



NYC DATA SCIENCE
ACADEMY

Data Science with R (Data Analytics)

Manipulating Data with 'dplyr'

Outline

1. Baby names & baby births datasets
2. Background of **dplyr**
3. Basic functions in **dplyr**
 - Subset, transform, and reorder datasets
 - filter
 - select
 - arrange
 - mutate
 - Join datasets
 - Groupwise operations on datasets
 - group_by
 - summary

Baby names data

Baby Names & Baby Births Datasets

Load the data

- Make sure the data sets are in your working directory.
- `stringsAsFactors = FALSE`.
 - Prevent R from reading in strings as factors (the default).

`#Remember to set your working directory.`

```
bnames = read.csv("data/bnames.csv.bz2", stringsAsFactors = FALSE)
births = read.csv("data/births.csv", stringsAsFactors = FALSE)
```

Baby Names Data

```
head(bnames,5)
```

	year	name	prop	sex	soundex
1	1880	John	0.0815	boy	J500
2	1880	William	0.0805	boy	W450
3	1880	James	0.0501	boy	J520
4	1880	Charles	0.0452	boy	C642
5	1880	George	0.0433	boy	G620

```
tail(bnames,5)
```

	year	name	prop	sex	soundex
257996	2008	Carleigh	0.000128	girl	C642
257997	2008	Iyana	0.000128	girl	I500
257998	2008	Kenley	0.000127	girl	K540
257999	2008	Sloane	0.000127	girl	S450
258000	2008	Elianna	0.000127	girl	E450

Baby names data

Example

- Use logical subsetting to extract your name from the dataset. Plot the trend over time.
- What geom should you use?
- Do you need any extra aesthetics?

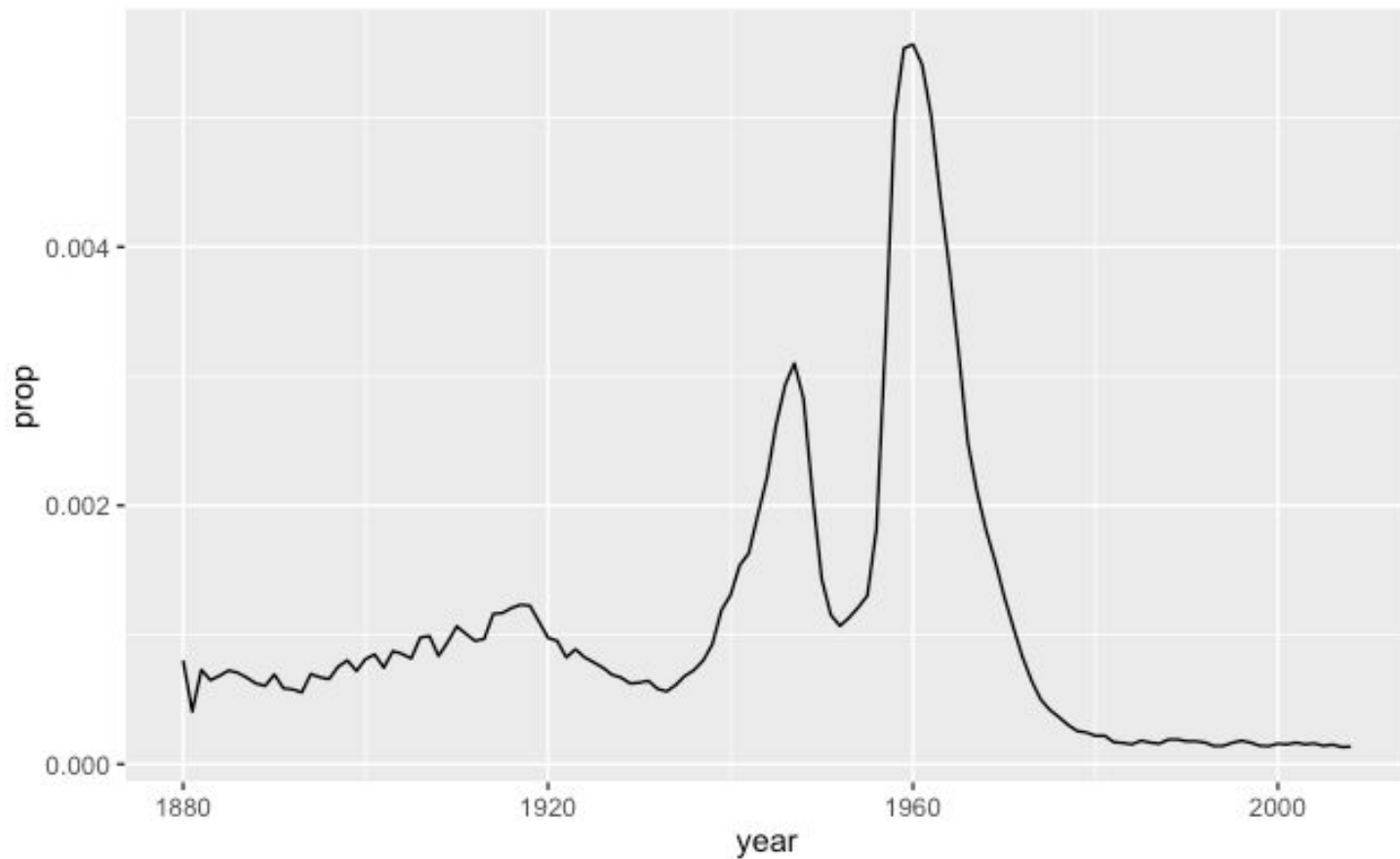
Baby names data

Answer

```
mike <- bnames[bnames$name == "Mike", ]  
qplot(year, prop, data = mike, geom = "line")
```

