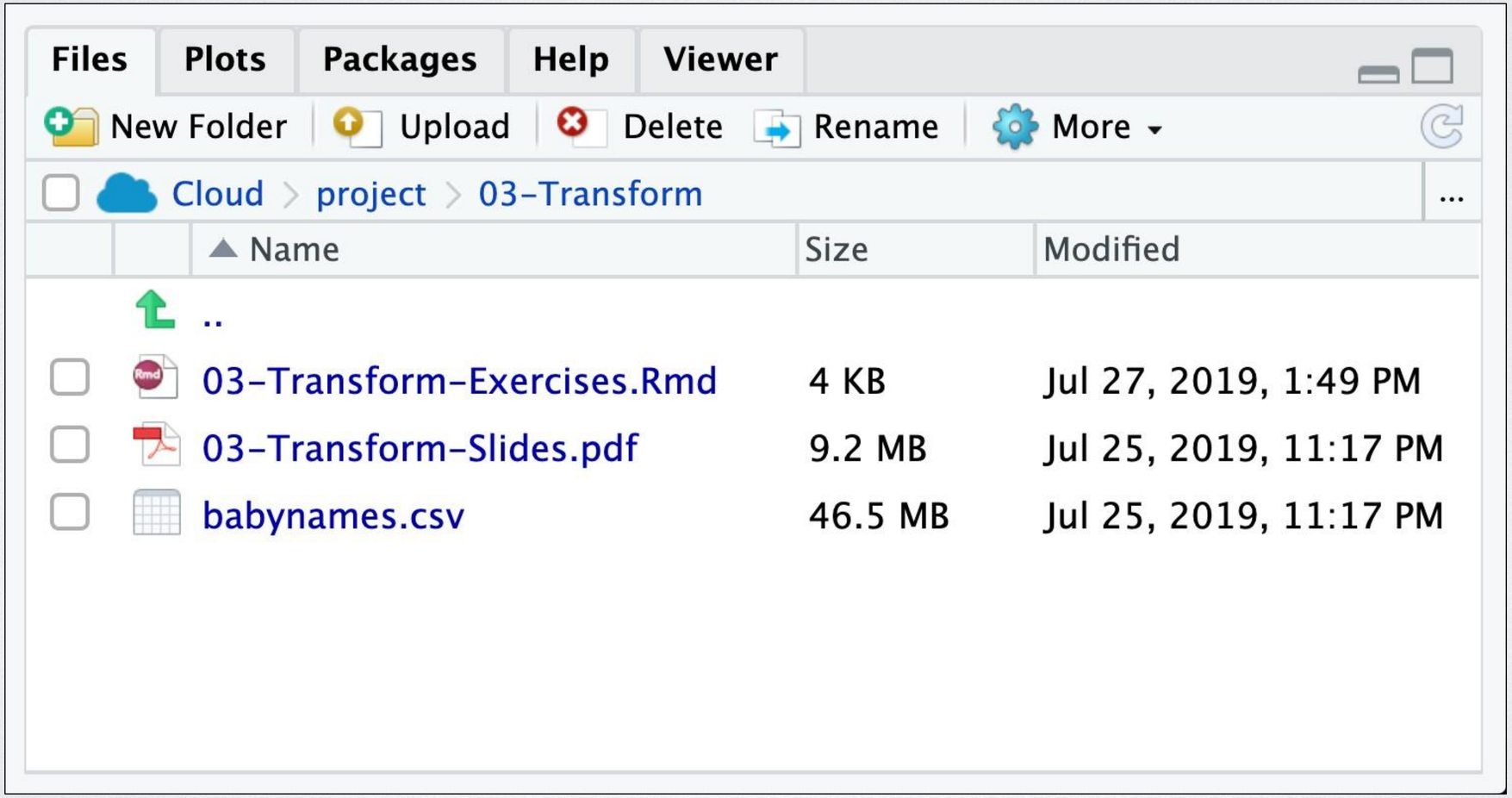
Transform Data with

Slides CC BY-SARStudio

babynames.csv

Names and sex of babies born in the US from 1880 to 2017. 1.9M rows.



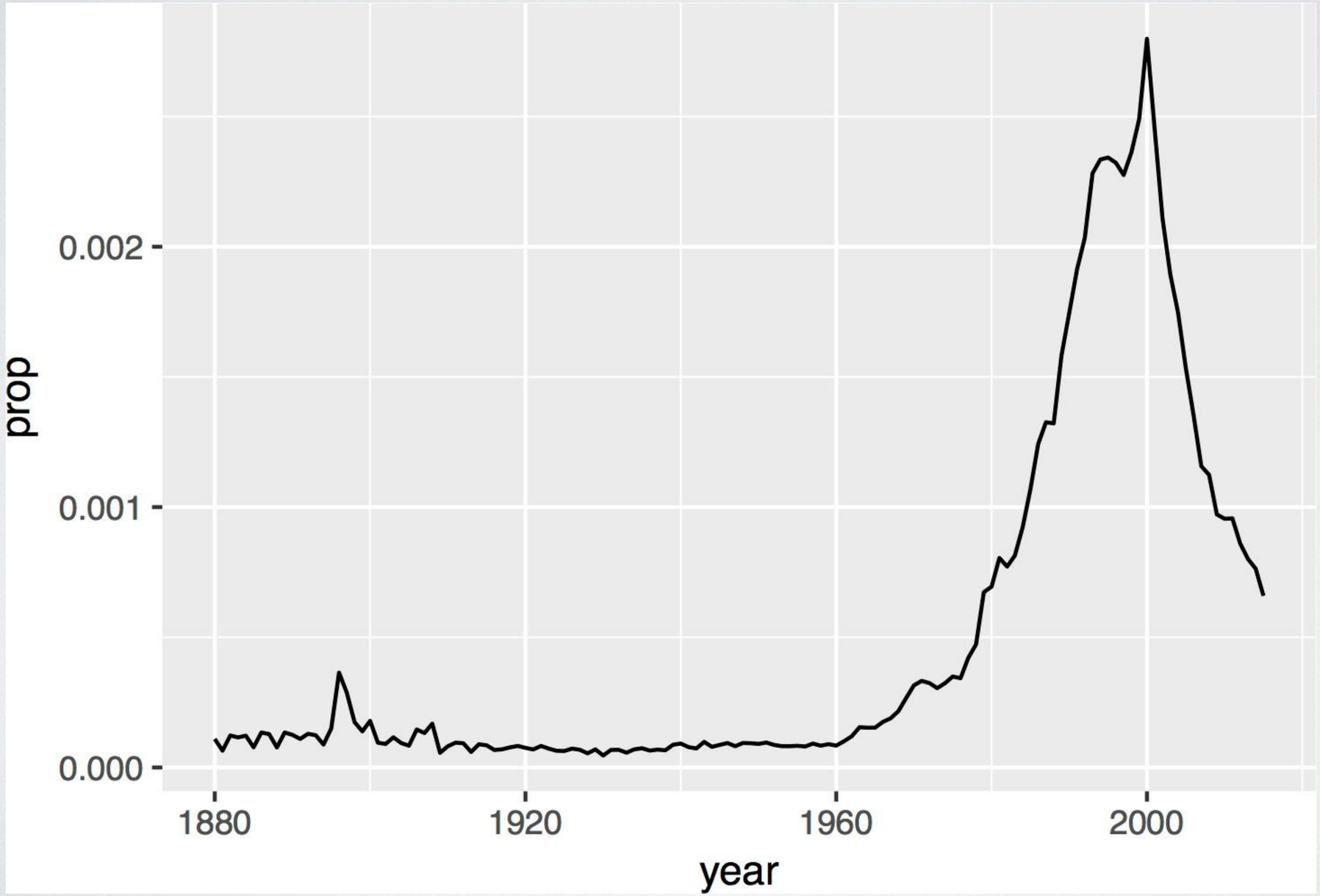
```
babynames
```

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

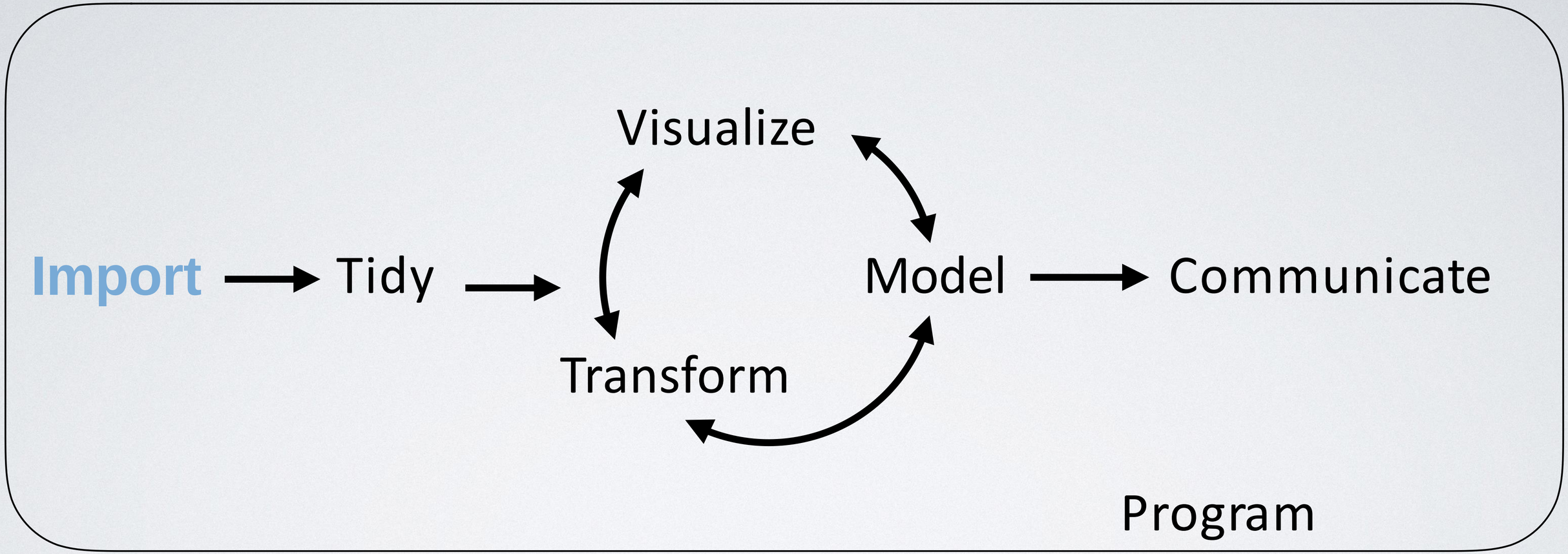
1–10 of 1,924,665 rows Previous 1 2 3 4 5 6 ... 100 Next

