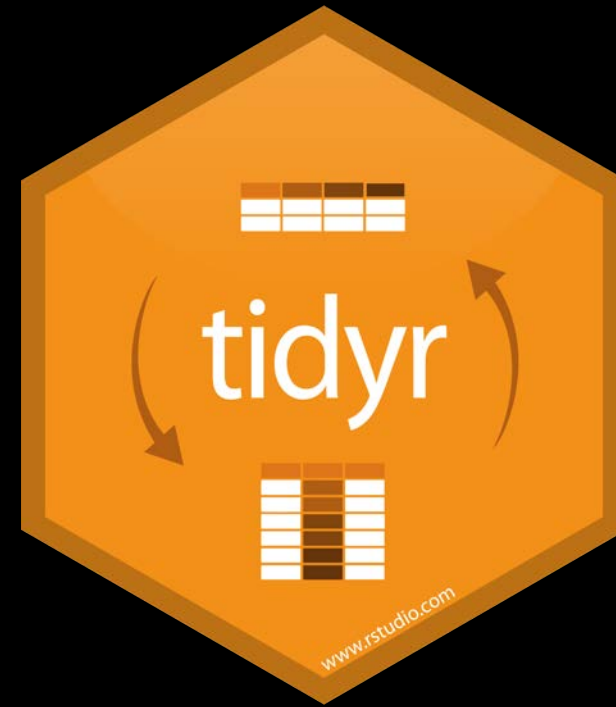


ANT 6973: DATA VISUALIZATION AND EXPLORATION

DATA MANIPULATION, PART 1

PACKAGES FOR WORKING WITH DATA



tidyr



dplyr

Both are
part of core



```
library("tidyverse")
```

Transform Data with



ACTIVITIES

- Open **babynames-manip.Rmd** and follow along.


babynames



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

```
# 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

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	Fernando	8	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	Fernando	6	0.0001
1881	M	Gideon	7	0.0001



year	sex	name	n	prop
1880	M	Fernando	8	0.0001
1881	M	Fernando	6	0.0001
...	...	Fernando

dplyr



A package that transforms data.
dplyr implements a *grammar* for
transforming tabular data.

SINGLE TABLE VERBS



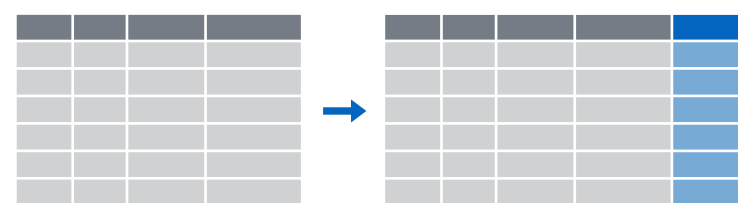
Extract variables with `select()`



Extract cases with `filter()`



Arrange cases with `arrange()`



Make new variables with `mutate()`



Make tables of summaries with `summarise()`
along with `group_by()`

select()

SELECT()

Extract columns by name.



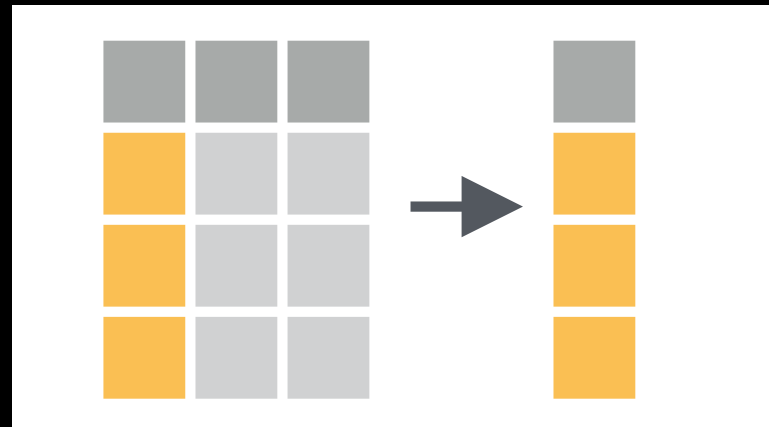
```
select(.data, ...)
```

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	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016167



name	prop
Mary	0.072383
Anna	0.026678
Emma	0.020521
Elizabeth	0.019865
Minnie	0.017888
Margaret	0.016167

SELECT()

Extract columns by name.



