

# Welcome to the tidyverse

**Week 2**

AEM 2850 / 5850 : R for Business Analytics

Cornell Dyson  
Spring 2023

Acknowledgements: **Grant McDermott, Allison Horst**

# Announcements

If you are new to the class, please review canvas and **course site**

**Prelim dates:** March 2 and May 9 in class

- SDS: we will use the Alternative Testing Program, submit form by Feb. 8

**TA office hours:** Mondays 12:30pm - 3:30pm in Warren 372

**My office hours:** Tuesdays 11am - 12pm in Warren 466; [aem2850.youcanbook.me](http://aem2850.youcanbook.me)

**Other reminders:**

- Bookmark **the schedule** and visit often
- Submit assignments via canvas by Monday at 11:59pm going forward

# Friday office hours poll

Hui Zhou will host additional TA office hours on Fridays

Informal poll: what times **do not** work for you on Fridays?

- 10:30 - 11:30
- 11:30 - 12:30
- 12:30 - 1:30
- 3:00 - 4:00

Questions before we get started?

# Plan for today

Prologue

Tidyverse basics

- tibbles and pipes

Data transformation with dplyr

- select
- filter
- arrange
- mutate
- summarize
- other goodies

# Prologue

# How we feel about the course, in a few words

- | excited
- | its lit
- | hope this class is not too hard haha
- | maybe the speaking speed is too fast

# Where we're from

How can we use your 39 survey responses to construct our origin story without having to read them one by one?

- **Caveat:** This is based on submissions as of 9:14 Tuesday

We can use R for this!

I won't go through details now

By the end of the course you should be able to do this kind of thing easily

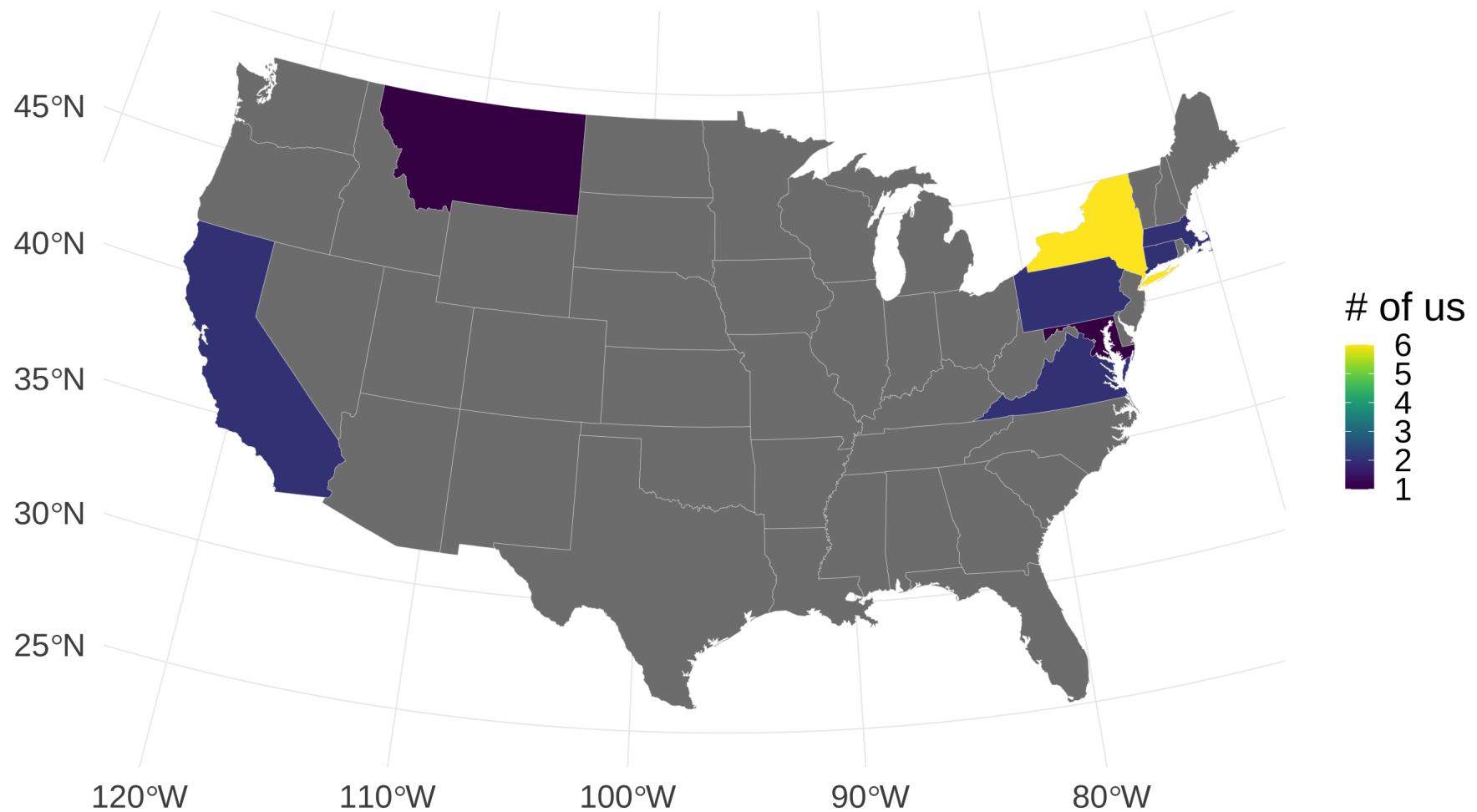
# Are we all from New York?

What share of us are from the state of New York?