dplyr

Proportion of boys with the name Garrett



(Applied) Data Science



Import data

babynames.csv

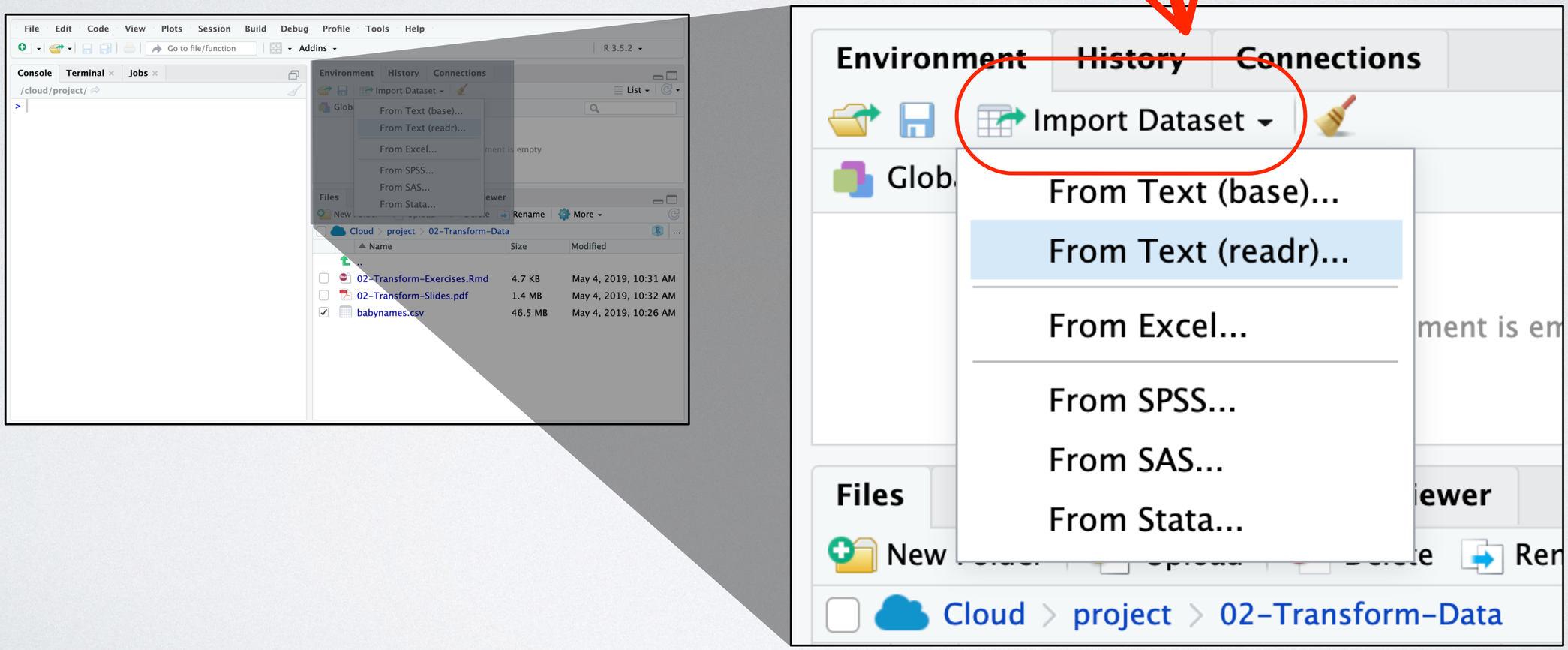
```
year,sex,name,n,prop
1880,F,Mary,7065,0.07238359
1880,F,Anna,2604,0.02667896
1880,F,Emma,2003,0.02052149
1880,F,Elizabeth,1939,0.01986579
1880,F,Minnie,1746,0.01788843
1880,F,Margaret,1578,0.0161672
1880,F,Ida,1472,0.01508119
1880,F,Alice,1414,0.01448696
```

babynames.csv

```
year,sex,name,n,prop
1880,F,Mary,7065,0.07238359
1880,F,Anna,2604,0.02667896
1880,F,Emma,2003,0.02052149
1880,F,Elizabeth,1939,0.01986579
1880,F,Minnie,1746,0.01788843
1880,F,Margaret,1578,0.0161672
1880,F,Ida,1472,0.01508119
1880,F,Alice,1414,0.01448696
```

Import

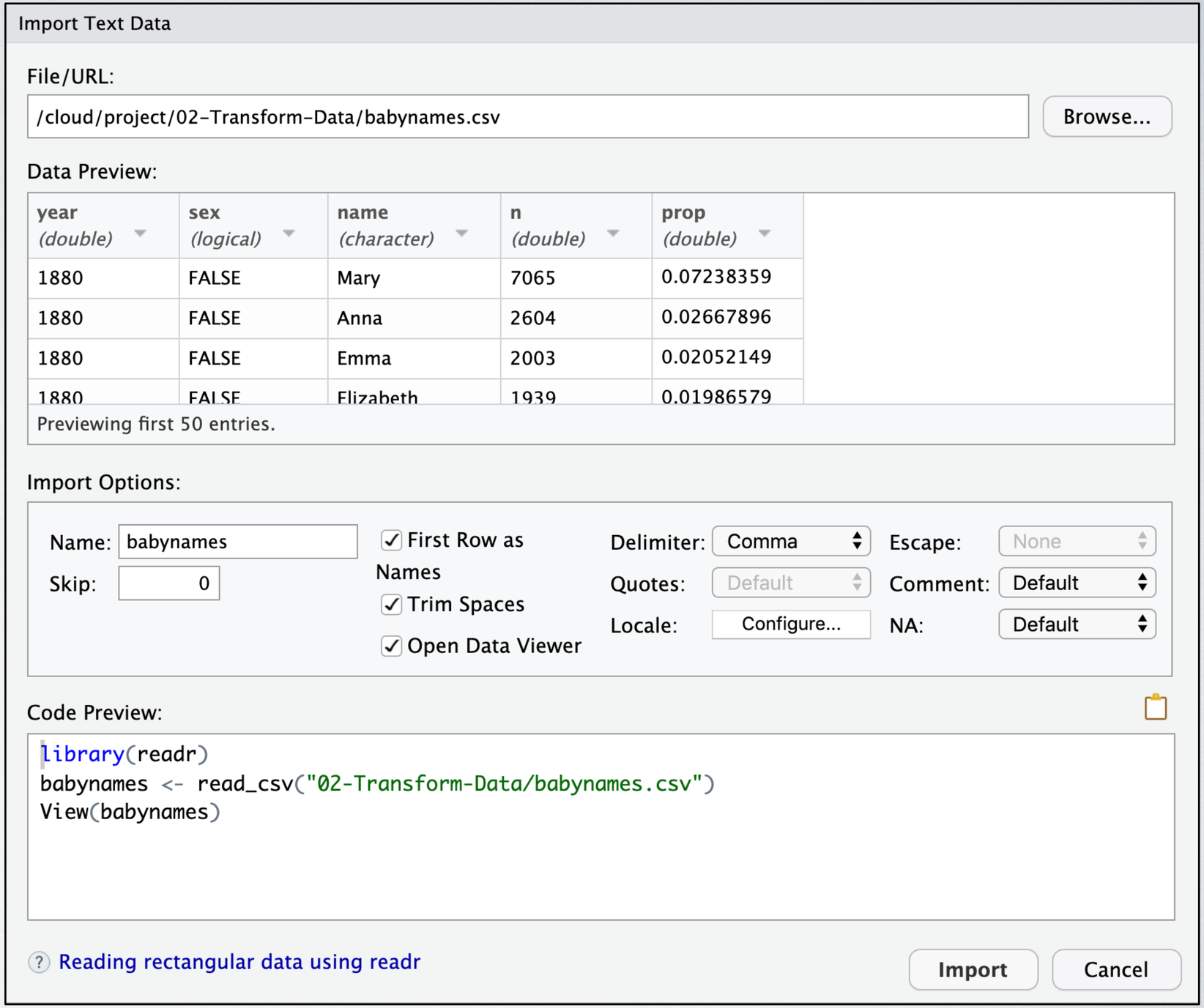
Click Import Dataset From Text (readr)...



Pop Quiz

But is this reproducible?

The Code Equivalent



One complication!

Import Text Data

File/URL:

/cloud/project/02-Transform-Data/babynames.csv

Browse...

Data Preview:

year (double)	sex (logical)	name (character)	n (double)	prop (double)
1880	FALSE	Mary	7065	0.07238359
1880	FALSE	Anna	2604	0.02667896
1880	FALSE	Emma	2003	0.02052149
1880	FALSE	Elizabeth	1939	0.01986579

Previewing first 50 entries.

Import Options:

Name: babynames

Skip: 0

☑ First Row as Names

☑ Trim Spaces

☑ Open Data Viewer

Delimiter: Comma

Quotes: Default

Locale: Configure...

Escape: None

Comment: Default

NA: Default

Code Preview:

```
library(readr)
babynames <- read_csv("02-Transform-Data/babynames.csv")
View(babynames)
```

Reading rectangular data using readr

Import

Cancel

Working directory

R associates itself with a folder (i.e. directory) on your computer. To see which one, run **getwd()** at the console.

- This folder is known as your "working directory"
- When you save files, R will save them here
- When you load files, R will look for them here

Quiz

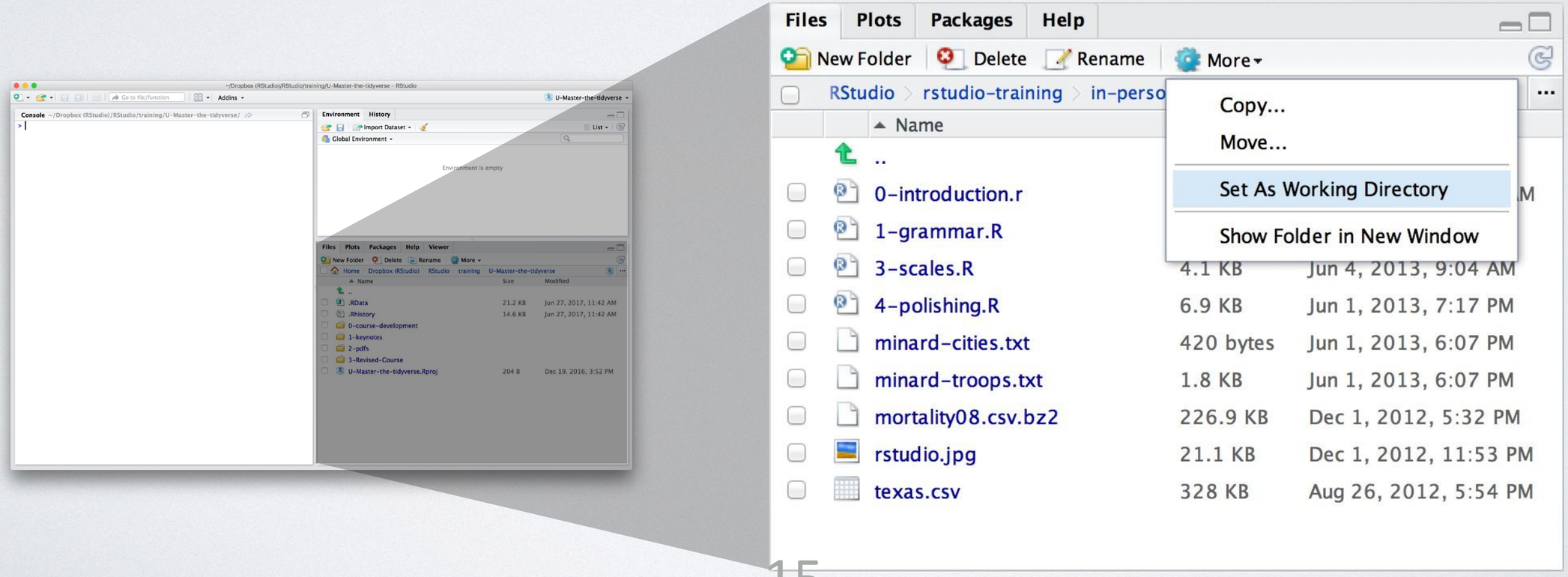
Where R look for files when you call them from an R Markdown document?

Working Directory $\neq$ .Rmd Directory

Changing the Working directory

Navigate in the files pane to a new directory. Click

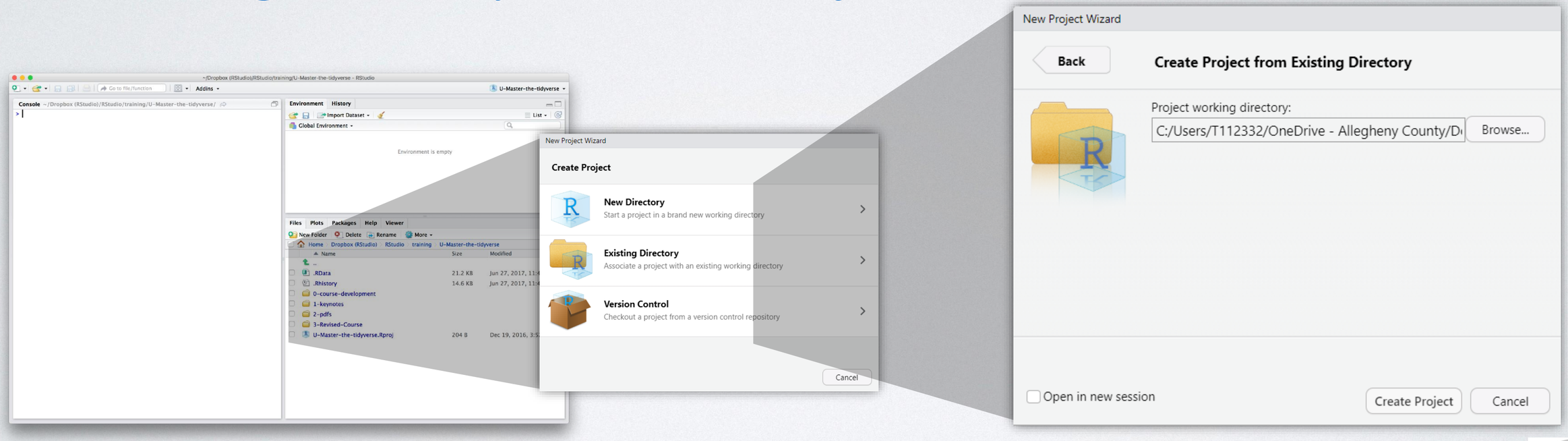
More > Set As Working Directory



Set with Projects

Create a project in the directory

File > New Project > Existing Directory > Navigate to Working Directory > Create Project



Your Turn 1

Move your working directory to the **03-Transform** folder.