```
babynames %>% # Same but with pipe  
select(name, prop)
```

`babynames`

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016167



name	prop
Mary	0.072383
Anna	0.026678
Emma	0.020521
Elizabeth	0.019865
Minnie	0.017888
Margaret	0.016167

ACTIVITY 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()

You can rename on the fly



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

babynames

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016167



name	popularity
Mary	0.07238359
Anna	0.02667896
Emma	0.02052149
Elizabeth	0.01986579
Minnie	0.01788843
Margaret	0.0161672

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)) # x1, x2, etc.
```


select() helpers

Data Transformation with dplyr Cheat Sheet

RStudio

dplyr functions work with pipes and expect tidy data. In tidy data:

- Each **variable** is in its own **column**
- Each **observation**, or **case**, is in its own **row**

pipes
x %>% f(y)
becomes f(x, y)

Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

- summarise(data, ...)**
Compute table of summaries. Also **summarise_()**
`summarise(mtcars, avg = mean(mpg))`
- count(x, ..., wt = NULL, sort = FALSE)**
Count number of rows in each group defined by the variables in ... Also **tally()**
`count(iris, Species)`

Variations

- summarise_all()** - Apply funs to every column.
- summarise_at()** - Apply funs to every column.
- summarise_if()** - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to created a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

- `mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))`
- group_by(data, ..., add = FALSE)**
Returns copy of table grouped by ...
`g_iris <- group_by(iris, Species)`
- ungroup(x, ...)**
Returns ungrouped copy of table.
`ungroup(g_iris)`

Manipulate Cases

Extract Cases

Row functions return a subset of rows as a new table. Use a variant that ends in **_()** for non-standard evaluation friendly code.

- filter(data, ...)**
Extract rows that meet logical criteria. Also **filter_()** `filter(iris, Sepal.Length > 7)`
- distinct(data, ..., keep_all = FALSE)**
Remove rows with duplicate values. Also **distinct_()** `distinct(iris, Species)`
- sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, env = parent.frame())**
Randomly select fraction of rows.
`sample_frac(iris, 0.5, replace = TRUE)`
- sample_n(tbl, size, replace = FALSE, weight = NULL, env = parent.frame())**
Randomly select size rows.
`sample_n(iris, 10, replace = TRUE)`
- slice(data, ...)**
Select rows by position. Also **slice_()**
`slice(iris, 10:15)`
- top_n(x, n, wt)**
Select and order top n entries (by group if grouped data). `top_n(iris, 5, Sepal.Width)`

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%		xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

Arrange Cases

- arrange(data, ...)**
Order rows by values of a column (low to high), use with **desc()** to order from high to low.
`arrange(mtcars, mpg)`
`arrange(mtcars, desc(mpg))`

Add Cases

- add_row(data, ..., before = NULL, .after = NULL)**
Add one or more rows to a table.
`add_row(faithful, eruptions = 1, waiting = 1)`

Manipulate Variables

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)	

Make New Variables

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

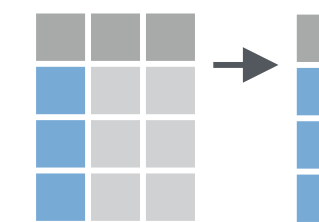
- mutate(data, ...)**
Compute new column(s).
`mutate(mtcars, gpm = 1/mpg)`
- transmute(data, ...)**
Compute new column(s), drop others.
`transmute(mtcars, gpm = 1/mpg)`
- mutate_all(tbl, funs, ...)**
Apply funs to every column. Use with **funs()**. `mutate_all(faithful, funs(log(), log2(), ...))`
- mutate_at(tbl, cols, funs, ...)**
Apply funs to specific columns. Use with **funs()** and the helper functions for **select()**.
`mutate_at(iris, ~Species, funs(log(), log2(), ...))`
- mutate_if(tbl, predicate, funs, ...)**
Apply funs to all columns of one type. Use with **funs()**.
`mutate_if(iris, is.numeric, funs(log(), log2(), ...))`
- add_column(data, ..., before = NULL, .after = NULL)**
Add new column(s).
`add_column(mtcars, new = 1:32)`
- rename(data, ...)**
Rename columns.
`rename(iris, Length = Sepal.Length)`