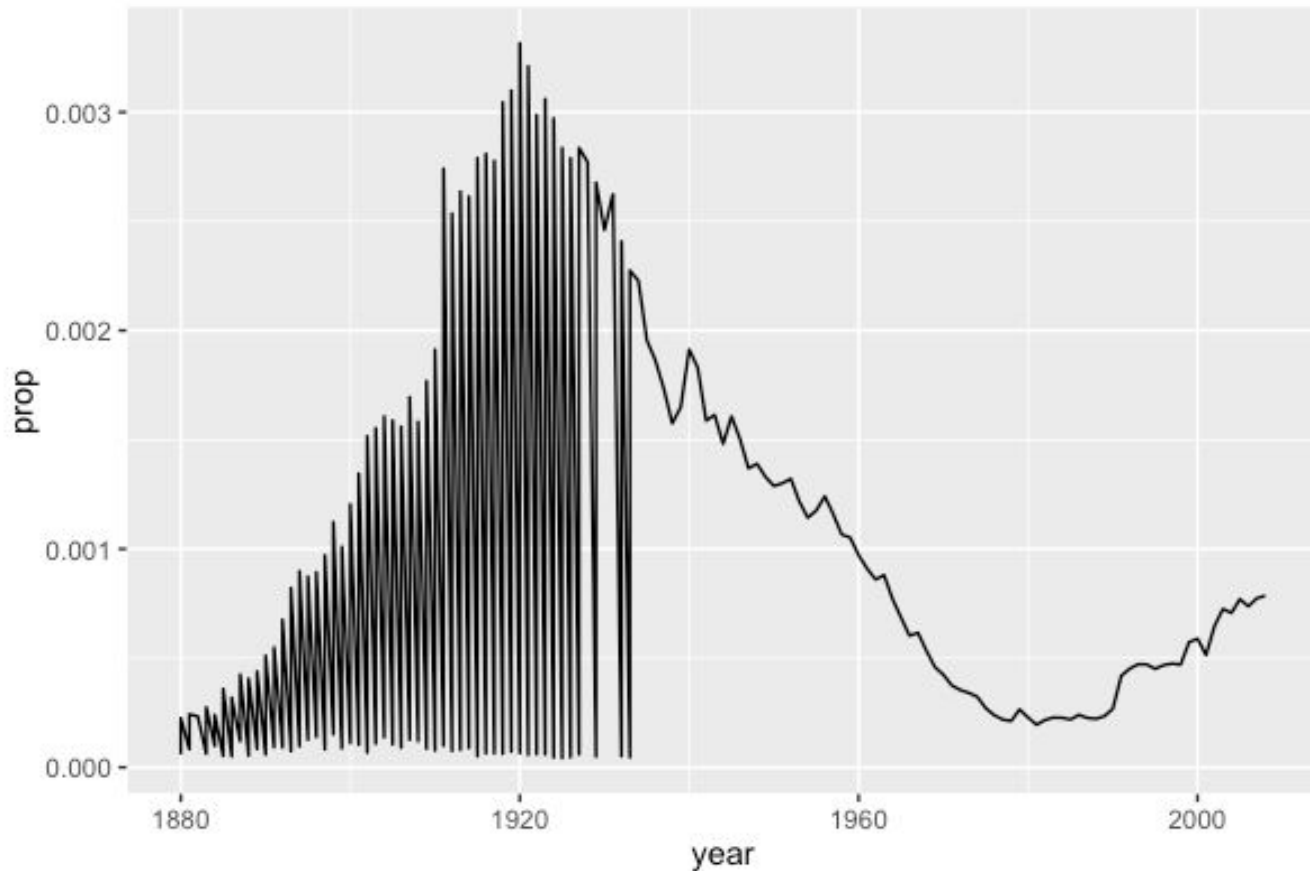
Baby names data

```
mike <- bnames[bnames$name == "Mike", ]  
qplot(year, prop, data = mike, geom = "line")
```



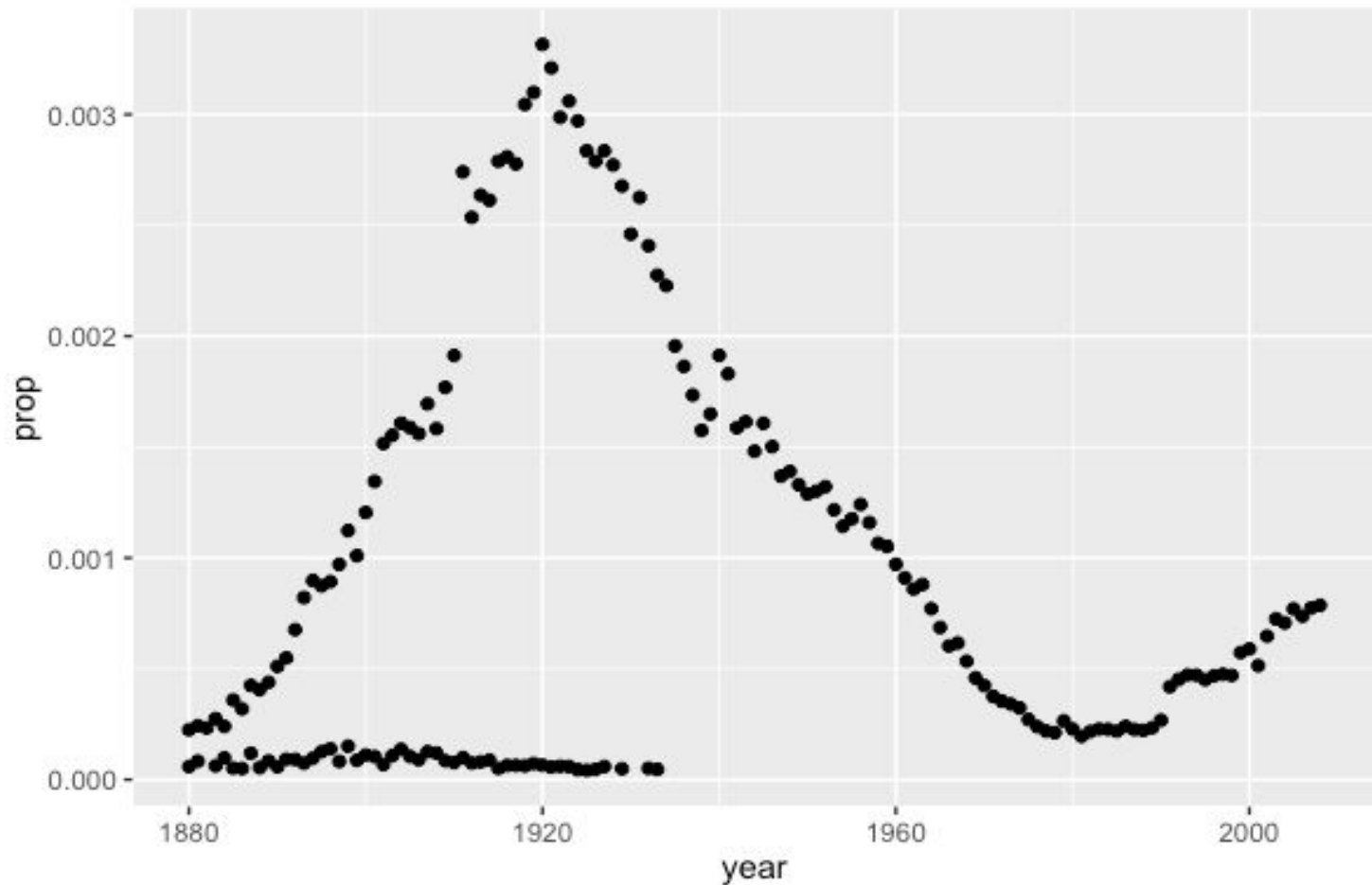
What's happening?

```
vivian <- bnames[bnames$name == "Vivian", ]  
qplot(year, prop, data = vivian, geom = "line")
```



What's happening?

