# Transform Data with

dplyr

www.rstudio.com

# babynames


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

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

```
babynames
```

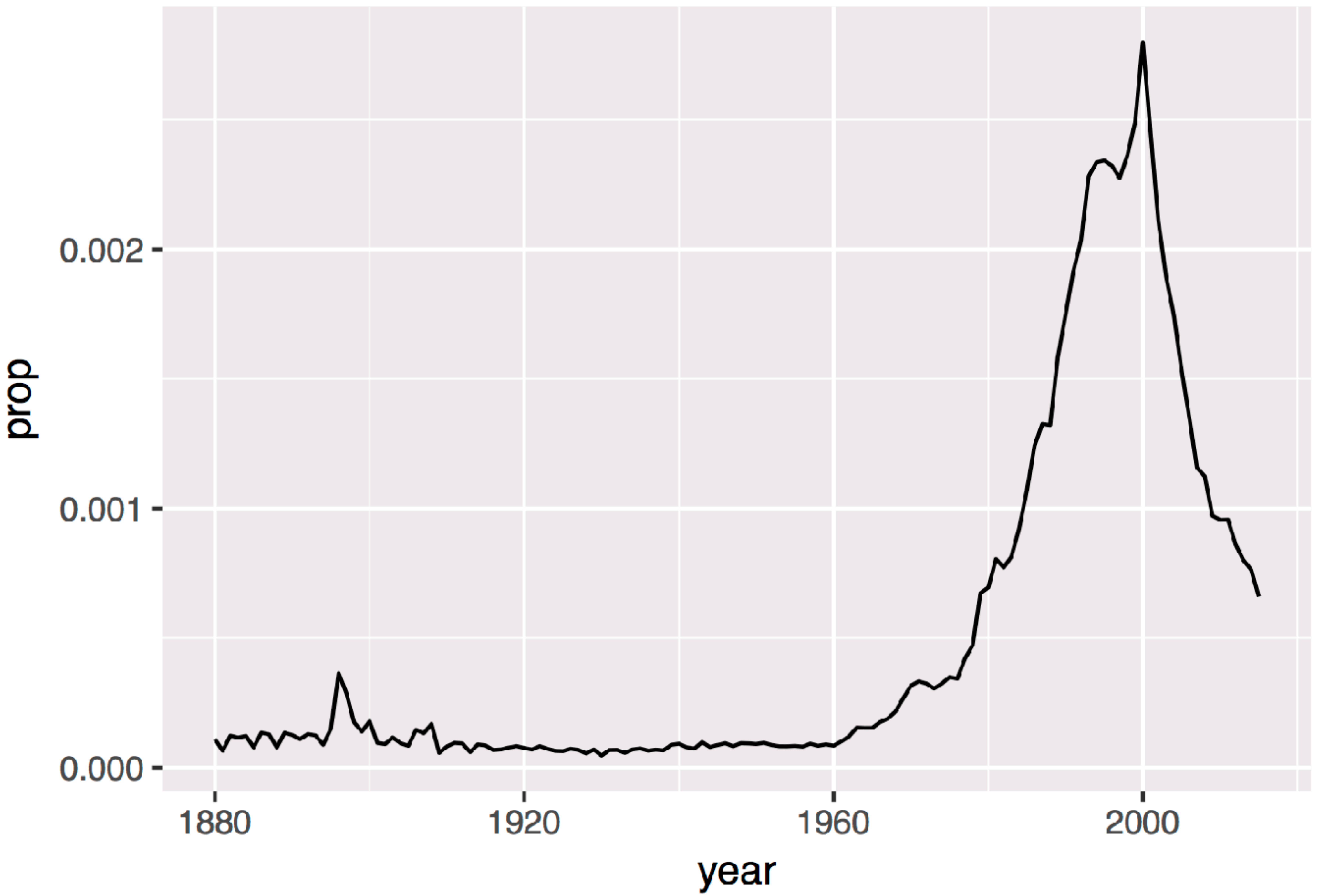
| year <dbl> | sex <chr> | name <chr> | n <int> | prop <dbl> |
|---|---|---|---|---|
| 1880 | F | Mary | 7065 | 7.238433e-02 |
| 1880 | F | Anna | 2604 | 2.667923e-02 |
| 1880 | F | Emma | 2003 | 2.052170e-02 |
| 1880 | F | Elizabeth | 1939 | 1.986599e-02 |
| 1880 | F | Minnie | 1746 | 1.788861e-02 |
| 1880 | F | Margaret | 1578 | 1.616737e-02 |
| 1880 | F | Ida | 1472 | 1.508135e-02 |
| 1880 | F | Alice | 1414 | 1.448711e-02 |
| 1880 | F | Bertha | 1320 | 1.352404e-02 |
| 1880 | F | Sarah | 1288 | 1.319618e-02 |

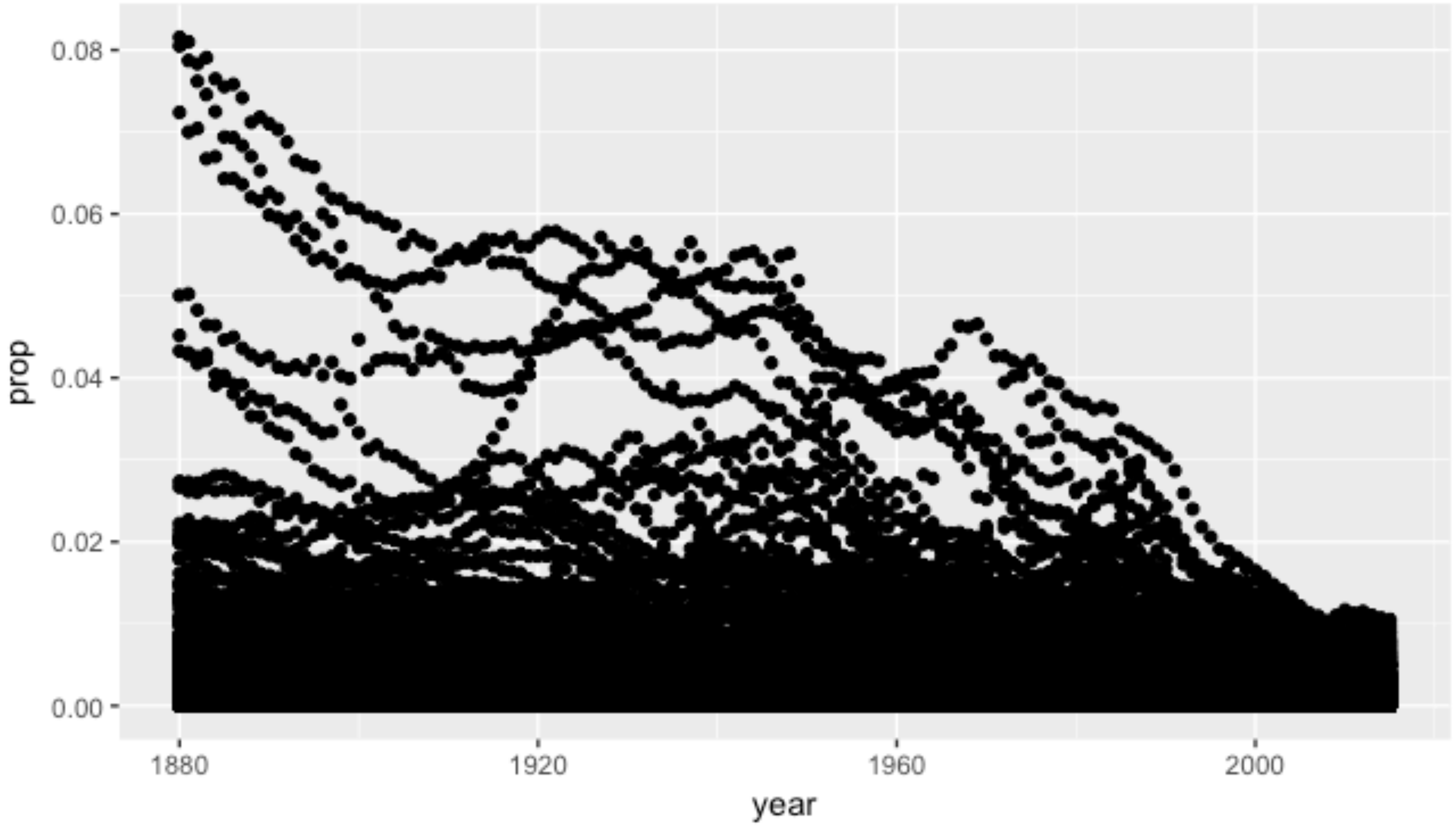
1–10 of 1,858,689 rows     Previous 1 2 3 4 5 6 ... 100 Next

