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.qmd** 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()`

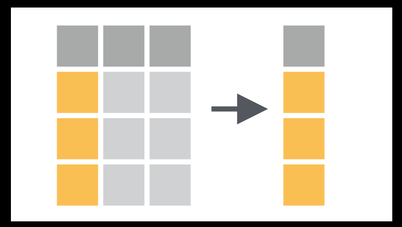
Make new variables with `mutate()`

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

select()

SELECT()

Extract columns by name.

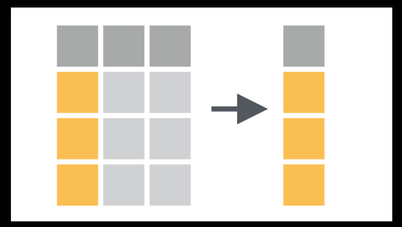


```
select(.data, …)
```

data frame to transform

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

SELECT()



Extract columns by name.

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

Select multiple variables by separating them with commas

Note how the order of columns is determined by the order of inputs

babynames

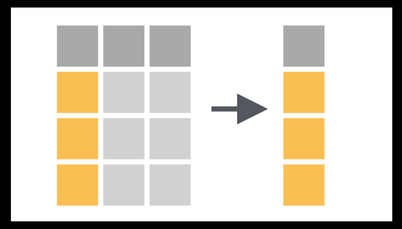
year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888
1880	F	Margaret	1578	0.016167

→

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

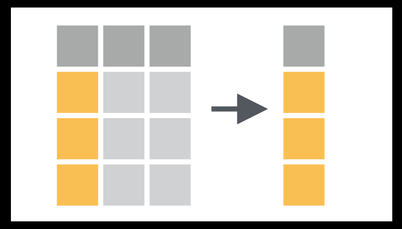
- Alter the code to select just the n column:

```
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	0.0723835
Anna	0.0266789
Emma	0.0205214
Elizabeth	0.0198657
Minnie	0.0178884
Margaret	0.0161672

`select()` helpers

`:` Select range of columns

```
select(penguins, species:sex)
```

`!` or `-` Negate selection (select every column but)

```
select(penguins, !c(island, sex))
```

`starts_with()` Select columns that start with…

```
select(penguins, starts_with("bill"))
```

`ends_with()` Select columns that end with…

```
select(penguins, ends_with("_mm"))
```

`select()` helpers

`contains()` Select columns whose names contain...

```
select(penguins, contains("length"))
```

`matches()` Select columns whose names match regular expression

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

`any_of()` Select columns whose names are in a set

```
select(penguins, any_of(c("year", "Year", "yEaR")))
```

`num_range()` Select columns named in prefix, number style