Import the **babynames.csv** dataset. Give it the name babynames.

Copy the import code into the code chunk in **03-Transform-Exercises** (so the document can reload it later).

babynames

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

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

write_csv()

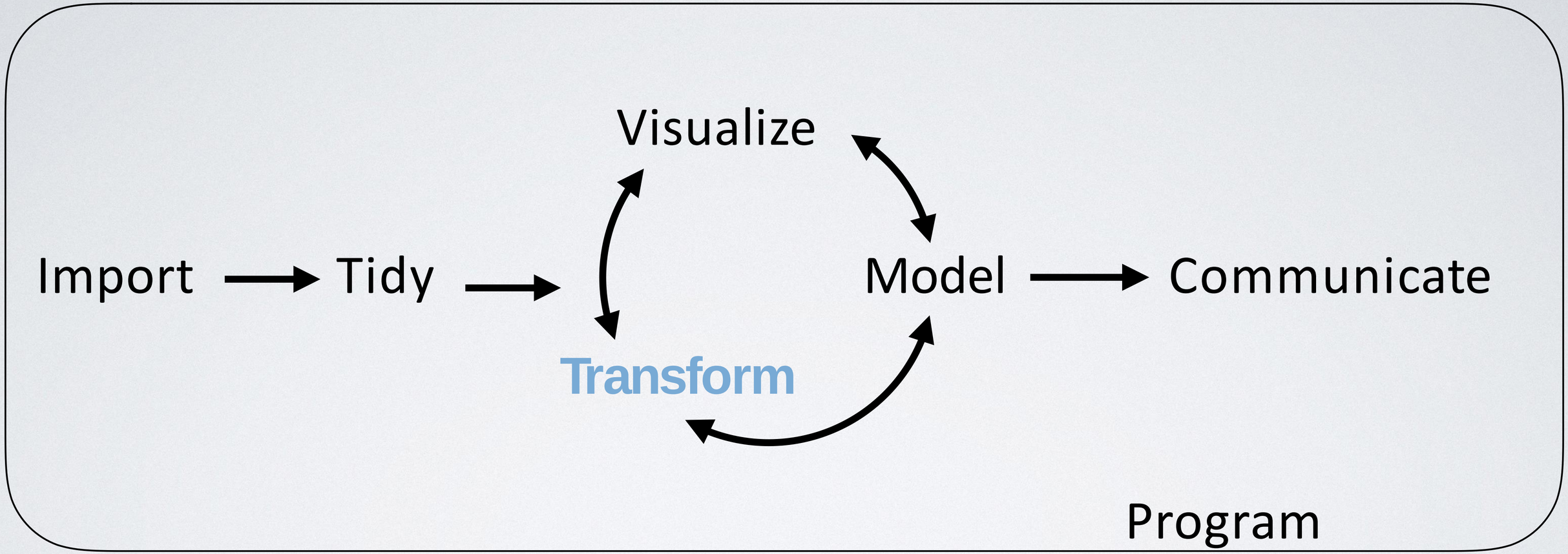
Saves data set as a csv on your computer.

```
write_csv(babynames, path = "babynames.csv")
```

Table to save

file path to saveat

(Applied) Data Science



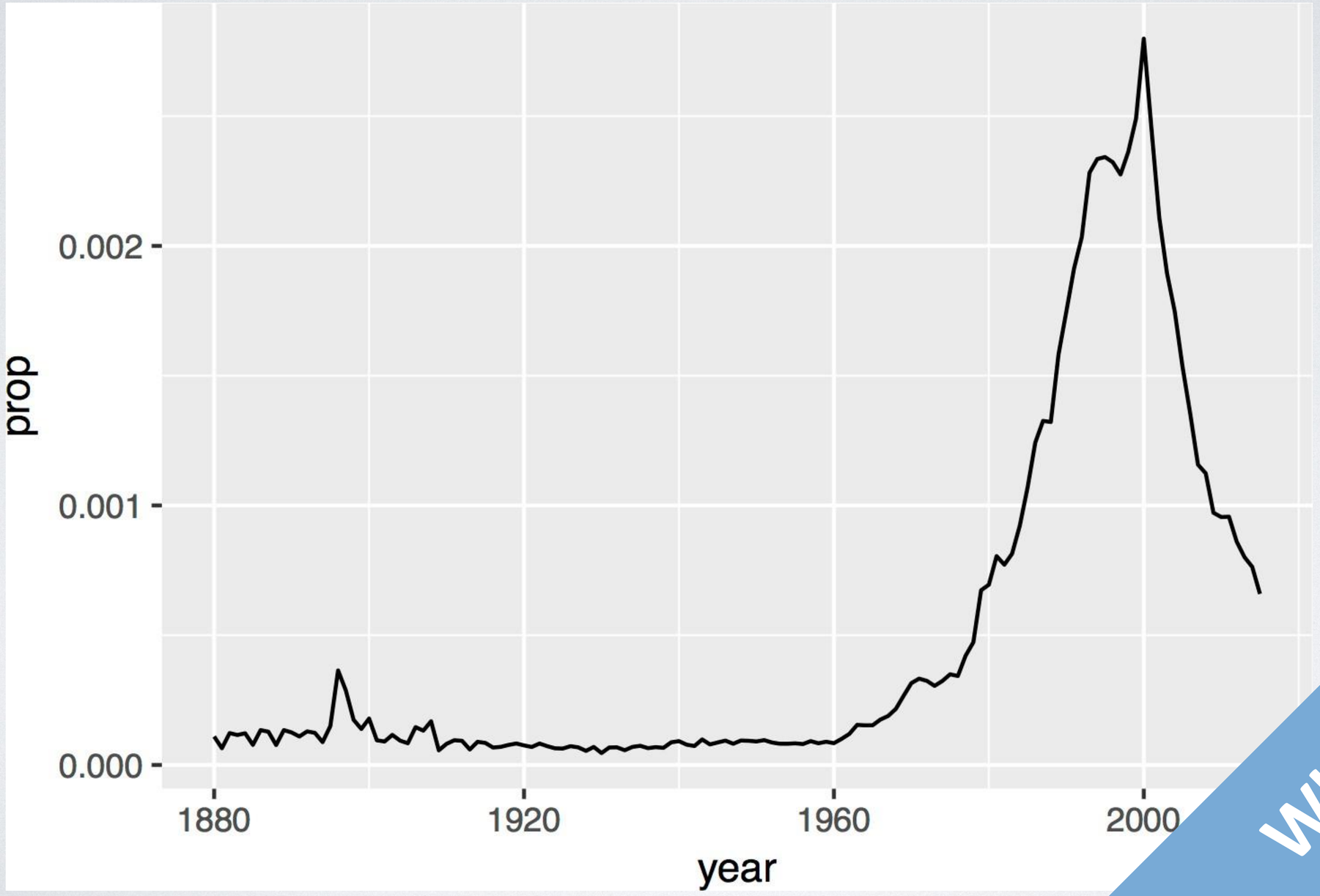
```
babynames
```

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

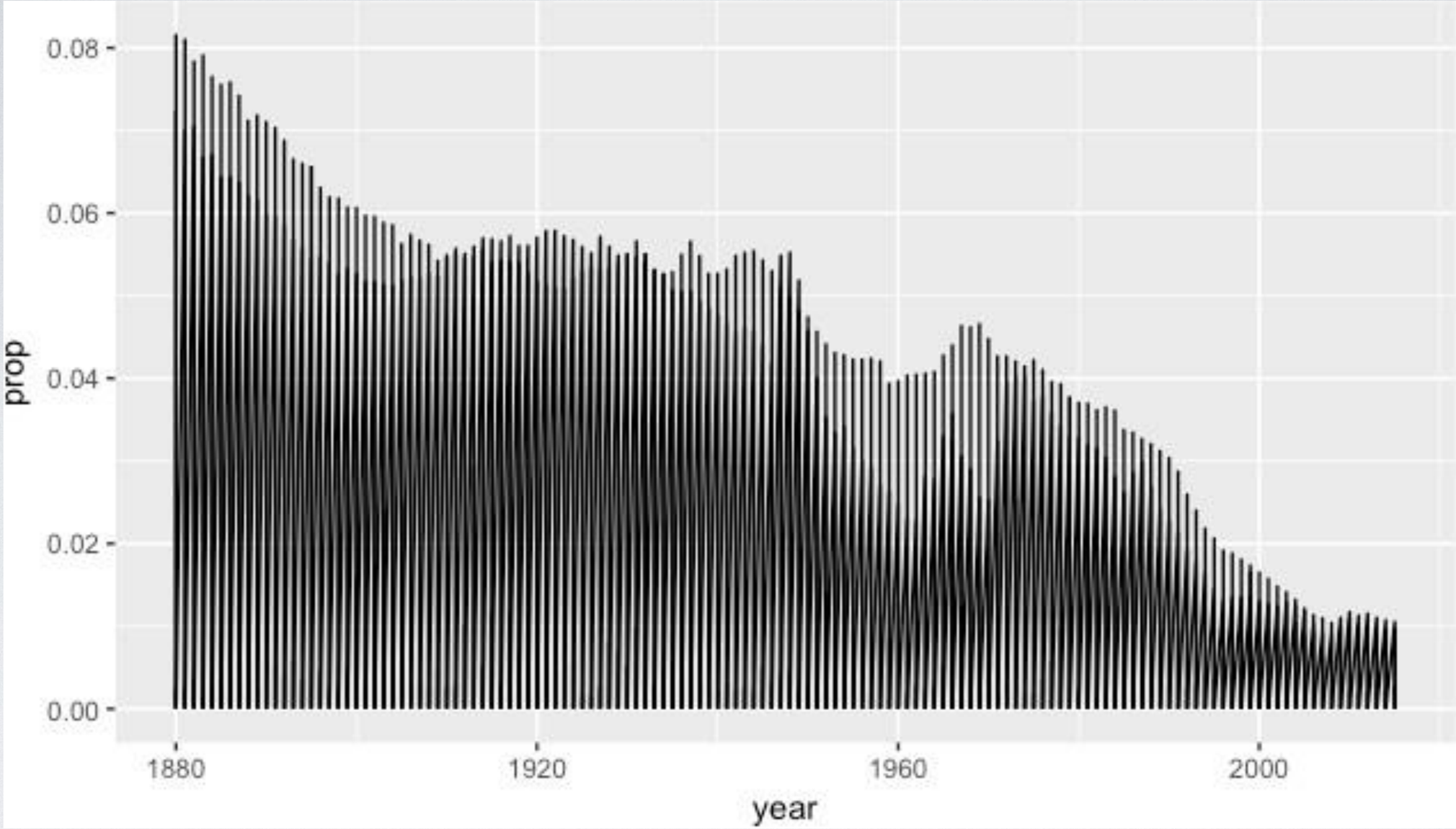
1–10 of 1,924,665 rows Previous 1 2 3 4 5 6 ... 100 Next

