

Welcome to the tidyverse

Week 3

AEM 2850: R for Business Analytics
Cornell Dyson
Spring 2022

Acknowledgements: Grant McDermott, Allison Horst

Announcements

If you are new to the class, please review canvas announcements

Reminders:

- Bookmark aem2850.toddgerarden.com/schedule and visit often
- Submit assignments via canvas
 - Lab 2 was due yesterday (Monday) at 11:59pm
 - Reflection - Week 3 is due Wednesday at 11:59pm

Bring your laptop to class on Thursday!

Questions before we get started?

Plan for today

Prologue

Tidyverse basics

Data transformation with dplyr

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

Prologue

What sports do we watch?

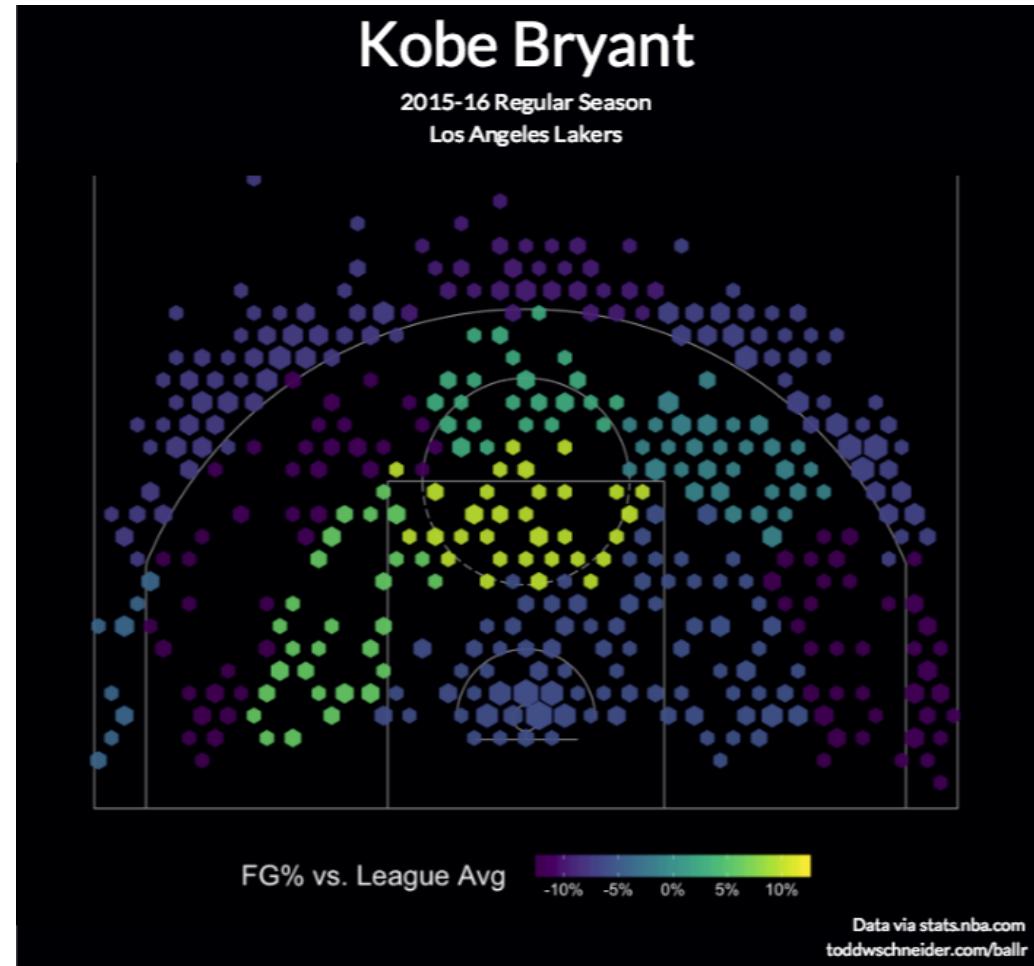
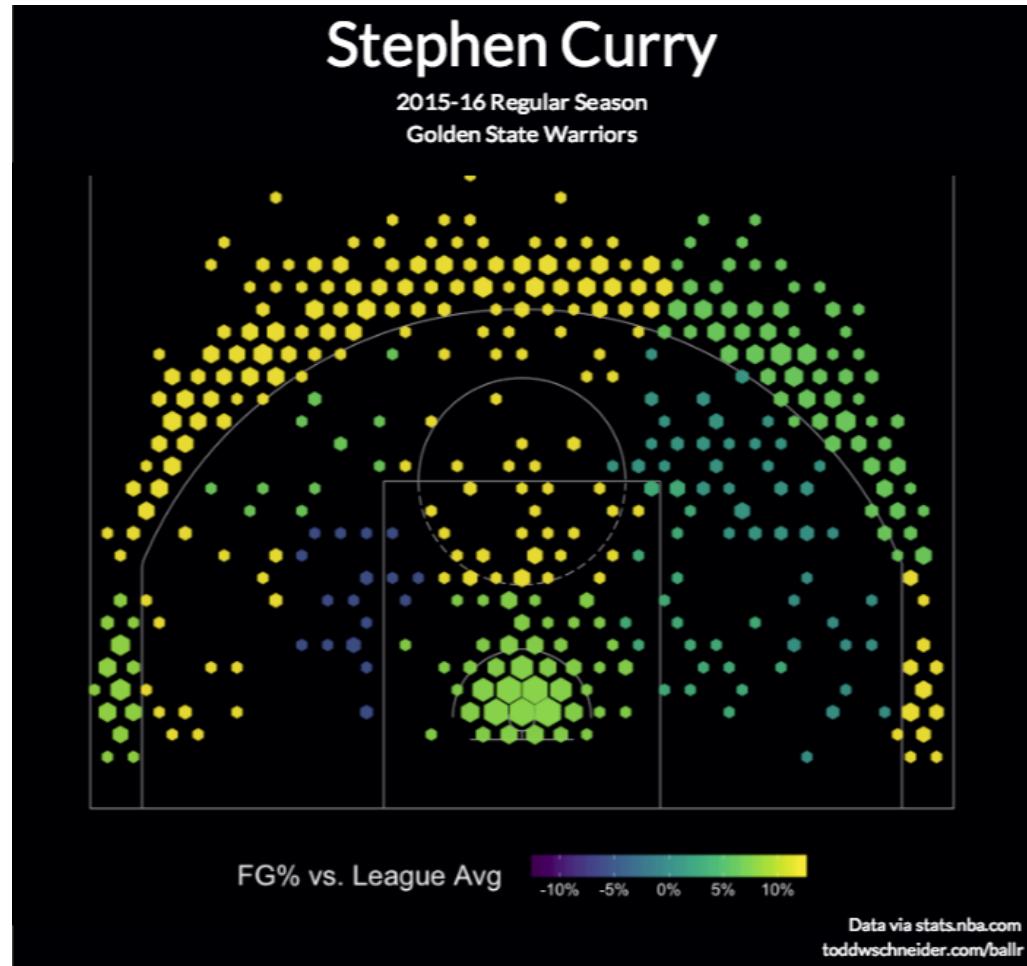
Take a guess: what's the most popular spectator sport among classmates?

```
## [1] "football"   "basketball"  "volleyball" "soccer"     NA
## [6] "baseball"    "football"    "basketball"  "basketball"  "football"
## [11] "gymnastics"  "gymnastics"  "squash"      NA          "basketball"
## [16] "gymnastics"  "tennis"      "baseball"    "basketball"  "basketball"
## [21] NA           "soccer"      "basketball"  "baseball"    "tennis"
## [26] "polo"        "hockey"      "basketball"  "gymnastics"  "soccer"
## [31] "tennis"      NA           "soccer"     "cricket"    "tennis"
## [36] "tennis"      "badminton"   "football"   "football"
```

Let's **count** and **arrange** to get the top 3:

```
## # A tibble: 3 × 2
##   sport      n
##   <chr>    <int>
## 1 basketball     8
## 2 football       5
## 3 tennis         5
```

R can be used for sports analytics, too!