```
select(billboard, num_range("wk", 10:15))
```

`select()` helpers

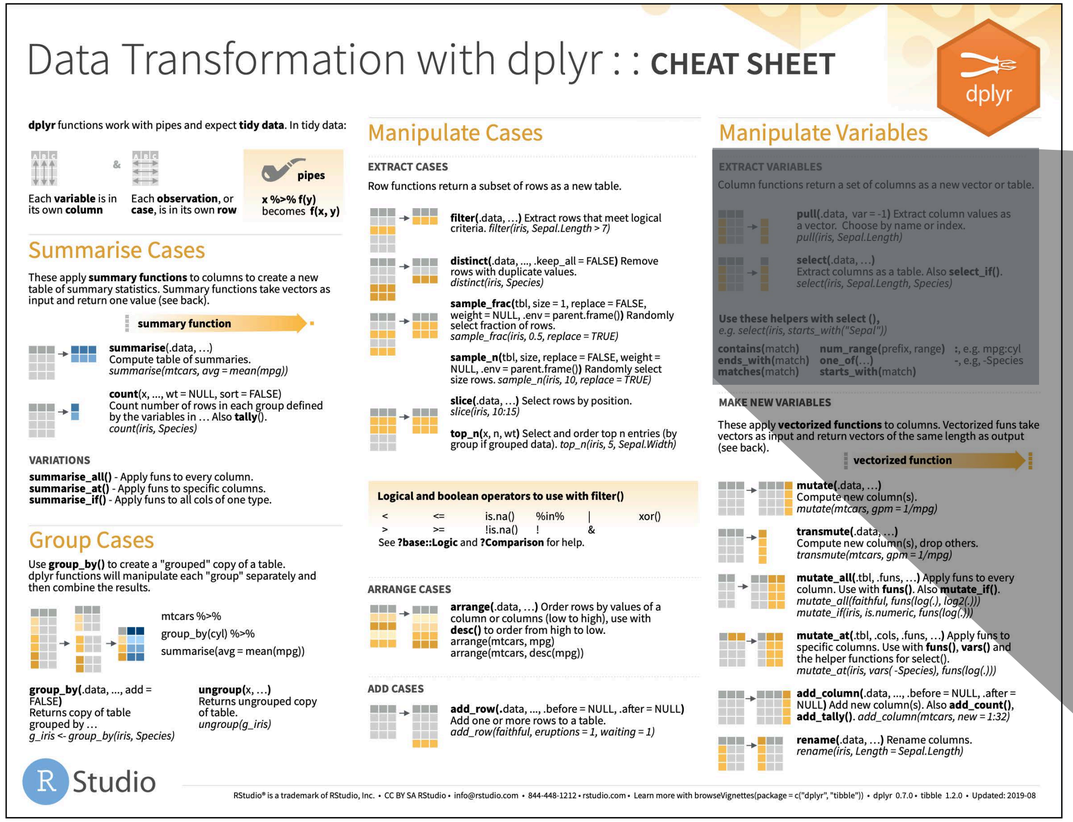
`everything()` Select all variables

```
select(penguins, everything())
```

`where()` Select columns for which function returns TRUE

```
select(penguins, where(is.numeric) & !year)
```

`select()` helpers



Data Transformation with dplyr : : **CHEAT SHEET**

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 of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarise(.data, …)
Compute table of summaries.
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 specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create 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.

filter(.data, …) Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
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.
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 or columns (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 vector or table.

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. 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).

vectorized function

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()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …) Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))

add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …) Rename columns.
rename(iris, Length = Sepal.Length)

RStudio

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table

pull(.data, var = -1) Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …)
Extract columns as a table. 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?

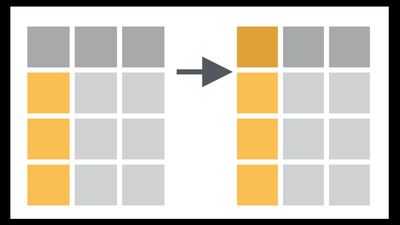
```
select(babynames, -c(year, sex, prop))

select(babynames, name:n)

select(babynames, starts_with("n"))

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

SELECT()'S COUSINS

For manipulating variables/columns

RENAME()

Rename columns without extracting.



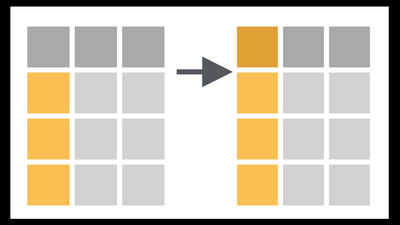
```
rename(babynames, popularity = prop)
```

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

→

year	sex	name	n	popularity
1880	F	Mary	7065	0.0723835
1880	F	Anna	2604	0.0266789
1880	F	Emma	2003	0.0205214
1880	F	Elizabeth	1939	0.0198657
1880	F	Minnie	1746	0.0178884

RELOCATE()