RStudio® is a trademark of RStudio, Inc. • [© RStudio](#) • [info@rstudio.com](#) • 844-448-1212 • [rstudio.com](#)

Learn more with `browseVignettes(package = c("dplyr", "tibble"))` • dplyr 0.5.0 • tibble 1.2.0 • Updated: 11/16

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)
ends_with(match)
matches(match)

num_range(prefix, range)
one_of(...)
starts_with(match)

:, e.g. mpg:cyl
-, e.g. -Species

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"))
```

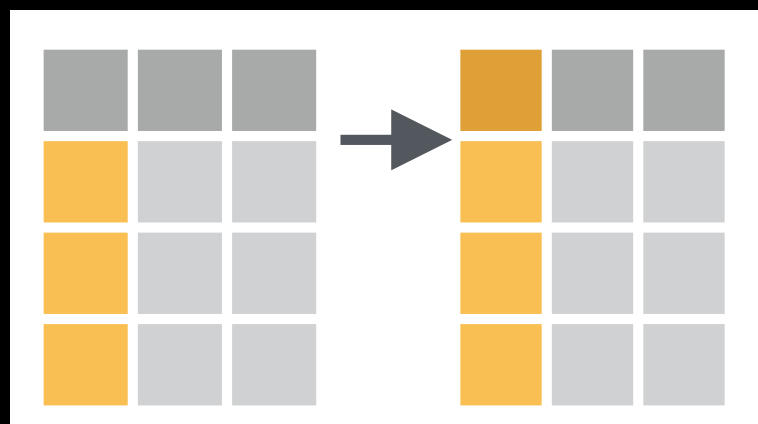
```
select(babynames, ends_with("n"))
```

SELECT()'S COUSINS

For manipulating variables/columns

RENAME()

Rename columns without extracting.



```
rename(babynames, popularity = prop)
```

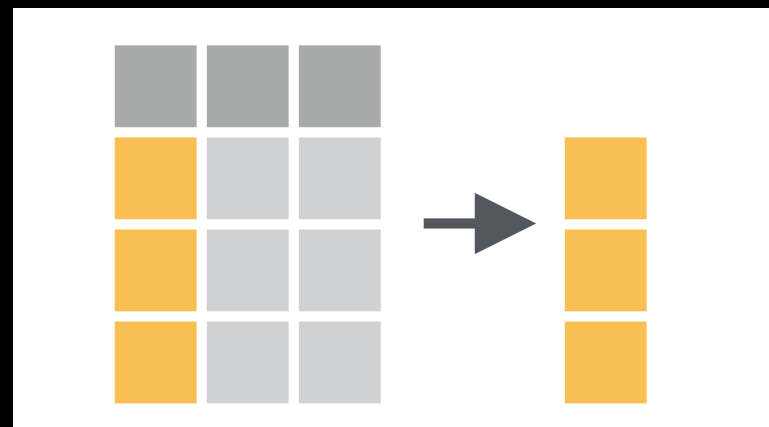
year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016172



year	sex	name	n	popularity
1880	F	Mary	7065	0.07238359
1880	F	Anna	2604	0.02667896
1880	F	Emma	2003	0.02052149
1880	F	Elizabeth	1939	0.01986579
1880	F	Minnie	1746	0.01788843
1880	F	Margaret	1578	0.016172

PULL()

Extract column values only.



```
pull(babynames, n)
```

```
# Does same thing as babynames$n
```

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016117



```
[1] 7065 2604 2003 1939 ...
```

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 == "Fernando")
```

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	Fernando	8	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop
1880	M	Fernando	8	0.0001
1881	M	Fernando	6	0.0001
...	...	Fernando

FILTER()

Extract rows that meet logical criteria.