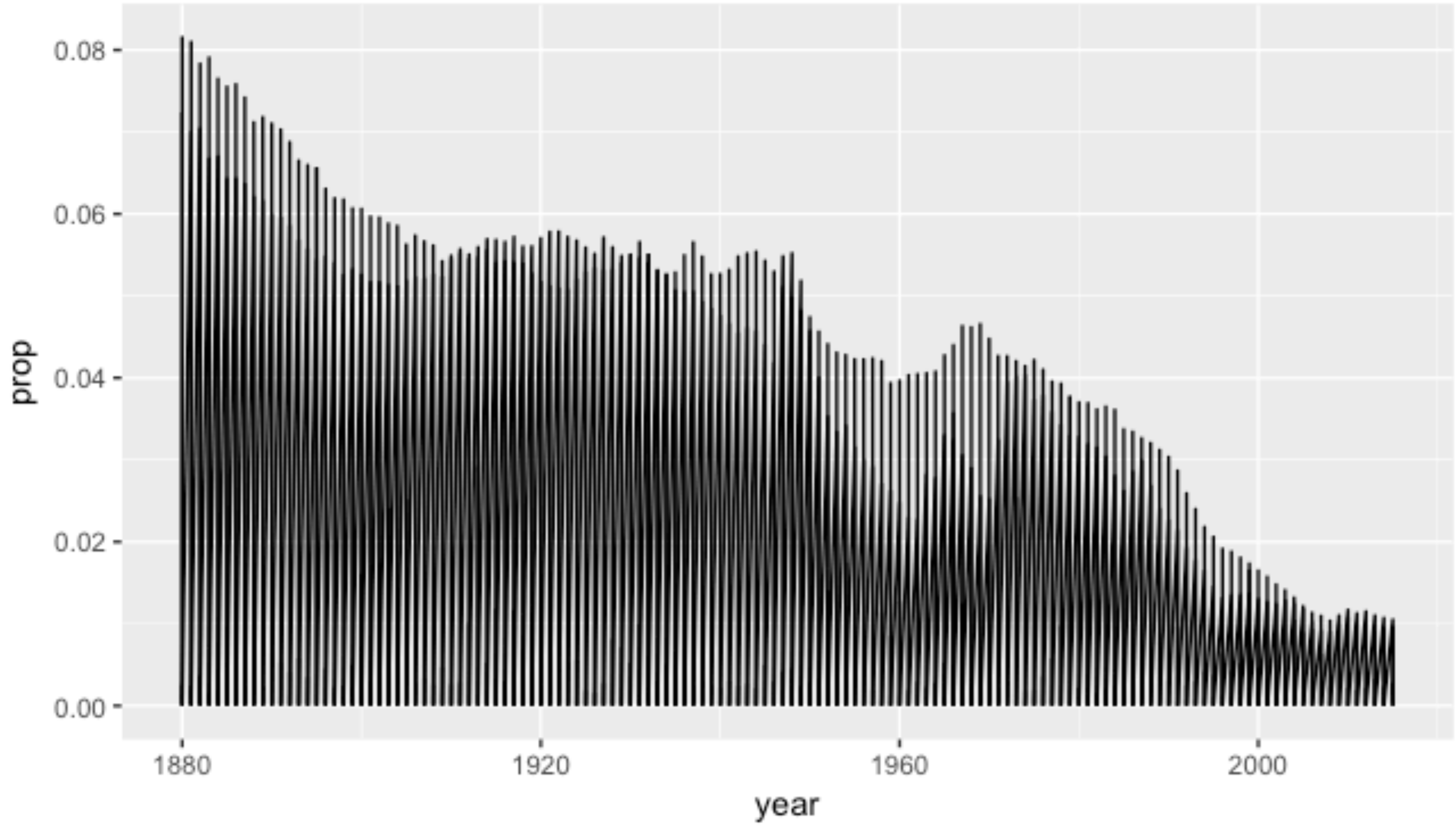


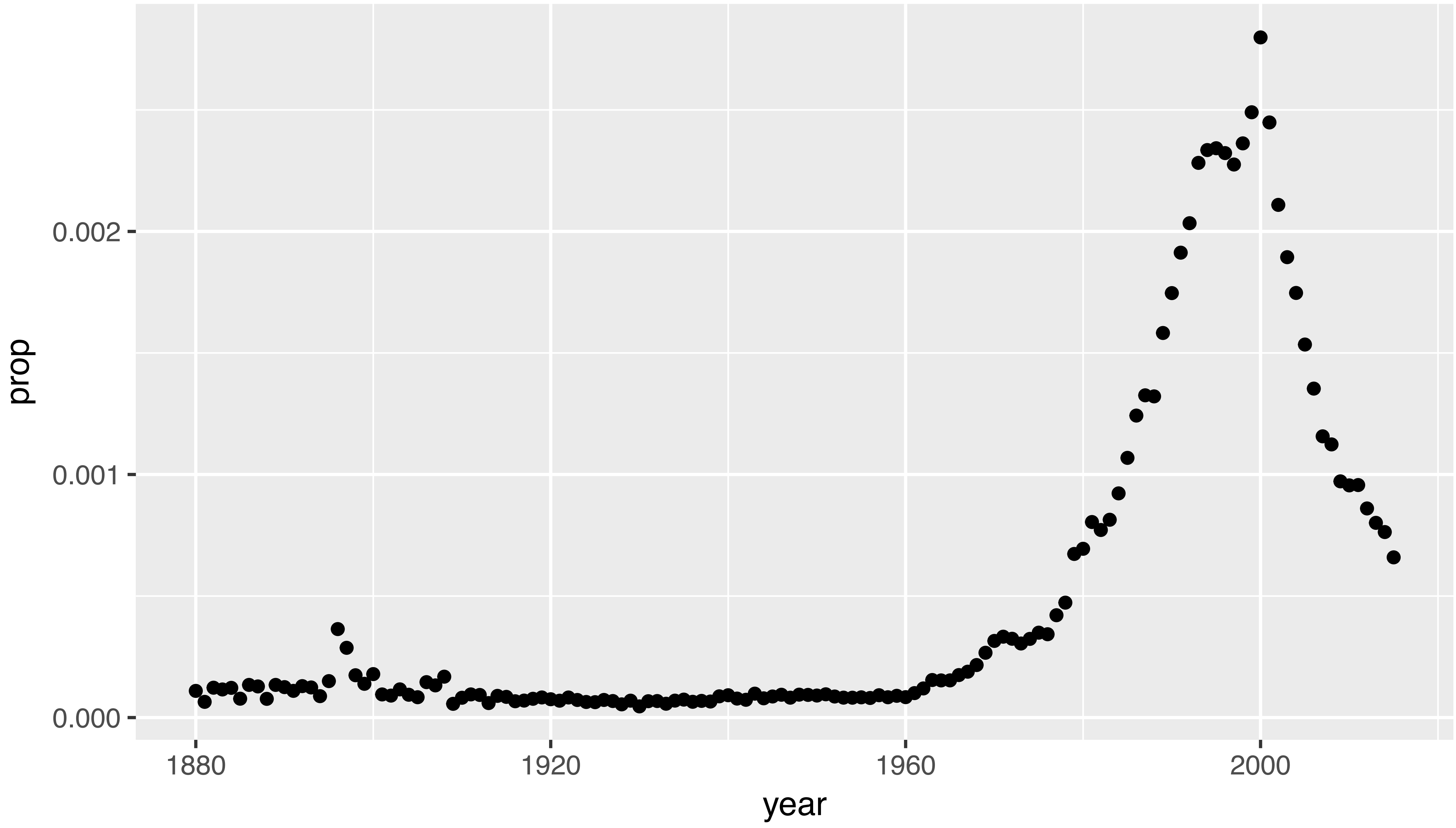
dplyr

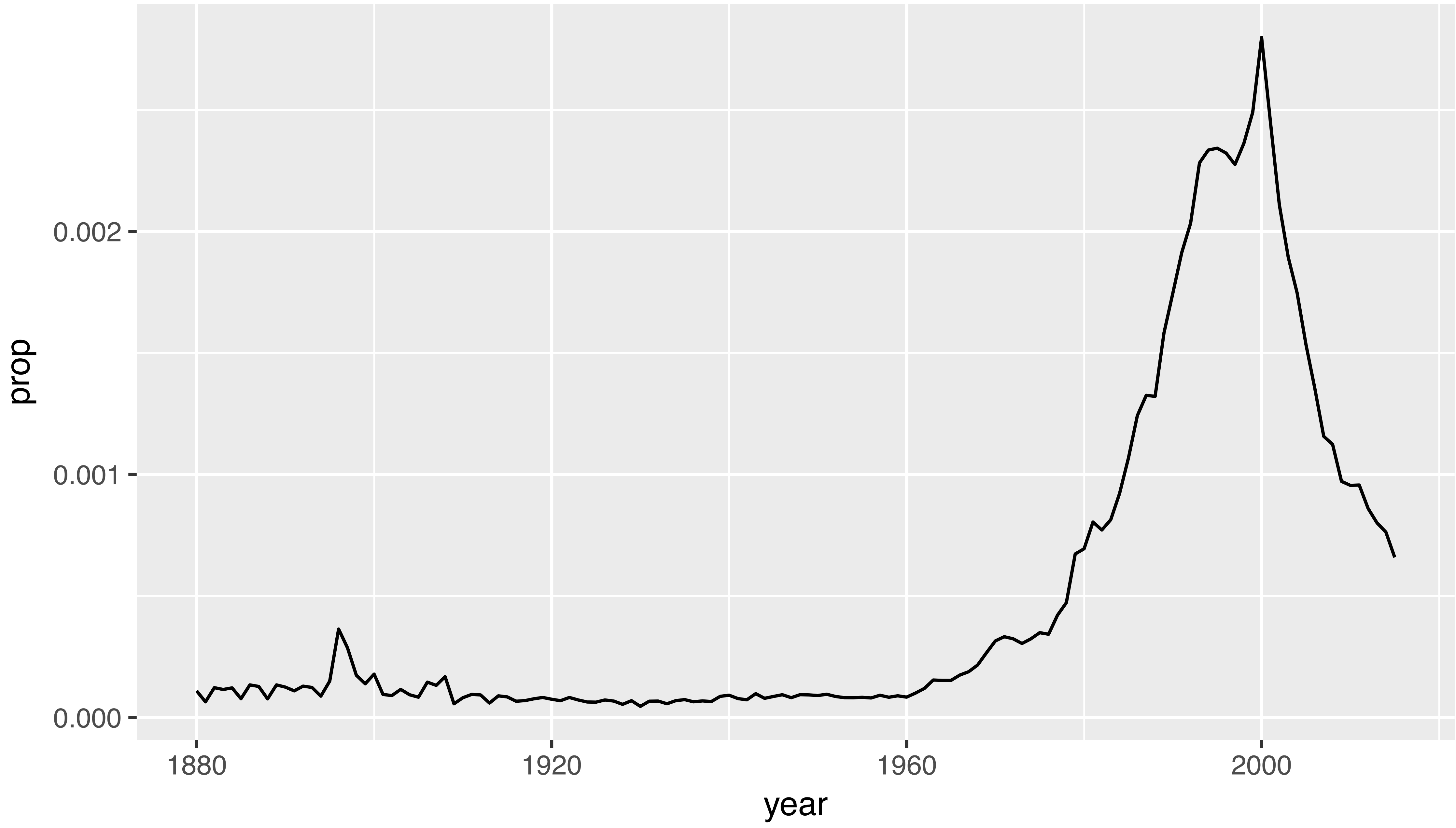
# Proportion of boys with the name Garrett




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

# 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 |



dplyr

# Your Turn 1

Alter the code to select just the **n** column:

```
select(babynames, name, prop)
```

```
select(babynames, n)
#       n
#   <int>
# 1  7065
# 2  2604
# 3  2003
# 4  1939
# 5  1746
# …    …
```

# select() helpers

**:** - Select range of columns

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

**-** - Select every column but

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

**starts_with()** - Select columns that start with…

```
select(storms, starts_with("w"))
```

**ends_with()** - Select columns that end with…

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

# select() helpers

**contains()** - Select columns whose names contain…

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

**matches()** - Select columns whose names match regular expression

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

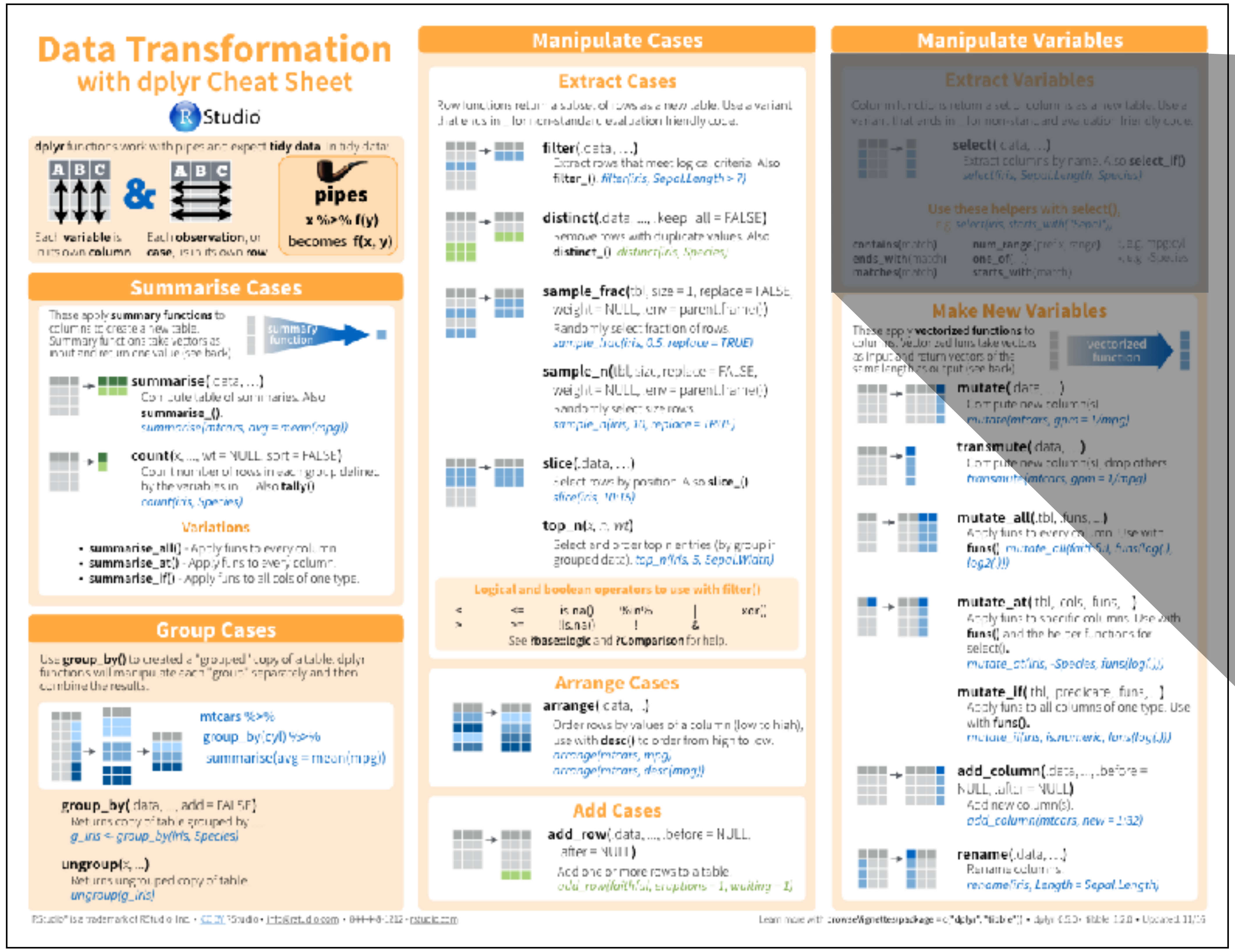
**one_of()** - Select columns whose names are one of a set

