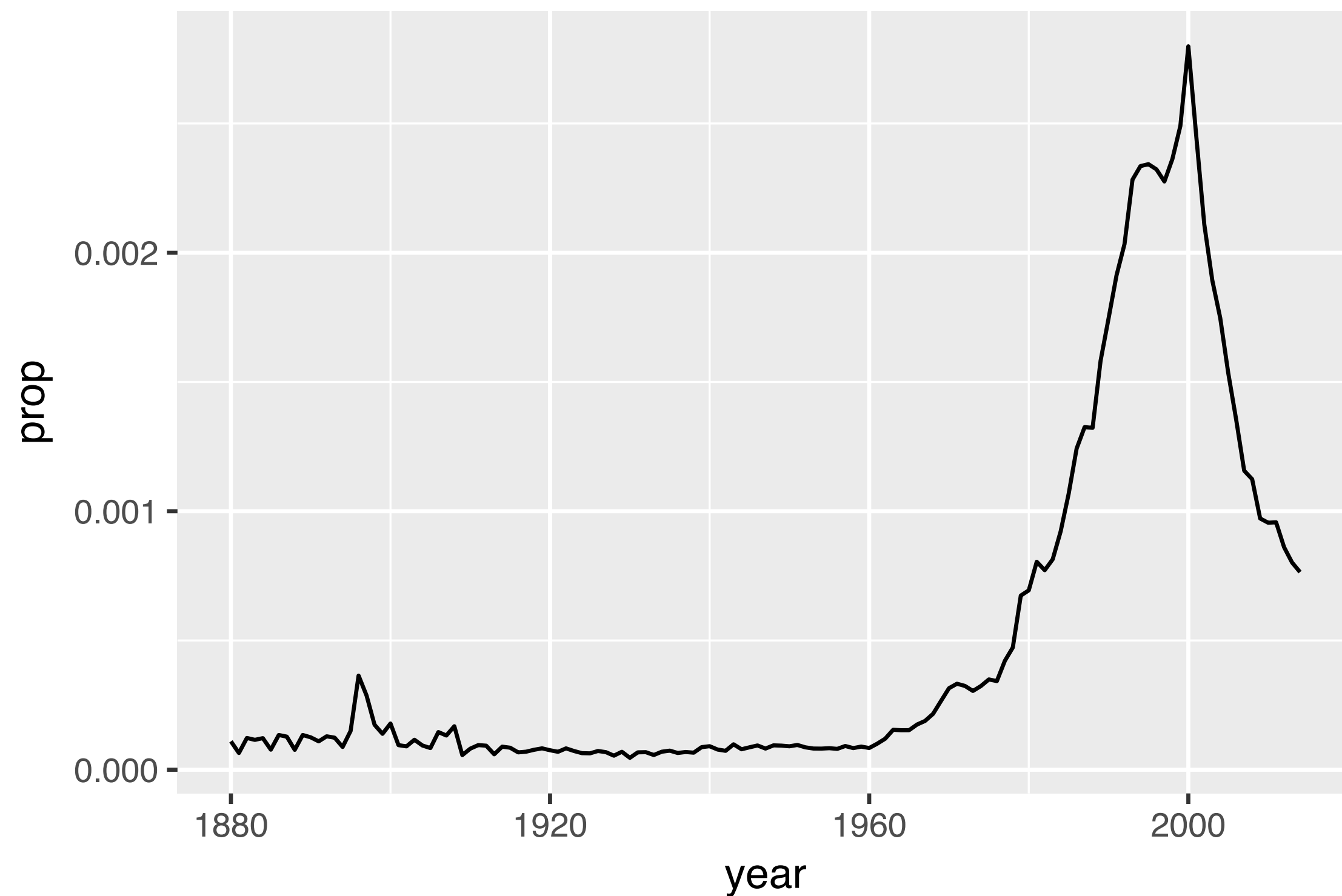
Your Turn

Use `filter()` to extract rows with your name *and* sex then recreate your plot.



02:00

```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
ggplot(data = garrett, mapping = aes(year, prop)) +  
  geom_line()
```



Logical tests

?Comparison

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

?base::Logic

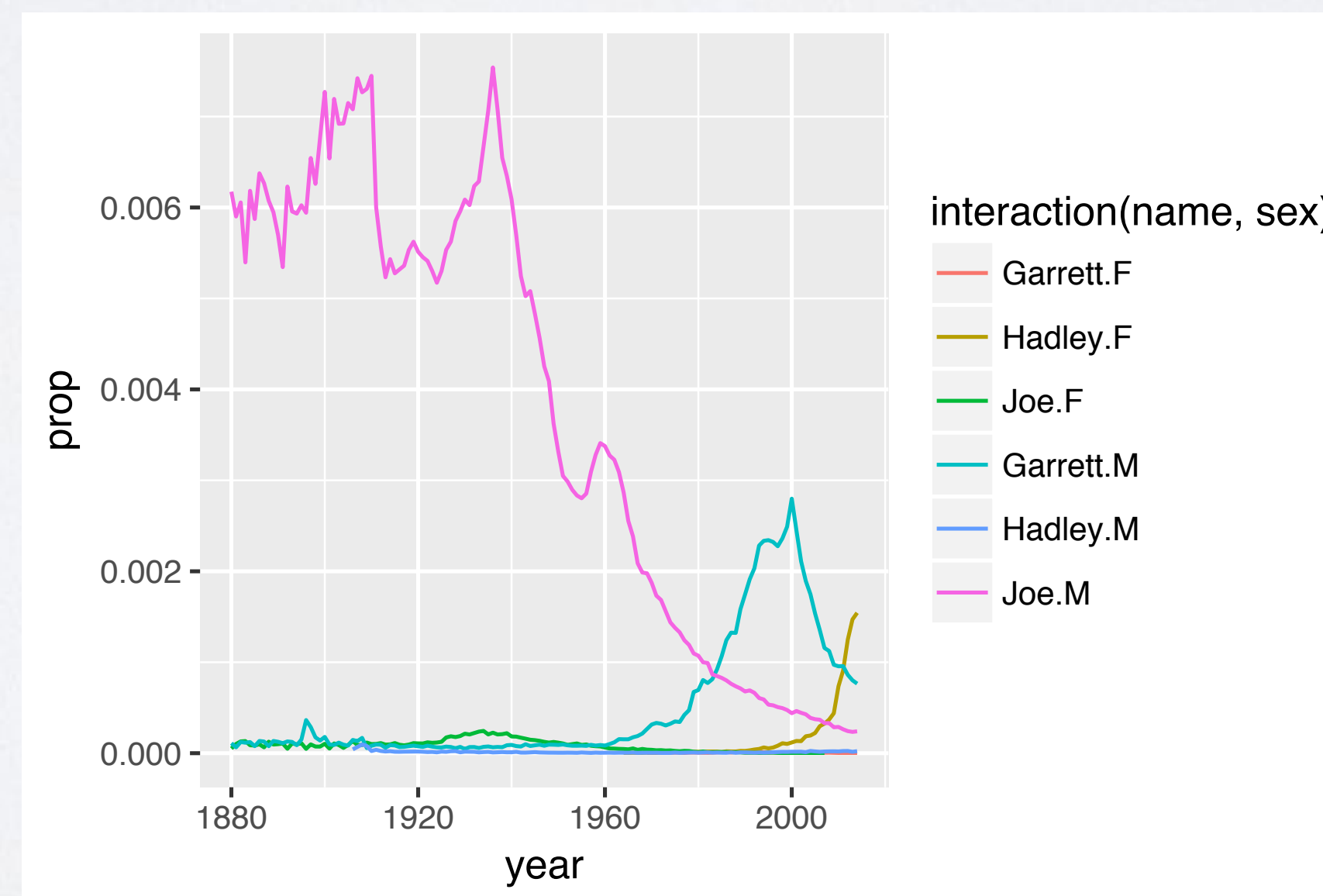
&	and
	or
xor()	exactly or
!	not
any()	any true
all()	all true