dplyr

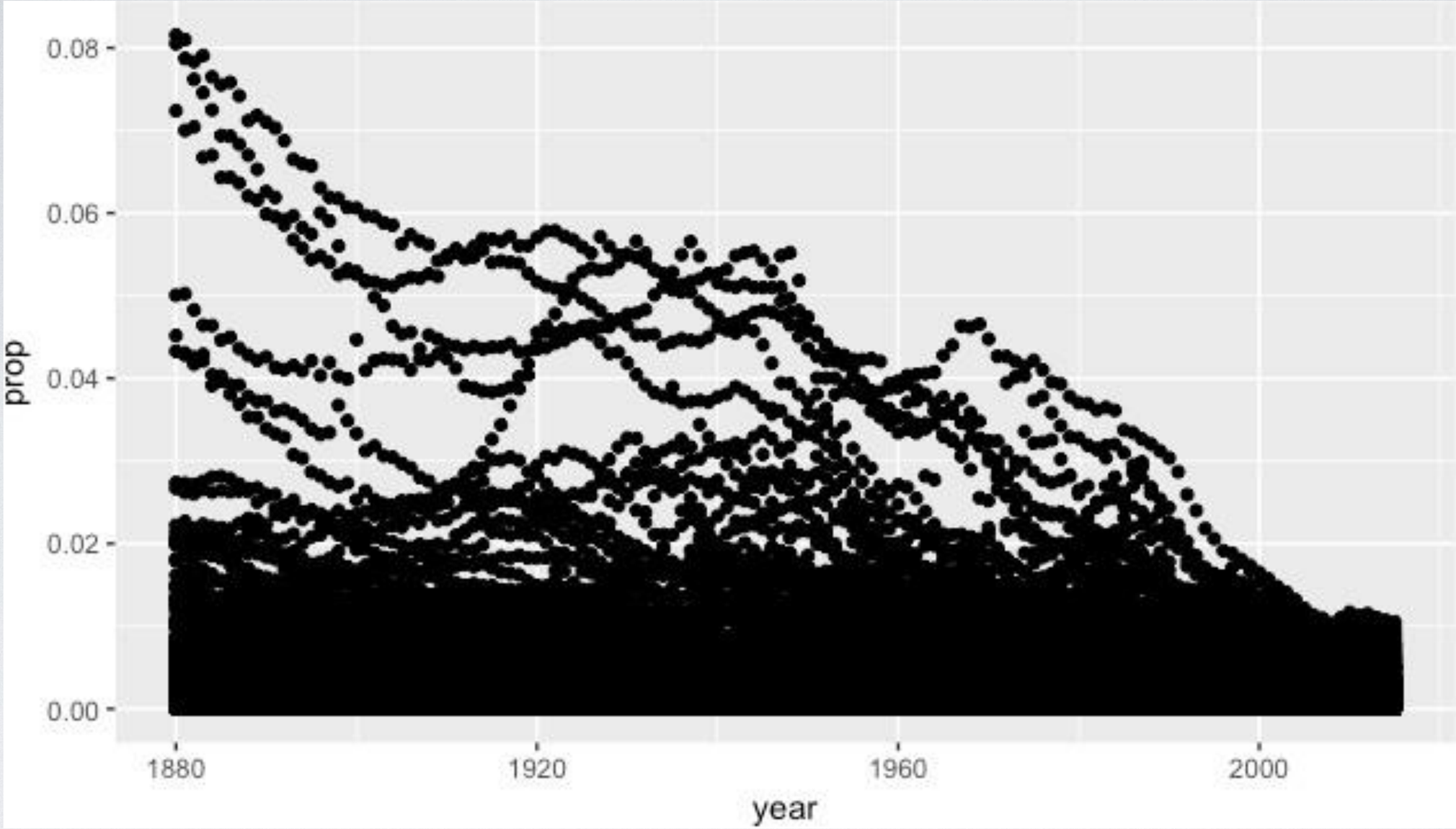
Proportion of boys with the name Garrett



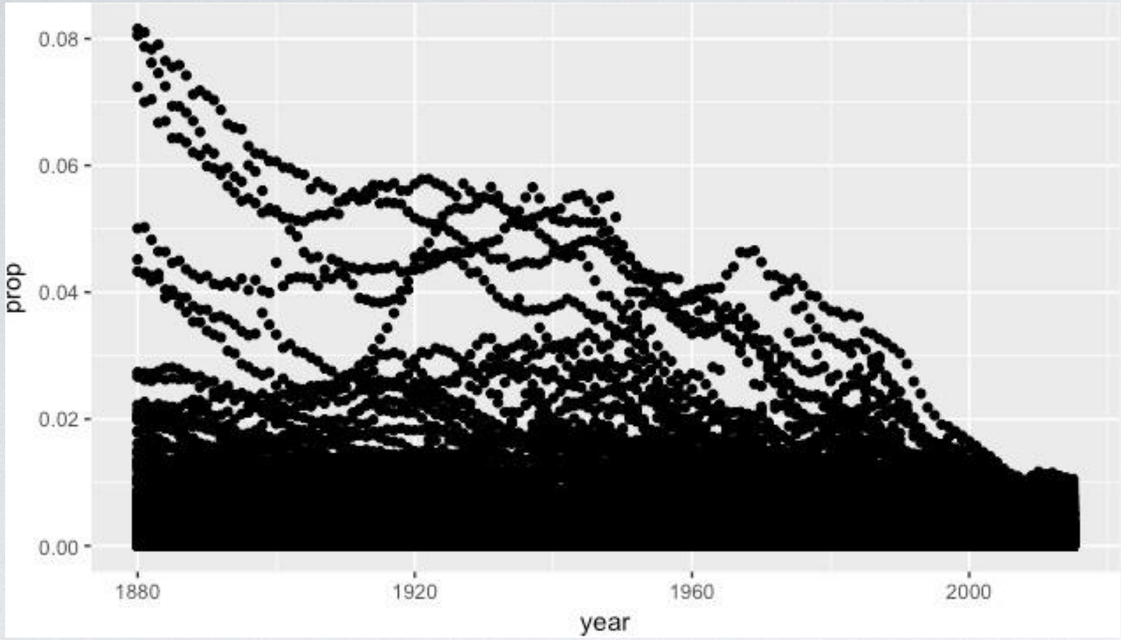
Which geom?

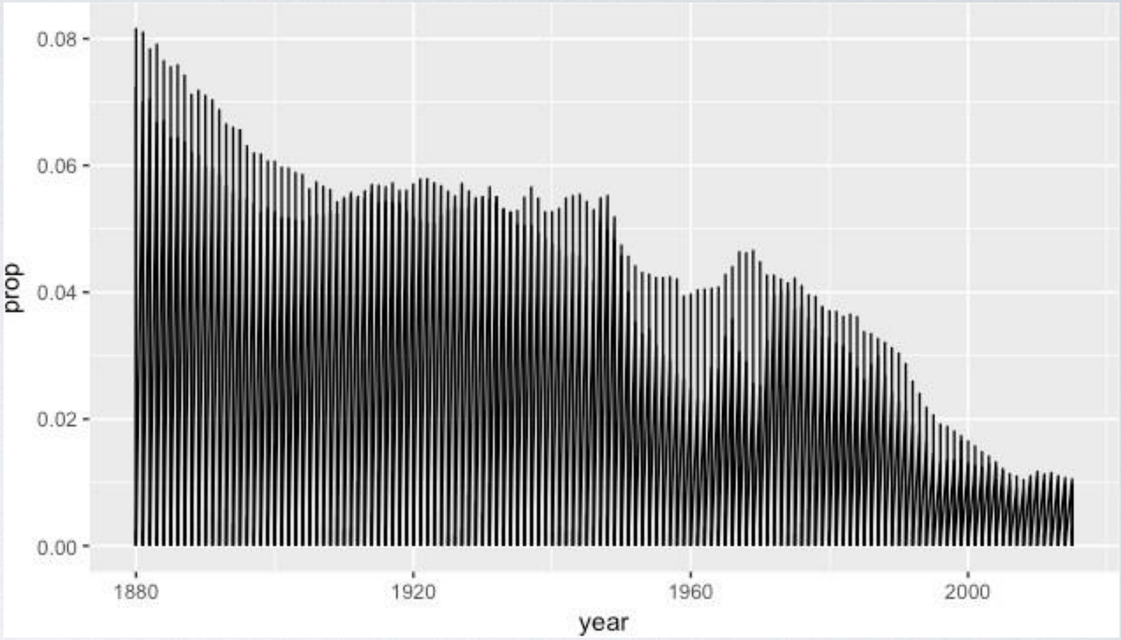


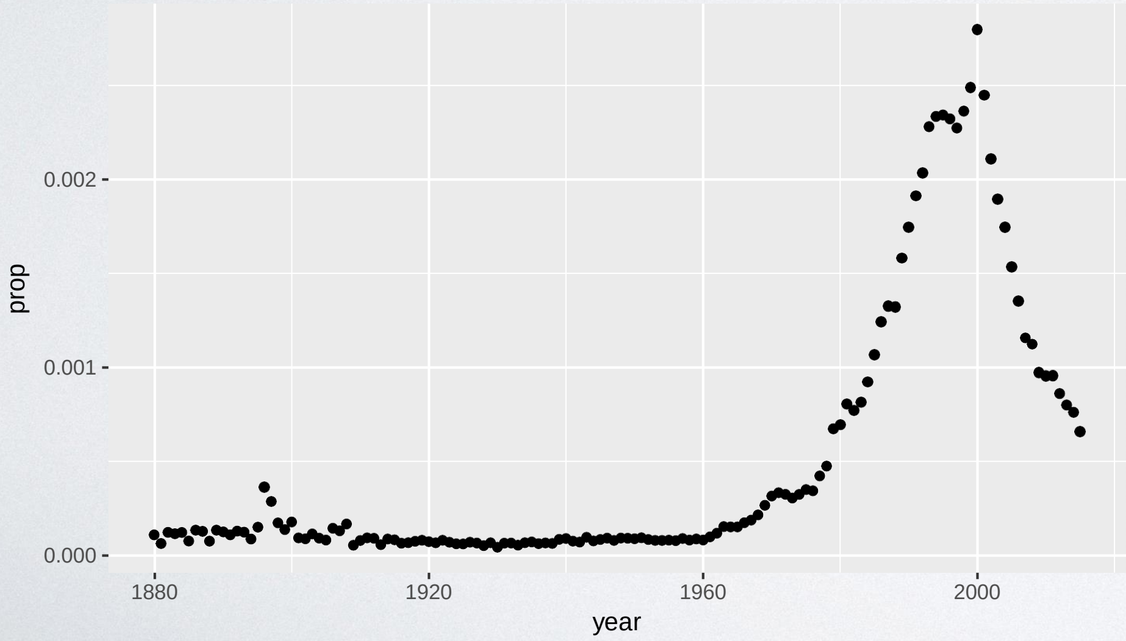

```
ggplot(data = babynames) +
  geom_line(mapping = aes(x = year, y = prop))
```

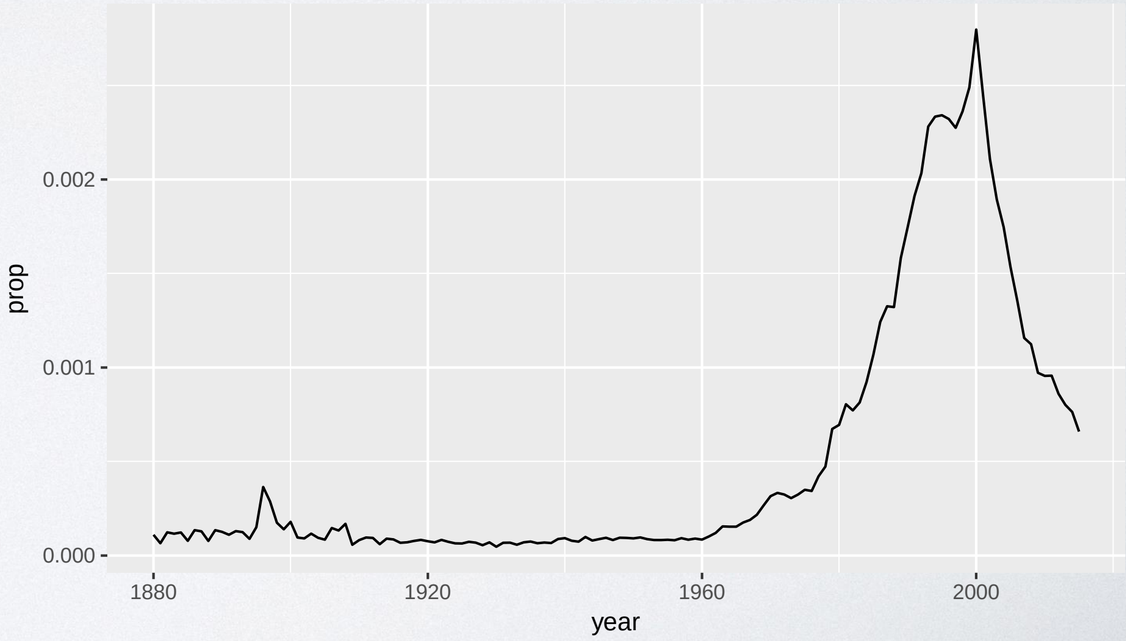



```
ggplot(data = babynames) +
  geom_point(mapping = aes(x = year, y = prop))
```


