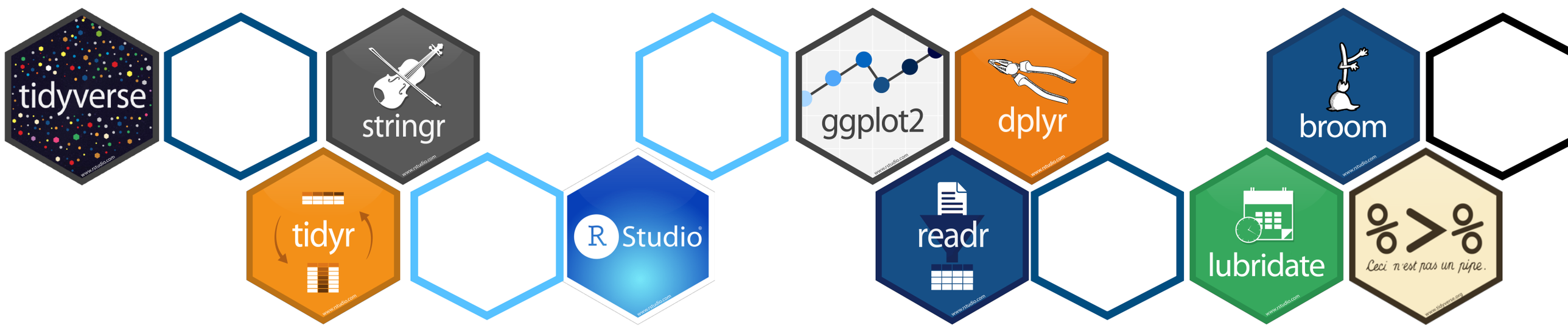


Data Manipulation with the Tidyverse

February 19-20 (1:30 PM - 4:30 PM)

Mudd Library, Large Classroom 2210

Katie Evans

NUIT Data Science Consultant

kathryn.evans@u.northwestern.edu

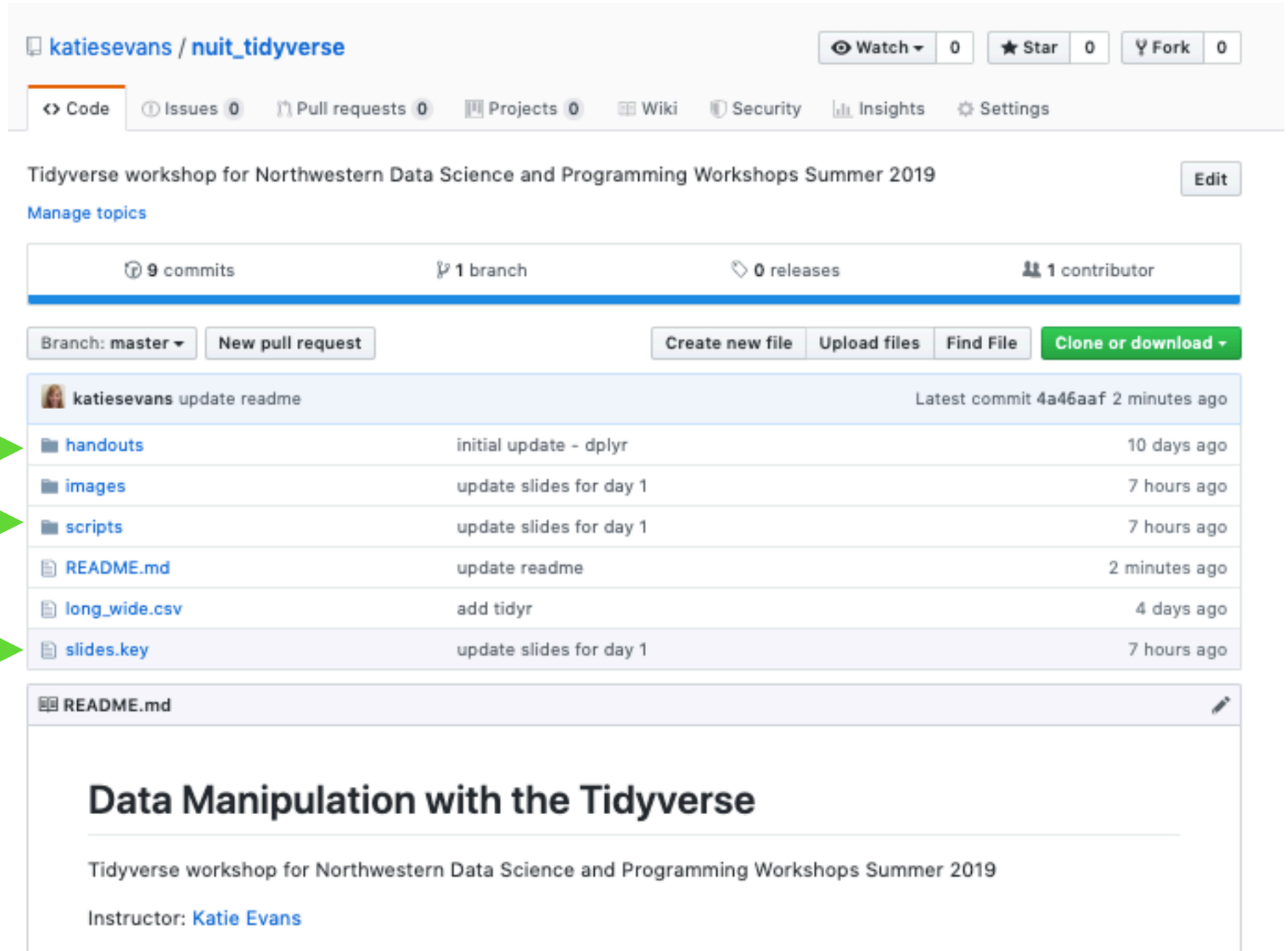
https://github.com/katiesevans/nuit_tidyverse

@katiesevans9

Day 1

intro, dplyr

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

color

merged cells

multiple column titles

	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	6.54205607476636
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

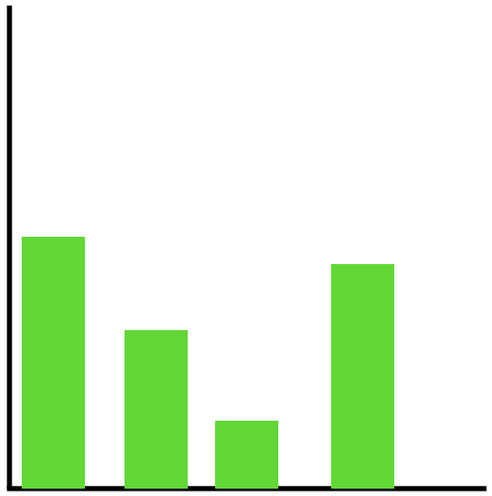
strain	survival	rep	day
BRC20067	0.873362445	A	1
DL238	64.21052631	A	1
A3	0	A	1
A6_R1	0	A	1
A6_R2	0	A	1
B2	1.069518716	A	1
B3	6.703910614	A	1
B6	11.71875	A	1
B8	0	A	1
B9	2.150537634	A	1
B10	1.063829787	A	1
C6	0	A	1
BRC20067	0	B	1
DL238	48.20359281	B	1
A3	0	B	1
A6_R1	0	B	1
A6_R2	2.34375	B	1
B2	0.606060606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

Rules for tidy data

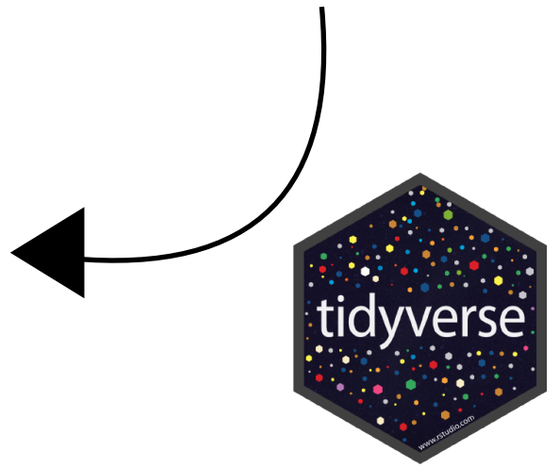
- Each **variable** must have its own **column**
- Each **observation** must have its own **row**
- Each **value** must have its own **cell**

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	6.54205607476636
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



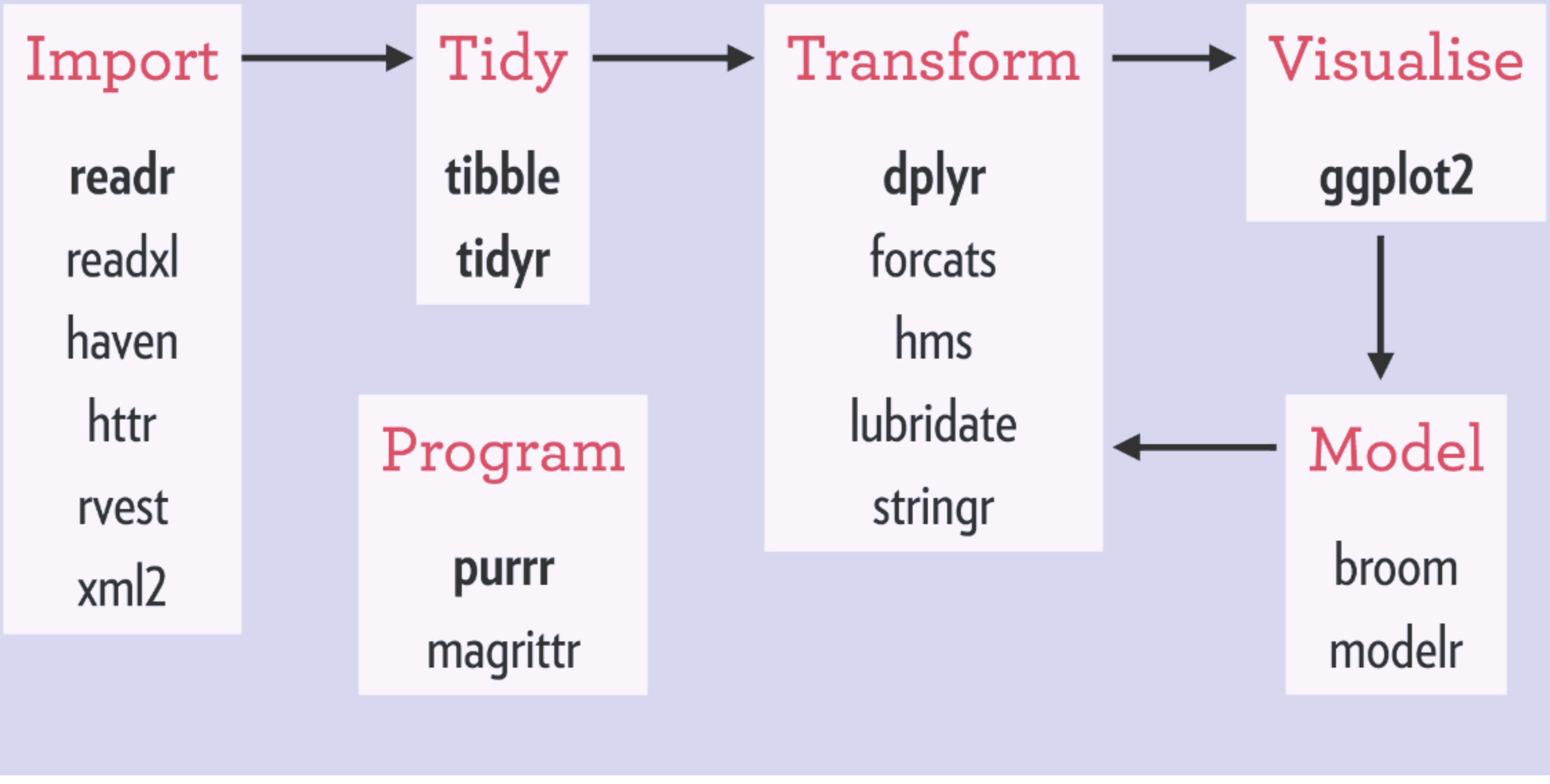
strain	survival	rep	day
BRC20067	0.873362445	A	1
DL238	64.210526315	A	1
A3	0	A	1
A6_R1	0	A	1
A6_R2	0	A	1
B2	1.069518716	A	1
B3	6.703910614	A	1
B6	11.71875	A	1
B8	0	A	1
B9	2.150537634	A	1



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

Starwars dataset


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

MESSY DATA

variables

name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
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")
C-3PO	167	75.0	*NA*	gold	yellow	112.0	*NA*	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Re...	character(0)	character(0)
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)
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
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)
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)
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)
R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)
Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
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...
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...
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)
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")
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")
Greedo	173	74.0	*NA*	green	black	44.0	male	Rodia	Rodian	A New Hope	character(0)	character(0)
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)
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
Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male	Bestine IV	Human	A New Hope	character(0)	X-wing
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)
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)
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
IG-88	200	140.0	none	metal	red	15.0	none	*NA*	Droid	The Empire Strikes Back	character(0)	character(0)
Bossk	190	113.0	none	green	red	53.0	male	Trandosha	Trandoshan	The Empire Strikes Back	character(0)	character(0)
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

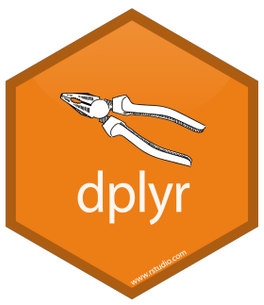
values

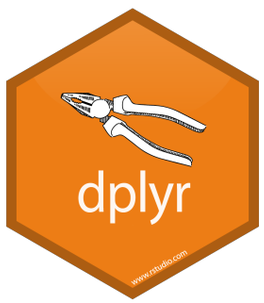
Introduction: **dplyr**

- 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





`dplyr::filter()`

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

dplyr::filter()

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

starwars

only humans

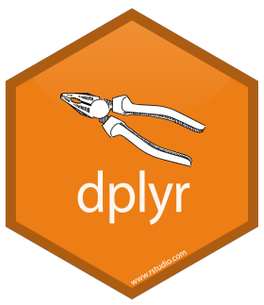
1. Figure out what you want to do

2. Describe your goal in words

Filter the starwars dataframe to only include humans

3. Describe your goal in code

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

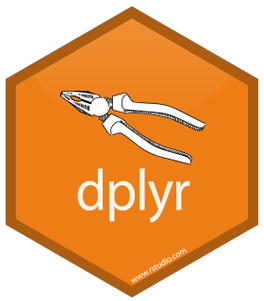


`dplyr::filter()` to keep rows based on values

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

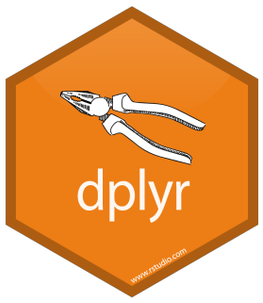
1. Figure out what you want to do

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")
```

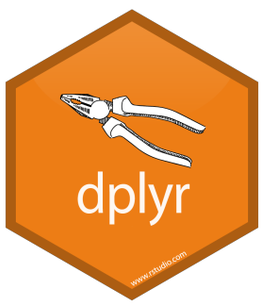


`dplyr::filter()` to keep rows based on values

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))`

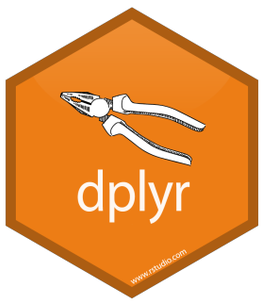
★ You can also filter based on multiple conditions

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.

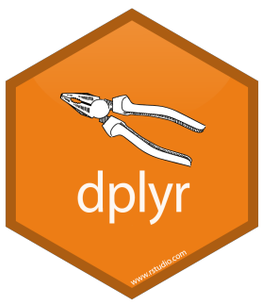
```
dplyr::filter(starwars, height < 100)
```

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

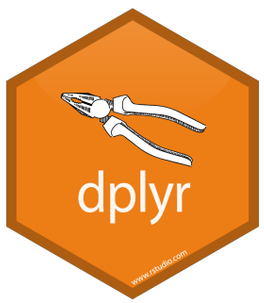
3. filter to keep only males

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

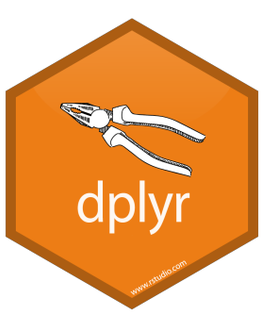
Select only name, species, and films variables from the starwars dataframe

