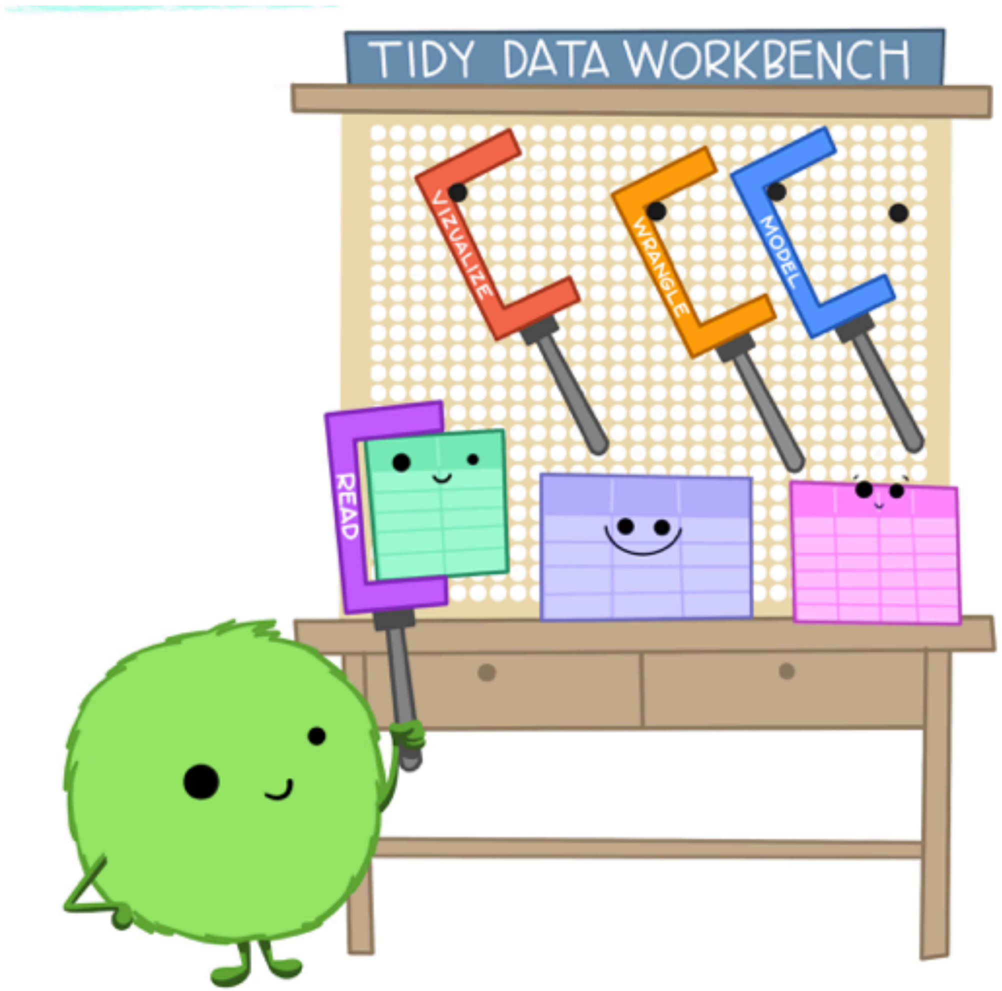
Intro to dplyr

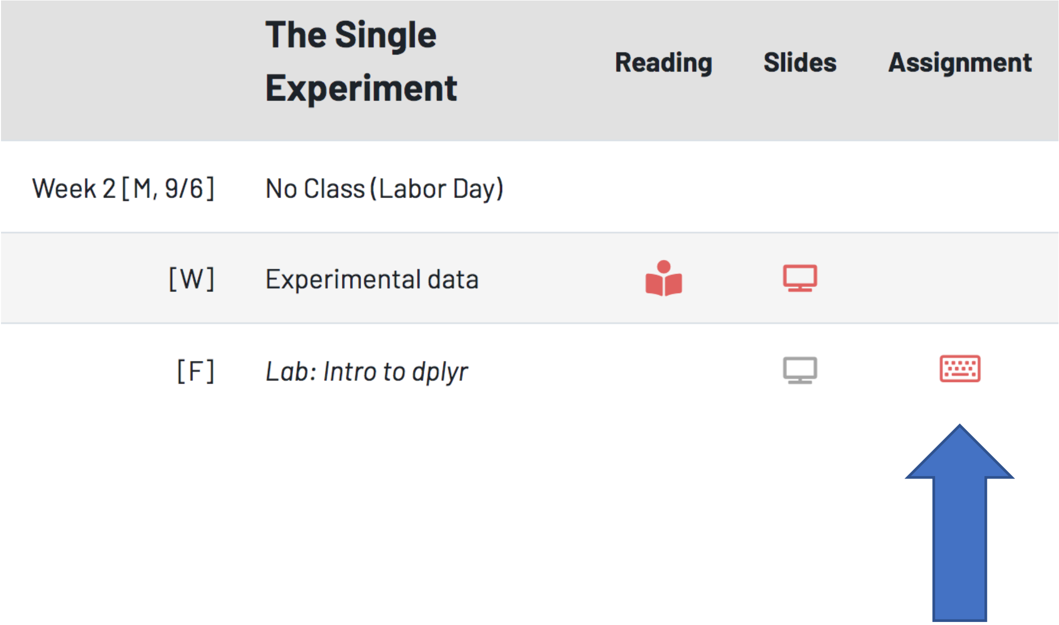
10 September 2021

Modern Research Methods

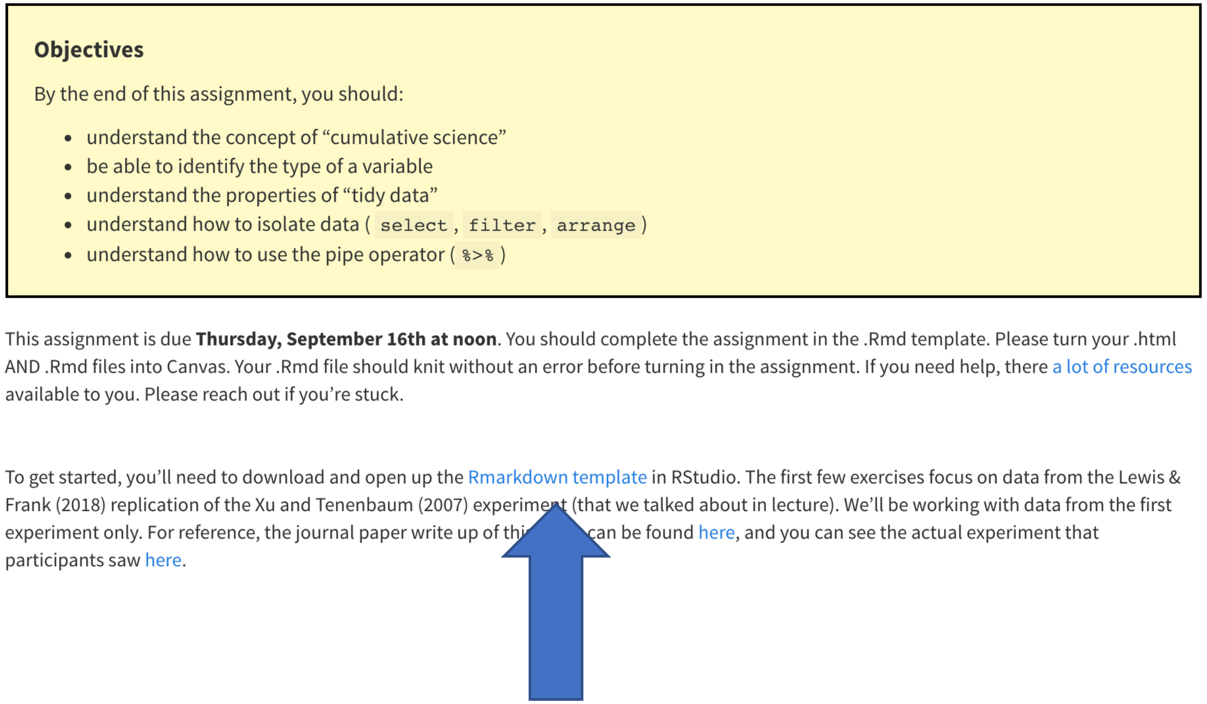


Artwork by
@allison_horst

On the class website, go to Schedule -> Assignment for today -> Rmarkdown template. Open up the template in RStudio.



	The Single Experiment	Reading	Slides	Assignment
Week 2 [M, 9/6]	No Class (Labor Day)			
[W]	Experimental data	📖	🖥	
[F]	*Lab: Intro to dplyr*		🖥	⌨



Objectives

By the end of this assignment, you should:

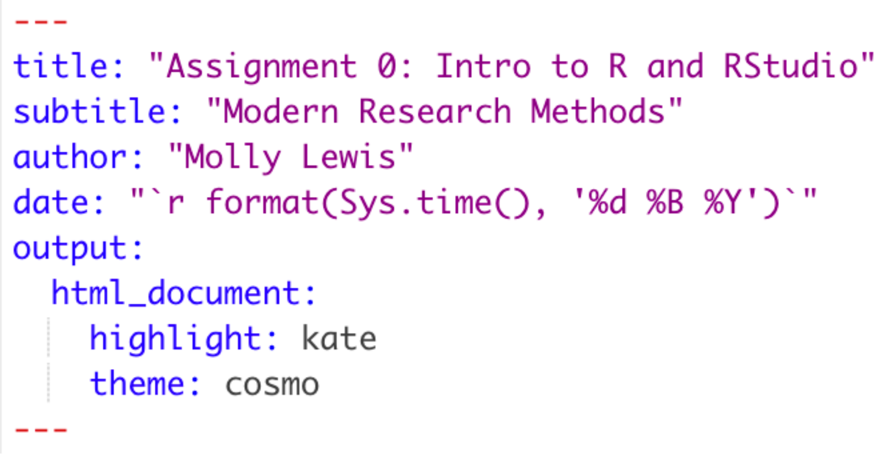
- understand the concept of "cumulative science"
- be able to identify the type of a variable
- understand the properties of "tidy data"
- understand how to isolate data (`select` , `filter` , `arrange`)
- understand how to use the pipe operator (`%>%`)

This assignment is due **Thursday, September 16th at noon**. You should complete the assignment in the .Rmd template. Please turn your .html AND .Rmd files into Canvas. Your .Rmd file should knit without an error before turning in the assignment. If you need help, there a lot of resources available to you. Please reach out if you're stuck.

To get started, you'll need to download and open up the Rmarkdown template in RStudio. The first few exercises focus on data from the Lewis & Frank (2018) replication of the Xu and Tenenbaum (2007) experiment (that we talked about in lecture). We'll be working with data from the first experiment only. For reference, the journal paper write up of thi can be found here, and you can see the actual experiment that participants saw here.