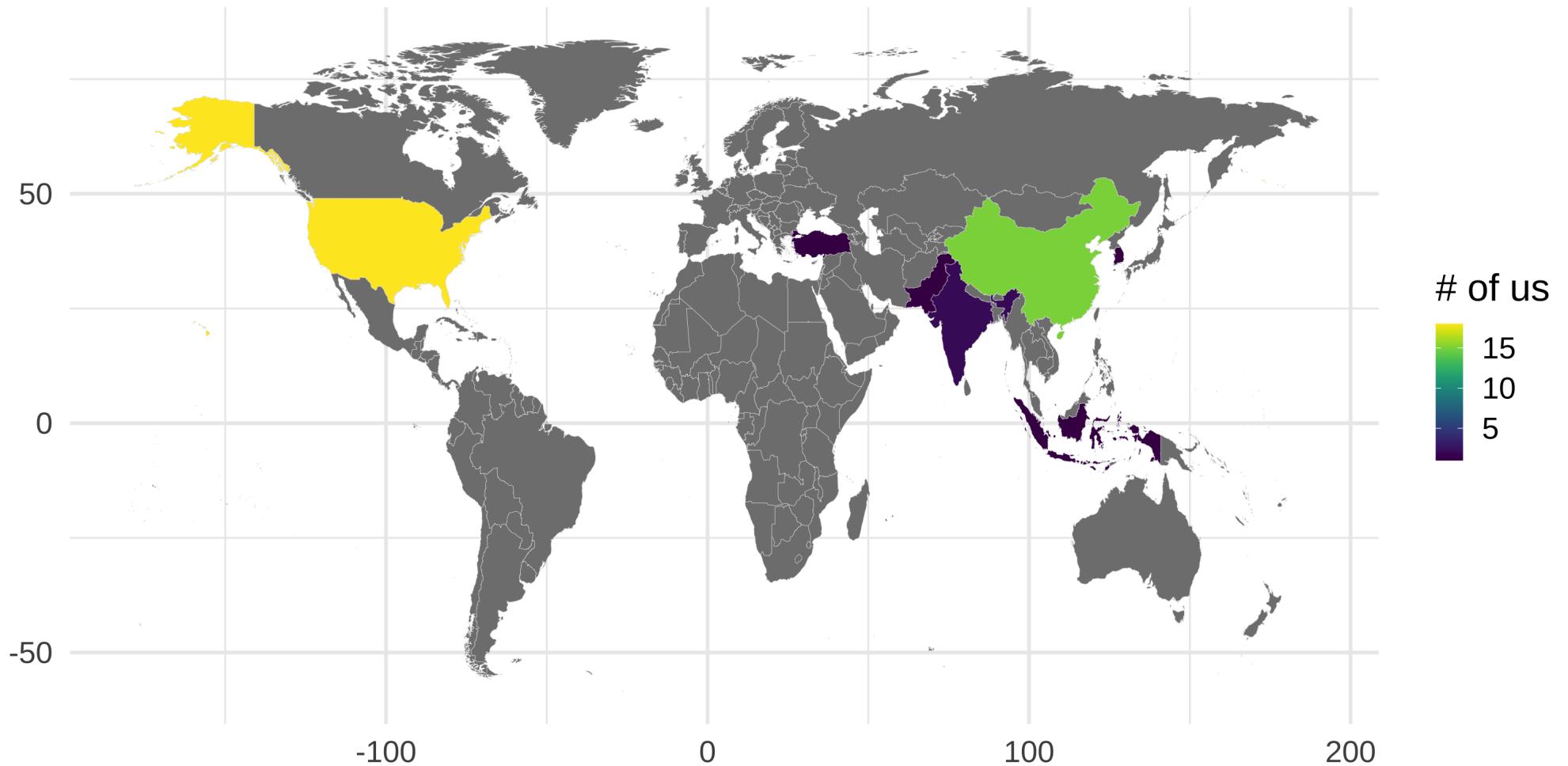
```
# step 1: do a bunch of stuff that's omitted for clarity  
# step 2: show us the answer!  
ny_share
```

```
## # A tibble: 1 × 2  
##   origin    percent  
##   <chr>      <dbl>  
## 1 new york     15
```

# Are we all from New York?



# Are we all from the U.S.?



# R "dialects"

Last week we *briefly* introduced several R programming concepts:

- logical expressions
- assignment
- object types
- [object names, namespace conflicts]
- [indexing]

We did this using base R

- **pro:** comes in the box, close parallels to other programming languages
- **con:** can be confusing for new programmers (esp. indexing operations)

# R "dialects"

This course will primarily use the tidyverse

You can think of it as a particular "dialect" of R

This dialect is similar to human language

Intuitive if you're new to programming

Base R and **data.table** are alternative "dialects"



# Tidyverse basics

# Tidyverse vs. base R

Lots of debate over tidyverse vs. base R

Why we're using the tidyverse:

- Consistent philosophy and syntax
- Great documentation and community support
- For data wrangling and plotting, tidyverse is 🔥

Base R is still great, can do some things tidyverse can't

**data.table** is another great alternative for data wrangling

# Tidyverse vs. base R

Often a correspondence between tidyverse and base R commands:

tidyverse	base
?readr::read_csv	?utils::read.csv
?dplyr::if_else	?base::ifelse
?tibble::tibble	?base::data.frame

Tidyverse alternatives typically offer extra features

**Remember:** There are always multiple ways to do something in R

# Let's load the tidyverse meta-package

```
library(tidyverse)
```

```
## — Attaching packages ————— tidyverse 1.3.2 —
## ✓ ggplot2 3.4.0      ✓ purrr   1.0.1
## ✓ tibble  3.1.8      ✓ dplyr   1.0.10
## ✓ tidyr   1.2.1      ✓ stringr 1.5.0
## ✓ readr   2.1.3      ✓ forcats 0.5.2
## — Conflicts ————— tidyverse_conflicts() —
## ✗ dplyr::filter() masks stats::filter()
## ✗ dplyr::lag()   masks stats::lag()
```

We just loaded a bunch of packages: **ggplot2**, **dplyr**, **tidyverse**, etc.

We also see some **namespace conflicts**

# Tidyverse packages

When you install the tidyverse, you actually get a lot more packages:

```
tidyverse_packages()

## [1] "broom"          "cli"            "crayon"         "dbplyr"
## [5] "dplyr"          "dtplyr"         "forcats"        "ggplot2"
## [9] "googledrive"   "googlesheets4" "haven"         "hms"
## [13] "httr"           "jsonlite"       "lubridate"      "magrittr"
## [17] "modelr"         "pillar"         "purrr"          "readr"
## [21] "readxl"         "reprex"         "rlang"          "rstudioapi"
## [25] "rvest"          "stringr"        "tibble"         "tidyverse"
## [29] "xml2"
```

These other packages have to be loaded separately

# Tidyverse packages

Today we'll focus on **dplyr**

But first let's talk about tibbles and pipes

# Review: What are objects?

There are many different *types* (or *classes*) of objects

Here are some objects that we'll be working with regularly:

- vectors
- matrices
- data frames
- lists
- functions

# Review: Data frames

The most important object we'll work with is the **data frame**

You can think of it as an Excel spreadsheet

```
# Create a small data frame called "d"  
d <- data.frame(x = 1:2, y = 3:4)  
d
```

```
##   x y  
## 1 1 3  
## 2 2 4
```

This is essentially just a table with columns named **x** and **y**

Each row is an observation telling us the values of both **x** and **y**

# Tibbles are data frames with opinions

For this class you can think of tibbles and data frames as synonymous

Tibbles have some advantages, such as pretty printing

We will primarily use tibbles in this course

Two functions may come in handy:

- `as_tibble()` converts data frames to tibbles
- `tibble()` creates new tibbles from scratch

See `vignette("tibble")` for more

# Printing a data.frame

```
as.data.frame(starwars)
```

		name	height	mass	hair_color	skin_color
## 1	Luke	Skywalker	172	77.0	blond	fair
## 2		C-3PO	167	75.0	<NA>	gold
## 3		R2-D2	96	32.0	<NA>	white, blue
## 4		Darth Vader	202	136.0	none	white
## 5		Leia Organa	150	49.0	brown	light
## 6		Owen Lars	178	120.0	brown, grey	light
## 7	Beru	Whitesun lars	165	75.0	brown	light
## 8		R5-D4	97	32.0	<NA>	white, red
## 9		Biggs Darklighter	183	84.0	black	light
## 10		Obi-Wan Kenobi	182	77.0	auburn, white	fair
## 11		Anakin Skywalker	188	84.0	blond	fair
## 12		Wilhuff Tarkin	180	NA	auburn, grey	fair
## 13		Chewbacca	228	112.0	brown	unknown
## 14		Han Solo	180	80.0	brown	fair
## 15		Greedo	173	74.0	<NA>	green
## 16	Jabba	Desilijic Tiure	175	1358.0	<NA>	green-tan, brown
## 17		Wedge Antilles	170	77.0	brown	fair
## 18		Jek Tono Porkins	180	110.0	brown	fair
## 19		Yoda	66	17.0	white	green

# How could we make this look nicer?

```
as.data.frame(starwars)
```

		name	height	mass	hair_color	skin_color
## 1	Luke	Skywalker	172	77.0	blond	fair
## 2		C-3PO	167	75.0	<NA>	gold
## 3		R2-D2	96	32.0	<NA>	white, blue
## 4		Darth Vader	202	136.0	none	white
## 5		Leia Organa	150	49.0	brown	light
## 6		Owen Lars	178	120.0	brown, grey	light
## 7	Beru	Whitesun lars	165	75.0	brown	light
## 8		R5-D4	97	32.0	<NA>	white, red
## 9		Biggs Darklighter	183	84.0	black	light
## 10		Obi-Wan Kenobi	182	77.0	auburn, white	fair
## 11		Anakin Skywalker	188	84.0	blond	fair
## 12		Wilhuff Tarkin	180	NA	auburn, grey	fair
## 13		Chewbacca	228	112.0	brown	unknown
## 14		Han Solo	180	80.0	brown	fair
## 15		Greedo	173	74.0	<NA>	green
## 16	Jabba	Desilijic Tiure	175	1358.0	<NA>	green-tan, brown
## 17		Wedge Antilles	170	77.0	brown	fair
## 18		Jek Tono Porkins	180	110.0	brown	fair
## 19		Yoda	66	17.0	white	green

# Printing a data.frame with head()

```
head(as.data.frame(starwars))
```

```
##           name height mass hair_color skin_color eye_color birth_year
## 1 Luke Skywalker    172   77     blond      fair     blue      19.0
## 2 C-3PO             167   75     <NA>      gold    yellow     112.0
## 3 R2-D2              96   32     <NA>  white, blue     red      33.0
## 4 Darth Vader       202  136     none      white    yellow     41.9
## 5 Leia Organa        150   49     brown     light    brown      19.0
## 6 Owen Lars          178  120 brown, grey     light     blue      52.0
##   sex   gender homeworld species
## 1 male masculine Tatooine Human
## 2 none masculine Tatooine Droid
## 3 none masculine Naboo Droid
## 4 male masculine Tatooine Human
## 5 female feminine Alderaan Human
## 6 male masculine Tatooine Human
##
## 1                               The Empire Strikes Back, Revenge of the Sith, Return of the
## 2                               The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of the
## 3 The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of the Sith, Return of the
## 4                               The Empire Strikes Back, Revenge of the Sith, Return of the
## 5                               The Empire Strikes Back, Revenge of the Sith, Return of the
```

# Printing a tibble

```
starwars
```

```
## # A tibble: 87 × 14
##   name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex   gender homew...⁵
##   <chr>     <int> <dbl> <chr>   <chr>   <chr>   <dbl> <chr>   <chr>   <chr>
## 1 Luke Skywalker 172    77 blond   fair    blue     19    male   masculin... Tatooi...
## 2 C-3PO          167    75 <NA>    gold    yellow   112   none   masculin... Tatooi...
## 3 R2-D2           96     32 <NA>    white,... red     33    none   masculin... Naboo
## 4 Darth Vader    202    136 none    white    yellow   41.9  male   masculin... Tatooi...
## 5 Leia Organa    150     49 brown   light    brown    19    femal... feminin... Aldera...
## 6 Owen Lars      178    120 brown,... light    blue     52    male   masculin... Tatooi...
## 7 Beru White...   165     75 brown   light    blue     47    femal... feminin... Tatooi...
## 8 R5-D4           97     32 <NA>    white,... red     NA    none   masculin... Tatooi...
## 9 Biggs Dark...   183     84 black   light    brown    24    male   masculin... Tatooi...
## 10 Obi-Wan Kenobi 182    77 auburn... fair    blue-g... 57    male   masculin... Stewjon
## # ... with 77 more rows, 4 more variables: species <chr>, films <list>,
## #   vehicles <list>, starships <list>, and abbreviated variable names
## #   ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year, ⁵homeworld
```

# Pipes

**Generic programming question:** Does anyone know what pipes do?

Pipes allow us to pass information through a sequence of operations

The tidyverse loads the `magrittr` pipe, denoted `%>%`

R now has a native pipe, denoted `|>` (since R version 4.1.0)

They are identical for simple cases, but have some differences

We will use `|>` in class

**Keyboard shortcut:** use `Ctrl/Cmd+Shift+m` to insert the pipe (with spaces)

- You can set the RStudio keyboard shortcut to use either `%>%` or `|>`

# Pipe usage

To see why using pipes is so powerful, consider this contrived example based on what we did this morning:

1. Wake up
2. Get out of bed
3. Get dressed
4. Drink coffee
5. Go to class

# A day without pipes

You could code this through a series of assignment operations:

```
me <- wake_up(me)
me <- get_out_of_bed(me)
me <- get_dressed(me)
me <- drink(me, "coffee")
me <- go(me, "class")
```

Or by putting all these operations in a single line of code:

```
me <- go(drink(get_dressed(get_out_of_bed(wake_up(me)))), "coffee"), "class")
```

Neither is ideal

# Pipes are the best of both worlds

We can do everything at once, in order:

```
me |>
  wake_up() |>
  get_out_of_bed() |>
  get_dressed() |>
  drink("coffee") |>
  go("class")
```

Emphasis is on verbs (e.g., `wake_up`, `get_out_of_bed`, etc.), not nouns (i.e., `me`)

# Pipes: Best practices

Write linear code

Spread code out for readability: use one line per verb

Don't use pipes...