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

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

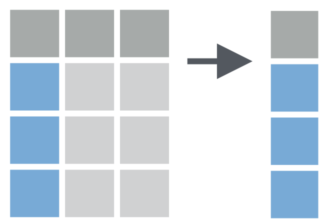
```
select(storms, num_range("x", 1:5))
```

# select() helpers




## 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) **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**

dplyr

# 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 | … | … |

dplyr

# 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)

dplyr

# 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 |
| `is.na(x)` | Is NA |
| `!is.na(x)` | Is not NA |

# Your Turn 2

See if you can use the logical operators to manipulate our code below to show:

- All of the names where **prop** is greater than or equal to 0.08
- All of the children named “Sea”
- All of the names that have a missing value for **n**
  (Hint: this should return an empty data set).

04:00

```
filter(babynames, prop >= 0.08)
#     year   sex     name     n        prop
# 1   1880     M     John  9655 0.08154630
# 2   1880     M  William  9531 0.08049899
# 3   1881     M     John  8769 0.08098299
```

```
filter(babynames, name == "Sea")
#     year   sex  name     n          prop
# 1   1982     F   Sea     5  2.756771e-06
# 2   1985     M   Sea     6  3.119547e-06
# 3   1986     M   Sea     5  2.603512e-06
# 4   1998     F   Sea     5  2.580377e-06
```

```
filter(babynames, is.na(n))
#  0 rows
```

# Two common mistakes

1. Using = instead of ==

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

2. Forgetting quotes

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

# filter()

Extract rows that meet *every* logical criteria.

```
filter(babynames, name == "Garrett", year == 1880)
```

babynames

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

→

| year | sex | name | n | prop |
|---|---|---|---|---|
| 1880 | M | Garrett | 13 | 0.0001 |

# Boolean operators

?base::Logic

| | |
|---|---|
| a & b | and |
| a \| b | or |
| xor(a, b) | exactly or |
| !a | not |

# 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 |



dplyr

# Your Turn 3

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

- Girls named Sea
- Names that were used by exactly 5 or 6 children in 1880
- Names that are one of Acura, Lexus, or Yugo

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

```
filter(babynames, name == "Sea", sex == "F")
#     year   sex   name      n          prop
# 1   1982     F    Sea      5 2.756771e-06
# 2   1998     F    Sea      5 2.580377e-06
```

```
filter(babynames, n == 5 | n == 6, year == 1880)
#     year   sex     name      n          prop
# 1   1880     F     Abby      6 6.147289e-05
# 2   1880     F   Aileen      6 6.147289e-05
# …      …     …        …      …             …
```

```
filter(babynames, name %in% c("Acura", "Lexus", "Yugo"))
#     year   sex   name      n          prop
# 1   1990     F Lexus      36 1.752932e-05
# 2   1990     M Lexus      12 5.579156e-06
# …      …     …     …       …             …
```

# 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))
```

# 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 |



dplyr

# Your Turn 4

Arrange babynames by **n**. Add **prop** as a second (tie breaking) variable to arrange on.

Can you tell what the smallest value of **n** is?

02 : 00

```
arrange(babynames, n, prop)
#     year   sex        name     n         prop
#  1  2007     M       Aaban     5 2.259872e-06
#  2  2007     M      Aareon     5 2.259872e-06
#  3  2007     M       Aaris     5 2.259872e-06
#  4  2007     M         Abd     5 2.259872e-06
#  5  2007     M  Abdulazeez     5 2.259872e-06
#  6  2007     M   Abdulhadi     5 2.259872e-06
#  7  2007     M  Abdulhamid     5 2.259872e-06
#  8  2007     M  Abdulkadir     5 2.259872e-06
#  9  2007     M Abdulraheem     5 2.259872e-06
# 10  2007     M  Abdulrahim     5 2.259872e-06
# ... with 1,858,679 more rows
```

# desc()

Changes ordering to largest to smallest.

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

babynames

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

→

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

# Your Turn 5

Use **desc()** to find the names with the highest **prop**.

Then, use **desc()** to find the names with the highest **n**.

02:00

```
arrange(babynames, desc(prop))
#      year   sex     name     n       prop
#  1   1880     M     John  9655 0.08154630
#  2   1881     M     John  8769 0.08098299
#  3   1880     M  William  9531 0.08049899
#  4   1883     M     John  8894 0.07907324
#  5   1881     M  William  8524 0.07872038
#  6   1882     M     John  9557 0.07831617
#  7   1884     M     John  9388 0.07648751
#  8   1882     M  William  9298 0.07619375
#  9   1886     M     John  9026 0.07582198
# 10   1885     M     John  8756 0.07551791
# ... with 1,858,679 more rows
```

```
arrange(babynames, desc(n))
#      year   sex     name      n       prop
#  1   1947     F    Linda  99680 0.05483609
#  2   1948     F    Linda  96211 0.05521159
#  3   1947     M    James  94763 0.05102057
#  4   1957     M  Michael  92726 0.04238659
#  5   1947     M   Robert  91646 0.04934237
#  6   1949     F    Linda  91010 0.05184281
#  7   1956     M  Michael  90623 0.04225479
#  8   1958     M  Michael  90517 0.04203881
#  9   1948     M    James  88588 0.04969679
# 10   1954     M  Michael  88493 0.04279403
# ... with 1,858,679 more rows
```

%>%

# 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

```
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))
```

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

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

vs.

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

# Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

# Your Turn 6

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

1. Filter babynames to just the girls that were born in 2015
2. Select the **name** and **n** columns
3. Arrange the results so that the most popular names are near the top.

05:00

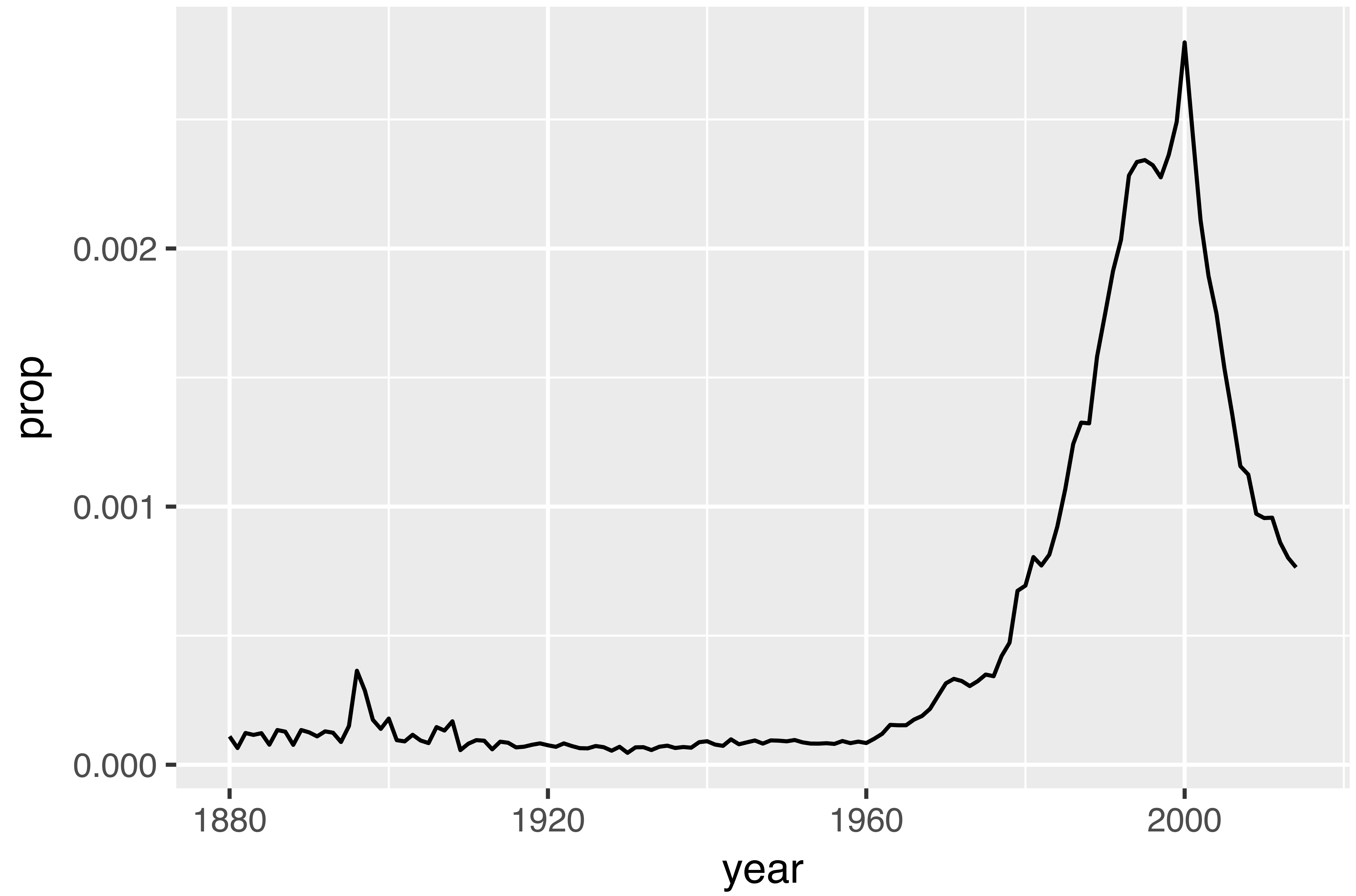
```
babynames %>%
  filter(year == 2015, sex == "F") %>%
  select(name, n) %>%
  arrange(desc(n))
#          name     n
#  1       Emma 20355
#  2     Olivia 19553
#  3     Sophia 17327
#  4        Ava 16286
#  5   Isabella 15504
#  6        Mia 14820
#  7    Abigail 12311
#  8      Emily 11727
#  9  Charlotte 11332
# 10     Harper 10241
# ... with 18,983 more rows
```

# Exam

1. Trim babynames to just the rows that contain your **name** and your **sex**
2. Trim the result to just the columns that will appear in your graph (not strictly necessary, but useful practice)
3. Plot the results as a line graph with **year** on the x axis and **prop** on the y axis

05:00

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



# What are the most popular names?

# How should we define popularity?

A name is popular if:

# How should we define popularity?

A name is popular if:

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

# How should we define popularity?

A name is popular if:

1. **Sums** - a large number of children have the name when you sum across years
2. **Ranks** - it consistently ranks among the top names from year to year.

# Quiz

Do we have enough information to:

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

# Deriving information

**summarise()** - summarise **variables**
**group_by()** - group **cases**
**mutate()** - create new **variables**

# summarise()

# summarise()

Compute table of summaries.

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

babynames

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

→

| total | max |
|---|---|
| 127538 | 99680 |



dplyr

# Your Turn 7

Use summarise() to compute three statistics *about the data*:

1. The first (minimum) year in the dataset
2. The last (maximum) year in the dataset
3. The total number of children represented in the data

03:00