Move columns without ext

If no other options provided, column(s) moved to the left side of data frame

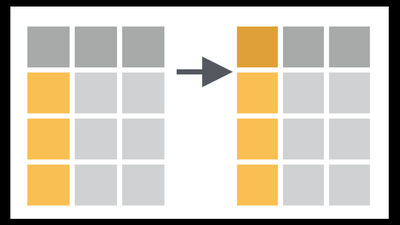
```
relocate(babynames, name)
```

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

→

name	year	sex	n	prop
Mary	1880	F	7065	0.0723835
Anna	1880	F	2604	0.0266789
Emma	1880	F	2003	0.0205214
Elizab	1880	F	1939	0.0198657
Minnie	1880	F	1746	0.0178884

RELOCATE()



Move columns without extracting

```
relocate(babynames, name, .before = sex)
```

Position can be specified with `.before` or `.after` options.

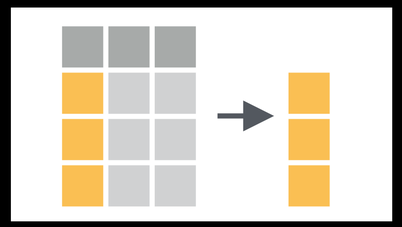
year	sex	name	n	prop
1880	F	Mary	7065	0.072383
1880	F	Anna	2604	0.026678
1880	F	Emma	2003	0.020521
1880	F	Elizabeth	1939	0.019865
1880	F	Minnie	1746	0.017888

→

year	name	sex	n	prop
1880	Mary	F	7065	0.0723835
1880	Anna	F	2604	0.0266789
1880	Emma	F	2003	0.0205214
1880	Elizab	F	1939	0.0198657
1880	Minnie	F	1746	0.0178884

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

→ `[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
`x < y`	LESS THAN
`x <= y`	LESS THAN OR EQUAL TO
`x > y`	GREATER THAN
`x >= y`	GREATER THAN OR EQUAL TO
`x == y`	EXACTLY EQUAL TO
`x != y`	NOT EQUAL TO
`x %in% y`	GROUP MEMBERSHIP
`is.na(x)`	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()

Multiple criteria to meet can be separated with commas...

Extract rows that meet *every* logical criterion.

```
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

FILTER()

… which is equivalent to using the & operator

Extract rows that meet *every* logical criterion.

```
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
`!x`	NOT
`x | y`	OR
`x & y`	AND
`xor(x, y)`	EXACTLY OR (FALSE IF BOTH ARE TRUE)

```
?base::Logic
```

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

SLICE() – USEFUL VARIANTS

Variable to order by

`slice_min()` Subset rows with lowest values of variable

```
slice_min(penguins, bill_length_mm, n = 5)
```

Number of rows to extract

`slice_max()` Subset rows with highest values of variable

```
slice_max(penguins, bill_length_mm, n = 5)
```

`slice_sample()` Subset rows randomly

```
slice_sample(penguins, n = 5)
```

SLICE() – USEFUL VARIANTS

`slice_min()` Subset rows with lowest values of variable

```
slice_min(penguins, bill_length_mm, prop = 0.1)
```

Watch for ties!

Or proportion of rows to extract

`slice_max()` Subset rows with highest values of variable

```
slice_max(penguins, bill_length_mm, prop = 0.1)
```

`slice_sample()` Subset rows randomly

```
slice_sample(penguins, prop = 0.1)
```

```
slice_min(penguins, body_mass_g, n = 5)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
	<fct>	<fct>	<dbl>	<dbl>	<int>	<int>	<fct>	<int>
1	Gentoo	Biscoe	49.2	15.2	221	6300	male	2007
2	Gentoo	Biscoe	59.6	17	230	6050	male	2007
3	Gentoo	Biscoe	51.1	16.3	220	6000	male	2008
4	Gentoo	Biscoe	48.8	16.2	222	6000	male	2009
5	Gentoo	Biscoe	45.2	16.4	223	5950	male	2008
6	Gentoo	Biscoe	49.8	15.9	229	5950	male	2009

