Video 20: dplyr - select, filter, mutate

Stats 102A

Miles Chen, PhD

dplyr

dplyr

`dplyr` is a core part of the tidyverse.

You can load the library with `library(dplyr)` or by loading all of the tidyverse with `library(tidyverse)`

dplyr vignette

When working with data you must:

- Figure out what you want to do.
- Describe those tasks in the form of a computer program.
- Execute the program.

The dplyr package makes these steps fast and easy:

- By constraining your options, it helps you think about your data manipulation challenges.
- It provides simple “verbs”, functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code.

dplyr vignette

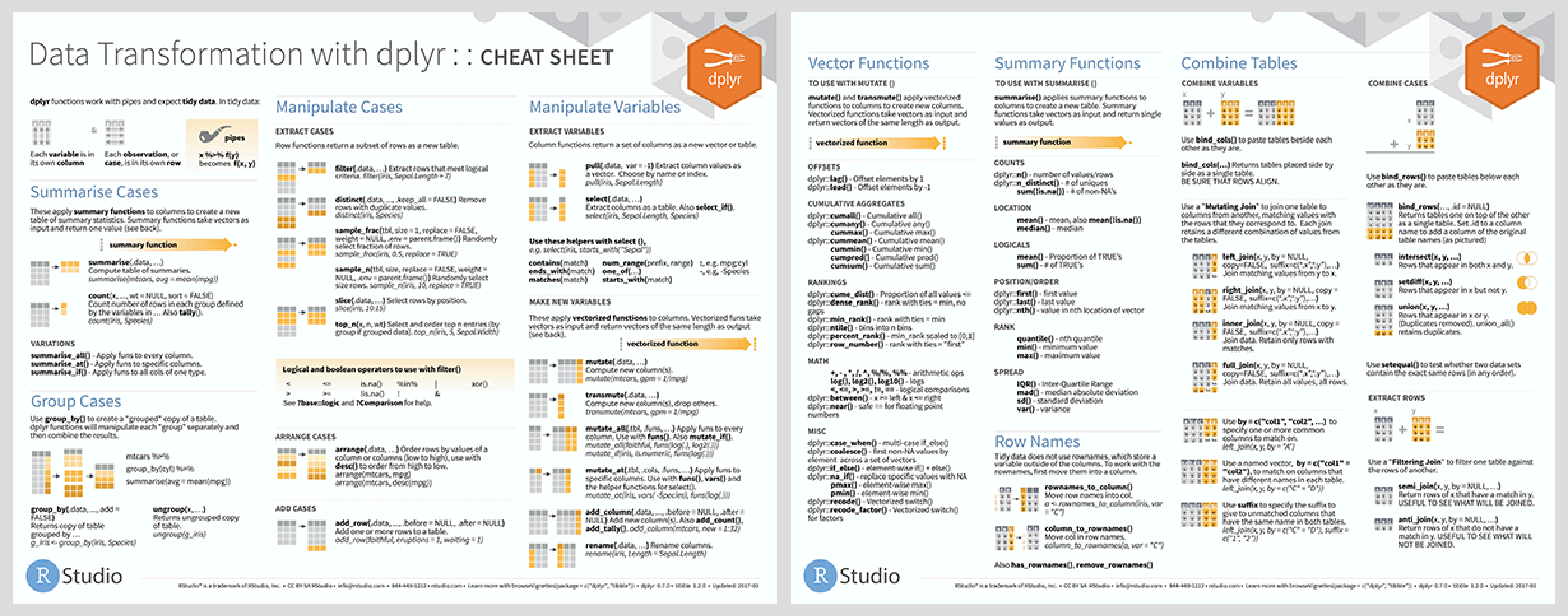
dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `mutate()` adds new variables that are functions of existing variables.
- `arrange()` changes the ordering of the rows.
- `summarise()` reduces multiple values down to a single summary.