```
babynames %>%
  summarise(first = min(year),
            last = max(year),
            total = sum(n))
#   first  last     total
# 1  1880  2015 340851912
```

# Your Turn 8

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

1. The total number of children named Khaleesi
2. The first **year** Khaleesi appeared in the data

03:00

```r
babynames %>%
  filter(name == "Khaleesi") %>%
  summarise(total = sum(n), first = min(year))
#   total first
# 1  1127  2011
```

# Summary functions

Take a vector as input.
Return a single value as output.

## Summary Functions

### to use with summarise()

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


#### Counts

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

#### Location

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

#### Logicals

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

#### Position/Order

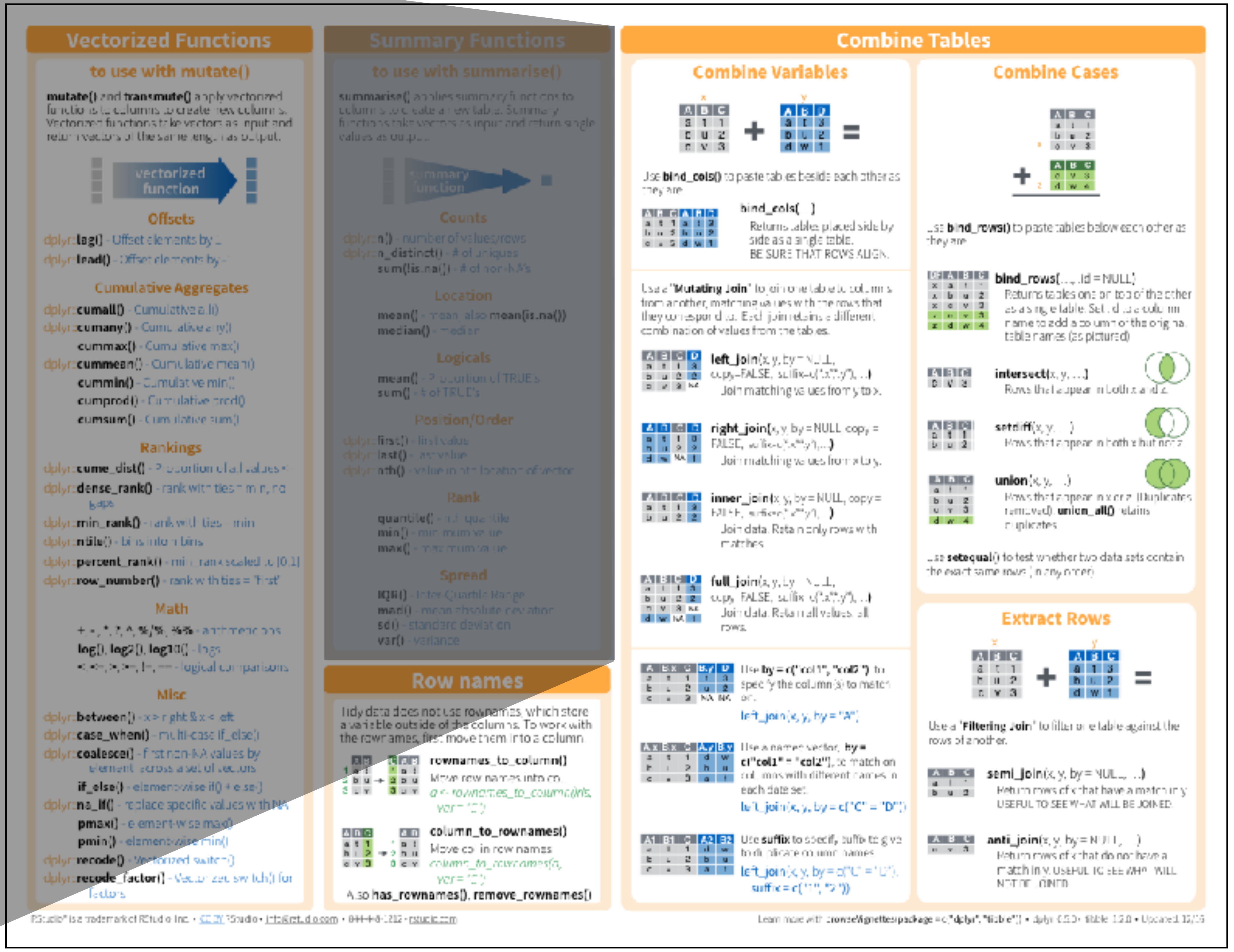
dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

#### Rank

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

#### Spread

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



# n()

The number of rows in a dataset/group

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

babynames

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

→

| n |
|---|
| 1858689 |

# n_distinct()

The number of distinct values in a variable

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

babynames

| year | sex | name | n | prop |
|---|---|---|---|---|
| 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 |
|---|---|
| 1858689 | 95025 |

# Grouping cases

# group_by()

Groups cases by common values of one or more columns.

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

```
Source: local data frame [1,825,433 x 5]
Groups: sex [2]

   year   sex     name     n       prop
  <dbl> <chr>    <chr> <int>      <dbl>
1  1880     F     Mary  7065 0.07238359
```

# group_by()

Groups cases by common values.

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

| sex | total |
| --- | --- |
| F | 167070477 |
| M | 170064949 |

# Transform Data Notebook

02-Transform-Data.Rmd

```
---
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 ($\mu g/m^3$) |
|---|---|---|
| New York | large | 23 |
| New York | small | 14 |
| London | large | 22 |
| London | small | 16 |
| Beijing | large | 121 |
| Beijing | small | 56 |

| city | particle size | amount (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 ($\mu g/m^3$) |
| --- | --- | --- |
| 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 |

dplyr

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

| mean | sum | n |
|---|---|---|
| 18.5 | 37 | 2 |
| 19.0 | 38 | 2 |
| 88.5 | 177 | 2 |

# group_by() + summarise()

# group_by()

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

| city | particle size | amount ($\mu g/m^3$) |
|---|---|---|
| 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 |

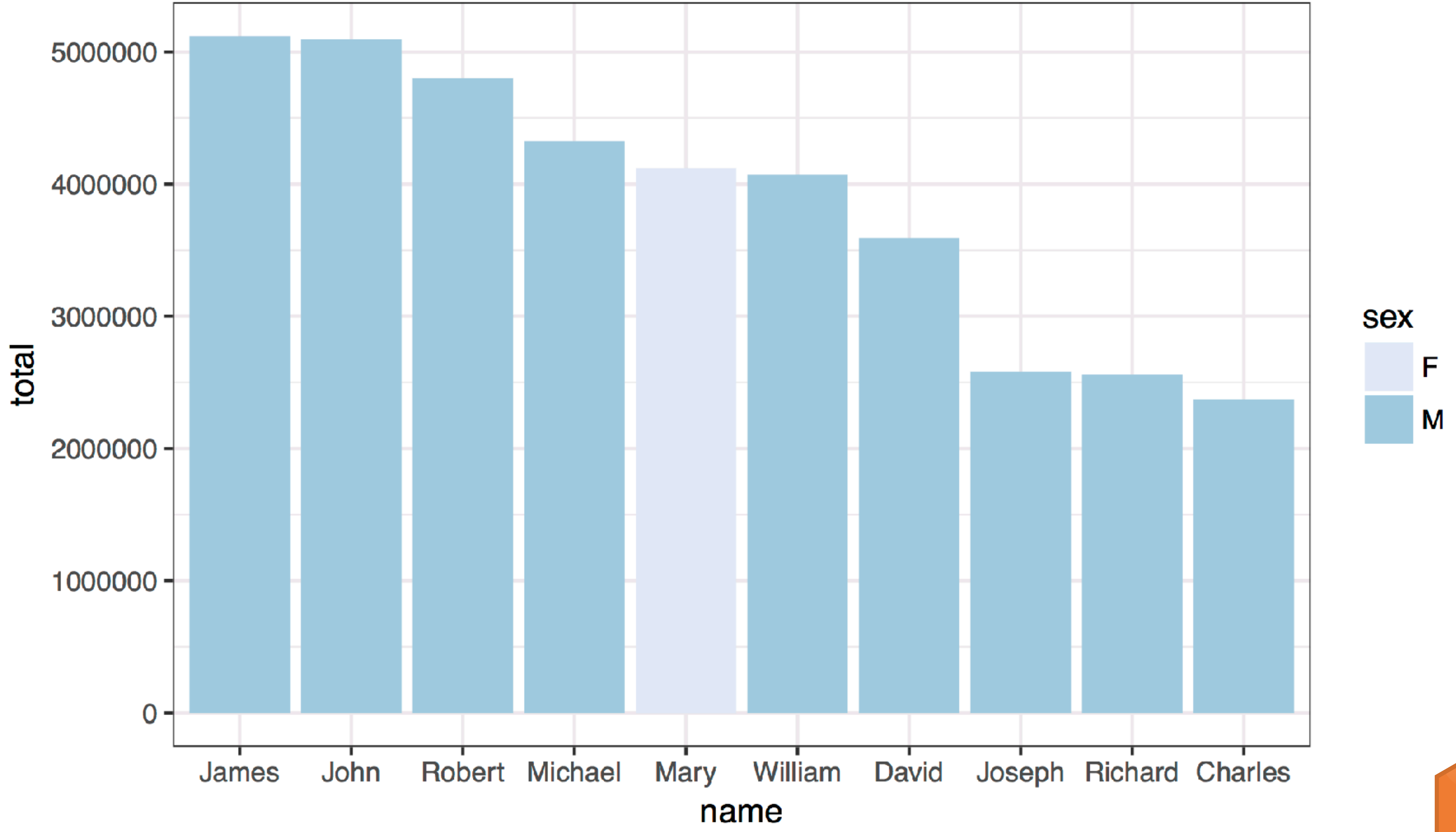
```
pollution %>%
  group_by(city) %>%
  summarise(mean = mean(amount), sum = sum(amount), n = n())
```

# Your Turn 9

Use **group_by()**, **summarise()**, and **arrange()** to display the ten most popular names. Compute popularity as the total number of children of a single gender given a name.

