

# Intro to dplyr

24 January 2020

*Modern Research Methods*

Course Website: <https://cumulativescience.netlify.com/>

# babynames




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

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



## babynames



<b>year</b>	<b>sex</b>	<b>name</b>	<b>n</b>	<b>prop</b>
<dbl>	<chr>	<chr>	<int>	<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



## How to isolate?

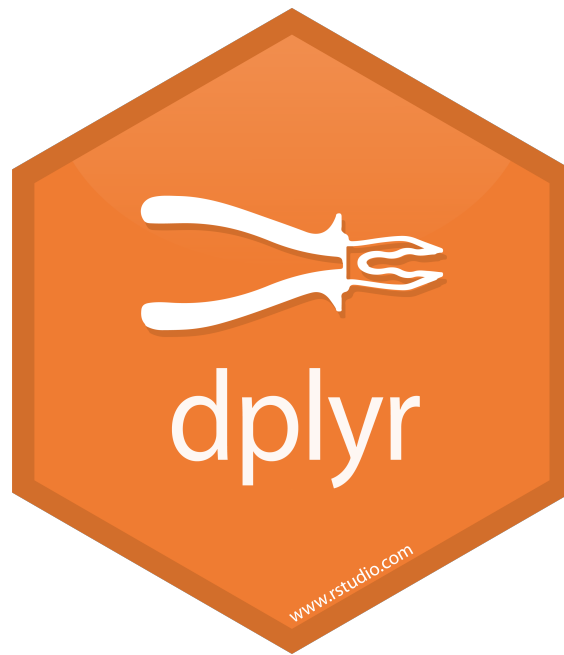
year	sex	name	n	prop
1880	M	John	9655	0.0815
1880	M	William	9532	0.0805
1880	M	James	5927	0.0501
1880	M	Charles	5348	0.0451
1880	M	Garrett	13	0.0001
1881	M	John	8769	0.081
1881	M	William	8524	0.0787
1881	M	James	5442	0.0503
1881	M	Charles	4664	0.0431
1881	M	Garrett	7	0.0001
1881	M	Gideon	7	0.0001



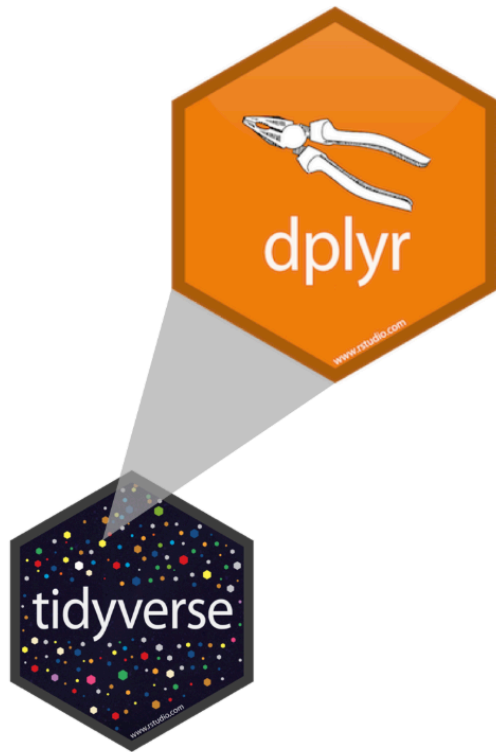
year	sex	name	n	prop
1880	M	Garrett	13	0.0001
1881	M	Garrett	7	0.0001
...	...	Garrett	...	...



# Transform Data with



**dplyr** is based on the concepts of functions as verbs that manipulate data frames.



- + filter: pick rows matching criteria
- + slice: pick rows using index(es)
- + select: pick columns by name
- + pull: grab a column as a vector
- + arrange: reorder rows
- + mutate: add new variables
- + distinct: filter for unique rows
- + sample\_n / sample\_frac: randomly sample rows
- + summarise: reduce variables to values
- + ... (many more)



# Isolating data

**select()** - extract **variables**

**filter()** - extract **cases**

**arrange()** - reorder **cases**



# Things to know about dplyr functions

- First argument is *always* a data frame
- Subsequent arguments say what to do with that data frame
- Always return a data frame
- Don't modify in place



`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



1. Go to the course website and open Assignment A1:  
<https://cumulativescience.netlify.com>

2. Go to R Cloud and open up Assignment A1:  
<https://rstudio.cloud/>

## ***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()



# 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)



# 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

<code>x &lt; y</code>	Less than
<code>x &gt; y</code>	Greater than
<code>x == y</code>	Equal to
<code>x &lt;= y</code>	Less than or equal to
<code>x &gt;= y</code>	Greater than or equal to
<code>x != y</code>	Not equal to
<code>x %in% y</code>	Group membership
<code>is.na(x)</code>	Is NA
<code>!is.na(x)</code>	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")
```



# 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

<code>a &amp; b</code>	and
<code>a   b</code>	or
<code>xor(a,b)</code>	exactly or
<code>!a</code>	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



# 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





## ***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
```



$\%0 > \%0$



# 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.

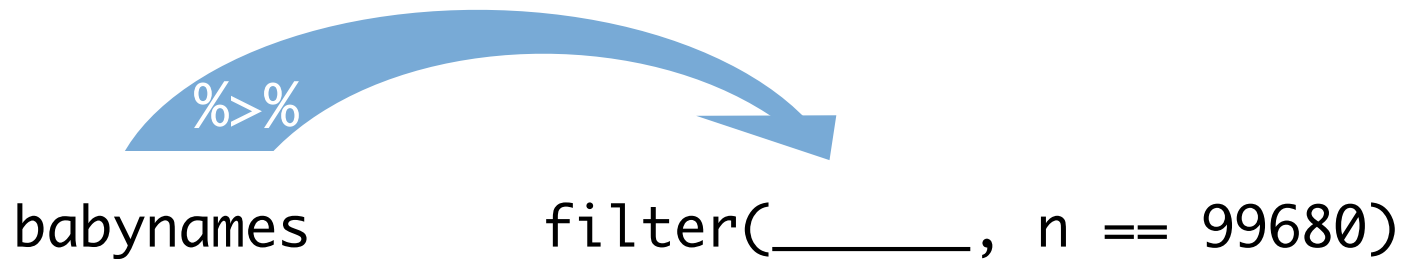
# Steps

```
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 %>%



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
```

# Assignment A1

- Due next Thursday (Jan. 30<sup>th</sup> at noon)
- Turn in both .Rmd and .html file to Canvas
- You are welcomed, and encouraged, to work with each other on the problems. But, you must turn in your own work.

# How to get help

- Check out the readings
- Look at the cheatsheets – linked on course website under “resources”
- Look at the help files
- Often it's a lot more pleasant an experience to get your questions answered in person. Make use of the instructors' office hours, we're here to help!
- Or, email us (please email both of us):
  - [mollylewis@cmu.edu](mailto:mollylewis@cmu.edu)
  - [jaeahk@andrew.cmu.edu](mailto:jaeahk@andrew.cmu.edu)

# How to get help

- Give your question context from course concepts not course assignments.
  - Good context: "I have a question on filtering"
  - Bad context: "I have a question on Assignment 1 Exercise 9"
- Where appropriate, provide links to specific files on Rstudio Cloud and the specific line number in the body of your email. This will help your helper understand your question.
- Be precise in your description:
  - Good description: "I am getting the following error and I'm not sure how to resolve it - Error: could not find function "read\_csv" "
  - Bad description: "R giving errors, help me! Aaaarrrrrgh!"