```
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`

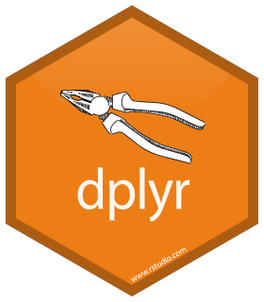
Select columns name, height, mass, hair color, and skin color

```
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)`

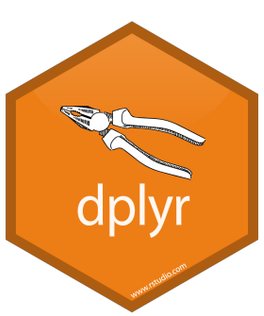
★ Reorder columns with select too!

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

dplyr::select()

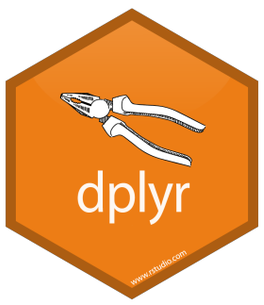
```
dplyr::select(dataframe,  new_name = old_name)
```

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

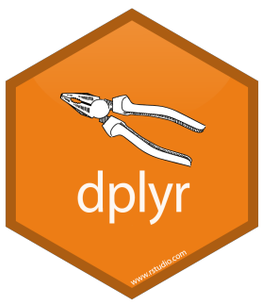
Combine filter and select

`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]



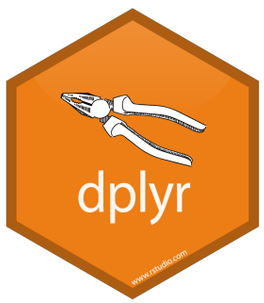
`dplyr::filter()` to keep rows based on values

`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::rename()`

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

dplyr::**rename**()

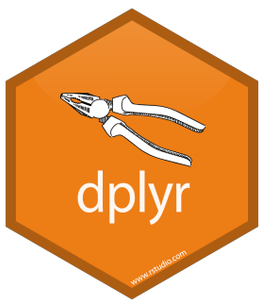
`dplyr::rename(`dataframe, new_name = old_name`)`

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)`

compare `select()` and `rename()` for renaming columns

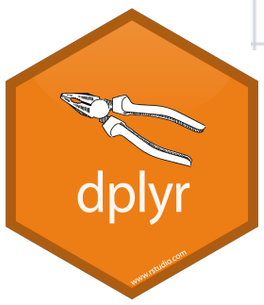
Rename the "name" column to "character" and the "mass" column to "weight" using `select()` and `rename()`

`dplyr::select()`

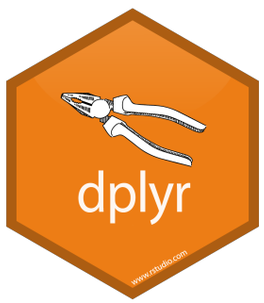
character	weight
Luke Skywalker	77.0
C-3PO	75.0
R2-D2	32.0
Darth Vader	136.0
Leia Organa	49.0
Owen Lars	120.0

`dplyr::rename()`

character	height	weight	hair_color	skin_color	eye_color	birt
Luke Skywalker	172	77.0	blond	fair	blue	
C-3PO	167	75.0	*NA*	gold	yellow	
R2-D2	96	32.0	*NA*	white, blue	red	
Darth Vader	202	136.0	none	white	yellow	
Leia Organa	150	49.0	brown	light	brown	
Owen Lars	178	120.0	brown, grey	light	blue	



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



`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

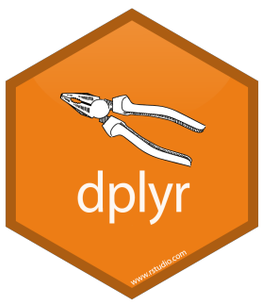
2. Describe your goal in words

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)`

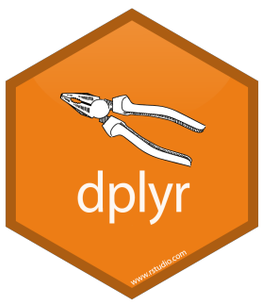
1. Figure out what you want to do

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

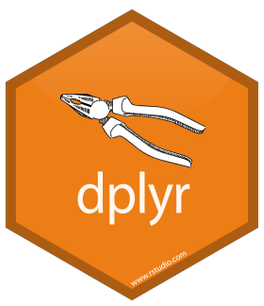
BMI = weight (kg) / [height (m)]^2

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

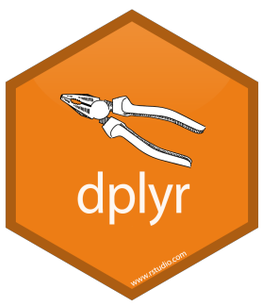
BMI = weight (kg) / [height (m)]^2

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**()

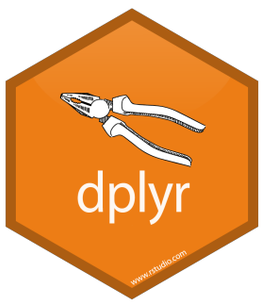
`dplyr::mutate(dataframe, new_column = expression)`

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**()

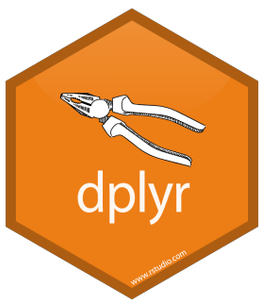
`dplyr::mutate(`dataframe, new_column = expression`)`

★ 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**()

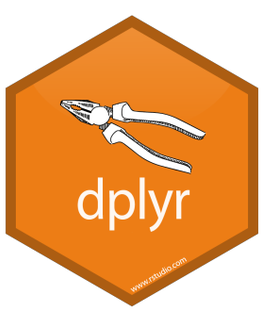
`dplyr::mutate(dataframe, new_column = expression)`

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

```
starwars %>%
   dplyr::mutate(height = h
   dplyr::mutate(avg_height
```

```
?mean
```

name	height	mass	avg_height
Palpatine	1.70	75.0	*NA*
Boba Fett	1.83	78.2	*NA*
IG-88	2.00	140.0	*NA*
Bossk	1.90	113.0	*NA*
Lando Calrissian	1.77	79.0	*NA*
Lobot	1.75	79.0	*NA*
Ackbar	1.80	83.0	*NA*
Mon Mothma	1.50	*NA*	*NA*
Arvel Crynyd	*NA*	*NA*	*NA*
Wicket Systri Warrick	0.88	20.0	*NA*
Nien Nunb	1.60	68.0	*NA*
Qui-Gon Jinn	1.93	89.0	*NA*
Nute Gunray	1.91	90.0	*NA*
Finis Valorum	1.70	*NA*	*NA*



`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
```

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

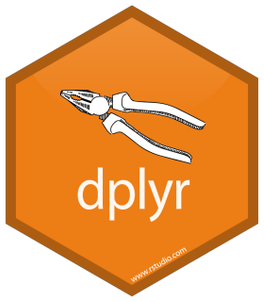
Usage

```
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

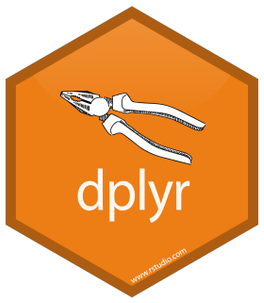
dplyr::**mutate**()

`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

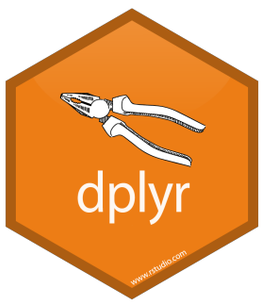
dplyr::**mutate**()

`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**()

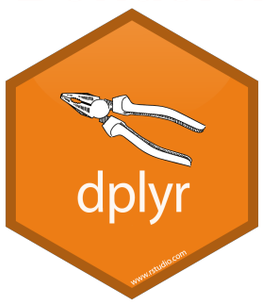
`dplyr::mutate(`dataframe, new_column = expression`)`

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**()

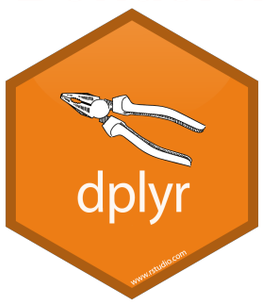
`dplyr::mutate(`dataframe, new_column = expression`)`

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**()

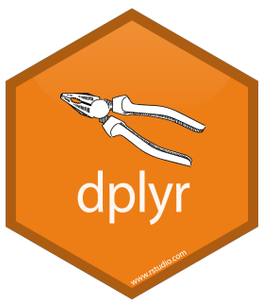
`dplyr::mutate(`dataframe, new_column = expression`)`

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

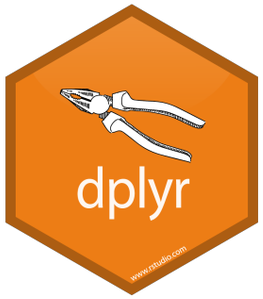
hint: try using `ifelse(condition, if_true_do_this, if_false_do_this)`

```
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()`

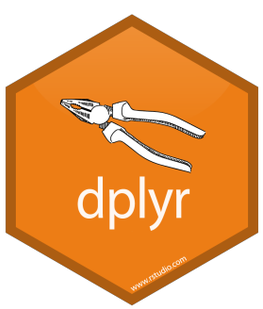
`dplyr::group_by(dataframe, column_to_group_by)`

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()

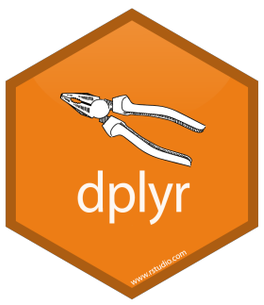
`dplyr::group_by(dataframe, column_to_group_by)`

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

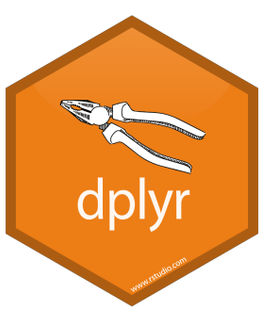
Group starwars dataframe by gender

```
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()

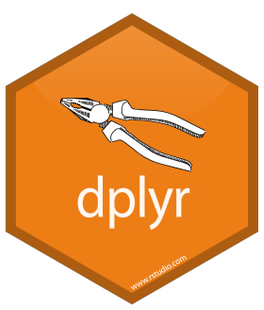
```
dplyr::group_by(dataframe, column_to_group_by)
```

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	gender	height	avg_height
Luke Skywalker	male	1.72	1.74358
C-3PO	*NA*	1.67	1.74358
R2-D2	*NA*	0.96	1.74358
Darth Vader	male	2.02	1.74358
Leia Organa	female	1.50	1.74358
Owen Lars	male	1.78	1.74358
Beru Whitesun lars	female	1.65	1.74358
R5-D4	*NA*	0.97	1.74358
Biggs Darklighter	male	1.83	1.74358
Obi-Wan Kenobi	male	1.82	1.74358
Anakin Skywalker	male	1.88	1.74358
Wilhuff Tarkin	male	1.80	1.74358
Chewbacca	male	2.28	1.74358
Han Solo	male	1.80	1.74358



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

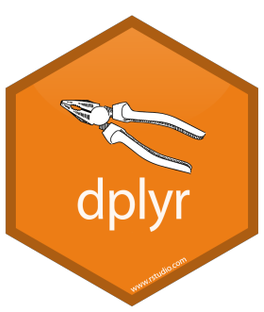
dplyr::group_by()

`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

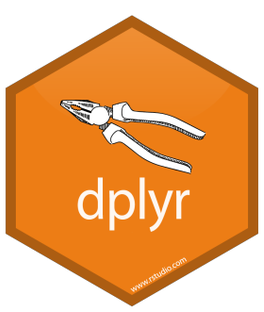
dplyr::group_by()

`dplyr::group_by(dataframe, column_to_group_by)`

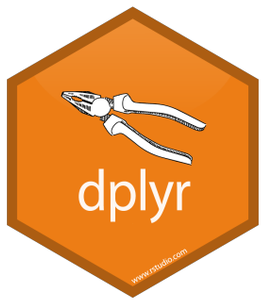
★ You can also group by multiple columns

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)`

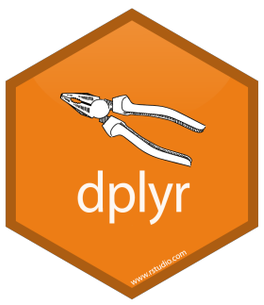
`summarize()` is similar to `mutate()` but only keeps grouped columns

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

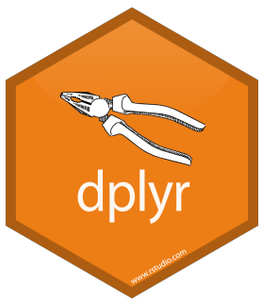
Compare `mutate()` and `summarize()` to calculate average height by gender

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

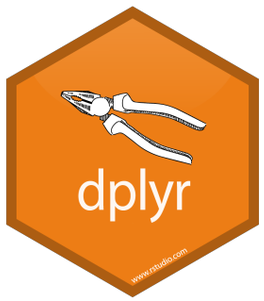
name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships	avg_height
Luke Skywalker	172	77.0	blond	fair	blue	19.0	male	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial 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

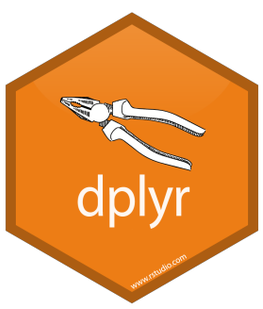
```
arranged_df <- dplyr::arrange(starwars, homeworld)
```

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

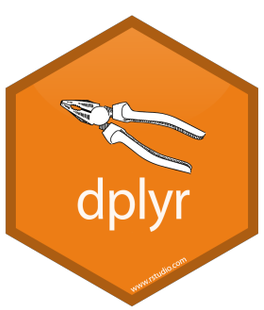
dplyr::**arrange**()

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

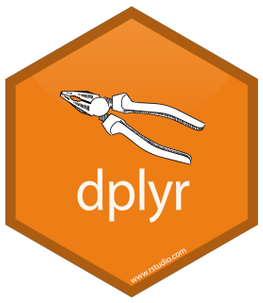
Arrange starwars by species first then height

```
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()

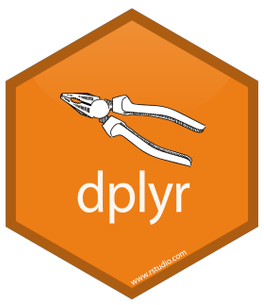
full_join()

inner_join()

semi_join()

anti_join()

nest_join()



`dplyr::xxx_join()` to combine datasets

dplyr::xxx_join()

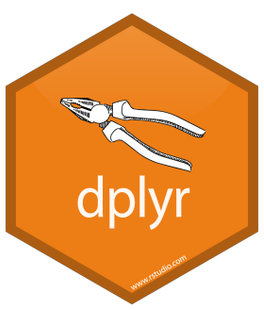
`dplyr::xxx_join(dataframe1, dataframe2, by = column_in_common)`

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

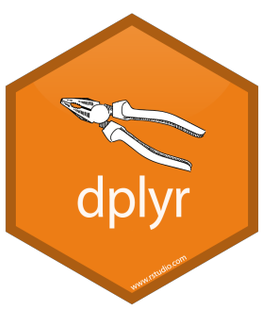
dplyr::left_join()

```
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

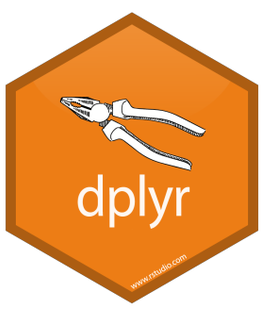
dplyr::left_join()

```
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

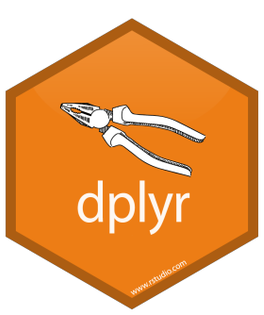
dplyr::right_join()

```
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)
```



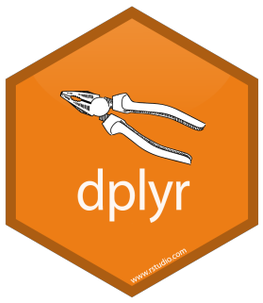
`dplyr::right_join()` to combine datasets

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)
```

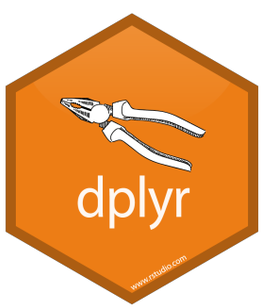
`dplyr::right_join()` to combine datasets

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)
```

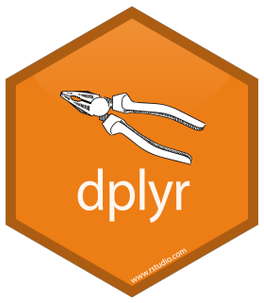
`dplyr::full_join()` to combine datasets

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()

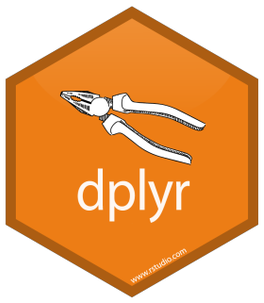
```
dplyr::full_join(dataframe1, dataframe2, by = column_in_common)
```

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::full_join()

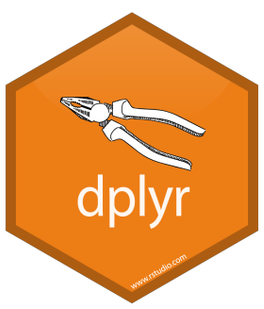
`dplyr::full_join(dataframe1, dataframe2, by = column_in_common)`

df1

name	height	mass
Luke Skywalker	172	77.0
C-3PO	167	75.0
R2-D2	96	32.0
Darth Vader	202	136.0
Leia Organa	150	49.0
Owen Lars	178	120.0
Beru Whitesun lars	165	75.0
R5-D4	97	32.0
Biggs Darklighter	183	84.0
Obi-Wan Kenobi	182	77.0
Anakin Skywalker	188	84.0
Wilhuff Tarkin	180	*NA*
Chewbacca	228	112.0
Han Solo	180	80.0

df2

name	eye_color	hair_color	gender
Ackbar	orange	none	male
Adi Gallia	blue	none	female
Anakin Skywalker	blue	blond	male
Arvel Crynyd	brown	brown	male
Ayla Secura	hazel	none	female
Bail Prestor Organa	brown	black	male
Barriss Offee	blue	black	female
BB8	black	none	none
Ben Quadinaros	orange	none	male
Beru Whitesun lars	blue	brown	female
Bib Fortuna	pink	none	male
Biggs Darklighter	brown	black	male
Boba Fett	brown	black	male
Bossk	red	none	male



`dplyr::full_join()` to combine datasets

dplyr::full_join()

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

df1

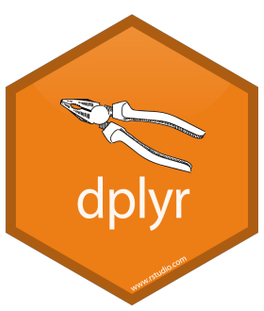
name	height	mass
Luke Skywalker	172	77.0
Leia Organa	150	49.0
Owen Lars	178	120.0
Beru Whitesun lars	165	75.0
R5-D4	97	32.0
Chewbacca	228	112.0
Han Solo	180	80.0

df2

name	eye_color	hair_color	gender
Ackbar	orange	none	male
	blue	none	female
walker	blue	blond	male
yd	brown	brown	male
Ayla Secura	hazel	none	female
Bail Prestor Organa	brown	black	male
Barriss Offee	blue	black	female
BB8	black	none	none
Boba Fett	brown	black	male
Bossk	red	none	male

```
df1 <- starwars %>%
    dplyr::select(name, height, mass)
```

```
df2 <- starwars %>%
    dplyr::select(name, eye_color, hair_color, gender) %>%
    dplyr::arrange(name)
```