05:00

```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total))
#        name   sex   total
#   1   James     M 5120990
#   2    John     M 5095674
#   3  Robert     M 4803068
#   4 Michael     M 4323928
#   5    Mary     F 4118058
#   6 William     M 4071645
#   7   David     M 3589754
#   8  Joseph     M 2581785
#   9 Richard     M 2558165
#  10 Charles     M 2371621
# ... with 105,376 more rows
```


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

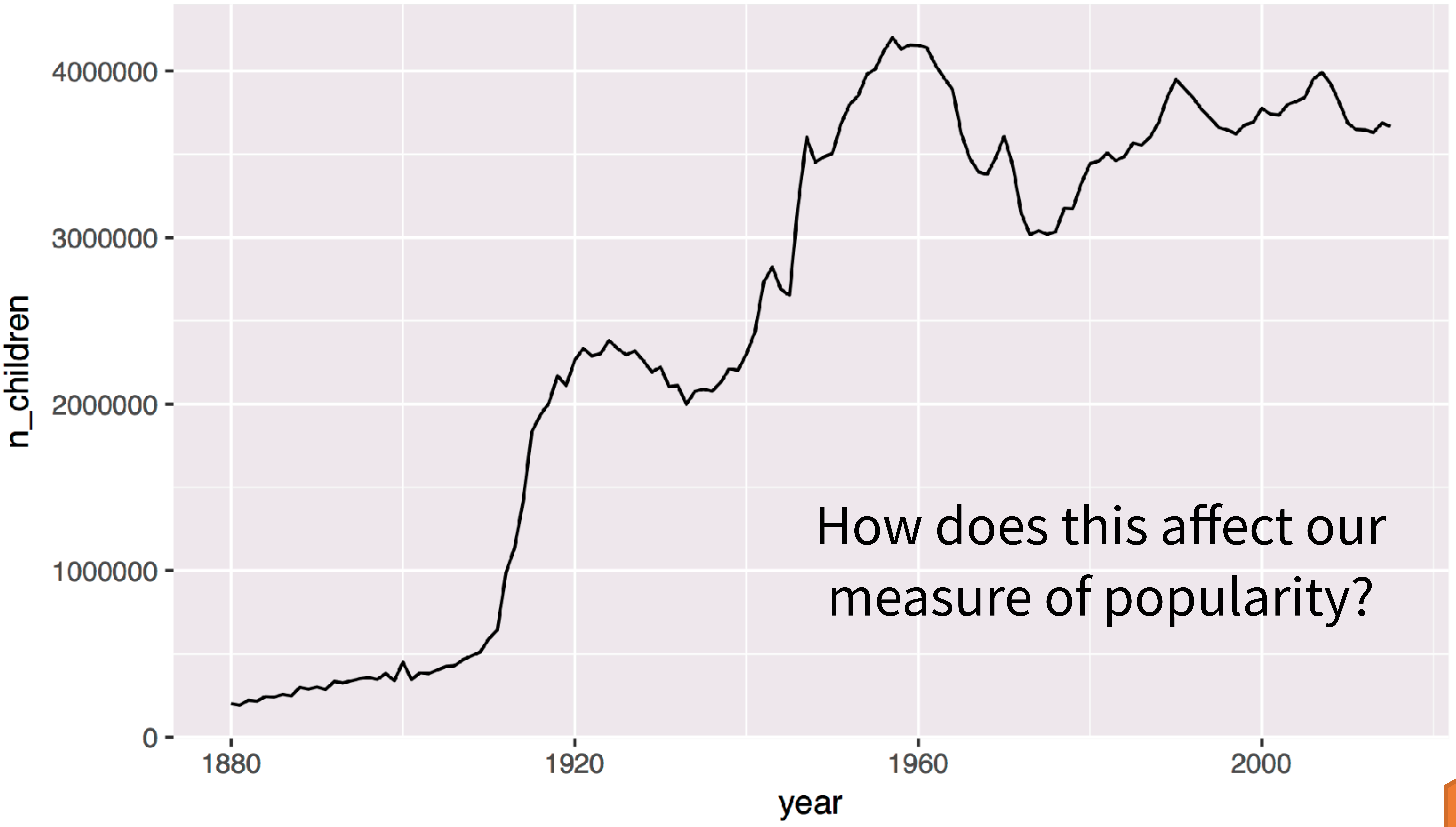
```
babynames %>%
  group_by(name, sex) %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total)) %>%
  ungroup() %>%
  slice(1:10) %>%
  ggplot() +
    geom_col(mapping = aes(x = fct_reorder(name,
desc(total)), y = total, fill = sex)) +
    theme_bw() +
    scale_fill_brewer() +
    labs(x = "name")
```

# Your Turn 10

Use grouping to calculate and then plot the number of children born each year over time.

05:00

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




How does this affect our measure of popularity?

# ungroup()

Removes grouping criteria from a data frame.

```
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
```

# ungroup()

Removes grouping criteria from a data frame.

```
babynames %>%
  group_by(name, sex) %>%
  ungroup() %>%
  summarise(total = sum(n)) %>%
  arrange(desc(total))
#       total
# 1 340851912
```

# 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 |

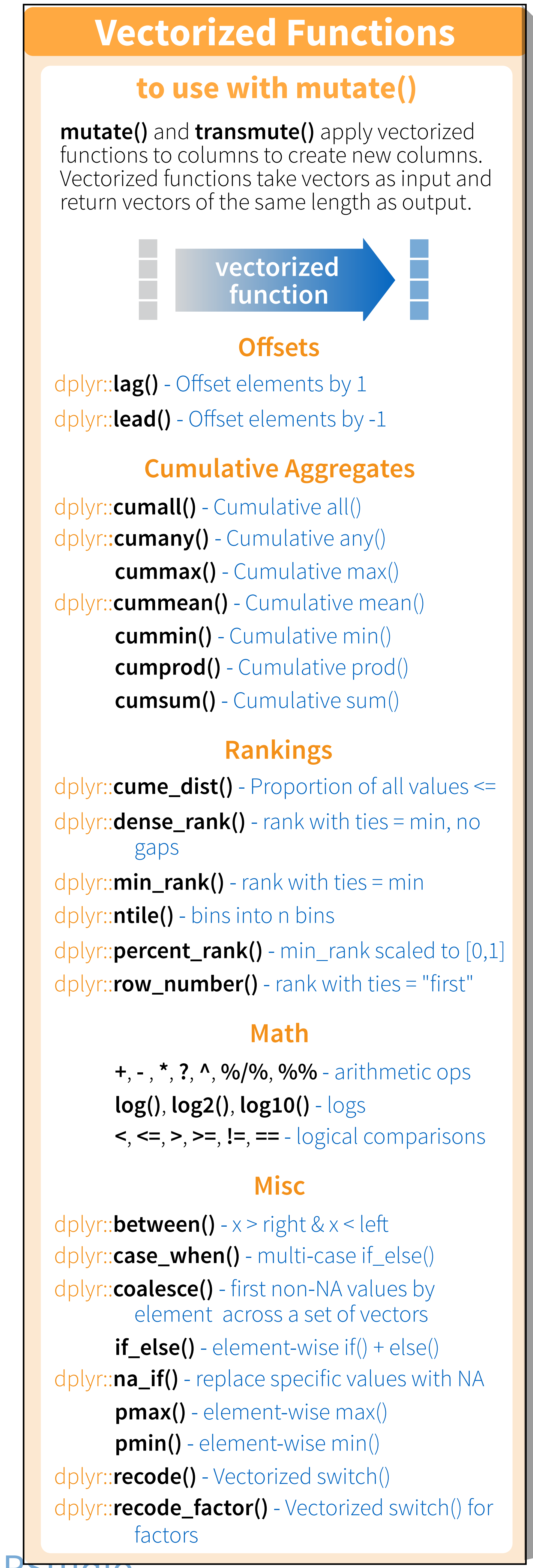
→

| 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 |

# 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

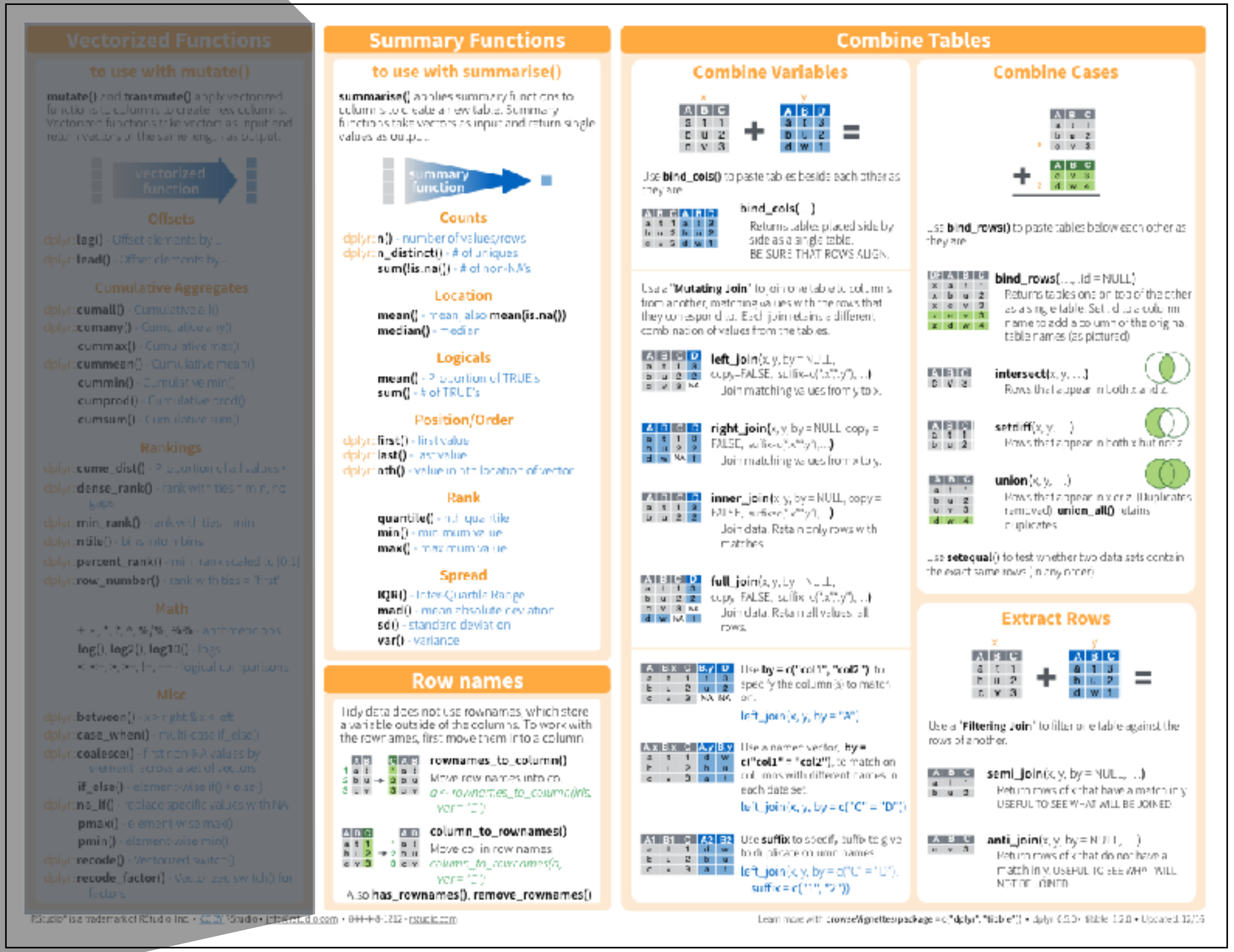
dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

#### Math

+, - , *, ?, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons

#### Misc

dplyr::**between()** - x > 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



# min_rank()

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

```
min_rank(c(50, 100, 1000))
# [1] 1 2 3
```

```
min_rank(desc(c(50, 100, 1000)))
# [1] 3 2 1
```

# Your Turn 11

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

02:00

```
babynames %>%
  mutate(rank = min_rank(desc(prop)))
#      year   sex      name    n       prop  rank
#   1  1880     F      Mary 7065 0.07238433    14
#   2  1880     F      Anna 2604 0.02667923   709
#   3  1880     F      Emma 2003 0.02052170  1131
#   4  1880     F Elizabeth 1939 0.01986599  1192
#   5  1880     F    Minnie 1746 0.01788861  1427
#   6  1880     F  Margaret 1578 0.01616737  1683
#   7  1880     F       Ida 1472 0.01508135  1897
#   8  1880     F     Alice 1414 0.01448711  2040
#   9  1880     F    Bertha 1320 0.01352404  2279
#  10  1880     F     Sarah 1288 0.01319618  2387
# ... with 1,858,679 more rows
```

