Programming Camp Day 2

Augustus Kmetz

2025-09-11

Welcome back!

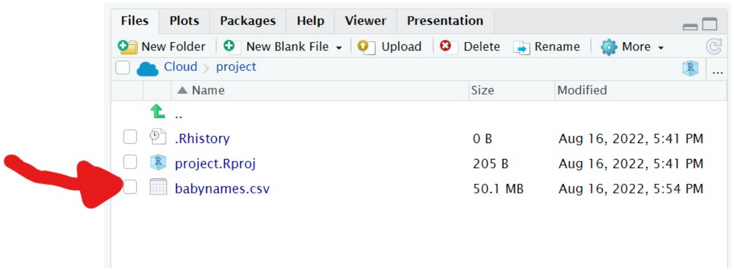
Plans for today:

- Importing data
- Transforming data
- Tidy data

babynames.csv

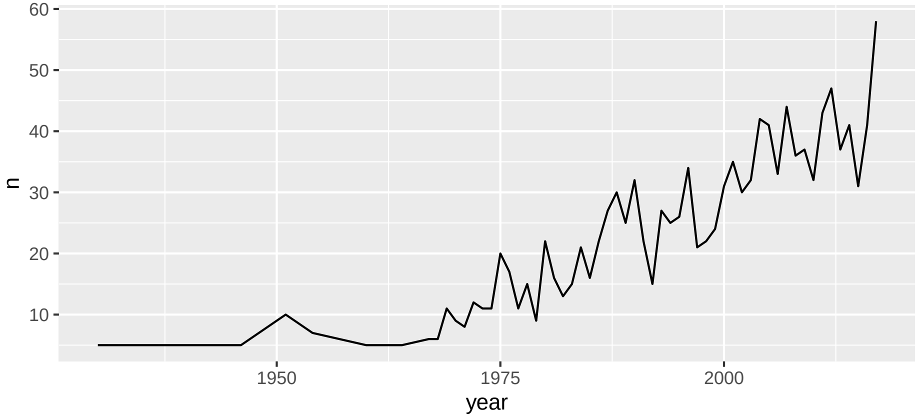
babynames.csv

- names and sex of babies born in the US from 1880 to 2017
- data compiled by the Social Security Administration
- 1.9M rows



Name	Size	Modified
..		
.Rhistory	0 B	Aug 16, 2022, 5:41 PM
project.Rproj	205 B	Aug 16, 2022, 5:41 PM
babynames.csv	50.1 MB	Aug 16, 2022, 5:54 PM

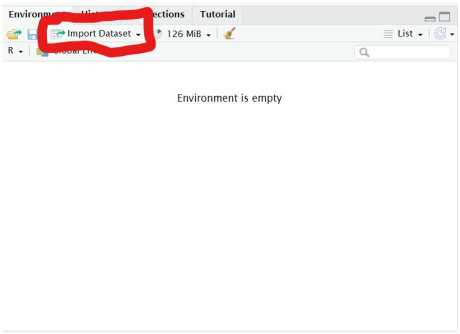
Number of girls with the name Shifra(h)



Q for later: Why is the correct spelling of my name not in this data set?

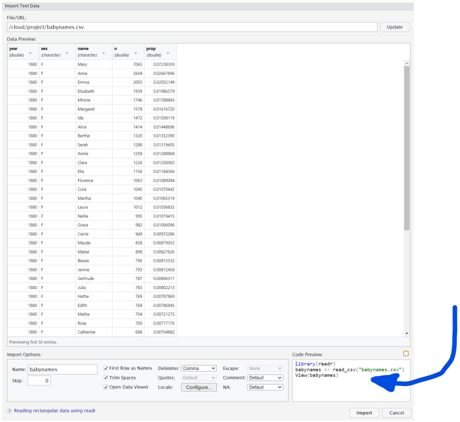
Import Data

Import a dataset (the wrong way)



Environment is empty

Import a dataset (the wrong way)



Import Text Data

File/URL:

/cloud/project/babynames.csv

Data Preview:

year (double)	sex (character)	name (character)	n (double)	prop (double)
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
1880	F	Annie	1258	0.01288868
1880	F	Clara	1226	0.01256083
1880	F	Ella	1156	0.01184366
1880	F	Florence	1063	0.01089084
1880	F	Cora	1045	0.01070642
1880	F	Martha	1040	0.01065519
1880	F	Laura	1012	0.01036832
1880	F	Nellie	995	0.01019415
1880	F	Grace	982	0.01006096
1880	F	Carrie	949	0.00972286
1880	F	Maude	858	0.00879053
1880	F	Mabel	808	0.00827826
1880	F	Bessie	796	0.00815532
1880	F	Jennie	793	0.00812458
1880	F	Gertrude	787	0.00806311
1880	F	Julia	783	0.00802213
1880	F	Hattie	769	0.00787869
1880	F	Edith	768	0.00786845
1880	F	Mattie	704	0.00721275
1880	F	Rose	700	0.00717176
1880	F	Catherine	688	0.00704882

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("babynames.csv")
View(babynames)
```

Q: But is this reproducible?

Working directory

- The current working directory is the location which R is currently pointing to
- Whenever you try to read or save a file without specifying the path explicitly that is where R will "look"
- To see the current working directory use `getwd()`
- To change the working directory use `setwd("path/to/directory")`
- When you open a R file that folder automatically becomes the wd
- **Best practice** is to specify files paths relative to the main folder of a project

Paths and directory names

- R inherits its file and folder naming conventions from UNIX and uses forward slashes for the directories, e.g. `/data/raw/FEMA/`
- Backslashes serve a different purpose - they are used as escape characters and to isolate special characters
- To avoid problems, directory names should **NOT** contain spaces and special characters

Back to babynames

- A common text file format is a comma delimited text file: .csv
- These files use comma as a column separator

```
year,sex,name,n,prop
1880,F,Mary,7065,0.0724
1880,F,Anna,2604,0.0267
```

- To read these files use the following command

```
my_data <- read_csv("path/to/filename.csv")
```

The readr package

- `readr` is for reading rectangular text data into R
- Very fast and easy to customize how file is parsed
- `readr` supports several file formats with general `read_<...>()` functions:
  - `read_csv()`: comma-separated files
  - `read_tsv()`: tab-separated files
  - `read_delim()`: general delimited files
  - `read_fwf()`: fixed-width files
  - `read_table()`: tabular files where columns are separated by white-space
  - `read_log()`: web log files

Importing other types of data

Rectangular data:

- package `haven` reads SPSS, Stata and SAS files
- package `readxl` reads excel files (both .xls and .xlsx)

Hierarchical data:

- `jsonlite` for json (common for browser-server communications)
- `xml2` for XML (common for textual data in web services)

And many many more...

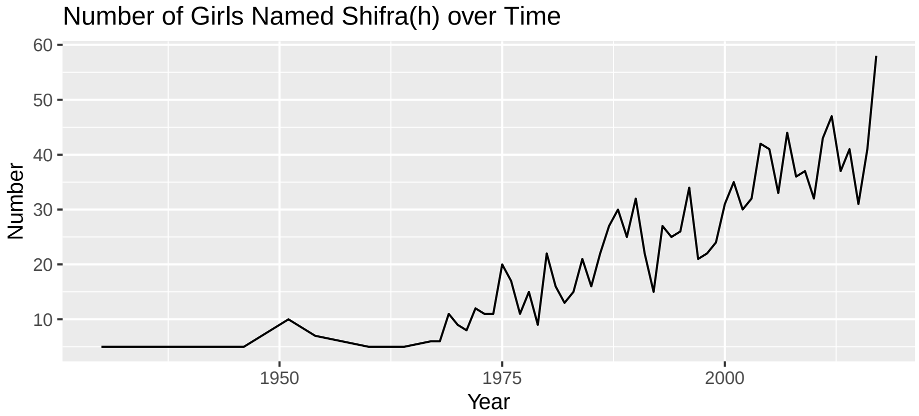
Back to babynames