0.08
0.06
0.04
0.02
0.00
prop
1880
1920
1960
2000
year


0.08
0.06
0.04
0.02
0.00
prop
1880
1920
1960
2000
year


0.002
0.001
0.000
prop
1880
1920
1960
2000
year


0.002
0.001
0.000
prop
1880
1920
1960
2000
year

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 2

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(mpg, cty:class)
```

- - Select every column but

```
select(mpg, -c(cty, hwy))
```

starts_with() - Select columns that start with…

```
select(mpg, starts_with("c"))
```

ends_with() - Select columns that end with…

```
select(mpg, ends_with("y"))
```

select() helpers

contains() - Select columns whose names contain…

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

matches() - Select columns whose names match regular expression

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

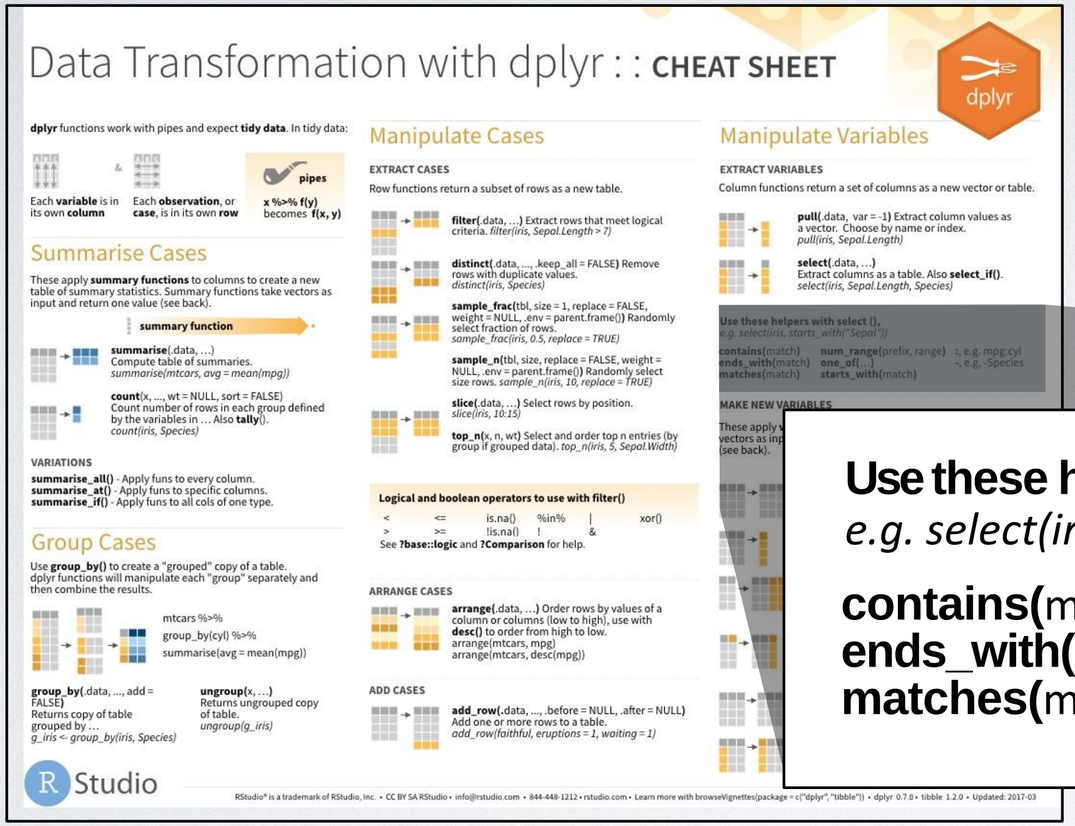
one_of() - Select columns whose names are one of a set

```
select(mpg, one_of(c("fl", "fuel", "Fuel")))
```

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

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

select() helpers



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

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

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")
```

babynames

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

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

`x < y`	Less than
`x > y`	Greater than
`x == y`	Equal to
`x <= y`	Less than or equal to
`x >= y`	Greater than or equal to
`x != y`	Not equal to
`x %in% y`	Group membership

```r
x <- 1
x >= 2
# FALSE
```

```r
x <- c(1, 2, 3)
x >= 2
# FALSE TRUE TRUE
```

Pop Quiz

What might NA stand for?

1 **"1"** **"one"** **NA**

Pop Quiz

What might NA stand for?

1 **"1"** **"one"** **NA**

Missing Value
(Not Available)

Pop Quiz

What is the result?

1 == 1

Pop Quiz

What is the result?

1 == 1

TRUE

Pop Quiz

What is the result?

1 == NA

Pop Quiz

What is the result?

1 == NA

NA

Pop Quiz

What is the result?

NA == NA

Pop Quiz

What is the result?

NA == NA

NA

Pop Quiz

What is the result?

is.na(NA)

TRUE

Logical tests

?Comparison

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

Your Turn 3

Use filter, babynames, and the logical operators to find:

- All of the rows where **prop** is greater than or equal to 0.08
- All of the children named "Sea"

```
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
```

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

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
`( )`	To group tests. & evaluates before |

```
x >= 2 & x < 3
```

↓

```
TRUE & TRUE
```

↓

```
TRUE
```

Your Turn 4

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

- *Boys* named Sue
- 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 == "Sue", sex == "M")
#    year   sex  name      n         prop
# 1  1917 M     Sue       7    0.0000073
# 2  1927 M     Sue       5    0.0000043
# …     … …       …       …            …
```

```
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
# …     …     …       …      …            …
```

Parentheses matter

```
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 == "Sue", sex == "M")
#    year   sex  name      n          prop
# 1  1917 M      Sue        7     0.0000073
# 2  1927 M      Sue        5     0.0000043
# …     … …        …        …            …
```

```
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
# …     …      …     …     …            …
```

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

desc()

Changes orderingto 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

Help me

What is the smallest value of **n**?

What is the largest?

```
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 1,924,655 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 1,924,655 more rows
```

%>%

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 %>%

%>%

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

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 5

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

1. Filters babynames to the girls that were born in 2017, *then…*
2. Selects the **name** and **n** columns, *then…*
3. Arranges the results so that the most popular names are near the top.

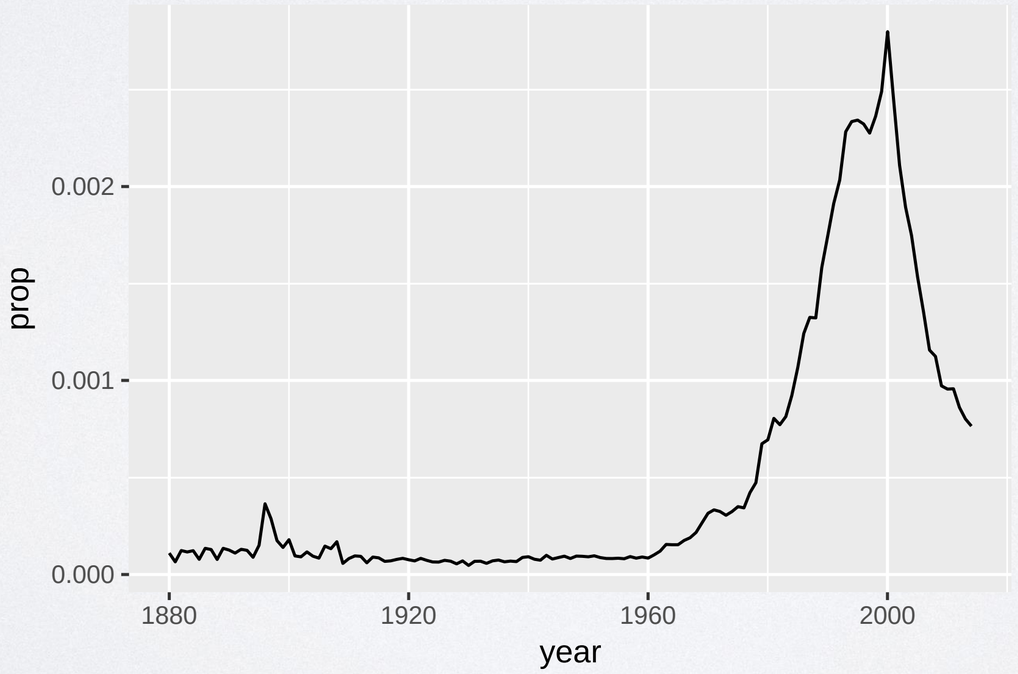
```
babynames %>%
  filter(year == 2017, sex == "F") %>%
  select(name, n) %>%
  arrange(desc(n))
#    name         n
#   1 Emma      19738
#   2 Olivia    18632
#   3 Ava       15902
#   4 Isabella  15100
#   5 Sophia    14831
#   6 Mia       13437
#   7 Charlotte 12893
#   8 Amelia    11800
#   9 Evelyn    10675
## … with 20,170 more rows
```

Payoff!

Your Turn 6 - 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

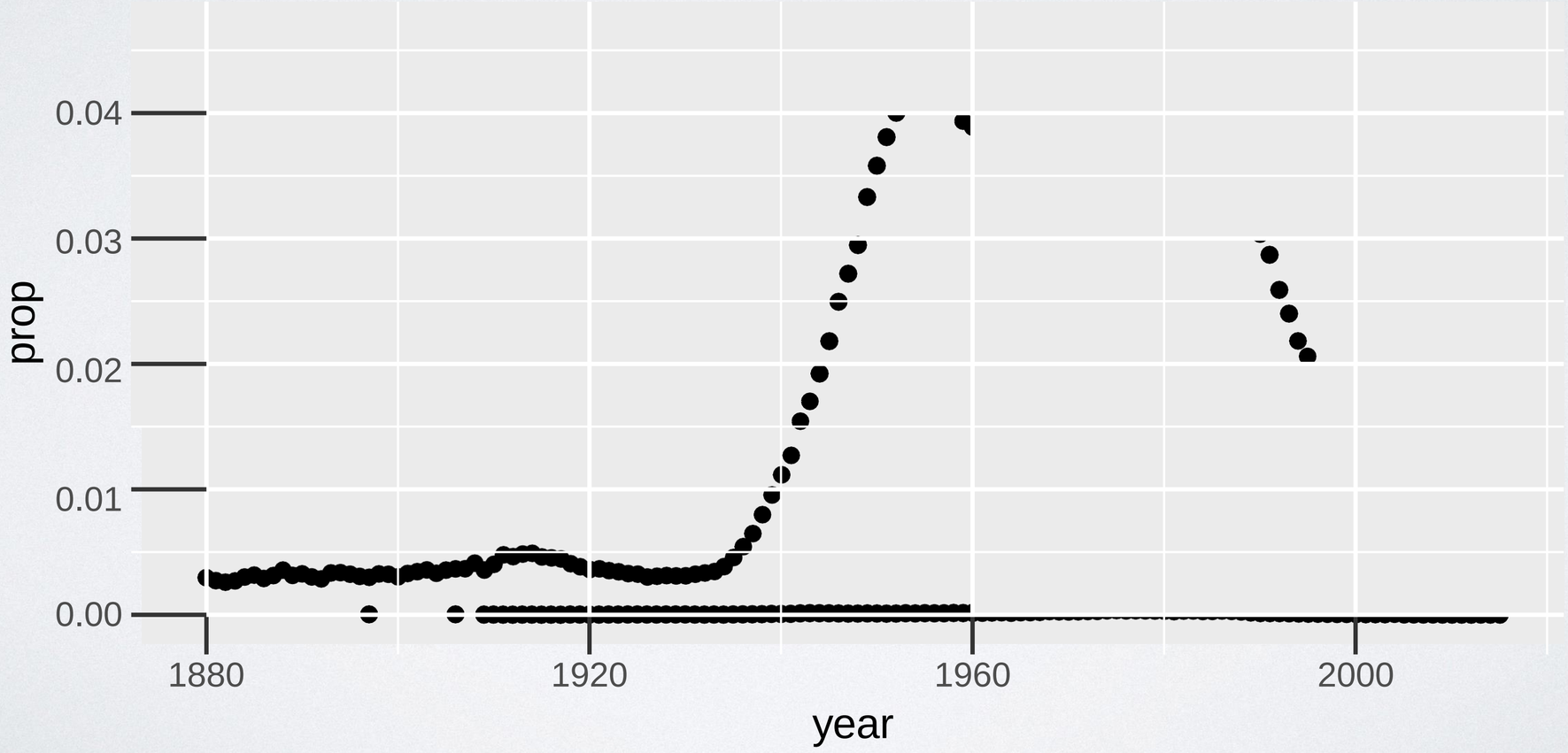
```
babynames %>%
  filter(name == "Garrett", sex == "M") %>%
  select(year, prop) %>%
  ggplot() +
    geom_line(mapping = aes(year, prop))
```



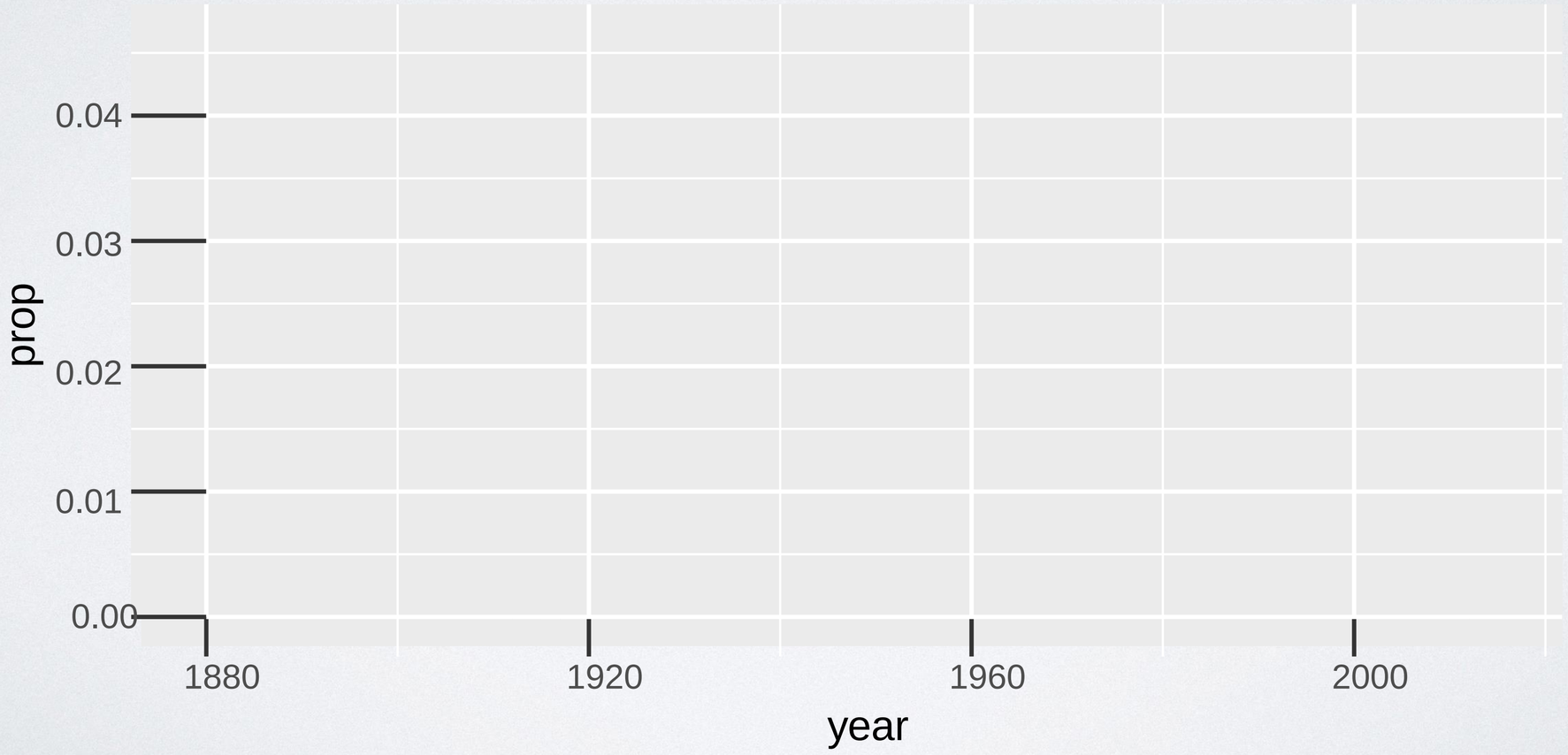
Plotting groups