R "dialects"

Last week we 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 week we will start using the tidyverse

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

This dialect is more like human language

More intuitive if you're new to programming

So if Lab 2 was confusing, don't get discouraged!



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 1.3.1 —

## ✓ ggplot2 3.3.5      ✓ purrr    0.3.4
## ✓ tibble   3.1.6      ✓ dplyr    1.0.7
## ✓ tidyr    1.1.4      ✓ stringr  1.4.0
## ✓ readr    2.1.1      ✓forcats  0.5.1

## — Conflicts ————— tidyverse_conflicts() —
## x tidyr::extract()  masks magrittr::extract()
## x dplyr::filter()   masks stats::filter()
## x dplyr::lag()      masks stats::lag()
## x purrr::set_names() masks magrittr::set_names()
```

We just loaded a bunch of packages: **ggplot2**, **tibble**, **dplyr**, 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"        "googledrive"
## [9] "googlesheets4" "ggplot2"        "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

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

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
##
##                                     The Empire Strikes Back, Revenge of the Sith, Return of the
## 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
```

Printing a tibble

```
starwars
```

```
## # A tibble: 87 × 14
##   name    height  mass hair_color skin_color eye_color birth_year sex gender
##   <chr>    <int> <dbl> <chr>     <chr>     <chr>        <dbl> <chr> <chr>
## 1 Luke S...     172     77 blond     fair      blue          19 male   mascul...
## 2 C-3PO          167     75 <NA>      gold      yellow       112 none   mascul...
## 3 R2-D2           96     32 <NA>      white, bl... red          33 none   mascul...
## 4 Darth ...       202    136 none      white      yellow       41.9 male   mascul...
## 5 Leia O...       150     49 brown     light      brown         19 fema... femin...
## 6 Owen L...       178    120 brown, grey light      blue          52 male   mascul...
## 7 Beru W...       165     75 brown     light      blue          47 fema... femin...
## 8 R5-D4           97     32 <NA>      white, red red          NA none   mascul...
## 9 Biggs ...       183     84 black     light      brown         24 male   mascul...
## 10 Obi-Wa...      182     77 auburn, wh... fair      blue-gray       57 male   mascul...
## # ... with 77 more rows, and 5 more variables: homeworld <chr>, species <chr>,
## #   films <list>, vehicles <list>, starships <list>
```

Pipes

Generic programming question: Does anyone know what pipes do?

Pipes allow us to pass information through a sequence of operations

The tidyverse loads its own pipe operator, denoted `%>%`

Keyboard shortcut: use `Ctrl/Cmd+Shift+m` to insert `%>%` (with spaces)

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

Aside: due to the popularity of `%>%`, base R now includes a native pipe `| >` with slightly different properties. We will use `%>%` 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 + %>%` 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