```
babynames <- read_csv("data/babynames.csv")
```

```
babynames
```

```
# A tibble: 1,924,665 x 5
    year sex   name          n   prop
   <dbl> <chr> <chr>     <int>  <dbl>
 1  1880 F     Mary       7065 0.0724
 2  1880 F     Anna       2604 0.0267
 3  1880 F     Emma       2003 0.0205
 4  1880 F     Elizabeth  1939 0.0199
 5  1880 F     Minnie     1746 0.0179
 6  1880 F     Margaret   1578 0.0162
 7  1880 F     Ida        1472 0.0151
 8  1880 F     Alice      1414 0.0145
 9  1880 F     Bertha     1320 0.0135
10  1880 F     Sarah      1288 0.0132
# i 1,924,655 more rows
```

Back to babynames

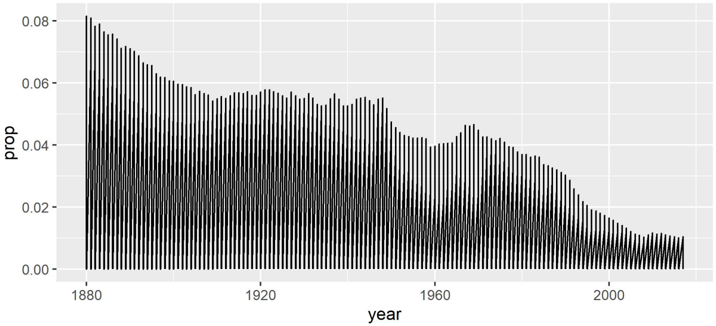


Q: Which geom?

Plotting babynames

What happens if we plot the data?

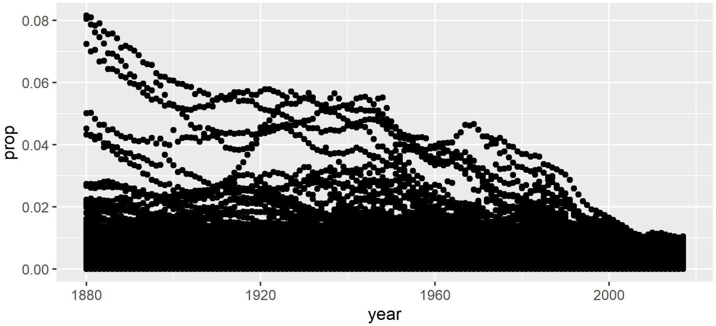
```
ggplot(babynames) +
  geom_line(mapping = aes(x = year, y = prop))
```



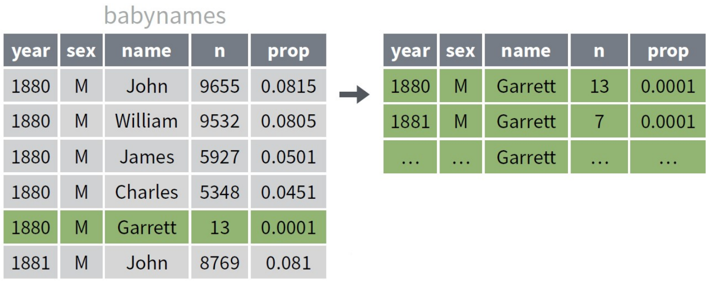
Plotting babynames

Try a different geom?

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



How to isolate?



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	…	…

Transforming data

Transforming data

The `dplyr` package is part of the `tidyverse` which:

- introduces a **grammar** for transforming tabular data
- is fast on data frames (written in C++) speed of C with ease of R
- intuitive to write and easy to read esp when using chaining syntax

dplyr verbs (functions)

`dplyr` verbs handle the vast majority of your data needs:

- `filter()` - for picking observations by their values
- `select()` - for picking variables by their names
- `arrange()` - for reordering rows
- `mutate()` - for creating new variables with functions on existing variables
- `summarise()` - for collapsing many values down to a single summary

All of the above can be done using ugly and slow base R functions

The structure of `dplyr` functions

All verbs work similarly:

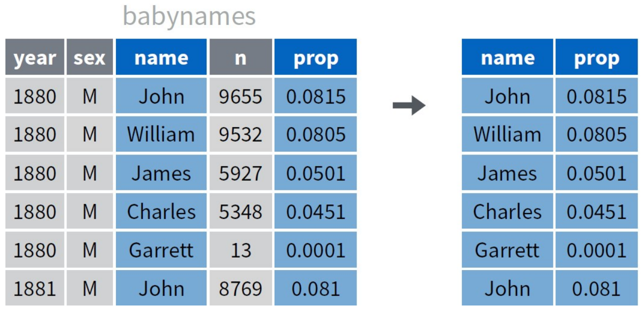
- The first argument is a tibble (or data frame)
- The subsequent ones describe what to do, using the variable names
- The results is a new tibble

select()

Extract columns by name