```
slice_min(penguins, body_mass_g, n = 5, with_ties = FALSE)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
	<fct>	<fct>	<dbl>	<dbl>	<int>	<int>	<fct>	<int>
1	Gentoo	Biscoe	49.2	15.2	221	6300	male	2007
2	Gentoo	Biscoe	59.6	17	230	6050	male	2007
3	Gentoo	Biscoe	51.1	16.3	220	6000	male	2008
4	Gentoo	Biscoe	48.8	16.2	222	6000	male	2009
5	Gentoo	Biscoe	45.2	16.4	223	5950	male	2008

DISTINCT()

Select only *unique/distinct* rows

```
distinct(penguins, species, body_mass_g)
```

When using multiple variables to determine uniqueness, separate with commas

```
distinct(penguins, species, body_mass_g, .keep_all = TRUE)
```

Keep all variables in the data? By default, only the columns used to determine uniqueness are retained

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, name)
```

```
#    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	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

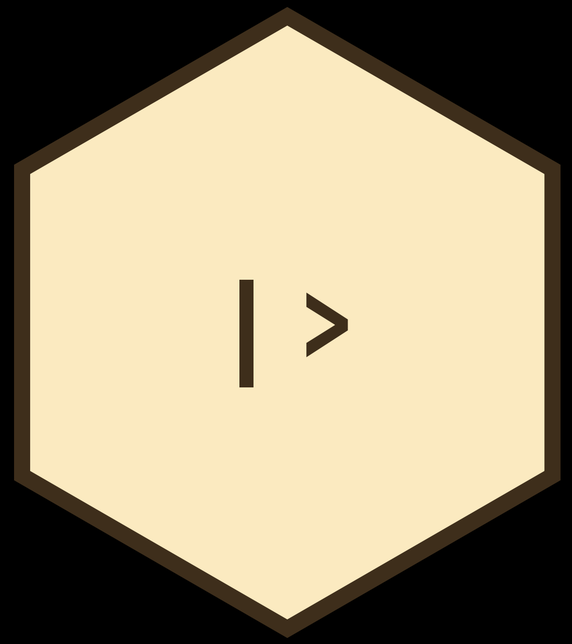
ACTIVITY 5

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

```
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
```

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.

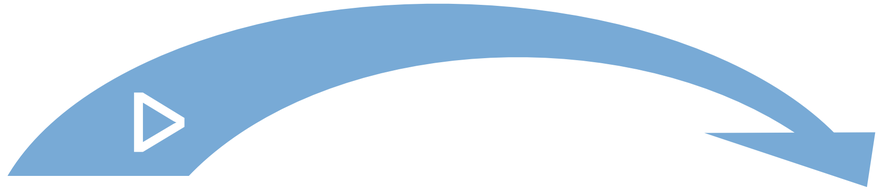
MULTI-STEP PROCEDURE


```
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 ▷



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

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

So, for example, these do the same thing.

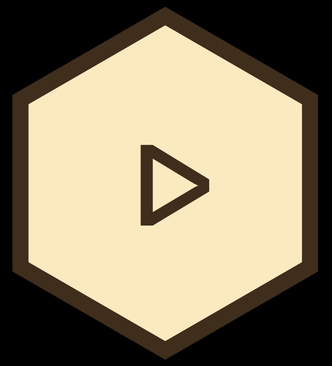
```
filter(babynames, n == 99680)
babynames ▷ filter(n == 99680)
```

MULTI-STEP PROCEDURE

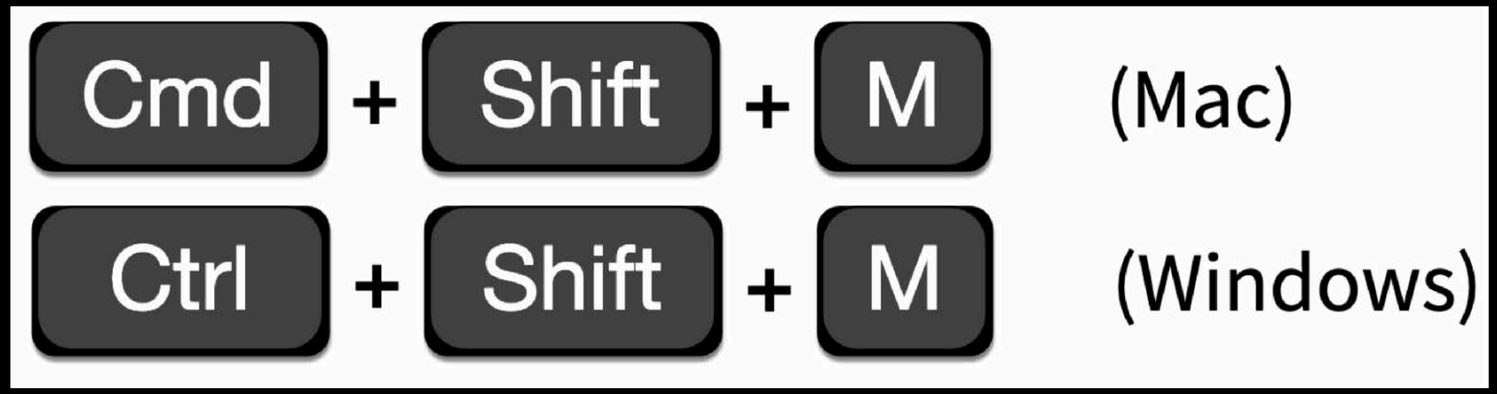

