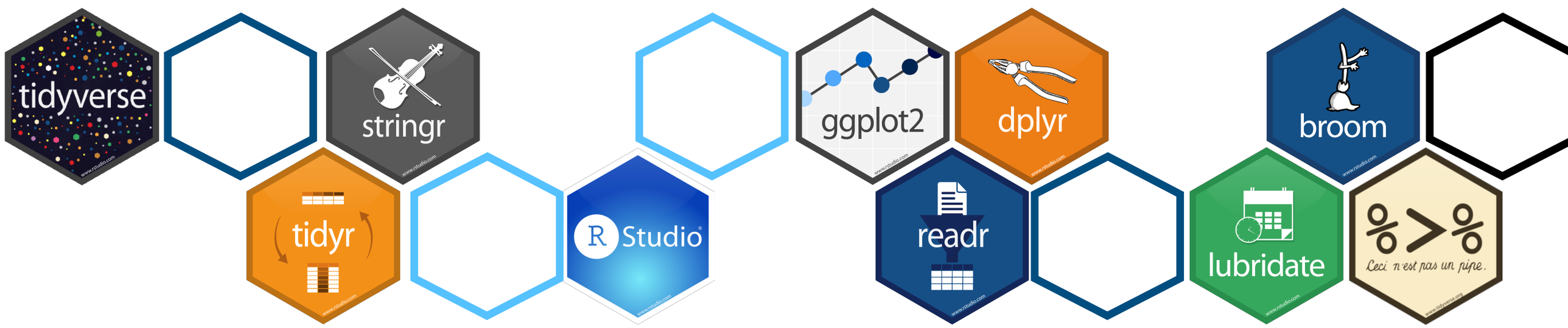


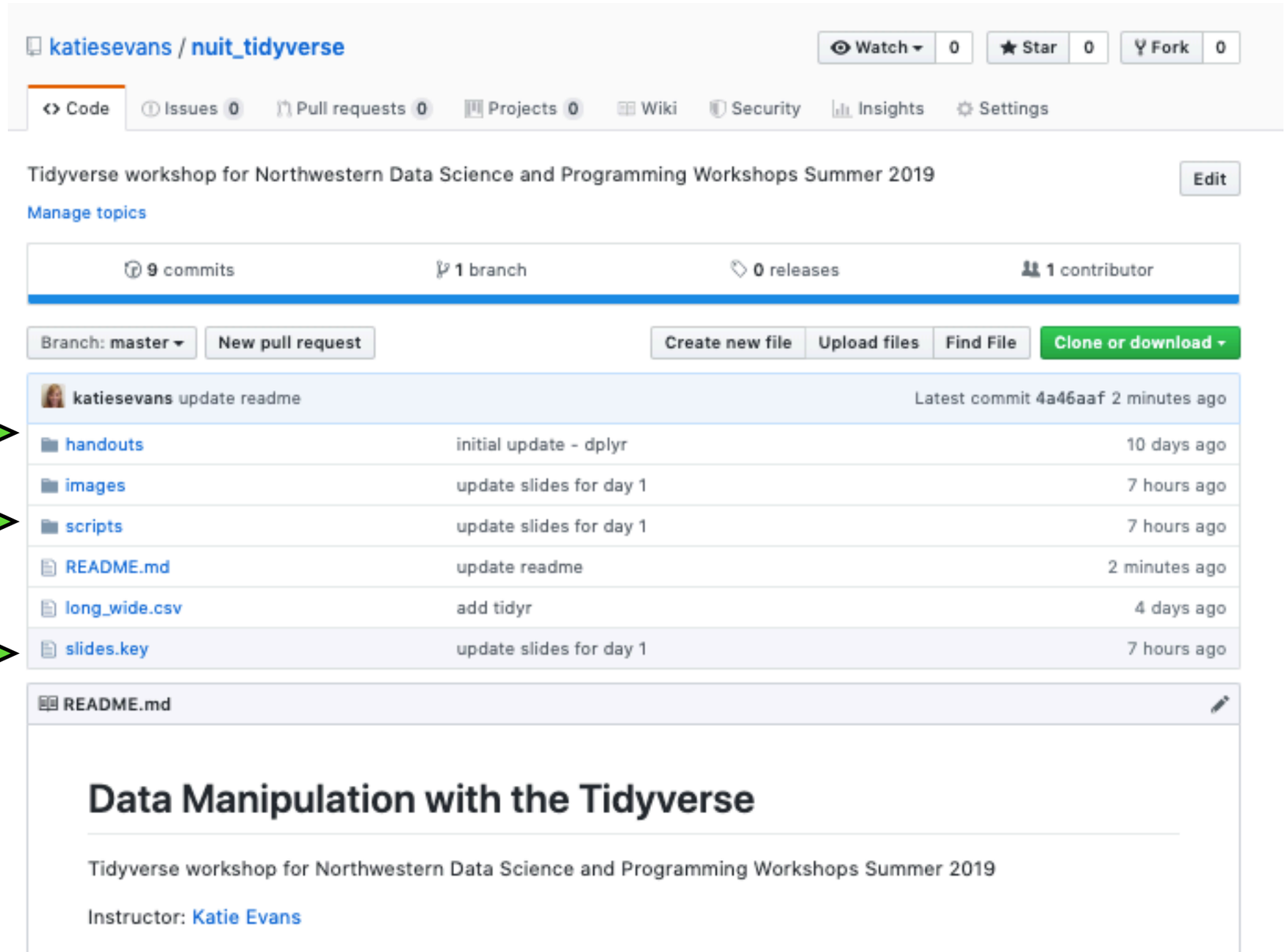
Data Manipulation with the Tidyverse

July 29-30, 2019 (9AM-12PM)

Chambers Hall, 600 Foster St., Lower Level

Katie Evans
NUIT Data Science Consultant
kathryn.evans@u.northwestern.edu
https://github.com/katiesevans/nuit_tidyverse

Workshop Resources



https://github.com/katiesevans/nuit_tidyverse

What is the Tidyverse?

- Collection of packages for data manipulation, exploration, and visualization that share a common syntax

- Intended to make data scientists more productive by guiding them through workflows

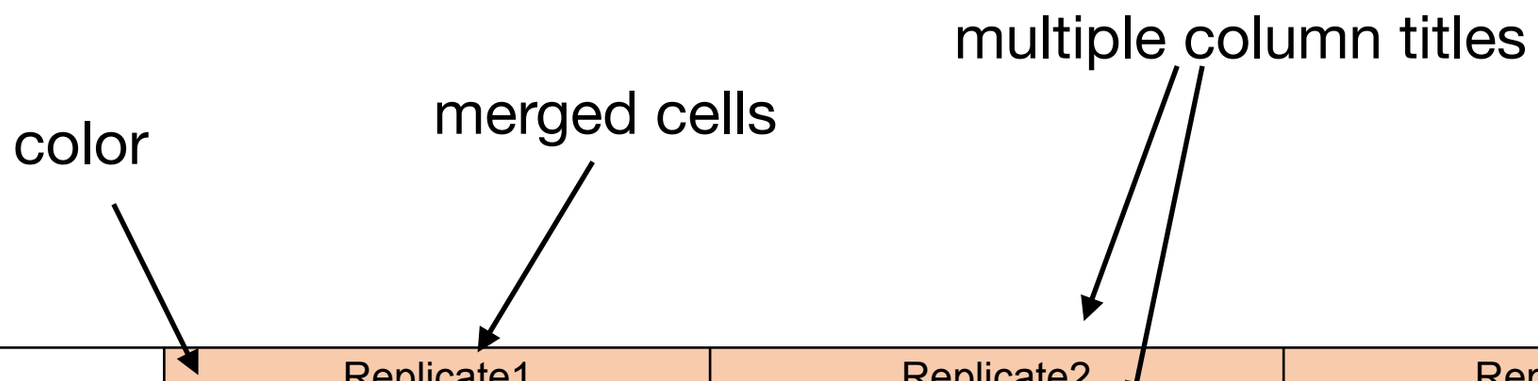
- Allows for connections between tools

Messy vs. clean data

“Happy families are all alike; every unhappy family is unhappy in its own way.” — Leo Tolstoy

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” — Hadley Wickham

Messy vs. clean data



	Replicate1			Replicate2			Replicate3			Replicate4		
PLATE1	day0	day2	day2/day0	day0	day2	day2/day0	day0	day2	day2/day0	day0	day2	day2/day0
BRC20067	229	2	0.87336244541	138	0	0	234	0	0	136	0	0
DL238	285	183	64.2105263157	334	161	48.203592814371	179	71	39.664804469273	224	110	49.1071428571429
A3	166	0	0	104	0	0	231	0	0	251	0	0
A6_R1	57	0	0	62	0	0	60	0	0	75	0	0
A6_R2	150	0	0	256	6	2.34375	265	4	1.5094339622641	236	0	0
B2	187	2	1.069518716577	165	1	0.60606060606060	183	0	0	171	4	2.33918128654971
B3	179	12	6.703910614525	202	26	12.871287128712	251	22	8.7649402390438	243	26	10.6995884773663
B6	128	15	11.71875	113	9	7.9646017699115	220	10	4.5454545454545	214	14	6.54205607476635
B8	268	0	0	155	0	0	164	1	0.6097560975609	175	0	0
B9	93	2	2.150537634408	118	2	1.6949152542372	132	0	0	111	2	1.8018018018018
B10	188	2	1.063829787234	379	2	0.5277044854881	225	0	0			
C6	169	0	0	129	1	0.7751937984496	130	0	0	115	1	0.869565217391304

Q: Clean or messy?

Messy vs. clean data

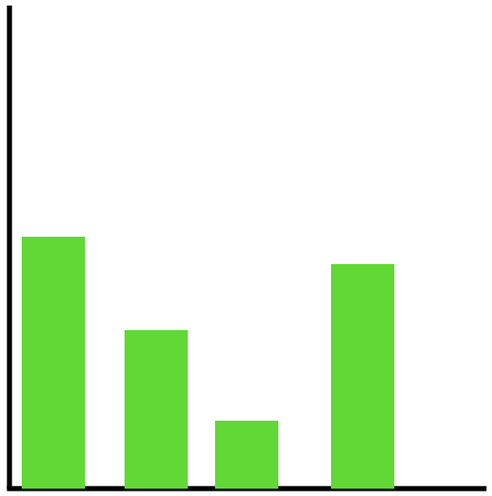
strain	survival	rep	day
BRC20067	0.873362445	A	1
DL238	64.21052631	A	1
A3	0	A	1
A6_R1	0	A	1
A6_R2	0	A	1
B2	1.069518716	A	1
B3	6.703910614	A	1
B6	11.71875	A	1
B8	0	A	1
B9	2.150537634	A	1
B10	1.063829787	A	1
C6	0	A	1
BRC20067	0	B	1
DL238	48.20359281	B	1
A3	0	B	1
A6_R1	0	B	1
A6_R2	2.34375	B	1
B2	0.606060606	B	1
B3	12.87128712	B	1
B6	7.964601769	B	1
B8	0	B	1

Rules for tidy data

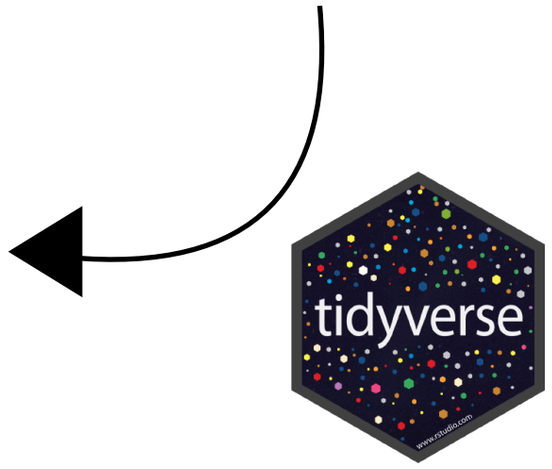
- Each **variable** must have its own **column**
- Each **observation** must have its own **row**
- Each **value** must have its own **cell**

Messy vs. clean data

	Replicate1			Replicate2			Replicate3			Replicate4		
PLATE1	day0	day2	day2/day0	day0	day2	day2/day0	day0	day2	day2/day0	day0	day2	day2/day0
BRC20067	229	2	0.873362445414	138	0	0	234	0	0	136	0	0
DL238	285	183	64.2105263157	334	161	48.203592814371	179	71	39.664804469273	224	110	49.1071428571429
A3	166	0	0	104	0	0	231	0	0	251	0	0
A6_R1	57	0	0	62	0	0	60	0	0	75	0	0
A6_R2	150	0	0	256	6	2.34375	265	4	1.5094339622641	236	0	0
B2	187	2	1.069518716577	165	1	0.60606060606060	183	0	0	171	4	2.33918128654971
B3	179	12	6.703910614525	202	26	12.871287128712	251	22	8.7649402390438	243	26	10.6995884773663
B6	128	15	11.71875	113	9	7.9646017699115	220	10	4.5454545454545	214	14	6.54205607476635
B8	268	0	0	155	0	0	164	1	0.6097560975609	175	0	0
B9	93	2	2.150537634408	118	2	1.69491525423729	132	0	0	111	2	1.8018018018018
B10	188	2	1.063829787234	379	2	0.52770448548811	225	0	0			
C6	169	0	0	129	1	0.7751937984496	130	0	0	115	1	0.869565217391304



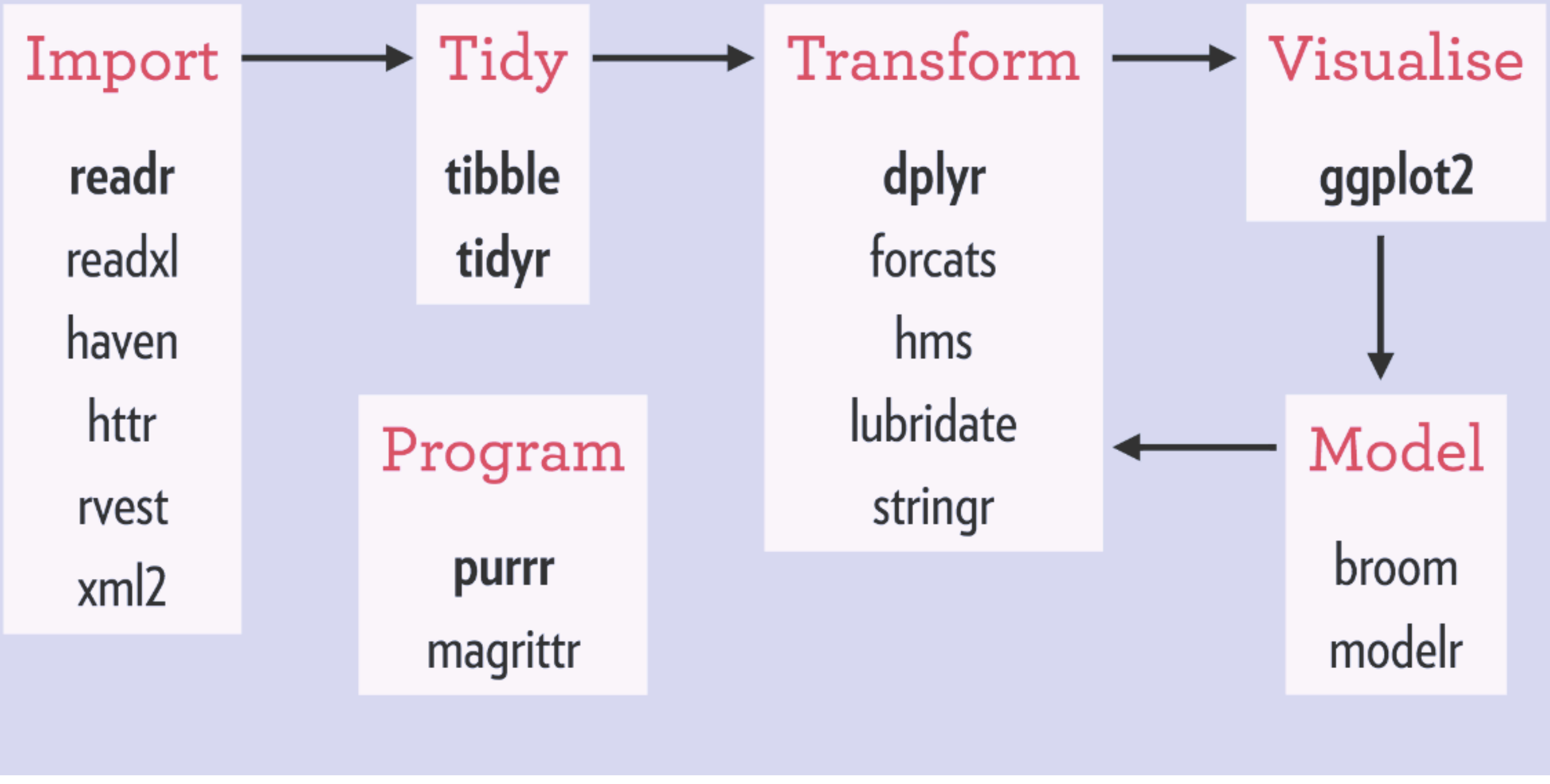
strain	survival	rep	day
BRC20067	0.873362445	A	1
DL238	64.210526315	A	1
A3	0	A	1
A6_R1	0	A	1
A6_R2	0	A	1
B2	1.069518716	A	1
B3	6.703910614	A	1
B6	11.71875	A	1
B8	0	A	1
B9	2.150537634	A	1