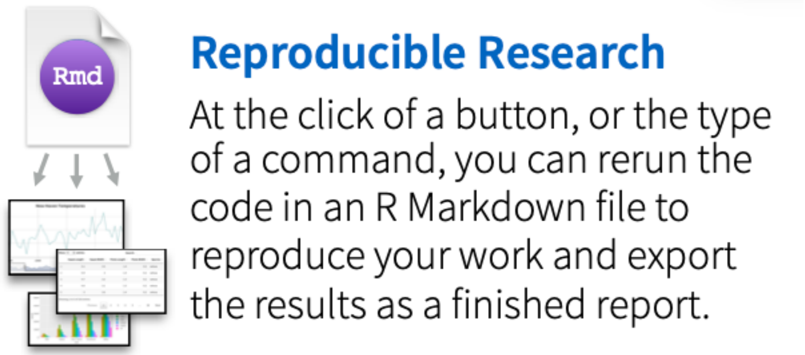
Rmarkdown

Interacting with R with an Rmarkdown notebook in RStudio



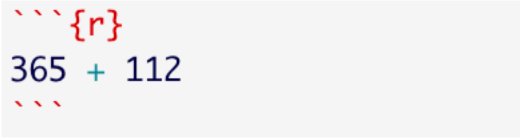
```
---
title: "Assignment 0: Intro to R and RStudio"
subtitle: "Modern Research Methods"
author: "Molly Lewis"
date: "`r format(Sys.time(), '%d %B %Y')`"
output:
  html_document:
    highlight: kate
    theme: cosmo
---
```

Header (YAML)



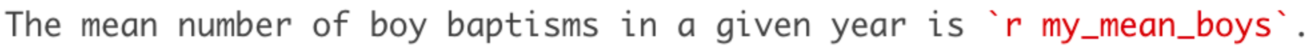
Reproducible Research

At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.



~~~
```{r}
365 + 112
```
~~~

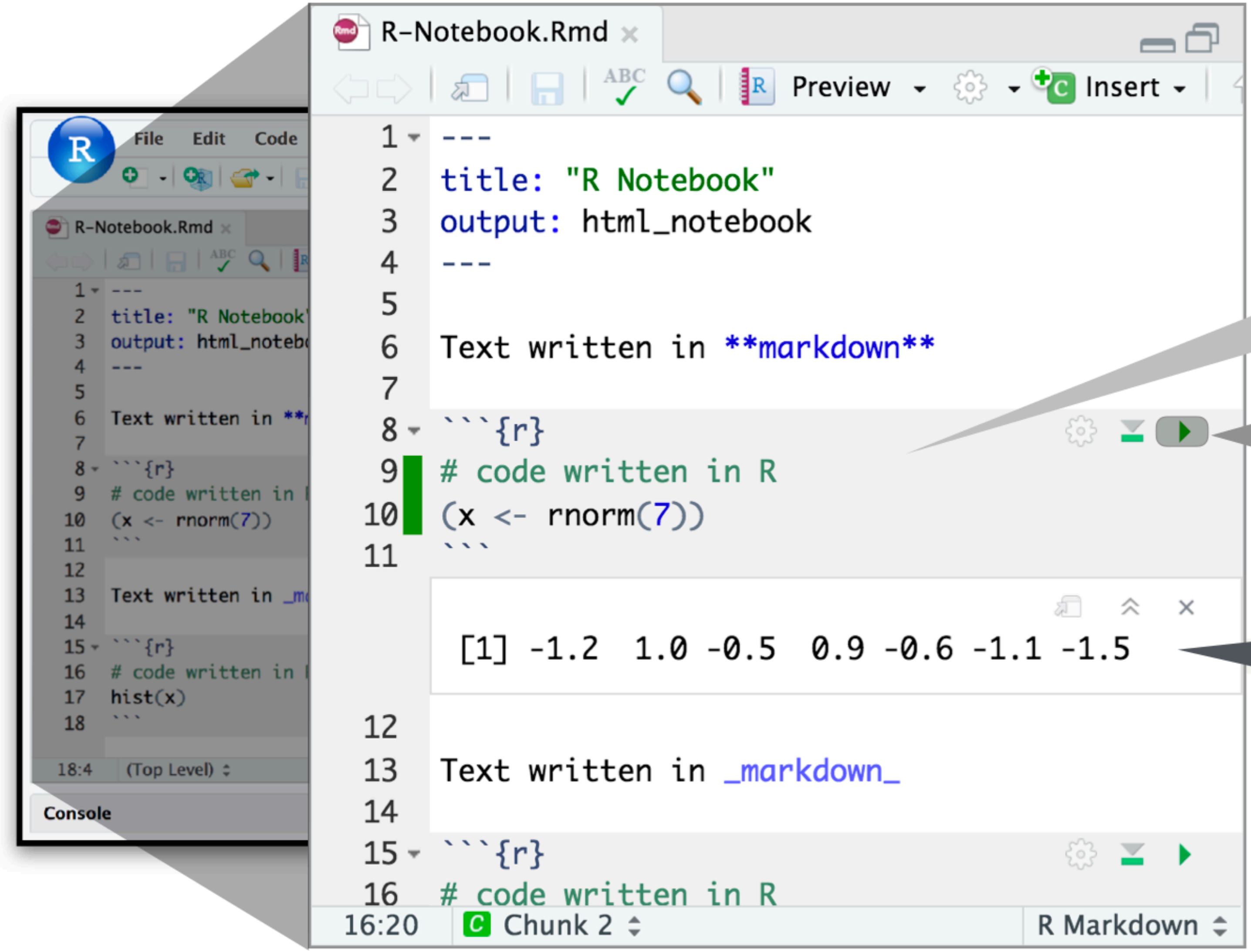
R chunks



```
The mean number of boy baptisms in a given year is `r my_mean_boys`.
```

Plain text (with inline R)

.Rmd -> “Knit” -> .html



~~~
---
title: "R Notebook"
output: html_notebook
---

Text written in **markdown**