```
babynames
boys_2015 ← filter(babynames, year == 2015, sex == "M")
boys_2015 ← select(boys_2015, name, n)
boys_2015 ← arrange(boys_2015, desc(n))
boys_2015
```

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

PIPES



- Shortcut to type |>



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.

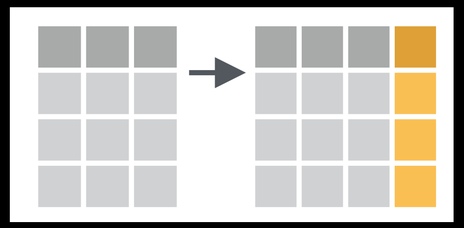
```
babynames ▷
  filter(year == 2015, sex == "F") ▷
  select(name, n) ▷
  arrange(desc(n))
```

```
#          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.



```
babynames ▷
  mutate(percent = prop * 100)
```

You can do basic arithmetic

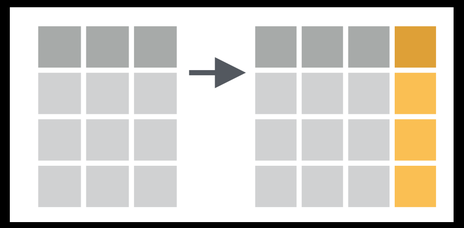
babynames

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.



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

Operations can be nested

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()

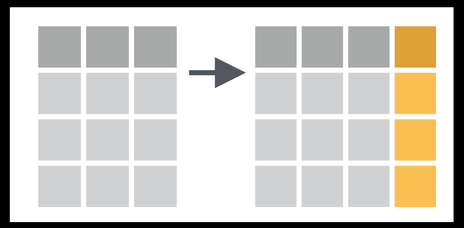
- Round a number to a specified number of decimal digits (0 by default)

```
x ← c(1.2, 1/3, 10.01)
round(x)
[1]  1  0 10
round(x, digits = 2)
```

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	pcnt_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.