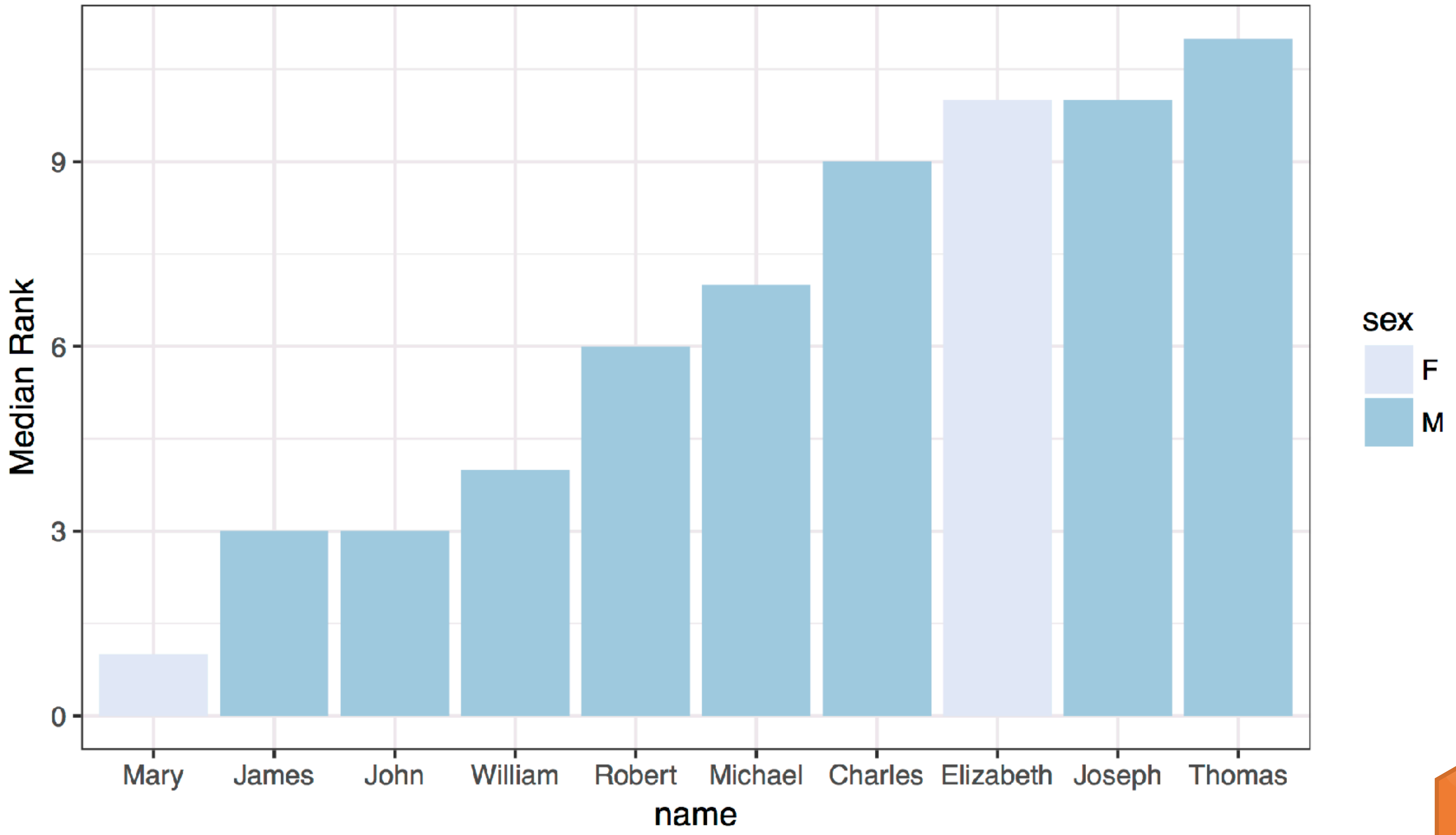
# Your Turn 12

Compute each name's rank *within its year and sex*.

Then compute the median rank for each combination of *name and sex*, and arrange the results from highest median rank to lowest.

05:00

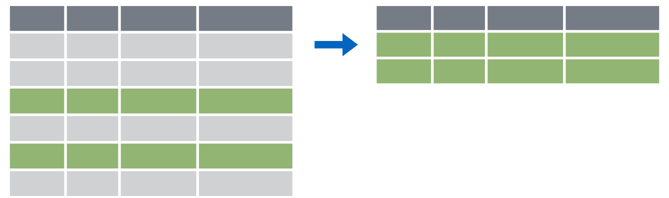
```
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  group_by(name, sex) %>%
  summarise(score = median(rank)) %>%
  arrange(score)
#          name   sex score
#  1       Mary     F     1
#  2      James     M     3
#  3       John     M     3
#  4    William     M     4
#  5     Robert     M     6
# ... with 105,381 more rows
```


9
6
3
0
Median Rank
Mary
James
John
William
Robert
Michael
Charles
Elizabeth
Joseph
Thomas
name
sex
F
M

```
babynames %>%
  group_by(year, sex) %>%
  mutate(rank = min_rank(desc(prop))) %>%
  group_by(name, sex) %>%
  summarise(score = median(rank)) %>%
  ungroup() %>%
  arrange(score) %>%
  slice(1:10) %>%
  ggplot() +
    geom_col(mapping = aes(x = fct_reorder(name, score), y = score,
fill = sex)) +
    theme_bw() +
    scale_fill_brewer() +
    labs(x = "name", y = "Median Rank")
```

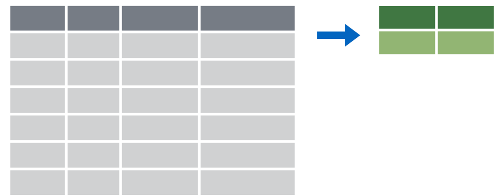
# Recap: Single table verbs

Extract variables with **select()**

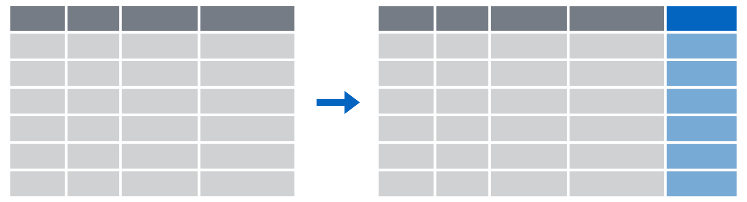


Extract cases with **filter()**

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



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



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

# Joining Datasets

# nycflights13

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

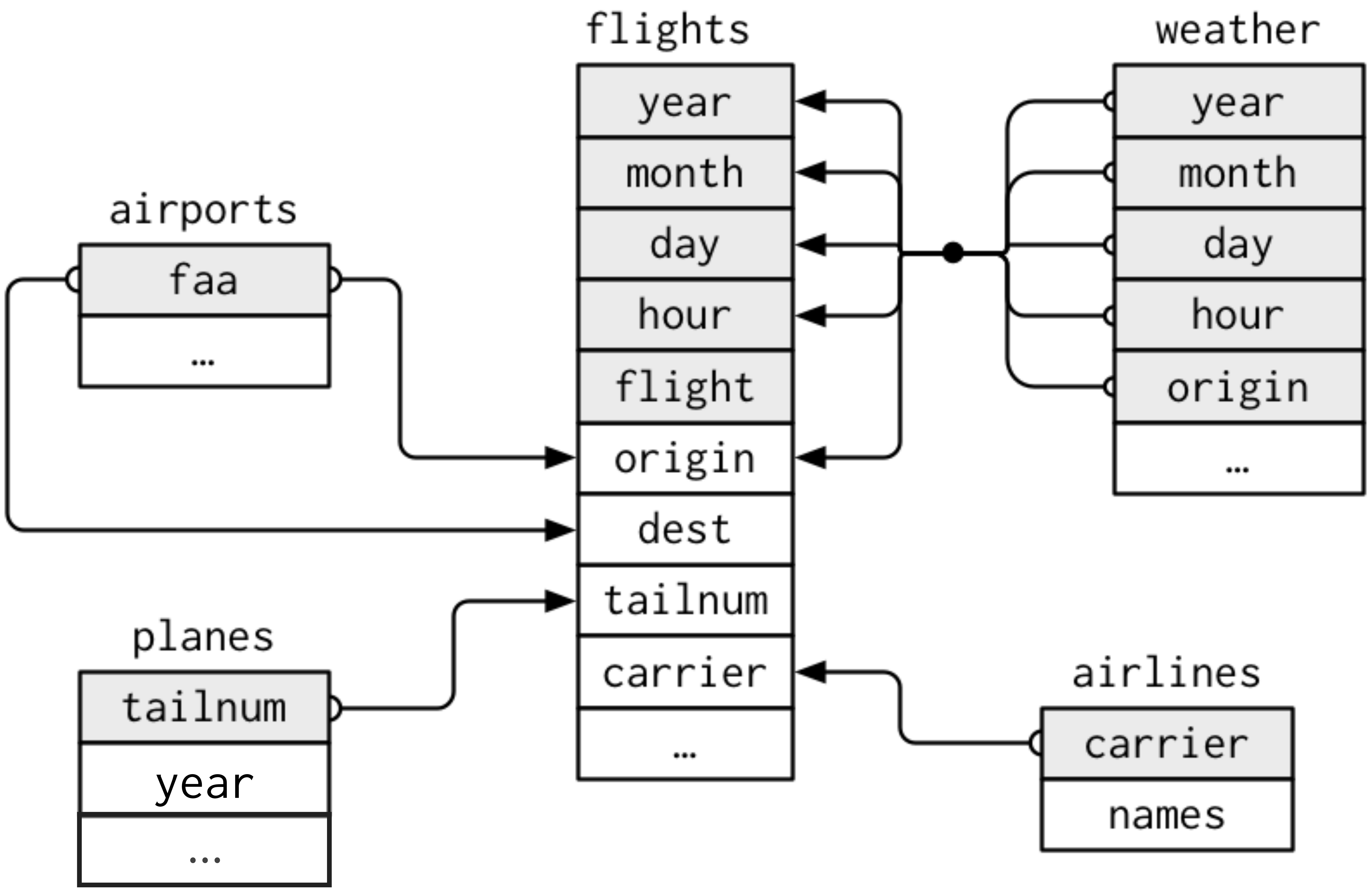
```
# install.packages("nycflights13")
library(nycflights13)
```