```
select(.data, name_col_1, name_col_2, ...)
```

```
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

select() helpers

: - select range of columns

```
select(babynames, year:name)
```

- - select every column but ...

```
select(babynames, -prop)
```

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

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

ends_with() - select every column that ends with ...

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

Any many more!

Quiz

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

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

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

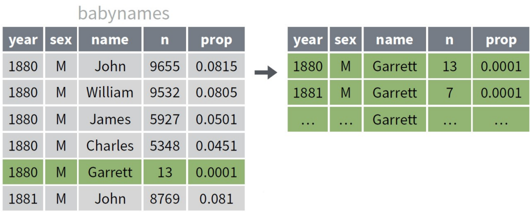
```
select(babynames, ends_with("n"))
```

filter()

Extract rows that meet logical criteria

```r
filter(.data, condition_1, condition_2, ...)
```

```r
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	…	…

Logical tests

`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

Logical tests

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

```
[1] FALSE
```

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

```
[1] FALSE  TRUE  TRUE
```

Missing Values

`NA` stands for a missing value

Handling missing/infinite/divide by 0 values can be a bit tricky.

What is the result?

```
1 == 1
1 == NA
NA == NA
1 + Inf
1 + NaN
Inf == Inf
```

Logical tests

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

Same syntax works for `Inf` and `NaN`: `is.inf()`, `is.nan()`

Your turn!

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

- Make a plot of your name's popularity over time (or linguistically similar name)
- Which name had the highest ever number/proportion of babies?

Two common mistakes

1. Using = instead of ==

```r
filter(babynames, name = "Shifra")
filter(babynames, name =="Shifra")
```

2. Forgetting quotes

```r
filter(babynames, name == Shifra)
filter(babynames, name =="Shifra")
```

Boolean operators

a & b	and
a | b	or
xor(a , b)	exactly or
!a	not
( )	To group tests . & evaluates before |

Your turn!

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

- Boys named Leslie
- Names that were used by exactly 5 or 6 children in 1880
- Names that are one of Anakin, Leia, Luke

```
filter(babyname, name == "Sea" | name == "Anemone")
```

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 use %in%)

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

arrange()

Order rows from smallest to largest values

```
arrange(.data, order_first, order_second, ...)
```

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

order rows from largest to smallest values

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

Q: Why is the name Shifrah not in `babynames`?

Steps

Goal: create a data set called `top_5_M_2015` that contains the 5 most popular boy baby names from 2015

Steps

1. filter to boys born in 2015
2. select the name and `n` columns
3. arrange these columns so that the most popular names appear near the top
4. use `head` to take the top 5

Steps

```r
top_5_M_2015 <- filter(babynames, year == 2015, sex == "M")
top_5_M_2015 <- select(top_5_M_2015, name, n)
top_5_M_2015 <- arrange(top_5_M_2015, desc(n))
top_5_M_2015 <- head(top_5_M_2015, n = 5)
```

```r
top_5_M_2015 <-
  head(arrange(select(filter(babynames, year == 2015, sex == '
  name, n), desc(n)), n = 5)
```

The pipe operator %>%