```
babynames %>%
  filter(name == "Michael") %>%
  ggplot() +
    geom_point(mapping = aes(year, prop))
```

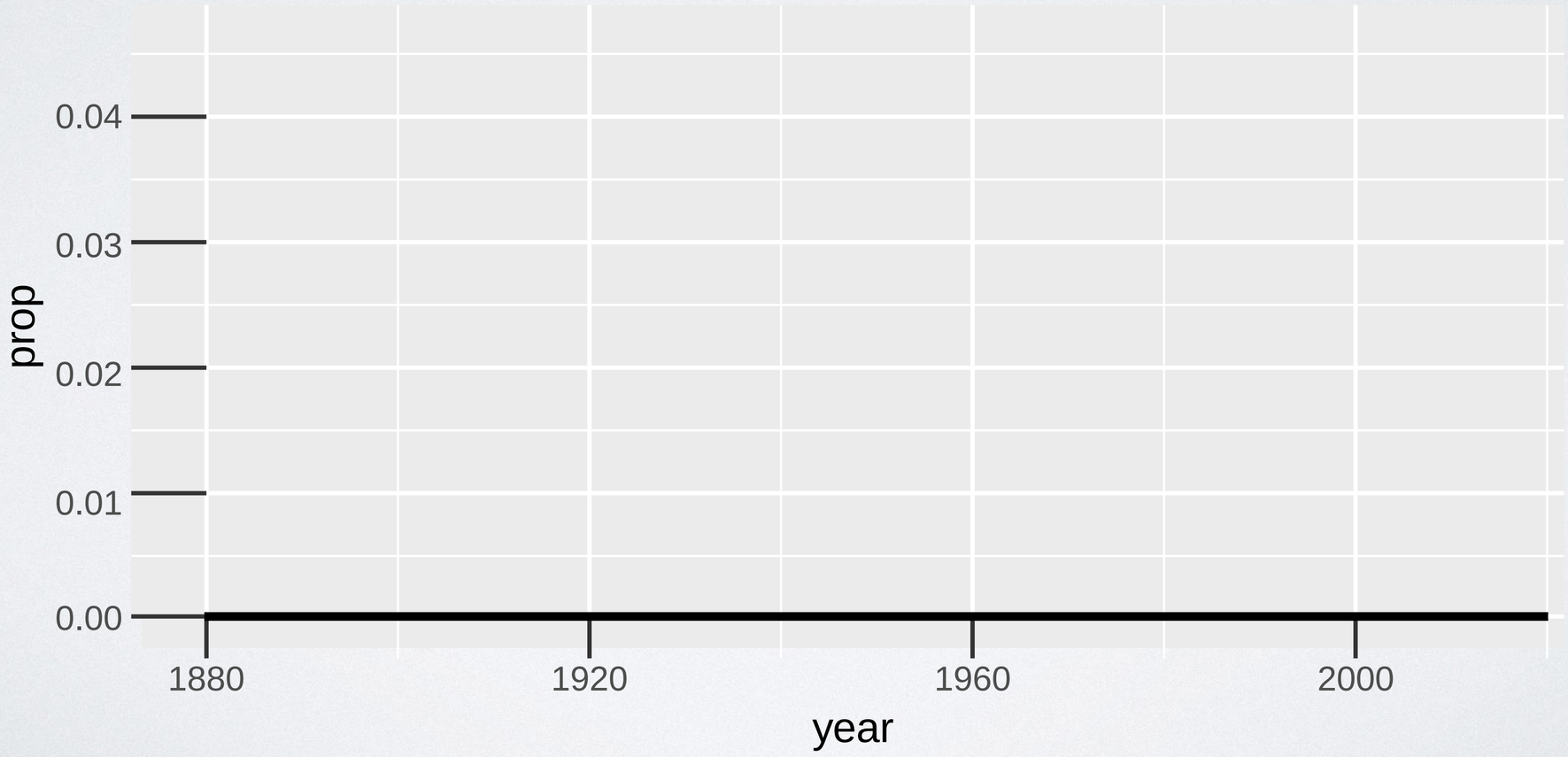
Did not filter on SEX



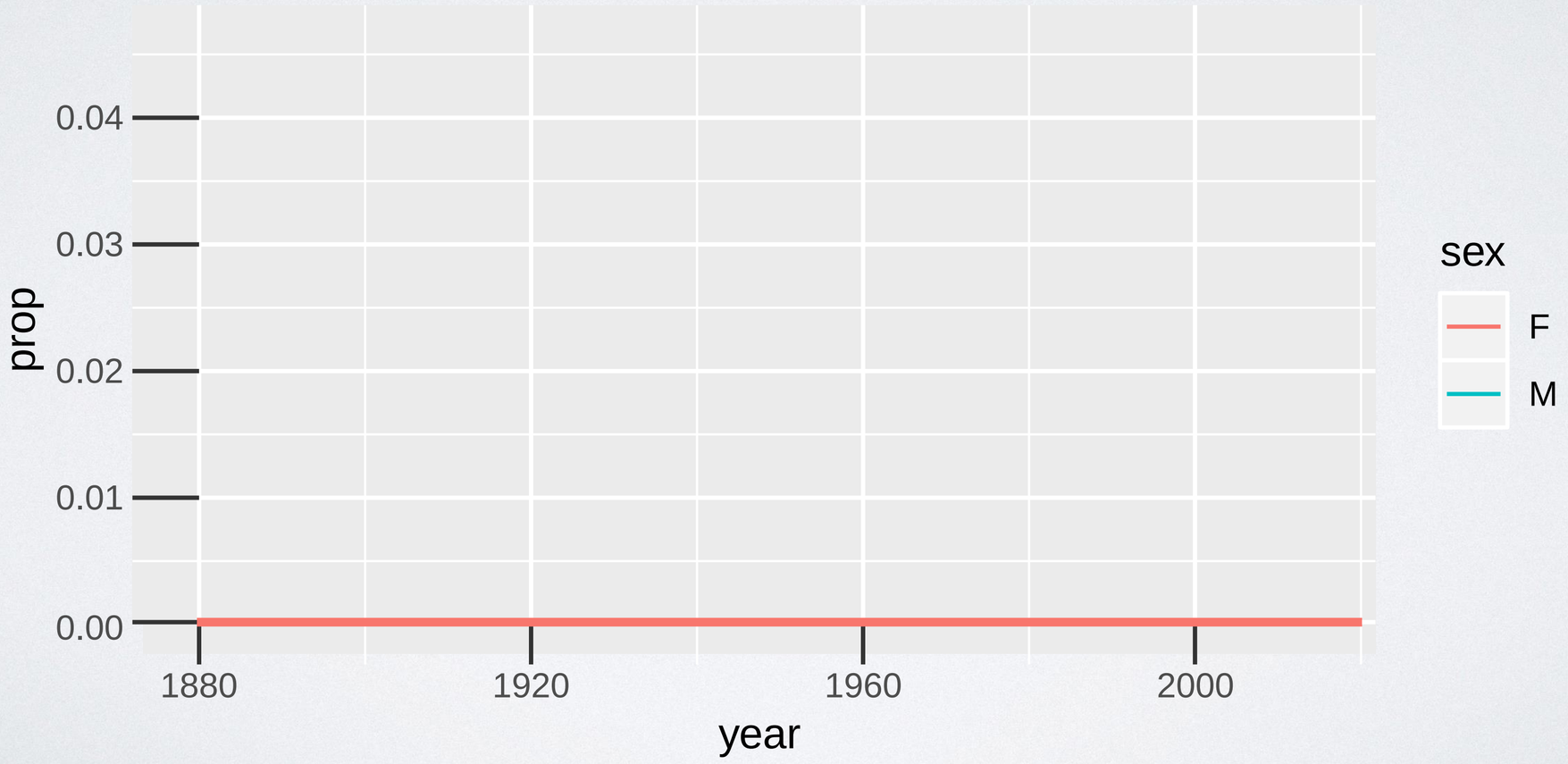
```
babynames %>%
  filter(name == "Michael") %>%
  ggplot() +
    geom_line(mapping = aes(year, prop))
```



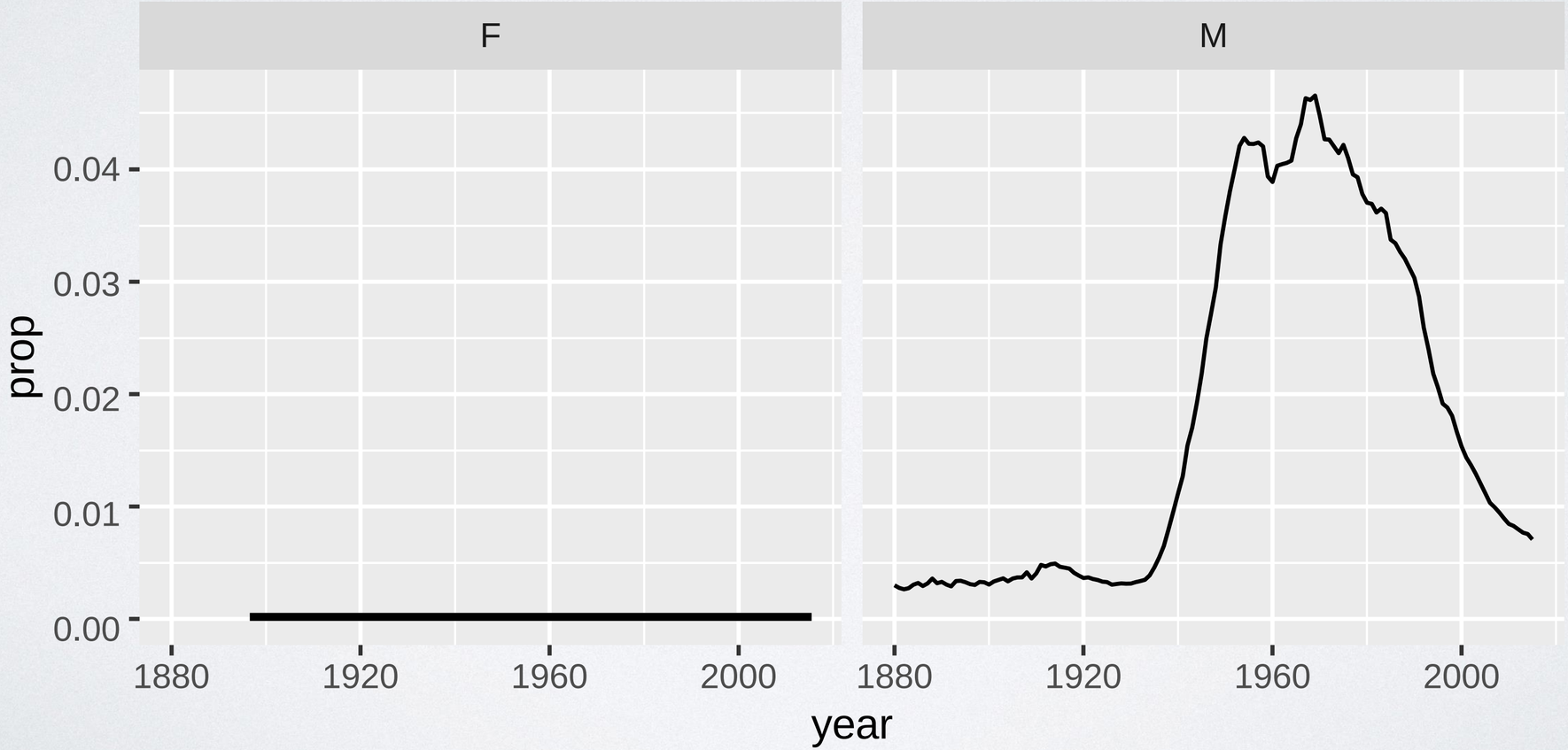
```
babynames %>%
  filter(name == "Michael") %>%
  ggplot() +
    geom_line(mapping = aes(year, prop, group = sex))
```



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