Your Turn

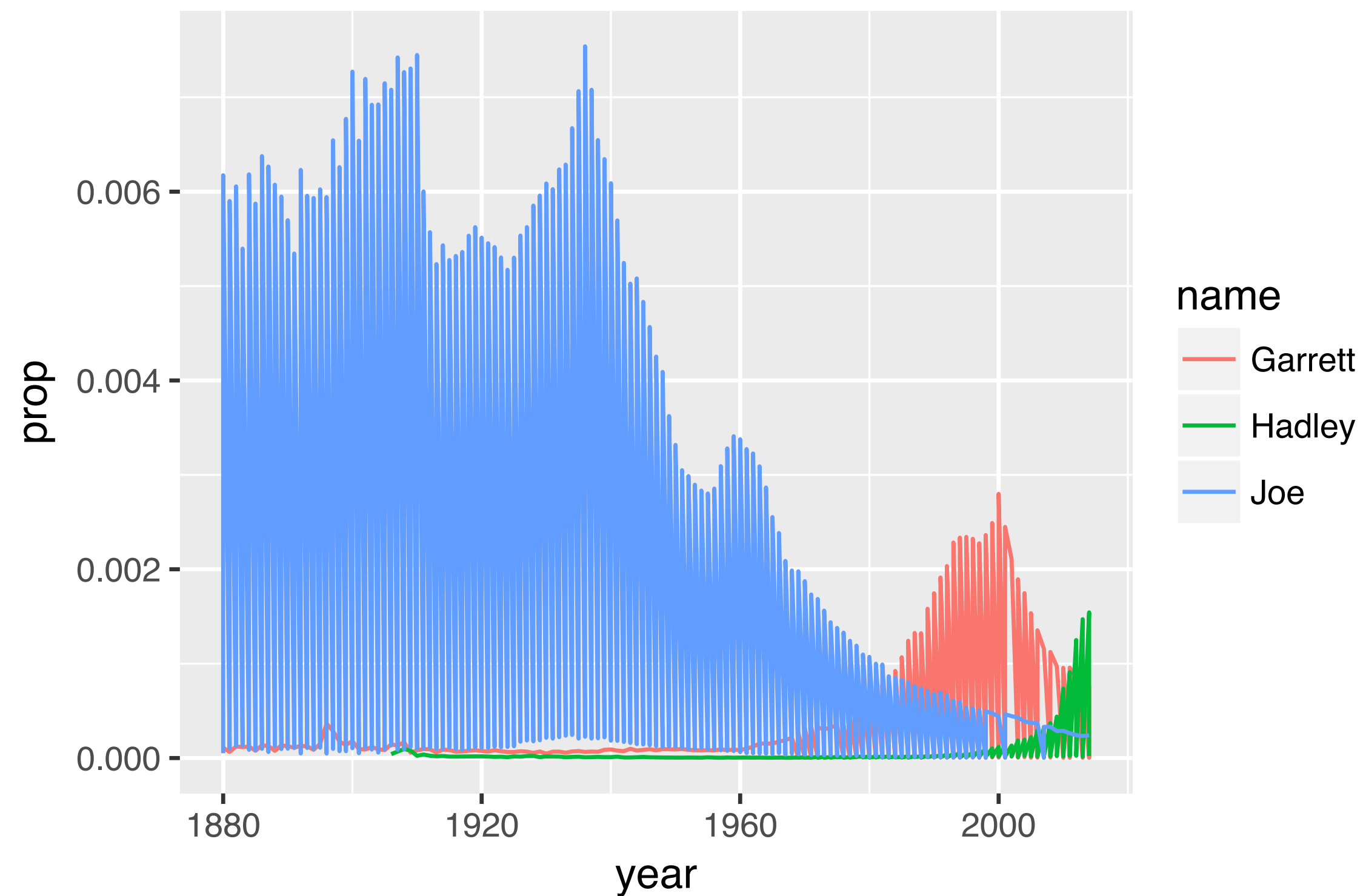
Compare the popularity of names within your group:

1. Create a vector that contains the name of each of your group members.
2. Extract all rows whose name value appears in the vector
3. Recreate the plot. Choose an aesthetic to distinguish different names

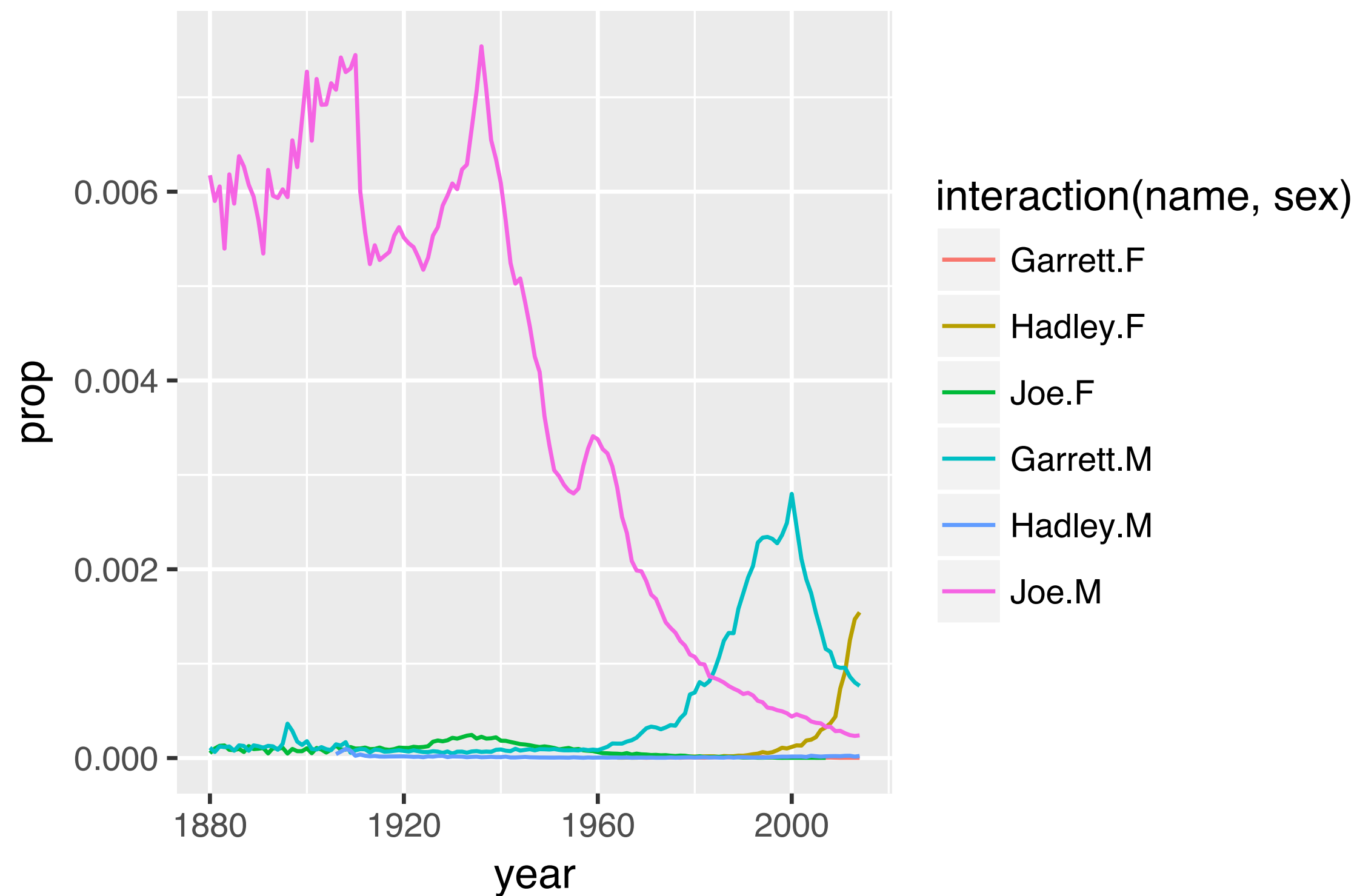


05:00

```
gnames <- c("Garrett", "Hadley", "Joe")  
group_names <- filter(babynames, name %in% gnames)  
ggplot(group_names, aes(year, prop, color = name)) +  
  geom_line()
```



```
gnames <- c("Garrett", "Hadley", "Joe")
group_names <- filter(babynames, name %in% gnames)
ggplot(group_names, aes(year, prop, color = interaction(name, sex))) +
  geom_line()
```



**interaction() to
separate combinations
of variables**

single table verbs

filter() - extract **cases**

arrange() - reorder **cases**

select() - extract **variables**

mutate() - create new **variables**

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

common syntax

Each function take 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**

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(storms, wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

desc()

Changes ordering to largest to smallest.

```
arrange(storms, desc(wind))
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01

Your Turn

Use `arrange()` (and perhaps `desc()`) to discover:

1. Which name was the most popular in a single year?
2. In what year was your name the most popular (hint: use the data set with just your name)

03:00

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

```
# 1    1880      M    John  9655 0.08154561
```

```
arrange(garrett, desc(prop))
```

```
# 1    2000      M Garrett 5840 0.002798524
```

select()



select()

Extract columns by name.

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

**data frame to
transform**

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



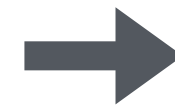
select()

Extract columns by name.

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

storms

storm	wind	pressure	date
Albert	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

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

RStudio

dplyr functions work with pipes and expect tidy data in tidy data format.

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

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

Summarise Cases

These apply summary functions to columns to create a new table. Summary functions take vectors as input and return one value (one row).

summary function

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

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

Variations

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

Group Cases

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

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

`group_by(data, ..., add = FALSE)`
Returns copy of table grouped by `g_vars = group_by(mtcars, Species)`

`ungroup(x, ...)`
Returns ungrouped copy of table `ungroup(g_mtcars)`

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, ewe = parentLFun())`
Randomly select fraction of rows.
`sample_frac(iris, 0.5, replace = TRUE)`

`sample_n(tbl, size, replace = FALSE, weight = NULL, ewe = parentLFun())`
Randomly select size rows.
`sample_n(iris, 10, replace = FALSE)`

`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 in grouped data).
`top_n(iris, 5, Sepal.Width)`

Logical and boolean operators to use with filter()

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

See `base::logic` and `?comparison` for help.

Arrange Cases

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

Add Cases

`add_row(data, ..., before = NULL, after = NULL)`
Add one or more rows to a table.
`add_row(mtcars, cylinders = 1, weight = 1)`

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)`
`one_of(...)`
`starts_with(match)`

:, e.g. mpg:cyl
-, e.g. -Species

Make New Variables

These apply vectorized functions to columns; vectorized funs take vectors as input and return vectors of the same length as input (one row).

vectorized function

`mutate(data, ...)`
Compute new column(s).
`mutate(mtcars, gpm = 1/avg)`

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

`mutate_all(tbl, funs, ...)`
Apply funs to every column. Use with `funs()`.
`mutate_all(mtcars, funs(log))`

`mutate_at(tbl, cols, funs, ...)`
Apply funs to specific columns. Use with `funs()` and the helper functions for `select()`.
`mutate_at(mtcars, c(Species, funs(log))`

`mutate_if(tbl, predicate, funs, ...)`
Apply funs to all columns of one type. Use with `funs()`.
`mutate_if(mtcars, is.numeric, funs(log))`

`add_column(data, ..., before = NULL, after = NULL)`
Add new column(s).
`add_column(mtcars, new = 1:32)`

`rename(data, ...)`
Rename column(s).
`rename(iris, Length = Sepal.Length)`

RStudio is a trademark of RStudio Inc. • © 2015 RStudio • 1518 Main St • Boston, MA 02111 • 617-552-3922 • rstudio.com

Learn more with `library(ggplot2); package = "dplyr"; ?library(package)` • dplyr (5.0+ stable 12.0 • Updated 11/15)

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)
one_of(...)
starts_with(match)

:, e.g. mpg:cyl
-, e.g. -Species



Your Turn

Write down as many ways as you can think of to select `storms$storm` with `select()`. Use each helper no more than once.

01:00

```
select(storms, storm)
select(storms, -c(wind, pressure, date))
select(storms, starts_with("s"))
select(storms, ends_with("m"))
select(storms, contains("st"))
select(storms, matches("storm"))
select(storms, one_of("storm"))
```



mutate()



mutate()

Create new columns.