As a reminder, this is what the `starwars` dataset looks like:

```
starwars
```

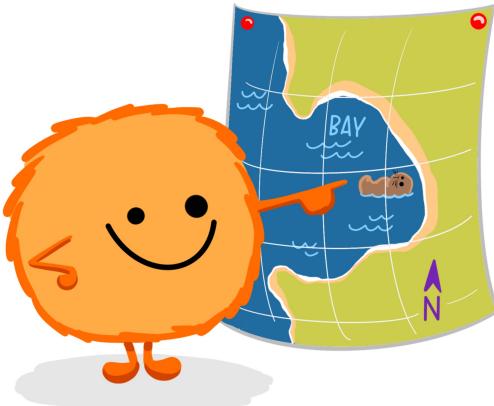
```
## # A tibble: 87 × 14
##   name    height  mass hair_color skin_color eye_color birth_year sex gender
##   <chr>    <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>
## 1 Luke Skywalker 172     77 blond      fair       blue            19   male  masculin...
## 2 C-3PO          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

1) dplyr::filter

dplyr:: filter()

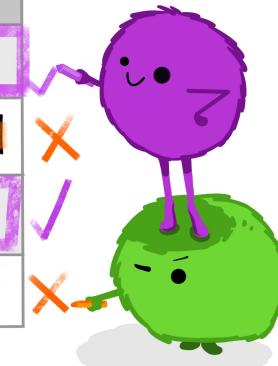
KEEP ROWS THAT
s.a.t.i.s.f.y
your CONDITIONS

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



1) 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_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>        <chr>       <chr>          <dbl> <chr> <chr>  
## 1 C-3PO      167     75 <NA>        gold       yellow           112 none  masculi...  
## 2 R2-D2       96     32 <NA>      white, blue red             33 none  masculi...  
## 3 R5-D4       97     32 <NA>      white, red red             NA none  masculi...  
## 4 IG-88      200    140 none      metal       red              15 none  masculi...  
## 5 R4-P17      96     NA none      silver, red red, blue       NA none  feminine  
## 6 BB8        NA     NA none       none       black            NA none  masculi...  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

1) 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_color skin_color eye_color birth_year sex   gender  
##   <chr>     <int>   <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>  
## 1 Darth V...     202     136 none        white       yellow        41.9 male  mascul...  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

1) 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_color skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>       <dbl> <chr> <chr>  
## 1 Luke Sk...     172     77 blond      fair        blue         19 male   masculin...  
## 2 Anakin ...    188     84 blond      fair        blue        41.9 male   masculin...  
## 3 Shmi Sk...    163     NA black      fair        brown        72 female feminin...  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

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

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

1) 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_color skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>       <dbl> <chr> <chr>  
## 1 Arvel C...     NA     NA brown      fair        brown          NA male  masculin...  
## 2 Finn          NA     NA black      dark        dark          NA male  masculin...  
## 3 Rey           NA     NA brown      light       hazel          NA female feminin...  
## 4 Poe Dam...    NA     NA brown      light       brown          NA male  masculin...  
## 5 BB8            NA     NA none       none       black          NA none  masculin...  
## 6 Captain...    NA     NA unknown    unknown    unknown          NA <NA> <NA>  
## # ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
## #   vehicles <list>, starships <list>
```

1) 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_color  skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>       <chr>      <chr>        <dbl> <chr> <chr>  
## 1 Luke S...     172     77 blond      fair       blue          19 male  masculin...  
## 2 C-3PO        167     75 <NA>       gold       yellow        112 none  masculin...  
## 3 R2-D2         96      32 <NA>      white, bl... red           33 none  masculin...  
## 4 Darth ...      202    136 none      white       yellow        41.9 male  masculin...  
## 5 Leia O...      150      49 brown      light      brown          19 female feminin...  
## 6 Owen L...      178    120 brown, grey light      blue          52 male  masculin...  
## 7 Beru W...      165      75 brown      light      blue          47 female feminin...  
## 8 R5-D4         97      32 <NA>      white, red red           NA none  masculin...  
## 9 Biggs ...      183      84 black      light      brown          24 male  masculin...  
## 10 Obi-Wa...     182      77 auburn, wh... fair      blue-gray       57 male  masculin...  
## # ... with 71 more rows, and 5 more variables: homeworld <chr>, species <chr>,  
## #   films <list>, vehicles <list>, starships <list>
```

2) 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_color skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>       <chr>       <chr>        <dbl> <chr> <chr>  
## 1 Wicket ...     88    20  brown      brown      brown          8 male  masculin...  
## 2 IG-88           200   140 none       metal      red            15 none  masculin...  
## 3 Luke Sk...      172    77 blond      fair       blue           19 male  masculin...  
## 4 Leia Or...      150    49 brown      light      brown          19 female feminin...  
## 5 Wedge A...      170    77 brown      fair       hazel          21 male  masculin...  
## 6 Plo Koon        188    80 none       orange     black           22 male  masculin...  
## 7 Biggs D...      183    84 black      light      brown          24 male  masculin...  
## 8 Han Solo        180    80 brown      fair       brown          29 male  masculin...  
## 9 Lando C...      177    79 black      dark       brown          31 male  masculin...  
## 10 Boba Fe...     183   78.2 black     fair       brown         31.5 male  masculin...  
## # ... with 77 more rows, and 5 more variables: homeworld <chr>, species <chr>,  
## #   films <list>, vehicles <list>, starships <list>
```