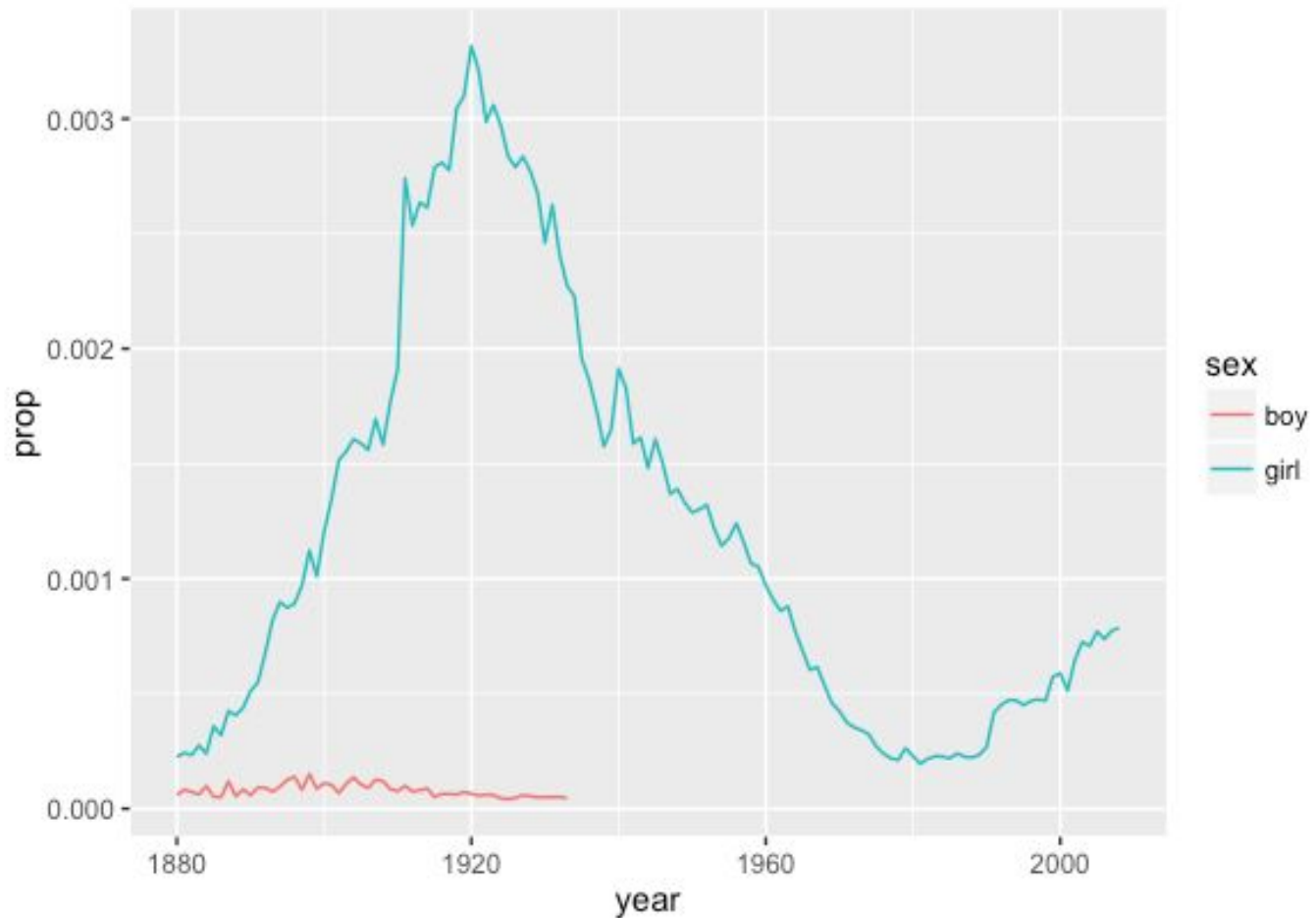
```
qplot(year, prop, data = vivian, geom = "point")
```



color = sex

- creates a different colored line for each sex

```
qplot(year, prop, data = vivian, geom = "line", color = sex)
```



Two names

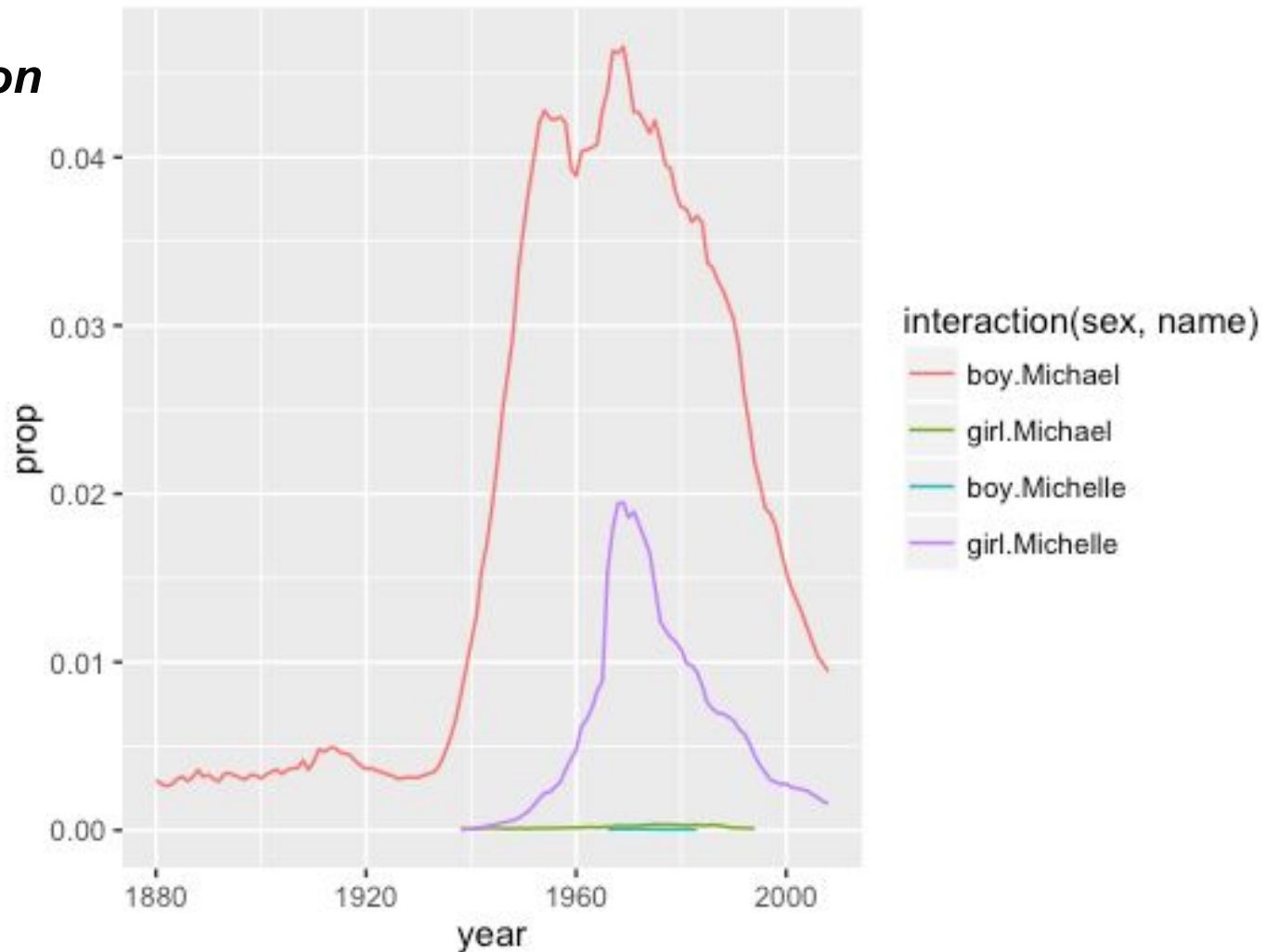
Use *interaction*

- `interaction(sex, name)`
 - use interaction to group on the combination of two variables.

```
michaels <- bnames[bnames$name == "Michael" |  
  bnames$name == "Michelle", ]  
qplot(year, prop, data = michaels, geom = "line",  
  color = interaction(sex, name))
```