- when you need more than 10 (use intermediate objects instead)
- for multiple inputs or outputs
- for non-linear relationships

**Reminder:** `| >` and `%>%` can both be used, though they have some important differences for more advanced work. We will us `| >` in this class.

# dplyr

# What is dplyr?

d p l y r : go wrangling



# What is dplyr?

The dplyr package provides a **grammar of data manipulation**

dplyr's functions are **verbs** that correspond to things we want to **do**

dplyr functions all have these things in common:

1. their first argument is a data frame
2. their other arguments describe what you want to do
3. their output is a new data frame

**1 + 3 + pipes** allow us to combine simple steps to achieve complex results

# dplyr has five key verbs we will use

1. `filter`: subset rows based on their values
2. `arrange`: reorder rows based on their values
3. `select`: select columns (i.e., variables)
4. `mutate`: create new columns
5. `summarize`: collapse multiple rows into a single summary value

# dplyr verbs operate on different things

- Rows:
  - `filter`: subset rows based on their values
  - `arrange`: reorder rows based on their values
- Columns:
  - `select`: select columns (i.e., variables)
  - `mutate`: create new columns
- Groups of rows:
  - `summarize`: collapse multiple rows into a single summary value

Let's study these commands together using the `starwars` data frame that comes pre-packaged with dplyr

# Starwars

This is what the `starwars` dataset looks like:

```
starwars
```

```
## # A tibble: 87 × 14
##   name      height  mass hair_color¹ skin_color² eye_color³ birth_year⁴ sex gender homeworld⁵
##   <chr>     <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr> <chr>
## 1 Luke Skywalker 172     77 blond    fair     blue       19 male   masculin Tatooine
## 2 C-3PO          167     75 <NA>     gold     yellow     112 none   masculin Tatooine
## 3 R2-D2          96      32 <NA>     white,... red      33 none   masculin Naboo
## 4 Darth Vader   202     136 none     white     yellow     41.9 male   masculin Tatooine
## 5 Leia Organa   150      49 brown    light     brown      19 female feminin Alderaan
## 6 Owen Lars     178     120 brown,... light     blue      52 male   masculin Tatooine
## 7 Beru White...  165      75 brown    light     blue      47 female feminin Tatooine
## 8 R5-D4          97      32 <NA>     white,... red      NA none   masculin Tatooine
## 9 Biggs Dark...  183     84 black    light     brown      24 male   masculin Tatooine
## 10 Obi-Wan Kenobi 182     77 auburn... fair     blue-g...  57 male   masculin Stewart
## # ... with 77 more rows, 4 more variables: species <chr>, films <list>,
## #   vehicles <list>, starships <list>, and abbreviated variable names
## #   `¹hair_color`, `²skin_color`, `³eye_color`, `⁴birth_year`, `⁵homeworld`
```

# 1) dplyr::select

**select** allows us to select columns / variables:

```
starwars |> select(name)
```

```
## # A tibble: 87 × 1
##   name
##   <chr>
## 1 Luke Skywalker
## 2 C-3PO
## 3 R2-D2
## 4 Darth Vader
## 5 Leia Organa
## 6 Owen Lars
## 7 Beru Whitesun lars
## 8 R5-D4
## 9 Biggs Darklighter
## 10 Obi-Wan Kenobi
## # ... with 77 more rows
```

**select** returns a tibble even though you only asked for one variable

To get a vector, you can use:

- **starwars |> pull(name)**

# 1) dplyr::select

You can get fancy: use commas to select multiple columns, deselect a column using "-", and select consecutive columns using "first:last"

```
starwars |>  
  select(name:skin_color, species, -height)  
  
## # A tibble: 87 × 5  
##   name           mass hair_color   skin_color species  
##   <chr>        <dbl> <chr>       <chr>      <chr>  
## 1 Luke Skywalker     77 blond       fair       Human  
## 2 C-3PO              75 <NA>       gold       Droid  
## 3 R2-D2              32 <NA>       white, blue Droid  
## 4 Darth Vader        136 none        white      Human  
## 5 Leia Organa         49 brown       light      Human  
## 6 Owen Lars           120 brown, grey light      Human  
## 7 Beru Whitesun lars  75 brown       light      Human  
## 8 R5-D4              32 <NA>       white, red Droid  
## 9 Biggs Darklighter    84 black       light      Human  
## 10 Obi-Wan Kenobi     77 auburn, white fair      Human  
## # ... with 77 more rows
```

# 1) dplyr::select

You can also rename your selected variables at the same time:

```
starwars |>  
  select(alias = name, crib = homeworld)
```

```
## # A tibble: 87 × 2  
##   alias            crib  
##   <chr>           <chr>  
## 1 Luke Skywalker Tatooine  
## 2 C-3PO            Tatooine  
## 3 R2-D2            Naboo  
## 4 Darth Vader     Tatooine  
## 5 Leia Organa     Alderaan  
## 6 Owen Lars       Tatooine  
## 7 Beru Whitesun lars Tatooine  
## 8 R5-D4            Tatooine  
## 9 Biggs Darklighter Tatooine  
## 10 Obi-Wan Kenobi Stewjon  
## # ... with 77 more rows
```

# 1) dplyr::select

If you just want to rename columns without subsetting them, use `rename`:

```
starwars |>  
  rename(alias = name, crib = homeworld)
```

```
## # A tibble: 87 × 14  
##   alias height mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex   gender crib  species  
##   <chr>    <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr> <chr> <chr>  
## 1 Luke...     172     77 blond    fair     blue      19 male   masculin Tato... Human  
## 2 C-3PO       167     75 <NA>     gold     yellow    112 none   masculin Tato... Droid  
## 3 R2-D2        96     32 <NA>     white,... red      33 none   masculin Naboo Droid  
## 4 Dart...      202    136 none     white    yellow     41.9 male   masculin Tato... Human  
## 5 Leia...      150     49 brown    light    brown      19 female feminin Alde... Human  
## 6 Owen...      178    120 brown,... light    blue      52 male   masculin Tato... Human  
## 7 Beru...      165     75 brown    light    blue      47 female feminin Tato... Human  
## 8 R5-D4        97     32 <NA>     white,... red      NA none   masculin Tato... Droid  
## 9 Bigg...      183     84 black    light    brown      24 male   masculin Tato... Human  
## 10 Obi-...      182     77 auburn... fair     blue-g...  57 male   masculin Stew... Human  
## # ... with 77 more rows, 3 more variables: films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names ¹hair_color, ²skin_color,  
## #   ³eye_color, ⁴birth_year
```