Tidyverse resources

- https://www.tidyverse.org
- https://www.rstudio.com/resources/cheatsheets
- https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html
- https://cran.r-project.org/web/packages/readr/vignettes/readr.html

Packages in Tidyverse



Thinking like a computer

1. Figure out what you want to do

2. Describe those tasks words

3. Describe those tasks in code

Starwars dataset


```
library(tidyverse)
data(starwars)
View(starwars)
```

variables

	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*	Naboo	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
4	Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	character(0)	TIE Advanced x1
5	Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	Imperial Speeder Bike	character(0)
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
7	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	Stewjon	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	Tatooine	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male	Eriadu	Human	c("Revenge of the Sith", "A New Hope")	character(0)	character(0)
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male	Kashyyyk	Wookiee	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	AT-ST	c("Millennium Falcon", "Imperial shuttle")
14	Han Solo	180	80.0	brown	fair	brown	29.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	character(0)	c("Millennium Falcon", "Imperial shuttle")
15	Greedo	173	74.0	*NA*	green	black	44.0	male	Rodia	Rodian	A New Hope	character(0)	character(0)
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphrodite	Nal Hutta	Hutt	c("The Phantom Menace", "Return of the Jedi", "A New ...	character(0)	character(0)
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	Snowspeeder	X-wing
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male	Bestine IV	Human	A New Hope	character(0)	X-wing
19	Yoda	66	17.0	white	green	brown	896.0	male	*NA*	Yoda's species	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male	Naboo	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
21	Boba Fett	183	78.2	black	fair	brown	31.5	male	Kamino	Human	c("Attack of the Clones", "Return of the Jedi", "The Em...	character(0)	Slave 1
22	IG-88	200	140.0	none	metal	red	15.0	none	*NA*	Droid	The Empire Strikes Back	character(0)	character(0)
23	Bossk	190	113.0	none	green	red	53.0	male	Trandosha	Trandoshan	The Empire Strikes Back	character(0)	character(0)
24	Lando Calrissian	177	79.0	black	dark	brown	31.0	male	Socorro	Human	c("Return of the Jedi", "The Empire Strikes Back")	character(0)	Millennium Falcon

observations

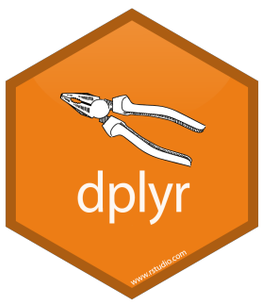
values

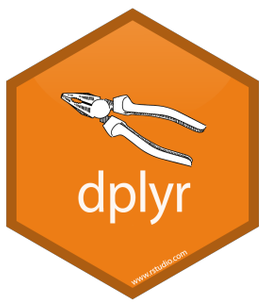
Introduction: **dplyr**

- Collection of functions as **verbs** to easily describe what you want to do with your data

Functions:

- `filter()` to keep rows based on values
- `select()` to keep columns based on names
- `mutate()` to add new (or change existing) columns
- `group_by()` to group rows by columns
- `summarize()` to condense multiple columns
- `arrange()` to reorder the rows
- `rename()` to give columns new names
- `xxx_join()` to combine data frames
- `bind_rows()` or `bind_cols()` to combine dataframes
- `pull()` to return a column as a vector





`dplyr::filter()`

`dplyr::filter()` to keep rows based on values

dplyr::filter()

`dplyr::filter(dataframe, condition(s))`

starwars

only humans

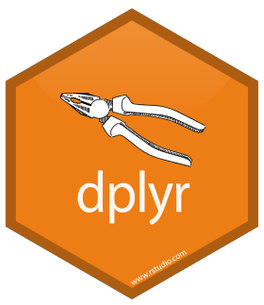
1. Figure out what you want to do

2. Describe your goal in words

Filter the starwars dataframe to only include humans

3. Describe your goal in code

```
dplyr::filter(starwars, ? )
```

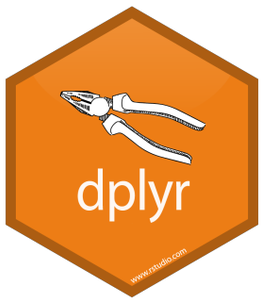


`dplyr::filter()` to keep rows based on values

dplyr::filter()

	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*	Naboo	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
4	Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	character(0)	TIE Advanced x1
5	Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	Imperial Speeder Bike	character(0)
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
7	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	Stewjon	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	Tatooine	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male	Eriadu	Human	c("Revenge of the Sith", "A New Hope")	character(0)	character(0)
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male	Kashyyyk	Wookiee	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	AT-ST	c("Millennium Falcon", "Imperial shuttle")
14	Han Solo	180	80.0	brown	fair	brown	29.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	character(0)	c("Millennium Falcon", "Imperial shuttle")
15	Greedo	173	74.0	*NA*	green	black	44.0	male	Rodia	Rodian	A New Hope	character(0)	character(0)
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphrodite	Nal Hutta	Hutt	c("The Phantom Menace", "Return of the Jedi", "A New ...	character(0)	character(0)
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	Snowspeeder	X-wing
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male	Bestine IV	Human	A New Hope	character(0)	X-wing
19	Yoda	66	17.0	white	green	brown	896.0	male	*NA*	Yoda's species	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male	Naboo	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
21	Boba Fett	183	78.2	black	fair	brown	31.5	male	Kamino	Human	c("Attack of the Clones", "Return of the Jedi", "The Em...	character(0)	Slave 1
22	IG-88	200	140.0	none	metal	red	15.0	none	*NA*	Droid	The Empire Strikes Back	character(0)	character(0)
23	Bossk	190	113.0	none	green	red	53.0	male	Trandosha	Trandoshan	The Empire Strikes Back	character(0)	character(0)
24	Lando Calrissian	177	79.0	black	dark	brown	31.0	male	Socorro	Human	c("Return of the Jedi", "The Empire Strikes Back")	character(0)	Millennium Falcon

`species == "Human"`



`dplyr::filter()` to keep rows based on values

dplyr::filter()

```
dplyr::filter(dataframe, condition(s))
```

starwars

only humans

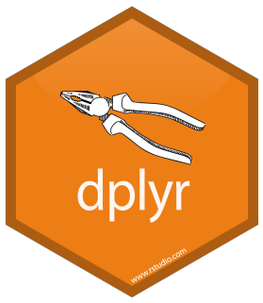
1. Figure out what you want to do

2. Describe your goal in words

Filter the starwars dataframe to only include humans

3. Describe your goal in code

```
dplyr::filter(starwars, species == "Human")
```

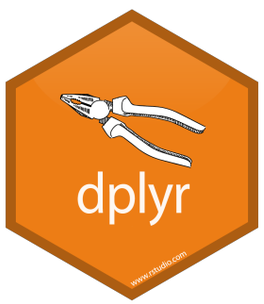


`dplyr::filter()` to keep rows based on values

dplyr::filter()

	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")
2	Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	character(0)	TIE Advanced x1
3	Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	Imperial Speeder Bike	character(0)
4	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
5	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)
6	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
7	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male	Stewjon	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...
8	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male	Tatooine	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...
9	Wilhuff Tarkin	180	NA	auburn, grey	fair	blue	64.0	male	Eriadu	Human	c("Revenge of the Sith", "A New Hope")	character(0)	character(0)
10	Han Solo	180	80.0	brown	fair	brown	29.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	character(0)	c("Millennium Falcon", "Imperial shuttle")
11	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	Snowspeeder	X-wing
12	Jek Tono Porkins	180	110.0	brown	fair	blue	NA	male	Bestine IV	Human	A New Hope	character(0)	X-wing
13	Palpatine	170	75.0	grey	pale	yellow	82.0	male	Naboo	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)
14	Boba Fett	183	78.2	black	fair	brown	31.5	male	Kamino	Human	c("Attack of the Clones", "Return of the Jedi", "The Em...	character(0)	Slave 1
15	Lando Calrissian	177	79.0	black	dark	brown	31.0	male	Socorro	Human	c("Return of the Jedi", "The Empire Strikes Back")	character(0)	Millennium Falcon
16	Lobot	175	79.0	none	light	blue	37.0	male	Bespin	Human	The Empire Strikes Back	character(0)	character(0)
17	Mon Mothma	150	NA	auburn	fair	blue	48.0	female	Chandrila	Human	Return of the Jedi	character(0)	character(0)
18	Arvel Crynyd	NA	NA	brown	fair	brown	NA	male	NA	Human	Return of the Jedi	character(0)	A-wing
19	Qui-Gon Jinn	193	89.0	brown	fair	blue	92.0	male	NA	Human	The Phantom Menace	Tribubble bongo	character(0)
20	Finis Valorum	170	NA	blond	fair	blue	91.0	male	Coruscant	Human	The Phantom Menace	character(0)	character(0)
21	Shmi Skywalker	163	NA	black	fair	brown	72.0	female	Tatooine	Human	c("Attack of the Clones", "The Phantom Menace")	character(0)	character(0)

```
dplyr::filter(starwars, species == "Human")
```



`dplyr::filter()` to keep rows based on values

dplyr::filter()

`dplyr::filter(dataframe, condition(s))`

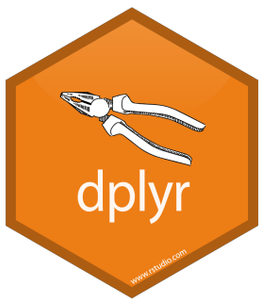
★ You can also filter based on multiple conditions

Filter the starwars dataframe to include only male humans.

```
dplyr::filter(starwars, species == "Human", gender == "male")
```

Filter the starwars dataframe to include humans and droids.

```
dplyr::filter(starwars, species %in% c("Human", "Droid"))
```



`dplyr::filter()` to keep rows based on values

dplyr::**filter**()

`dplyr::filter(`dataframe, condition(s)`)`

Filter starwars to include only people with height less than 100.

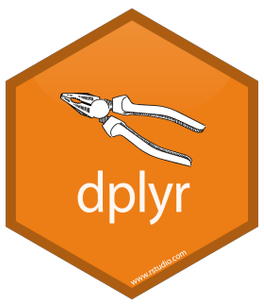
```
dplyr::filter(starwars, height < 100)
```

Keep all non-humans

```
dplyr::filter(starwars, species != “Human”)
```

Remove characters with no known species

```
dplyr::filter(starwars, !is.na(species))
```



`dplyr::filter()` to keep rows based on values

pipe (%>%)

takes output of left side and makes it input of right side

Piping in Tidyverse

Filter the starwars dataframe to include only male humans.

Without pipes

```
dplyr::filter(starwars, species == "Human", gender == "male")
```

1. start with the starwars dataframe

2. filter to keep only humans

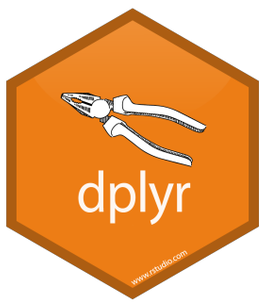
3. filter to keep only males

With pipes

```
starwars %>%
   dplyr::filter(species == "Human") %>%
   dplyr::filter(gender == "male")
```

Can be useful to combine several tidyverse "verbs" to one block of code

takes output of left side and makes it input of right side



`dplyr::select()`

`dplyr::select()` to keep columns based on names

dplyr::select()

```
dplyr::select(dataframe, columns_to_keep)
```

starwars

name, species, films

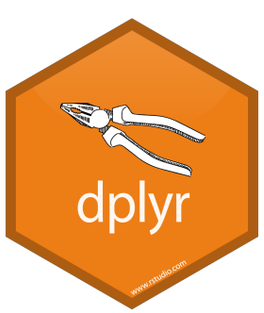
Select only name, species, and films variables from the starwars dataframe

```
dplyr::select(starwars, name, species, films)
```

★ Deselect a column with `-column_name`

Select all columns except the gender

```
dplyr::select(starwars, -gender)
```



`dplyr::select()` to keep columns based on names

dplyr::**select**()

```
dplyr::select(dataframe, columns_to_keep)
```

★ Select a range of columns with `column1:column2`

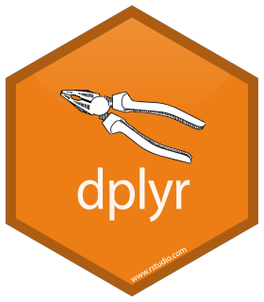
Select columns name, height, mass, hair color, and skin color

```
dplyr::select(starwars, name:skin_color)
```

Select all columns except hair color, skin color, and eye color

```
dplyr::select(starwars, -hair_color, -skin_color, -eye_color)
```

```
dplyr::select(starwars, -(hair_color:eye_color))
```



`dplyr::select()` to keep columns based on names

`dplyr::select()`

`dplyr::select(dataframe, columns_to_keep)`

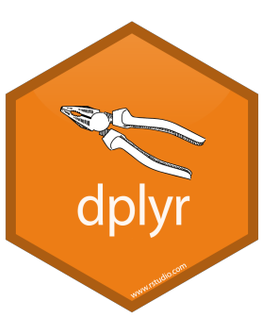
★ Reorder columns with select too!

Select columns name, height, mass, hair color, and skin color

```
dplyr::select(starwars, name:skin_color)
```

Select columns name, mass, skin color, hair color, and height (in order)

```
dplyr::select(starwars, name, mass, skin_color, hair_color, height)
```



`dplyr::select()` to keep columns based on names

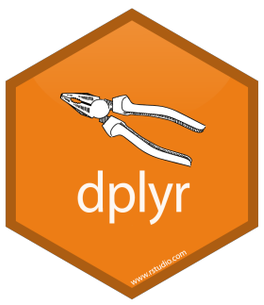
Combine filter and select

`dplyr::filter(dataframe, condition(s))`

`dplyr::select(dataframe, columns_to_keep)`

Challenge OYO:

[Keep: height greater than 100 &
Keep: humans &
Remove: brown hair color &
Remove: vehicles &
Keep: name, homeworld, height, species, hair color]



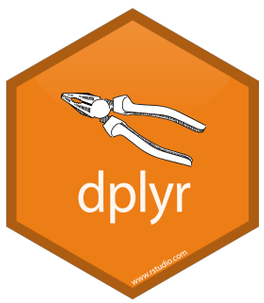
`dplyr::filter()` to keep rows based on values

`dplyr::select()` to keep columns based on names

Combine filter and select

	name	homeworld	height	species	hair_color
1	Luke Skywalker	Tatooine	172	Human	blond
2	Darth Vader	Tatooine	202	Human	none
3	Owen Lars	Tatooine	178	Human	brown, grey
4	Biggs Darklighter	Tatooine	183	Human	black
5	Obi-Wan Kenobi	Stewjon	182	Human	auburn, white
6	Anakin Skywalker	Tatooine	188	Human	blond
7	Wilhuff Tarkin	Eriadu	180	Human	auburn, grey
8	Palpatine	Naboo	170	Human	grey
9	Boba Fett	Kamino	183	Human	black
10	Lando Calrissian	Socorro	177	Human	black
11	Lobot	Bespin	175	Human	none
12	Mon Mothma	Chandrila	150	Human	auburn
13	Finis Valorum	Coruscant	170	Human	blond
14	Shmi Skywalker	Tatooine	163	Human	black
15	Mace Windu	Haruun Kal	188	Human	none
16	Gregar Typho	Naboo	185	Human	black

```
starwars %>%
    dplyr::filter(height > 100) %>%
    dplyr::filter(species == "Human") %>%
    dplyr::filter(hair_color != "brown") %>%
    dplyr::select(-vehicles) %>%
    dplyr::select(name, homeworld, height, species, hair_color)
```



`dplyr::mutate()`

`dplyr::mutate()` to add new (or change existing) columns

`dplyr::mutate()`

```
dplyr::mutate(dataframe,  new_column = expression)
```

1. Figure out what you want to do

calculate BMI of all starwars characters

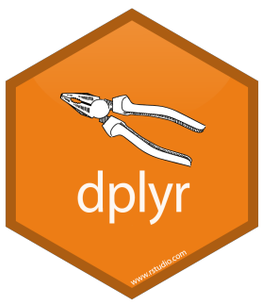
2. Describe your goal in words

cm

BMI = weight (kg) / [height (m)]^2

	name	height	mass	hair_color	skin_color	eye_color
1	Luke Skywalker	172	77.0	blond	fair	blue
2	C-3PO	167	75.0	*NA*	gold	yellow
3	R2-D2	96	32.0	*NA*	white, blue	red
4	Darth Vader	202	136.0	none	white	yellow