Two names

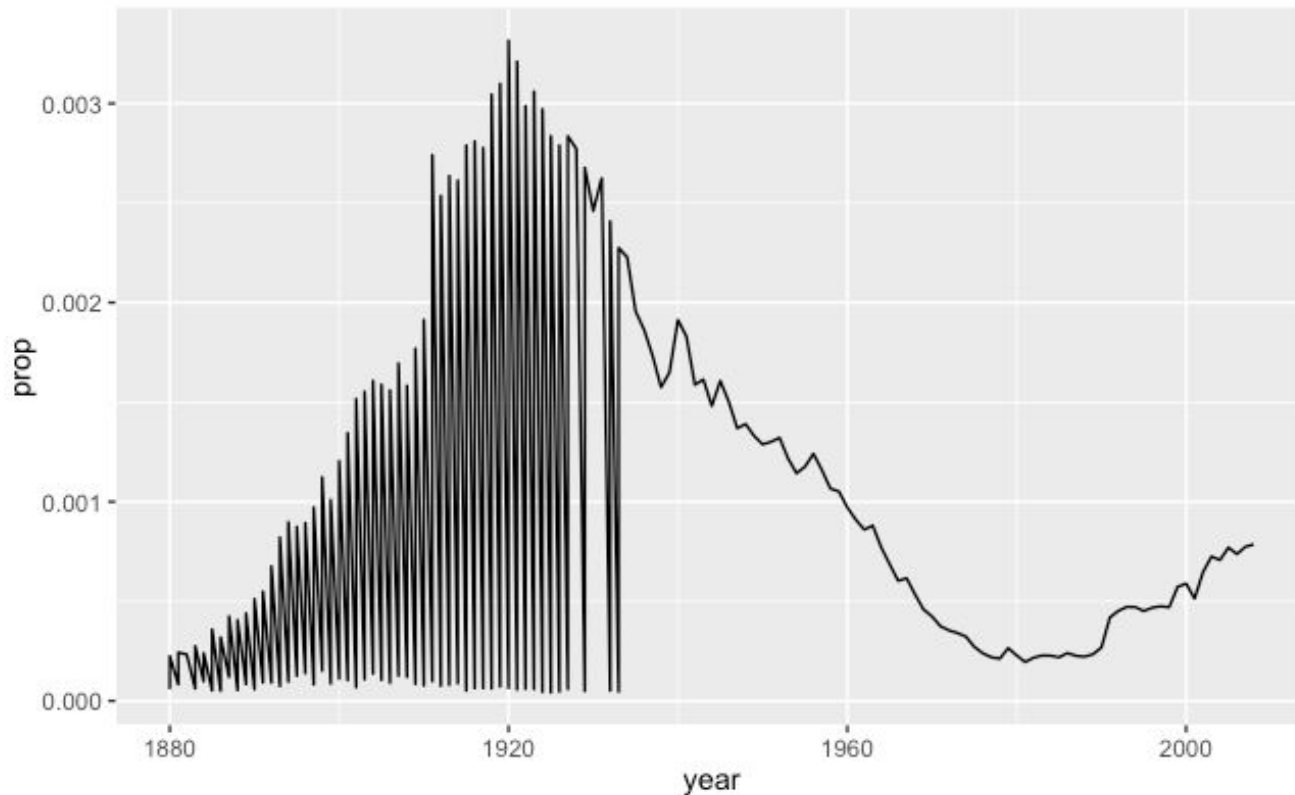
Use *interaction*



Back to the plot

Saw tooth appearance implies grouping is incorrect.

```
vivian <- bnames[bnames$name == "Vivian", ]  
qplot(year, prop, data = vivian, geom = "line")
```



dplyr

dplyr

A Grammar of Data Manipulation.

A fast, consistent tool for working with
dataframe-like objects, both in memory
and out of memory.

An R package to manipulate data!

- Easier!
- Faster!!

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

When working with data you must:

- Figure out what you want to do.
- Precisely describe what you want in the form of a computer program.
- Execute the code.

The **dplyr** package makes each of these steps faster and easier by:

- Elucidating the most common data manipulation operations.
 - Options are helpfully constrained when thinking about how to tackle a problem.
- Providing simple functions that correspond to the most common data manipulation verbs.
 - Easily translate your thoughts into code.
- Using efficient data storage
 - Spend as **little time** as possible waiting for the computer.

tbl_df

- **tbl** is a special case of dataframe that can be manipulated more easily.
- use **tbl_df** function

```
bnames = tbl_df(bnames)
births = tbl_df(births)
class(bnames)
```

```
[1] "tbl_df"      "tbl"        "data.frame"
```

R will show only the part of the tbl that fits the console.

```
Console ~/Dropbox (RStudio)/RStudio/rstudio-training/in-person-intro/Two-
> tbl_df(diamonds)
Source: local data frame [53,940 x 10]

  carat    cut color clarity depth table price
1  0.23   Ideal    E    SI2   61.5    55   326
2  0.21  Premium    E    SI1   59.8    61   326
3  0.23    Good    E    VS1   56.9    65   327
4  0.29  Premium    I    VS2   62.4    58   334
5  0.31    Good    J    SI2   63.3    58   335
6  0.24 Very Good    J   VVS2   62.8    57   336
7  0.24 Very Good    I   VVS1   62.3    57   336
8  0.26 Very Good    H    SI1   61.9    55   337
9  0.22    Fair    E    VS2   65.1    61   337
10 0.23 Very Good    H    VS1   59.4    61   338
.. ...
Variables not shown: x (dbl), y (dbl), z (dbl)
>
```

```
Console ~/Dropbox (RStudio)/RStudio/rstudio-training/
> tbl_df(diamonds)
Source: local data frame [53,940 x 10]

  carat    cut color clarity
1  0.23   Ideal    E    SI2
2  0.21  Premium    E    SI1
3  0.23    Good    E    VS1
4  0.29  Premium    I    VS2
5  0.31    Good    J    SI2
6  0.24 Very Good    J   VVS2
7  0.24 Very Good    I   VVS1
8  0.26 Very Good    H    SI1
9  0.22    Fair    E    VS2
10 0.23 Very Good    H    VS1
.. ...
Variables not shown: depth (dbl),
  table (dbl), price (int), x
  (dbl), y (dbl), z (dbl)
>
```

Use View() to see more

Subset, Transform and Reorder

Subset, Transform and Reorder

1. filter
2. select
3. arrange
4. mutate
5. summarise

Data Manipulation

Structure

- The first argument for these functions is a data frame.
- Subsequent arguments say what to do with that data frame.
- The functions always return a data frame.