```
mutate(.data, ...)
```

**data frame to
transform**

named argument
(that consists of the name of
column to create set equal to
an expression that creates it)

mutate()

Create new columns.

```
mutate(storm, ratio = pressure / wind)
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21	22.44



mutate()

Create new columns.

```
mutate(ratio = pressure / wind, inverse = ratio^-1)
```

storms

storm	wind	pressure	date		storm	wind	pressure	date	ratio	inverse
Alberto	110	1007	2000-08-12	→	Alberto	110	1007	2000-08-12	9.15	0.11
Alex	45	1009	1998-07-30		Alex	45	1009	1998-07-30	22.42	0.04
Allison	65	1005	1995-06-04		Allison	65	1005	1995-06-04	15.46	0.06
Ana	40	1013	1997-07-01		Ana	40	1013	1997-07-01	25.32	0.04
Arlene	50	1010	1999-06-13		Arlene	50	1010	1999-06-13	20.20	0.05
Arthur	45	1010	1996-06-21		Arthur	45	1010	1996-06-21	22.44	0.0



transmute()

Create new columns. Drop the old.

```
transmute(ratio = pressure / wind, inverse = ratio^-1)
```

storms

storm	wind	pressure	date		ratio	inverse
Alberto	110	1007	2000-08-12	→	9.15	0.11
Alex	45	1009	1998-07-30		22.42	0.04
Allison	65	1005	1995-06-04		15.46	0.06
Ana	40	1013	1997-07-01		25.32	0.04
Arlene	50	1010	1999-06-13		20.20	0.05
Arthur	45	1010	1996-06-21		22.44	0.04

Vectorized functions

Take a vector as input.

Return a vector of the same length as output.

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

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

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::distinct() - # of unique
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() - quantile
min() - minimum value
max() - maximum value

Spread

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

Row names

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

rownames_to_column() - Move row names into column
column_to_rownames() - Move col in row names to column_to_rownames() var = ""

Also has rownames(), remove_rownames()

Combine Tables

Combine Variables

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

bind_cols()

Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

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

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

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

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

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

Combine Cases

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

bind_rows(..., id = NULL)

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

intersect(x, y, ...)

Returns the intersection of columns.

setdiff(x, y, ...)

Returns the set difference of columns.

union(x, y, ...)

Returns the union of columns. (Duplicates removed) union_all() retains duplicates.

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

Extract Rows

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

semi_join(x, y, by = NULL, ...)

Returns rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)

Returns rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.



summarise()



summarise()

Compute table of summaries.

```
summarise(.data, ...)
```

**data frame to
transform**

named argument
(that consists of the name of
column to create set equal to
an expression that creates it)



summarise()

Compute table of summaries.

```
summarise(storms, avg_wind = mean(wind), n = n())
```

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



avg_wind	n
59.17	6

Returns the number of cases
(rows) Very useful!

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 `n`
- `dplyr::lead()` - Offset elements by `n`

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, no gaps
- `dplyr::min_rank()` - rank with ties, min
- `dplyr::ntile()` - divide into bins
- `dplyr::percent_rank()` - min - rank scaled to [0,1]
- `dplyr::row_number()` - rank with ties = first

Math

- `+`, `*`, `/`, `^`, `%/%`, `%/%` - arithmetic ops
- `log()`, `log2()`, `log10()` - logs
- `<`, `>`, `==`, `!=`, `is.na()` - 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()` - elementwise 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

Row names

- If data does not use rownames, which store a variable outside of the columns. To work with the rownames, first, move them into a column.
- `rownames_to_column()` - Move row names into column
- `column_to_rownames()` - Move col in row names
- `remove_rownames()` - Remove rownames

Combine Tables

Combine Variables

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

`bind_cols()`

Returns tables placed side by side as a single table. BE SURE THAT ROWS ALIGN.

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

- `left_join(x, y, by = NULL, copy = FALSE, suffixes = c("x", "y"), ...)` - Join matching rows from x to y.
- `right_join(x, y, by = NULL, copy = FALSE, suffixes = c("x", "y"), ...)` - Join matching rows from y to x.
- `inner_join(x, y, by = NULL, copy = TRUE, suffixes = c("x", "y"), ...)` - Join data. Return only rows with matches.
- `full_join(x, y, by = NULL, copy = TRUE, suffixes = c("x", "y"), ...)` - Join data. Return all values, all rows.

Combine Cases

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

`bind_rows(..., id = NULL)`

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

`intersect(x, y, ...)`

Returns the intersection of columns x and y.

`setdiff(x, y, ...)`

Returns the elements in x that are not in y.

`union(x, y, ...)`

Returns the union of columns x and y. Duplicates are removed. `union_all()` retains duplicates.

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

Extract Rows

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

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

Returns rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

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

Returns rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.



n() and n_distinct()

Two helper functions for summarise.

```
summarise(storms,  
  n = n(), # Number of cases / rows  
  n_wind = n_distinct(wind) # number of unique values  
)
```

storm	wind	pressure	date	→		n	n_wind
Alberto	110	1007	2000-08-12			6	5
Alex	45	1009	1998-07-30				
Allison	65	1005	1995-06-04				
Ana	40	1013	1997-07-01				
Arlene	50	1010	1999-06-13				
Arthur	45	1010	1996-06-21				

Your Turn

1. Add a new column to the data that changes the prop to a percentage
2. Determine how many unique names appear in the data set.
3. Create a summary that displays the min, mean, and max prop for your name.
4. Use three dplyr verbs plus n() to determine how many times a single name was given to more than 1% of the boys or girls in a year.

07:00


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

```
summarize(babynames, n = n_distinct(name))
```

```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
summarise(garrett, min = min(prop), mean = mean(prop),  
  max = max(prop))
```

```
babynames2 <- mutate(babynames, percent = prop * 100)  
babynames3 <- filter(babynames2, percent > 1)  
summarise(babynames3, nn = n())
```

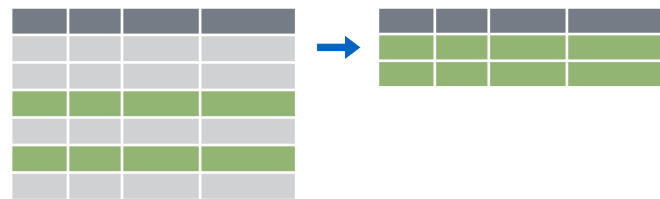
```
mutate(babynames, percent = prop * 100)
```

```
summarize(babynames, n = n_distinct(name))
```

```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
summarise(garrett, min = min(prop), mean = mean(prop),  
  max = max(prop))
```

```
babynames2 <- mutate(babynames, percent = prop * 100)  
babynames3 <- filter(babynames2, percent > 1)  
count(babynames3)
```

Recap: Single table verbs



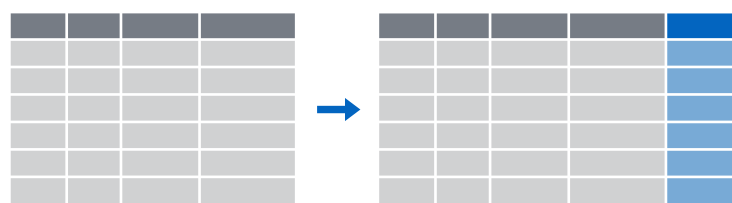
Extract cases with **filter()**



Extract variables with **select()**



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



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



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



$\% \geq \%$



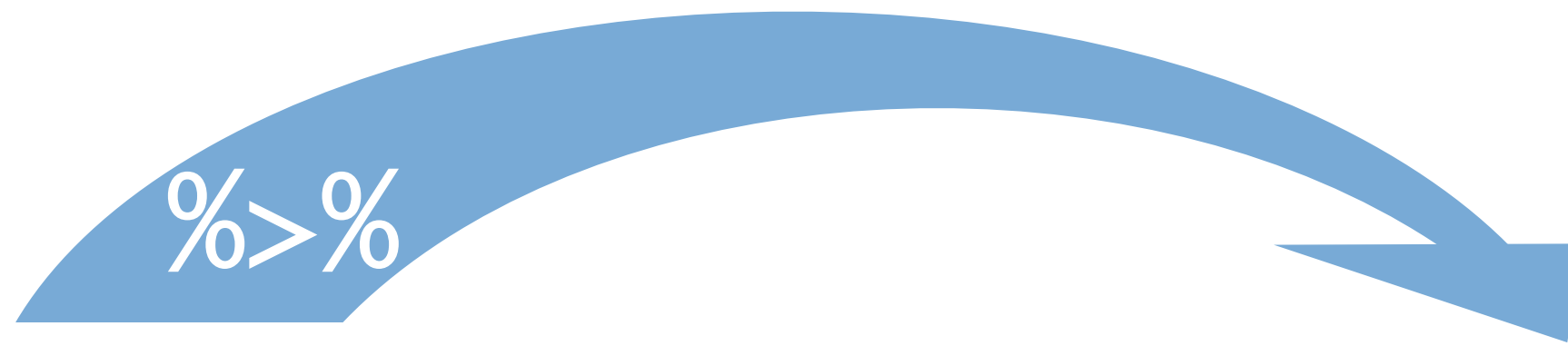
```
mutate(babynames, percent = prop * 100)
```

```
summarize(babynames, n = n_distinct(name))
```

```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
summarise(garrett, min = min(prop), mean = mean(prop),  
  max = max(prop))
```

```
babynames2 <- mutate(babynames, percent = prop * 100)  
babynames3 <- filter(babynames2, percent > 1)  
summarise(babynames3, nn = n())
```

The pipe operator %>%



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

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

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

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




```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
summarise(garrett, min = min(prop), mean = mean(prop),  
           max = max(prop))
```

```
filter(babynames, name == "Garrett", sex == "M") %>%  
  summarise(min = min(prop), mean = mean(prop),  
            max = max(prop))
```

```
garrett <- filter(babynames, name == "Garrett", sex == "M")  
summarise(garrett, min = min(prop), mean = mean(prop),  
  max = max(prop))
```

```
babynames %>%  
  filter(name == "Garrett", sex == "M") %>%  
  summarise(min = min(prop), mean = mean(prop),  
    max = max(prop))
```

Shortcut to type %>%

Cmd + **Shift** + **M** (Mac)

Ctrl + **Shift** + **M** (Windows)

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

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

VS.