```
babynames %>% filter(name == "Michael") %>%
  ggplot() +
    geom_line(mapping = aes(year, prop)) +
    facet_wrap(~ sex)
```



What are the most popular names?

Quiz

Do we have enough information to:

1. Calculate the total number of children with each name?

Deriving information

summarise() - summarise **variables**

group_by() - group **cases**

mutate() - create new **variables**

summarise()

summarise()

Compute table of summaries.

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

babynames

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

→

total	max
348120517	99686

Your Turn 7

Complete the code to extract the rows where **name == "Khaleesi"**. Then use **summarise()** and **sum()** and **min()** 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  1964  2011
```

Summary functions

Take a vector as input.

Return a single value as output.

Summary Functions

TO USE WITH SUMMARISE ()

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

summary function

COUNTS

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

LOCATION

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

LOGICALS

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

POSITION/ORDER

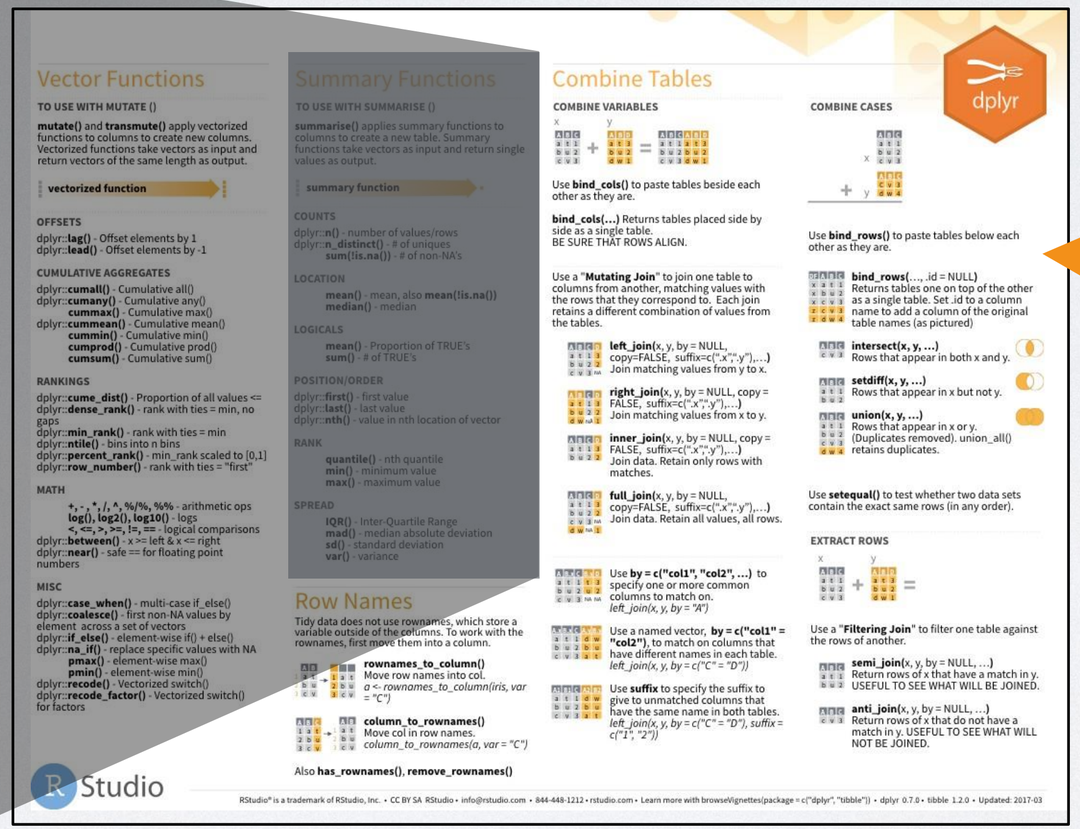
dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

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

SPREAD

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



on back

n()

The number of rows in a dataset/group

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

babynames

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

→

n
1924665

n_distinct()

The number of distinct values in a variable

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

babynames

year	sex	name	n	prop
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	nname
1924665	97310

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

```r
babynames %>%
  filter(name == "Khaleesi" & sex == "F")
```

```
##    year sex   name         n      prop
## 1  2011 F     Khaleesi    28 0.0000145
## 2  2012 F     Khaleesi   146 0.0000754
## 3  2013 F     Khaleesi   243 0.000126
## 4  2014 F     Khaleesi   369 0.000189
## 5  2015 F     Khaleesi   341 0.000175
## 6  2016 F     Khaleesi   371 0.000192
## 7  2017 F     Khaleesi   466 0.000249
```

```
babynames %>%
  filter(name == "Khaleesi" & sex == "F") %>%
  summarise(total = sum(n))
##    total
## 1   1964
```

Can we do this for each name?

Grouping cases

03-Transform-Exercises.Rmd

02-Transform-Data.Rmd

Preview Insert Run

```
---
title: "Transform Data"
output: html_notebook
---

```{r setup}
library(tidyverse)
library(babynames)

# Toy datasets to use

pollution <- tribble(
       ~city,    ~size, ~
  "New York", "large",
  "New York", "small",
    "London", "large",
    "London", "small",
   "Beijing", "large",
   "Beijing", "small",
)

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

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

Your Turn 1
```

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

```r
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 (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount (µg/m³)
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 (µg/m³)
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 (µg/m³)
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 (µg/m³)
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()

Groups cases by common values of one or more columns.

```
pollution %>%
  group_by(city)
```

```
# A tibble: 6 x 3
# Groups:   city [3]
  city     size  amount
  <chr>    <chr>  <dbl>
1 New York large     23
2 New York small     14
```

group_by()

city	particle size	amount (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount (µg/m³)
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 (µg/m³)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