`?starwars`



`dplyr::mutate()` to add new (or change existing) columns

`dplyr::mutate()`

`dplyr::mutate(dataframe, new_column = expression)`

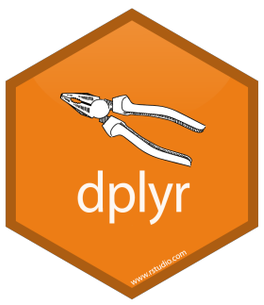
1. Figure out what you want to do

calculate BMI of all starwars characters

2. Describe your goal in words

BMI = weight (kg) / [height (m)]^2

First: convert height in cm to height in meters
Second: calculate BMI as new column



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

```
dplyr::mutate(dataframe,  new_column = expression)
```

starwars — height_m — `height / 100`

2. Describe your goal in words

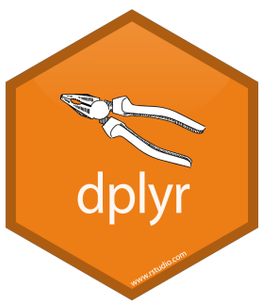
BMI = weight (kg) / [height (m)]^2

First: convert height in cm to height in meters

Second: calculate BMI as new column

3. Describe your goal in code

```
new_starwars <- dplyr::mutate(starwars, height_m = height / 100)
```



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

```
dplyr::mutate(dataframe,  new_column = expression)
```

new_starwars

bmi

`mass / height_m^2`

2. Describe your goal in words

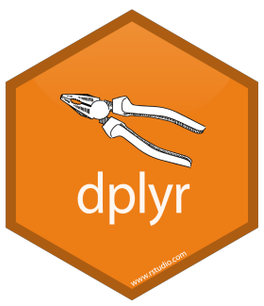
BMI = weight (kg) / [height (m)]^2

First: convert height in cm to height in meters

Second: calculate BMI as new column

3. Describe your goal in code

```
new_starwars <- starwars %>%
   dplyr::mutate(height_m = height / 100) %>%
   dplyr::mutate(bmi = mass / height_m^2)
```



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

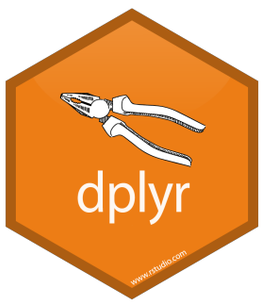
`dplyr::mutate(dataframe, new_column = expression)`

Select only name, height, mass, height_m, and bmi and view dataframe

```
new_starwars <- new_starwars %>%
  dplyr::select(name, height, mass, height_m, bmi)

View(new_starwars)
```

name	height	mass	height_m	bmi
Luke Skywalker	172	77.0	1.72	26.02758
C-3PO	167	75.0	1.67	26.89232
R2-D2	96	32.0	0.96	34.72222
Darth Vader	202	136.0	2.02	33.33007
Leia Organa	150	49.0	1.50	21.77778
Owen Lars	178	120.0	1.78	37.87401
Beru Whitesun lars	165	75.0	1.65	27.54821
R5-D4	97	32.0	0.97	34.00999
Biggs Darklighter	183	84.0	1.83	25.08286



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

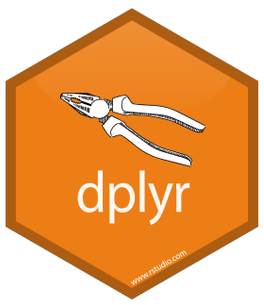
`dplyr::mutate(`dataframe, new_column = expression`)`

★ We can change existing columns if we use the same name

```
new_starwars <- starwars %>%
  dplyr::mutate(height = height / 100) %>%
  dplyr::mutate(bmi = mass / height^2)
```

```
View(new_starwars)
```

name	height	mass	bmi
Luke Skywalker	1.72	77.0	26.02758
C-3PO	1.67	75.0	26.89232
R2-D2	0.96	32.0	34.72222
Darth Vader	2.02	136.0	33.33007
Leia Organa	1.50	49.0	21.77778
Owen Lars	1.78	120.0	37.87401



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

```
dplyr::mutate(dataframe,  new_column = expression)
```

Make a new column to find the average height (in meters)

```
starwars %
  dplyr::m
  dplyr::m

?mean
```

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

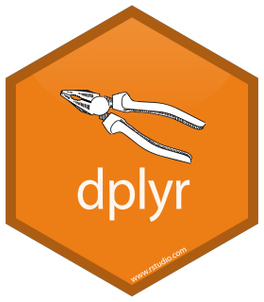
Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.



`dplyr::mutate()` to add new (or change existing) columns

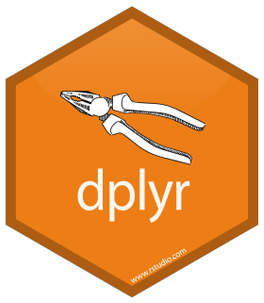
dplyr::**mutate**()

`dplyr::mutate(dataframe, new_column = expression)`

Make a new column to find the average height (in meters)

```
starwars %>%
   dplyr::mutate(height = height / 100) %>%
   dplyr::mutate(avg_height = mean(height, na.rm = TRUE))
```

name	height	mass	avg_height
Luke Skywalker	172	77.0	1.74358
C-3PO	167	75.0	1.74358
R2-D2	96	32.0	1.74358
Darth Vader	202	136.0	1.74358
Leia Organa	150	49.0	1.74358
Owen Lars	178	120.0	1.74358
Beru Whitesun lars	165	75.0	1.74358
R5-D4	97	32.0	1.74358



`dplyr::mutate()` to add new (or change existing) columns

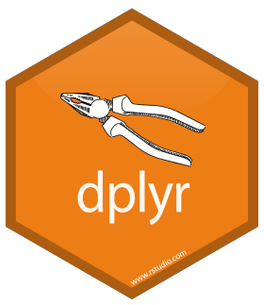
dplyr::**mutate**()

`dplyr::mutate(`dataframe, new_column = expression`)`

Standardize the heights of the starwars characters (height / avg_height)

```
starwars %>%
   dplyr::mutate(height = height / 100) %>%
   dplyr::mutate(avg_height = mean(height, na.rm = TRUE)) %>%
   dplyr::mutate(std_height = height / avg_height)
```

name	height	mass	avg_height	std_height
Luke Skywalker	1.72	77.0	1.74358	0.9864760
C-3PO	1.67	75.0	1.74358	0.9577993
R2-D2	0.96	32.0	1.74358	0.5505912
Darth Vader	2.02	136.0	1.74358	1.1585357
Leia Organa	1.50	49.0	1.74358	0.8602988
Owen Lars	1.78	120.0	1.74358	1.0208879
Beru Whitesun lars	1.65	75.0	1.74358	0.9463287
R5-D4	0.97	32.0	1.74358	0.5563266
Biggs Darklighter	1.83	84.0	1.74358	1.0495645
Obi-Wan Kenobi	1.82	77.0	1.74358	1.0438292



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

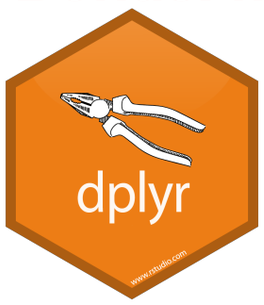
`dplyr::mutate(`dataframe, new_column = expression`)`

Make a new column to see if each character is above or below the average height

hint: try using `ifelse(condition, if_true_do_this, if_false_do_this)`

```
test <- starwars %>%
    dplyr::mutate(height = height / 100) %>%
    dplyr::mutate(avg_height = mean(height, na.rm = TRUE)) %>%
    dplyr::mutate(std_height = height / avg_height) %>%
    dplyr::select(name, height, mass, avg_height, std_height) %>%
    dplyr::mutate(relative_height =
                      ifelse(std_height > 1, "above", "below")) %>%
    dplyr::filter(relative_height == "below")
```

Bonus: make a new dataframe that only keeps characters with heights BELOW average



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

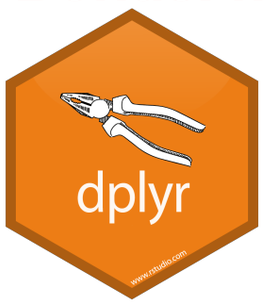
`dplyr::mutate(`dataframe, new_column = expression`)`

Make a new column to see if each character is above or below the average height

hint: try using `ifelse(condition, if_true_do_this, if_false_do_this)`

```
test <- starwars %>%
    dplyr::mutate(height = height / 100) %>%
    dplyr::mutate(avg_height = mean(height, na.rm = TRUE)) %>%
    dplyr::mutate(std_height = height / avg_height) %>%
    dplyr::select(name, height, mass, avg_height, std_height) %>%
    dplyr::mutate(relative_height =
                      ifelse(std_height > 1, "above", "below")) %>%
    dplyr::filter(std_height < 1)
```

Bonus: make a new dataframe that only keeps characters with heights BELOW average



`dplyr::mutate()` to add new (or change existing) columns

dplyr::**mutate**()

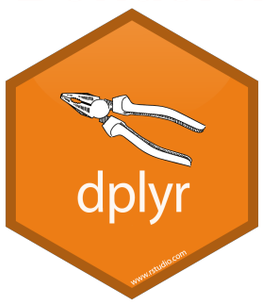
`dplyr::mutate(`dataframe, new_column = expression`)`

Make a new column to see if each character is above or below the average height

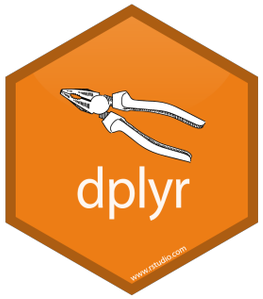
hint: try using `ifelse(condition, if_true_do_this, if_false_do_this)`

```
test <- starwars %>%
    dplyr::mutate(height = height / 100) %>%
    dplyr::mutate(avg_height = mean(height, na.rm = TRUE)) %>%
    dplyr::mutate(std_height = height / avg_height) %>%
    dplyr::select(name, height, mass, avg_height, std_height) %>%
    dplyr::mutate(relative_height =
                      ifelse(std_height > 1, "above", "below")) %>%
    dplyr::filter(height < avg_height)
```

Bonus: make a new dataframe that only keeps characters with heights BELOW average



`dplyr::mutate()` to add new (or change existing) columns



dplyr::**group_by**()

`dplyr::group_by()` to group rows by columns

`dplyr::group_by()`

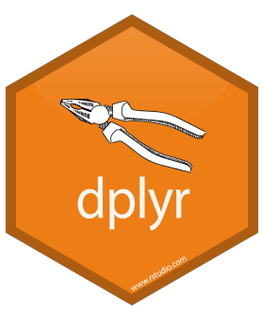
`dplyr::group_by(dataframe, column_to_group_by)`

Doesn't change how the data looks, changes how the data interacts with other dplyr verbs

Group starwars dataframe by gender

```
grouped_starwars <- starwars %>%
   dplyr::group_by(gender)
```

```
View(starwars)
View(grouped_starwars)
```



`dplyr::group_by()` to group rows by columns

dplyr::group_by()

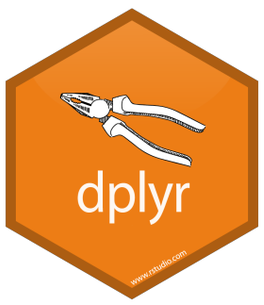
`dplyr::group_by(dataframe, column_to_group_by)`

ungrouped

	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*
4	Darth Vader	202	136.0	none	white	yellow	41.9	male
5	Leia Organa	150	49.0	brown	light	brown	19.0	female
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male
7	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male
14	Han Solo	180	80.0	brown	fair	brown	29.0	male
15	Greedo	173	74.0	*NA*	green	black	44.0	male
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphrodite
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male
19	Yoda	66	17.0	white	green	brown	896.0	male
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male
21	Boba Fett	183	78.2	black	fair	brown	31.5	male
22	IG-88	200	140.0	none	metal	red	15.0	none
23	Bossk	190	113.0	none	green	red	53.0	male
24	Lando Calrissian	177	79.0	black	dark	brown	31.0	male

grouped by gender

	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender
1	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male
2	C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*
3	R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*
4	Darth Vader	202	136.0	none	white	yellow	41.9	male
5	Leia Organa	150	49.0	brown	light	brown	19.0	female
6	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male
7	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female
8	R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*
9	Biggs Darklighter	183	84.0	black	light	brown	24.0	male
10	Obi-Wan Kenobi	182	77.0	auburn, white	fair	blue-gray	57.0	male
11	Anakin Skywalker	188	84.0	blond	fair	blue	41.9	male
12	Wilhuff Tarkin	180	*NA*	auburn, grey	fair	blue	64.0	male
13	Chewbacca	228	112.0	brown	unknown	blue	200.0	male
14	Han Solo	180	80.0	brown	fair	brown	29.0	male
15	Greedo	173	74.0	*NA*	green	black	44.0	male
16	Jabba Desilijic Tiure	175	1358.0	*NA*	green-tan, brown	orange	600.0	hermaphrodite
17	Wedge Antilles	170	77.0	brown	fair	hazel	21.0	male
18	Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male
19	Yoda	66	17.0	white	green	brown	896.0	male
20	Palpatine	170	75.0	grey	pale	yellow	82.0	male
21	Boba Fett	183	78.2	black	fair	brown	31.5	male
22	IG-88	200	140.0	none	metal	red	15.0	none
23	Bossk	190	113.0	none	green	red	53.0	male
24	Lando Calrissian	177	79.0	black	dark	brown	31.0	male



`dplyr::group_by()` to group rows by columns

`dplyr::group_by()`

```
dplyr::group_by(dataframe, column_to_group_by)
```

Doesn't change how the data looks, changes how the data interacts with other dplyr verbs

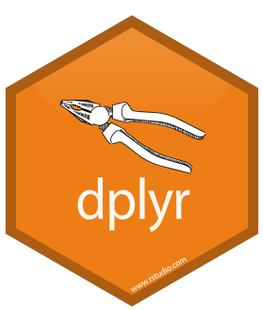
Group starwars dataframe by gender

```
grouped_starwars <- starwars %>%
  dplyr::group_by(gender)
```

Calculate average height *PER GENDER*

hint: group by gender FIRST

```
grouped_starwars <- starwars %>%
  dplyr::group_by(gender) %>%
  dplyr::mutate(avg_height = mean(height, na.rm = TRUE))
```



`dplyr::group_by()` to group rows by columns

dplyr::group_by()

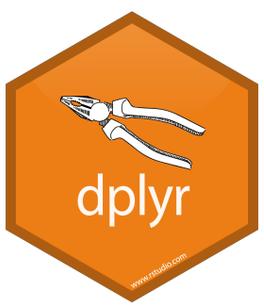
```
dplyr::group_by(dataframe, column_to_group_by)
```

Calculate average height *PER GENDER*

name	gender	height	avg_height
Luke Skywalker	male	1.72	1.792373
C-3PO	*NA*	1.67	1.200000
R2-D2	*NA*	0.96	1.200000
Darth Vader	male	2.02	1.792373
Leia Organa	female	1.50	1.654706
Owen Lars	male	1.78	1.792373
Beru Whitesun lars	female	1.65	1.654706
R5-D4	*NA*	0.97	1.200000
Biggs Darklighter	male	1.83	1.792373
Obi-Wan Kenobi	male	1.82	1.792373
Anakin Skywalker	male	1.88	1.792373
Wilhuff Tarkin	male	1.80	1.792373
Chewbacca	male	2.28	1.792373
Han Solo	male	1.80	1.792373

Calculate average height

name	height	mass	avg_height
Luke Skywalker	172	77.0	1.74358
C-3PO	167	75.0	1.74358
R2-D2	96	32.0	1.74358
Darth Vader	202	136.0	1.74358
Leia Organa	150	49.0	1.74358
Owen Lars	178	120.0	1.74358
Beru Whitesun lars	165	75.0	1.74358
R5-D4	97	32.0	1.74358



`dplyr::group_by()` to group rows by columns

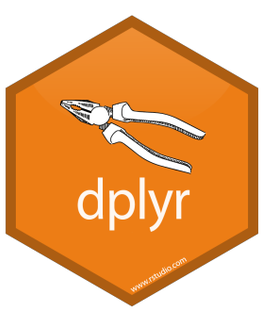
dplyr::group_by()

`dplyr::group_by(dataframe, column_to_group_by)`

★ `dplyr::ungroup()` removes all groups

Ungroup your grouped dataframe and re-calculate average height

```
ungrouped_starwars <- grouped_starwars %>%
   dplyr::ungroup() %>%
   dplyr::mutate(avg_height = mean(height, na.rm = TRUE))
```



`dplyr::group_by()` to group rows by columns

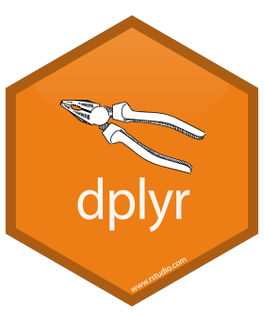
dplyr::group_by()