These all combine naturally with `group_by()` which allows you to perform any operation “by group.”

the dplyr cheat sheet

https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf



Data Transformation with dplyr : : CHEAT SHEET

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

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

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

Summarise Cases

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

summary function

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

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

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

Group Cases

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

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

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

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

Manipulate Cases

EXTRACT CASES

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

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

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

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

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

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

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

Logical and boolean operators to use with filter()

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

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

ARRANGE CASES

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

ADD CASES

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

Manipulate Variables

EXTRACT VARIABLES

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

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

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

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

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

MAKE NEW VARIABLES

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

vectorized function

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

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

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

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

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

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

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

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

Summary Functions

TO USE WITH SUMMARISE ()

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

summary function

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

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

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

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

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

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

Row Names

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

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

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

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

Combine Tables

COMBINE VARIABLES

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

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

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

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

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

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

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

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

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

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

COMBINE CASES

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

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

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

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

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

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

EXTRACT ROWS

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

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

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

RStudio

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

dplyr examples

The starwars data set

The Star Wars data set is included with dplyr. It contains information about various Star Wars characters from the first 7 Star Wars movies.

```
starwars
```

```
# A tibble: 87 × 14
   name     height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue            19   male
mascu…
 2 C-3PO       167    75 <NA>       gold       yellow         112   none
mascu…
 3 R2-D2        96    32 <NA>       white, bl… red             33   none
mascu…
 4 Darth V…    202   136 none       white      yellow          41.9 male
mascu…
 5 Leia Or…    150    49 brown      light      brown           19   fema…
femin…
 6 Owen La…    178   120 brown, gr… light      blue            52   male
```

Select columns with `select()`

When using `select()`, you do not need to put quotes around the column names if there are no spaces in the names.

```
select(starwars, name, homeworld, species, films)
```

```
# A tibble: 87 × 4
   name               homeworld species films    
   <chr>              <chr>     <chr>   <list>   
 1 Luke Skywalker     Tatooine  Human   <chr [5]>
 2 C-3PO              Tatooine  Droid   <chr [6]>
 3 R2-D2              Naboo     Droid   <chr [7]>
 4 Darth Vader        Tatooine  Human   <chr [4]>
 5 Leia Organa        Alderaan  Human   <chr [5]>
 6 Owen Lars          Tatooine  Human   <chr [3]>
 7 Beru Whitesun Lars Tatooine  Human   <chr [3]>
 8 R5-D4              Tatooine  Droid   <chr [1]>
 9 Biggs Darklighter  Tatooine  Human   <chr [1]>
10 Obi-Wan Kenobi     Stewjon   Human   <chr [6]>
# ℹ 77 more rows
```

Using the pipe

The pipe `%>%` takes the result of what is in front of the pipe and inserts it as the first argument in the function that comes after the pipe. `x %>% f(y)` turns into `f(x, y)` so the result from one step is then “piped” into the next step.

```r
# select(starwars, name, homeworld, species, films) is exactly equivalent to
starwars %>%
  select(name, homeworld, species, films)
```

```
# A tibble: 87 × 4
   name               homeworld species films    
   <chr>              <chr>     <chr>   <list>   
 1 Luke Skywalker     Tatooine  Human   <chr [5]>
 2 C-3PO              Tatooine  Droid   <chr [6]>
 3 R2-D2              Naboo     Droid   <chr [7]>
 4 Darth Vader        Tatooine  Human   <chr [4]>
 5 Leia Organa        Alderaan  Human   <chr [5]>
 6 Owen Lars          Tatooine  Human   <chr [3]>
 7 Beru Whitesun Lars Tatooine  Human   <chr [3]>
 8 R5-D4              Tatooine  Droid   <chr [1]>
 9 Biggs Darklighter  Tatooine  Human   <chr [1]>
10 Obi-Wan Kenobi     Stewjon   Human   <chr [6]>
# ℹ 77 more rows
```

Native Pipe |>

R versions 4.1 and later have a native pipe. |>

The functionality is almost identical to the pipe that is part of the tidyverse %>%

More information: https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/

```r
# select(starwars, name, homeworld, species, films) is exactly equivalent to
starwars |>
  select(name, homeworld, species, films)
```

```
# A tibble: 87 × 4
   name               homeworld species films    
   <chr>              <chr>     <chr>   <list>   
 1 Luke Skywalker     Tatooine  Human   <chr [5]>
 2 C-3PO              Tatooine  Droid   <chr [6]>
 3 R2-D2              Naboo     Droid   <chr [7]>
 4 Darth Vader        Tatooine  Human   <chr [4]>
 5 Leia Organa        Alderaan  Human   <chr [5]>
 6 Owen Lars          Tatooine  Human   <chr [3]>
 7 Beru Whitesun Lars Tatooine  Human   <chr [3]>
 8 R5-D4              Tatooine  Droid   <chr [1]>
 9 Biggs Darklighter  Tatooine  Human   <chr [1]>
10 Obi-Wan Kenobi     Stewjon   Human   <chr [6]>
# ℹ 77 more rows
```