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

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	Fernando	8	0.0001
1881	M	John	8769	0.081

= sets

(returns nothing)

== tests if equal

(returns TRUE or

BASIC R: OPERATORS

COMPARISON OPERATORS

OPERATOR	DESCRIPTION
<code>x < y</code>	LESS THAN
<code>x <= y</code>	LESS THAN OR EQUAL TO
<code>x > y</code>	GREATER THAN
<code>x >= y</code>	GREATER THAN OR EQUAL TO
<code>x == y</code>	EXACTLY EQUAL TO
<code>x != y</code>	NOT EQUAL TO
<code>x %in% y</code>	GROUP MEMBERSHIP
<code>is.na(x)</code>	IS NA

?Comparison

ACTIVITY 2

- See if you can use the logical operators to manipulate babynames to show:
 1. All of the names where `prop` is greater than or equal to 0.08
 2. All of the children named "Sea"
 3. All of the names that have a missing value for `n`
(Hint: this should return an empty data set).

```
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 == "Fernando", 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	Fernando	8	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop
1880	M	Fernando	8	0.0001

BASIC R: OPERATORS

LOGICAL OPERATORS

OPERATOR	DESCRIPTION
<code>!x</code>	NOT
<code>x y</code>	OR
<code>x & y</code>	AND
<code>xor(x, y)</code>	EXACTLY OR (FALSE IF BOTH ARE TRUE)

?base::Logic

FILTER()

Extract rows that meet *every* logical criteria.

```
filter(babynames, name == "Fernando" & 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	Fernando	8	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop
1880	M	Fernando	8	0.0001

ACTIVITY 3

- Use logical operators to alter the code below to return only the rows that contain:
 1. Girls named Sea
 2. Names that were used by exactly 5 or 6 children in 1880
 3. 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))
```

FILTER()'S COUSINS

For manipulating cases/rows

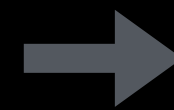
SLICE()

Extract rows based on *position* rather than criteria.

```
slice(babynames, 1:2)
```

babynames

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016167



year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678

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	Fernando	8	0.0001
1881	M	John	8769	0.081



year	sex	name	n	prop
1880	M	Fernando	8	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

ACTIVITY 4

- Arrange `babynames` by `n`, and add `name` as a second (tie breaking) variable by which to arrange.
- Can you tell what the smallest value of `n` is?

```
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	Abdulahadi	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	Fernando	8	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	Fernando	8	0.0001