# 1) dplyr::select

`select(contains(PATTERN))` provides a handy shortcut:

```
starwars |>  
  select(name, contains("color"))
```

```
## # A tibble: 87 × 4  
##   name          hair_color    skin_color eye_color  
##   <chr>         <chr>        <chr>      <chr>  
## 1 Luke Skywalker  blond       fair        blue  
## 2 C-3PO           <NA>        gold        yellow  
## 3 R2-D2           <NA>        white, blue red  
## 4 Darth Vader    none        white        yellow  
## 5 Leia Organa    brown       light        brown  
## 6 Owen Lars      brown, grey light        blue  
## 7 Beru Whitesun lars brown       light        blue  
## 8 R5-D4           <NA>        white, red red  
## 9 Biggs Darklighter black       light        brown  
## 10 Obi-Wan Kenobi auburn, white fair        blue-gray  
## # ... with 77 more rows
```

# 1) dplyr::select

`select(..., everything())` is another useful shortcut to move important variables to the "front" of a data frame

```
starwars |>  
  select(species, homeworld, everything()) |>  
  head(5)
```

```
## # A tibble: 5 × 14  
##   species homeworld name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex  
##   <chr>     <chr>    <chr>     <int>   <dbl> <chr>    <chr>    <chr>    <dbl> <chr>  
## 1 Human     Tatooine  Luke Sky...     172     77 blond    fair     blue      19 male  
## 2 Droid      Tatooine  C-3P0       167     75 <NA>     gold     yellow    112 none  
## 3 Droid      Naboo     R2-D2        96      32 <NA>     white,... red      33 none  
## 4 Human     Tatooine  Darth Va...     202    136 none     white     yellow    41.9 male  
## 5 Human     Alderaan  Leia Org...     150     49 brown    light     brown     19 fema...  
## # ... with 4 more variables: gender <chr>, films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names `hair_color`, `skin_color`,  
## #   `eye_color`, `birth_year`
```

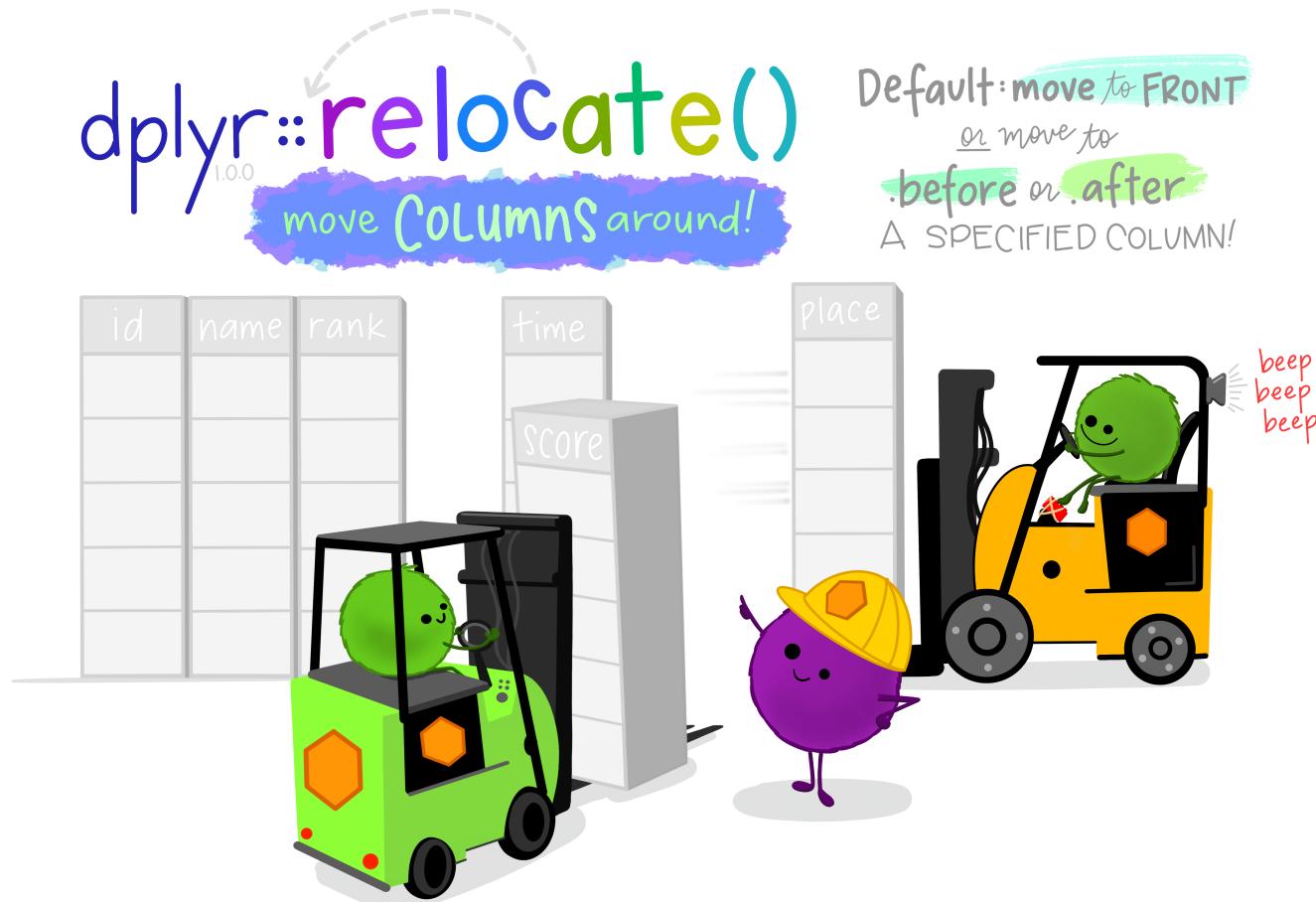
# 1) dplyr::~~select~~ relocate

You can also use `relocate` to reorder columns without subsetting:

```
starwars |>  
  relocate(species, homeworld) |>  
  head(5)
```

```
## # A tibble: 5 × 14  
##   species homeworld name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex  
##   <chr>     <chr>    <chr>     <int>   <dbl> <chr>    <chr>    <chr>    <dbl> <chr>  
## 1 Human     Tatooine  Luke Sky...     172     77 blond    fair     blue      19 male  
## 2 Droid      Tatooine  C-3P0       167     75 <NA>     gold     yellow    112 none  
## 3 Droid      Naboo     R2-D2        96      32 <NA>     white,... red      33 none  
## 4 Human     Tatooine  Darth Va...     202    136 none     white     yellow    41.9 male  
## 5 Human     Alderaan  Leia Org...     150     49 brown    light     brown     19 fema...  
## # ... with 4 more variables: gender <chr>, films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names `hair_color`, `skin_color`,  
## #   `eye_color`, `birth_year`
```