Shortcut to insert the pipe

Shortcut to insert the pipe:

CTRL(CMD) + SHIFT + M

Select columns with `select()`

- Use a negative sign to deselect columns

```r
starwars %>%
  select( -name, -eye_color, -birth_year) %>%
  head(3)
```

```
# A tibble: 3 × 11
  height  mass hair_color skin_color  sex   gender    homeworld species films
   <int> <dbl> <chr>      <chr>       <chr> <chr>     <chr>     <chr>   <list>
1    172    77 blond      fair        male  masculine Tatooine  Human   <chr>
2    167    75 <NA>       gold        none  masculine Tatooine  Droid   <chr>
3     96    32 <NA>       white, blue none  masculine Naboo     Droid   <chr>
# ℹ 2 more variables: vehicles <list>, starships <list>
```

select() example

- Use colon notation to select a range of columns

```r
starwars %>%
  select(name:eye_color) %>%
  head(3)
```

```
# A tibble: 3 × 6
  name           height  mass hair_color skin_color  eye_color
  <chr>           <int> <dbl> <chr>      <chr>       <chr>    
1 Luke Skywalker    172    77 blond      fair        blue     
2 C-3PO             167    75 <NA>       gold        yellow   
3 R2-D2              96    32 <NA>       white, blue red      
```

Special selection function

`dplyr` has special selection functions. See `?tidyselect::select_helpers`

- `contains()` Select columns that contain a character string
- `starts_with()` Select columns that start with a character string
- `ends_with()` Select columns that end with a string
- `matches()` Select columns that match a regular expression
- `everything()` Select all columns
- `num_range()` Select columns named something like x1, x2, x3, x4, x5
- `one_of(name_vector)` Select columns where the names are stored in a vector

Selection function examples

```r
starwars %>%
  select(name, ends_with("color")) %>% # selects name and cols endi
  head(3)
```

```
# A tibble: 3 × 4
  name           hair_color skin_color  eye_color
  <chr>          <chr>      <chr>       <chr>
1 Luke Skywalker blond      fair        blue
2 C-3PO          <NA>       gold        yellow
3 R2-D2          <NA>       white, blue red
```

```r
# selects name column and columns that match the regex, which says
starwars %>%
  select(name, matches("s$")) %>%
  head(3)
```

```
# A tibble: 3 × 6
  name            mass species films     vehicles  starships
  <chr>          <dbl> <chr>   <list>    <list>    <list>
1 Luke Skywalker    77 Human   <chr [5]> <chr [2]> <chr [2]>
2 C-3PO             75 Droid   <chr [6]> <chr [0]> <chr [0]>
3 R2-D2             32 Droid   <chr [7]> <chr [0]> <chr [0]>
```

Selecting with a variable

You can also select with a vector of names. To accomplish this, use the functions `all_of()` or `any_of()`

```r
vars <- c("name", "mass", "height")
starwars %>% select(all_of(vars))
```

```
# A tibble: 87 × 3
   name                mass height
   <chr>              <dbl>  <int>
 1 Luke Skywalker        77    172
 2 C-3PO                 75    167
 3 R2-D2                 32     96
 4 Darth Vader          136    202
 5 Leia Organa           49    150
 6 Owen Lars            120    178
 7 Beru Whitesun Lars    75    165
 8 R5-D4                 32     97
 9 Biggs Darklighter     84    183
10 Obi-Wan Kenobi        77    182
# ℹ 77 more rows
```

Filter rows with `filter()`

With `filter()` you specify conditions to filter the rows in the data. Filter can use any condition that can be expressed as a logical vector with length equal to the number of rows.

```r
starwars %>%
  filter(name == "R2-D2")
```

```
# A tibble: 1 × 14
  name   height  mass hair_color skin_color  eye_color birth_year sex   gender
  <chr>   <int> <dbl> <chr>      <chr>       <chr>          <dbl> <chr> <chr>
1 R2-D2      96    32 <NA>       white, blue red               33 none
masculine
# ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

filter() examples

Multiple conditions can be applied. Using the comma is equivalent to using &