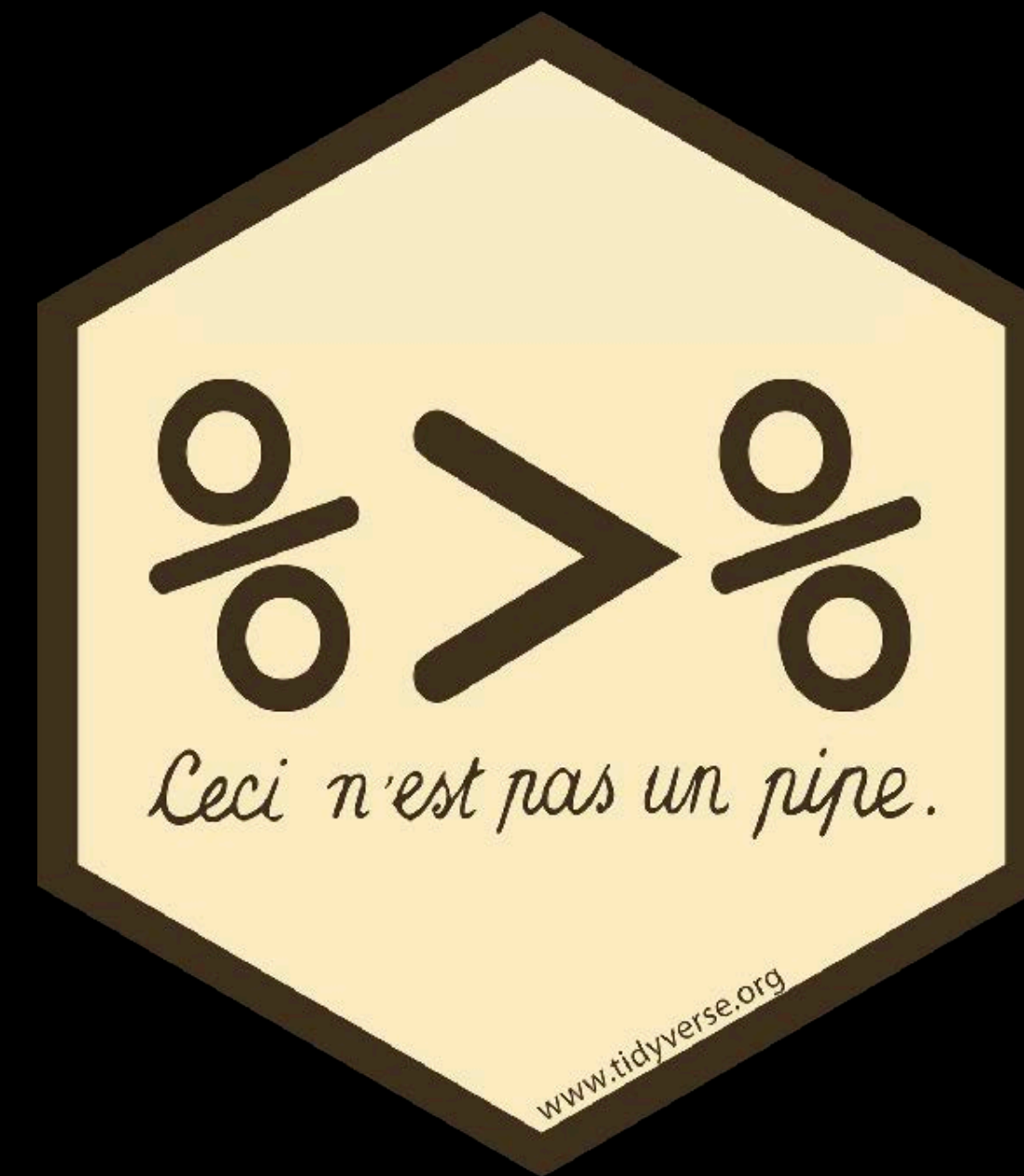
ACTIVITY 5

- Use `desc()` to find the names with the highest prop.
- Then, use `desc()` to find the names with the highest n.

#		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					

#		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					

PIPES



Here's where they become really useful!

MULTI-STEP PROCEDURE

```
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.

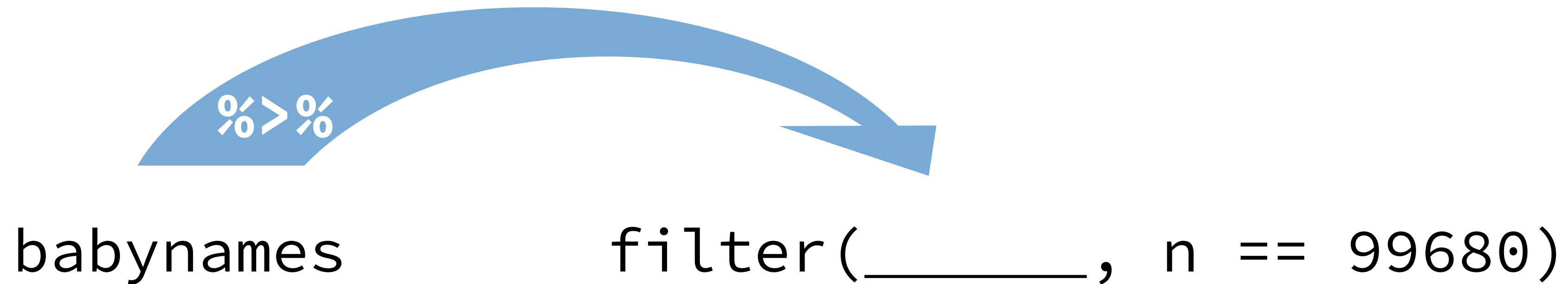
MULTI-STEP PROCEDURE

```
boys_2015 <- filter(babynames, year == 2015, sex == "M")  
boys_2015 <- select(boys_2015, name, n)  
boys_2015 <- arrange(boys_2015, desc(n))  
boys_2015
```

MULTI-STEP PROCEDURE

```
arrange(select(filter(babynames, year == 2015,  
  sex == "M"), name, n), desc(n))
```

RECALL: THE PIPE OPERATOR %>%



Passes result on left into first argument of function on right.

So, for example, these do the same thing.