Data Manipulation

Structure

- Initialize the data:

```
df = data.frame(  
  color = c("blue", "black", "blue", "blue", "black"),  
  value = 1:5)  
tbl = tbl_df(df)
```

Filter

filter

```
filter(tbl, color == "blue")
```

Source: local data frame [3 x 2]

	color	value
1	blue	1
2	blue	3
3	blue	4

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5



color	value
blue	1
blue	3
blue	4

filter

```
filter(df, value %in% c(1, 4))
```

	color	value
1	blue	1
2	blue	4

df

color	value
blue	1
black	2
blue	3
blue	4
black	5



color	value
blue	1
blue	4

filter

Example

- In the bnames dataset:
 - a. Find all of the names that are in the **same soundex** as the name “Vivian”.
 - b. Find all of the girls born in **1900** or **2000**.
 - c. How many times did a name reach a prop **greater than 0.01 after the year 2000?**

filter

Example

- Find all of the names that are in the **same soundex** as the name “Vivian”.

```
vivian = filter(bnames, name == "Vivian")  
vivian$soundex[1]
```

```
[1] "V150"
```

```
filter(bnames, soundex == "V150")
```

Source: local data frame [251 x 5]

	year	name	prop	sex	soundex
1	1880	Vivian	0.000059	boy	V150
2	1881	Vivian	0.000083	boy	V150
3	1883	Vivian	0.000062	boy	V150

...

filter

Example

- Find all of the girls born in 1900 or 2000.

```
filter(bnames, sex == "girl" & (year == 1900 | year == 2000))
```

Source: local data frame [2,000 x 5]

	year	name	prop	sex	soundex
1	1900	Mary	0.0526	girl	M600
2	1900	Helen	0.0200	girl	H450
3	1900	Anna	0.0192	girl	A500
4	1900	Margaret	0.0167	girl	M626
5	1900	Ruth	0.0150	girl	R300
6	1900	Elizabeth	0.0129	girl	E421
7	1900	Florence	0.0123	girl	F465
8	1900	Ethel	0.0123	girl	E340
9	1900	Marie	0.0121	girl	M600
10	1900	Lillian	0.0107	girl	L450

...

filter

Example

- How many times did a name reach a prop greater than 0.01 after the year 2000?

```
filter(bnames, year > 2000 & prop > 0.01)
```

Source: local data frame [57 x 5]

	year	name	prop	sex	soundex
1	2001	Jacob	0.0157	boy	J210
2	2001	Michael	0.0144	boy	M240
3	2001	Matthew	0.0130	boy	M300
4	2001	Joshua	0.0126	boy	J200
5	2001	Christopher	0.0112	boy	C623
6	2001	Nicholas	0.0111	boy	N242

select

select

Use the select statement to select a subset of the overall data.

What do you think this statement will do?

```
select(tbl, color)
```

select

```
select(tbl, color)
```

Source: local data frame [5 x 1]

color

1 blue
2 black
3 blue
4 blue
5 black

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5



color
blue
black
blue
blue
black

select

```
select(tbl, -color)
```

Source: local data frame [5 x 1]

value

1	1
2	2
3	3
4	4
5	5

df

color	value
blue	1
black	2
blue	3
blue	4
black	5

→

value
1
2
3
4
5

select

Example

- Let's try bringing up the help documentation for select.

```
help(select)
```

- Scroll down to the “Special Functions” section:
 - a. What are some other ways you can select variables?

select

- `starts_with(x, ignore.case = TRUE)`: names starting with x.
- `ends_with(x, ignore.case = TRUE)`: names ending in x.
- `contains(x, ignore.case = TRUE)`: selects all variables whose name contains x.
- `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression x.
- `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
- `one_of("x", "y", "z")`: selects variables provided in a character vector.
- `everything()`: selects all variables.

select

Example

The following statements all select the soundex variable from the baby names dataset; one of them selects more than just the soundex variable.

```
select(bnames, soundex)  
select(bnames, starts_with("sound"))  
select(bnames, ends_with("ex"))
```

Difference Between *filter* and *select*

- **filter:**
 - Keep rows by criteria.
- **select:**
 - Pick columns by name.

rename

- **select** can also rename the variables in the resulting dataset:
- **select(iris, petal_length = Petal.Length)**
- **select()** keeps only the variables you specify.

```
head(  
  select(iris, petal_length = Petal.Length)  
)
```

	petal_length
1	1.4
2	1.4
3	1.3
4	1.5
5	1.4
6	1.7

rename

- **rename** function is similar to the **select** function.
- **rename(iris, petal_length = Petal.Length)**.
- **rename()** keeps all variables.

```
head(  
  rename(iris, petal_length = Petal.Length)  
)
```

	Sepal.Length	Sepal.Width	petal_length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

arrange

arrange

```
df1 = data.frame(color = c(4,1,5,3,2),  
                  value = 1:5)  
arrange(df1, color)
```

color	value
4	1
1	2
5	3
3	4
2	5

color	value
1	2
2	5
3	4
4	1
5	3

Original Method

order function:

```
df_order = order(df1$color) #Returns the indices for ascending  
order.
```

```
df1[df_order,]
```

	color	value
2	1	2
5	2	5
4	3	4
1	4	1
3	5	3

arrange

```
arrange(df1, desc(color)) #Arranging the data in descending order.
```

color	value
4	1
1	2
5	3
3	4
2	5

color	value
5	3
4	1
3	4
2	5
1	2

arrange

Example

- Reorder the baby names dataset by `prop` in descending order.

```
arrange(bnames, desc(prop))[3,]
```

Source: local data frame [3 x 5]

	year	name	prop	sex	soundex
1	1880	John	0.0815	boy	J500
2	1881	John	0.0809	boy	J500
3	1880	William	0.0805	boy	W450

arrange

Example

- In what year was Vivian's name the most popular?
 - (Hint: First filter the data by the name "Vivian".)

```
arrange(filter(bnames, name == "Vivian"), desc(prop))[1,]
```

Source: **local data frame [1 x 5]**

	year	name	prop	sex	soundex
1	1920	Vivian	0.00332	girl	V150

mutate

mutate

We can also add columns to datasets by manipulating existing variables.

```
mutate(tbl, double = 2 * value)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double
blue	1	2
black	2	4
blue	3	6
blue	4	8
black	5	10

mutate

```
mutate(tbl, double = 2 * value, quadruple = 4 * value)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double	quadruple
blue	1	2	4
black	2	4	8
blue	3	6	12
blue	4	8	16
black	5	10	20

transmute

- A function which is similar to **mutate**.
- Drops old variables and only retains the newly defined variables.

```
transmute(tbl, double = 2 * value, quadruple = 4 * value)
```

Source: local data frame [5 x 2]

	double	quadruple
1	2	4
2	4	8
3	6	12
4	8	16
5	10	20

summarise

Aggregate Functions

Use `summarise()` with aggregate functions, which take a vector of values, and return a single number.

In base R : `min()`, `max()`, `mean()`, `sum()`, `sd()`, `median()`, `IQR()`.

`dplyr` provides a handful of others:

- `n()`: number of observations in the current group.
- `n_distinct(x)`: count the number of unique values in x.
- `first(x)`, `last(x)` and `nth(x, n)` get the first, last and the nth x.

summarise

```
summarise(tbl, total = sum(value))
```

Source: **local data frame** [1 x 1]

	total
1	15

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5



total
15

summarise

```
summarise(tbl, total = sum(value), avg = mean(value))
```

Source: local data frame [1 x 2]

	total	avg
1	15	3

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5



total	avg
15	3

Operations to Change Datasets

- **mutate:**
 - Add new variables
- **summarise:**
 - Reduce variables to values
- **arrange:**
 - Reorder rows

Example

- With the vivian data frame:
 - a. Add a new column to the data that changes the prop to a percentage.
 - b. Create a summary that displays the **min, mean, and max prop** Vivian's name.