`dplyr::full_join()` to combine datasets

dplyr::full_join()

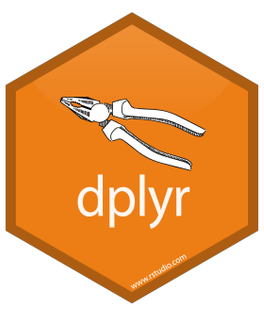
`dplyr::full_join(dataframe1, dataframe2, by = column_in_common)`

df1

name	height	mass
Luke Skywalker	172	77.0
C-3PO	167	75.0
R2-D2	96	32.0
Darth Vader	202	136.0
Leia Organa	150	49.0
Owen Lars	178	120.0
Beru Whitesun lars	165	75.0
R5-D4	97	32.0
Biggs Darklighter	183	84.0
Obi-Wan Kenobi	182	77.0
Anakin Skywalker	188	84.0
Wilhuff Tarkin	180	*NA*
Chewbacca	228	112.0
Han Solo	180	80.0

df2

name	eye_color	hair_color	gender
Ackbar	orange	none	male
Adi Gallia	blue	none	female
Anakin Skywalker	blue	blond	male
Arvel Crynyd	brown	brown	male
Ayla Secura	hazel	none	female
Bail Prestor Organa	brown	black	male
Barriss Offee	blue	black	female
BB8	black	none	none
Ben Quadinaros	orange	none	male
Beru Whitesun lars	blue	brown	female
Bib Fortuna	pink	none	male
Biggs Darklighter	brown	black	male
Boba Fett	brown	black	male
Bossk	red	none	male



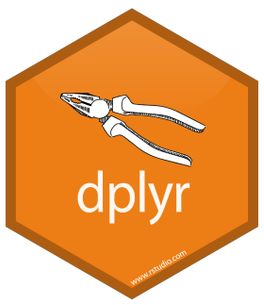
`dplyr::full_join()` to combine datasets

dplyr::full_join()

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

name	height	mass	eye_color	hair_color	gender
Luke Skywalker	172	77.0	blue	blond	male
C-3PO	167	75.0	yellow	*NA*	*NA*
R2-D2	96	32.0	red	*NA*	*NA*
Darth Vader	202	136.0	yellow	none	male
Leia Organa	150	49.0	brown	brown	female
Owen Lars	178	120.0	blue	brown, grey	male
Beru Whitesun lars	165	75.0	blue	brown	female
R5-D4	97	32.0	red	*NA*	*NA*
Biggs Darklighter	183	84.0	brown	black	male
Obi-Wan Kenobi	182	77.0	blue-gray	auburn, white	male
Anakin Skywalker	188	84.0	blue	blond	male
Wilhuff Tarkin	180	*NA*	blue	auburn, grey	male
Chewbacca	228	112.0	blue	brown	male
Han Solo	180	80.0	brown	brown	male

```
joined_df <- dplyr::full_join(df1, df2, by = "name")
```



`dplyr::full_join()` to combine datasets

dplyr::left_join()

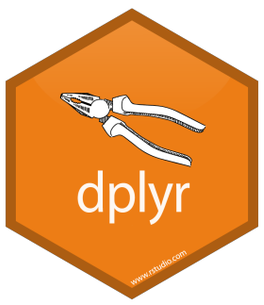
`dplyr::left_join(dataframe1, dataframe2, by = column_in_common)`

df3

name	gender
Luke Skywalker	male
C-3PO	*NA*
R2-D2	*NA*
Darth Vader	male
Leia Organa	female
Owen Lars	male
Beru Whitesun lars	female
R5-D4	*NA*
Biggs Darklighter	male
Obi-Wan Kenobi	male
Anakin Skywalker	male
Wilhuff Tarkin	male
Chewbacca	male
Han Solo	male

df4

gender	number
female	19
male	62



`dplyr::left_join()` to combine datasets

dplyr::left_join()

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

df3

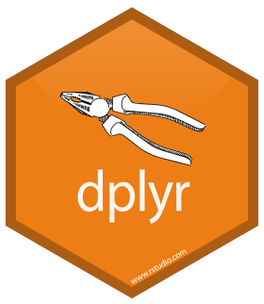
name	gender
Luke Skywalker	male
Leia Organa	female
Owen Lars	male
Beru Whitesun lars	female
Chewbacca	male
Han Solo	male

```
df3 <- starwars %>%
    dplyr::select(name, gender)
```

df4

gender	number
female	19
male	62

```
df4 <- starwars %>%
    dplyr::filter(gender %in% c("male", "female"))%>%
    dplyr::group_by(gender) %>%
    dplyr::summarize(number = n())
```



`dplyr::left_join()` to combine datasets

dplyr::left_join()

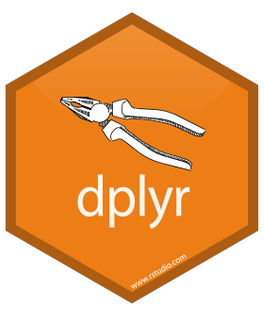
`dplyr::left_join(dataframe1, dataframe2, by = column_in_common)`

df3

name	gender
Luke Skywalker	male
C-3PO	*NA*
R2-D2	*NA*
Darth Vader	male
Leia Organa	female
Owen Lars	male
Beru Whitesun lars	female
R5-D4	*NA*
Biggs Darklighter	male
Obi-Wan Kenobi	male
Anakin Skywalker	male
Wilhuff Tarkin	male
Chewbacca	male
Han Solo	male

df4

gender	number
female	19
male	62



`dplyr::left_join()` to combine datasets

dplyr::left_join()

Which data frame is `left_joined()`?

LEFT

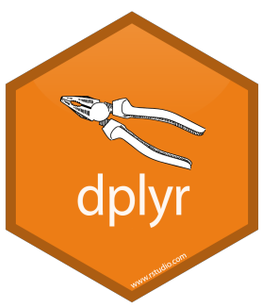
RIGHT

name	gender	number
Luke Skywalker	male	62
C-3PO	*NA*	*NA*
R2-D2	*NA*	*NA*
Darth Vader	male	62
Leia Organa	female	19
Owen Lars	male	62
Beru Whitesun lars	female	19
R5-D4	*NA*	*NA*
Biggs Darklighter	male	62
Obi-Wan Kenobi	male	62
Anakin Skywalker	male	62
Wilhuff Tarkin	male	62
Chewbacca	male	62
Han Solo	male	62
Greedo	male	62
Jabba Desilijic Tiure	hermaphrodite	*NA*
Wedge Antilles	male	62
Jek Tono Porkins	male	62
Yoda	male	62
Palpatine	male	62
Boba Fett	male	62
IG-88	none	*NA*
Bossk	male	62

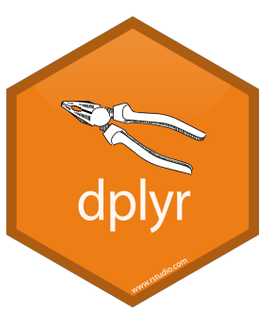
name	gender	number
Leia Organa	female	19
Beru Whitesun lars	female	19
Mon Mothma	female	19
Shmi Skywalker	female	19
Ayla Secura	female	19
Adi Gallia	female	19
Cordé	female	19
Luminara Unduli	female	19
Barriss Offee	female	19
Dormé	female	19
Zam Wesell	female	19
Taun We	female	19
Jocasta Nu	female	19
R4-P17	female	19
Shaak Ti	female	19
Sly Moore	female	19
Rey	female	19
Captain Phasma	female	19
Padmé Amidala	female	19
Luke Skywalker	male	62
Darth Vader	male	62
Owen Lars	male	62
Biggs Darklighter	male	62

LEFT

RIGHT



`dplyr::left_join()` to combine datasets

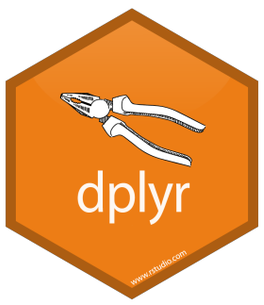


`dplyr::bind_rows()`
`dplyr::bind_cols()`

`dplyr::bind_xxx()` to combine datasets

dplyr::**bind_rows**()

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



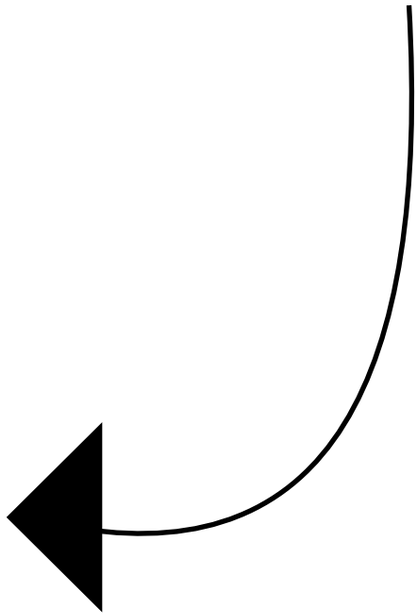
`dplyr::bind_rows()` to combine datasets

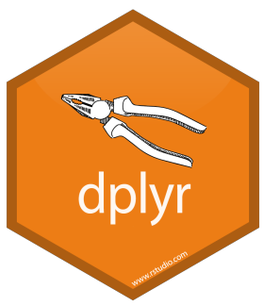
dplyr::bind_rows()

```
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

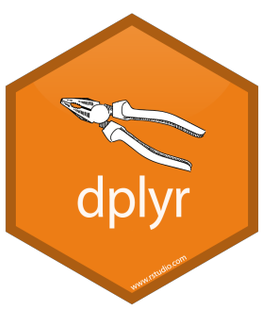
`dplyr::bind_rows()`

```
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_rows()

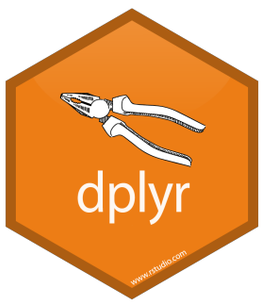
```
dplyr::bind_rows(dataframe1, dataframe2)
```

females

name	gender	species
Leia Organa	female	Human
Beru Whitesun lars	female	Human
Mon Mothma	female	Human
Shmi Skywalker	female	Human
Ayla Secura	female	Twi'lek
Adi Gallia	female	Tholothian
Cordé	female	Human
Luminara Unduli	female	Mirialan
Barriss Offee	female	Mirialan
Dormé	female	Human

males

name	gender	species
Luke Skywalker	male	Human
Darth Vader	male	Human
Owen Lars	male	Human
Biggs Darklighter	male	Human
Obi-Wan Kenobi	male	Human
Anakin Skywalker	male	Human
Wilhuff Tarkin	male	Human
Chewbacca	male	Wookiee
Han Solo	male	Human
Greedo	male	Rodian



`dplyr::bind_rows()` to combine datasets

dplyr::bind_rows()

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

females

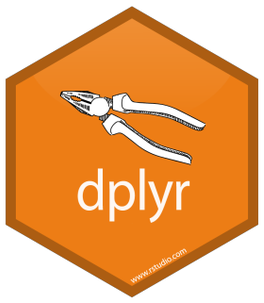
name	gender	species
Leia Organa	female	Human
Adi Gallia	female	Tholothian
Cordé	female	Human
Luminara Unduli	female	Mirialan
Barriss Offee	female	Mirialan
Dormé	fe	

```
females <- starwars %>%
    dplyr::select(name, gender, species) %>%
    dplyr::filter(gender == "female")
```

males

name	gender	species
Luke Skywalker	male	Human
	nale	Human
	nale	Human
	nale	Human
	nale	Human
Anakin Skywalker	male	Human
Wilhuff Tarkin	male	Human
Chewbacca	male	Wookiee
Han Solo	male	Human

```
males <- starwars %>%
    dplyr::select(name, gender, species) %>%
    dplyr::filter(gender == "male")
```



`dplyr::bind_rows()` to combine datasets

dplyr::bind_rows()

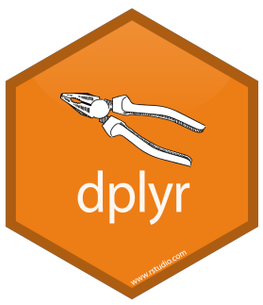
```
dplyr::bind_rows(dataframe1, dataframe2)
```

females

name	gender	species
Leia Organa	female	Human
Beru Whitesun lars	female	Human
Mon Mothma	female	Human
Shmi Skywalker	female	Human
Ayla Secura	female	Twi'lek
Adi Gallia	female	Tholothian
Cordé	female	Human
Luminara Unduli	female	Mirialan
Barriss Offee	female	Mirialan
Dormé	female	Human

males

name	gender	species
Luke Skywalker	male	Human
Darth Vader	male	Human
Owen Lars	male	Human
Biggs Darklighter	male	Human
Obi-Wan Kenobi	male	Human
Anakin Skywalker	male	Human
Wilhuff Tarkin	male	Human
Chewbacca	male	Wookiee
Han Solo	male	Human
Greedo	male	Rodian



`dplyr::bind_rows()` to combine datasets

dplyr::bind_rows()

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

name	gender	species
Leia Organa	female	Human
Beru Whitesun lars	female	Human
Mon Mothma	female	Human
Shmi Skywalker	female	Human
Ayla Secura	female	Twi'lek
Adi Gallia	female	Tholothian
Cordé	female	Human
Luminara Unduli	female	Mirialan
Barriss Offee	female	Mirialan
Dormé	female	Human
Luke Skywalker	male	Human
Darth Vader	male	Human
Owen Lars	male	Human
Biggs Darklighter	male	Human
Obi-Wan Kenobi	male	Human
Anakin Skywalker	male	Human

Benefit: can bind MANY data frames

```
bound <- females %>%
         dplyr::bind_rows(males)
```

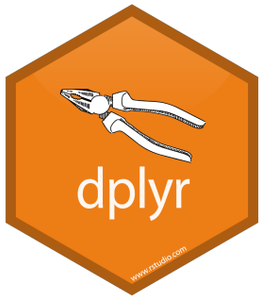
Or

```
bound <- dplyr::bind_rows(females, males)
```

Compare to:

```
bound <- rbind(females, males)
```

NOT `dplyr`



`dplyr::bind_rows()` to combine datasets

dplyr::bind_rows()

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

name	gender	species
Leia Organa	female	Human
Beru Whitesun lars	female	Human
Mon Mothma	female	Human
Shmi Skywalker	female	Human
Ayla Secura	female	Twi'lek
Adi Gallia	female	Tholothian
Cordé	female	Human
Luminara Unduli	female	Mirialan
Barriss Offee	female	Mirialan
Dormé	female	Human
Luke Skywalker	male	Human
Darth Vader	male	Human
Owen Lars	male	Human
Biggs Darklighter	male	Human
Obi-Wan Kenobi	male	Human
Anakin Skywalker	male	Human

Benefit: can bind MANY data frames
Benefit: works with pipes! (%>%)

```
bound <- females %>%
         dplyr::bind_rows(males)
```

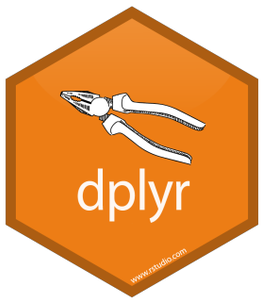
Or

```
bound <- dplyr::bind_rows(females, males)
```

Compare to:

```
bound <- rbind(females, males)
```

NOT `dplyr`



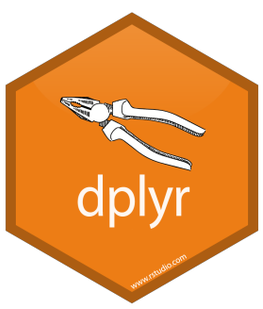
`dplyr::bind_rows()` to combine datasets

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



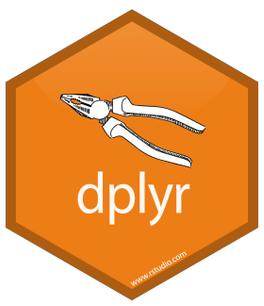
`dplyr::bind_cols()` to combine datasets

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)
```

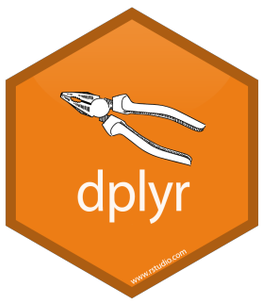
A	B	C	D
red	2	3	3
orange	4	6	5
yellow	8	9	7
green	0	0	1
indigo	3	3	3
blue	1	1	6
purple	5	5	NA
white	8	2	NA

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::bind_cols()

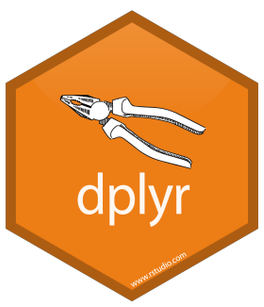
```
dplyr::bind_cols(dataframe1, dataframe2)
```

```
bound <- dplyr::bind_cols(df1, df2)
```

Compare to:

```
joined_df <- dplyr::full_join(df1, df2, by = "name")
```

name	height	mass	name1	eye_color	hair_color	gender
Luke Skywalker	172	77.0	Ackbar	orange	none	male
C-3PO	167	75.0	Adi Gallia	blue	none	female
R2-D2	96	32.0	Anakin Skywalker	blue	blond	male
Darth Vader	202	136.0	Arvel Crynyd	brown	brown	male
Leia Organa	150	49.0	Ayla Secura	hazel	none	female
Owen Lars	178	120.0	Bail Prestor Organa	brown	black	male
Beru Whitesun lars	165	75.0	Barriss Offee	blue	black	female
R5-D4	97	32.0	BB8	black	none	none



`dplyr::bind_cols()` to combine datasets

Data Transformation with dplyr : : CHEAT SHEET

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

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

pipes

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

Summarise Cases

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

summary function

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

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

Group Cases

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

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

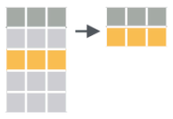
group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Returns ungrouped copy of table.
ungroup(g_iris)

Manipulate Cases

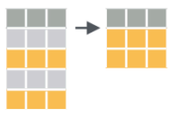
EXTRACT CASES

Row functions return a subset of rows as a new table.



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

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

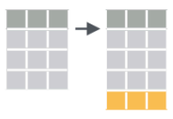
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

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

ARRANGE CASES

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

ADD CASES



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

Manipulate Variables

EXTRACT VARIABLES

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

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

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

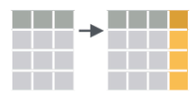
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

MAKE NEW VARIABLES

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

vectorized function

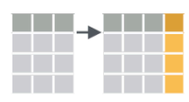


mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

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



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

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

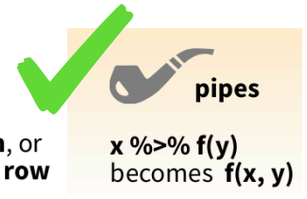
Data Transformation with dplyr : : CHEAT SHEET

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

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**



Summarise Cases

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

summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

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

Group Cases

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

mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))

group_by(.data, ..., add = FALSE**)**
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …**)**
Returns ungrouped copy of table.
ungroup(g_iris)

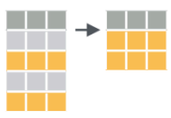
Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

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

distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*

slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

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

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

ARRANGE CASES

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

ADD CASES

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

Manipulate Variables

EXTRACT VARIABLES

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

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

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

MAKE NEW VARIABLES

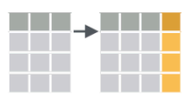
These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

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



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

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

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

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

MISC

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

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

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

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

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

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

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

Combine Tables

COMBINE VARIABLES

x + y =

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

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

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

x + y

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

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

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

x + y =

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

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

MISC

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

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

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

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

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

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

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

Combine Tables

COMBINE VARIABLES

x + y =

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.

full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

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

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

x
+ y
=

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, …)
Rows that appear in both x and y.

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

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

x + y =

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

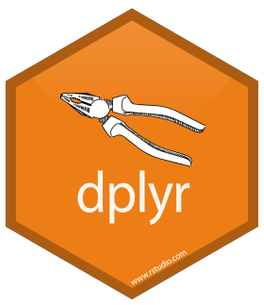
dplyr practice!

Calculate the birth year of each character given the column 'birth_year' is actually age in 2019…

GOAL:

name	age	birth_year
Luke Skywalker	19.0	2000.0
C-3PO	112.0	1907.0
R2-D2	33.0	1986.0
Darth Vader	41.9	1977.1
Leia Organa	19.0	2000.0
Owen Lars	52.0	1967.0
Beru Whitesun lars	47.0	1972.0
Biggs Darklighter	24.0	1995.0
Obi-Wan Kenobi	57.0	1962.0
Anakin Skywalker	41.9	1977.1
Wilhuff Tarkin	64.0	1955.0
Chewbacca	200.0	1819.0
Han Solo	29.0	1990.0
Greedo	44.0	1975.0
Jabba Desilijic Tiure	600.0	1419.0
Wedge Antilles	21.0	1998.0
Yoda	896.0	1123.0
Palpatine	82.0	1937.0
Boba Fett	31.5	1987.5
IG-88	15.0	2004.0

Hint: rename "birth_year" as "age" first then calculate "birth_year"



dplyr practice!

Calculate the birth year of each character given the column 'birth_year' is actually age in 2019…

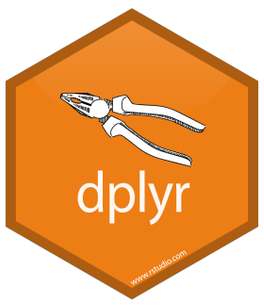
GOAL:

name	age	birth_year
Luke Skywalker	19.0	2000.0
C-3PO	112.0	1907.0
R2-D2	33.0	1986.0
Darth Vader	41.9	1977.1
Leia Organa	19.0	2000.0
Owen Lars		
Beru Whitesun lars		
Biggs Darklighter		
Obi-Wan Kenobi		
Anakin Skywalker		
Wilhuff Tarkin		
Chewbacca	2	
Han Solo		
Greedo	44.0	1975.0
Jabba Desilijic Tiure	600.0	1419.0
Wedge Antilles	21.0	1998.0
Yoda	896.0	1123.0
Palpatine	82.0	1937.0
Boba Fett	31.5	1987.5
IG-88	15.0	2004.0

Hint: rename "birth_year" as "age" first then calculate "birth_year"

CODE:

```
practice1 <- starwars %>%
    dplyr::select(name, age = birth_year) %>%
    dplyr::mutate(birth_year = 2019 - age) %>%
    dplyr::filter(!is.na(age))
```



dplyr practice!

Calculate the number of characters with each hair color combination

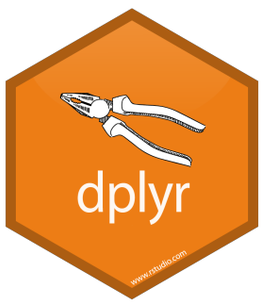
GOAL:

hair_color	number
auburn	1
auburn, grey	1
auburn, white	1
black	13
blond	3
blonde	1
brown	18
brown, grey	1
grey	1
none	37
unknown	1
white	4
NA	5

Note: "brown, grey" is different than "brown" or "grey"

Hint: the function `n()` *counts the number of observations in a group*

BONUS: Combine "blond" and "blonde" as one category and re-calculate



dplyr practice!

Calculate the number of characters with each hair color combination

GOAL:

hair_color	number
auburn	1
auburn, grey	1
auburn, white	1
black	13
blond	3
blonde	1
brown	18
brown, grey	1
grey	
none	
unknown	
white	
NA	

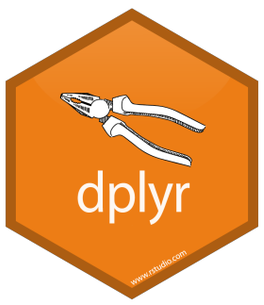
Note: "brown, grey" is different than "brown" or "grey"

Hint: the function `n()` counts the number of observations in a group

BONUS: Combine "blond" and "blonde" as one category and re-calculate

CODE:

```
practice2 <- starwars %>%
    dplyr::select(name, hair_color) %>%
    dplyr::group_by(hair_color) %>%
    dplyr::summarize(number = n())
```



dplyr practice!

Calculate the number of characters with each hair color combination

GOAL:

hair_color	number
auburn	1
auburn, grey	1
auburn, white	1
black	13
blond	3
blonde	1
brown	18

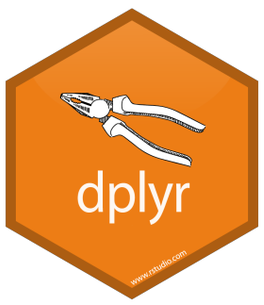
Note: "brown, grey" is different than "brown" or "grey"

Hint: the function `n()` *counts the number of observations in a group*

BONUS: Combine "blond" and "blonde" as one category and re-calculate

CODE:

```
practice2 <- starwars %>%
    dplyr::select(name, hair_color) %>%
    dplyr::mutate(hair_color = ifelse(hair_color == "blond",
                                      "blonde", hair_color)) %>%
    dplyr::group_by(hair_color) %>%
    dplyr::summarize(number = n())
```



dplyr practice!

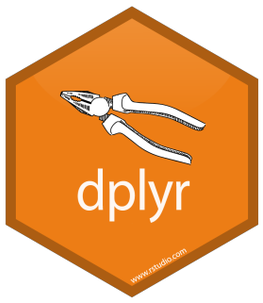
Calculate the number of characters with each hair color combination
Note: Keep ALL characters

GOAL:

name	hair_color	number
Luke Skywalker	blond	3
C-3PO	*NA*	5
R2-D2	*NA*	5
Darth Vader	none	37
Leia Organa	brown	18
Owen Lars	brown, grey	1
Beru Whitesun lars	brown	18
R5-D4	*NA*	5
Biggs Darklighter	black	13
Obi-Wan Kenobi	auburn, white	1
Anakin Skywalker	blond	3
Wilhuff Tarkin	auburn, grey	1
Chewbacca	brown	18
Han Solo	brown	18
Greedo	*NA*	5
Jabba Desilijic Tiure	*NA*	5
Wedge Antilles	brown	18
Jek Tono Porkins	brown	18

Note: "brown, grey" is different than "brown" or "grey"

Hint: the function `n()` *counts the number of observations in a group*



dplyr practice!

Calculate the number of characters with each hair color combination
Note: Keep ALL characters

GOAL:

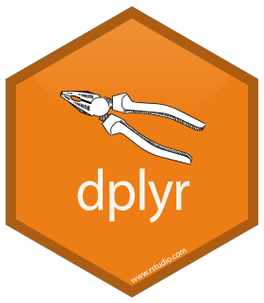
name	hair_color	number
Luke Skywalker	blond	3
C-3PO	*NA*	5
R2-D2	*NA*	5
Darth Vader	none	37
Leia Organa	brown	18
Owen Lars	brown, grey	1
Beru Whitesun lars	brown	
R5-D4	*NA*	
Biggs Darklighter	black	
Obi-Wan Kenobi	auburn, white	
Anakin Skywalker	blond	
Wilhuff Tarkin	auburn, grey	
Chewbacca	brown	
Han Solo	brown	
Greedo	*NA*	5
Jabba Desilijic Tiure	*NA*	5
Wedge Antilles	brown	18
Jek Tono Porkins	brown	18

Note: "brown, grey" is different than "brown" or "grey"

Hint: the function **`n()`** *counts the number of observations in a group*

CODE:

```
practice3 <- starwars %>%
    dplyr::select(name, hair_color) %>%
    dplyr::group_by(hair_color) %>%
    dplyr::mutate(number = n())
```



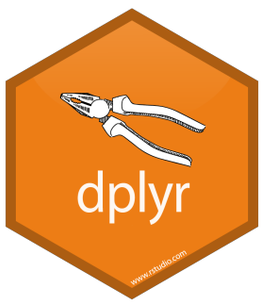
dplyr practice!

Select all characters and their homeworld in the film “Return of the Jedi”

GOAL:

name	homeworld
Luke Skywalker	Tatooine
C-3PO	Tatooine
R2-D2	Naboo
Darth Vader	Tatooine
Leia Organa	Alderaan
Obi-Wan Kenobi	Stewjon
Chewbacca	Kashyyyk
Han Solo	Corellia
Jabba Desilijic Tiure	Nal Hutta
Wedge Antilles	Corellia
Yoda	*NA*
Palpatine	Naboo
Boba Fett	Kamino
Lando Calrissian	Socorro
Ackbar	Mon Cala
Mon Mothma	Chandrila
Arvel Crynyd	*NA*
Wicket Systri Warrick	Endor
Nien Nunb	Sullust
Bib Fortuna	Ryloth

Hint: try using `grepl()`



dplyr practice!

Select all characters and their homeworld in the film “Return of the Jedi”

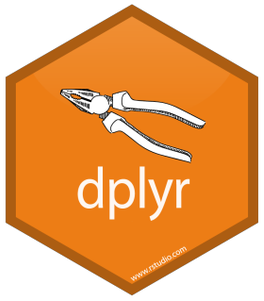
Hint: try using `grepl()`

GOAL:

name	homeworld
Luke Skywalker	Tatooine
C-3PO	Tatooine
R2-	
Dar	
Leia	
Obi	
Che	
Har	
Jabl	
Wed	
Yod	
Palp	
Boba Fett	Kamino
Lando Calrissian	Socorro
Ackbar	Mon Cala
Mon Mothma	Chandrila
Arvel Crynyd	*NA*
Wicket Systri Warrick	Endor
Nien Nunb	Sullust
Bib Fortuna	Ryloth

CODE:

```
practice4 <- starwars %>%
    dplyr::filter(grepl("Return of the Jedi", films)) %>%
    dplyr::select(name, homeworld)
```



Day 2

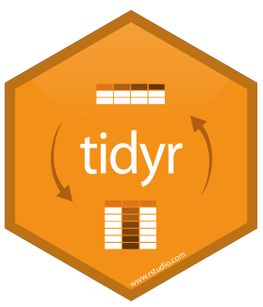
tidyr, readr, stringr, lubridate

Introduction: **tidyr**

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

<u>Functions:</u>

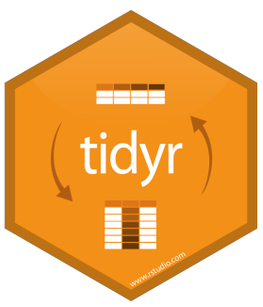
- `gather()` to collapse multiple columns
- `spread()` to expand one column to multiple
- `unite()` to combine multiple columns into one
- `separate()` to split one column into two
- `separate_rows()` to split one cell into several rows
- `drop_na()` to remove rows with NA values



Wide vs. Long data

Wide

Long



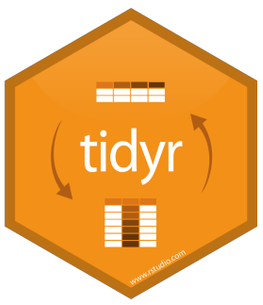
Wide vs. Long data

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?

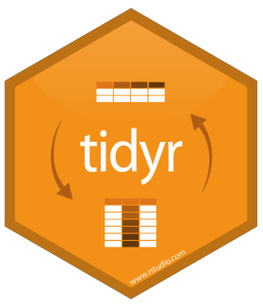
Wide vs. Long data

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



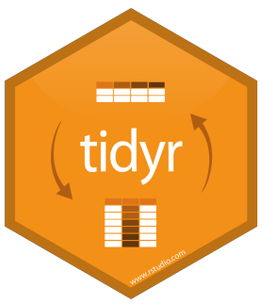
Q1: Find the average of each student's midterms

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

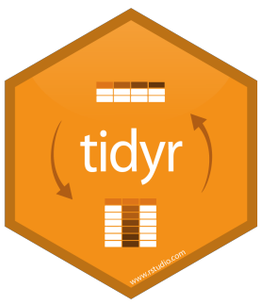
Wide vs. Long data

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



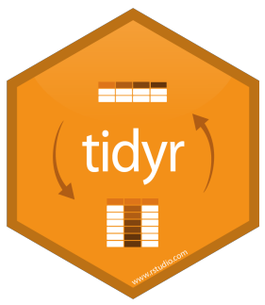
Q1: Find the average of each student's midterms

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

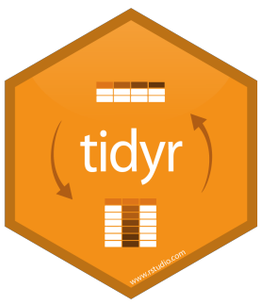
Wide vs. Long data

```
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

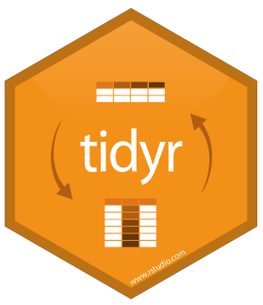
Wide vs. Long data

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



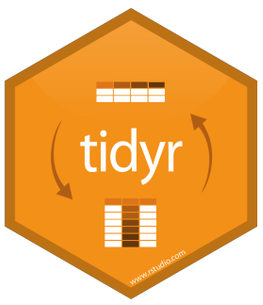
Q2: Plot each student's score by midterm

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

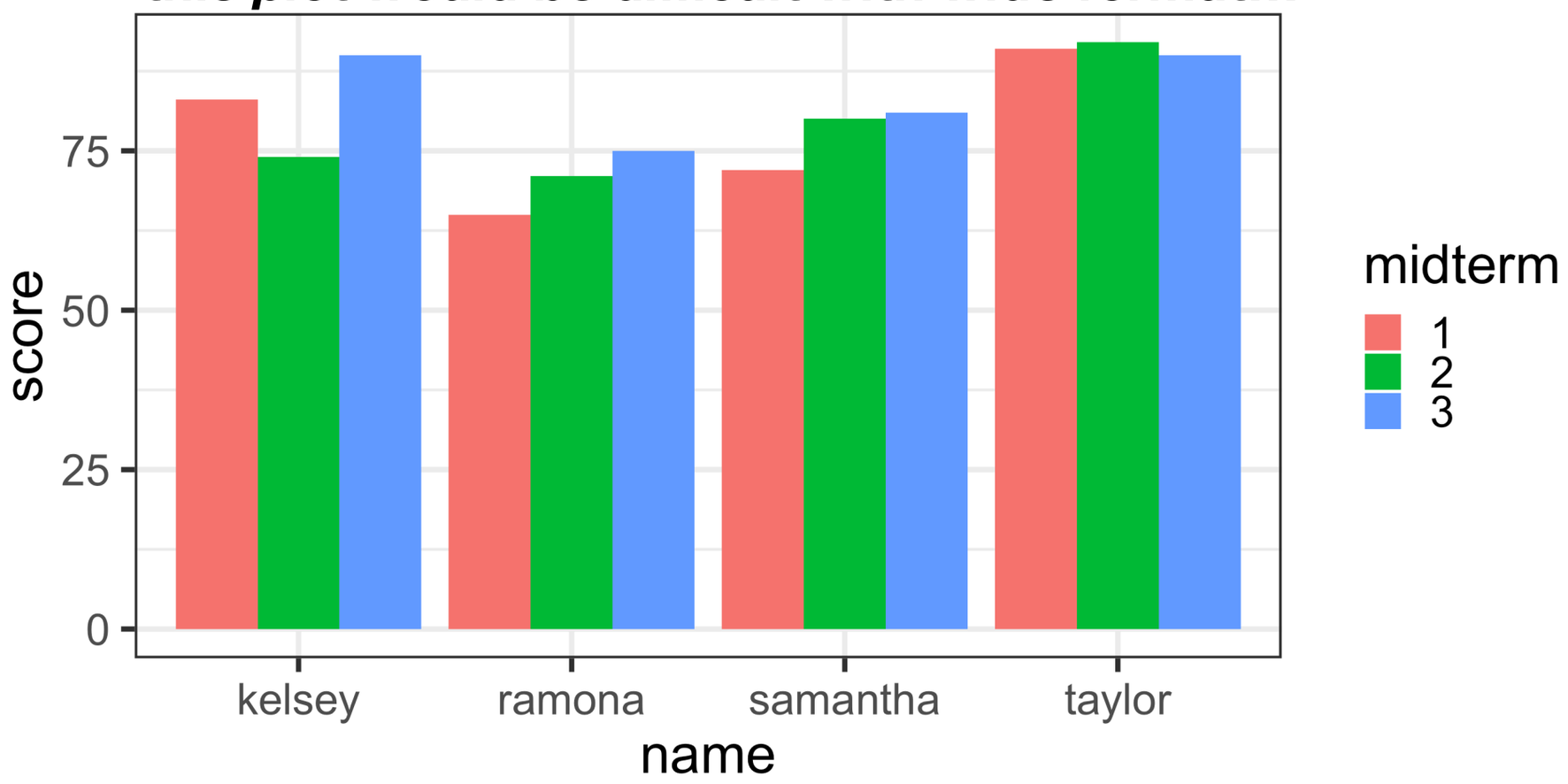


Q2: Plot each student's score by midterm

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

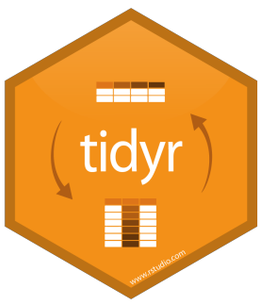
Wide vs. Long data

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



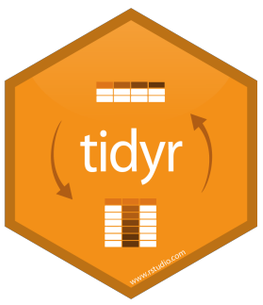
Q3: Find the ratio between midterm 1 and 2