```
filter(babynames, n == 99680)
```

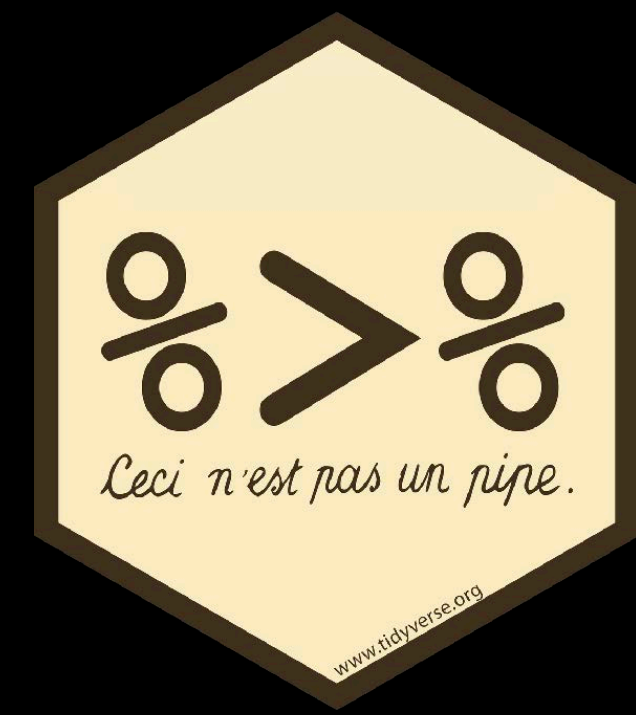
```
babynames %>% filter(n == 99680)
```

MULTI-STEP PROCEDURE

```
babynames
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))
```


PIPES



- Shortcut to type %>%

Cmd	+	Shift	+	M	(Mac)
Ctrl	+	Shift	+	M	(Windows)

ACTIVITY 6

- Use `%>%` to write the following sequence of functions:
 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.

```
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
```

mutate()

MUTATE()

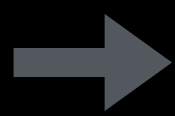
Create new columns.

You can do basic arithmetic

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

babynames

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888

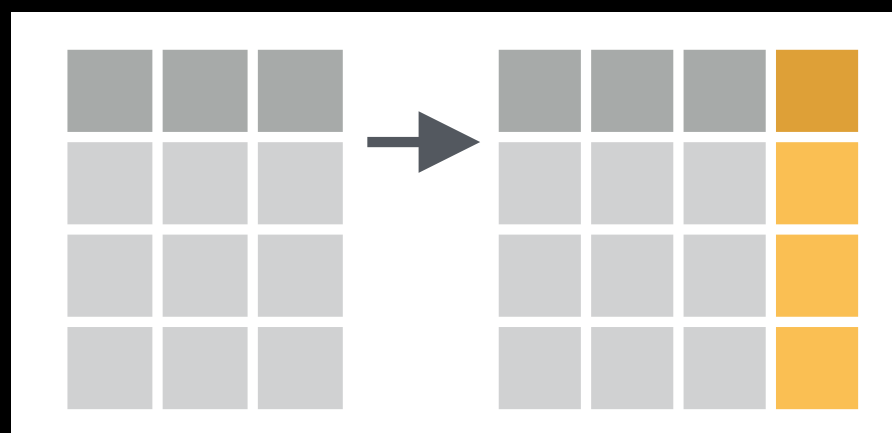


year	sex	name	n	prop	percent
1880	F	Mary	7065	0.072383	7.238359
1880	F	Anna	2604	0.026678	2.667896
1880	F	Emma	2003	0.020521	2.052149
1880	F	Elizabeth	1939	0.019865	1.986579
1880	F	Minnie	1746	0.017888	1.788843

MUTATE()

Create new columns.

Operations can be nested