# Flights

```
View(flights)
```

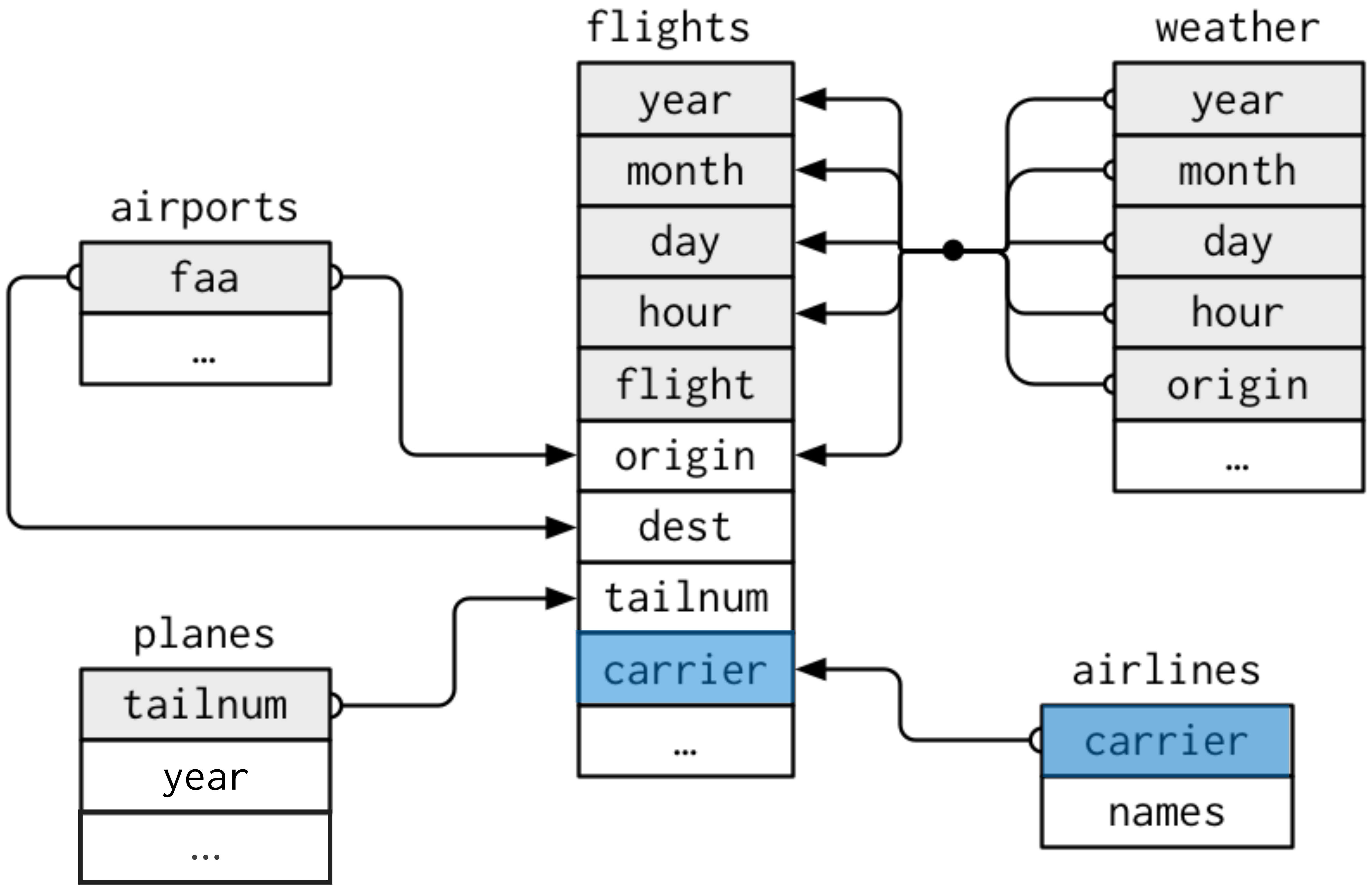
| year | month | day | dep_time | sched_dep_time | dep_delay | arr_time | sched_arr_time | arr_delay | carrier | flight | tailnum | origin | dest | air_time | distance | hour | minute | time_hour |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2013 | 1 | 1 | 517 | 515 | 2 | 830 | 819 | 11 | UA | 1545 | N14228 | EWR | IAH | 227 | 1400 | 5 | 15 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 533 | 529 | 4 | 850 | 830 | 20 | UA | 1714 | N24211 | LGA | IAH | 227 | 1416 | 5 | 29 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 542 | 540 | 2 | 923 | 850 | 33 | AA | 1141 | N619AA | JFK | MIA | 160 | 1089 | 5 | 40 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 544 | 545 | -1 | 1004 | 1022 | -18 | B6 | 725 | N804JB | JFK | BQN | 183 | 1576 | 5 | 45 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 554 | 600 | -6 | 812 | 837 | -25 | DL | 461 | N668DN | LGA | ATL | 116 | 762 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 554 | 558 | -4 | 740 | 728 | 12 | UA | 1696 | N39463 | EWR | ORD | 150 | 719 | 5 | 58 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 555 | 600 | -5 | 913 | 854 | 19 | B6 | 507 | N516JB | EWR | FLL | 158 | 1065 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 557 | 600 | -3 | 709 | 723 | -14 | EV | 5708 | N829AS | LGA | IAD | 53 | 229 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 557 | 600 | -3 | 838 | 846 | -8 | B6 | 79 | N593JB | JFK | MCO | 140 | 944 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 558 | 600 | -2 | 753 | 745 | 8 | AA | 301 | N3ALAA | LGA | ORD | 138 | 733 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 558 | 600 | -2 | 849 | 851 | -2 | B6 | 49 | N793JB | JFK | PBI | 149 | 1028 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 558 | 600 | -2 | 853 | 856 | -3 | B6 | 71 | N657JB | JFK | TPA | 158 | 1005 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 558 | 600 | -2 | 924 | 917 | 7 | UA | 194 | N29129 | JFK | LAX | 345 | 2475 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 558 | 600 | -2 | 923 | 937 | -14 | UA | 1124 | N53441 | EWR | SFO | 361 | 2565 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 559 | 600 | -1 | 941 | 910 | 31 | AA | 707 | N3DUAA | LGA | DFW | 257 | 1389 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 559 | 559 | 0 | 702 | 706 | -4 | B6 | 1806 | N708JB | JFK | BOS | 44 | 187 | 5 | 59 | 2013-01-01 05:00:00 |
| 2013 | 1 | 1 | 559 | 600 | -1 | 854 | 902 | -8 | UA | 1187 | N76515 | EWR | LAS | 337 | 2227 | 6 | 0 | 2013-01-01 06:00:00 |
| 2013 | 1 | 1 | 600 | 600 | 0 | 851 | 858 | -7 | B6 | 371 | N595JB | LGA | FLL | 152 | 1076 | 6 | 0 | 2013-01-01 06:00:00 |

# 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

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

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

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

instrument

| name | plays |
|---|---|
| John | guitar |
| Paul | bass |
| Keith | guitar |

# Toy data

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

instrument

| name | plays |
|---|---|
| John | guitar |
| Paul | bass |
| Keith | guitar |

# left

```
band %>% left_join(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

```
band %>% right_join(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

```r
band %>% full_join(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

```
band %>% inner_join(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 13

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

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

```
instrument2 <- tribble(
    ~artist,   ~plays,
   "John", "guitar",
   "Paul",   "bass",
  "Keith", "guitar"
)
```

instrument2

| 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.

```
band %>% left_join(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

dplyr

# common syntax - matching names

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

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

+

instrument2

| artist | plays |
|---|---|
| John | guitar |
| Paul | bass |
| Keith | guitar |

=

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

# Airport names

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

```
View(airports)
```

| | dest |
|---|---|
| 1 | IAH |
| 2 | IAH |
| 3 | MIA |
| 4 | BQN |
| 5 | ATL |
| 6 | ORD |

| faa | name |
|---|---|
| 04G | Lansdowne Airport |
| 06A | Moton Field Municipal Airport |
| 06C | Schaumburg Regional |
| 06N | Randall Airport |
| 09J | Jekyll Island Airport |
| 0A9 | Elizabethton Municipal Airport |

# Airport names

```
airports %>% select(1:3)
```

| faa <chr> | name <chr> |
|---|---|
| 04G | Lansdowne Airport |
| 06A | Moton Field Municipal Airport |
| 06C | Schaumburg Regional |
| 06N | Randall Airport |
| 09J | Jekyll Island Airport |
| 0A9 | Elizabethton Municipal Airport |
| 0G6 | Williams County Airport |
| 0G7 | Finger Lakes Regional Airport |

```
flights %>% select(14:15)
```

| dest <chr> | air_time <dbl> |
|---|---|
| IAH | 227 |
| IAH | 227 |
| MIA | 160 |
| BQN | 183 |
| ATL | 116 |
| ORD | 150 |
| FLL | 158 |
| IAD | 53 |

# common syntax - matching names

```
airports %>% left_join(flights, by = c("faa" = "dest"))
```

| faa <chr> | name <chr> |
|---|---|
| 04G | Lansdowne Airport |
| 06A | Moton Field Municipal Airport |
| 06C | Schaumburg Regional |
| 06N | Randall Airport |
| 09J | Jekyll Island Airport |
| 0A9 | Elizabethton Municipal Airport |
| 0G6 | Williams County Airport |
| 0G7 | Finger Lakes Regional Airport |

| dest <chr> | air_time <dbl> |
|---|---|
| IAH | 227 |
| IAH | 227 |
| MIA | 160 |
| BQN | 183 |
| ATL | 116 |
| ORD | 150 |
| FLL | 158 |
| IAD | 53 |



dplyr

# 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

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

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

\+

instrument

| name | plays |
|---|---|
| John | guitar |
| Paul | bass |
| Keith | guitar |

=

| name | band |
|---|---|
| John | Beatles |
| Paul | Beatles |

# anti

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

band

| name | band |
|---|---|
| Mick | Stones |
| John | Beatles |
| Paul | Beatles |

+

instrument

| name | plays |
|---|---|
| John | guitar |
| Paul | bass |
| Keith | guitar |

=

| name | band |
|---|---|
| Mick | Stones |

dplyr

# Airport names

```
airports %>% select(1:3)
```

| faa <chr> | name <chr> |
|---|---|
| 04G | Lansdowne Airport |
| 06A | Moton Field Municipal Airport |
| 06C | Schaumburg Regional |
| 06N | Randall Airport |
| 09J | Jekyll Island Airport |
| 0A9 | Elizabethton Municipal Airport |
| 0G6 | Williams County Airport |
| 0G7 | Finger Lakes Regional Airport |

```
flights %>% select(14:15)
```

| dest <chr> | air_time <dbl> |
|---|---|
| IAH | 227 |
| IAH | 227 |
| MIA | 160 |
| BQN | 183 |
| ATL | 116 |
| ORD | 150 |
| FLL | 158 |
| IAD | 53 |

# Your Turn 14

How many airports in **airports** are serviced by flights originating in New York (i.e. flights in our dataset?)

Notice that the column to join on is named **faa** in the **airports** dataset and **dest** in the **flights** dataset.

05:00

```r
airports %>%
  semi_join(flights, by = c("faa" = "dest")) %>%
  distinct(faa)
```

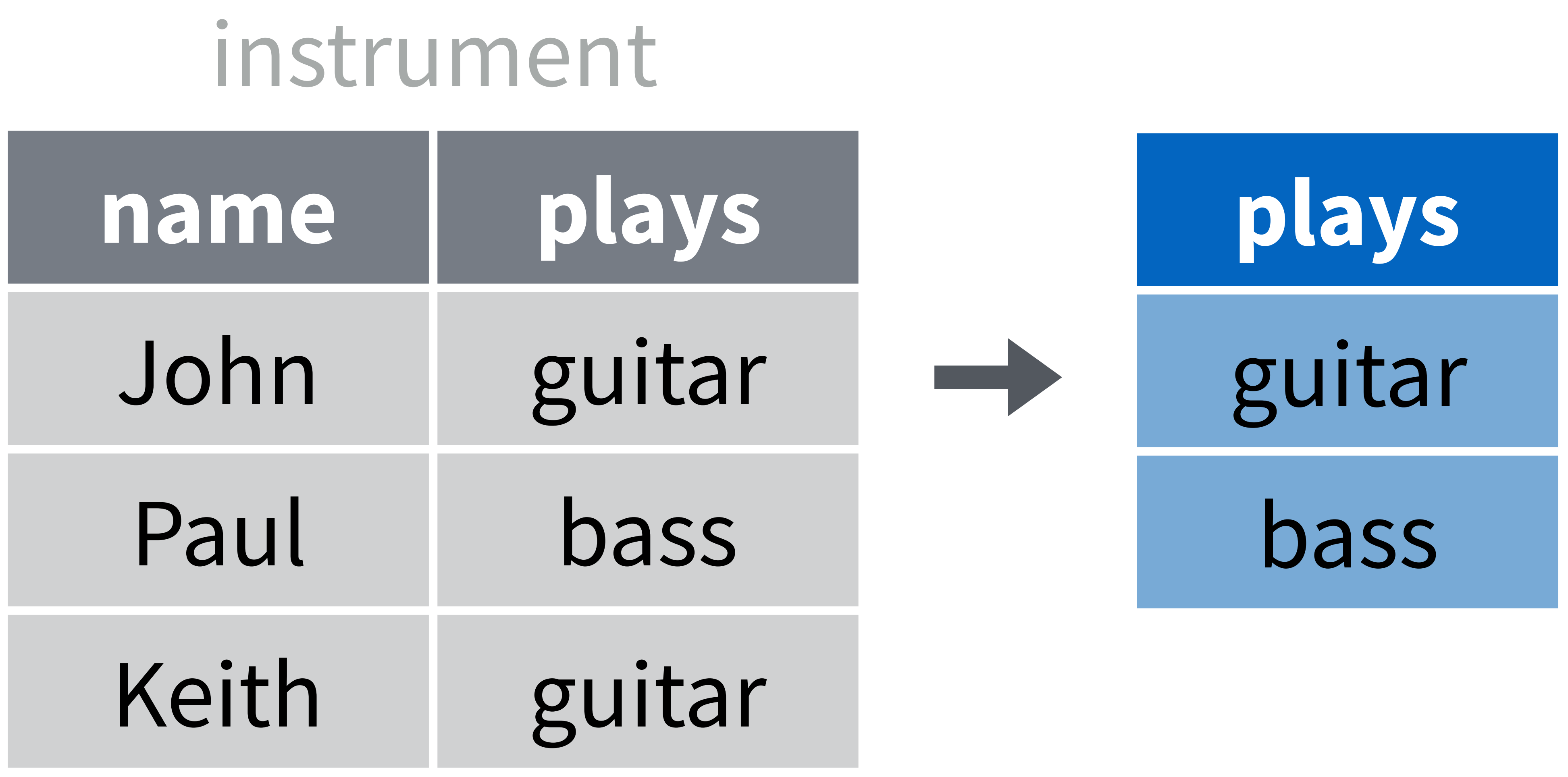
| faa <chr> |
|---|
| IAH |
| MIA |
| ATL |
| ORD |
| FLL |
| IAD |
| MCO |
| PBI |
| TPA |
| LAX |

1-10 of 101 rows    Previous 1 2 3 4 5 6 ... 11 Next

# distinct()

Removes rows with duplicate values (in a column).