```{r}
# code written in R
(x <- rnorm(7))
~~~
```

```
[1] -1.2  1.0 -0.5  0.9 -0.6 -1.1 -1.5
```

```

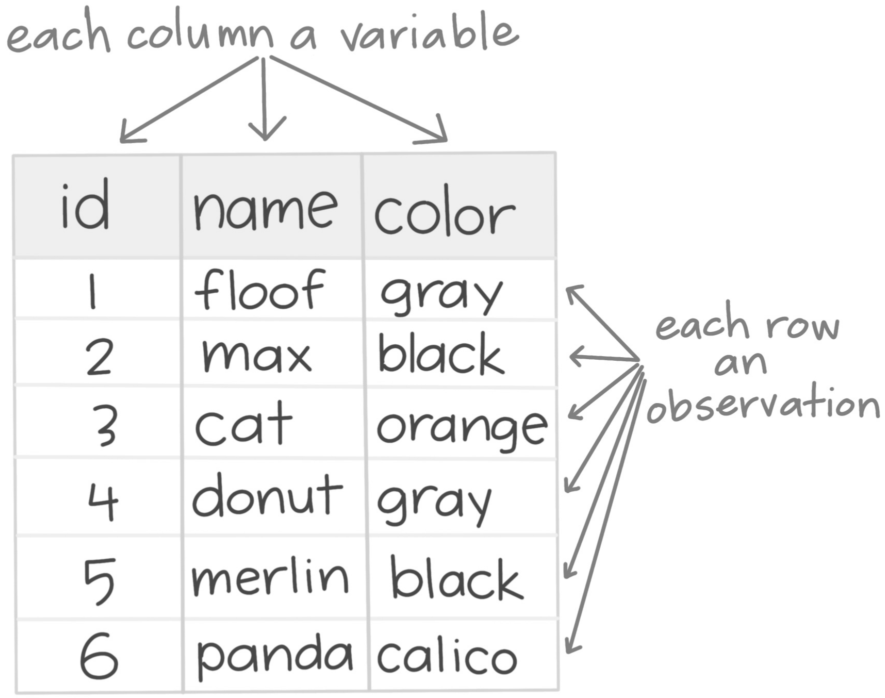
Text written in _markdown_

```{r}
# code written in R
hist(x)
```
```

Code goes in a chunk

Click to run code in chunk

Code result
```

> “TIDY DATA is a standard way of mapping the meaning of a dataset to its structure.”
>
> —HADLEY WICKHAM

## In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



| id | name | color |
|---|---|---|
| 1 | floof | gray |
| 2 | max | black |
| 3 | cat | orange |
| 4 | donut | gray |
| 5 | merlin | black |
| 6 | panda | calico |

Wickham, H. (2014). Tidy Data. Journal of Statistical Software 59 (10). DOI: 10.18637/jss.v059.i10

Artwork by
@allison_horst

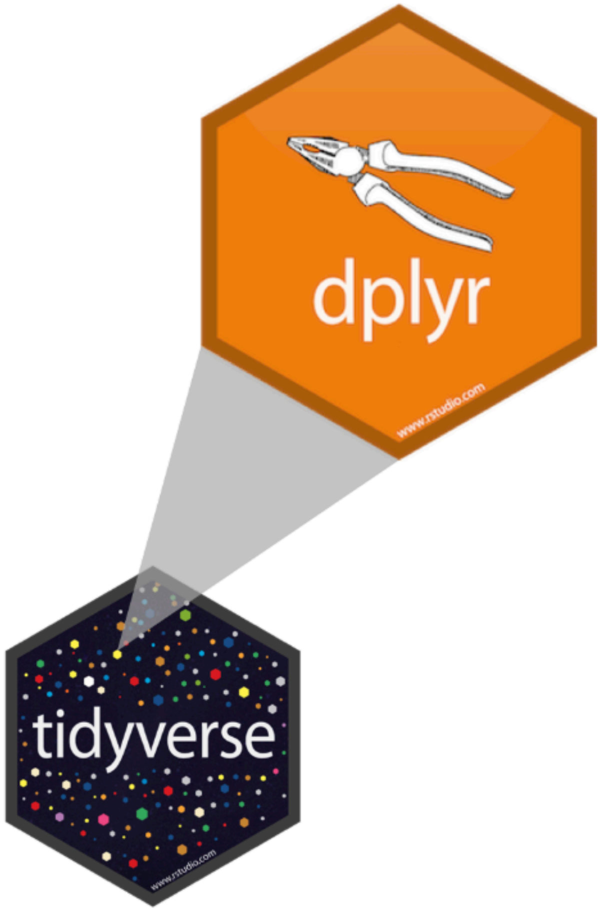
# How to isolate?

| 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 | … | … |

# Manipulating data



dplyr is organized around **verbs** that **manipulate** data frames

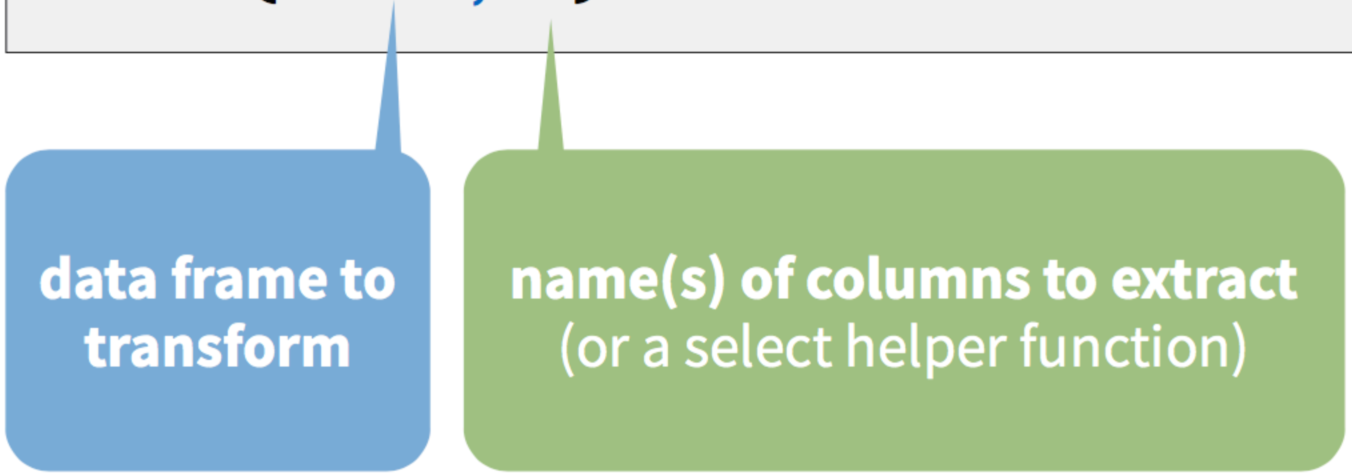
Isolating data:

- **`select()`** extracts columns
- **`filter()`** extracts rows
- **`arrange()`** reorders rows

# select()

Extract columns by name.