Answer

A. Add a new column to the data that changes the prop to a percentage.

```
head(  
  mutate(vivian, perc = prop * 100)  
)
```

Source: local data frame [6 x 6]

	year	name	prop	sex	souindex	perc
1	1880	Vivian	5.9e-05	boy	V150	0.0059
2	1881	Vivian	8.3e-05	boy	V150	0.0083
3	1883	Vivian	6.2e-05	boy	V150	0.0062
4	1884	Vivian	9.8e-05	boy	V150	0.0098
5	1885	Vivian	5.2e-05	boy	V150	0.0052
6	1886	Vivian	5.0e-05	boy	V150	0.0050

Answer

- B. Create a summary that displays the **min**, **mean**, and **max prop** for Vivian's name.

```
summarise(vivian,  
  min = min(prop),  
  mean = mean(prop),  
  max = max(prop))
```

Source: **local data frame** [1 x 3]

	min	mean	max
1	4.2e-05	0.000888	0.00332

What Do These Functions Do Again?

1. `filter`
2. `select`
3. `arrange`
4. `mutate`
5. `summarise`

What Do These Functions Do Again?

1. **filter**: keep rows matching given criteria.
2. **select**: pick columns by name.
3. **arrange**: reorder rows.
4. **mutate**: add new variables. (Use **transmute** to drop old variables.)
5. **summarise**: reduce variables to values.

Joining Data Sets

Joining Data Sets

Example

- Why might `prop` be a bad way to compare names across different years?

births

Source: local data frame [260 x 3]

	year	sex	births
1	1880	boy	118405
2	1881	boy	108290
3	1882	boy	122034
4	1883	boy	112487
5	1884	boy	122745
6	1885	boy	115948
7	1886	boy	119046
8	1887	boy	109312
9	1888	boy	129914
10	1889	boy	119044
...

Combine Two Data Sets

- How would you combine these data sets?
- Describe a strategy.

head(bnames)

Source: local data frame [6 x 5]

	year	name	prop	sex	
soundex					
1	1880	John	0.0815	boy	J500
2	1880	William	0.0805	boy	W450
3	1880	James	0.0501	boy	J520
4	1880	Charles	0.0452	boy	C642
5	1880	George	0.0433	boy	G620
6	1880	Frank	0.0274	boy	F652

head(births)

Source: local data frame [6 x 3]

	year	sex	births
1	1880	boy	118405
2	1881	boy	108290
3	1882	boy	122034
4	1883	boy	112487
5	1884	boy	122745
6	1885	boy	115948

Combine Two Data Sets

- How would you combine these data sets?
- Describe a strategy.

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

Combine Two Data Sets

Initialize the demo data

```
x = data.frame(  
  name = c("John", "Paul", "George", "Ringo", "Stuart", "Pete"),  
  instrument = c("guitar", "bass", "guitar", "drums", "bass",  
                 "drums"))  
y = data.frame(  
  name = c("John", "Paul", "George", "Ringo", "Brian"),  
  band = c("TRUE", "TRUE", "TRUE", "TRUE", "FALSE"))
```

Combine Two Data Sets

Type	Action
left_join	Include all of x, and matching rows of y
inner_join	Include rows of x that appear in y, and matching rows of y
semi_join	Include rows of x that appear in y
anti_join	Include rows of x that <i>do not</i> appear in y

Combine Two Data Sets

left_join

- Include all of x, and matching rows of y.

```
left_join(x, y, by = "name") #To which column is the error referring?
```

x			y					
name	instrument		name	band		name	instrument	band
John	guitar		John	T		John	guitar	T
Paul	bass		Paul	T		Paul	bass	T
George	guitar	+	George	T	=	George	guitar	T
Ringo	drums		Ringo	T		Ringo	drums	T
Stuart	bass		Brian	F		Stuart	bass	NA
Pete	drums					Pete	drums	NA

```
left_join(x, y, by = "name")
```

Combine Two Data Sets

inner_join

- Include rows of x that appear in y, **and matching rows of y**

```
inner_join(x, y, by = "name")
```

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

name	instrument	band
John	guitar	T
Paul	bass	T
George	guitar	T
Ringo	drums	T

Combine Two Data Sets

semi_join

- Include rows of x that appear in y

```
semi_join(x, y, by = "name")
```

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums

Combine Two Data Sets

anti_join

- Include rows of x that **do not** appear in y

```
anti_join(x, y, by = "name")
```

name	instrument
John	guitar
Paul	bass
George	guitar
Ringo	drums
Stuart	bass
Pete	drums

name	band
John	T
Paul	T
George	T
Ringo	T
Brian	F

name	instrument
Stuart	bass
Pete	drums

Combine Two Data Sets

Example

- Combine bnames with births.
- Add a new column that shows the number of babies whose name is the corresponding **name** and born in the corresponding **year**.

For example:

	year	sex	name	prop	soundex	births	n
1	1880	boy	John	0.081541	J500	118405	9655

There are 9655 babies born in 1880 and named John.

Combine Two Data Sets

Answer

- First combine bnames with births:

```
bnames2 = left_join(bnames, births, by = c("year", "sex"))  
bnames2
```

Source: local data frame [258,000 x 6]

	year	sex	name	prop	soundex	births
1	1880	boy	John	0.081541	J500	118405
2	1880	boy	William	0.080511	W450	118405
3	1880	boy	James	0.050057	J520	118405
4	1880	boy	Charles	0.045167	C642	118405
5	1880	boy	George	0.043292	G620	118405
...						

Combine Two Data Sets

Answer

- Then create a new column that shows the total number of babies born each year for each name.

```
bnames2 = mutate(bnames2, n = round(prop * births))  
bnames2
```

Source: local data frame [258,000 x 7]

	year	sex	name	prop	soundex	births	n
1	1880	boy	John	0.081541	J500	118405	9655
2	1880	boy	William	0.080511	W450	118405	9533
3	1880	boy	James	0.050057	J520	118405	5927
4	1880	boy	Charles	0.045167	C642	118405	5348
5	1880	boy	George	0.043292	G620	118405	5126
...							

Groupwise Operations

Groupwise Operations

- Total number of people per name.
- Do we have enough information to calculate this?

	name	total
1	Aaden	959
2	Aaliyah	39665
3	Aarav	219
4	Aaron	509464
5	Ab	25
6	Abigail	2682
7	Abb	16
8	Abbey	14348
9	Abbie	16622
10	Abbigail	6800

Groupwise Operations

Example

- Start small. Using the `bnames2` dataset, calculate the total for “Vivian”.

Groupwise Operations

Answer

- Using the bnames2 dataset, calculate the total for “Vivian”.

```
vivian = filter(bnames2, name == "Vivian")  
sum(vivian$n)
```

```
[1] 183011
```

```
#We could have also used the summarise function.  
summarise(vivian, total = sum(n))
```