```r
filter(babynames, year == 2015, sex == "M")
babynames %>% filter(year == 2015, sex == "M")
```

Passes results on left into first argument of function on right

In pipe notation you use:

- `x %>% f(y)` rather than `f(x, y)`
- `x %>% f(y) %>% h(z)` rather than `h(f(x,y),z)`

This is similar to Unix pipes

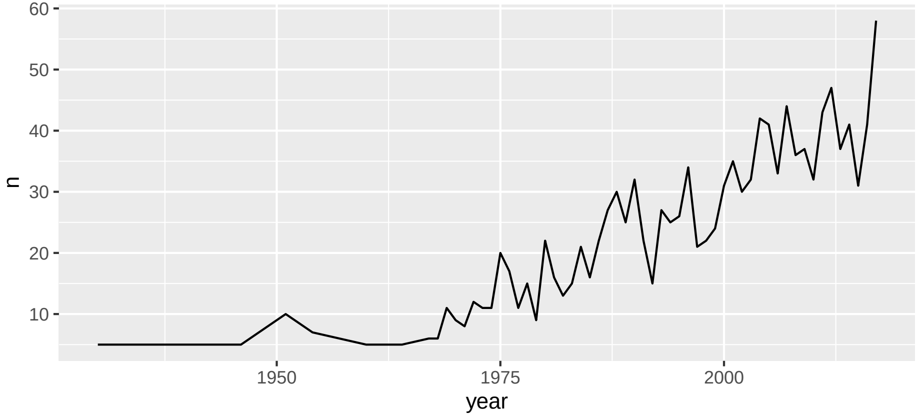
Pipes

```r
top_5_M_2015 <-
  babynames %>%
  filter(year == 2015, sex == "M") %>%
  select(name, n) %>%
  arrange(desc(n)) %>%
  head(n = 5)
```

Try to use %>% to write a sequence of functions that make a plot of your name's popularity over time (or linguistically similar name)

Pipes

```
babynames %>%
  filter(name == "Shifra", sex == "F") %>%
  ggplot() +
  geom_line(aes(x = year, y = n))
```



What are the most popular names?

Do we have tools to:

1. Calculate the total number of children with each name?
2. Calculate the number of unique names over time?
3. Calculate the share of babies captured by the top 10 names?

...not quite yet

summarise()

compute table of summaries

```
summarise(.data,
          summary_stat_1 = function_1(variable_1),
          summary_stat_2 = function_2(variable_2))
```

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

```
babynames %>%
  filter(name == "Shifra") %>%
  summarise(total = sum(n), max = max(n))
```

Q: How can we do this for each name?

group_by()

Groups cases by common values

```
group_by(.data, first_variable, second_variable, ...)
```

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

ungroup() - removes grouping criteria from a data frame

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

Elementary but useful summary functions

Aggregation functions return one value per group

- `min(x)`, `median(x)`, `max(x)`, `quantile(x, p)`
- `n()`, `n_distinct()`, `sum(x)`, `mean(x)`
- `sum(x > 10)`, `mean(x > 0)`
- `sd(x)`, `var(x)`

Window functions return multiple values per group

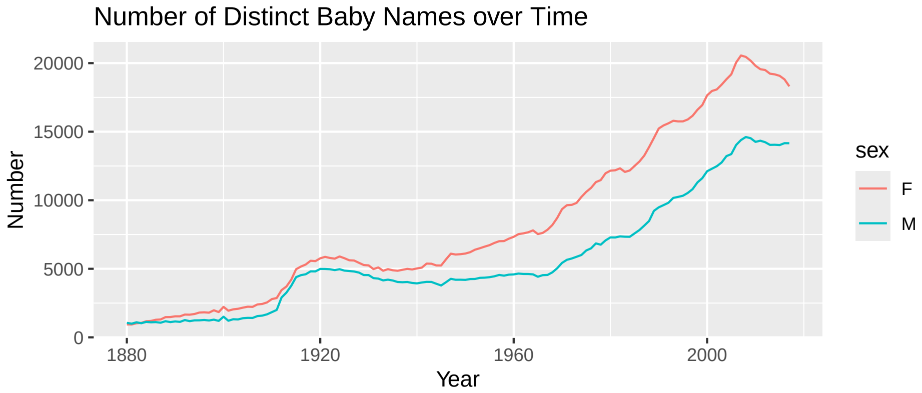
- `top_n()`
- `lead()`, `lag()`

Your turn!

Partner up

1. For each year calculate the number of distinct names by sex. Plot these time series.
2. Plot the share of babies with a name among the top 10 names over time by sex

Distinct names Over time

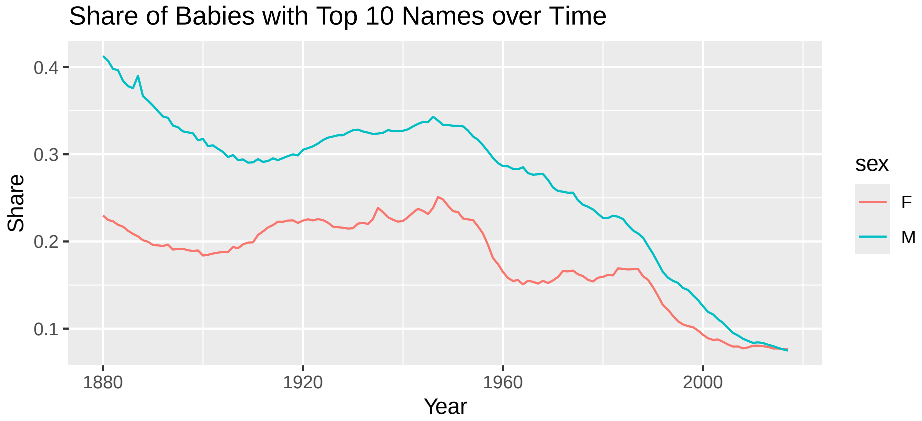


Note: Only names with at least five records are included in the SSA data.

Distinct names Over time