```
select(.data, …)
```



data frame to transform

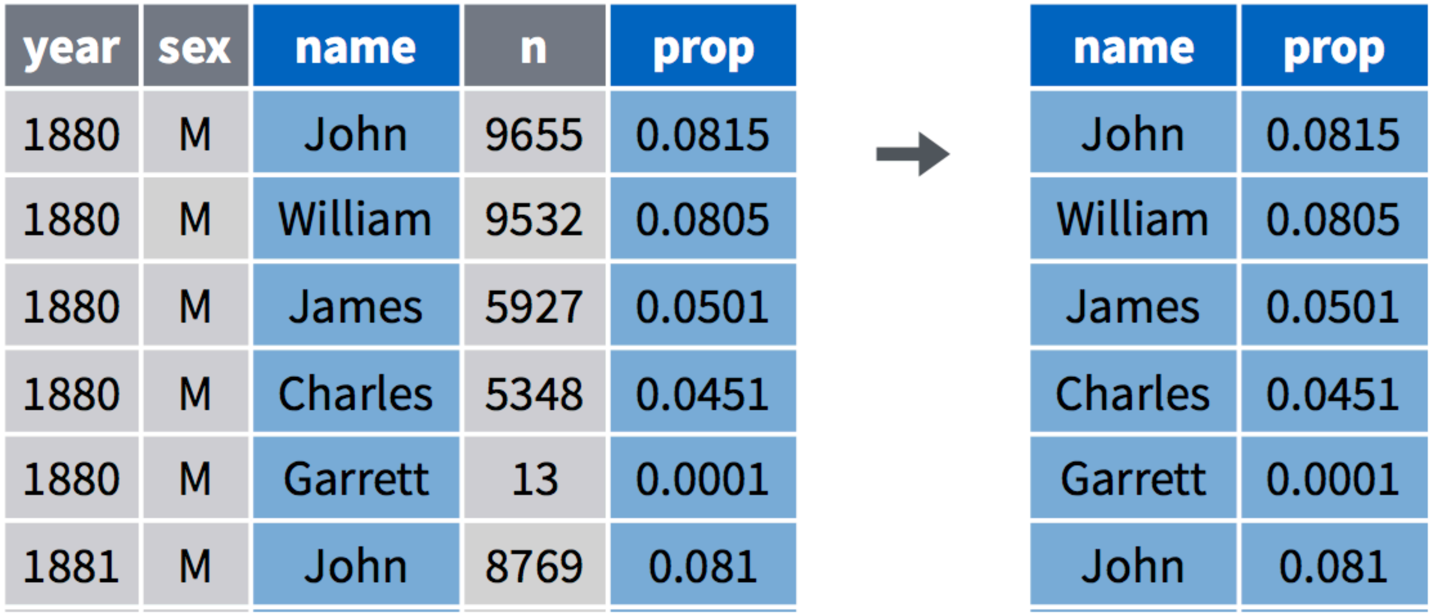
name(s) of columns to extract
(or a select helper function)

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

Exercise 1

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

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

01:00

```
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"))
```

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

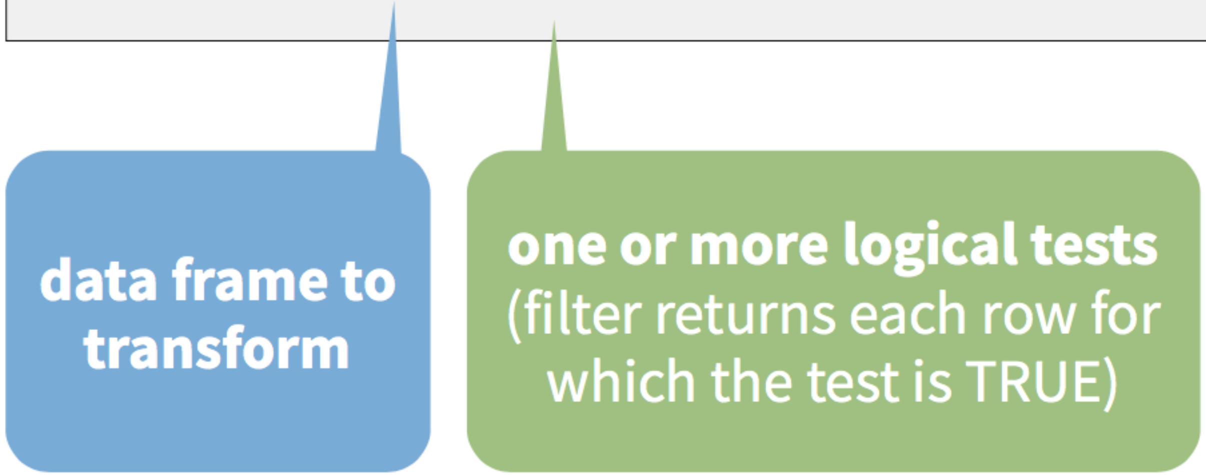
Which of these is NOT a way to select the **name** and **n** columns together?

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

select(babynames, name:n)

select(babynames, starts_with("n"))

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

filter()

Extract rows that meet logical criteria.