# 1) dplyr::~~select~~ relocate



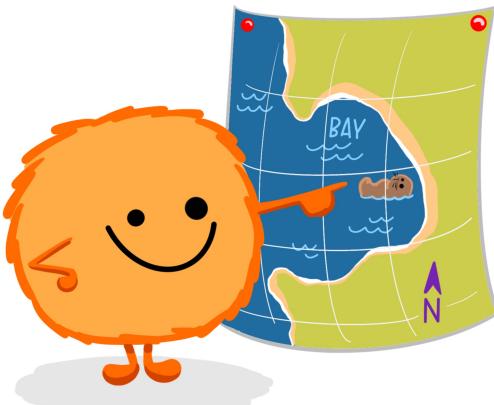
Let's all work through an example on Posit  
Cloud

## 2) dplyr::filter

# dplyr:: filter()

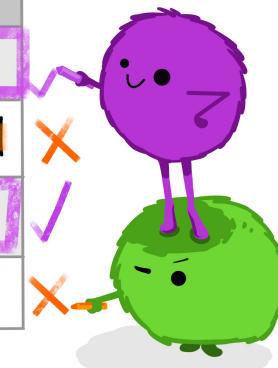
KEEP ROWS THAT  
s.a.t.i.s.f.y  
*your CONDITIONS*

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"  
filter(df, type == "otter" & site == "bay")



	type	food	site
otter	urchin	bay	
Shark	seal	channel	
otter	abalone	bay	
otter	crab	wharf	

@allison\_horst



## 2) dplyr::filter

`filter` allows us to focus on certain rows / observations

For example, you may only be interested in droids:

```
starwars |>  
  filter(species == "Droid")
```

```
## # A tibble: 6 × 14  
##   name    height  mass hair_color skin_color eye_c...¹ birth...² sex   gender homew...³  
##   <chr>     <int> <dbl> <chr>       <chr>      <chr>     <dbl> <chr> <chr> <chr>  
## 1 C-3PO      167     75 <NA>        gold       yellow     112 none  masculin... Tatooi...  
## 2 R2-D2       96      32 <NA>        white, bl... red        33 none  masculin... Naboo  
## 3 R5-D4       97      32 <NA>        white, red red        NA none  masculin... Tatooi...  
## 4 IG-88      200     140 none        metal      red        15 none  masculin... <NA>  
## 5 R4-P17      96      NA none        silver, r... red, b... NA none  feminin... <NA>  
## 6 BB8         NA      NA none        none       black      NA none  masculin... <NA>  
## # ... with 4 more variables: species <chr>, films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names ¹eye_color, ²birth_year,  
## #   ³homeworld
```

## 2) dplyr::filter

Or perhaps you want to subset the humans that are taller than I am:

```
starwars |>  
  filter(  
    species == "Human",  
    height >= 194  
  )
```

Can you name one? *Hint: there's only one*

```
## # A tibble: 1 × 14  
##   name      height  mass hair_c...¹ skin_c...² eye_c...³ birth...⁴ sex   gender homew...⁵  
##   <chr>     <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr> <chr>  <chr>  
## 1 Darth Vader     202    136 none     white    yellow     41.9 male   masculin... Tatooi...  
## # ... with 4 more variables: species <chr>, films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names `hair_color`, `skin_color`,  
## #   `eye_color`, `birth_year`, `homeworld`
```

## 2) dplyr::filter

Regular expressions work here too, with the help of the **stringr** package:

```
starwars |>
  filter(stringr::str_detect(name, "Skywalker"))

## # A tibble: 3 × 14
##   name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex   gender homew...⁵
##   <chr>     <int> <dbl> <chr>   <chr>   <chr>   <dbl> <chr> <chr>   <chr>
## 1 Luke Skywalker    172     77 blond   fair    blue     19 male   masculin... Tatooi...
## 2 Anakin Skywalker   188     84 blond   fair    blue    41.9 male   masculin... Tatooi...
## 3 Shmi Skywalker     163     NA black   fair    brown    72 female feminin... Tatooi...
## # ... with 4 more variables: species <chr>, films <list>, vehicles <list>,
## #   starships <list>, and abbreviated variable names `hair_color`, `skin_color`,
## #   `eye_color`, `birth_year`, `homeworld`
```

Or you can use **grepl** to achieve the same goal:

```
starwars |>
  filter(grepl("Skywalker", name))
```

## 2) dplyr::filter

A very common **filter** use case is identifying/removing missing data:

```
starwars |>  
  filter(is.na(height))
```

```
## # A tibble: 6 × 14  
##   name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex   gender homew...⁵  
##   <chr>     <int> <dbl> <chr>   <chr>   <chr>   <dbl> <chr> <chr> <chr> <chr>  
## 1 Arvel Crynyd     NA     NA brown   fair    brown      NA male  mascul... <NA>  
## 2 Finn            NA     NA black   dark    dark      NA male  mascul... <NA>  
## 3 Rey             NA     NA brown   light   hazel      NA fema... femin... <NA>  
## 4 Poe Dameron    NA     NA brown   light   brown      NA male  mascul... <NA>  
## 5 BB8              NA     NA none    none    black      NA none  mascul... <NA>  
## 6 Captain Pha...   NA     NA unknown unknown unknown      NA <NA>  <NA>  <NA>  
## # ... with 4 more variables: species <chr>, films <list>, vehicles <list>,  
## #   starships <list>, and abbreviated variable names `hair_color`, `skin_color`,  
## #   `eye_color`, `birth_year`, `homeworld`
```