`dplyr::group_by(dataframe, column_to_group_by)`

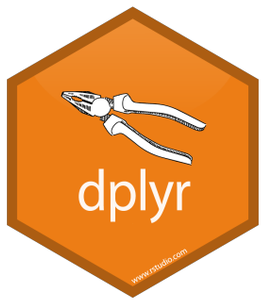
★ You can also group by multiple columns

Calculate the average height per gender AND eye color

```
grouped_starwars <- starwars %>%
   dplyr::group_by(gender, eye_color) %>%
   dplyr::mutate(avg_height = mean(height, na.rm = TRUE))
```



`dplyr::group_by()` to group rows by columns



`dplyr::summarize()`
`dplyr::summarise()`

`dplyr::summarize()` to condense multiple columns

dplyr::summarize()

`dplyr::summarize(dataframe, new_column = expression)`

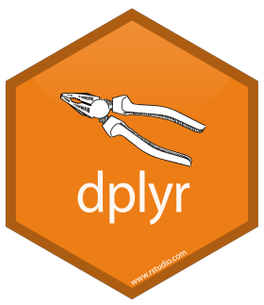
`summarize()` is similar to `mutate()` but only keeps grouped columns

Calculate average height *PER GENDER*

hint: group by gender FIRST

```
grouped_starwars <- starwars %>%
   dplyr::group_by(gender) %>%
   dplyr::summarize(avg_height = mean(height, na.rm = TRUE))
```

gender	avg_height
NA	120.0000
female	165.4706
hermaphrodite	175.0000
male	179.2373
none	200.0000



`dplyr::summarize()` to condense multiple columns

dplyr::summarize()

summarize() is similar to mutate() but only keeps grouped columns

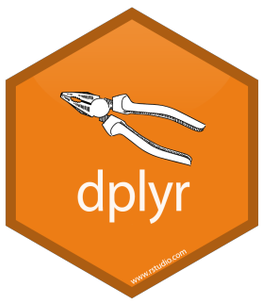
Compare `mutate()` and `summarize()` to calculate average height by gender

```
dplyr::mutate(avg_height = mean(height, na.rm = TRUE))
```

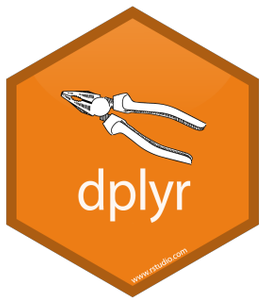
name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships	avg_height
Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")	179.2373
C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)	120.0000
R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*	Naboo	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)	120.0000
Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	character(0)	TIE Advanced x1	179.2373
Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	Imperial Speeder Bike	character(0)	165.4706
Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)	179.2373
Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)	165.4706
R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)	120.0000

```
dplyr::summarize(avg_height = mean(height, na.rm = TRUE))
```

gender	avg_height
NA	120.0000
female	165.4706
hermaphrodite	175.0000
male	179.2373
none	200.0000



`dplyr::summarize()` to condense multiple columns



dplyr::**arrange**()

`dplyr::arrange()` to reorder the rows

dplyr::**arrange**()

`dplyr::arrange(`dataframe, variable`)`

Arrange the starwars dataframe by homeworld

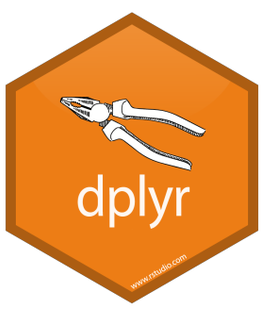
```
arranged_df <- dplyr::arrange(starwars, homeworld)
```

homeworld arranged A > Z

use `desc()` to arrange in descending order

```
arranged_df <- dplyr::arrange(starwars, desc(homeworld))
```

homeworld arranged Z > A



`dplyr::arrange()` to reorder the rows

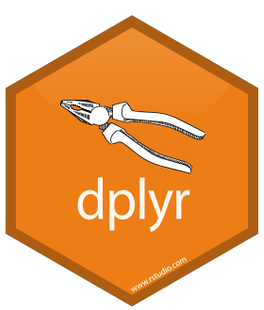
dplyr::**arrange**()

`dplyr::arrange(`dataframe, variable`)`

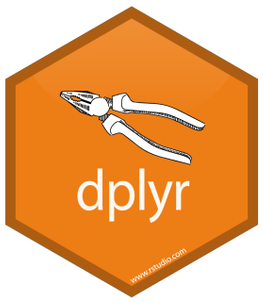
Arrange starwars by species first then height

```
arranged_df <- dplyr::arrange(starwars, species, height)
```

name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species
Ratts Tyerell	79	15.0	none	grey, blue	unknown	NA	male	Aleen Minor	Aleena
Dexter Jettster	198	102.0	none	brown	yellow	NA	male	Ojom	Besalisk
Ki-Adi-Mundi	198	82.0	white	pale	yellow	92.0	male	Cerea	Cerean
Mas Amedda	196	NA	none	blue	blue	NA	male	Champala	Chagrian
Zam Wesell	168	55.0	blonde	fair, green, yellow	yellow	NA	female	Zolan	Clawdite
R2-D2	96	32.0	NA	white, blue	red	33.0	NA	Naboo	Droid
R5-D4	97	32.0	NA	white, red	red	NA	NA	Tatooine	Droid
C-3PO	167	75.0	NA	gold	yellow	112.0	NA	Tatooine	Droid
IG-88	200	140.0	none	metal	red	15.0	none	NA	Droid
BB8	NA	NA	none	none	black	NA	none	NA	Droid
Sebulba	112	40.0	none	grey, red	orange	NA	male	Malastare	Dug
Wicket Systri Warrick	88	20.0	brown	brown	brown	8.0	male	Endor	Ewok
Poggle the Lesser	183	80.0	none	green	yellow	NA	male	Geonosis	Geonosian
Jar Jar Binks	196	66.0	none	orange	orange	52.0	male	Naboo	Gungan



`dplyr::arrange()` to reorder the rows



`dplyr::xxx_join()`

`dplyr::xxx_join()` to combine datasets

dplyr::**xxx_join**()

`dplyr::`**`xxx_join`**`(dataframe1, dataframe2, by = column_in_common)`

left_join()

right_join()

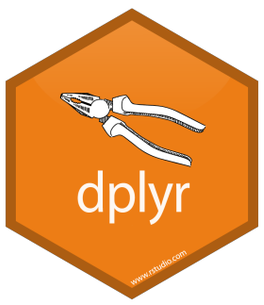
full_join()

inner_join()

semi_join()

anti_join()

nest_join()



`dplyr::xxx_join()` to combine datasets

dplyr::xxx_join()

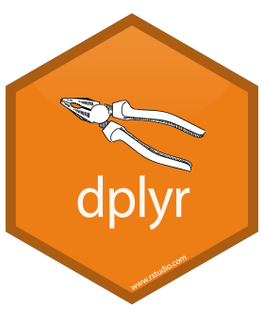
`dplyr::xxx_join(dataframe1, dataframe2, by = column_in_common)`

data/join_df1.csv

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

data/join_df2.csv

A	D
red	3
orange	5
yellow	7
green	1
indigo	3
blue	6
pink	9



`dplyr::xxx_join()` to combine datasets

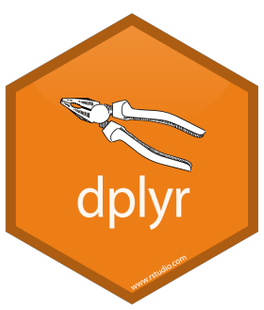
dplyr::left_join()

```
dplyr::left_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

A	D
red	3
orange	5
yellow	7
green	1
indigo	3
blue	6
pink	9

```
dplyr::left_join(df1, df2)
```



`dplyr::left_join()` to combine datasets

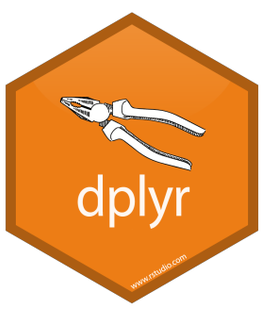
dplyr::left_join()

```
dplyr::left_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA

- Return all rows from dataframe1
- Return all columns from dataframe1 and dataframe2
- Rows in dataframe1 with no match in dataframe2 will be returned as NA

```
dplyr::left_join(df1, df2)
```



`dplyr::left_join()` to combine datasets

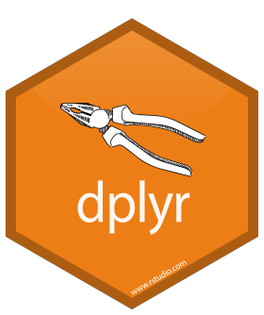
dplyr::right_join()

```
dplyr::right_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

A	D
red	3
orange	5
yellow	7
green	1
indigo	3
blue	6
pink	9

```
dplyr::right_join(df1, df2)
```



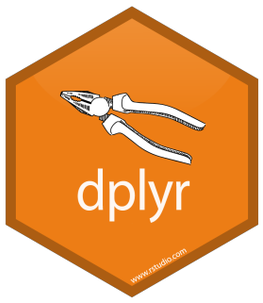
`dplyr::right_join()` to combine datasets

dplyr::**right_join**()

```
dplyr::right_join(dataframe1, dataframe2, by = column_in_common)
```

- Return all rows from dataframe2
- Return all columns from dataframe1 and dataframe2
- Rows in dataframe2 with no match in dataframe1 will be returned as NA

A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
pink	NA	NA	9




```
dplyr::right_join(df1, df2)
```

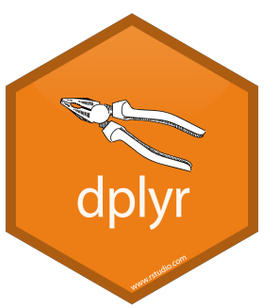
`dplyr::right_join()` to combine datasets

dplyr::full_join()

```
dplyr::full_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

A	D
red	3
orange	5
yellow	7
green	1
indigo	3
blue	6
pink	9




```
dplyr::full_join(df1, df2)
```

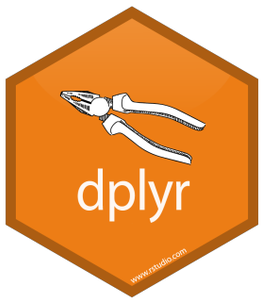
`dplyr::full_join()` to combine datasets

dplyr::full_join()

```
dplyr::full_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA
pink	NA	NA	9

- Return all rows from dataframe1 and data frame2
- Return all columns from dataframe1 and dataframe2
- Where there are not matching values, return NA




```
dplyr::full_join(df1, df2)
```

`dplyr::full_join()` to combine datasets

dplyr::full_join()

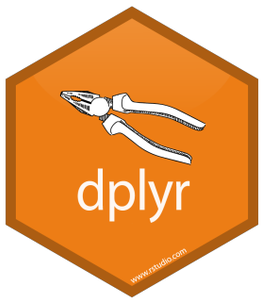
```
dplyr::full_join(dataframe1, dataframe2, by = column_in_common)
```

A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA
pink	NA	NA	9

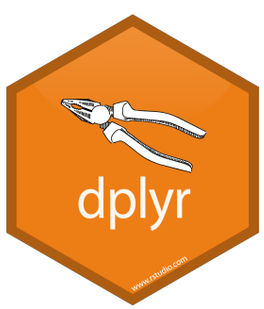
- Return all rows from dataframe1 and data frame2
- Return all columns from dataframe1 and dataframe2
- Where there are not matching values, return NA

Specify which column to join by

```
dplyr::full_join(df1, df2, by = "A")
```



`dplyr::full_join()` to combine datasets

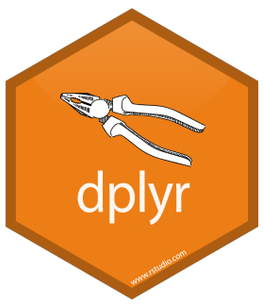


`dplyr::bind_rows()`
`dplyr::bind_cols()`

`dplyr::bind_xxx()` to combine datasets

dplyr::**bind_rows**()

```
dplyr::bind_rows(dataframe1, dataframe2)
```



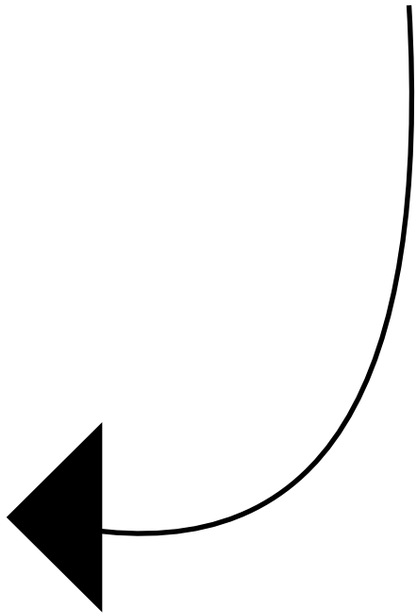
`dplyr::bind_rows()` to combine datasets

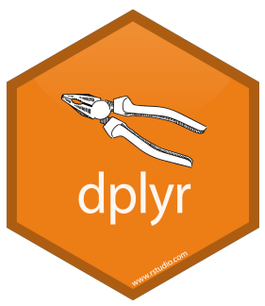
dplyr::bind_rows()

```
dplyr::bind_rows(dataframe1, dataframe2)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

A	B	C
pink	1	3
black	2	2





`dplyr::bind_rows()` to combine datasets

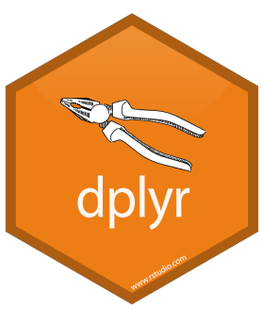
`dplyr::bind_rows()`

```
dplyr::bind_rows(dataframe1, dataframe2)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2
pink	1	3
black	2	2

- Does not “join”, simply merges two datasets.
- Requires same number of columns in each dataset
- Does not check to make sure each column is the same, **be careful!**

```
dplyr::bind_rows(df1, df3)
```



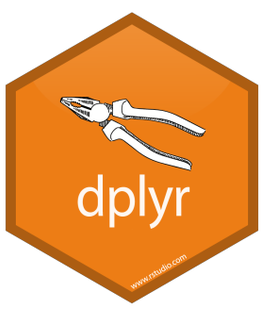
`dplyr::bind_rows()` to combine datasets

dplyr::bind_cols()

```
dplyr::bind_cols(dataframe1, dataframe2)
```

A	B	C
red	2	3
orange	4	6
yellow	8	9
green	0	0
indigo	3	3
blue	1	1
purple	5	5
white	8	2

D
3
5
7
1
3
6
NA
NA



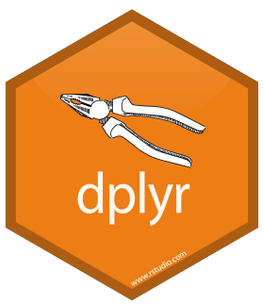
`dplyr::bind_cols()` to combine datasets

dplyr::bind_cols()

```
dplyr::bind_cols(dataframe1, dataframe2)
```

A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA

- Does not “join”, simply merges two datasets.
- Requires same number of rows in each dataset
- Does not check to make sure each row is the same, **be careful!**



`dplyr::bind_cols()` to combine datasets

dplyr::bind_cols()

```
dplyr::bind_cols(dataframe1, dataframe2)
```

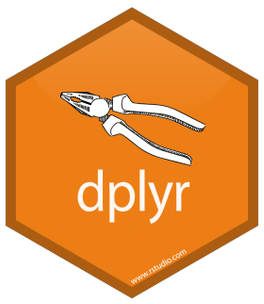
A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA

Make “d” vector

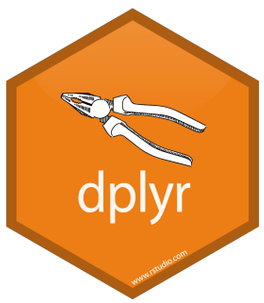
```
d <- c(3, 5, 7, 1, 3, 6, NA, NA)
```

Bind df1 with d vector

```
dplyr::bind_cols(df1, D = d)
```



`dplyr::bind_cols()` to combine datasets



`dplyr::rename()`

`dplyr::rename()` to give columns new names

dplyr::**rename**()

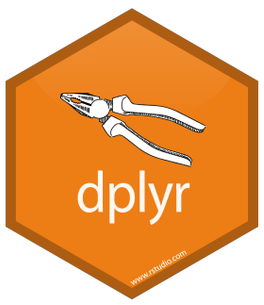
`dplyr::rename(`dataframe, new_name = old_name`)`

Rename the “name” column to “character”

```
renamed <- starwars %>%
  dplyr::rename(character = name)
```

Rename the “name” column to “character” and the “mass” column to “weight”