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

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

```
bop_on(  
  scoop_up(  
    hop_through(foo_foo, forest),  
    field_mouse  
  ),  
  head  
)
```

Your Turn

Rewrite the code below to use the pipe operator. Then run it to ensure that it works.

```
babynames
```

```
babynames2 <- mutate(babynames, percent = prop * 100)
```

```
babynames3 <- filter(babynames2, percent > 1)
```

```
summarise(babynames3, nn = n())
```

04:00


```
babynames %>%  
  mutate(percent = prop * 100) %>%  
  filter(percent >= 1) %>%  
  summarize(nn = n())
```



Grouping cases



Your Turn

Are there more boys in the data set or girls?

Outline a strategy with your group and then find out.

05:00

```
girls <- babynames %>%  
  filter(sex == "F") %>%  
  summarise(total = sum(n))  
# 167070477
```

```
boys <- babynames %>%  
  filter(sex == "M") %>%  
  summarise(total = sum(n))  
# 170064949
```



Toy data

```
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
)
```

pollution

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

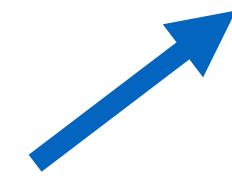


88.5	177	2
------	-----	---

`group_by() + summarise()`

group_by()

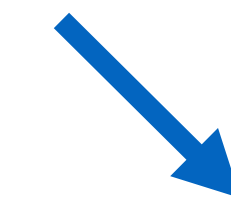
city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



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()

Groups cases by common values.

```
group_by(.data, ...)
```

**data frame to
transform**

column(s) to group by
(will group by combinations of
values if more than one column is
specified)

group_by()

Groups cases by common values.

```
.data %>% group_by(...)
```

**data frame to
transform**

column(s) to group by
(will group by combinations of
values if more than one column is
specified)

group_by()

Groups cases by common values.

```
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))
```

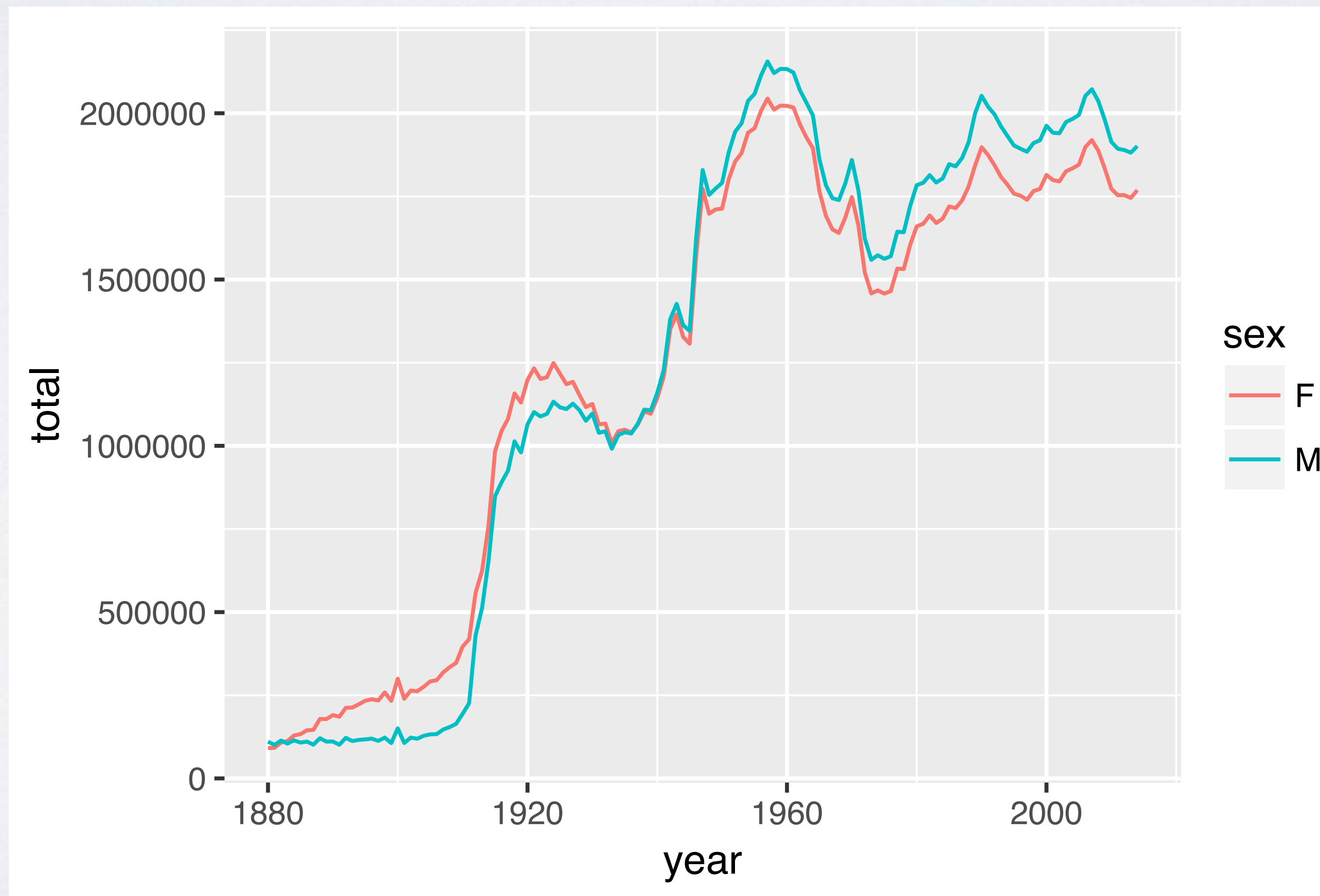
sex	total
F	167070477
M	170064949



Your Turn

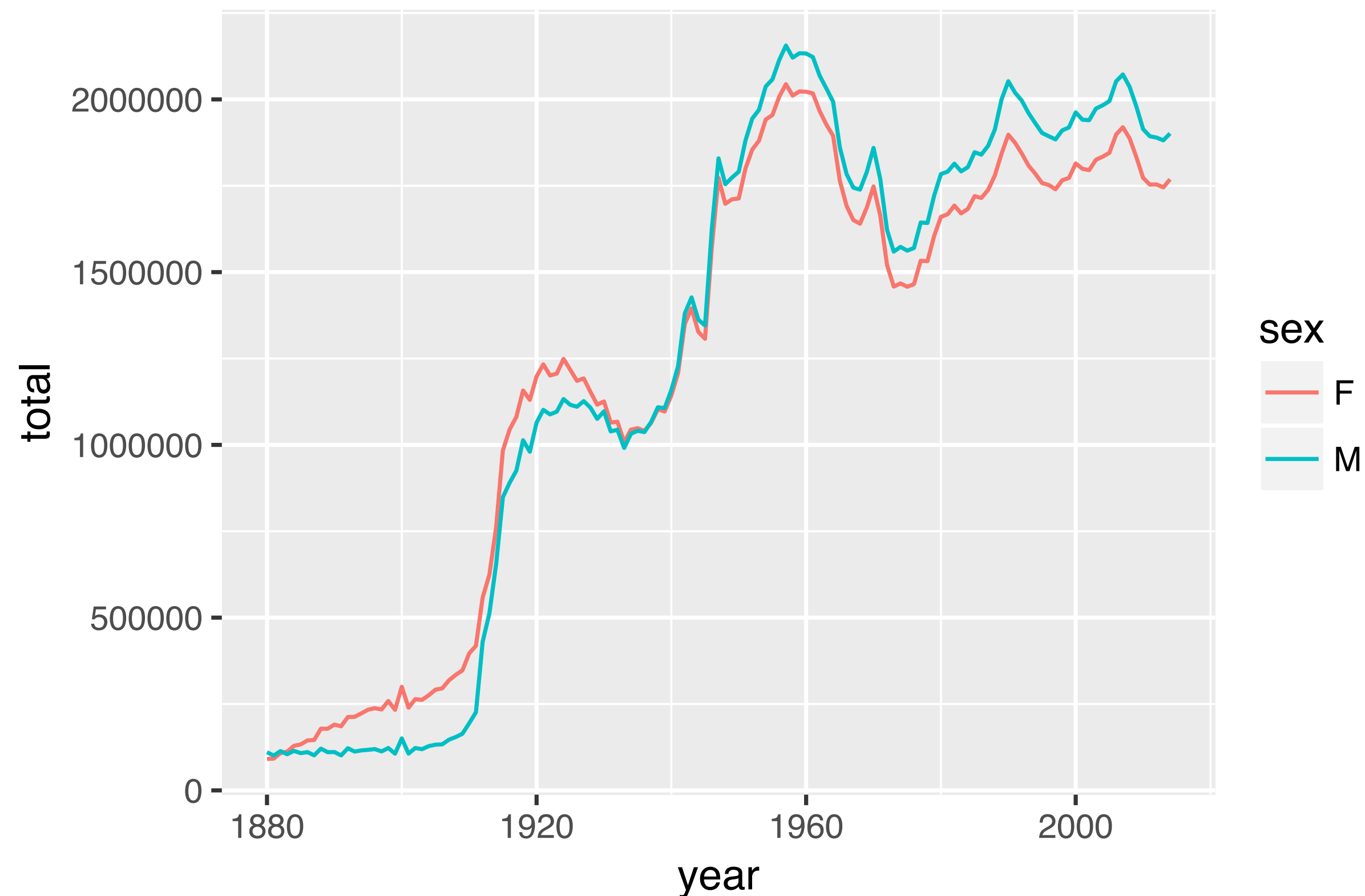
Does the ratio of boys and girls change over time?