city	particle size	amount (µg/m³)
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())
```

group_by()

Groups cases by common values.

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

sex	total
F	172371079
M	175749438

ungroup()

Removes grouping criteria from a data frame.

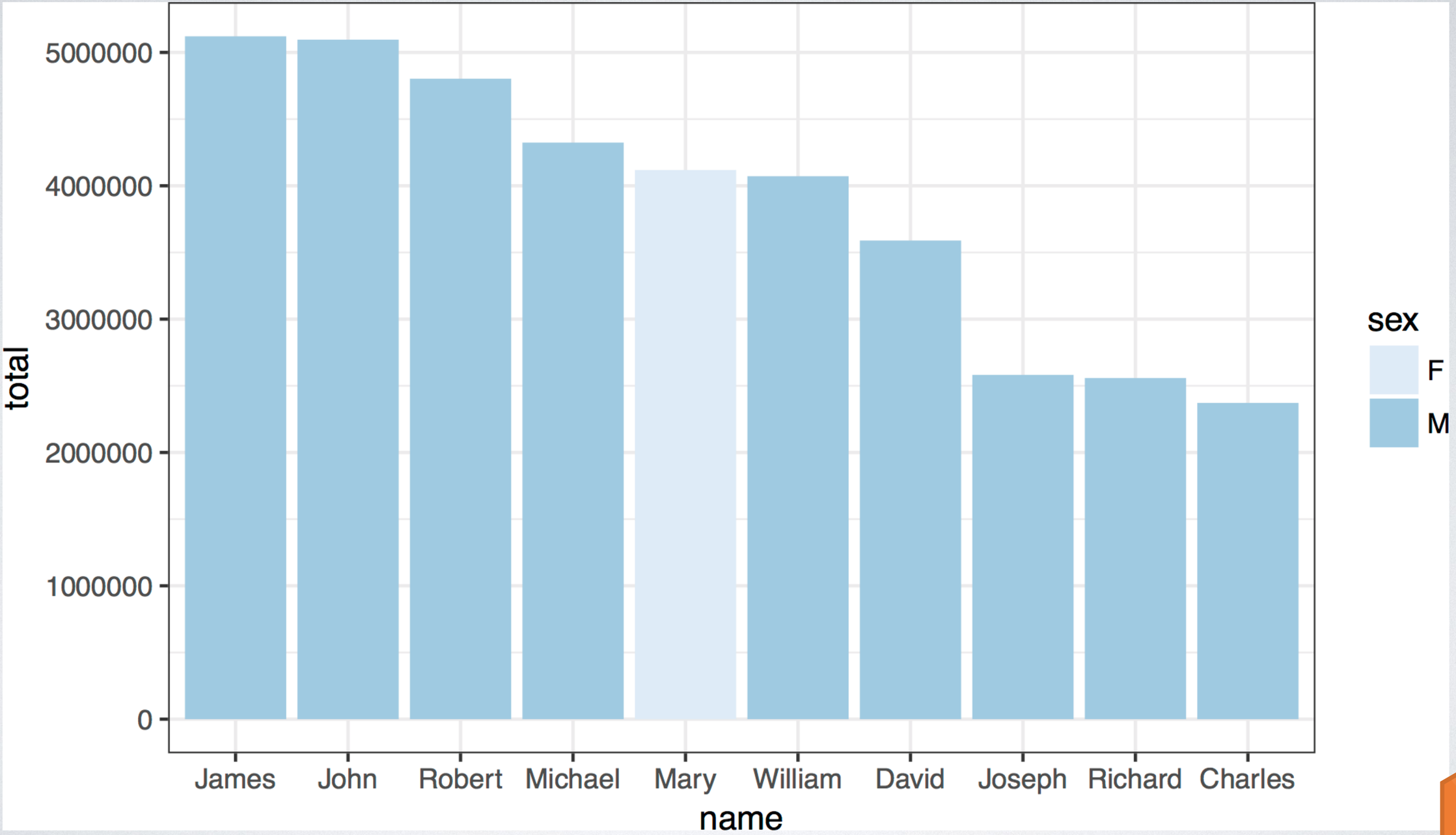
```
babynames %>%
  group_by(sex) %>%
  ungroup() %>%
  summarise(total = sum(n))
```

total
348120517

Your Turn 8

Complete the code with **group_by()**, **summarise()**, and **arrange()** to display the ten most popular **name** and **sex** combinations. Compute popularity as the *total* number of children with a given name and sex.

```
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 107,963 more rows
```


5000000
4000000
3000000
2000000
1000000
0
total
James
John
Robert
Michael
Mary
William
David
Joseph
Richard
Charles
name
sex
F
M

```r
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")
```

```
babynames %>%
  filter(name == "Khaleesi" & sex == "F") %>%
  summarise(total = sum(n))
##    total
## 1   1964
```

Can we do this for each name?

```
babynames %>%
  filter(name == "Khaleesi" & sex == "F") %>%
  summarise(total = sum(n))
##    total
## 1   1964


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

```
babynames %>%
  filter(name == "Khaleesi" & sex == "F") %>%
  summarise(total = sum(n))
```

Group by the variables you used to get your test case

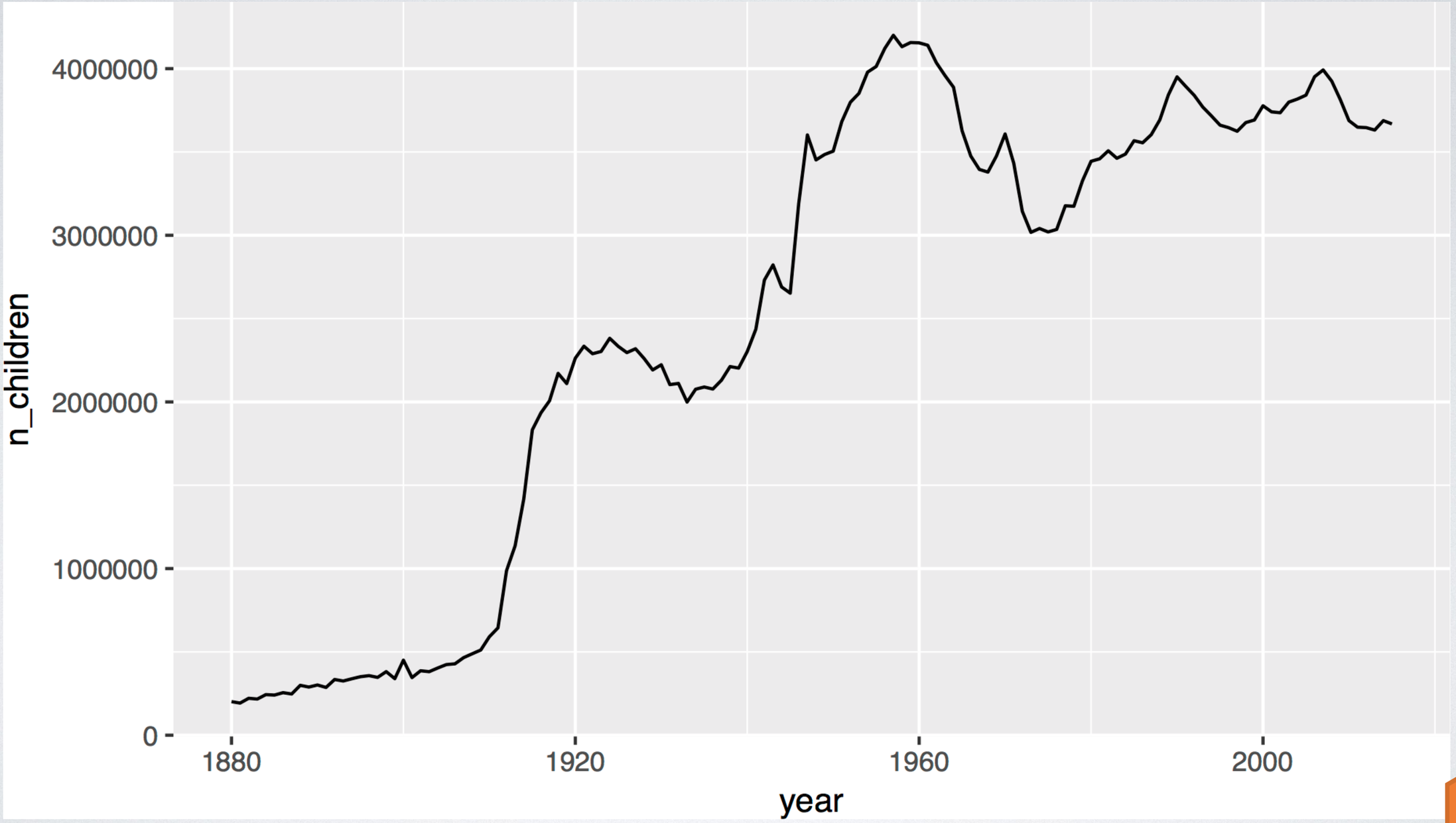
```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n))
```

Your Turn 9

Use group_by() to calculate the total number of children born **for every year**.

Plot the results as a line graph: total vs. year.

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


4000000
3000000
2000000
1000000
0
n_children
1880
1920
1960
2000
year

What was the top ranked name for each year?

Quiz

Do we have enough information to:

1. Rank names within each year?

mutate()

mutate()

Create new columns.

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

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	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
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	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

```
babynames %>%
  mutate(rank =          ?          )
```

Vectorized functions

Take a vector as input.

Return a vector of the same length as output.

Vector Functions

TO USE WITH MUTATE ()

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

vectorized function

OFFSETS

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

CUMULATIVE AGGREGATES

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

RANKINGS

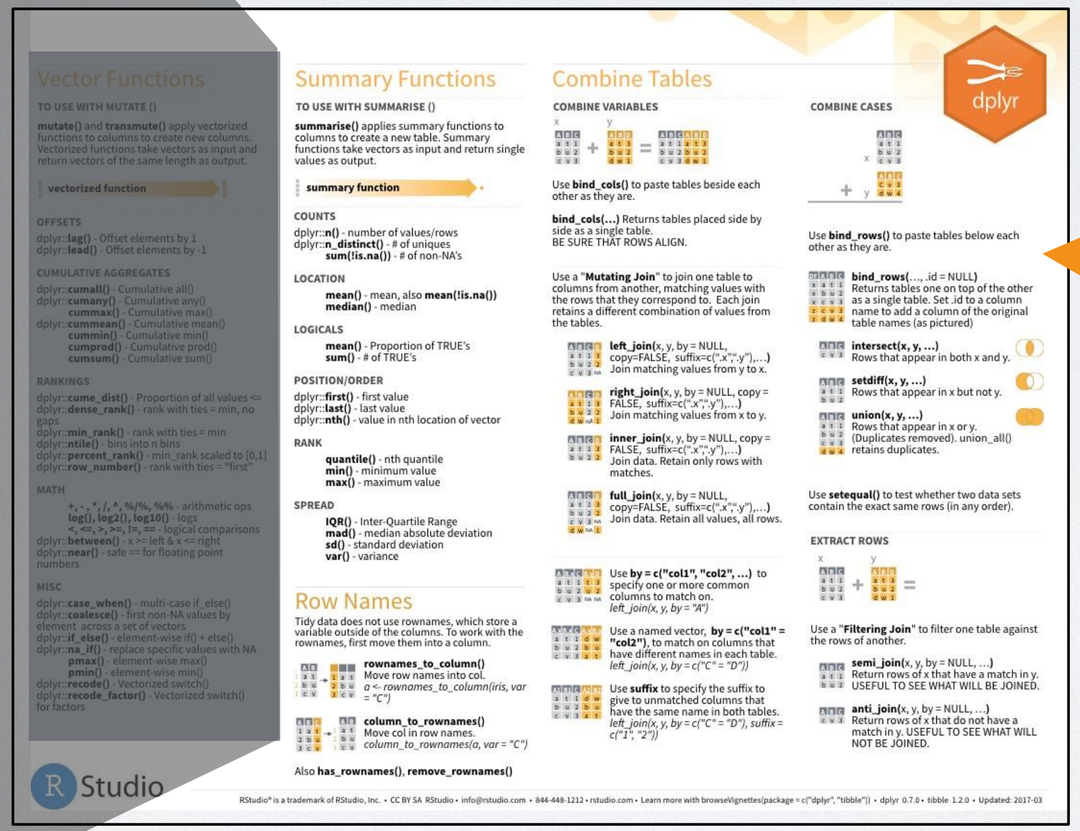
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

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

MISC

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



on back

min_rank()

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

```
min_rank(c(50, 100, 100, 1000))
# [1] 1 2 2 4
```

```
min_rank(desc(c(50, 100, 100, 1000)))
# [1] 4 2 2 1
```

Your Turn 10

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

```r
babynames %>%
  mutate(rank = min_rank(desc(prop)))
```

```
## A tibble: 1,924,665 x 6
#    year sex   name          n   prop  rank
# 1  1880 F     Mary       7065 0.0724    14
# 2  1880 F     Anna       2604 0.0267   709
# 3  1880 F     Emma       2003 0.0205  1131
# 4  1880 F     Elizabeth  1939 0.0199  1192
# 5  1880 F     Minnie     1746 0.0179  1427
# 6  1880 F     Margaret   1578 0.0162  1683
## … with 1,924,659 more rows
```

Your Turn 11

Group babynames by **year** and then re-rank the data. Filter the results to just rows where **rank == 1**.

```r
babynames %>%
  group_by(year) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  filter(rank == 1)


# A tibble: 138 x 6
# Groups:   year [138]
    year sex   name      n   prop  rank
 1  1880 M     John   9655 0.0815     1
 2  1881 M     John   8769 0.0810     1
 3  1882 M     John   9557 0.0783     1
# … with 135 more rows
```

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()**.

$

select()

Extract columns by name.

```
select(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

→

n
9655
9532
5927
5348
13
8769

Still a dataframe

$

Extract column contents as a vector.

```
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

→ `9655 9532 5927 5348 ...`

$

Extract column contents as a vector.

```
babynames$n
```

data frame

$

column name (no quotes)

pull()

Pipe friendly version of$

```
babynames %>% pull(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

→ `9655 9532 5927 5348 …`

Transform Data with

Slides CC BY-SARStudio