```r
starwars %>%
  filter(species %in% c("Human", "Droid"), height < 175)
```

```
# A tibble: 14 × 14
   name     height  mass hair_color skin_color eye_color birth_year sex
gender
   <chr>     <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
 1 Luke Sk…    172    77 blond      fair       blue              19 male
mascu…
 2 C-3PO       167    75 <NA>       gold       yellow           112 none
mascu…
 3 R2-D2        96    32 <NA>       white, bl… red               33 none
mascu…
 4 Leia Or…    150    49 brown      light      brown             19 fema…
femin…
 5 Beru Wh…    165    75 brown      light      blue              47 fema…
femin…
 6 R5-D4        97    32 <NA>       white, red red               NA none
```

filter() is very powerful with regular expressions

We’ll learn regular expressions in the next lecture.

str_detect() returns a logical vector.

```r
starwars %>%
  filter(str_detect(name, "^F")) # the name starts with F
```

```
# A tibble: 2 × 14
  name      height  mass hair_color skin_color eye_color birth_year sex  
gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Finis Va…    170    NA blond      fair       blue              91 male 
mascu…
2 Finn          NA    NA black      dark       dark              NA male 
mascu…
# ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

The dplyr functions can be piped into each other

- use | for ’OR`

```r
starwars %>%
  filter(hair_color == "none" | eye_color == "black") %>%
  select(name, species, homeworld, hair_color, eye_color)
```

```
# A tibble: 39 × 5
   name           species      homeworld      hair_color eye_color
   <chr>          <chr>        <chr>          <chr>      <chr>
 1 Darth Vader    Human        Tatooine       none       yellow
 2 Greedo         Rodian       Rodia          <NA>       black
 3 IG-88          Droid        <NA>           none       red
 4 Bossk          Trandoshan   Trandosha      none       red
 5 Lobot          Human        Bespin         none       blue
 6 Ackbar         Mon Calamari Mon Cala       none       orange
 7 Nien Nunb      Sullustan    Sullust        none       black
 8 Nute Gunray    Neimodian    Cato Neimoidia none       red
 9 Jar Jar Binks  Gungan       Naboo          none       orange
```

Sort rows with arrange()

If you want to put things in descending order, wrap the variable name with desc()

```r
starwars %>%
  select(name, birth_year, height, mass) %>%
  arrange(desc(birth_year), mass)
```

```
# A tibble: 87 × 4
   name                  birth_year height  mass
   <chr>                      <dbl>  <int> <dbl>
 1 Yoda                         896     66    17
 2 Jabba Desilijic Tiure        600    175  1358
 3 Chewbacca                    200    228   112
 4 C-3PO                        112    167    75
 5 Dooku                        102    193    80
 6 Ki-Adi-Mundi                  92    198    82
 7 Qui-Gon Jinn                  92    193    89
 8 Finis Valorum                 91    170    NA
 9 Palpatine                     82    170    75
10 Cliegg Lars                   82    183    NA
# ℹ 77 more rows
```

Select rows based on their position with `slice()`

`slice()` lets you select rows based on their locations. The following selects rows 5 through 10

```r
starwars %>% slice(5:10)
```

```
# A tibble: 6 × 14
  name      height  mass hair_color skin_color eye_color birth_year sex
gender
  <chr>      <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Leia Org…    150    49 brown      light      brown             19 fema…
femin…
2 Owen Lars    178   120 brown, gr… light      blue              52 male
mascu…
3 Beru Whi…    165    75 brown      light      blue              47 fema…
femin…
4 R5-D4         97    32 <NA>       white, red red               NA none
mascu…
5 Biggs Da…    183    84 black      light      brown             24 male
mascu…
```

slice_sample()

slice_sample() lets you randomly select rows which can be useful to get a peek at portions of the entire tibble rather than just the head

```r
starwars %>% slice_sample(n = 5)
```

```
# A tibble: 5 × 14
  name       height  mass hair_color skin_color eye_color birth_year sex  
gender
  <chr>       <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Adi Gall…     184    50 none       dark       blue              NA fema…
femin…
2 BB8            NA    NA none       none       black             NA none 
mascu…
3 Mas Amed…     196    NA none       blue       blue              NA male 
mascu…
4 Arvel Cr…      NA    NA brown      fair       brown             NA male 
mascu…
5 Bib Fort…     180    NA none       pale       pink              NA male 
mascu…
# ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>
```