Use `group_by()` to group by *year and sex*. Then use the result to make this plot:



05:00

```
babynames %>%  
  group_by(year, sex) %>%  
  summarise(total = sum(n)) %>%  
  ggplot(aes(x = year, y = total, color = sex)) +  
    geom_line()
```



ungrouping

summarise() removes **one** grouping variable each time you call it.

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

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

Groups: year, sex [270]

	year	sex	name	n	prop
	<dbl>	<chr>	<chr>	<int>	<dbl>
1	1880	F	Mary	7065	0.07238359
2	1880	F	Anna	2604	0.02667896



ungrouping

summarise() removes **one** grouping variable each time you call it.

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

Source: local data frame [270 x 3]

Groups: year [?]

	year	sex	total
	<dbl>	<chr>	<int>
1	1880	F	90993

ungrouping

summarise() removes **one** grouping variable each time you call it.

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

```
# A tibble: 135 × 2
```

```
  year total
```

```
  <dbl> <int>
```

```
1  1880     2
```

```
2  1881     2
```

ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%  
  group_by(year, sex) %>%  
  ungroup()
```

```
# A tibble: 1,825,433 × 5
```

	year	sex	name	n	prop
	<dbl>	<chr>	<chr>	<int>	<dbl>
1	1880	F	Mary	7065	0.07238359
2	1880	F	Anna	2604	0.02667896
3	1880	F	Emma	2003	0.02052149



Your Turn

For both exercises, use the `%>%` operator whenever possible:

1. Plot the number of unique names by year over time. Does it change?
2. Plot the average number of children per name by year over time (total number of children / number of unique names). Does the plot tell a different story? Why?

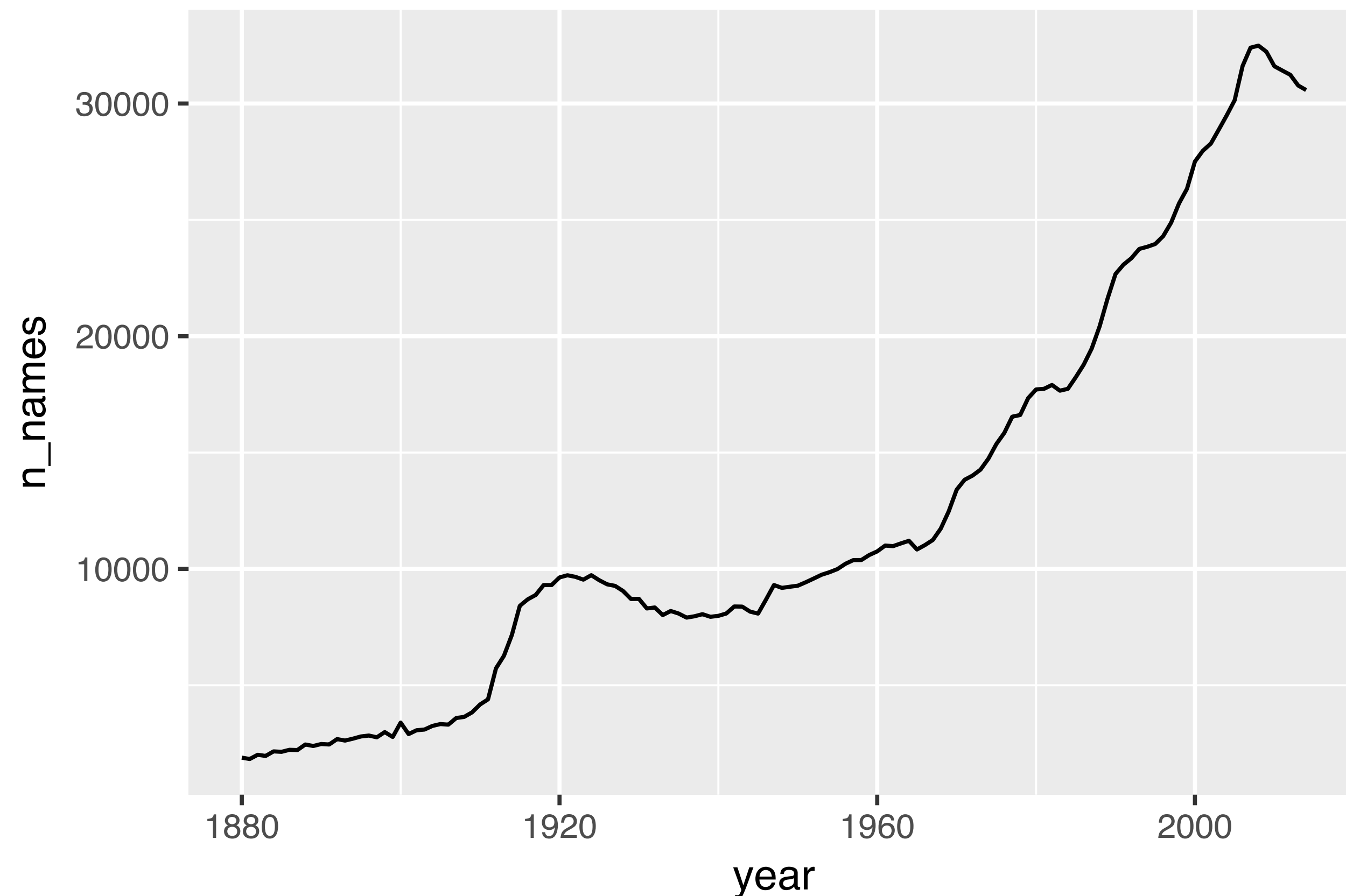
A digital timer with a black border, displaying the time 08:00 in a large, black, digital font. The digits are slightly shadowed, giving it a 3D appearance.

```
babynames %>%
```

```
  group_by(year) %>%
```

```
  summarise(n_names = n_distinct(name)) %>%
```

```
  ggplot(aes(year, n_names)) + geom_line()
```



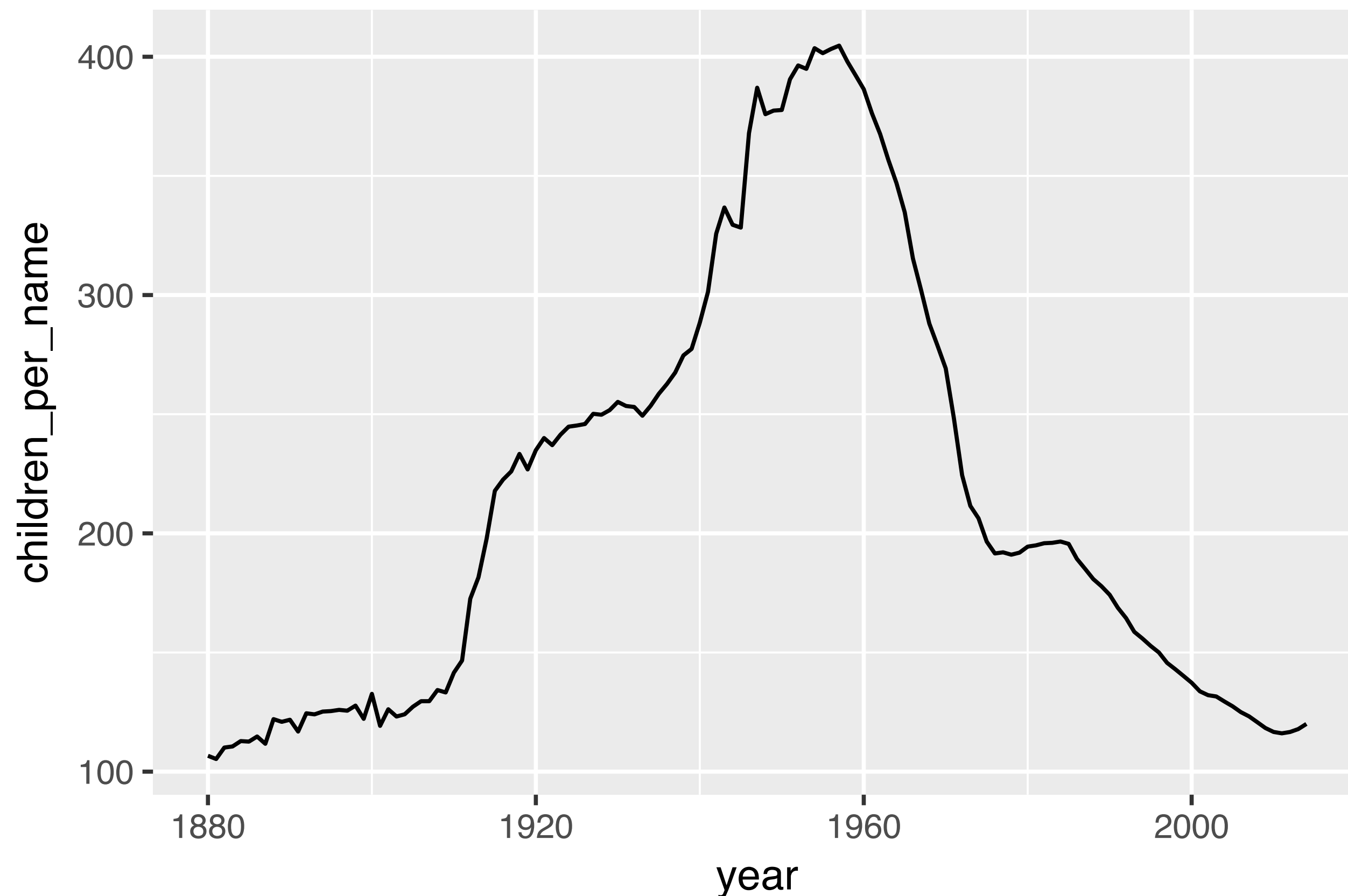

```
babynames %>%
```

```
  group_by(year) %>%
```

```
  summarise(n_names = n_distinct(name), n_children = sum(n)) %>%
```

```
  mutate(children_per_name = n_children / n_names) %>%
```

```
  ggplot(aes(year, children_per_name)) + geom_line()
```



Quiz

For a given year, what does the difference between `n()` and `n_distinct(name)` reveal?