2) 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_color skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  
## 1 Yoda       66     17  white       green      brown           896 male  masculin...  
## 2 Jabba ...  175    1358 <NA>      green-tan,... orange           600 herm... masculin...  
## 3 Chewba...  228     112 brown      unknown     blue            200 male  masculin...  
## 4 C-3PO      167     75 <NA>       gold       yellow          112 none  masculin...  
## 5 Dooku      193     80 white      fair        brown           102 male  masculin...  
## 6 Qui-Go...  193     89 brown      fair        blue            92 male  masculin...  
## 7 Ki-Adi...  198     82 white      pale        yellow          92 male  masculin...  
## 8 Finis ...  170     NA blond     fair        blue            91 male  masculin...  
## 9 Palpat...  170     75 grey       pale        yellow          82 male  masculin...  
## 10 Cliegg... 183     NA brown     fair        blue           82 male  masculin...  
## # ... with 77 more rows, and 5 more variables: homeworld <chr>, species <chr>,  
## #   films <list>, vehicles <list>, starships <list>
```

3) 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)**

Or, from base R:

- **starwars %>% .\$name**
- **starwars %>% .[["name"]]**

3) 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
```

3) 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
```

3) 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_color skin_color eye_color birth_year sex gender  
##   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  
## 1 Luke S...     172     77 blond      fair       blue            19 male  masculin...  
## 2 C-3PO        167     75 <NA>       gold       yellow         112 none  masculin...  
## 3 R2-D2         96      32 <NA>      white, bl... red             33 none  masculin...  
## 4 Darth ...      202    136 none      white       yellow         41.9 male  masculin...  
## 5 Leia O...      150      49 brown      light      brown            19 female feminin...  
## 6 Owen L...      178    120 brown, grey light      blue            52 male  masculin...  
## 7 Beru W...      165      75 brown      light      blue            47 female feminin...  
## 8 R5-D4         97      32 <NA>      white, red red             NA none  masculin...  
## 9 Biggs ...      183      84 black      light      brown            24 male  masculin...  
## 10 Obi-Wa...     182      77 auburn, wh... fair      blue-gray         57 male  masculin...  
## # ... with 77 more rows, and 5 more variables: crib <chr>, species <chr>,  
## #   films <list>, vehicles <list>, starships <list>
```

3) 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
```

3) 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_color skin_color eye_color
##   <chr>     <chr>    <chr>        <int>  <dbl>   <chr>       <chr>      <chr>
## 1 Human     Tatooine  Luke Skywalker    172     77  blond       fair       blue
## 2 Droid      Tatooine  C-3PO          167     75 <NA>        gold       yellow
## 3 Droid      Naboo     R2-D2           96      32 <NA>      white, blue red
## 4 Human     Tatooine  Darth Vader     202    136 none        white      yellow
## 5 Human     Alderaan  Leia Organa     150     49 brown      light      brown
## # ... with 6 more variables: birth_year <dbl>, sex <chr>, gender <chr>,
## #   films <list>, vehicles <list>, starships <list>
```