```
renamed <- starwars %>%
  dplyr::rename(character = name, weight = mass)
```



`dplyr::rename()` to give columns new names

dplyr::rename()

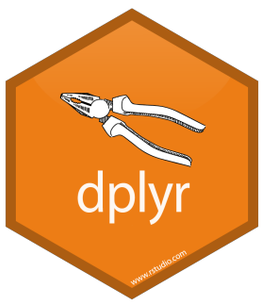
```
dplyr::rename(dataframe,  new_name = old_name)
```

You can also use `select()` to rename columns

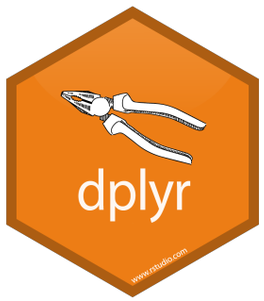
Rename the “name” column to “character” and the “mass” column to “weight” using `select()`

```
renamed <- starwars %>%
   dplyr::select(character = name, weight = mass)
```

character	weight
Luke Skywalker	77.0
C-3PO	75.0
R2-D2	32.0
Darth Vader	136.0
Leia Organa	49.0
Owen Lars	120.0



`dplyr::rename()` to give columns new names



`dplyr::pull()`

`dplyr::pull()` to return a column as a vector

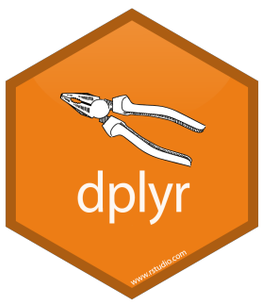
`dplyr::pull()`

`dplyr::pull(dataframe, variable)`

Extract the birth year column as a vector

```
starwars %>%
  dplyr::pull(birth_year)
```

```
 [1]  19.0 112.0  33.0  41.9  19.0  52.0  47.0    NA  24.0  57.0  41.9  64.0 200.0  29.0  44.0 600.0  21.0    NA 896.0  82.0  31.5
[22]  15.0  53.0  31.0  37.0  41.0  48.0    NA   8.0    NA  92.0    NA  91.0  52.0    NA    NA    NA    NA    NA  62.0  72.0  54.0
[43]    NA  48.0    NA    NA    NA  72.0  92.0    NA    NA    NA    NA    NA  22.0    NA    NA    NA  82.0    NA  58.0  40.0    NA
[64] 102.0  67.0  66.0    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
[85]    NA    NA  46.0
```



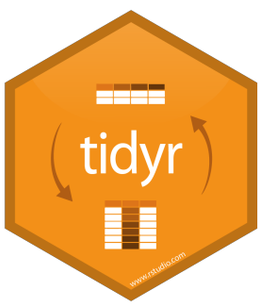
`dplyr::pull()` to return a column as a vector

Introduction: **tidyr**

- Collection of functions as **verbs** to easily "tidy" your data

<u>Functions:</u>

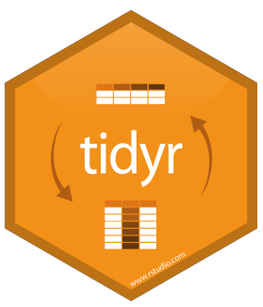
- `gather()` to collapse multiple columns
- `spread()` to expand one column to multiple
- `unite()` to combine multiple columns into one
- `separate()` to split one column into two
- `separate_rows()` to split one cell into several rows
- `drop_na()` to remove rows with NA values



Wide vs. Long data

Wide

Long



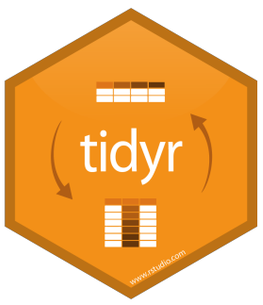
Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



Which is better?

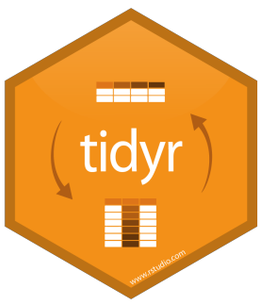
Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



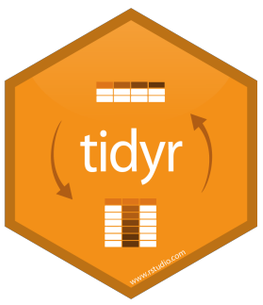
Q1: Find the average of each student's midterms

Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

```
df %>%
   dplyr::mutate(average = mean(midterm_1, midterm_2, midterm_3))
```



Q1: Find the average of each student's midterms

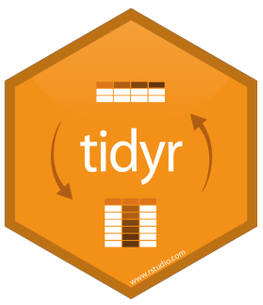
Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3	average
samantha	72	80	81	77.6
taylor	91	92	90	91
kelsey	83	74	90	82.3
ramona	65	71	75	70.3

```
df %>%
   dplyr::mutate(average = mean(midterm_1, midterm_2, midterm_3))
```

Imagine you have 100 midterms to average… this would be difficult to script



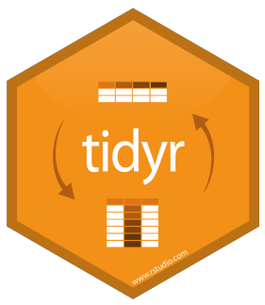
Q1: Find the average of each student's midterms

Wide vs. Long data

```
df %>%
   dplyr::group_by(name) %>%
   dplyr::mutate(average = mean(score))
```

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



Q1: Find the average of each student's midterms

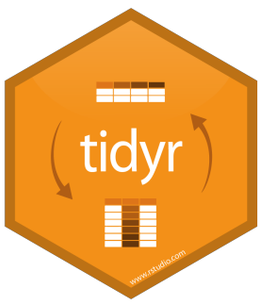
Wide vs. Long data

```
df %>%
   dplyr::group_by(name) %>%
   dplyr::mutate(average = mean(score))
```

The script won't change no matter how many midterms you have to score!

Long

name	midterm	score	average
samantha	midterm_1	72	77.6
samantha	midterm_2	80	77.6
samantha	midterm_3	81	77.6
taylor	midterm_1	91	91
taylor	midterm_2	92	91
taylor	midterm_3	90	91
kelsey	midterm_1	83	82.3
kelsey	midterm_2	74	82.3
kelsey	midterm_3	90	82.3
ramona	midterm_1	65	70.3
ramona	midterm_2	71	70.3
ramona	midterm_3	75	70.3



Q1: Find the average of each student's midterms

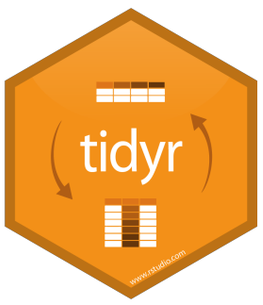
Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



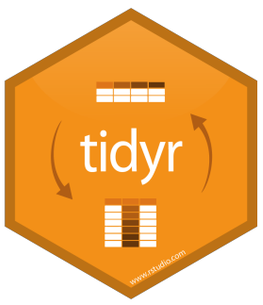
Q2: Plot each student's score by midterm

Wide vs. Long data

```
df %>%
   ggplot2::ggplot(.) +
   ggplot2::aes(x = name, y = score, fill = midterm) +
   ggplot2::geom_bar(stat = "identity", position = "dodge")
```

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75

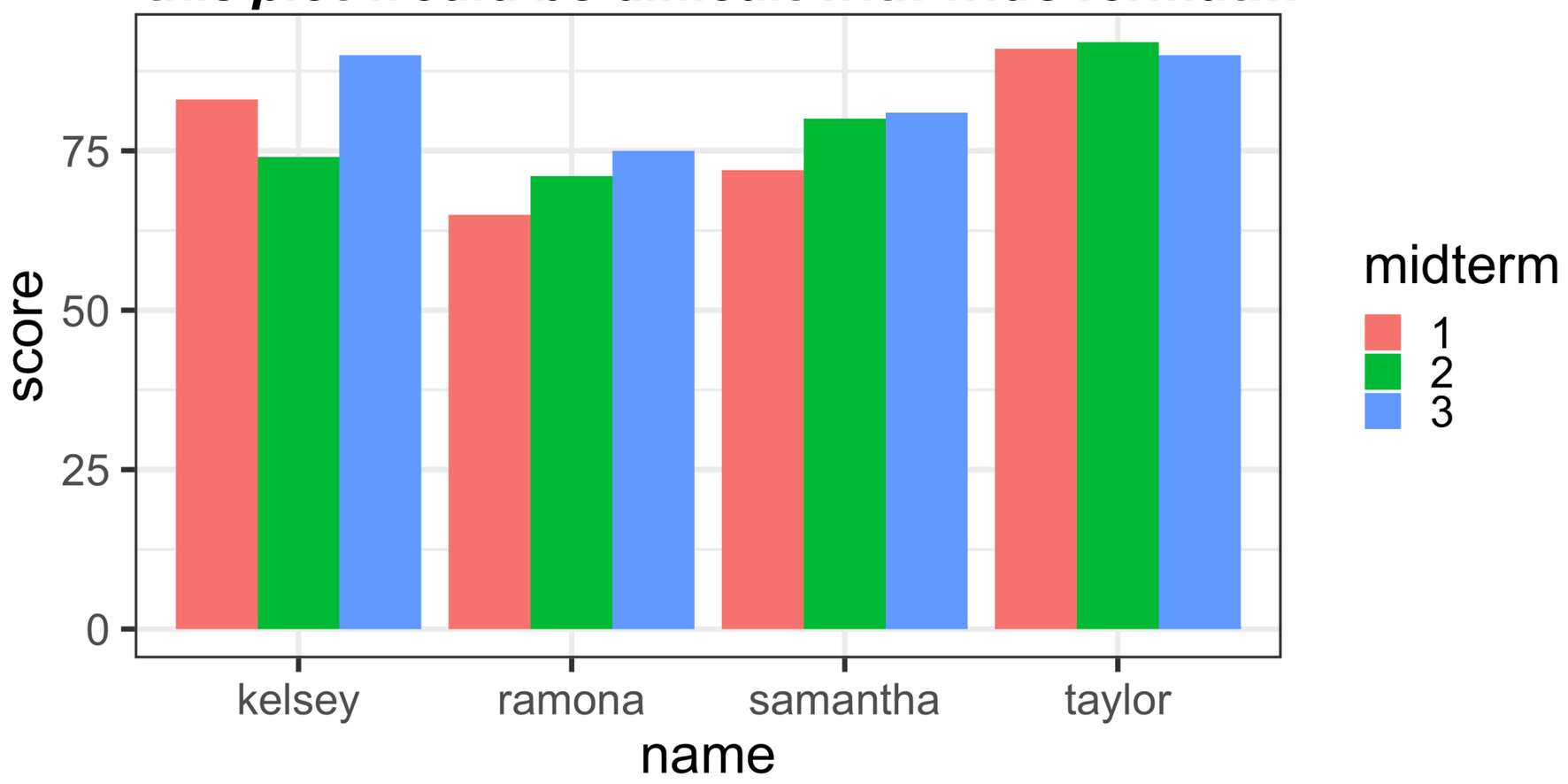


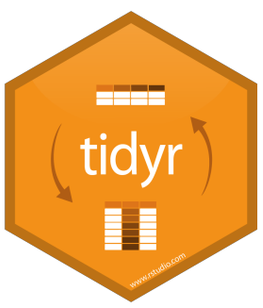
Q2: Plot each student's score by midterm

Wide vs. Long data

```
df %>%
   ggplot2::ggplot(.) +
   ggplot2::aes(x = name, y = score, fill = midterm) +
   ggplot2::geom_bar(stat = "identity", position = "dodge")
```

**** this plot would be difficult with wide format…***





Q2: Plot each student's score by midterm

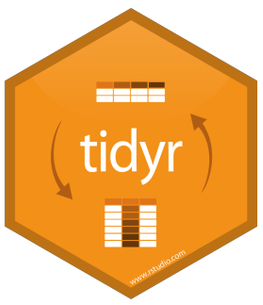
Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

Long

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



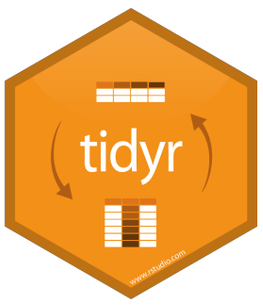
Q3: Find the ratio between midterm 1 and 2

Wide vs. Long data

Wide

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

```
df %>%
   dplyr::mutate(ratio = midterm_1 / midterm_2)
```



Q3: Find the ratio between midterm 1 and 2

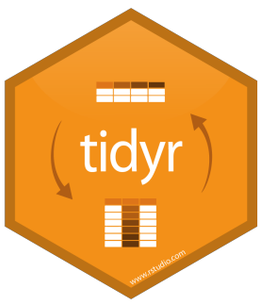
Wide vs. Long data

Wide

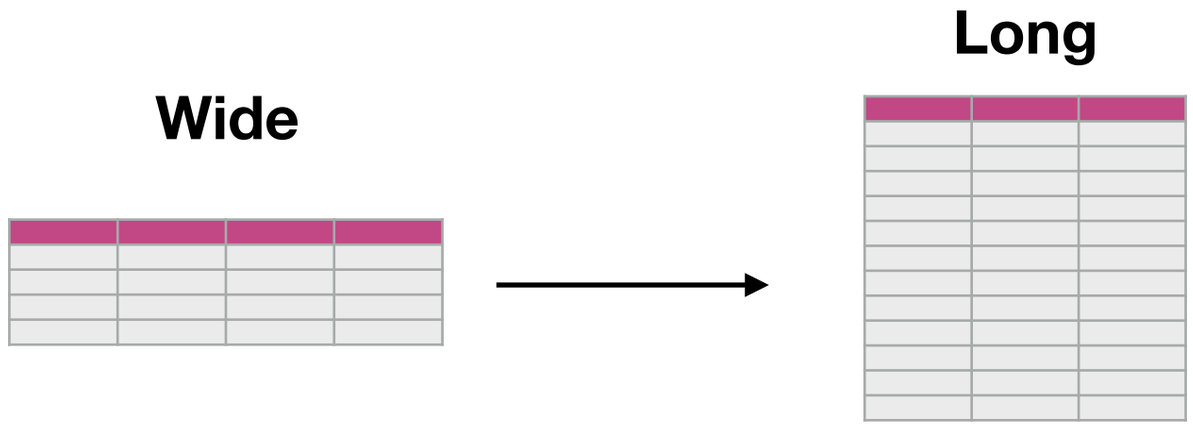
name	midterm_1	midterm_2	midterm_3	ratio
samantha	72	80	81	0.9
taylor	91	92	90	0.989
kelsey	83	74	90	1.12
ramona	65	71	75	0.915

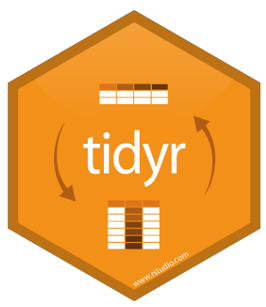
```
df %>%
   dplyr::mutate(ratio = midterm_1 / midterm_2)
```

This would be more difficult to do with the long data…



Q3: Find the ratio between midterm 1 and 2



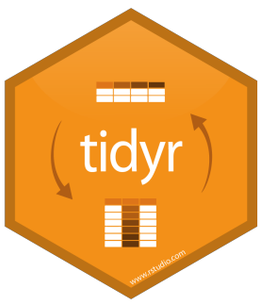


`tidyr::gather()`

`tidyr::gather()` to collapse multiple columns

`tidyr::gather()`

`tidyr::gather(`dataframe, key, value, columns_to_include`)`

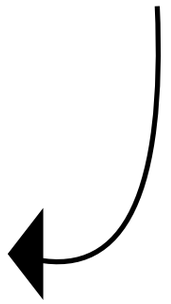


`tidyr::gather()` to collapse multiple columns

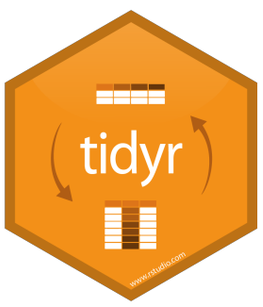
`tidyr::gather()`

`tidyr::gather(dataframe, key, value, columns_to_include)`

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75



name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



`tidyr::gather()` to collapse multiple columns

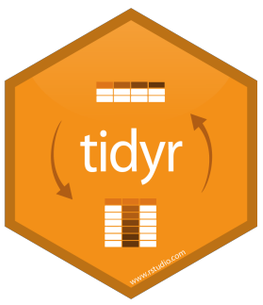
`tidyr::gather()`

`tidyr::gather(`dataframe, key, value, columns_to_include`)`

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75