```
babynames %>%  
  filter(year == 1930, n > 500) %>%  
  summarise(n = n(),  
            n_name = n_distinct(name)) %>%  
  mutate(diff = n - n_name)  
# A tibble: 1 × 3  
      n n_name diff  
  <int> <int> <int>  
1   549   541     8
```

Quiz

For a given year, what does the difference between `n()` and `n_distinct(name)` reveal?

```
babynames %>%  
  filter(year == 1930, n >= 500) %>%  
  summarise(n = n(),  
            n_name = n_distinct(name)) %>%  
  mutate(diff = n - n_name)  
# A tibble: 1 × 3  
      n n_name diff  
  <int> <int> <int>  
1   549   541     8
```

name
Billie
Bobbie
Jackie
Jessie
Jimmie
Johnnie
Marion
Willie

Your Turn

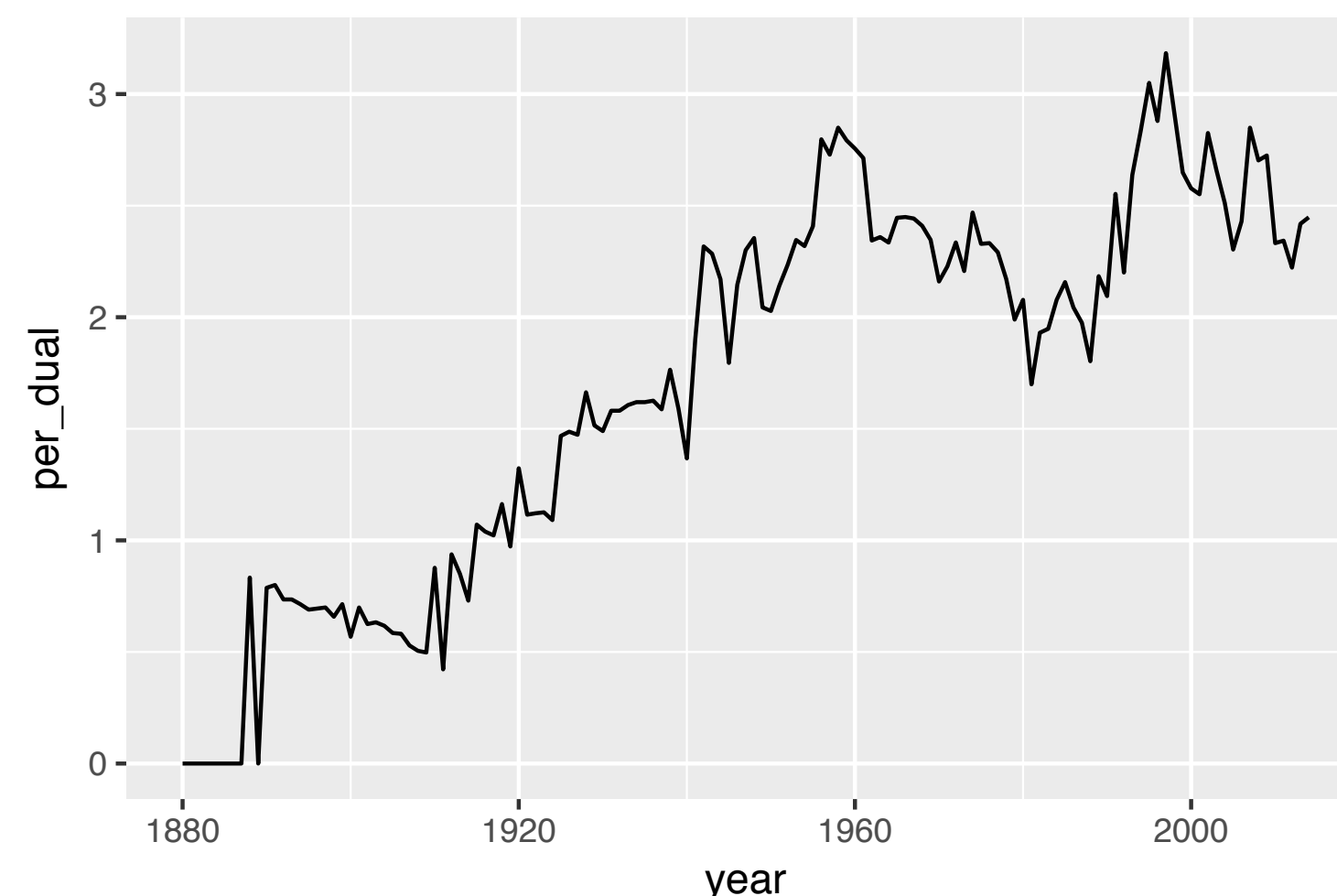
Remove all cases where $n < 500$ (to minimize data entry errors).

Then, for each year, calculate the percent of names that are used for both sexes.

Then plot how the percent of dual sex names changes over time.

A digital timer display showing the time 05:00. The digits are black with a white outline, set against a white background. The timer is enclosed in a thin black rectangular border.