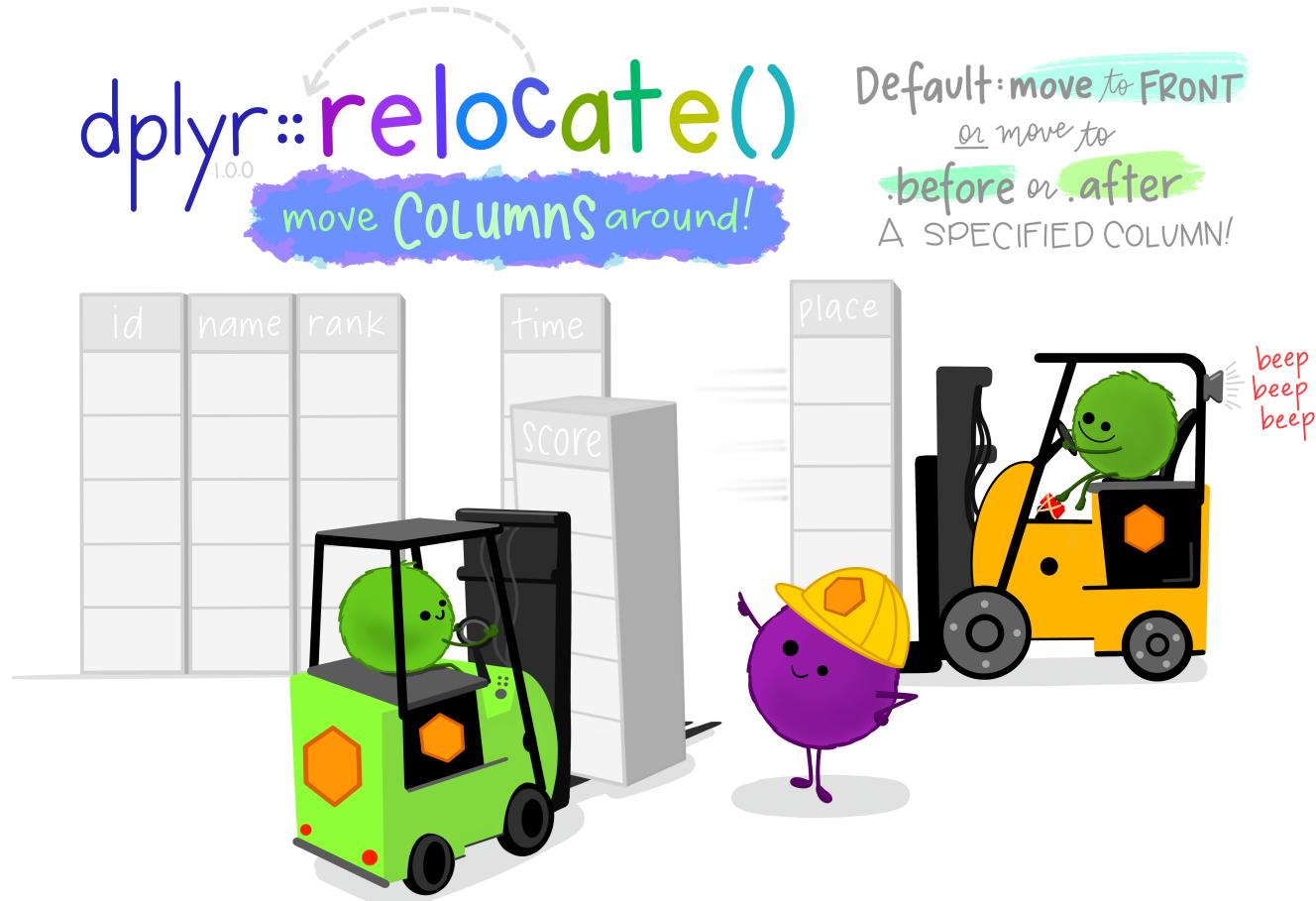
3) dplyr::~~select~~ relocate

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_color skin_color eye_color  
##   <chr>     <chr>    <chr>       <int>  <dbl>   <chr>      <chr>      <chr>  
## 1 Human     Tatooine  Luke Skywalker     172    77  blond      fair       blue  
## 2 Droid      Tatooine  C-3P0          167    75 <NA>       gold       yellow  
## 3 Droid      Naboo     R2-D2          96     32 <NA>      white, blue red  
## 4 Human     Tatooine  Darth Vader     202   136 none       white      yellow  
## 5 Human     Alderaan  Leia Organa     150    49 brown      light      brown  
## # ... with 6 more variables: birth_year <dbl>, sex <chr>, gender <chr>,  
## #   films <list>, vehicles <list>, starships <list>
```

3) dplyr::~~select~~ relocate



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.
```

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.
```

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 from Lab 2: "What are the minimum, maximum, and average prices of the 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.

- 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