## 2) dplyr::filter

How could you build on this to remove missing observations?

```
starwars |>  
  filter(!is.na(height)) # use ! for negation
```

```
## # A tibble: 81 × 14  
##   name      height  mass hair_color¹ skin_color² eye_color³ birth_year⁴ sex gender homeworld⁵  
##   <chr>     <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr> <chr>    <chr>  
## 1 Luke Skywalker 172     77  blond    fair     blue      19  male  masculin Tatooine  
## 2 C-3PO          167     75 <NA>     gold     yellow    112  none  masculin Tatooine  
## 3 R2-D2          96      32 <NA>     white,... red      33  none  masculin Naboo  
## 4 Darth Vader   202     136 none     white     yellow    41.9 male  masculin Tatooine  
## 5 Leia Organa   150      49 brown    light     brown     19  femal feminin Alderaan  
## 6 Owen Lars    178     120 brown,... light     blue      52  male  masculin Tatooine  
## 7 Beru White... 165      75 brown    light     blue      47  femal feminin Tatooine  
## 8 R5-D4          97      32 <NA>     white,... red      NA  none  masculin Tatooine  
## 9 Biggs Dark... 183      84 black    light     brown     24  male  masculin Tatooine  
## 10 Obi-Wan Kenobi 182     77 auburn... fair     blue-g...  57  male  masculin Stewart  
## # ... with 71 more rows, 4 more variables: species <chr>, films <list>,  
## #   vehicles <list>, starships <list>, and abbreviated variable names  
## #   `¹hair_color`, `²skin_color`, `³eye_color`, `⁴birth_year`, `⁵homeworld`
```

# 3) dplyr::arrange

`arrange` sorts the data frame based on the variable(s) you supply:

```
starwars |>  
  arrange(birth_year)
```

```
## # A tibble: 87 × 14  
##   name      height  mass hair_color¹ skin_color² eye_color³ birth_year⁴ sex gender homew…⁵  
##   <chr>     <int> <dbl> <chr>    <chr>    <chr>    <dbl> <chr> <chr> <chr> <chr>  
## 1 Wicket Sys…     88    20  brown    brown    brown       8 male  masculin… Endor  
## 2 IG-88          200   140 none     metal    red        15 none  masculin… <NA>  
## 3 Luke Skywalker 172    77  blond    fair     blue       19 male  masculin… Tatooi…  
## 4 Leia Organa    150    49  brown    light    brown       19 female feminin… Aldera…  
## 5 Wedge Antilles 170    77  brown    fair     hazel      21 male  masculin… Corell…  
## 6 Plo Koon        188    80  none     orange   black       22 male  masculin… Dorin  
## 7 Biggs Darkley  183    84  black    light    brown       24 male  masculin… Tatooi…  
## 8 Han Solo        180    80  brown    fair     brown       29 male  masculin… Corell…  
## 9 Lando Calrissian 177    79  black    dark     brown      31 male  masculin… Socorro  
## 10 Boba Fett      183   78.2 black   fair     brown      31.5 male  masculin… Kamino  
## # ... with 77 more rows, 4 more variables: species <chr>, films <list>,  
## #   vehicles <list>, starships <list>, and abbreviated variable names  
## #   `¹hair_color`, `²skin_color`, `³eye_color`, `⁴birth_year`, `⁵homeworld`
```

# 3) dplyr::arrange

We can also arrange items in descending order using `arrange(desc())`:

```
starwars |>  
  arrange(desc(birth_year))
```

```
## # A tibble: 87 × 14  
##   name      height  mass hair_...¹ skin_...² eye_c...³ birth...⁴ sex   gender homew...⁵  
##   <chr>     <int> <dbl> <chr>   <chr>   <chr>   <dbl> <chr> <chr>   <chr>   <chr>  
## 1 Yoda       66     17  white   green   brown    896 male   mascul... <NA>  
## 2 Jabba Desi... 175    1358 <NA>   green-... orange   600 herm... mascul... Nal Hu...  
## 3 Chewbacca   228    112  brown  unknown  blue     200 male   mascul... Kashyy...  
## 4 C-3PO       167     75  <NA>   gold    yellow   112 none   mascul... Tatooi...  
## 5 Dooku       193     80  white  fair    brown    102 male   mascul... Serenno  
## 6 Qui-Gon Ji... 193     89  brown  fair    blue     92 male   mascul... <NA>  
## 7 Ki-Adi-Mun... 198     82  white  pale    yellow   92 male   mascul... Cerea  
## 8 Finis Valo... 170     NA  blond  fair    blue     91 male   mascul... Corusc...  
## 9 Palpatine    170     75  grey   pale    yellow   82 male   mascul... Naboo  
## 10 Cliegg Lars 183    NA  brown  fair    blue     82 male   mascul... Tatooi...  
## # ... with 77 more rows, 4 more variables: species <chr>, films <list>,  
## #   vehicles <list>, starships <list>, and abbreviated variable names  
## #   ¹hair_color, ²skin_color, ³eye_color, ⁴birth_year, ⁵homeworld
```

## 4) dplyr::mutate



# 4) dplyr::mutate

You can create new columns from scratch or by transforming existing columns:

```
starwars |>
  select(name, birth_year) |>
  mutate(dog_years = birth_year * 7) |>
  mutate(comment = paste0(name, " is ", dog_years, " in dog years.))
```

```
## # A tibble: 87 × 4
##   name      birth_year  dog_years comment
##   <chr>        <dbl>     <dbl> <chr>
## 1 Luke Skywalker    19       133  Luke Skywalker is 133 in dog years.
## 2 C-3PO            112      784  C-3PO is 784 in dog years.
## 3 R2-D2             33       231  R2-D2 is 231 in dog years.
## 4 Darth Vader      41.9     293. Darth Vader is 293.3 in dog years.
## 5 Leia Organa       19       133  Leia Organa is 133 in dog years.
## 6 Owen Lars          52       364  Owen Lars is 364 in dog years.
## 7 Beru Whitesun lars 47       329  Beru Whitesun lars is 329 in dog yea...
## 8 R5-D4              NA       NA   R5-D4 is NA in dog years.
## 9 Biggs Darklighter  24       168  Biggs Darklighter is 168 in dog year...
## 10 Obi-Wan Kenobi     57       399  Obi-Wan Kenobi is 399 in dog years.
## # ... with 77 more rows
```

# 4) dplyr::mutate

**mutate** creates variables in order, so we can chain them together in a single call:

```
starwars |>
  select(name, birth_year) |>
  mutate(dog_years = birth_year * 7, # separate with a comma
         comment = paste0(name, " is ", dog_years, " in dog years."))
```

```
## # A tibble: 87 × 4
##   name      birth_year  dog_years comment
##   <chr>        <dbl>     <dbl> <chr>
## 1 Luke Skywalker    19       133  Luke Skywalker is 133 in dog years.
## 2 C-3PO            112      784  C-3PO is 784 in dog years.
## 3 R2-D2             33      231  R2-D2 is 231 in dog years.
## 4 Darth Vader      41.9    293. Darth Vader is 293.3 in dog years.
## 5 Leia Organa       19       133  Leia Organa is 133 in dog years.
## 6 Owen Lars          52      364  Owen Lars is 364 in dog years.
## 7 Beru Whitesun lars 47      329  Beru Whitesun lars is 329 in dog yea...
## 8 R5-D4              NA       NA   R5-D4 is NA in dog years.
## 9 Biggs Darklighter  24      168  Biggs Darklighter is 168 in dog year...
## 10 Obi-Wan Kenobi     57      399  Obi-Wan Kenobi is 399 in dog years.
## # ... with 77 more rows
```