```
long_df <- tidyr::gather(wide_df, midterm, score,
                         midterm_1:midterm_3)
```



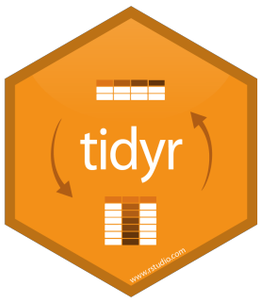
`tidyr::gather()` to collapse multiple columns

`tidyr::gather()`

`tidyr::gather(dataframe, key, value, columns_to_include)`

Convert the starwars dataframe from wide to long, containing three columns: *name, characteristic, value*

name	characteristic	value
Luke Skywalker	height	172
C-3PO	height	167
R2-D2	height	96
Darth Vader	height	202
Leia Organa	height	150
Owen Lars	height	178
Beru Whitesun lars	height	165
R5-D4	height	97
Biggs Darklighter	height	183
Obi-Wan Kenobi	height	182
Anakin Skywalker	height	188
Wilhuff Tarkin	height	180



`tidyr::gather()` to collapse multiple columns

`tidyr::gather()`

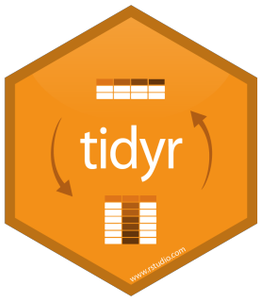
`tidyr::gather(dataframe, key, value, columns_to_include)`

Convert the starwars dataframe from wide to long, containing three columns: *name, characteristic, value*

```
gathered <- starwars %>%
  tidyr::gather(characteristic, value, height:starships)
```

You can also select which columns NOT to include instead

```
gathered <- starwars %>%
  tidyr::gather(characteristic, value, -name)
```



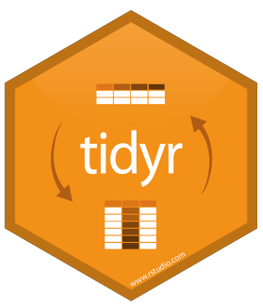
`tidyr::gather()` to collapse multiple columns

`tidyr::gather()`

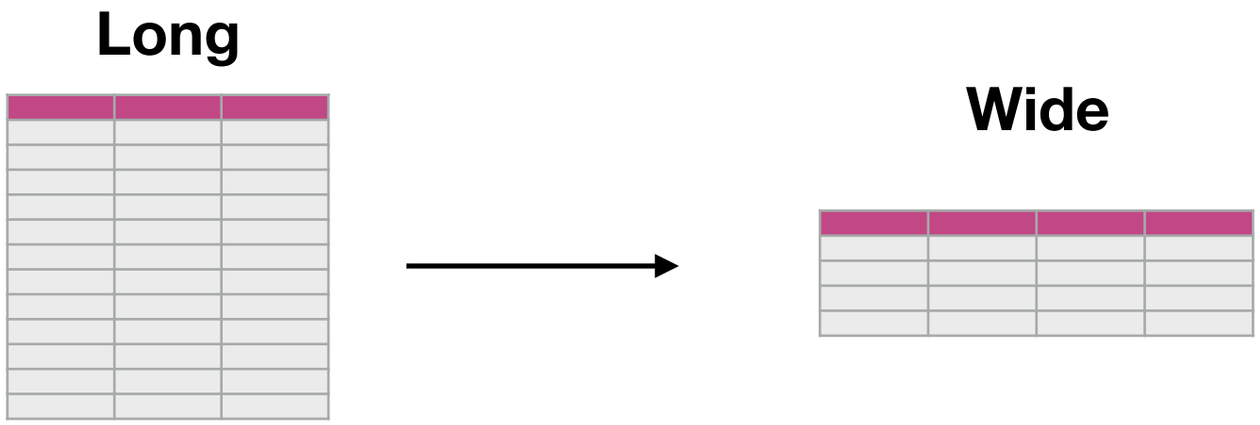
`tidyr::gather(dataframe, key, value, columns_to_include)`

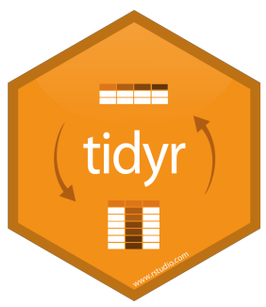
Gather height, mass, eye color, skin color, and gender

```
gathered <- starwars %>%
  tidyr::gather(characteristic, value, height, mass,
                eye_color, skin_color, gender)
```



`tidyr::gather()` to collapse multiple columns



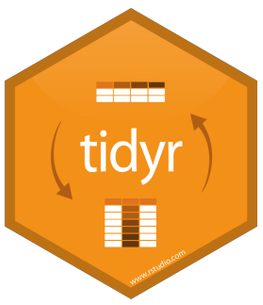


`tidyr::spread()`

`tidyr::spread()` to expand one column to multiple

`tidyr::spread()`

`tidyr::spread(`dataframe, key, value`)`



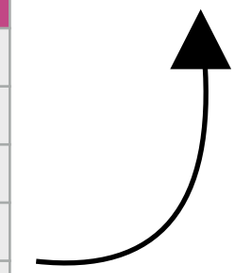
`tidyr::spread()` to expand one column to multiple

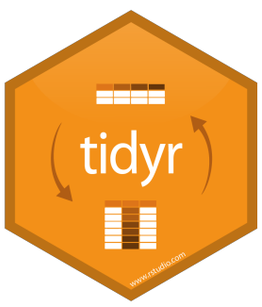
`tidyr::spread()`

`tidyr::spread(dataframe, key, value)`

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75





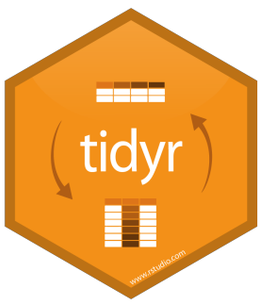
`tidyr::spread()` to expand one column to multiple

`tidyr::spread()`

`tidyr::spread(`dataframe, key, value`)`

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_1	83
kelsey	midterm_2	74
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75



`tidyr::spread()` to expand one column to multiple

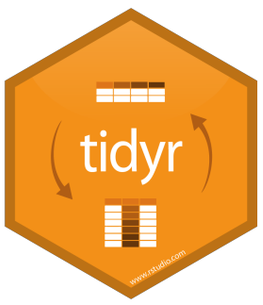
`tidyr::spread()`

`tidyr::spread(`dataframe, key, value`)`

name	midterm_1	midterm_2	midterm_3
samantha	72	80	81
taylor	91	92	90
kelsey	83	74	90
ramona	65	71	75

name	midterm	score
samantha	midterm_1	72
samantha	midterm_2	80
samantha	midterm_3	81
taylor	midterm_1	91
taylor	midterm_2	92
taylor	midterm_3	90
kelsey	midterm_3	90
ramona	midterm_1	65
ramona	midterm_2	71
ramona	midterm_3	75

```
wide_df <- tidyr::spread(long_df, midterm, score)
```



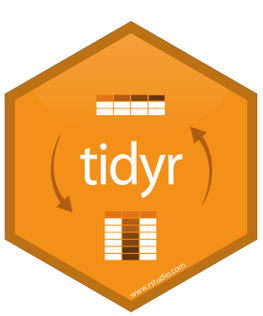
`tidyr::spread()` to expand one column to multiple

`tidyr::spread()`

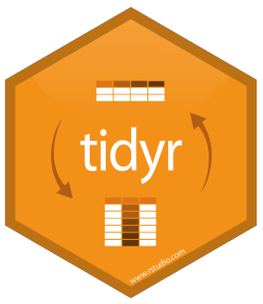
`tidyr::spread(`dataframe, key, value`)`

Un-gather (spread) the starwars data frame

```
ungathered <- gathered %>%
  tidyr::spread(characteristic, value)
```



`tidyr::spread()` to expand one column to multiple



`tidyr::unite()`

`tidyr::unite()` to combine multiple columns into one

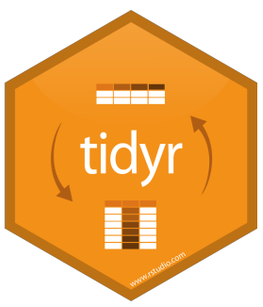
`tidyr::unite()`

`tidyr::unite(`dataframe, old_columns, `col =` new_column, `sep =` separator`)`

Combine name and homeworld into one column named character
(e.g. Luke Skywalker, Tatooine)

```
united <- starwars %>%
  tidyr::unite(name, homeworld, col = "character", sep = ", ")
```

character	height	mass	hair_color	skin_color	eye_color	birth_year
Luke Skywalker, Tatooine	172	77.0	blond	fair	blue	19.0
C-3PO, Tatooine	167	75.0	*NA*	gold	yellow	112.0
R2-D2, Naboo	96	32.0	*NA*	white, blue	red	33.0
Darth Vader, Tatooine	202	136.0	none	white	yellow	41.9
Leia Organa, Alderaan	150	49.0	brown	light	brown	19.0
Owen Lars, Tatooine	178	120.0	brown, grey	light	blue	52.0
Beru Whitesun lars, Tatooine	165	75.0	brown	light	blue	47.0



`tidyr::unite()` to combine multiple columns into one

tidyr::**unite**()

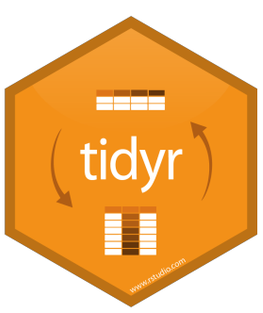
`tidyr::unite(`dataframe, old_columns, `col =` new_column, `sep =` separator`)`

Combine name, homeworld, and species into one column named character
(e.g. Luke Skywalker, Tatooine, Human)

```
united <- starwars %>%
  tidyr::unite(name, homeworld, species,
  col = "character", sep = ", ")
```

Combine name, homeworld, and species into one column named character
(e.g. Luke Skywalker, Tatooine (Human))

```
united <- starwars %>%
   tidyr::unite(name, homeworld, col = "character", sep = ", ") %>%
   dplyr::mutate(species = paste("(", species, ")", sep = "")) %>%
   tidyr::unite(character, species, col = "character", sep = " ")
```



`tidyr::unite()` to combine multiple columns into one

`tidyr::unite()`

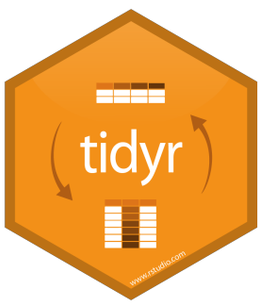
`tidyr::unite(`dataframe, old_columns, `col =` new_column, `sep =` separator`)`

★ Keep the original columns with argument `remove = FALSE`

Combine name, homeworld, and species into one column named character but keep name and homeworld columns also

(e.g. Luke Skywalker, Tatooine, Human)

```
united <- starwars %>%
   tidyr::unite(name, homeworld, species,
   col = “character", sep = “, “, remove = “FALSE”)
```



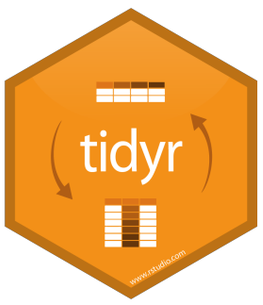
`tidyr::unite()` to combine multiple columns into one

`tidyr::unite()`

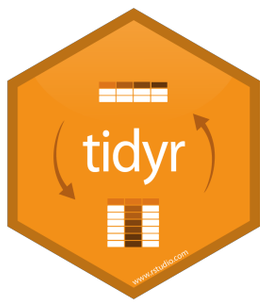
`tidyr::unite(`dataframe, old_columns, `col =` new_column, `sep =` separator`)`

Keep the original columns with argument `remove = FALSE`

character	name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld
Luke Skywalker, Tatooine	Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine
C-3PO, Tatooine	C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*	Tatooine
R2-D2, Naboo	R2-D2	96	32.0	*NA*	white, blue	red	33.0	*NA*	Naboo
Darth Vader, Tatooine	Darth Vader	202	136.0	none	white	yellow	41.9	male	Tatooine
Leia Organa, Alderaan	Leia Organa	150	49.0	brown	light	brown	19.0	female	Alderaan
Owen Lars, Tatooine	Owen Lars	178	120.0	brown, grey	light	blue	52.0	male	Tatooine
Beru Whitesun lars, Tatooine	Beru Whitesun lars	165	75.0	brown	light	blue	47.0	female	Tatooine
R5-D4, Tatooine	R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine
Biggs Darklighter, Tatooine	Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine



`tidyr::unite()` to combine multiple columns into one



`tidyr::separate()`

`tidyr::separate()` to split one column into two

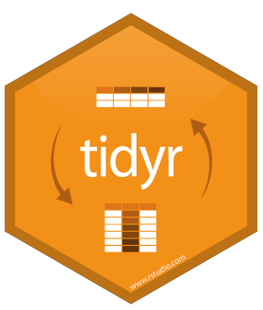
`tidyr::separate()`

`tidyr::separate(`dataframe, old_column, `into = c(`new_columns), `sep =` separator`)`

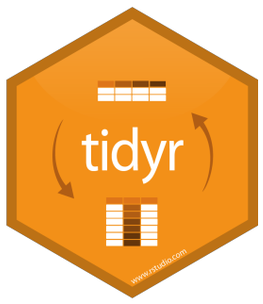
Separate the 'character' column back into name and homeworld

```
separated <- starwars %>%
tidyr::unite(name, homeworld, col = "character", sep = ", ") %>%
tidyr::separate(character, into = c("name", "homeworld"), sep = ",")
```

Keep the original columns with argument `remove = FALSE`



`tidyr::separate()` to split one column into two



`tidyr::separate_rows()`

`tidyr::separate_rows()` to split one cell into several rows

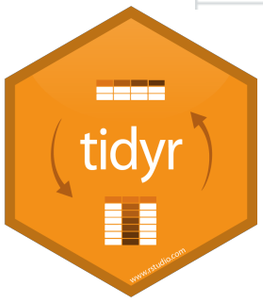
`tidyr::separate_rows()`

`tidyr::separate_rows(`dataframe, column(s), `sep =` separator`)`

Split the films column so that each film is represented in its own row per character

name	films
Luke Skywalker	c("Revenge of the Sith", "Return of the Jedi", "The Emp...
C-3PO	c("Attack of the Clones", "The Phantom Menace", "Rev...
R2-D2	c("Attack of the Clones", "The Phantom Menace", "Rev...
Darth Vader	c("Revenge of the Sith", "Return of the Jedi", "The Emp...
Leia Organa	c("Revenge of the Sith", "Return of the Jedi", "The Emp...
Owen Lars	c("Attack of the Clones", "Revenge of the Sith", "A New...
Beru Whitesun lars	c("Attack of the Clones", "Revenge of the Sith", "A New...
R5-D4	A New Hope
Biggs Darklighter	A New Hope
Obi-Wan Kenobi	c("Attack of the Clones", "The Phantom Menace", "Rev...
Anakin Skywalker	c("Attack of the Clones", "The Phantom Menace", "Rev...
Wilhuff Tarkin	c("Revenge of the Sith", "A New Hope")

name	films
Luke Skywalker	c("Revenge of the Sith"
Luke Skywalker	"Return of the Jedi"
Luke Skywalker	"The Empire Strikes Back"
Luke Skywalker	"A New Hope"
Luke Skywalker	"The Force Awakens")
C-3PO	c("Attack of the Clones"
C-3PO	"The Phantom Menace"
C-3PO	"Revenge of the Sith"
C-3PO	"Return of the Jedi"
C-3PO	"The Empire Strikes Back"
C-3PO	"A New Hope")
R2-D2	c("Attack of the Clones"



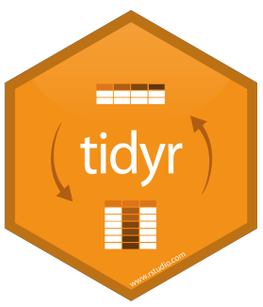
`tidyr::separate_rows()` to split one cell into several rows

tidyr::separate_rows()

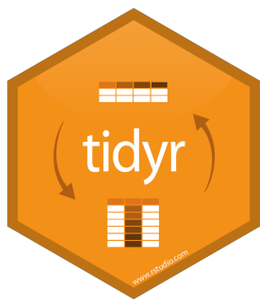
`tidyr::separate_rows(`dataframe, column(s), `sep = ` separator`)`

Split the films column so that each film is represented in its own row per character

```
separated <- starwars %>%
  tidyr::separate_rows(films, sep = ",")
```



`tidyr::separate_rows()` to split one cell into several rows



`tidyr::drop_na()`

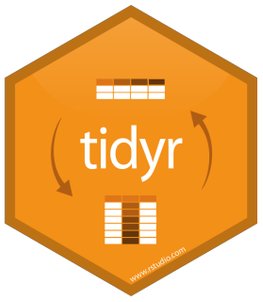
`tidyr::drop_na()` to remove rows with NA values

`tidyr::drop_na()`

`tidyr::drop_na(dataframe, column_with_na)`

Remove all observations with no species

```
no_species <- starwars %>%
  tidyr::drop_na(species)
```



`tidyr::drop_na()` to remove rows with NA values

Introduction: **readr**

- Collection of functions to import and export your data quickly and efficiently

Functions:

- `read_csv()` and `read_tsv()` to import data frames
- `write_csv()` and `write_tsv()` to export data frames

readr::**write_csv**()
readr::**write_tsv**()

`readr::write_csv()` to export data frames

readr::**write_csv**()

`readr::write_csv(object, file_path)`

Save the starwars data frame (name:species) as a .csv file

```
starwars %>%
  dplyr::select(name:species) %>%
  readr::write_csv("starwars.csv")
```

★ Remove column names with `col_names = FALSE`

```
starwars %>%
   dplyr::select(name:species) %>%
   readr::write_csv("starwars.csv", col_names = FALSE)
```

`readr::read_csv()` to import data frames as tibbles

`readr::write_delim()`

`readr::write_delim(`object, file_path, `delim = " ")`

Can write as any type of file (.csv, .tsv, .txt, .fwf etc…)

```
starwars %>%
  dplyr::select(name:species) %>%
  readr::write_tsv("starwars.csv")
```

```
starwars %>%
   dplyr::select(name:species) %>%
   readr::write_delim("starwars.csv", delim = ";")
```

`readr::read_csv()` to import data frames as tibbles

`readr::read_csv()`
`readr::read_tsv()`

`readr::read_csv()` to import data frames as tibbles

readr::read_csv()

`readr::read_csv(file)`

Read the starwars .csv file you just saved

hint: where is your working directory?