slice_min() and slice_max()

slice_min() and slice_max() lets you select rows with the lowest or highest values in a variable. It is similar to using arrange() on a single variable and then head().

```r
starwars %>% slice_max(mass, n = 3)
```

```
# A tibble: 3 × 14
  name       height  mass hair_color skin_color eye_color birth_year sex
gender
  <chr>       <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Jabba De…     175  1358 <NA>       green-tan… orange           600 herm…
mascu…
2 Grievous      216   159 none       brown, wh… green, y…         NA male
mascu…
3 IG-88         200   140 none       metal      red               15 none
mascu…
# ℹ 5 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>
```

Create new variables with mutate()

Use mutate() to create new variables based on existing variables. The new variable will be the last column, so we frequently use it with select.

```
starwars %>%
  mutate(height_in = height / 2.54) %>% head(1)
```

```
# A tibble: 1 × 15
  name       height  mass hair_color skin_color eye_color birth_year sex
gender
  <chr>       <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr>
<chr>
1 Luke Sky…     172    77 blond      fair       blue              19 male
mascu…
# ℹ 6 more variables: homeworld <chr>, species <chr>, films <list>,
#   vehicles <list>, starships <list>, height_in <dbl>
```

```
starwars %>%
  mutate(height_in = height / 2.54) %>%
  select(name, height, height_in) %>% head(1)
```

```
# A tibble: 1 × 3
  name           height height_in
```

New variables must have the same number of rows

Important: Because `mutate()` adds a new column to the data set, the variable you are creating must have the same number of values as rows in the data set.

```r
starwars %>%
  select(name, mass) %>%
  mutate(cumulative_mean = cummean(mass))
```

```
# A tibble: 87 × 3
   name                mass cumulative_mean
   <chr>              <dbl>           <dbl>
 1 Luke Skywalker        77            77  
 2 C-3PO                 75            76  
 3 R2-D2                 32            61.3
 4 Darth Vader          136            80  
 5 Leia Organa           49            73.8
 6 Owen Lars            120            81.5
 7 Beru Whitesun Lars    75            80.6
 8 R5-D4                 32            74.5
 9 Biggs Darklighter     84            75.6
10 Obi-Wan Kenobi        77            75.7
# ℹ 77 more rows
```

Some useful functions for `mutate()`

- `pmin()`, `pmax()` Element-wise min and max
- `cummin()`, `cummax()` Cumulative min and max
- `cumsum()`, `cumprod()` Cumulative sum and product
- `between()` Are values between a and b?
- `cummean()` Cumulative mean
- `lead()`, `lag()` Copy values with offset
- `ntile()` Bin vector into n buckets

mutate() examples

```r
starwars %>%
  select(name, mass, birth_year) %>%
  mutate(
    cummin_mass = cummin(mass), # cummin gives the min value seen s
    ratio = mass / mean(mass, na.rm = TRUE), # we divide mass/by th
    massyear_pmin = pmin(mass, birth_year), # pmin gives the elemen
    lag2 = lag(massyear_pmin, 2)) # lag offsets the column values
```

```
# A tibble: 87 × 7
   name                mass birth_year cummin_mass ratio massyear_pmin  lag2
   <chr>              <dbl>      <dbl>       <dbl> <dbl>         <dbl> <dbl>
 1 Luke Skywalker        77       19            77 0.791          19    NA
 2 C-3PO                 75      112            75 0.771          75    NA
 3 R2-D2                 32       33            32 0.329          32    19
 4 Darth Vader          136       41.9          32 1.40           41.9  75
 5 Leia Organa           49       19            32 0.504          19    32
 6 Owen Lars            120       52            32 1.23           52    41.9
 7 Beru Whitesun Lars    75       47            32 0.771          47    19
 8 R5-D4                 32       NA            32 0.329          NA    52
 9 Biggs Darklighter     84       24            32 0.863          24    47
10 Obi-Wan Kenobi        77       57            32 0.791          57    NA
# ℹ 77 more rows
```