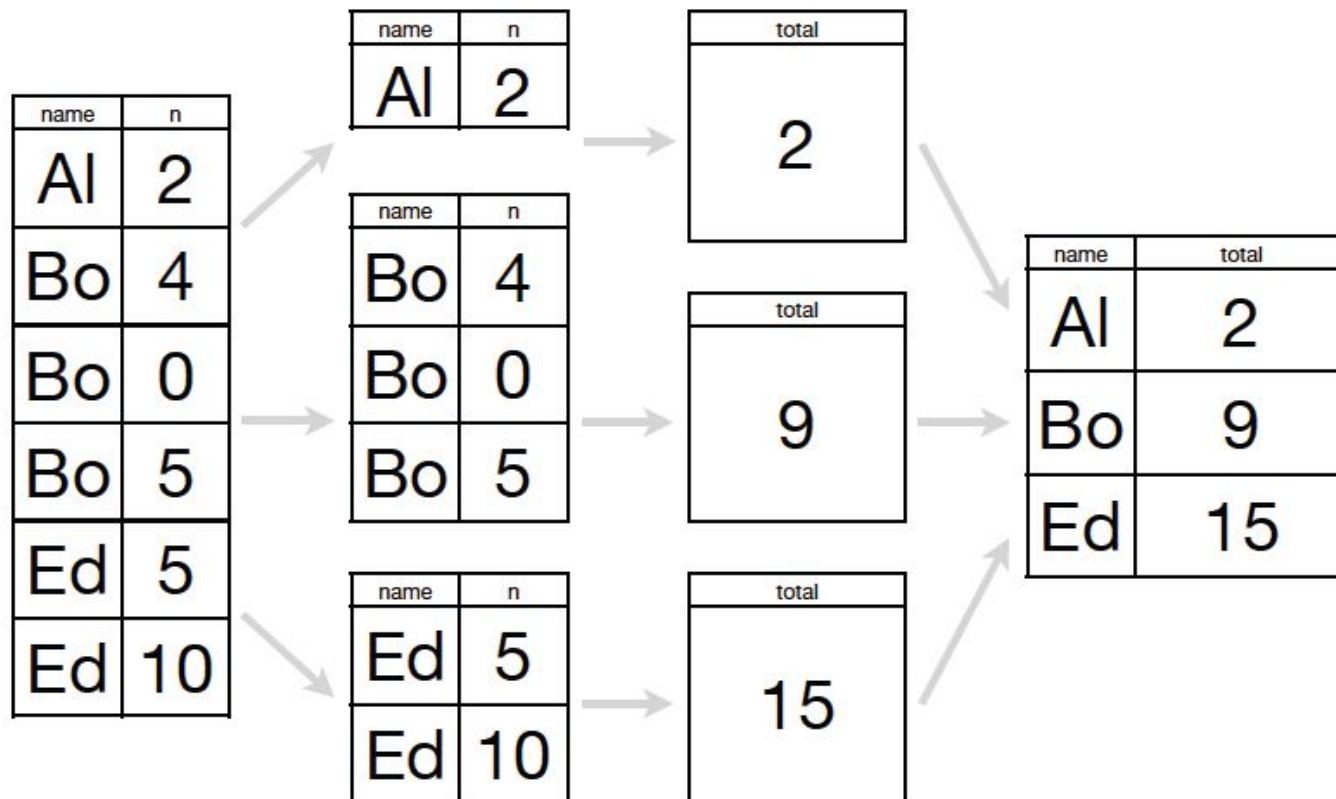
```
Source: local data frame [1 x 1]
```

```
  total  
1 183011
```

Groupwise Operations

Question

- What if we wanted to do this for every name in the dataset? Uh oh...



group_by

group_by

```
summarise(tbl, total = sum(value))
```

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5

→

total
15

group_by

```
by_color = group_by(tbl, color) #Has a new grouped table component.
```

```
summarise(by_color, total = sum(value))
```

tbl

color	value
blue	1
black	2
blue	3
blue	4
black	5



color	total
blue	8
black	7

group_by

```
group_by(bnames2, name) #This looks very similar to bnames2; what's different?
```

Source: local data frame [258,000 x 7]

Groups: name

	year	name	prop	sex	soundex	births	n
1	1880	John	0.0815	boy	J500	118405	9655
2	1880	William	0.0805	boy	W450	118405	9533
3	1880	James	0.0501	boy	J520	118405	5927
4	1880	Charles	0.0452	boy	C642	118405	5348
5	1880	George	0.0433	boy	G620	118405	5126
..

- Groups: name
- mutate, summarise, and arrange will execute groupwise on this variable.

group_by

```
by_name = group_by(bnames2, name)
totals = summarise(by_name, total = sum(n))
head(totals)
```

Source: **local data frame** [6 x 2]

	name	total
1	Aaden	959
2	Aaliyah	39665
3	Aarav	219
4	Aaron	509464
5	Ab	25
6	Abagail	2682

That was so much easier!

Interactions

Use **multiple** variables to create groups based on the interaction of variables.

```
group_by(bnames2, name, sex)
```

Source:	local	data	frame	[6	x	7]	
Groups:			name,			sex	
	year	name	prop	sex	soundex	births	n
1	1880	John	0.0815	boy	J500	118405	9655
2	1880	William	0.0805	boy	W450	118405	9533
3	1880	James	0.0501	boy	J520	118405	5927

- **Groups: name, sex**
- **dplyr** will treat each unique combination of these values as a separate group.

Interactions

Successive `group_by` calls will forget the old groups and replace them by the new.

```
by_name = group_by(bnames2, name)
group_by(by_name, sex) #Doesn't yield what we really wanted...
```

Source: local data frame [258,000 x 7]

Groups: sex

	year	sex	name	prop	soundex	births	n
1	1880	boy	John	0.081541	J500	118405	9655
2	1880	boy	William	0.080511	W450	118405	9533
3	1880	boy	James	0.050057	J520	118405	5927
4	1880	boy	Charles	0.045167	C642	118405	5348
5	1880	boy	George	0.043292	G620	118405	5126
6	1880	boy	Frank	0.027380	F652	118405	3242

...

Interactions

Use **summarise** to solve this problem of trying to use multiple **group_by** statements.

```
name_sex = group_by(bnames2, name, sex)
totals2 = summarise(name_sex, total = sum(n))
head(totals2)
```

Source: **local data frame** [6 x 3]

Groups: **name**

	name	sex	total
1	Aaden	boy	959
2	Aaliyah	girl	39665
3	Aarav	boy	219
4	Aaron	boy	508094
5	Aaron	girl	1370
6	Ab	boy	25

Ungroup

Use ungroup to remove group specifications.

```
by_name_sex = group_by(bnames2, name, sex)
ungroup(by_name_sex)
```

Source: local data frame [258,000 x 7]

	year	sex	name	prop	soundex	births	n
1	1880	boy	John	0.081541	J500	118405	9655
2	1880	boy	William	0.080511	W450	118405	9533
3	1880	boy	James	0.050057	J520	118405	5927
4	1880	boy	Charles	0.045167	C642	118405	5348
5	1880	boy	George	0.043292	G620	118405	5126
6	1880	boy	Frank	0.027380	F652	118405	3242

...

Examples

1. Calculate the total number of babies in each soundex.
2. What is the most popular soundex?
3. What names are in the most popular soundex?
4. Calculate the total number of boys and the total number of girls for each year.
5. Calculate the rank of each name within each year and within each gender. (Hint: Use a mutate statement with the new column `rank = rank(desc(prop))`).
6. Which names were ranked #1?
7. How many times did each of the #1 ranked names appear as the top ranked name? Order these names by the most frequent.

Answer

1. Calculate the total number of babies in each soundex.

```
by_soundex = group_by(bnames2, soundex)
totals = summarise(by_soundex, total = sum(n))
totals
```

Source: local data frame [1,392 x 2]

	soundex	total
1	A000	11
2	A100	193837
3	A120	15652
4	A124	256458
5	A130	11
6	A134	5181
7	A135	901
...		