```
new_starwars <- readr::read_csv("starwars.csv")
```

```
> new_starwars <- readr::read_csv("starwars.csv")
Parsed with column specification:
cols(
  name = col_character(),
  height = col_double(),
  mass = col_double(),
  hair_color = col_character(),
  skin_color = col_character(),
  eye_color = col_character(),
  birth_year = col_double(),
  gender = col_character(),
  homeworld = col_character(),
  species = col_character()
)
```

`readr::read_csv()` to import data frames as tibbles

readr::read_csv()

`readr::read_csv(file)`

No header? `col_names = FALSE`

```
new_starwars <- readr::read_csv("starwars.csv", col_names = FALSE)
```

Add header? `col_names = c(col_1, col_2)`

```
new_starwars <- readr::read_csv("starwars.csv",
     col_names = c("name", "height", "mass", "hair_color", . . .)
```

`readr::read_csv()` to import data frames as tibbles

`readr::read_csv()`

```
readr::read_csv(file)
```

★ Skip lines? `skip = number_lines_to_skip`

★ Only read in 10 lines? `n_max = 10`

★ Missing values: `na = c("", "NA", "n/a")`

★ Trim white space: `trim_ws = TRUE`

" blue" vs. "blue"

`readr::read_csv()` to import data frames as tibbles

readr::read_csv()

`readr::read_csv(file)`

Parse columns with specific types

```
new_starwars <- readr::read_csv("starwars.csv",
        col_types = cols(
            name = col_character(),
            height = col_integer()
        )
)


    str(new_starwars)
```

`readr::read_csv()` to import data frames as tibbles

readr::read_csv()

```
readr::read_csv(file)
```

★ Parse columns with specific types

```
Classes ‘spec_tbl_df’, ‘tbl_df’, ‘tbl’ and 'data.frame':	87 obs. of  10 variables:
$ name      : chr  "Luke Skywalker" "C-3PO" "R2-D2" "Darth Vader" ...
$ height    : int  172 167 96 202 150 178 165 97 183 182 ...
$ mass      : num  77 75 32 136 49 120 75 32 84 77 ...
$ hair_color: chr  "blond" NA NA "none" ...
$ skin_color: chr  "fair" "gold" "white, blue" "white" ...
$ eye_color : chr  "blue" "yellow" "red" "yellow" ...
$ birth_year: num  19 112 33 41.9 19 52 47 NA 24 57 ...
$ gender    : chr  "male" NA NA "male" ...
$ homeworld : chr  "Tatooine" "Tatooine" "Naboo" "Tatooine" ...
$ species   : chr  "Human" "Droid" "Droid" "Human" ...
- attr(*, "spec")=
```

```
str(new_starwars)
```

`readr::read_csv()` to import data frames as tibbles

readr::read_csv()

`readr::read_csv(file)`

★ Parse columns with specific types

```
new_starwars <- readr::read_csv("starwars.csv",
      col_types = "ci????????""
)
```

Character **Integer** **Best guess**

`readr::read_csv()` to import data frames as tibbles

`readr::read_csv()`

`readr::read_csv(file)`

Parse columns with specific types

- `col_logical()` [l], containing only `T`, `F`, `TRUE` or `FALSE`.
- `col_integer()` [i], integers.
- `col_double()` [d], doubles.
- `col_character()` [c], everything else.
- `col_factor(levels, ordered)` [f], a fixed set of values.
- `col_date(format = "")` [D]: with the locale's `date_format`.
- `col_time(format = "")` [t]: with the locale's `time_format`.
- `col_datetime(format = "")` [T]: ISO8601 date times
- `col_number()` [n], numbers containing the `grouping_mark`
- `col_skip()` [_, -], don't import this column.
- `col_guess()` [?], parse using the "best" type based on the input.

`readr::read_csv()` to import data frames as tibbles

Introduction: **stringr**

- Collection of functions to work with character strings

Sections:

- Detect matches
- Subset strings
- Manage lengths
- Mutate strings
- Join and split
- Order strings
- ***Regular expressions***

stringr - detect matches

```
stringr::str_detect(string, pattern)
```

Keep all hair colors that mention brown

```
starwars %>%
  dplyr::filter(stringr::str_detect(hair_color, “brown”))
```

```
stringr::str_count(string, pattern)
```

Make a new column counting how many vowels are in each character’s name

```
starwars %>%
  dplyr::mutate(vowels = stringr::str_count(name, “a|e|i|o|u”))
```

stringr - mutate strings

`stringr::str_to_lower(string)`

Make a new column counting how many vowels are in each character's name

hint: change all names to lower case first

```
starwars %>%
  dplyr::mutate(name = stringr::str_to_lower(name)) %>%
  dplyr::mutate(vowels = stringr::str_count(name, "a|e|i|o|u"))
```

`stringr::str_to_upper(string)`

Change string to all uppercase

`stringr::str_to_title(string)`

Change string to "title" case (First Letter Upper)

stringr - mutate strings

```
stringr::str_replace(string, pattern, replacement)
```

Replace all mentions of "Human" with "Homo sapien"

```
species <- starwars %>%
  dplyr::mutate(species =
          stringr::str_replace(species, "Human", "Homo sapiens"))
```

stringr - join and split

```
stringr::str_c(string1, string2)
```

Give each starwars character a PhD

hint: add ", PhD" to the end of the name column

```
doctorates <- starwars %>%
  dplyr::mutate(name = stringr::str_c(name, ", PhD"))
```

```
stringr::str_split_fixed(string, pattern, n)
```

Split "name" into "first name" and "last name"

hint: try str_split_fixed("Luke Skywalker", " ", 2) first to see output

```
names <- starwars %>%
  dplyr::mutate(first_name =
          stringr::str_split_fixed(name, " ", 2)[,1],
              last_name =
          stringr::str_split_fixed(name, " ", 2)[,2])
```

stringr - manage lengths

```
stringr::str_length(string)
```

Add a new column that counts the number of letters in each name

```
counts <- starwars %>%
  dplyr::mutate(name_counts = stringr::str_length(name))
```

```
stringr::str_trim(string, side = c("left", "right", "both")
```

Remove whitespace surrounding " test "

```
stringr::str_trim(" test ")
```

stringr - regular expressions

Regular Expressions - Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS

see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

string (type this)	**regexp** (to mean this)	**matches** (which matches this)	**example**	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	\.	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	\!	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	\?	?	see("\\?")	abc ABC 123 .!?\(){}
\\\\	\\	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	\(	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	\)	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	\{	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	\}	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	\n	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	\t	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	\s	any whitespace *(\S for non-whitespaces)*	see("\\s")	abc ABC 123 .!?\(){}
\\d	\d	any digit *(\D for non-digits)*	see("\\d")	abc ABC 123 .!?\(){}
\\w	\w	any word character *(\W for non-word chars)*	see("\\w")	abc ABC 123 .!?\(){}
\\b	\b	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:] [1]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:] [1]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:] [1]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:] [1]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:] [1]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:] [1]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:] [1]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:] [1]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:] [1]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

[1] Many base R functions require classes to be wrapped in a second set of [ ], e.g. **[[:digit:]]**

stringr

Introduction: **lubridate**

- Collection of functions to work with dates/times

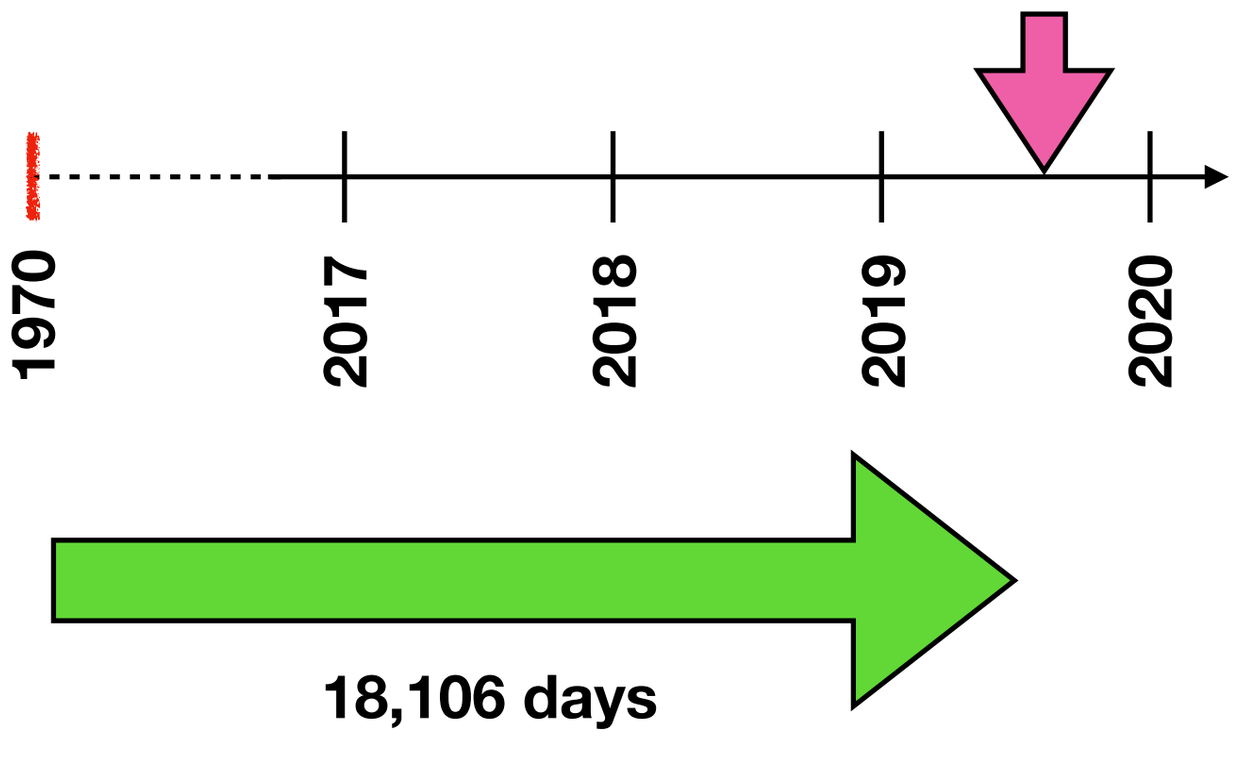
Sections:

- Parsing datetimes
- Getting and setting dates/times
- Periods
- Durations
- Intervals

lubridate - parsing date-times

Date-times

"2019-07-29 22:05:19 CDT"




```
lubridate::as_date(18107)

lubridate::now()
```

lubridate - parsing date-times

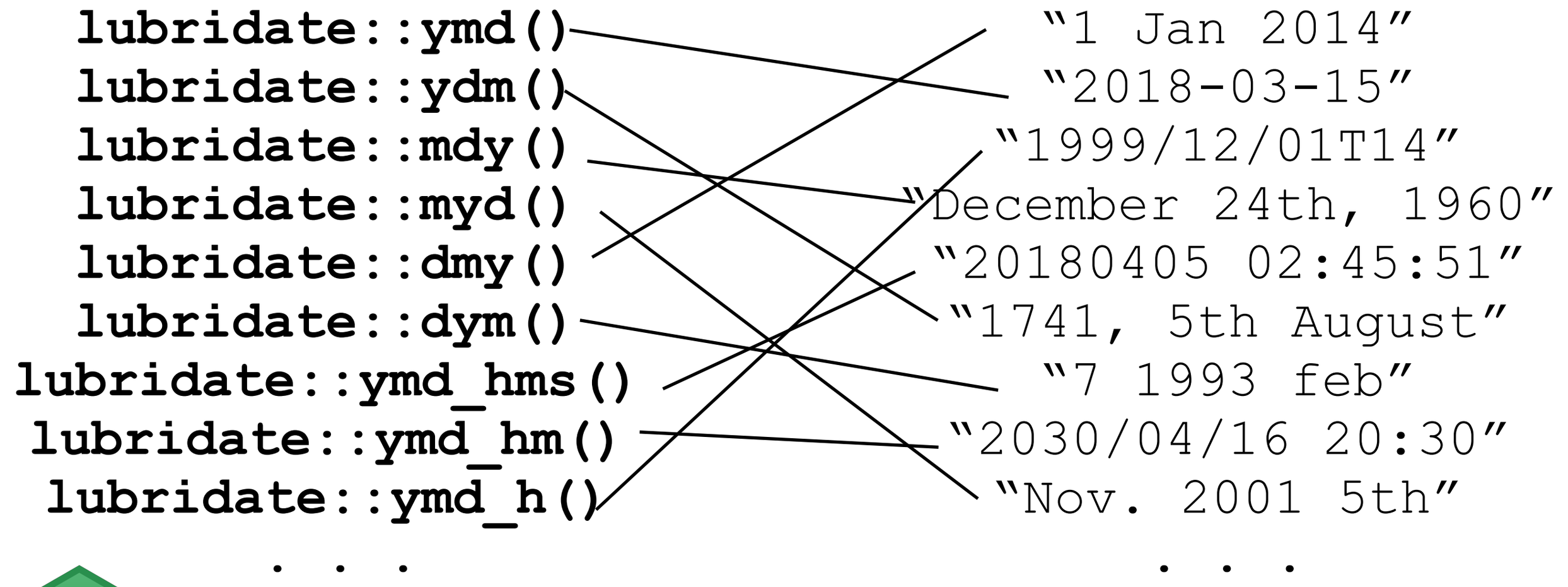
Match the code (left) that should be used to parse the dates (right)

`lubridate::ymd()`	"1 Jan 2014"
`lubridate::ydm()`	"2018-03-15"
`lubridate::mdy()`	"1999/12/01T14"
`lubridate::myd()`	"December 24th, 1960"
`lubridate::dmy()`	"20180405 02:45:51"
`lubridate::dym()`	"1741, 5th August"
`lubridate::ymd_hms()`	"7 1993 feb"
`lubridate::ymd_hm()`	"2030/04/16 20:30"
`lubridate::ymd_h()`	"Nov. 2001 5th"
. . .	. . .

Convert strings and numbers to date-times

lubridate - parsing date-times

Match the code (left) that should be used to parse the dates (right)



Convert strings and numbers to date-times

lubridate - get and set times

2019-07-30 11:01:59

`lubridate::date()`	"2019-07-30"
`lubridate::year()`	2019
`lubridate::month()`	7
`lubridate::day()`	30
`lubridate::wday()`	3
`lubridate::hour()`	11
`lubridate::minute()`	1
`lubridate::second()`	59
`lubridate::week()`	31
`lubridate::am()`	TRUE
`lubridate::dst()`	FALSE
`lubridate::leap_year()`	FALSE

Convert strings and numbers to date-times

lubridate - math with date-time

How much time until Christmas??