```
distinct(instrument, plays)
```



# Recap: Two table verbs

**left_join()** retains all cases in **left** data set

**right_join()** retains all cases in **right** data set

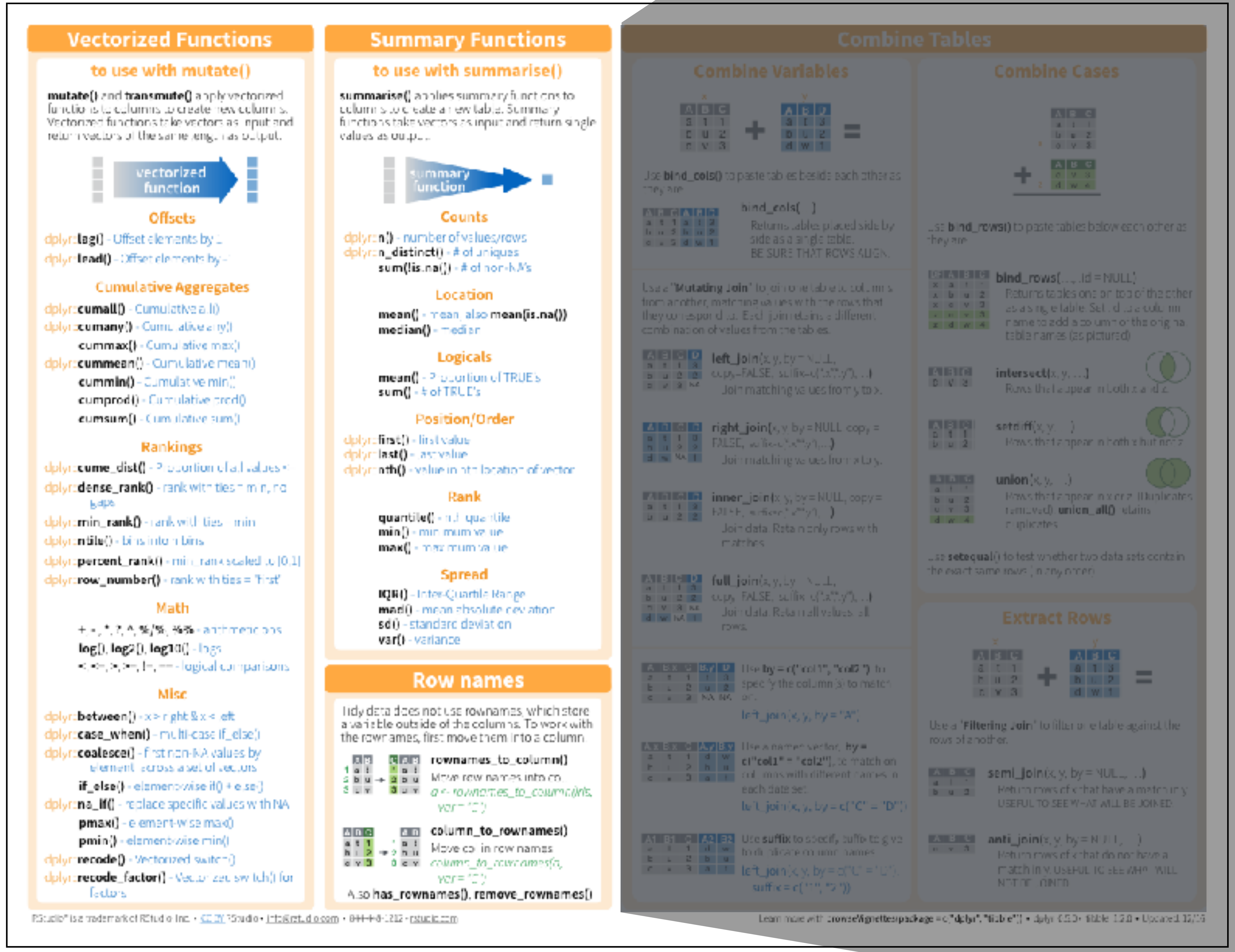
**full_join()** retains all cases in **either** data set

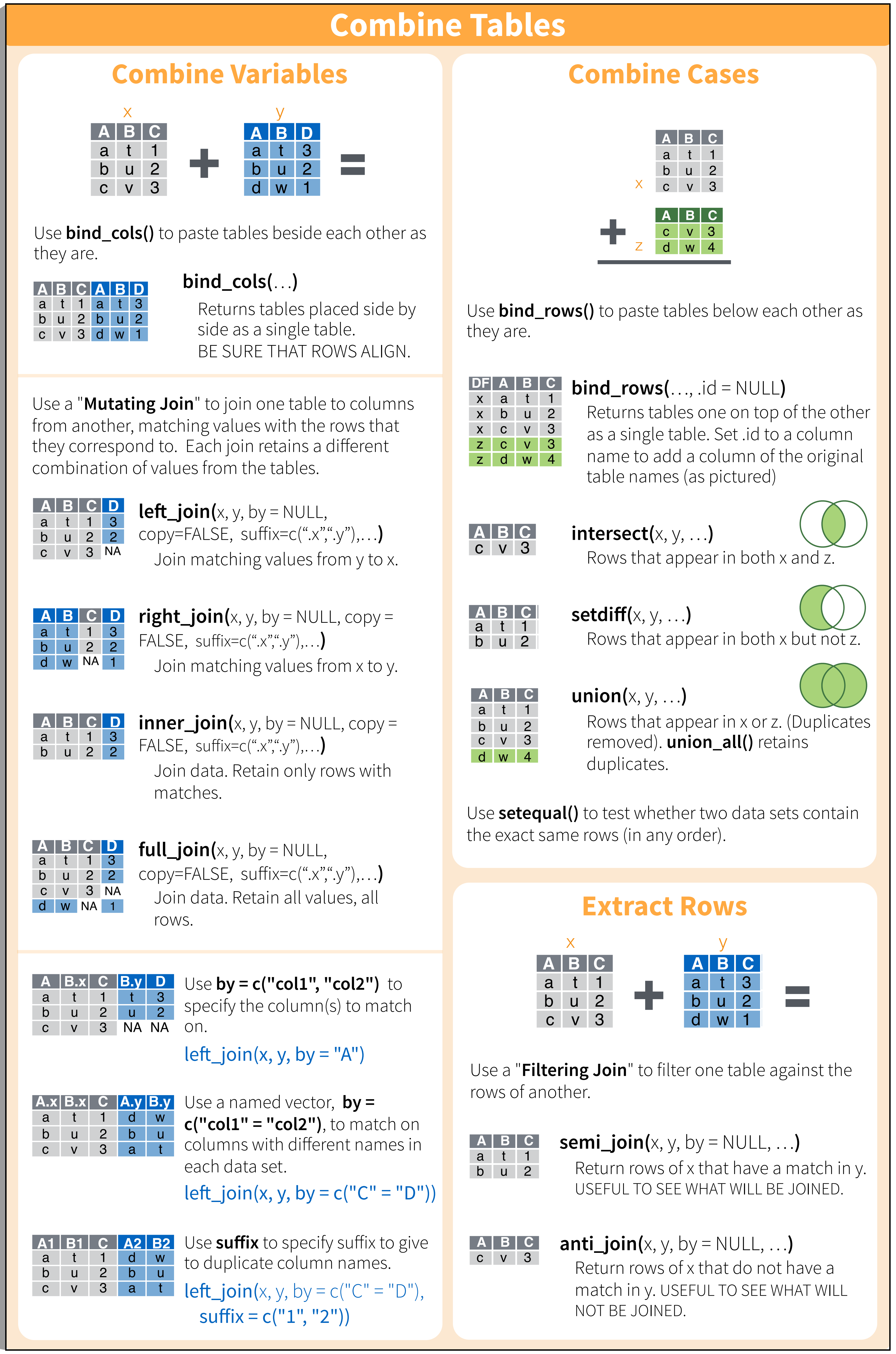
**inner_join()** retains only cases in **both** data sets

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

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

# Two table verbs



## 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 to 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, suffix=c(".x",".y"),…)
Join matching values from y to x.

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

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

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

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

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"))

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

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 of the original table names (as pictured)

**intersect(**x, y, …)
Rows that appear in both x and z.

**setdiff(**x, y, …)
Rows that appear in both x but not z.

**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

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

**semi_join(**x, y, by = NULL, …)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

**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:

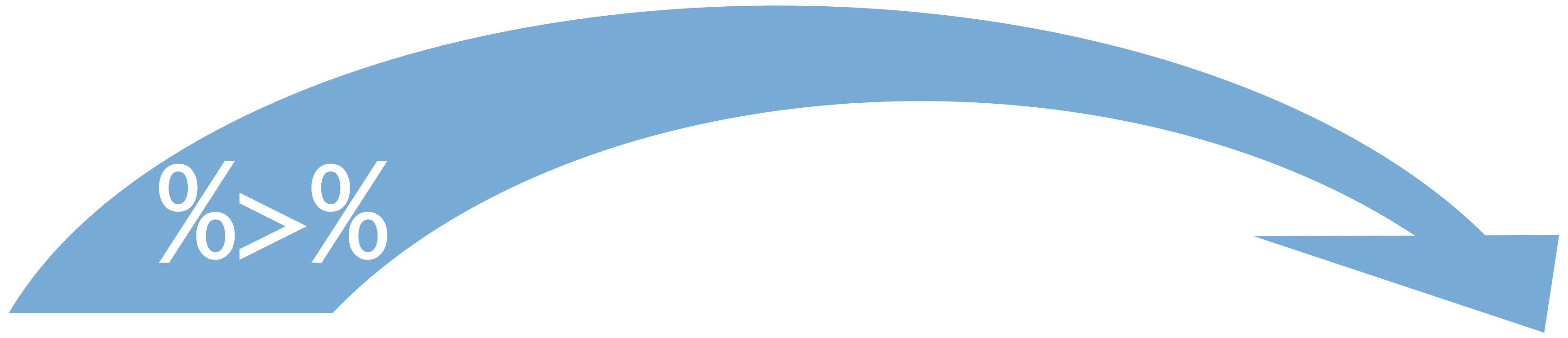
1. **Simple** - They do one thing, and they do it well
2. **Composable** - They can be combined with other functions for multi-step operations
3. **Smart** - They can use R objects as input.

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 :)
```

# Transform Data with

dplyr

www.rstudio.com