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

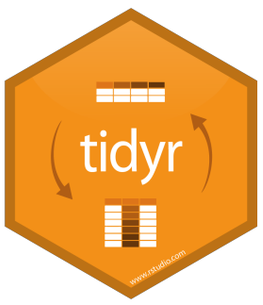
Wide vs. Long data

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

Starwars dataset

WIDE

LONG

variables

name	height	mass	hair_color	skin_color	eye_color	birth_year	gender	homeworld	species	films	vehicles	starships
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")
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)
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)
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
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)
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)
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)
R5-D4	97	32.0	*NA*	white, red	red	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)
Biggs Darklighter	183	84.0	black	light	brown	24.0	male	Tatooine	Human	A New Hope	character(0)	X-wing
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...
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...
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)
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")
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")
Greedo	173	74.0	*NA*	green	black	44.0	male	Rodia	Rodian	A New Hope	character(0)	character(0)
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)
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
Jek Tono Porkins	180	110.0	brown	fair	blue	*NA*	male	Bestine IV	Human	A New Hope	character(0)	X-wing
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)
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)
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
IG-88	200	140.0	none	metal	red	15.0	none	*NA*	Droid	The Empire Strikes Back	character(0)	character(0)
Bossk	190	113.0	none	green	red	53.0	male	Trandosha	Trandoshan	The Empire Strikes Back	character(0)	character(0)
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

Starwars dataset

☐ **WIDE**

☑ **LONG**

variables

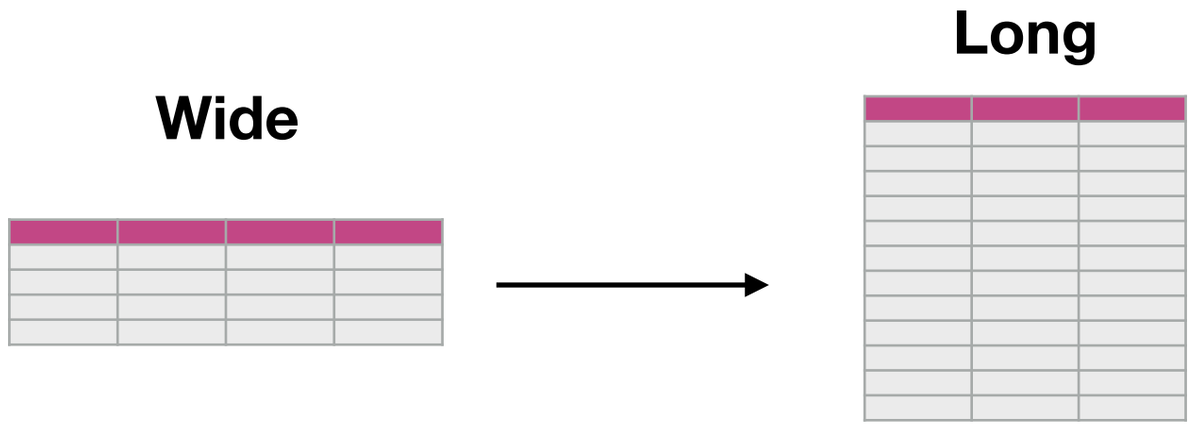
name	characteristic	value
Luke Skywalker	height	172
Luke Skywalker	mass	77
Luke Skywalker	hair_color	blond
Luke Skywalker	skin_color	fair
Luke Skywalker	eye_color	blue
Luke Skywalker	birth_year	19
Luke Skywalker	gender	male
Luke Skywalker	homeworld	Tatooine
Luke Skywalker	species	Human
Luke Skywalker	films	c("Revenge of the Sith", "Return of the Jedi", "The Emp...
Luke Skywalker	vehicles	c("Snowspeeder", "Imperial Speeder Bike")
Luke Skywalker	starships	c("X-wing", "Imperial shuttle")
Luminara Unduli	height	170
Luminara Unduli	mass	56.2
Luminara Unduli	hair_color	black
Luminara Unduli	skin_color	yellow
Luminara Unduli	eye_color	blue
Luminara Unduli	birth_year	58
Luminara Unduli	gender	female

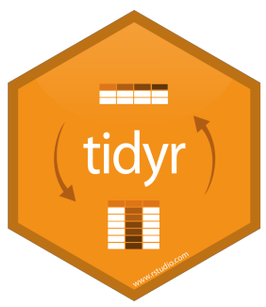
observations

values

MESSY DATA

- Each **value** must have its own **cell**



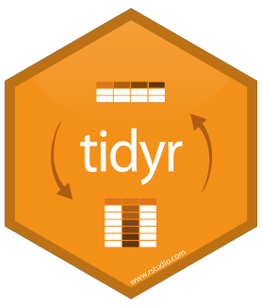


`tidyr::gather()`

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

`tidyr::gather()`

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

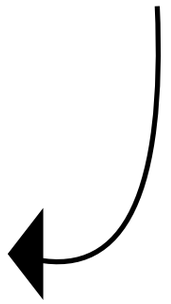


`tidyr::gather()` to collapse multiple columns

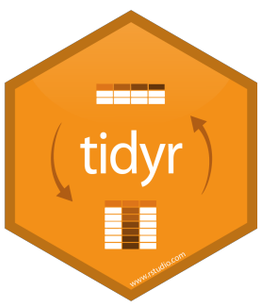
`tidyr::gather()`

`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

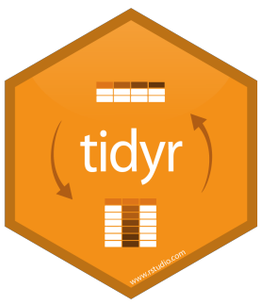
`tidyr::gather()`

`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)
```



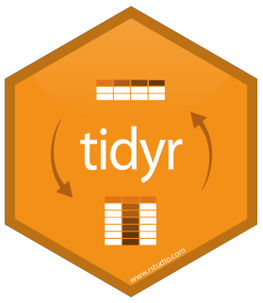
`tidyr::gather()` to collapse multiple columns

`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()`

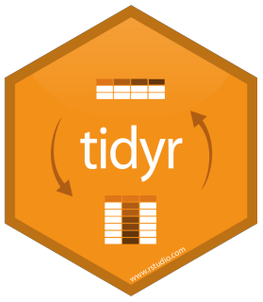
`tidyr::gather(dataframe, key, value, columns_to_include)`

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)
```



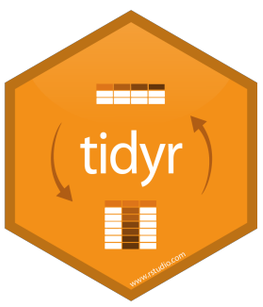
`tidyr::gather()` to collapse multiple columns

`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)
```



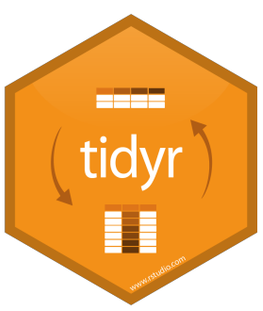
`tidyr::gather()` to collapse multiple columns

`tidyr::gather()`

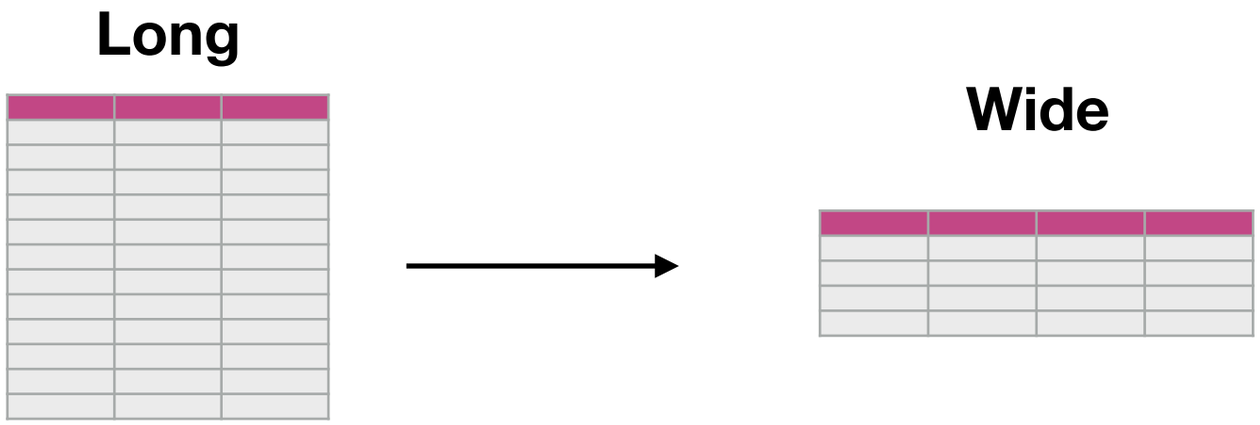
```
tidyr::gather(dataframe, key, value, columns_to_include)
```

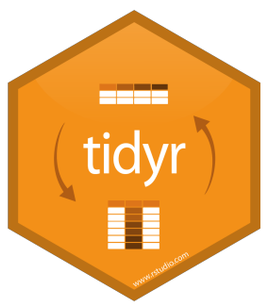
Gather height, mass, eye color, skin color, and gender

name	hair_color	birth_year	homeworld	species	films	vehicles	starships	characteristic	value
Luke Skywalker	blond	19.0	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")	height	172
C-3PO	*NA*	112.0	Tatooine	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)	height	167
R2-D2	*NA*	33.0	Naboo	Droid	c("Attack of the Clones", "The Phantom Menace", "Rev...	character(0)	character(0)	height	96
Darth Vader	none	41.9	Tatooine	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	character(0)	TIE Advanced x1	height	202
Leia Organa	brown	19.0	Alderaan	Human	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	Imperial Speeder Bike	character(0)	height	150
Owen Lars	brown, grey	52.0	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)	height	178
Beru Whitesun lars	brown	47.0	Tatooine	Human	c("Attack of the Clones", "Revenge of the Sith", "A New...	character(0)	character(0)	height	165
R5-D4	*NA*	*NA*	Tatooine	Droid	A New Hope	character(0)	character(0)	height	97
Biggs Darklighter	black	24.0	Tatooine	Human	A New Hope	character(0)	X-wing	height	183
Obi-Wan Kenobi	auburn, white	57.0	Stewjon	Human	c("Attack of the Clones", "The Phantom Menace", "Rev...	Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...	height	182
Anakin Skywalker	blond	41.9	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...	height	188
Wilhuff Tarkin	auburn, grey	64.0	Eriadu	Human	c("Revenge of the Sith", "A New Hope")	character(0)	character(0)	height	180
Chewbacca	brown	200.0	Kashyyyk	Wookiee	c("Revenge of the Sith", "Return of the Jedi", "The Emp...	AT-ST	c("Millennium Falcon", "Imperial shuttle")	height	228
Han Solo	brown	29.0	Corellia	Human	c("Return of the Jedi", "The Empire Strikes Back", "A N...	character(0)	c("Millennium Falcon", "Imperial shuttle")	height	180
Greedo	*NA*	44.0	Rodia	Rodian	A New Hope	character(0)	character(0)	height	173



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



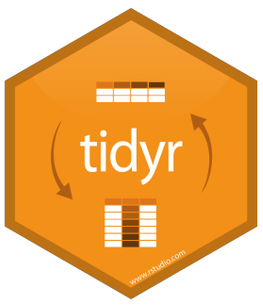


`tidyr::spread()`

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

`tidyr::spread()`

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



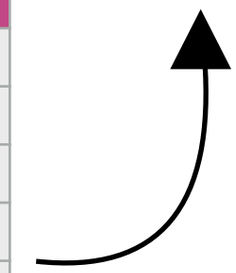
`tidyr::spread()` to expand one column to multiple

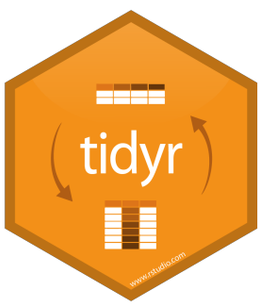
`tidyr::spread()`

`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





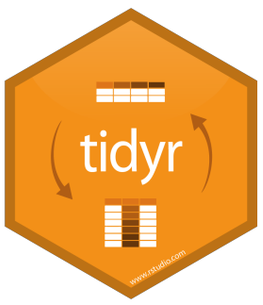
`tidyr::spread()` to expand one column to multiple

`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

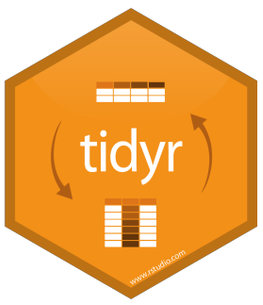
`tidyr::spread()`

`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)`

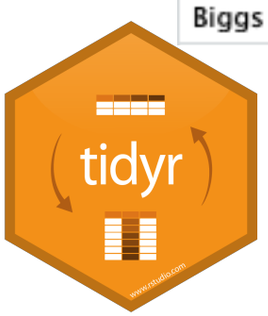
Un-gather (spread) the starwars data frame

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

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

→

name	height	mass	hair_color	skin_color	eye_color	bir
Luke Skywalker	172	77.0	blond	fair	blue	
C-3PO	167	75.0	*NA*	gold	yellow	
R2-D2	96	32.0	*NA*	white, blue	red	
Darth Vader	202	136.0	none	white	yellow	
Leia Organa	150	49.0	brown	light	brown	
Owen Lars	178	120.0	brown, grey	light	blue	
Beru Whitesun lars	165	75.0	brown	light	blue	
R5-D4	97	32.0	*NA*	white, red	red	
Biggs Darklighter	183	84.0	black	light	brown	



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

tidyr::spread()

name = key

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

GOAL:

Ackbar	Adi Gallia	Anakin Skywalker	Arvel Crynyd	Ayla Secura	Bail Prestor Organa	Barriss Offee	BB8	Ben Quadinaros	Beru Whitesun lars	Bib Fortuna	Biggs Darklighter	Boba Fett	Bossk	C-3PO	Captain Phasma	Chewbacca
orange	blue	blue	brown	hazel	brown	blue	black	orange	blue	pink	brown	brown	red	yellow	unknown	blue

eye_color = value

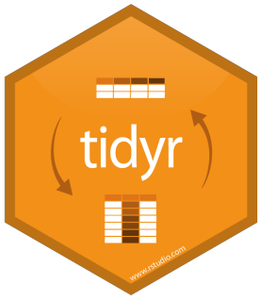
1. Select columns to keep

name, eye_color

2. Spread data frame

What is the key? What is the value?

```
eyes <- starwars %>%
   dplyr::select(name, eye_color) %>%
   tidyr::spread(name, eye_color)
```



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

tidyr::spread()

name = key

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

GOAL:

Ackbar	Adi Gallia	Anakin Skywalker	Arvel Crynyd	Ayla Secura	Bail Prestor Organa	Barriss Offee	BB8	Ben Quadinaros	Beru Whitesun lars	Bib Fortuna	Biggs Darklighter	Boba Fett	Bossk	C-3PO	Captain Phasma	Chewbacca
orange	blue	blue	brown	hazel	brown	blue	black	orange	blue	pink	brown	brown	red	yellow	unknown	blue

eye_color = value

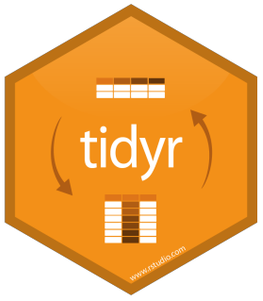
~~**1. Select columns to keep**~~

~~name, eye_color~~

2. Spread data frame

What is the key? What is the value?