```
lubridate::ymd("2019-12-25 00:00:01")-lubridate::now()
```

What day will it be in 30 days?

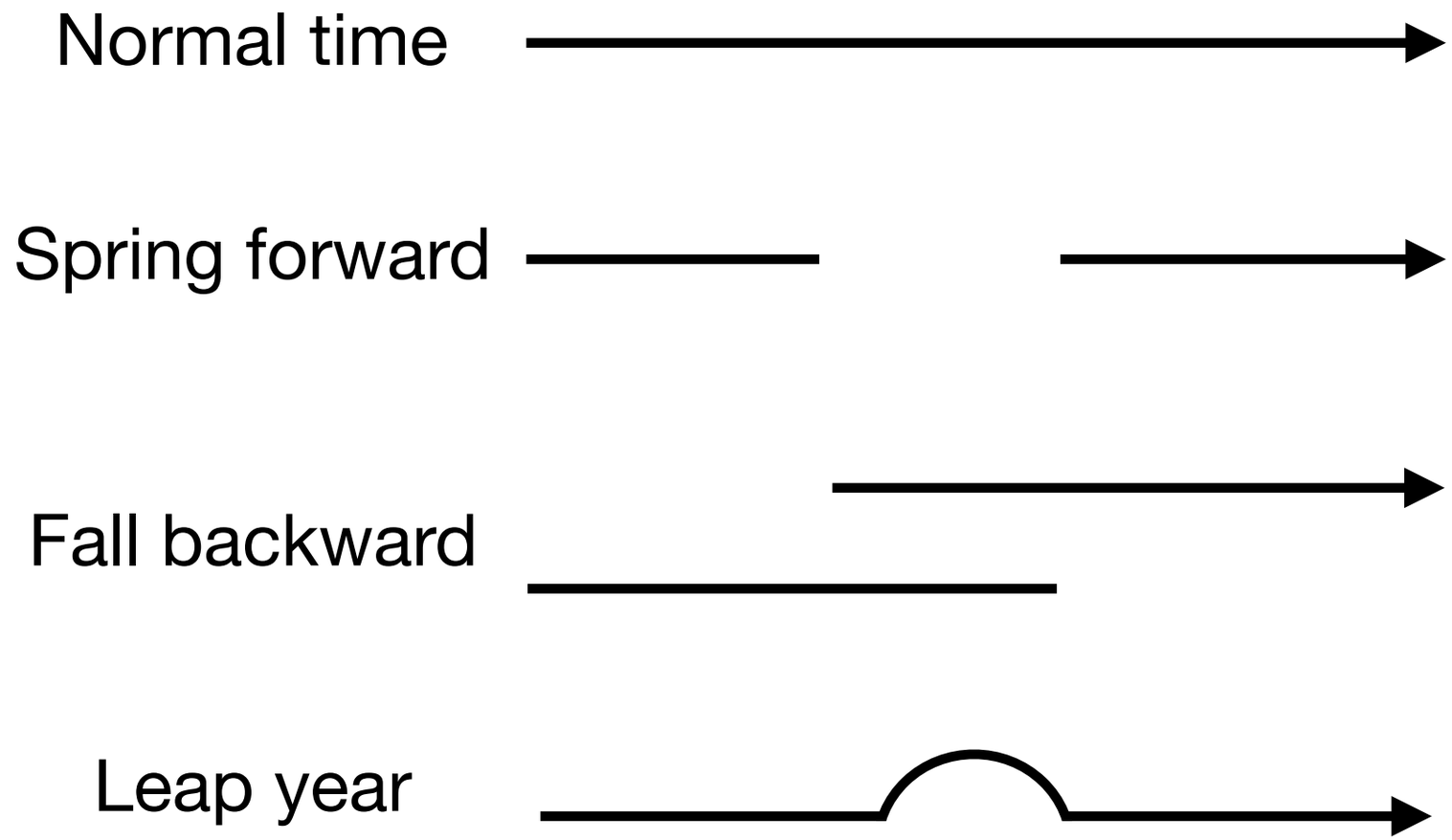
```
lubridate::now() + lubridate::days(30)
```

What time is it in London?

```
lubridate::with_tz(now(), tzone = "GMT")
```

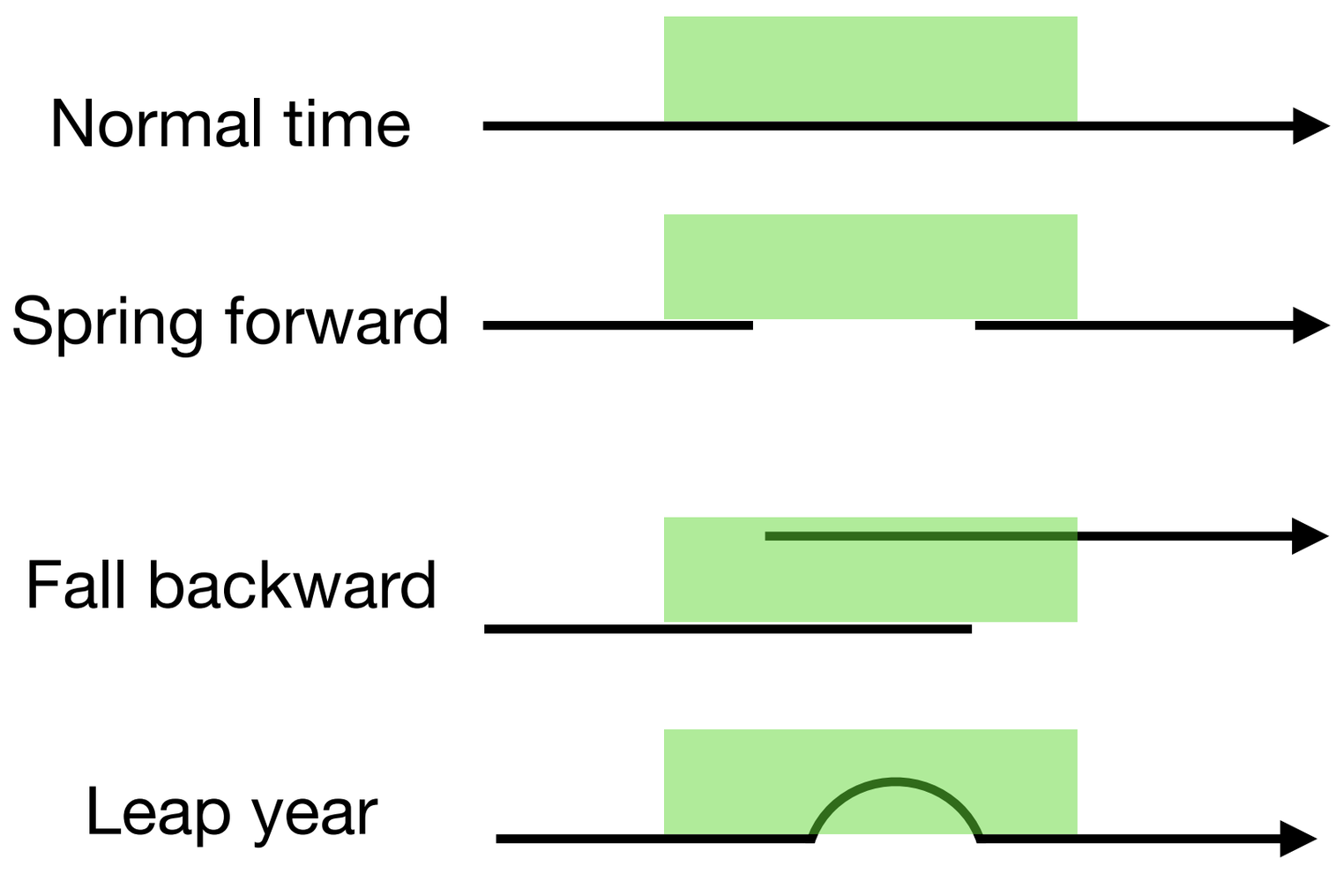
Intervals: represent specific intervals of the timeline

lubridate - math with date-time



Convert strings and numbers to date-times

lubridate - periods



Make a period with the name of the time unit ***pluralized***

```
lubridate::years()
lubridate::months()
lubridate::weeks()
lubridate::days()
lubridate::hours()
lubridate::minutes()
. . .
```

Periods: track changes in clock times, ignore irregularities

lubridate - periods

Add 2 hours to “2019-07-29 11:01:59”

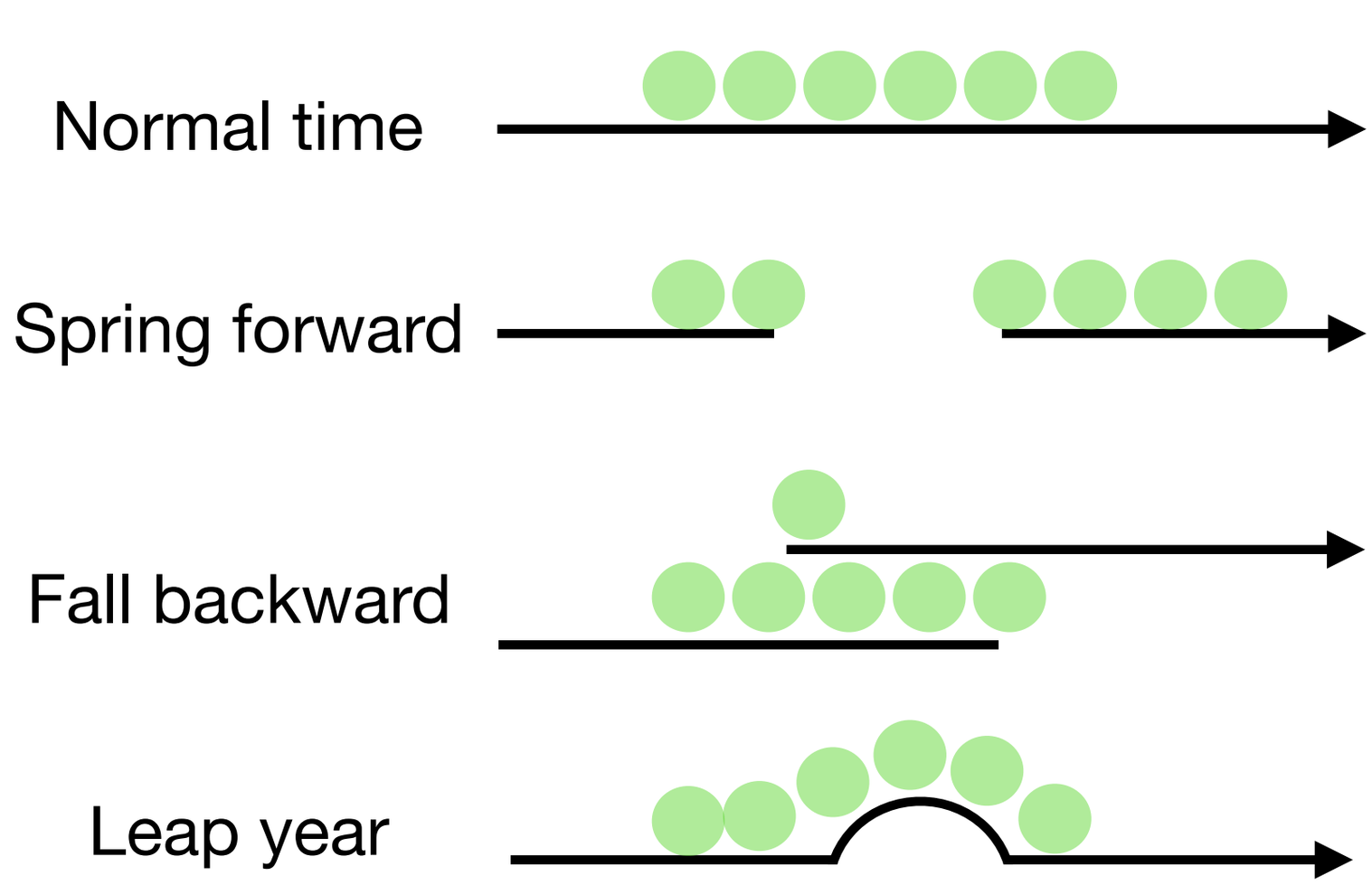
```
lubridate::ymd_hms(“2019-07-29 11:01:59”) +
        lubridate::hours(2)
```

Make a period with the name of the time unit ***pluralized***

```
lubridate::years()
lubridate::months()
lubridate::weeks()
lubridate::days()
lubridate::hours()
lubridate::minutes()
      . . .
```

Periods: track changes in clock times, ignore irregularities

lubridate - durations



Make a duration with the time period prefixed with a **d**

```
lubridate::dyears()
lubridate::dmonths()
lubridate::dweeks()
lubridate::ddays()
lubridate::dhours()
lubridate::dminutes()
      . . .
```

Durations: track the passage of physical time

lubridate - durations

Find the number of seconds in 30 days

```
lubridate::ddays(30)
```

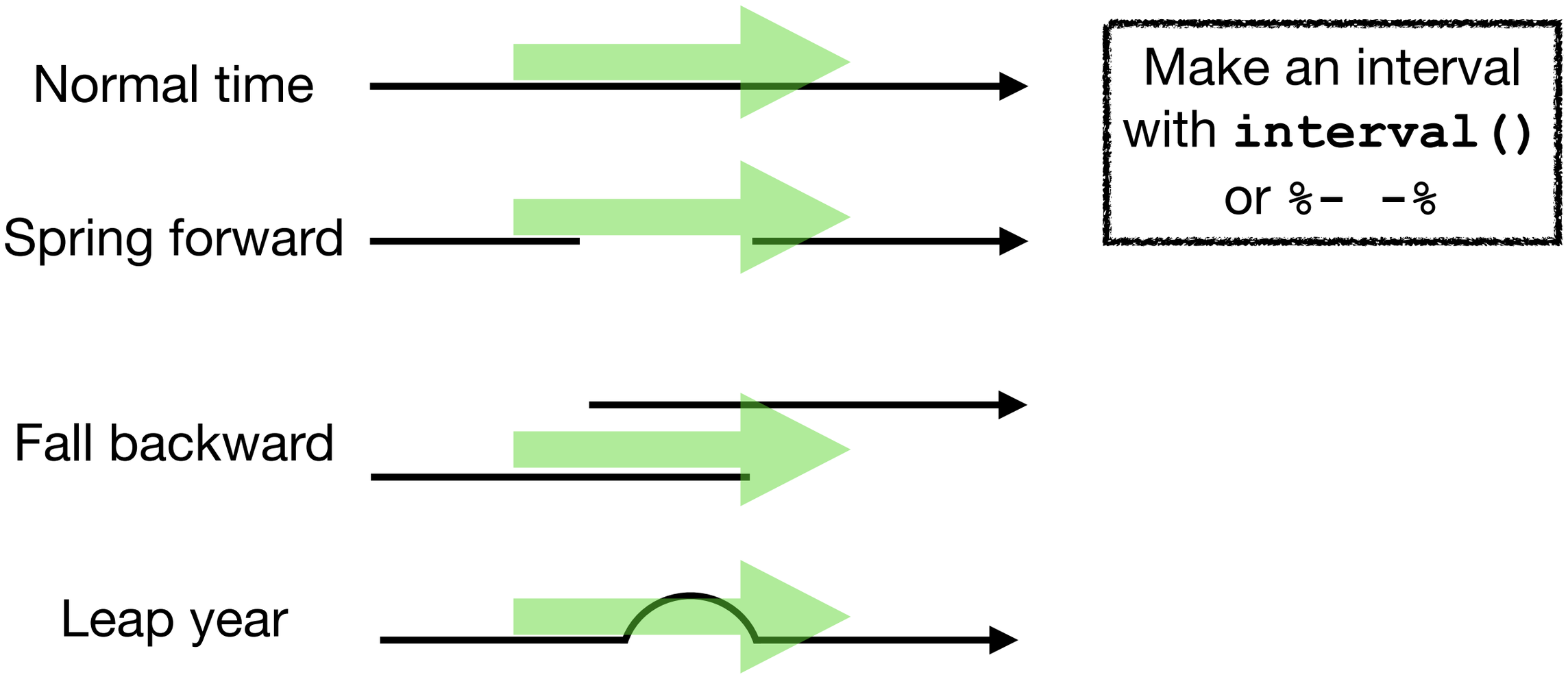
Durations are always stored as seconds

Make a duration with the time period prefixed with a **d**

```
lubridate::dyears()
lubridate::dmonths()
lubridate::dweeks()
lubridate::ddays()
lubridate::dhours()
lubridate::dminutes()
      . . .
```

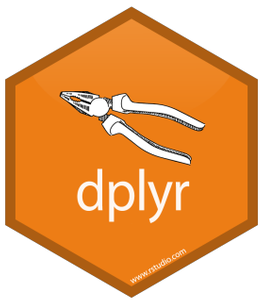
Durations: track the passage of physical time

lubridate - intervals

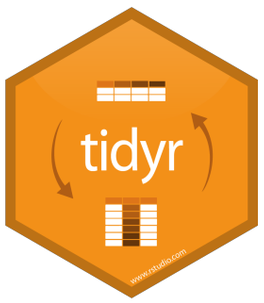


Intervals: represent specific intervals of the timeline

Overview of Tidyverse



The **dplyr** package is the most useful package in R for data manipulation. One of the greatest advantages of the package is that you can use the pipe function (%>%) to combine different functions.

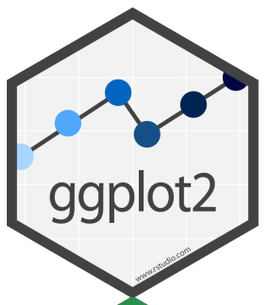


The **tidyr** package complements dplyr perfectly. It boosts the power of dplyr for data manipulation and pre-processing

The **readr** package is used to import and export data as tibbles in R.

The **stringr** package is used for strings. It provides a cohesive set of functions designed to make working with strings as easy as possible.

Other Tidyverse packages



Data scientists universally love using **ggplot2** to produce their charts and visualizations!

The **lubridate** package is the best way to deal with dates and times in R! From converting strings to dates to calculating hours between two time points.

The **purrr** package in R provides a complete toolkit for enhancing R's functional programming. We can use the functions provide by purrr to avoid many loops with just one line of code.

The **forecats** package is dedicated to dealing with categorical variables or factors.

The **broom** package takes the messy output of built-in functions in R and turns them into tidy dataframes