```
babynames %>%
```

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

babynames

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888



year	sex	name	n	prop	percent
1880	F	Mary	7065	0.072383	7.24
1880	F	Anna	2604	0.026678	2.67
1880	F	Emma	2003	0.020521	2.05
1880	F	Elizabeth	1939	0.019865	1.99
1880	F	Minnie	1746	0.017888	1.79

ASIDE: ROUND()

- Round a number to a specified number of decimal digits (0 by default)

```
x <- c(1.2, 1/3, 10.01)
```

```
round(x)
```

```
[1] 1 0 10
```

```
round(x, digits = 2)
```

```
[1] 1.20 0.33 10.01
```

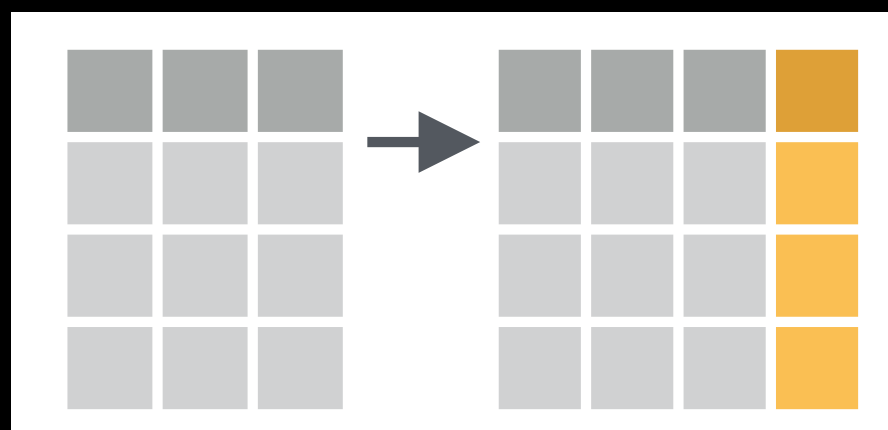
When using a function in `mutate()` the argument will be a column name from the data

```
babynames %>%
```

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


MUTATE()

Create new columns.



```
babynames %>%
```

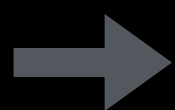
```
  mutate(percent = prop * 100,
```

```
         pct_rnd = round(percent, 2))
```

Newly created variables are available immediately for further manipulation

`babynames`

year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888



year	sex	name	n	prop	percent	pct_rnd
1880	F	Mary	7065	0.072383	7.238359	7.2
1880	F	Anna	2604	0.026678	2.667896	2.7
1880	F	Emma	2003	0.020521	2.052149	2.1
1880	F	Elizabeth	1939	0.019865	1.986579	2
1880	F	Minnie	1746	0.017888	1.788843	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() - $x > \text{right} \ \& \ x < \text{left}$

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()

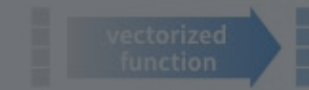
dplyr::recode() - Vectorized switch()

dplyr::recode_factor() - Vectorized switch() for factors

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() - $x > \text{right} \ \& \ x < \text{left}$

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

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

Row names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

rownames_to_column()
Move row names into col.
`a <- rownames_to_column(in, var = "C")`

column_to_rownames()
Move col in row names.
`column_to_rownames(a, var = "C")`

Also **has_rownames()**, **remove_rownames()**

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.

Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.

Use **suffix** to specify suffix to give to duplicate column names.

Use **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.

min_rank()

A convenient 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
```