Vector 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.

vectorized function

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 w 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

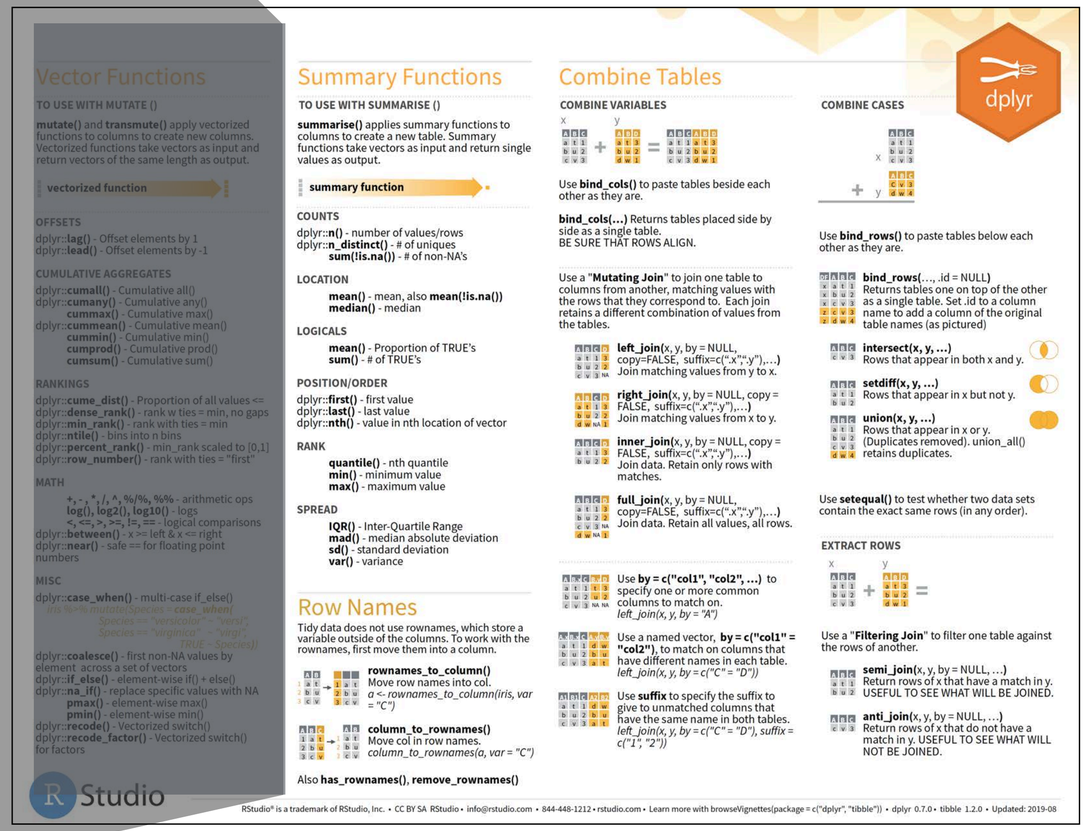
+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()

```
iris %>% mutate(Species = case_when(
                Species == "versicolor" ~ "versi",
                Species == "virginica"  ~ "virgi",
                TRUE ~ Species))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**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.

summary function

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() - median 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(iris, 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 one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
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 y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y. (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(c(50, 100, 100, 1000))
# [1] 1 2 2 4
```

```
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?

- How could you accomplish the same thing using some version of `slice()`?

```
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

```
babynames |>
  slice_max(prop, n = 10)
```

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

`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 calculate year-to-year change in popularity (using the `n` column) for your name/sex, then arrange the data set by this new variable with the year of greatest increase at the top.

```
babynames |>
    filter(name == "Fernando" & sex == "M") |>
    mutate(n_change = n - lag(n)) |>
    arrange(desc(n))
```

	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()**.

THE GLUE PACKAGE


```
library("glue")

my_name ← "Fernando"
glue("My name is {my_name}.")

my_nums ← 1:5
glue("This is row {my_nums}.")

glue("The price is now ${1:5}.00 higher.")

glue("The average of 10, 14, and 33 is: {mean(c(10, 14, 33))}")



name_sequence ← c("first name", "middle name", "last name")
my_names ← c("Fernando", "Alonso", "Campos")
glue("My {name_sequence} is {my_names}.")
```