```
babynames %>%
  group_by(year, sex) %>%
  summarise(n_names = n_distinct(name)) %>%
  ggplot(aes(x = year, y = n_names, color = sex)) +
  geom_line() +
  labs(x = "Year",
       y = "Number",
       title = "Number of Distinct Baby Names over Time",
       caption = "Note: Only names with at least five records
```

Top share over time



Top share over time

```r
babynames %>%
  group_by(year, sex) %>%
  arrange(prop) %>%
  top_n(10, wt = prop) %>%
  summarise(total_prop = sum(prop)) %>%
  ggplot(aes(x = year, y = total_prop, color = sex)) +
  geom_line() +
  labs(x = "Year",
       y = "Share",
       title = "Share of Babies with Top 10 Names over Time")
```

`mutate()`

Create new columns

```
mutate(.data,
        new_var_1 = function_1(variable_1),
        new_var_2 = function_2(variable_2))
```

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

Any vectorized function can be used with `mutate()` including:

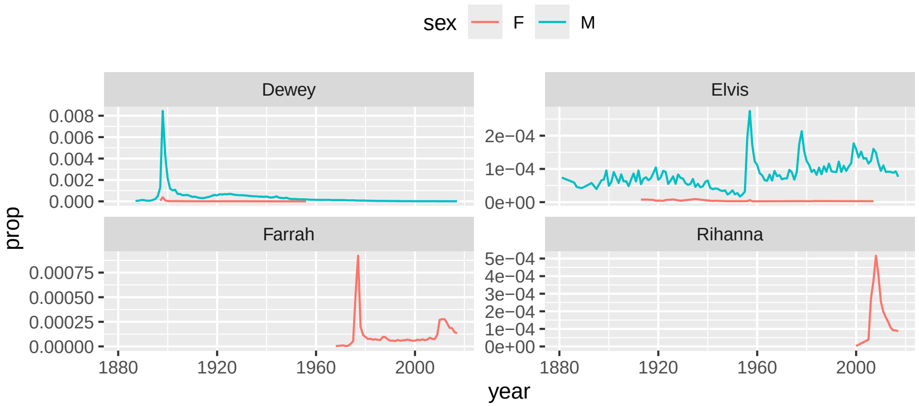
- arithmetic operators (+, −, /, ∗, etc)
- logical operators (<, <=, >=, ==, `!=`)
- logarithmic and exponential transformations (`log`, `log10`, `exp`)
- offsets (`lead`, `lag`)
- cumulative rolling aggregates (`cumsum`, `cumprod`, `cummin`, `cummax`)
- ranking (`min_rank`, `percent_rank`)

Other useful `dplyr` functions

- `rename()` - change the name of variables
- `count()` - shortcut for `group_by() %>% summarise(n = n())`
- `distinct()` - 1 row for each variable value