```
babynames %>%  
  filter(n > 500) %>%  
  group_by(year) %>%  
  summarise(nn = n(), n_names = n_distinct(name)) %>%  
  mutate(per_dual = (nn - n_names) / n_names * 100) %>%  
  ggplot(aes(x = year, y = per_dual)) +  
    geom_line()
```



Two table verbs

(joining data sets)

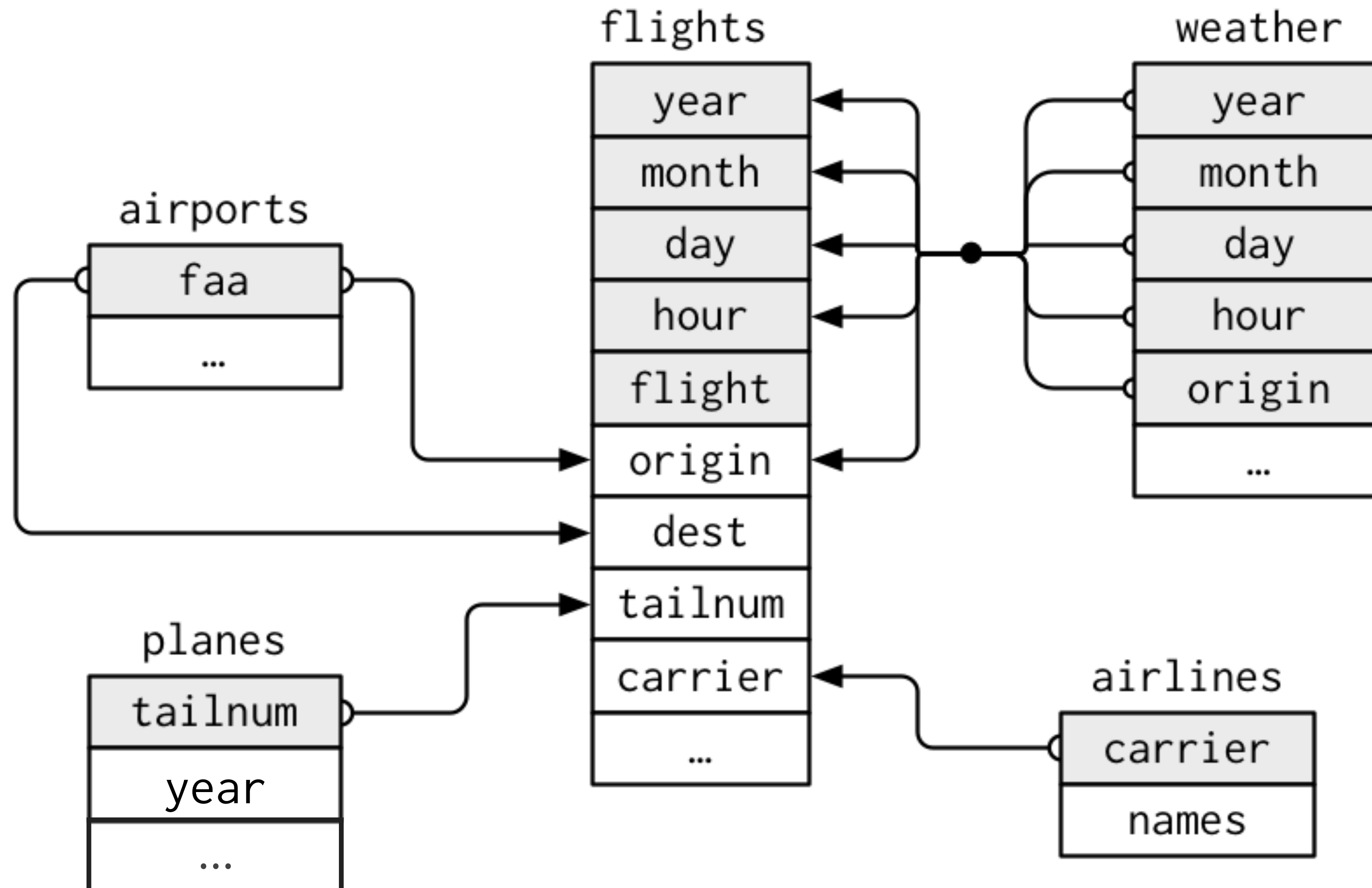
nycflights13



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

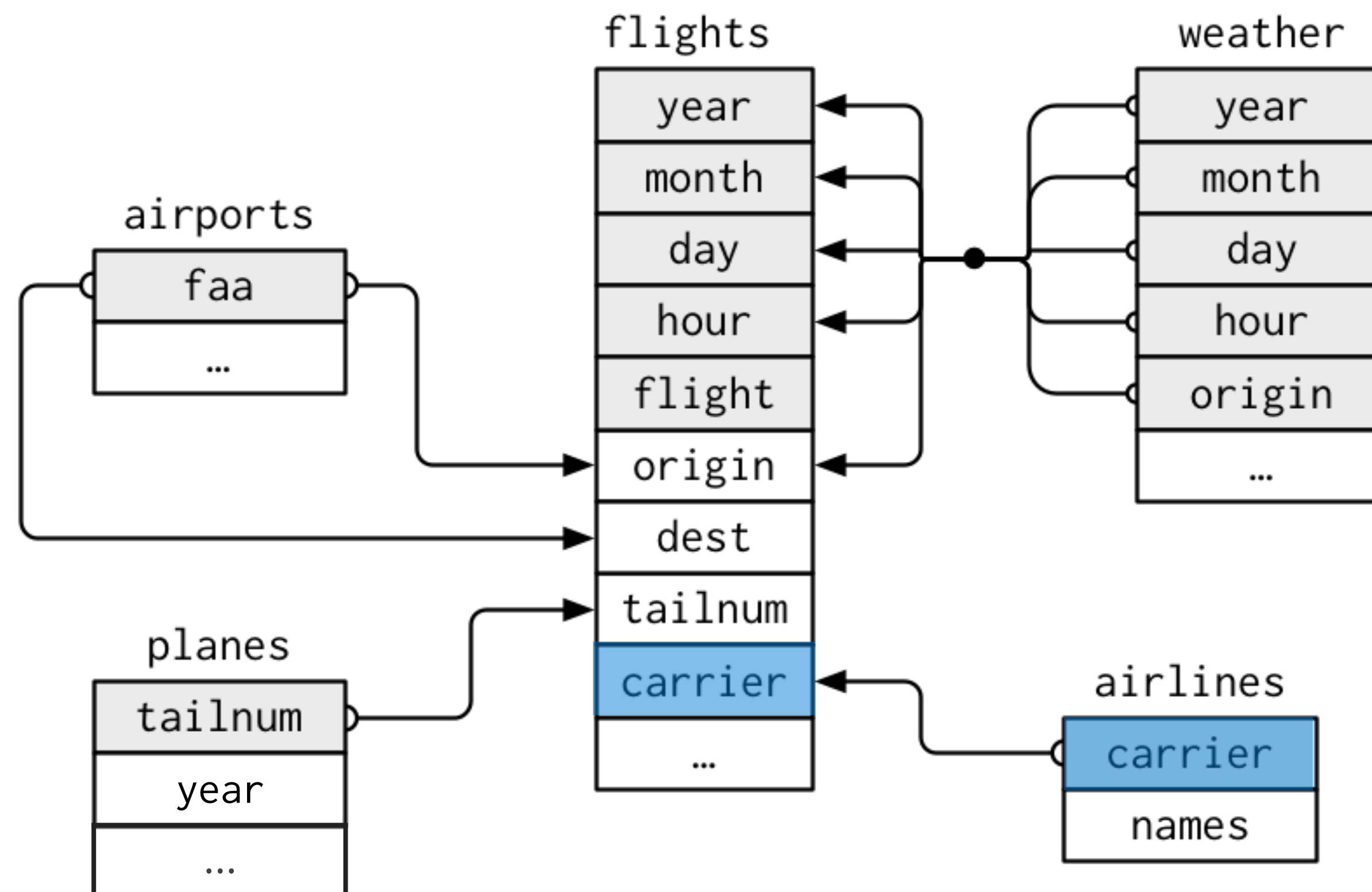
```
# install.packages("nycflights13")  
library(nycflights13)
```


nycflights13



nycflights13

What airline had the longest delays?



Airline names

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

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.

mutating joins



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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

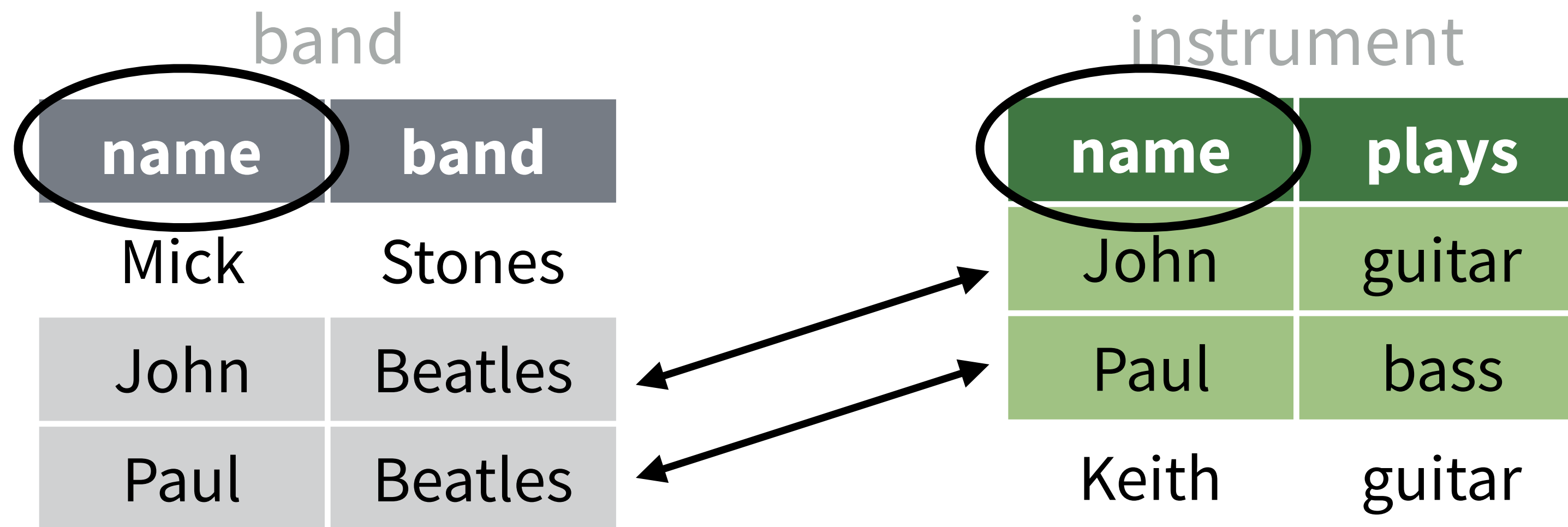
```
instrument <- tribble(
  ~name,    ~plays,
  "John",   "guitar",
  "Paul",   "bass",
  "Keith",  "guitar"
)
```

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar



Toy data



left

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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

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

right

```
right_join(band, 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

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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

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

inner

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

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

Airline names

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

	carrier
1	UA
2	UA
3	AA
4	B6
5	DL
6	UA
7	B6

```
View(airlines)
```

	carrier	name
1	9E	Endeavor Air Inc.
2	AA	American Airlines Inc.
3	AS	Alaska Airlines Inc.
4	B6	JetBlue Airways
5	DL	Delta Air Lines Inc.
6	EV	ExpressJet Airlines Inc.
7	F9	Frontier Airlines Inc.

Your Turn

Which airlines had the largest arrival delays? Work in groups to complete the code below.

```
flights %>%
```

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

```
  _____ %>%
```

```
  group_by(_____) %>%
```

```
  _____ %>%
```

```
  arrange(_____)
```

1. Join airlines to flights

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

06:00

```
flights %>%  
  drop_na(arr_delay) %>%  
  left_join(airlines, by = "carrier") %>%  
  group_by(name) %>%  
  summarise(delay = mean(arr_delay)) %>%  
  arrange(delay)
```

```
## # A tibble: 16 × 2
```

```
##           name      delay  
##          <chr>    <dbl>  
## 1 Alaska Airlines Inc. -9.9308886  
## 2 Hawaiian Airlines Inc. -6.9152047  
## 3 American Airlines Inc.  0.3642909  
## 4 Delta Air Lines Inc.   1.6443409  
## 5 Virgin America      1.7644644
```



Toy data

band

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.

```
left_join(band, 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

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

band		instrument2						
name	band		artist	plays		name	band	plays
Mick	Stones	+	John	guitar	=	Mick	Stones	<NA>
John	Beatles		Paul	bass		John	Beatles	guitar
Paul	Beatles		Keith	guitar		Paul	Beatles	bass

Airport names

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

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

```
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
0GG	William Grant Airport

Your Turn

Use flights and airports to compute the distance and average arr_delay by destination airport (names only, not codes). Order by average delay, worst to best.

Hint: use first() to get distance.