```
eyes <- starwars %>%
  ~~dplyr::select(name, eye_color) %>%~~
  tidyr::spread(name, eye_color)
```

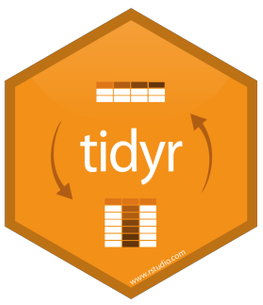


`tidyr::spread()` to expand one column to multiple

tidyr::spread()

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

vehicles	starships	Ackbar	Adi Gallia	Anakin Skywalker	Arvel Crynyd	Ayla Secura	Bail Prestor Organa	Barriss Offee	BB8	Ben Quadinaros	Beru Whitesun lars	Bib Fortuna	B D
c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	TIE Advanced x1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
Imperial Speeder Bike	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	blue	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	X-wing	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...	NA	NA	blue	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
AT-ST	c("Millennium Falcon", "Imperial shuttle")	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	c("Millennium Falcon", "Imperial shuttle")	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
Snowspeeder	X-wing	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	X-wing	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	Slave 1	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	Millennium Falcon	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	orange	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	A-wing	NA	NA	NA	brown	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	Millennium Falcon	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
Tribubble bongo	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	
character(0)	character(0)	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	



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

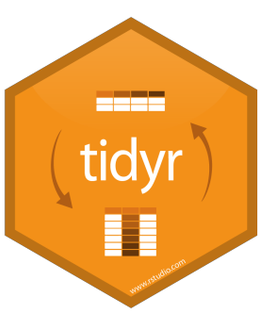
`tidyr::spread()`

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

vehicles	starships	Ackbar	Adi Gallia	Anakin Skywalker	Arvel Crynyd	Ayla Secura	Bail Prestor Organa	Barriss Offee	BB8	Ben Quadinaros	Beru Whitesun lars	Bib Fortuna	B D
c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	TIE Advanced x1	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
Imperial Speeder Bike	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	blue	*NA*	
character(0)	character(0)	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
character(0)	X-wing	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	
c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...	*NA*	*NA*	blue	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	

★ Can fill empty values with `fill = "xxx"`

```
eyes <- starwars %>%
  dplyr::select(name, eye_color) %>%
  tidyr::spread(name, eye_color, fill = 0)
```



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

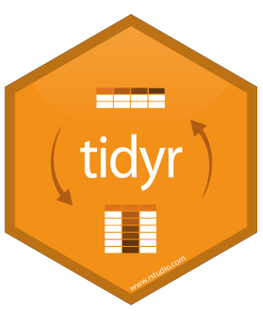
tidyr::spread()

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

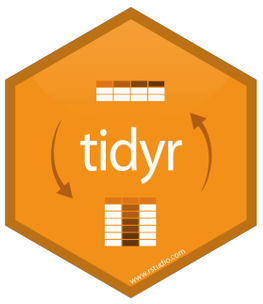
hicles	starships	Ackbar	Adi Gallia	Anakin Skywalker	Arvel Crynyd	Ayla Secura	Bail Prestor Organa	Barriss Offee	BB8	Ben Quadinaros	Beru Whitesun lars	Bib Fortuna	B D
c("Snowspeeder", "Imperial Speeder Bike")	c("X-wing", "Imperial shuttle")	0	0	0	0	0	0	0	0	0	0	0	
character(0)	character(0)	0	0	0	0	0	0	0	0	0	0	0	
character(0)	character(0)	0	0	0	0	0	0	0	0	0	0	0	
character(0)	TIE Advanced x1	0	0	0	0	0	0	0	0	0	0	0	
Imperial Speeder Bike	character(0)	0	0	0	0	0	0	0	0	0	0	0	
character(0)	character(0)	0	0	0	0	0	0	0	0	0	0	0	
character(0)	character(0)	0	0	0	0	0	0	0	0	0	blue	0	
character(0)	character(0)	0	0	0	0	0	0	0	0	0	0	0	
character(0)	X-wing	0	0	0	0	0	0	0	0	0	0	0	
Tribubble bongo	c("Jedi starfighter", "Trade Federation cruiser", "Naboo...	0	0	0	0	0	0	0	0	0	0	0	
c("Zephyr-G swoop bike", "XJ-6 airspeeder")	c("Trade Federation cruiser", "Jedi Interceptor", "Nabo...	0	0	blue	0	0	0	0	0	0	0	0	

Can fill empty values with `fill = "xxx"`

```
eyes <- starwars %>%
  dplyr::select(name, eye_color) %>%
  tidyr::spread(name, eye_color, fill = 0)
```



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



`tidyr::unite()`

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

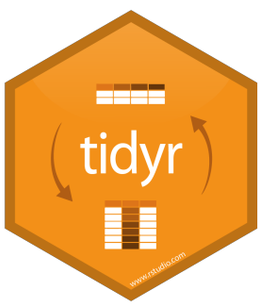
`tidyr::unite()`

`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()`

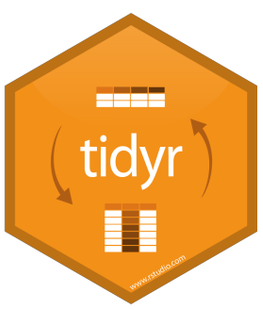
`tidyr::unite(`dataframe, old_columns, `col =` new_column, `sep =` separator`)`

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 = ", ")
```

Keep the original columns with argument `remove = FALSE`

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



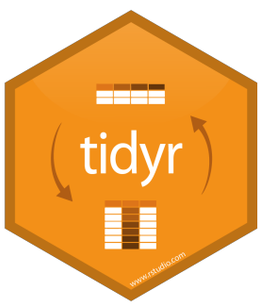
`tidyr::unite()` to combine multiple columns into one

`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::unite()`

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

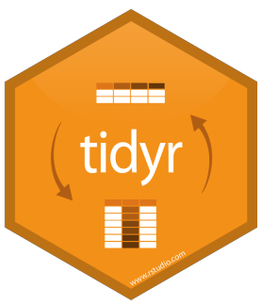
Combine name, homeworld, and species into one column named character

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

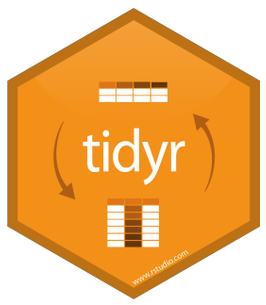
Hint: This might take multiple steps…

Hint: `paste("My name is", "Katie", sep = " ")` = "My name is Katie"

```
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::separate()`

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

tidyr::separate()

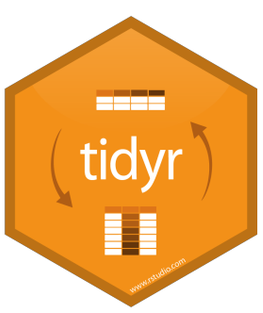
`tidyr::separate(dataframe, old_column, into = c(new_columns), sep = separator)`

Separate the 'character' column back into name and homeworld

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

Keep the original columns with argument `remove = FALSE`

```
separated <- united %>%
 tidyr::separate(character, into = c("name", "homeworld"),
                 sep = “,”, remove = FALSE)
```



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

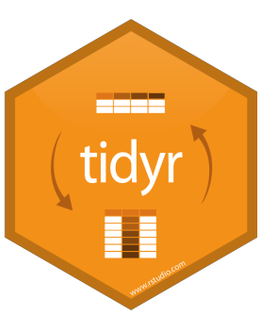
tidyr::separate()

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

Separate characters into first and last names

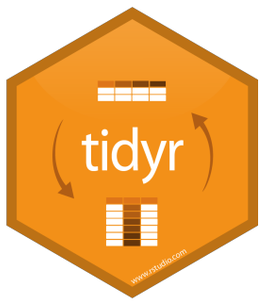
```
names <- starwars %>%
tidyr::separate(name, into = c("first_name", "last_name"), sep = " ")
```

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



Beru Whitesun lars

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



`tidyr::separate_rows()`

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

`tidyr::separate_rows()`

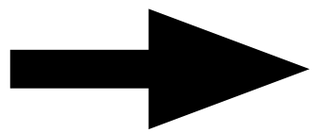
`tidyr::separate_rows(`dataframe, column(s), `sep =` separator`)`

Separate the hair_color to be two observations (rows)

```
hair_cleaned <- starwars %>%
 tidyr::separate_rows(hair_color, sep = ", ")
```

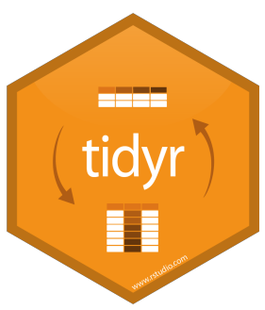
name	height	mass	hair_color
Luke Skywalker	172	77.0	blond
C-3PO	167	75.0	*NA*
R2-D2	96	32.0	*NA*
Darth Vader	202	136.0	none
Leia Organa	150	49.0	brown
Owen Lars	178	120.0	brown, grey
Beru Whitesun lars	165	75.0	brown
R5-D4	97	32.0	*NA*
Biggs Darklighter	183	84.0	black
Obi-Wan Kenobi	182	77.0	auburn, white

MESSY DATA



name	height	mass	hair_color
Luke Skywalker	172	77.0	blond
C-3PO	167	75.0	*NA*
R2-D2	96	32.0	*NA*
Darth Vader	202	136.0	none
Leia Organa	150	49.0	brown
Owen Lars	178	120.0	brown
Owen Lars	178	120.0	grey
Beru Whitesun lars	165	75.0	brown
R5-D4	97	32.0	*NA*
Biggs Darklighter	183	84.0	black

TIDY DATA



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

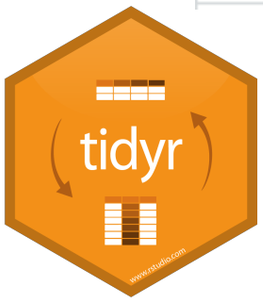
`tidyr::separate_rows()`

`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

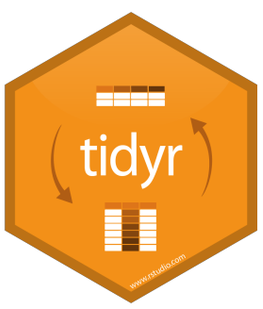
```
separated <- starwars %>%
tidyr::separate_rows(films, sep = ", ")
```

We will come back to this later to clean up the new films column…

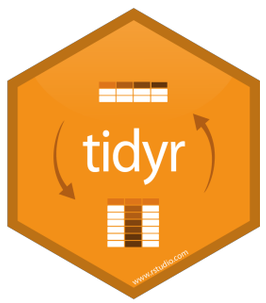
name	films
Luke Skywalker	c("Revenge of the Sith"
Luke Skywalker	"Return of the Jedi"
Luke Skywalker	"The Empire Strikes Back"
Luke Skywalker	"A New Hope"
Luke Skywalker	"The Force Awakens")

→

name	films
Luke Skywalker	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



`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

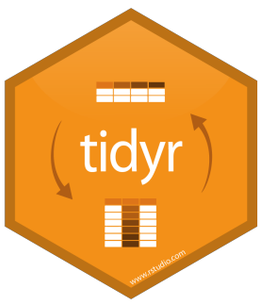
```
no_species <- starwars %>%
  tidyr::drop_na(species)
```

87 rows > 82 rows

Remove all rows with any NA value

```
no_na <- starwars %>%
  na.omit()
```

87 rows > 29 rows

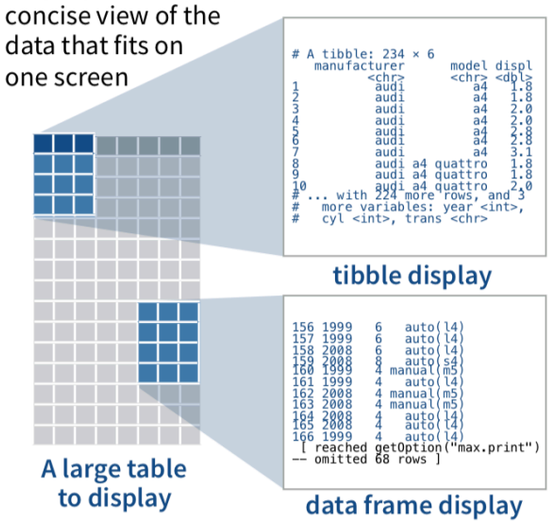


`tidyr::drop_na()` to remove rows with NA values

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen




```
# A tibble: 234 × 6
   manufacturer       model displ
          <chr>       <chr> <dbl>
1          audi          a4   1.8
2          audi          a4   1.8
3          audi          a4   2.0
4          audi          a4   2.0
5          audi          a4   2.8
6          audi          a4   2.8
7          audi          a4   3.1
8          audi a4 quattro   1.8
9          audi a4 quattro   1.8
10         audi a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

```
156 1999   6   auto(l4)
157 1999   6   auto(l4)
158 2008   6   auto(l4)
159 2008   8   auto(s4)
160 1999   4 manual(m5)
161 1999   4   auto(l4)
162 2008   4 manual(m5)
163 2008   4 manual(m5)
164 2008   4   auto(l4)
165 2008   4   auto(l4)
166 1999   4   auto(l4)
 [ reached getOption("max.print")
-- omitted 68 rows ]
```

- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.

```
tribble( ~x,   ~y,
          1,   "a",
          2,   "b",
          3,   "c")
```

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

Tidy Data with tidyr

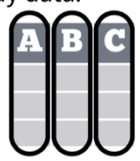
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

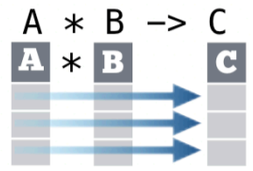
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, sep = "/", into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate, sep = "/")

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	00
Brazil	19	99
Brazil	20	00
China	19	99
China	20	00

→

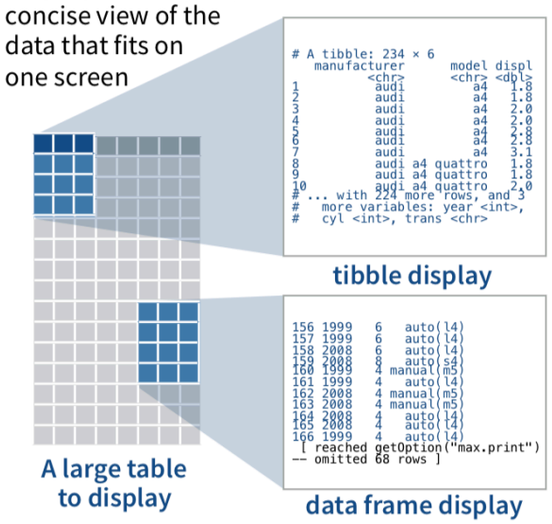
country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen



- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.
tribble(~x, ~y, 1, "a", 2, "b", 3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

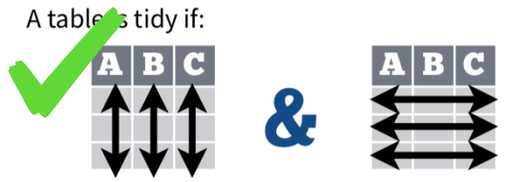
enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

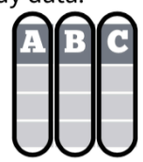
A table is tidy if:



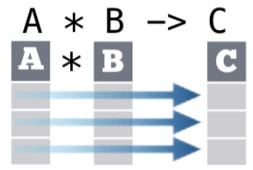
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list())
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...)
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, sep = "/", into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

Separate each cell in a column to make several rows.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate, sep = "/")

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	00
Brazil	19	99
Brazil	20	00
China	19	99
China	20	00

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

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.tsv")
```

```
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

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv"**)**
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	
4	5	NA

Comma Delimited Files
read_csv("file.csv"**)**
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table()**.
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

1	2	3
4	5	NA

Skip lines
read_c**skip = 1**)

A	B	C
1	2	3
4	5	NA

No header
read_c**col_names = FALSE**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_n mes = c("x", "y", "z")**)

A	B	C
NA	2	3
4	5	NA

Missing Value
read_csv(f, **= c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the def
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

x$A <- parse_number(x$A)

Introduction: **stringr**

- Collection of functions to work with character strings
- Output of stringr functions is a vector of TRUE/FALSE

Sections:

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

Output of `dplyr::filter()`
is an object (e.g. dataframe)

Can use stringr functions
in combination with dplyr!

stringr - detect matches

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

Keep all hair colors that mention brown

Hint: combine `filter()` *and* `stringr()`

```
stringr::str_detect(starwars$hair_color, “brown”)
```

```
 [1] FALSE    NA    NA FALSE  TRUE  TRUE  TRUE    NA FALSE FALSE FALSE FALSE  TRUE  TRUE    NA    NA
[38] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[75] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
```

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

name	height	mass	hair_color
Leia Organa	150	49	brown
Owen Lars	178	120	brown, grey
Beru Whitesun lars	165	75	brown
Chewbacca	228	112	brown
Han Solo	180	80	brown
Wedge Antilles	170	77	brown

stringr - detect matches

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

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

```
starwars %>%
  dplyr::mutate(letter_a = stringr::str_count(name, "a"))
```

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

*Hint: “this|that” reads “this **OR** that”*

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

name	vowels
Luke Skywalker	4
C-3PO	0
R2-D2	0
Darth Vader	3
Leia Organa	5

6

stringr - detect matches

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

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

```
starwars %>%
  dplyr::mutate(letter_a = stringr::str_count(name, "a"))
```

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

*Hint: “this|that” reads “this **OR** that”*

```
starwars %>%
  dplyr::mutate(vowels = stringr::str_count(name,
                                            "a|e|i|o|u|A|E|I|O|U"))
```

name	vowels
Luke Skywalker	4
C-3PO	1
R2-D2	0
Darth Vader	3
Leia Organa	6

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

name	vowels
luke skywalker	4
c-3po	1
r2-d2	0
darth vader	3
leia organa	6

`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"))
```

name	species
Luke Skywalker	Homo sapiens
C-3PO	Droid
R2-D2	Droid
Darth Vader	Homo sapiens
Leia Organa	Homo sapiens
Owen Lars	Homo sapiens
Beru Whitesun lars	Homo sapiens
R5-D4	Droid
Biggs Darklighter	Homo sapiens
Obi-Wan Kenobi	Homo sapiens
Anakin Skywalker	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”))
```

name
Luke Skywalker, PhD
C-3PO, PhD
R2-D2, PhD
Darth Vader, PhD
Leia Organa, PhD
Owen Lars, PhD