- `sample_n()`, `sample_frac()` - random sample of tibble
- `pull()` - extract a single column as a vector
- `slice()` - subset rows using their position

Names that "flash in the pan"

What names experience a dramatic rise/fall in popularity?



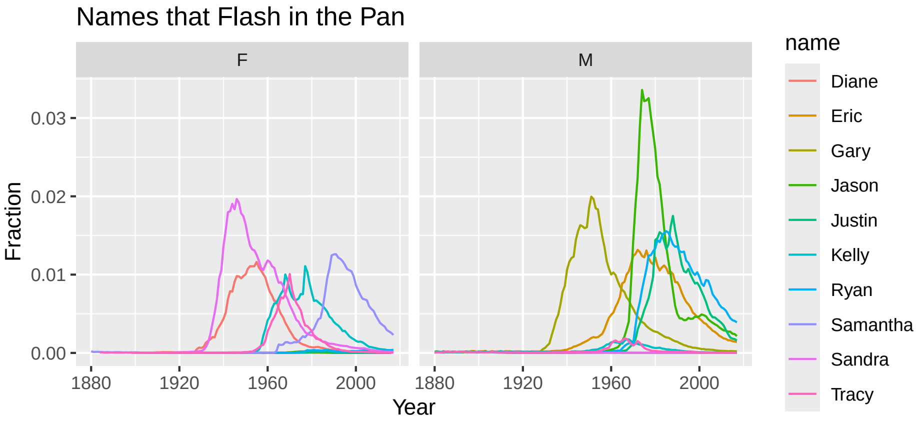
Your turn

Step 1: find the names that experience a collapse in popularity

1. Filter to names that have at least 1% babies with that name (for a sex)
2. For each name calculate the percent change in `prop` over time
3. Take the top 10 most dramatic collapses over time

Step 2: plot the popularity over time for those names

Names that “flash in the pan”

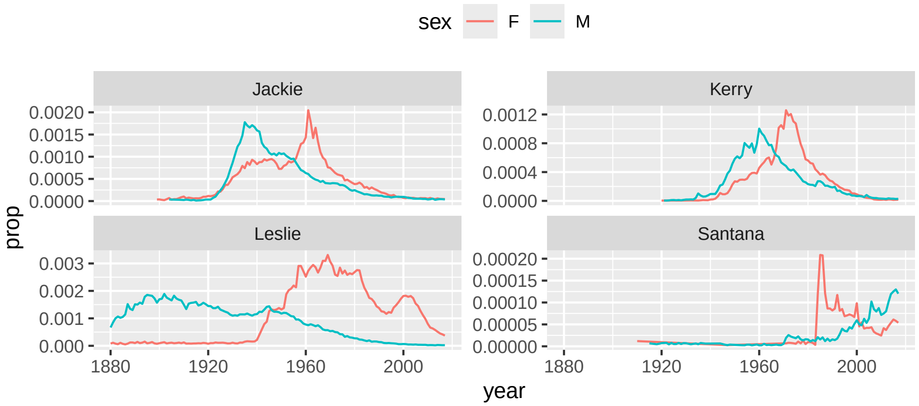


Names that “flash in the pan”

```r
babynames_flash <-
  babynames %>%
  group_by(name, sex) %>%
  mutate(max_prop = max(prop)) %>%
  filter(max_prop > 0.01) %>%
  mutate(change_in_pop = (prop/lag(prop, 1) - 1)) %>%
  summarise(min_change = min(change_in_pop, na.rm = T)) %>%
  group_by(sex) %>%
  arrange(min_change) %>%
  slice(1:5) %>%
  pull(name)
```

Names that "switch" - optional homework

Challenge: Come up with a way to identify names that "switch"



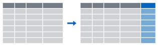
dplyr recap

Extract variables with **select()**

Extract cases with **filter()**

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

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


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