04:00

```

flights %>%
  drop_na(arr_delay) %>%
  left_join(airports, by = c("dest" = "faa")) %>%
  group_by(name) %>%
  summarise(distance = first(distance),
            delay = mean(arr_delay)) %>%
  arrange(desc(delay))
## # A tibble: 101 × 3
##           name distance delay
##           <chr>    <dbl> <dbl>
## 1 Columbia Metropolitan    602 41.76415
## 2 Tulsa Intl             1215 33.65986
## 3 Will Rogers World       1325 30.61905

```

filtering joins



filtering joins

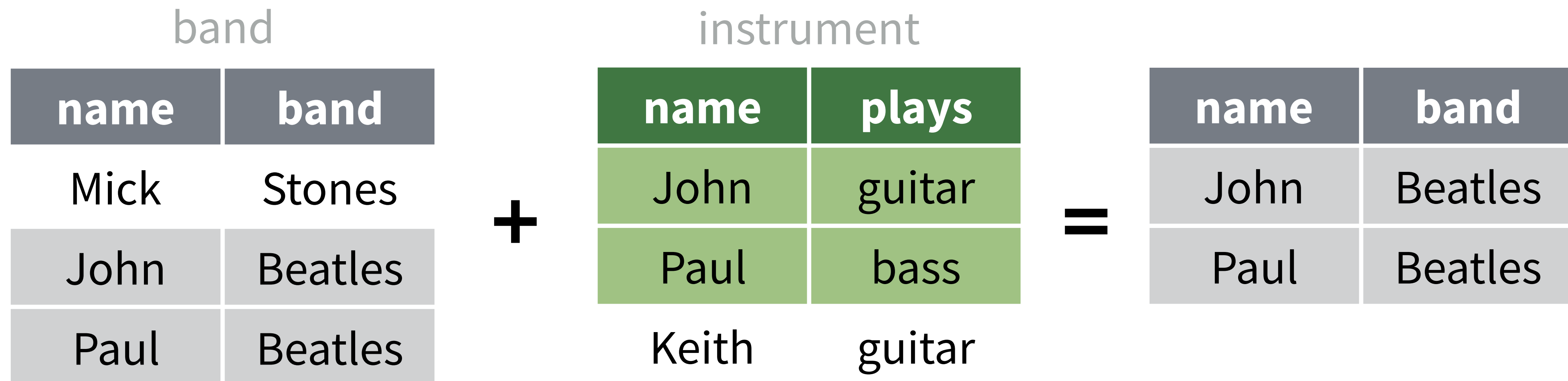
Mutating joins use information from one data set **to add variables** to another data set (like **mutate()**)

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



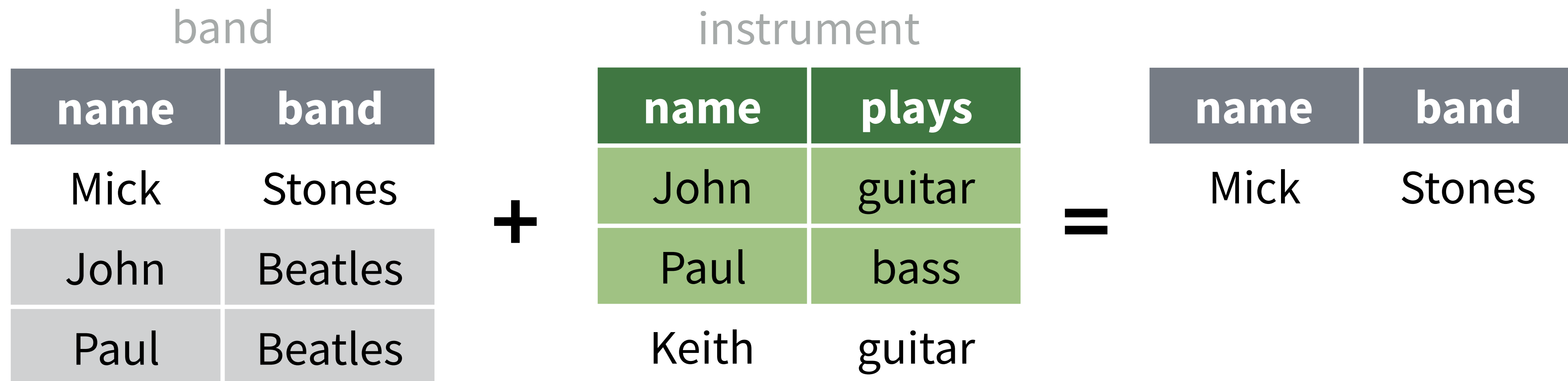
semi

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



anti

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



Airport names

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

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

```
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
0GG	William Grant Airport

Your Turn

1. How many airports in airports are serviced by flights originating in New York (i.e. flights in our data set)?
2. How many flights in flights flew to an airport not listed in airports?

06:00

```
airports %>%  
  semi_join(flights, by = c("faa" = "dest"))  
## # A tibble: 101 × 7
```

```
flights %>%  
  anti_join(airports, by = c("dest" = "faa"))  
## # A tibble: 7,602 × 19
```

```
flights %>%  
  anti_join(airports, by = c("dest" = "faa")) %>%  
  distinct(dest)  
# A tibble: 4 × 1  
  dest  
  <chr>  
1  PSE  
2  STT  
3  SJU  
4  BQN
```



distinct()

Removes rows with duplicate values (in a column).

```
distinct(df, name)
```

df

name	band
Mick	Stones
John	Beatles
John	Zeppelin

→

name
Mick
John

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

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

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



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 `n`.
- `dplyr::lead()` - Offset elements by `n`.

Cumulative Aggregates

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

Rankings

- `dplyr::cume_dist()` - Population cdf values.
- `dplyr::dense_rank()` - rank with ties = min, no gaps.
- `dplyr::min_rank()` - rank with ties = min.
- `dplyr::ntile()` - bin 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::if_else()` - replace specific values with NA.
- `pmax()` - element-wise max.
- `pmin()` - element-wise min.
- `dplyr::recode()` - Vectorized subset().
- `dplyr::recode_factor()` - Vectorized subset() 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::distinct()` - # of unique.
- `sum(is.na())` - # of non-NA's.

Location

- `mean()` - mean, also `mean(is.na())`.
- `median()` - median.

Logicals

- `mean()` - % population 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()` - n.d. quantile like min().
- `min()` - min. minimum value.
- `max()` - max. maximum value.

Spread

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

Row names

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

rownames_to_column()

Move row names into column `q`. `rownames_to_column(data, var = "q")`

column_to_rownames()

Move `col` in row names `column_to_rownames(data, var = "col")`

Also `has_rownames()`, `remove_rownames()`

Combine Tables

Combine Variables

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

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

A	B	C	A	B	C
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
a	t	1	3
b	u	2	2
d	w	NA	1

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

A	B	C	D
a	t	1	3
b	u	2	2

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

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

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

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

Use **by = c("col1", "col2")** to specify the column(s) to match on.

left_join(x, y, by = "A")

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

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

left_join(x, y, by = c("C" = "D"))

A1	B1	C	A2	B2
a	t	1	d	w
b	u	2	b	u
c	v	3	a	t

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

left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

Combine Cases

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

z		
A	B	C
c	v	3
d	w	4

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

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

A	B	C
c	v	3

intersect(x, y, ...)

Rows that appear in both x and z.

A	B	C
a	t	1
b	u	2

setdiff(x, y, ...)

Rows that appear in both x but not z.

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

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).

Extract Rows

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

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

A	B	C
a	t	1
b	u	2

semi_join(x, y, by = NULL, ...)

Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

A	B	C
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



Tidy tools

Functions are easiest to use when they are:

1. **Simple** - They do one thing, and they do it well
2. **Composable** - They can be combined with other functions for multi-step operations
3. **Smart** - They can use R objects as input.

Tidy functions do these things in a specific way.

1. Simple - They do one thing, and they do it well

filter() - extract **cases**

arrange() - reorder **cases**

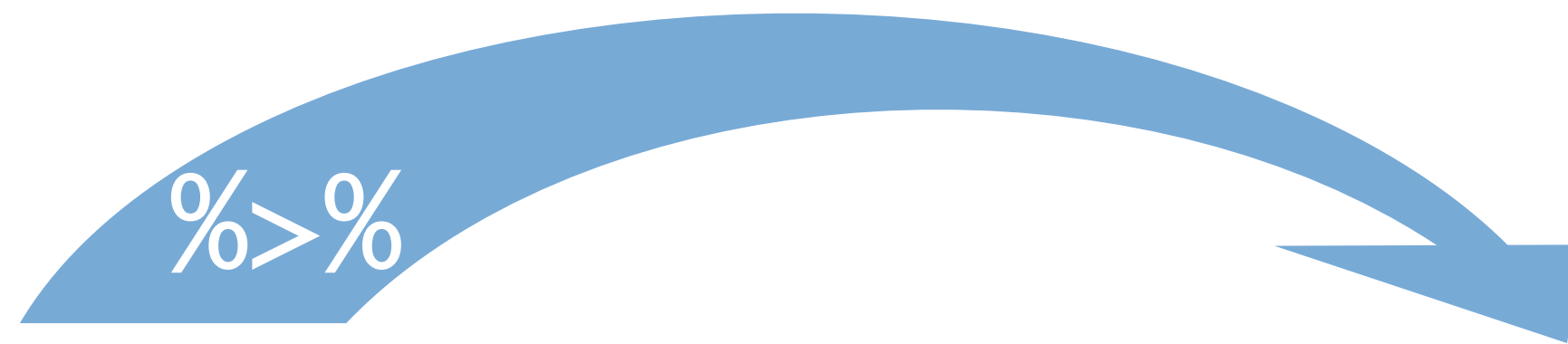
group_by() - group **cases**

select() - extract **variables**

mutate() - create new **variables**

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

2. **Composable** - They can be combined with other functions for multi-step operations



`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.

3. **Smart** - They can use R objects as input.

TODO: Dplyr functions use non-standard evaluation, which means that you cannot simply pass an R object.


```
babynames %>% filter(n > 500)
```

These objects are looked
up in the scope of x

These objects are looked
up in babynames

```
x <- n > 500
```

```
babynames %>% filter(x) # ERROR!
```

_functions

Every major dplyr function comes with a programming friendly version that accepts formulas. The version has an _ at the end of its name.

~ saves the
expression as a
formula

filter_()

```
x <- ~n > 500
```

```
babynames %>% filter_(x) # Works :)
```

Data Transformation with