Compare to `paste()`

```
doctorates2 <- starwars %>%
  dplyr::mutate(name = paste(name, “, PhD”, sep = “”))
```

stringr - join and split

```
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*

```
> stringr::str_split_fixed("Luke Skywalker", " ", 2)
     [,1]   [,2]
[1,] "Luke" "Skywalker"
```

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

Compare to `tidyr::separate()`


```
names <- starwars %>%
  tidyr::separate(name, c(“first_name”, “last_name”), sep = “ “)
```

stringr - join and split

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

stringr::str_split_fixed()

name	first_name	last_name
Luke Skywalker	Luke	Skywalker
C-3PO	C-3PO	
R2-D2	R2-D2	
Darth Vader	Darth	Vader
Leia Organa	Leia	Organa
Owen Lars	Owen	Lars
Beru Whitesun lars	Beru	Whitesun lars
R5-D4	R5-D4	
Biggs Darklighter	Biggs	Darklighter
Obi-Wan Kenobi	Obi-Wan	Kenobi

tidyr::separate()

first_name	last_name	
Luke	Skywalker	*NA*
C	3PO	*NA*
R2	D2	*NA*
Darth	Vader	*NA*
Leia	Organa	*NA*
Owen	Lars	*NA*
Beru	Whitesun	lars
R5	D4	*NA*
Biggs	Darklighter	*NA*
Obi	Wan	Kenobi

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 .!?\(){}

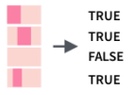
[1] Many base R functions require classes to be wrapped in a second set of [ ], e.g. **[[:digit:]]**

stringr

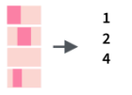
String manipulation with stringr : : **CHEAT SHEET**

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches



str_detect(string, **pattern**) Detect the presence of a pattern match in a string.
str_detect(fruit, "a")

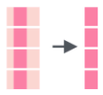


str_which(string, **pattern**) Find the indexes of strings that contain a pattern match.
str_which(fruit, "a")

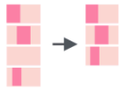
str_count(string, **pattern**) Count the number of matches in a string.
str_count(fruit, "a")

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

Subset Strings



str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector.
str_sub(fruit, 1, 3); str_sub(fruit, -2)

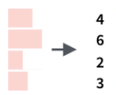


str_subset(string, **pattern**) Return only the strings that contain a pattern match.
str_subset(fruit, "b")

str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**.
str_match(sentences, "(a|the) ([^]+)")

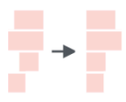
Manage Lengths



str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

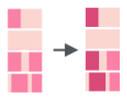
str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis.
str_trunc(fruit, 3)



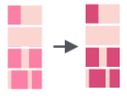
str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results.
str_sub(fruit, 1, 3) <- "str"



str_replace(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*



str_replace_all(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

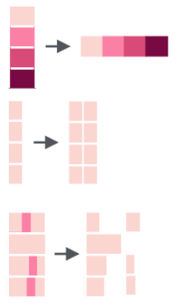
str_to_lower(string, locale = "en")[1] Convert strings to lower case.
str_to_lower(sentences)

str_to_upper(string, locale = "en")[1] Convert strings to upper case.
str_to_upper(sentences)

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split

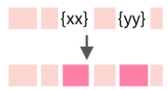
str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string.
str_c(letters, LETTERS)



str_c(..., sep = "", **collapse = ""**) Collapse a vector of strings into a single string.
str_c(letters, collapse = "")

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings.
str_split_fixed(fruit, " ", n=2)



str_glue(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate.
str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")

Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector.
str_sort(x)

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches.
str_view_all(fruit, "[aeiou]")

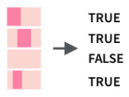
str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches



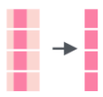
str_detect(string, **pattern**) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

str_which(string, **pattern**) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

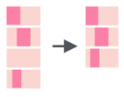
str_count(string, **pattern**) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

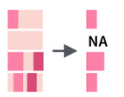
Subset Strings



str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*



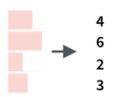
str_subset(string, **pattern**) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*



str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

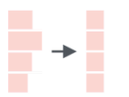
str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

Manage Lengths

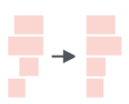


str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

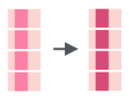


str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

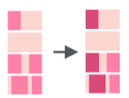


str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

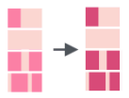
Mutate Strings



str_sub() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*



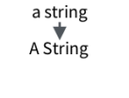
str_replace(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*



str_replace_all(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

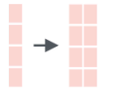


str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

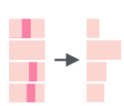
Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

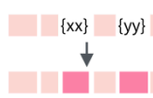
str_c(..., sep = "", **collapse = ""**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*



str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*



str_split_fixed(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*



str_glue(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*

str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

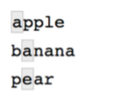
Order Strings

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

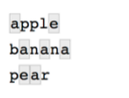
str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*



str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*



str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See bit.ly/ISO639-1 for a complete list of locales.

Introduction: **lubridate**

- Collection of functions to work with dates/times

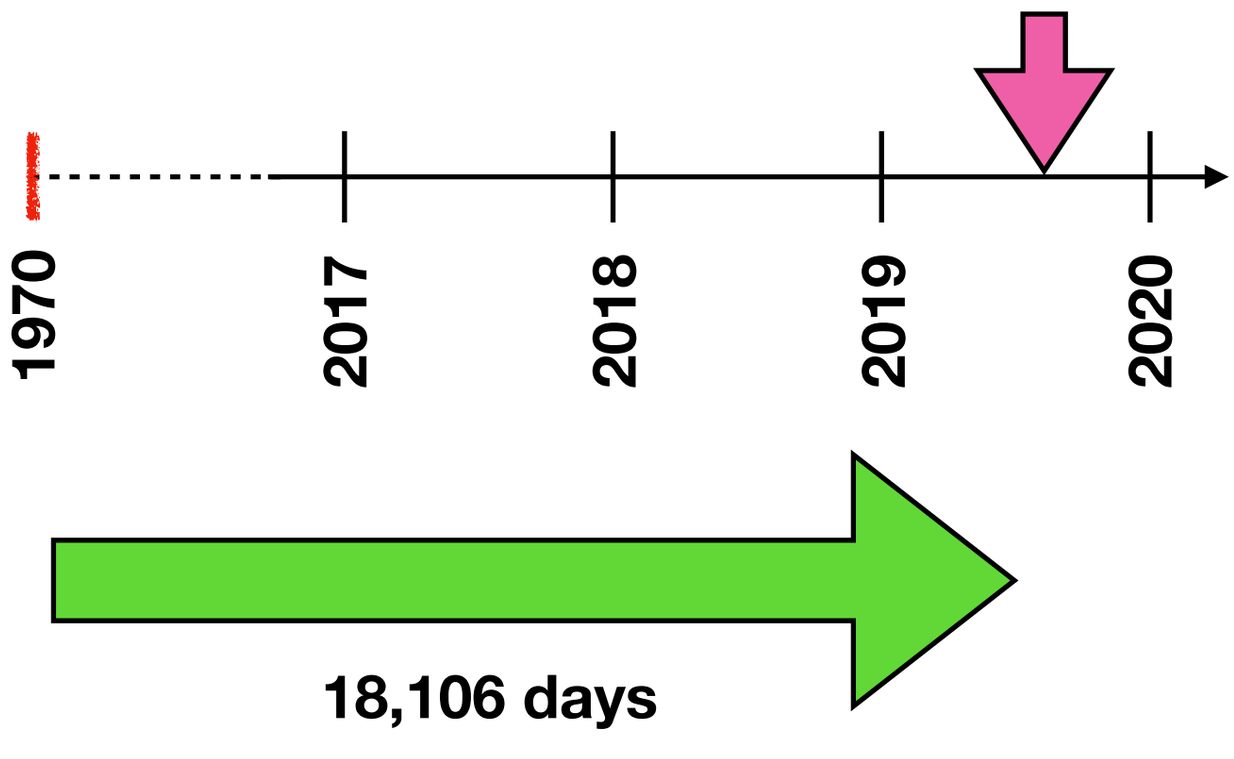
Sections:

- 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

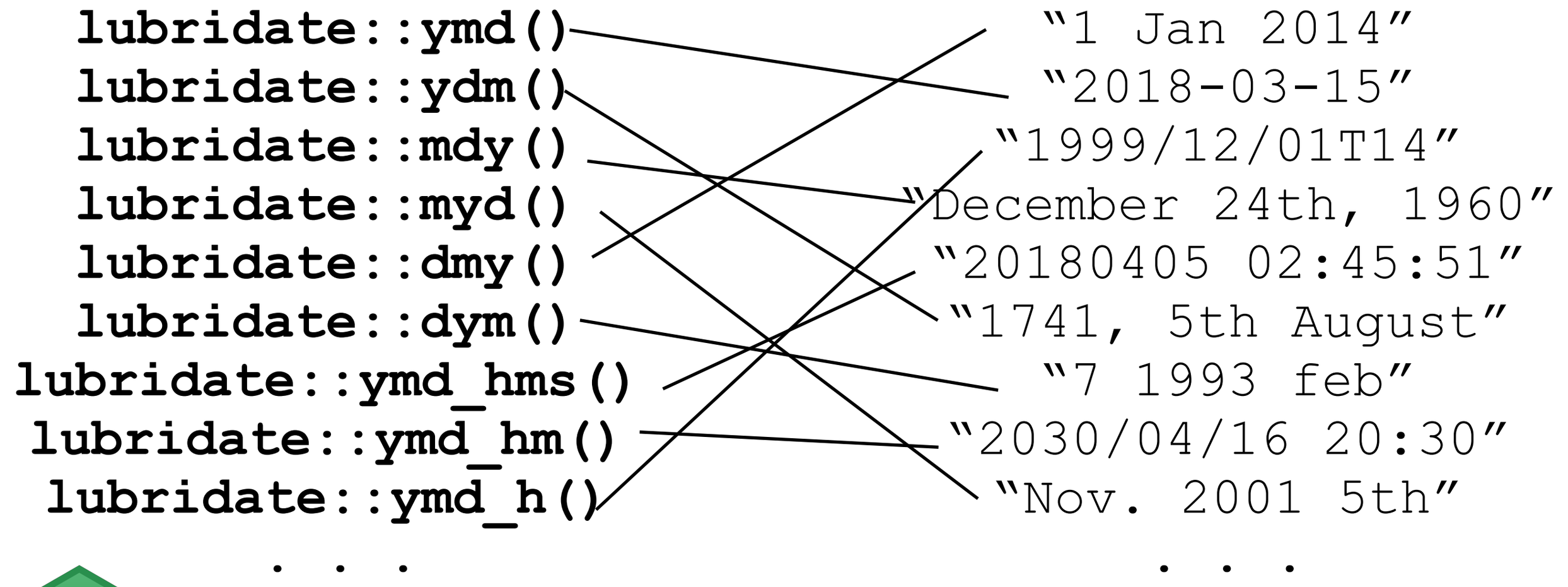
Match the code (left) that should be used to parse the dates (right)

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

What day will it be in 30 days?

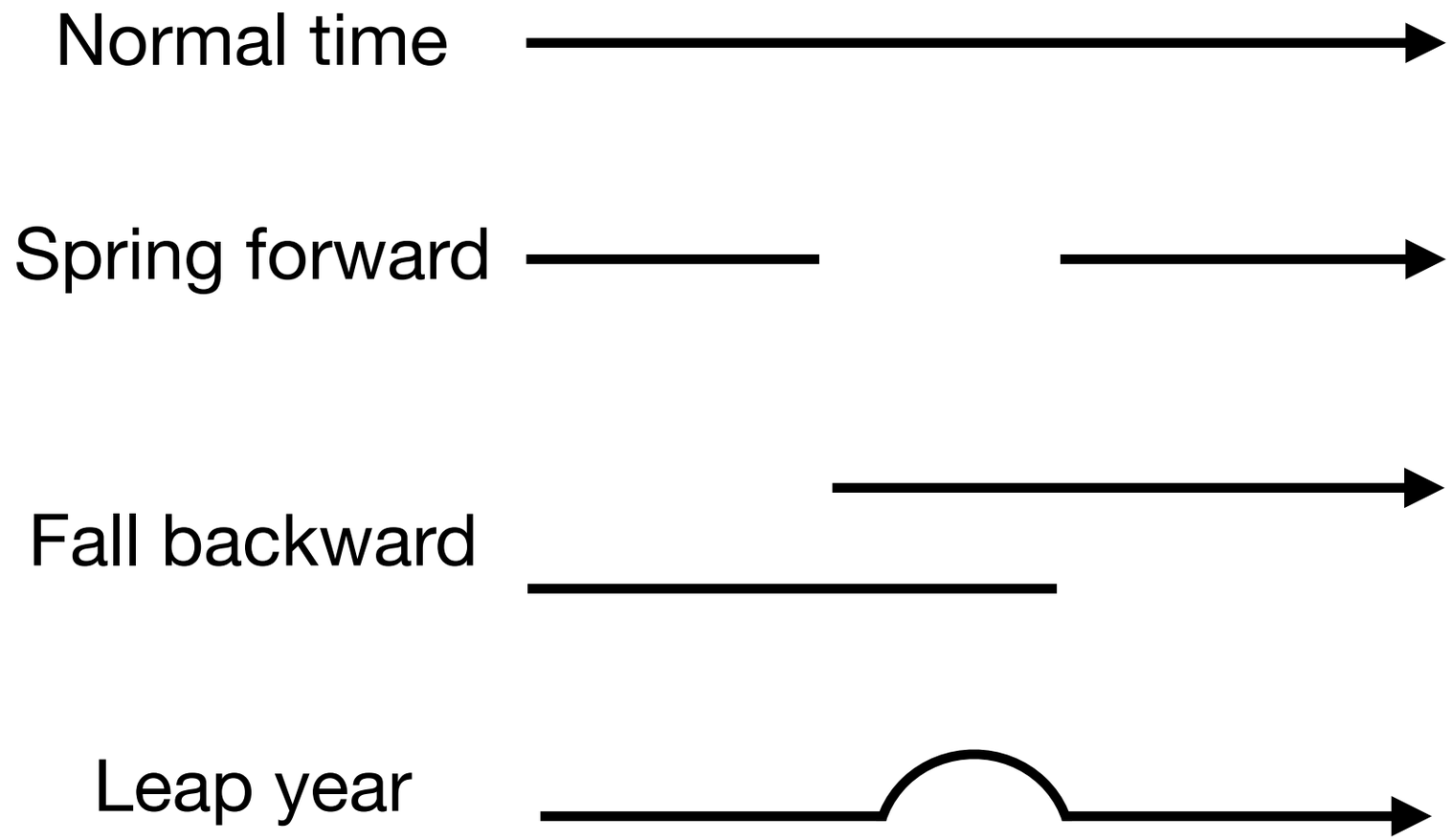
```
lubridate::now() + lubridate::days(30)
```

What time is it in London?

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

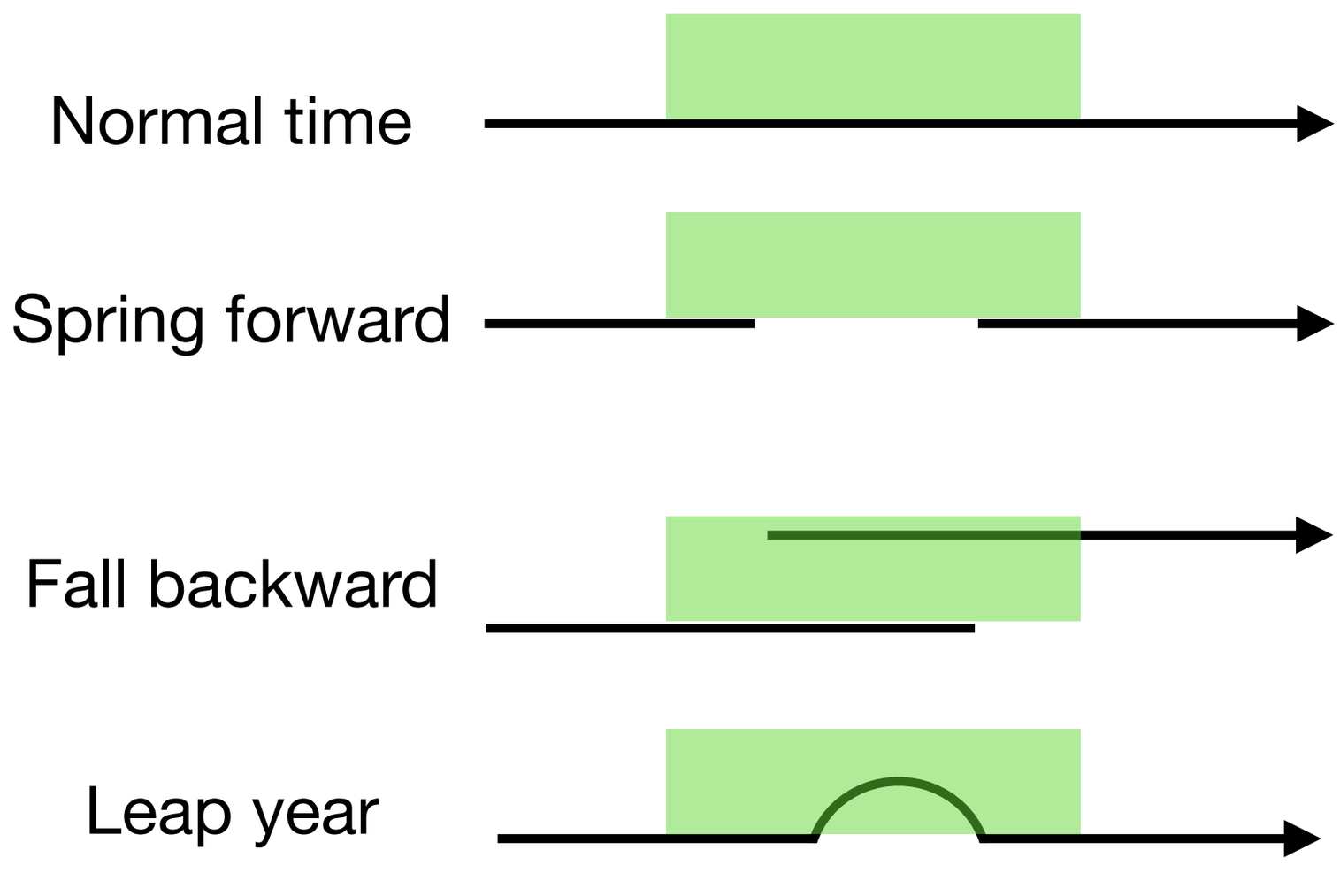
Intervals: represent specific intervals of the timeline

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”

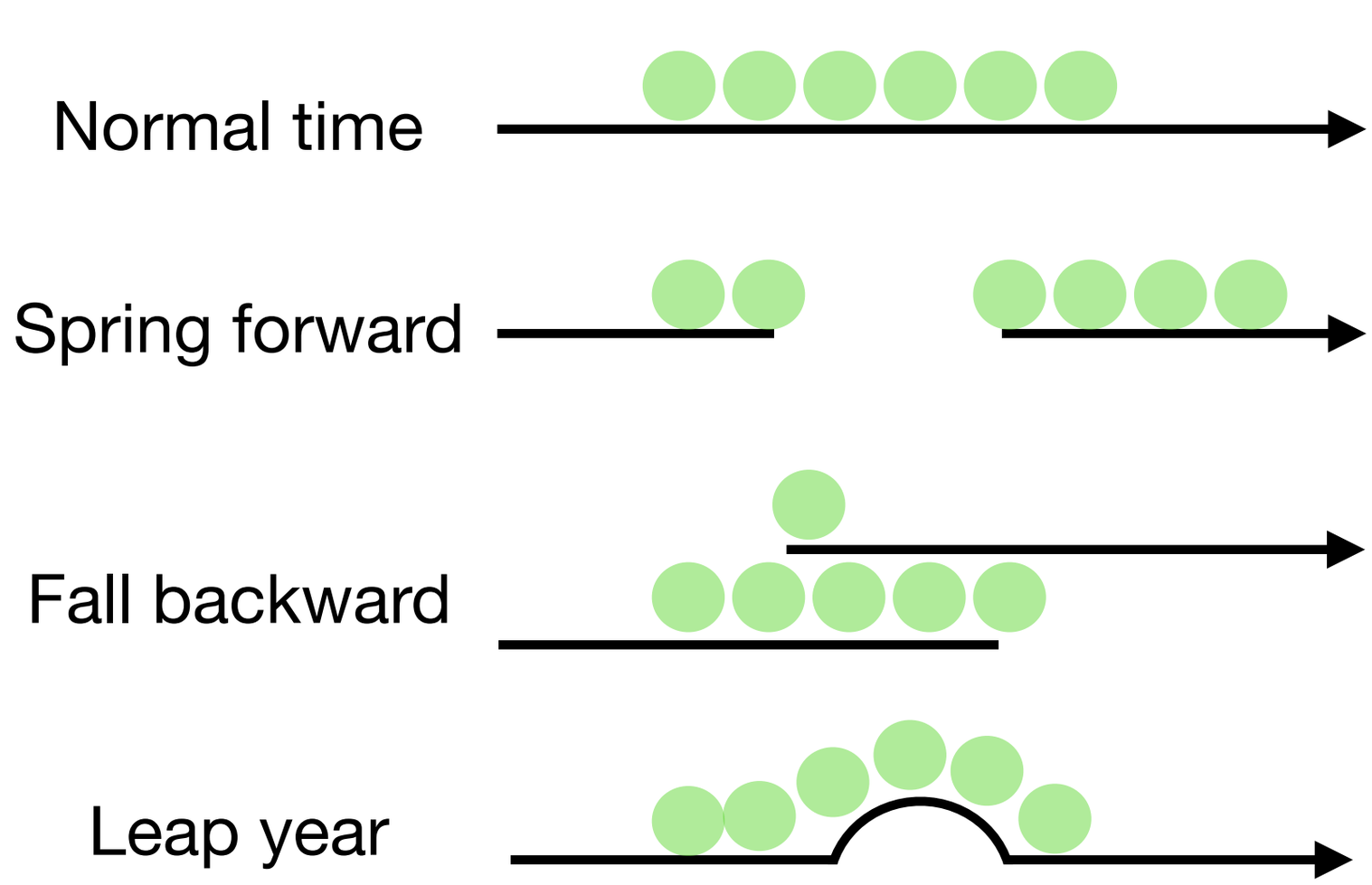
```
lubridate::ymd_hms(“2019-07-29 11:01:59”) +
        lubridate::hours(2)
```

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)
```