# 4) dplyr::mutate

Boolean, logical, and conditional operators all work well with `mutate`:

```
starwars |>
  select(name, height) |>
  filter(name %in% c("Luke Skywalker", "Anakin Skywalker")) |>
  mutate(tall1 = height > 180) |>
  mutate(tall2 = ifelse(height > 180, "Tall", "Short")) # same effect, but can choose labels
```

How many rows do you think this will return? How many columns?

```
## # A tibble: 2 × 4
##   name           height tall1 tall2
##   <chr>        <int> <lgl> <chr>
## 1 Luke Skywalker     172 FALSE Short
## 2 Anakin Skywalker    188 TRUE  Tall
```

# 5) dplyr::summarize

Often we want to get summary statistics or *collapse* our data

`summarize` provides an easy way to do this

**Example:** "What are the minimum, maximum, and average prices of `diamonds`?"

```
diamonds |>
  summarize(min = min(price),           # calculate the min price
            max = max(price),           # calculate the max price
            average = mean(price)) # calculate the mean price
```

```
## # A tibble: 1 × 3
##       min     max   average
##   <int> <int>     <dbl>
## 1    326  18823     3933.
```

# 5) dplyr::summarize

Including `na.rm = TRUE` keeps NAs from propagating to the end result:

```
# Probably not what we want
starwars |>
  summarize(mh = mean(height))
```

```
## # A tibble: 1 × 1
##       mh
##   <dbl>
## 1    NA
```

```
# Much better
starwars |>
  summarize(mh = mean(height, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##       mh
##   <dbl>
## 1 174.
```

Is this a feature or a bug?

It depends on the context!

# Bonus: dplyr::group\_by

`summarize` is particularly useful in combination with `group_by`:

```
starwars |>
  group_by(species, gender) |> # for each species-gender combo
  summarize(mean_height = mean(height, na.rm = TRUE)) # calculate the mean height
```

```
## `summarise()` has grouped output by 'species'. You can override using the
## `.`groups` argument.
```

```
## # A tibble: 42 × 3
## # Groups:   species [38]
##   species   gender   mean_height
##   <chr>     <chr>       <dbl>
## 1 Aleena    masculine    79
## 2 Besalisk  masculine   198
## 3 Cerean    masculine   198
## 4 Chagrian  masculine   196
## 5 Clawdite  feminine    168
## 6 Droid     feminine     96
## 7 Droid     masculine   140
## 8 Dug       masculine   112
```

# Bonus: dplyr::ungroup



Groups are persistent (sort of), **ungroup** removes them

Other dplyr goodies for your reference

# Other dplyr goodies: across

## dplyr::across()

EXAMPLE:

```
df %>%
  group_by(species) %>%
  summarise(
    across(where(is.numeric), mean)
  )
```

use within `mutate()`  
or `summarize()` to  
apply function(s) to  
a selection of columns!



species	mass_g	age_yr	range_sqmi
pika	163	2.4	0.40
marmot	1509	3.0	0.87
marmot	2417	5.6	0.62

@allison\_horst

# Other dplyr goodies: across

Combining `mutate` with `across` allows you to work on a subset of variables:

```
starwars |>
  select(name:eye_color) |>
  mutate(across(where(is.character), toupper)) |> # capitalize all character variables
  head(5)
```

```
## # A tibble: 5 × 6
##   name           height   mass hair_color skin_color eye_color
##   <chr>        <int>   <dbl>   <chr>      <chr>      <chr>
## 1 LUKE SKYWALKER     172     77  BLOND       FAIR       BLUE
## 2 C-3PO              167     75 <NA>        GOLD       YELLOW
## 3 R2-D2                96     32 <NA>      WHITE,  BLUE  RED
## 4 DARTH VADER         202    136  NONE        WHITE      YELLOW
## 5 LEIA ORGANA          150     49  BROWN      LIGHT      BROWN
```

# Other dplyr goodies: across

We can also use `across` within `summarize`:

```
starwars |>
  group_by(species) |>
  summarize(across(where(is.numeric), mean, na.rm=T)) |> # take the mean of all numeric variables
  head(5)
```

```
## # A tibble: 5 × 4
##   species    height   mass birth_year
##   <chr>      <dbl>   <dbl>     <dbl>
## 1 Aleena       79     15       NaN
## 2 Besalisk     198    102       NaN
## 3 Cerean       198     82       92
## 4 Chagrian     196     NaN       NaN
## 5 Clawdite     168     55       NaN
```

# Other dplyr goodies: count

count to get the number of observations:

```
starwars |>  
  count(species)
```

```
## # A tibble: 38 × 2  
##   species     n  
##   <chr>     <int>  
## 1 Aleena      1  
## 2 Besalisk    1  
## 3 Cerean      1  
## 4 Chagrian    1  
## 5 Clawdite    1  
## 6 Droid       6  
## 7 Dug         1  
## 8 Ewok         1  
## 9 Geonosian   1  
## 10 Gungan     3  
## # ... with 28 more rows
```

```
starwars |>  
  group_by(species) |> summarize(num = n())
```

```
## # A tibble: 38 × 2  
##   species     num  
##   <chr>     <int>  
## 1 Aleena      1  
## 2 Besalisk    1  
## 3 Cerean      1  
## 4 Chagrian    1  
## 5 Clawdite    1  
## 6 Droid       6  
## 7 Dug         1  
## 8 Ewok         1  
## 9 Geonosian   1  
## 10 Gungan     3  
## # ... with 28 more rows
```

# Other dplyr goodies: distinct

`distinct` to isolate unique observations:

```
starwars |> distinct(species)
```

```
## # A tibble: 38 × 1
##   species
##   <chr>
## 1 Human
## 2 Droid
## 3 Wookiee
## 4 Rodian
## 5 Hutt
## 6 Yoda's species
## 7 Trandoshan
## 8 Mon Calamari
## 9 Ewok
## 10 Sullustan
## # ... with 28 more rows
```

# Other dplyr goodies: window functions

Window functions make it easy to get leads and lags, percentiles, cumulative sums, etc.

- These are often used in conjunction with grouped mutates
- See `vignette("window-functions")`

The final set of dplyr "goodies" worth mentioning are the family of join operations, but we'll come back to those later