ACTIVITY 7

- Use `mutate()` and `min_rank()` to rank each row in `babynames` from largest `n` to smallest `n`, and filter to extract the top 10 ranking names in the entire data set.
- Do the same but use `prop` rather than `n` to rank the names.
- What differences do you see in these rankings?

```
babynames %>%
```

```
  mutate(rank = min_rank(desc(n))) %>%
```

```
  filter(rank <= 10)
```

	year	sex	name	n	prop	rank
	<dbl>	<chr>	<chr>	<int>	<dbl>	<int>
1	1947	F	Linda	99686	0.0548	1
2	1947	M	James	94756	0.0510	3
3	1947	M	Robert	91642	0.0493	5
4	1948	F	Linda	96209	0.0552	2
5	1948	M	James	88588	0.0497	9
6	1949	F	Linda	91016	0.0518	6
7	1954	M	Michael	88514	0.0428	10
8	1956	M	Michael	90620	0.0423	7
9	1957	M	Michael	92695	0.0424	4
10	1958	M	Michael	90520	0.0420	8


```
babynames %>%
```

```
  mutate(rank = min_rank(desc(prop))) %>%
```

```
  filter(rank <= 10)
```

	year	sex	name	n	prop	rank
	<dbl>	<chr>	<chr>	<int>	<dbl>	<int>
1	1880	M	John	9655	0.0815	1
2	1880	M	William	9532	0.0805	3
3	1881	M	John	8769	0.0810	2
4	1881	M	William	8524	0.0787	5
5	1882	M	John	9557	0.0783	6
6	1882	M	William	9298	0.0762	8
7	1883	M	John	8894	0.0791	4
8	1884	M	John	9388	0.0765	7
9	1885	M	John	8756	0.0755	10
10	1886	M	John	9026	0.0758	9

lead() and lag()

Find the "next" or "previous" values.

```
1:10
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
lead(1:10)
```

```
[1]  2  3  4  5  6  7  8  9 10 NA
```

```
lag(1:10)
```

```
[1] NA  1  2  3  4  5  6  7  8  9
```

ACTIVITY 7

- Use `mutate()` and `lag()` to find the year-to-year change with the largest increase in `n` for your name and sex.

```
babynames %>%
```

```
  filter(name == "Fernando" & sex == "M") %>%
```

```
  mutate(n_change = n - lag(n)) %>%
```

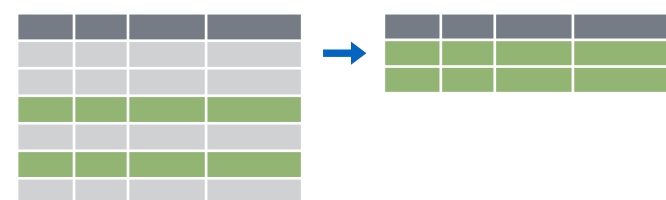
```
  arrange(desc(n_change))
```

	year	sex	name	n	prop	n_change
	<dbl>	<chr>	<chr>	<int>	<dbl>	<int>
1	1997	M	Fernando	2315	0.00116	439
2	2000	M	Fernando	2601	0.00125	426
3	2006	M	Fernando	2758	0.00126	318
4	1977	M	Fernando	1135	0.000664	249
5	1989	M	Fernando	1581	0.000754	221
6	1991	M	Fernando	1897	0.000895	167
7	1988	M	Fernando	1360	0.000680	157
8	1990	M	Fernando	1730	0.000804	149
9	1980	M	Fernando	1264	0.000681	134
10	2003	M	Fernando	2551	0.00121	128

RECAP: SINGLE TABLE VERBS



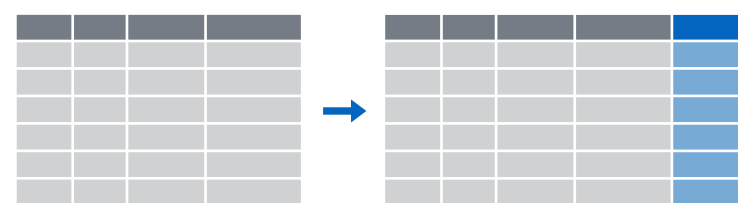
Extract variables with **select()**



Extract cases with **filter()**



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



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



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

ACKNOWLEDGEMENTS

- Some ideas, examples, and figures from RStudio webinars, which are licensed CC by SA.