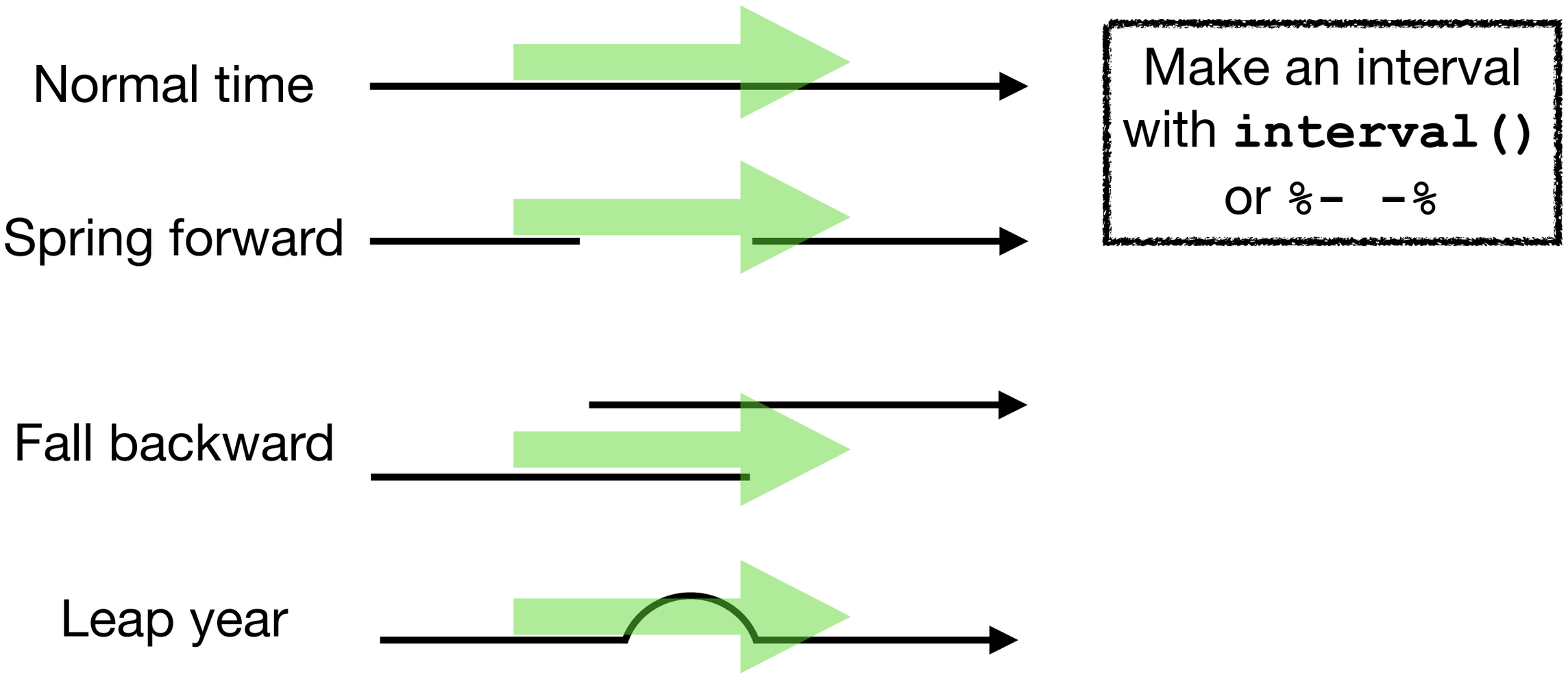
Durations are always stored as seconds

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

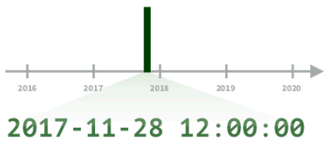
lubridate - intervals



Intervals: represent specific intervals of the timeline

Dates and times with lubridate : : CHEAT SHEET

Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

dt <- ***as_datetime****(1511870400)*
"2017-11-28 12:00:00 UTC"

2017-11-28
A **date** is a day stored as the number of days since 1970-01-01

d <- ***as_date****(17498)*
"2017-11-28"

12:00:00
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::****as.hms****(85)*
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00 — **ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*

2017-22-12 10:00:00 — **ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*

11/28/2017 1:02:03 — **mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*

1 Jan 2017 23:59:59 — **dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*

20170131 — **ymd**(), **ydm**(). *ymd(20170131)*

July 4th, 2000 — **mdy**(), **myd**(). *mdy("July 4th, 2000")*

4th of July '99 — **dmy**(), **dym**(). *dmy("4th of July '99")*

2001: Q3 — **yq**() Q for quarter. *yq("2001: Q3")*

2:01 — hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

2017.5 — **date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*

now(tzone = "") Current time in tz (defaults to system tz). *now()*

today(tzone = "") Current date in a tz (defaults to system tz). *today()*

fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*

parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.
d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.
day(d) <- 1
d ## "2017-11-01"

2018-01-31 11:59:59 — **date**(x) Date component. *date(dt)*

2018-01-31 11:59:59 — **year**(x) Year. *year(dt)*
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

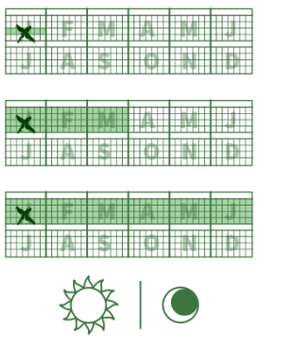
2018-01-31 11:59:59 — **month**(x, label, abbr) Month. *month(dt)*

2018-01-31 11:59:59 — **day**(x) Day of month. *day(dt)*
wday(x,label,abbr) Day of week.
qday(x) Day of quarter.

2018-01-31 11:59:59 — **hour**(x) Hour. *hour(dt)*

2018-01-31 11:59:59 — **minute**(x) Minutes. *minute(dt)*

2018-01-31 11:59:59 — **second**(x) Seconds. *second(dt)*



week(x) Week of the year. *week(dt)*
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

quarter(x, with_year = FALSE) Quarter. *quarter(dt)*

semester(x, with_year = FALSE) Semester. *semester(dt)*

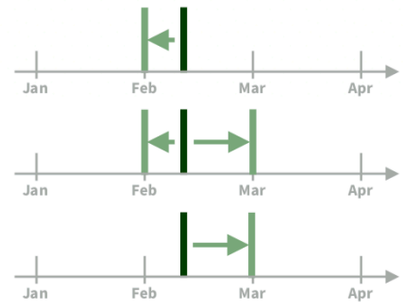
am(x) Is it in the am? *am(dt)*
pm(x) Is it in the pm? *pm(dt)*

dst(x) Is it daylight savings? *dst(d)*

leap_year(x) Is it a leap year? *leap_year(d)*

update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

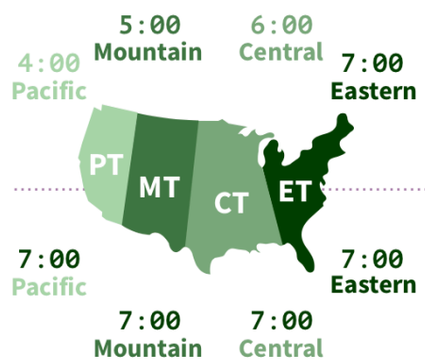
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

tidyr practice!

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

We will come back to this later to clean up the new films column…

GOAL:

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")

→

name	films
Luke Skywalker	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

CODE:

```
practice5 <- starwars %>%
    dplyr::select(name, films) %>%
    tidyr::separate_rows(films, sep = ",") %>%
    dplyr::mutate(films = stringr::str_replace_all(string = films,
                                pattern = 'c\\(|\\)|\\"',
                                replacement = ""))
```

\\ is used to escape special characters like (and “

You can combine multiple patterns using ‘`pattern1|pattern2`’

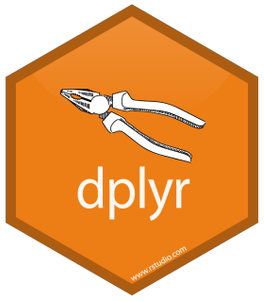
tidyr practice!

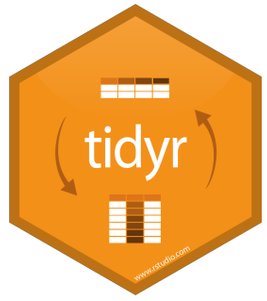
Combine hair, eye, and skin color as one “color” column (separated by “&”)

GOAL:

name	color (hair & eye & skin)
Luke Skywalker	blond & blue & fair
Darth Vader	none & yellow & white
Leia Organa	brown & brown & light
Owen Lars	brown, grey & blue & light
Beru Whitesun lars	brown & blue & light
Biggs Darklighter	black & brown & light
Obi-Wan Kenobi	auburn, white & blue-gray & fair
Anakin Skywalker	blond & blue & fair
Wilhuff Tarkin	auburn, grey & blue & fair
Chewbacca	brown & blue & unknown
Han Solo	brown & brown & fair
Wedge Antilles	brown & hazel & fair
Jek Tono Porkins	brown & blue & fair
Yoda	white & brown & green
Palpatine	grey & yellow & pale
Boba Fett	black & brown & fair
IG-88	none & red & metal
Bossk	none & red & green

First remove anyone with “NA” in any of these columns





tidyr practice!

Combine hair, eye, and skin color as one “color” column (separated by “&”)

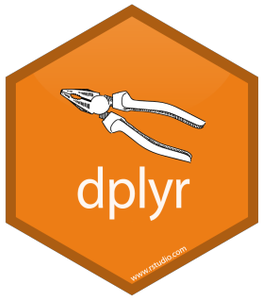
GOAL:

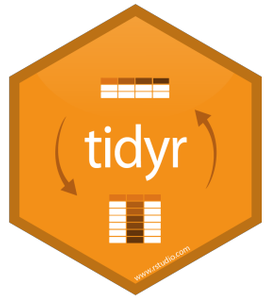
name	color (hair & eye & skin)
Luke Skywalker	blond & blue & fair
Darth Vader	none & yellow & white
Leia Organa	brown & brown & light
Owen Lars	brown, grey & blue & light
Beru Whitesun lars	brown & blue & light
IG-88	none & red & metal
Bossk	none & red & green

First remove anyone with “NA” in any of these columns

CODE:

```
practice6 <- starwars %>%
    dplyr::select(name, hair_color, eye_color, skin_color) %>%
    dplyr::filter(!is.na(hair_color),
                  !is.na(eye_color),
                  !is.na(skin_color)) %>%
    tidyr::unite(hair_color, eye_color, skin_color,
                 col = "color (hair & eye & skin)", sep = " & ")
```





tidyr practice!

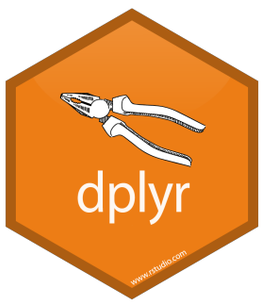
GOAL:

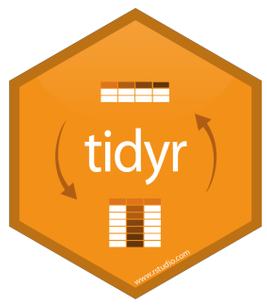
variable	Beru Whitesunlars	Cordé	Dormé	Jocasta Nu	Leia Organa	Mon Mothma	Padmé Amidala	Rey	Shmi Skywalker
birth_year	47	*NA*	*NA*	*NA*	19	48	46	*NA*	72
eye_color	blue	brown	brown	blue	brown	blue	brown	hazel	brown
gender	female	female	female	female	female	female	female	female	female
hair_color	brown	brown	brown	white	brown	auburn	brown	brown	black
height	165	157	165	167	150	150	165	*NA*	163
homeworld	Tatooine	Naboo	Naboo	Coruscant	Alderaan	Chandrila	Naboo	*NA*	Tatooine
mass	75	*NA*	*NA*	*NA*	49	*NA*	45	*NA*	*NA*
skin_color	light	light	light	fair	light	fair	light	light	fair
species	Human	Human	Human	Human	Human	Human	Human	Human	Human

Missing variables??

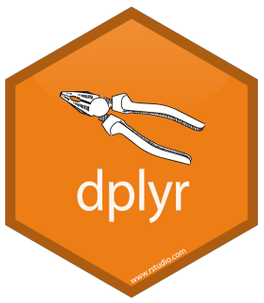
CODE:

```
practice7 <- starwars %>%
    dplyr::select(name:species) %>%
    dplyr::filter(species == "Human",
                  gender == "female") %>%
    tidyr::gather(variable, value, -name) %>%
    tidyr::spread(name, value)
```

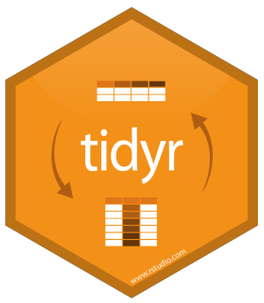




Overview of Tidyverse



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.

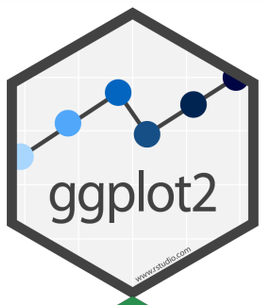


The **tidyr** package complements dplyr perfectly. It boosts the power of dplyr for data manipulation and pre-processing

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

Tidyverse