```
filter(.data, … )
```



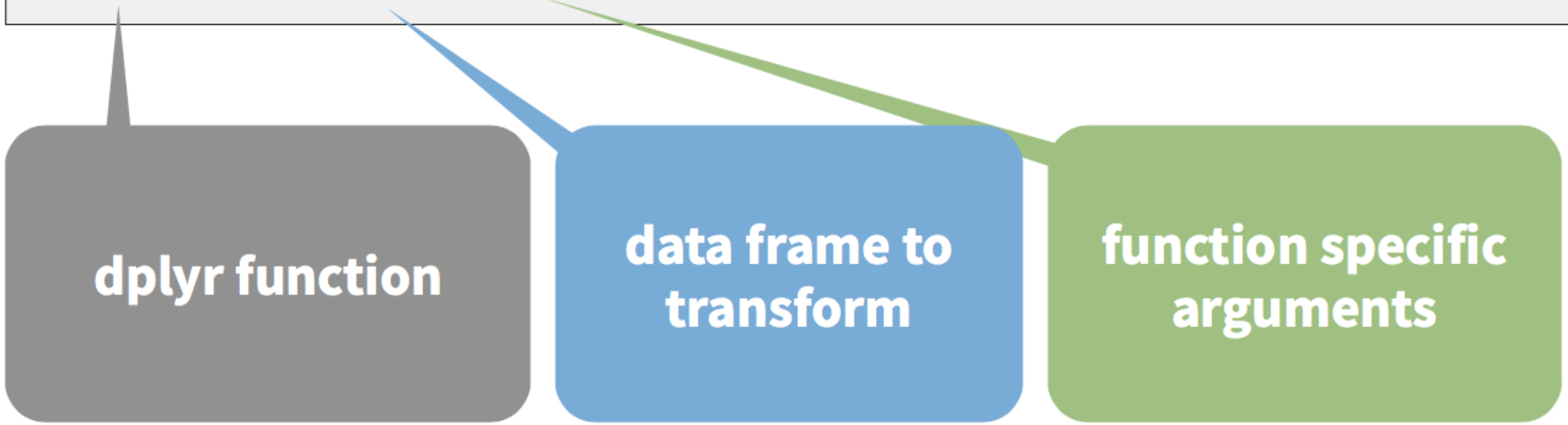
data frame to transform

one or more logical tests (filter returns each row for which the test is TRUE)

common syntax

Each function takes a data frame / tibble as its first argument and returns a data frame / tibble.

```
filter(.data, … )
```



dplyr function

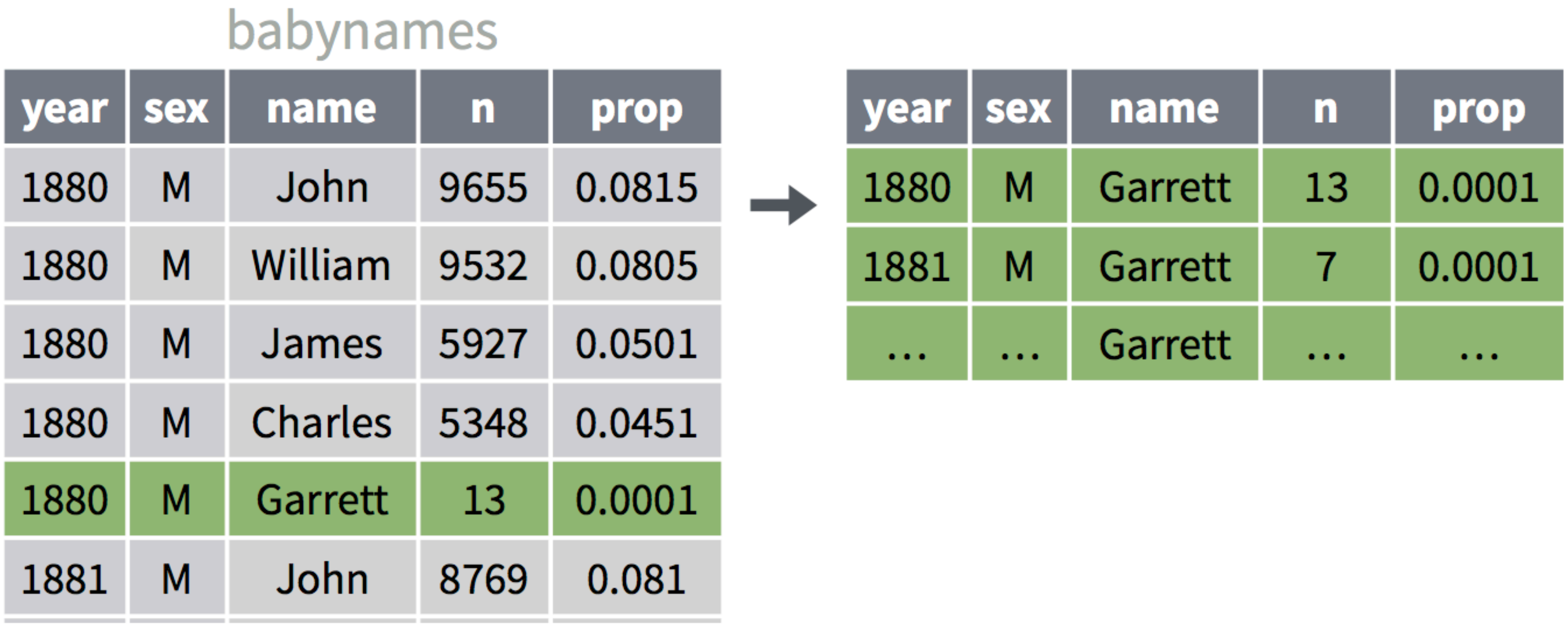
data frame to transform

function specific arguments

filter()

Extract rows that meet logical criteria.

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



babynames

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

year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
…	…	Garrett	…	…

filter()

Extract rows that meet logical criteria.

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

babynames

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

= sets
(returns nothing)

== tests if equal
(returns TRUE or FALSE)

Logical tests

?Comparison

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

Exercise 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")
```