Answer

2. What is the most popular soundex?

```
arrange(stotals, desc(total))
```

Source: local data frame [1,392 x 2]

	soundex	total
1	J500	9991737
2	M240	5823791
3	M600	5553703
4	J520	5524958
5	R163	5047182
6	W450	4116109

Answer

3. What names are in the most popular soundex?

```
j500 = filter(bnames, soundex == "J500")  
unique(j500$name)
```

```
[1] "John"    "Jim"     "Juan"    "Jimmie"  "Johnnie" "Johnny"  "Johnie"  "Jean"  
[9] "June"    "Jonah"   "Jennie"  "Jimmy"   "Johny"   "Jonnie"  "Johney"  
"Jamie"  
[17] "Jon"     "Joan"    "Jan"     "Jame"    "Jaime"   "Jamey"   "Jaimie"  
"Jammie"  
[25] "Jayme"   "Juwan"   "Johan"   "Jaheim"  "Jahiem"  "Jaheem"  "Jane"    "Janie"  
[33] "Johanna" "Joanna"  "Jannie"  "Jenny"   "Jeanne"  "Johannah" "Juana"  "Junie"  
[41] "Jinnie"  "Jeanie"  "Jeannie" "Junia"   "Janey"   "Jeane"   "Joanne"  
"Joann"  
[49] "Jayne"   "Jana"    "Janna"   "Jann"    "Joni"    "Joanie"  "Jeanna"  
"Jami"  
[57] "Johnna"  "Jeana"   "Jonna"   "Jena"    "Jenni"   "Jenna"   "Janae"  
"Jaimee"  
[65] "Janay"   "Joana"   "Janiya"  "Johana"  "Jamyia"  "Janiyah" "Janiah"  "Jamiya"
```

Answer

4. Calculate the total number of boys and the total number of girls for each year.

```
year_sex = group_by(bnames2, year, sex)
ytotals = summarise(year_sex, births = sum(n))
ytotals
```

Source: local data frame [258 x 3]

Groups: year

	year	sex	births
1	1880	boy	110207
2	1880	girl	91227
3	1881	boy	100763
4	1881	girl	92204
5	1882	boy	113194
...			

Answer

5. Calculate the rank of each name within each year and within each gender.

```
year_sex = group_by(bnames2, year, sex)
ranks = mutate(year_sex, rank = rank(desc(prop)))
ranks
```

Source: local data frame [258,000 x 8]

Groups: year, sex

	year	sex	name	prop	soundex	births	n	rank
1	1880	boy	John	0.0815	J500	118405	9655	1
2	1880	boy	William	0.0805	W450	118405	9533	2
3	1880	boy	James	0.0501	J520	118405	5927	3
4	1880	boy	Charles	0.0452	C642	118405	5348	4
5	1880	boy	George	0.0433	G620	118405	5126	5
6	1880	boy	Frank	0.0274	F652	118405	3242	6

...

Answer

6. Which names were ranked number 1?

```
ones = filter(ranks, rank == 1)
ones = select(ones, year, name, sex)
unique(ones$name)
```

```
[1] "John"    "Robert"  "James"   "Michael" "David"   "Jacob"   "Mary"
[8] "Linda"   "Lisa"    "Jennifer" "Jessica" "Ashley"  "Emily"   "Emma"
```


Answer

7. How many times did each of the #1 ranked names appear as the top ranked name?
Order these names by the most frequent.

```
arrange(summarise(group_by(ones, name), count = n()), desc(count))
```

Source: local data frame [14 x 2]

	name	count
1	Mary	76
2	John	44
3	Michael	44
4	Robert	17
5	Jennifer	15
...		

Wow...that's complicated!

Answer

Is there an easier way?

- Use the pipe function `%>%` with the period marker `.` to transmit the result of one call as an argument to the next call.

```
arrange(summarise(group_by(ones, name), count = n()), desc(count))
```

#Or...

```
group_by(ones,name) %>%  
  summarise(.,count=n()) %>%  
  arrange(.,desc(count))
```

Source: local data frame [14 x 2]

	name	count
1	Mary	76
2	John	44
3	Michael	44
4	Robert	17

Summarise

- “Unwrap” groups with summarise.
- Summarise will also remove one level of grouping from its output.

Summarise

```
bnames3 = select(bnames2,-soundex) #Dropping the soundex column.  
name_sex = group_by(bnames3, name, sex)  
name_sex
```

Source: local data frame [258,000 x 6]

Groups: name, sex

	year	name	prop	sex	births	n
1	1880	John	0.0815	boy	118405	9655
2	1880	William	0.0805	boy	118405	9533
3	1880	James	0.0501	boy	118405	5927
4	1880	Charles	0.0452	boy	118405	5348
5	1880	George	0.0433	boy	118405	5126
6	1880	Frank	0.0274	boy	118405	3242
7	1880	Joseph	0.0222	boy	118405	2632
8	1880	Thomas	0.0214	boy	118405	2534
9	1880	Henry	0.0206	boy	118405	2444
10	1880	Robert	0.0204	boy	118405	2416
..

Summarise

```
bnames3 = select(bnames2,-soundex)
name_sex = group_by(bnames3, name, sex)
summary1 = summarise(name_sex, total = sum(n))
summary1 #Summarises first by going across the sex group.
```

Source: local data frame [7,455 x 3]

Groups: name

	name	sex	total
1	Aaden	boy	959
2	Aaliyah	girl	39665
3	Aarav	boy	219
4	Aaron	boy	508094
5	Aaron	girl	1370
6	Ab	boy	25
7	Abagail	girl	2682
8	Abb	boy	16
9	Abbey	girl	14348
10	Abbie	boy	10
..

Summarise

```
bnames3 = select(bnames2,-soundex)
name_sex = group_by(bnames3, name, sex)
summary1 = summarise(name_sex, total = sum(n))
summary2 = summarise(summary1, total = sum(total))
summary2 #Summarises second by going across the name group.
```

Source: **local data frame** [6,782 x 2]

	name	total
1	Aaden	959
2	Aaliyah	39665
3	Aarav	219
4	Aaron	509464
5	Ab	25
6	Abigail	2682
7	Abb	16
8	Abbey	14348
9	Abbie	16622
10	Abbigail	6800
..

Summarise

```
bnames3 = select(bnames2,-soundex)
name_sex = group_by(bnames3, name, sex)
summary1 = summarise(name_sex, total = sum(n))
summary2 = summarise(summary1, total = sum(total))
summary3 = summarise(summary2, total = sum(total))
summary3 #Summarises by compressing the remaining information.
```

Source: local data frame [1 x 1]

	total
1	290255364

Summary

Function Reference

FUNCTION	USAGE
filter	Keep rows matching criteria
select	Pick columns by name
rename	Rename the variables and keep others
arrange	Reorder rows
mutate	Add new variables
transmute	Drops existing variables
summarise	Reduce variables to values
left_join	Include all of x, and matching rows of y.
inner_join	Include rows of x that appear in y, and matching rows of y
semi_join	Include rows of x that appear in y
anti_join	Include rows of x that do not appear in y
group_by	Groupwise operations
%>%	Pipe function

Vignettes

Read dplyr's built in vignettes to learn more

```
browseVignettes(package = "dplyr")
```

Vignettes in package dplyr

- Adding new database support to dplyr - [HTML](#) [source](#) [R code](#)
- Databases - [HTML](#) [source](#) [R code](#)
- Hybrid Evaluation - [HTML](#) [source](#) [R code](#)
- Introduction to dplyr - [HTML](#) [source](#) [R code](#)
- Memory usage - [HTML](#) [source](#) [R code](#)
- Non-standard evaluation - [HTML](#) [source](#) [R code](#)
- Window functions - [HTML](#) [source](#) [R code](#)