Boolean operators

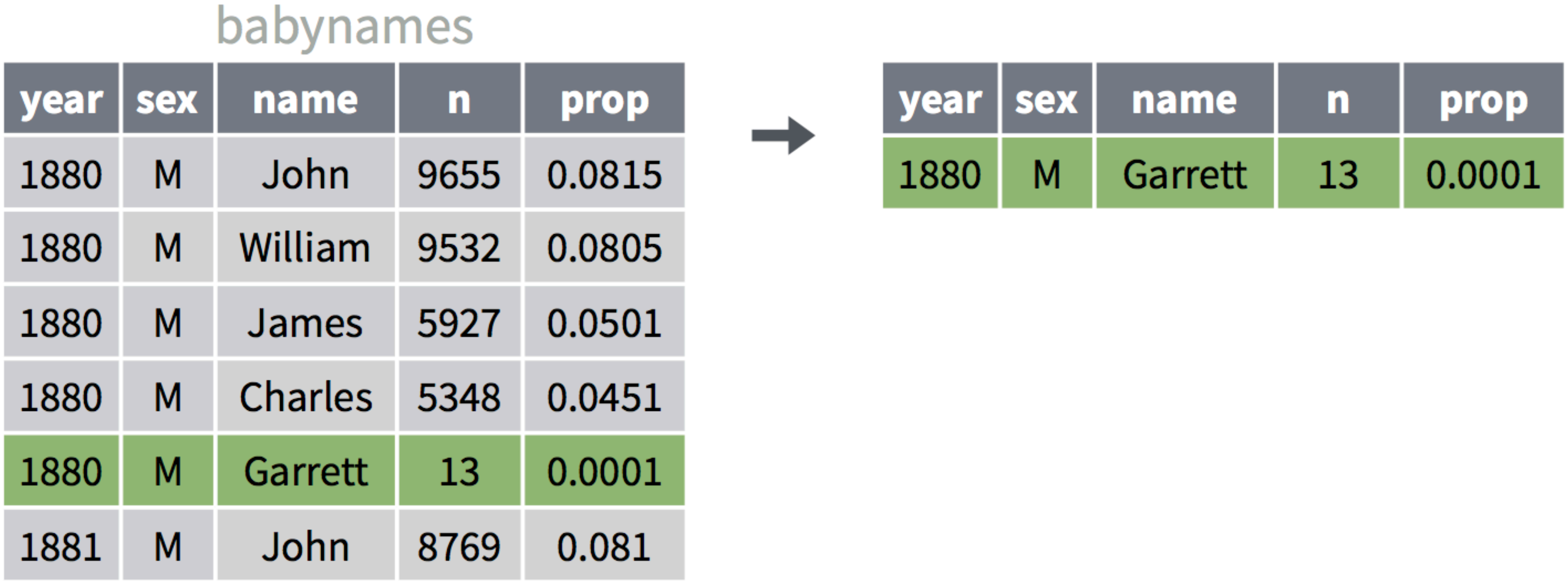
?base::Logic

<table>
<tr><td>a & b</td><td>and</td></tr>
<tr><td>a | b</td><td>or</td></tr>
<tr><td>xor(a,b)</td><td>exactly or</td></tr>
<tr><td>!a</td><td>not</td></tr>
</table>

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

Two more common mistakes

3. Collapsing multiple tests into one

```
filter(babynames, 10 < n < 20)
filter(babynames, 10 < n, n < 20)
```

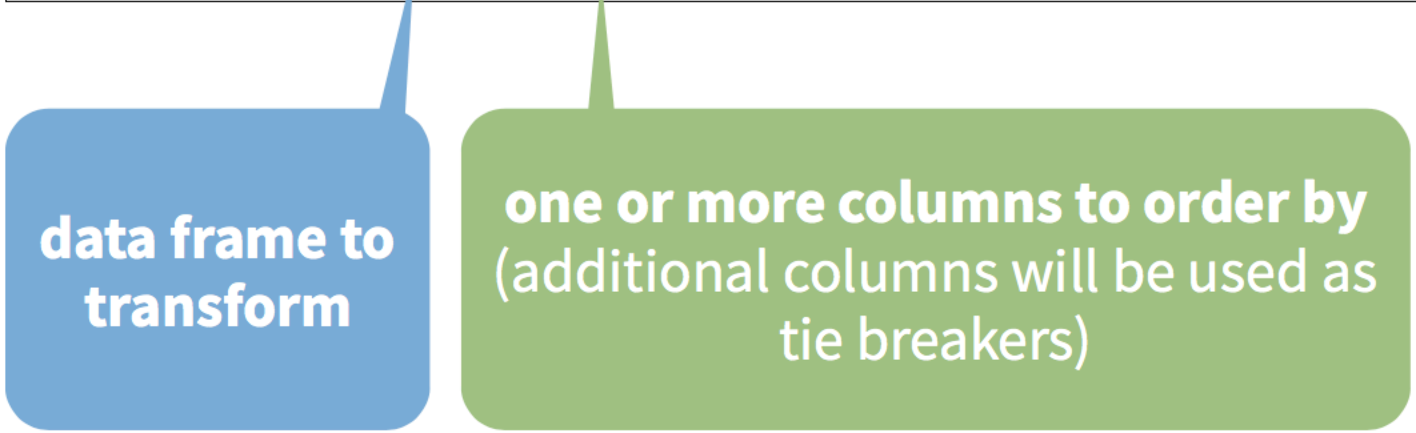
4. Stringing together many tests (when you could use %in%)

```
filter(babynames, n == 5 | n == 6 | n == 7 | n == 8)
filter(babynames, n %in% c(5, 6, 7, 8))
```

arrange()

Order rows from smallest to largest values.

```
arrange(.data, …)
```



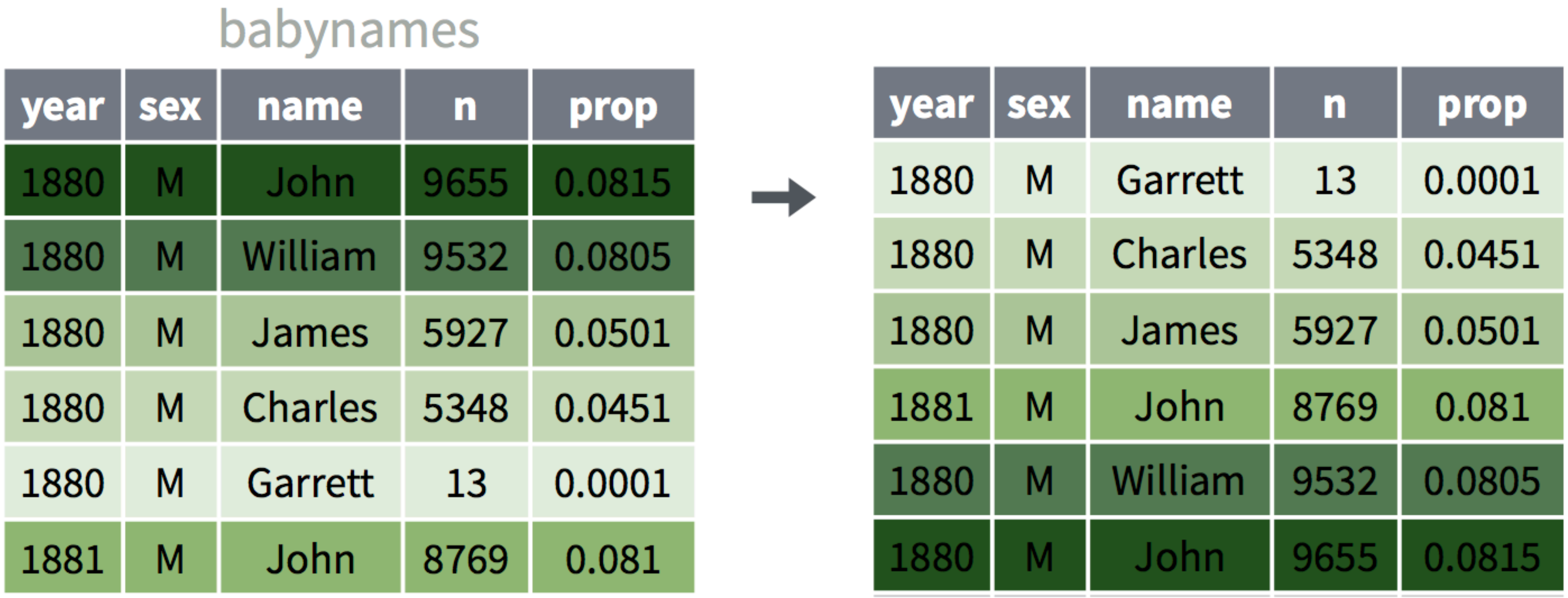
data frame to transform

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

Exercise 3

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
```

“pipe”

%>%

Turns code into **sentences** that read left to right

Steps

```r
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.

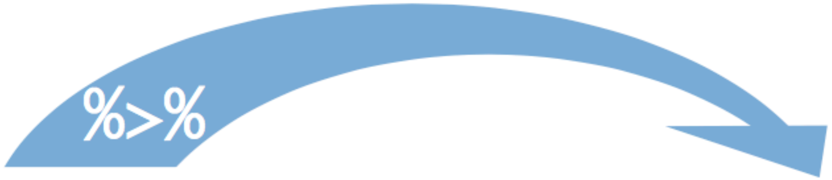
Steps

```r
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))
```

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

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

vs.

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

Shortcut to type %>%

Cmd + Shift + M (Mac)

Ctrl + Shift + M (Windows)

Exercise 4

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
```

Wrap-up

- Assignment 1: due next Thursday (Sept. 16th at noon)
- Turn in both .Rmd and .html file to Canvas
- Short 5 minute quiz in class next time – bring your laptop!
- Office Hours:
  - Roderick M 3:30-5:30 (virtual - email)
  - Molly W 2:45-4:45 (in person or virtual)

Acknowledgements

Slides adapted from datasciencebox and Rstudio by CC