Data Wrangling with dplyr

Stats 102A

Miles Chen

Department of Statistics

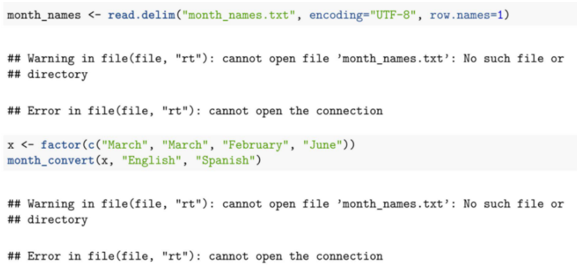
Week 4 Monday

UCLA

grumpy professor chat

It is your responsibility to read through your entire HW output file before you submit it. Make sure all requested output is visible.

I've received a few regrade requests for submissions that look like this, which unfortunately will not earn points in a regrade.



```
month_names <- read.delim("month_names.txt", encoding="UTF-8", row.names=1)
```

```
## Warning in file(file, "rt"): cannot open file 'month_names.txt': No such file or
## directory

## Error in file(file, "rt"): cannot open the connection
```

```
x <- factor(c("March", "March", "February", "June"))
month_convert(x, "English", "Spanish")
```

```
## Warning in file(file, "rt"): cannot open file 'month_names.txt': No such file or
## directory

## Error in file(file, "rt"): cannot open the connection
```

A silly mistake can make a big difference

It is not fun losing points because of a silly mistake. A silly mistake on the HW might cost you a percentage point or two in your final course grade. This is a relatively small price to pay and (in my opinion) worth the consequence if it motivates someone to pay more attention to detail.

Many job applications will say they need someone "detail oriented." This means that silly mistakes can have large consequences and they need someone who double or triple checks their own work before submission.

Imagine the consequences of:

- Submitting the wrong version of your personal statement on a school application
- CCing the wrong person on an email with confidential information
- Entering the wrong digit of a bank account number on transfer order
- An extra 0 in the dosage of a medication

I want you to achieve your dreams and potential. I don't want silly mistakes to hold you back.

Section 1

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

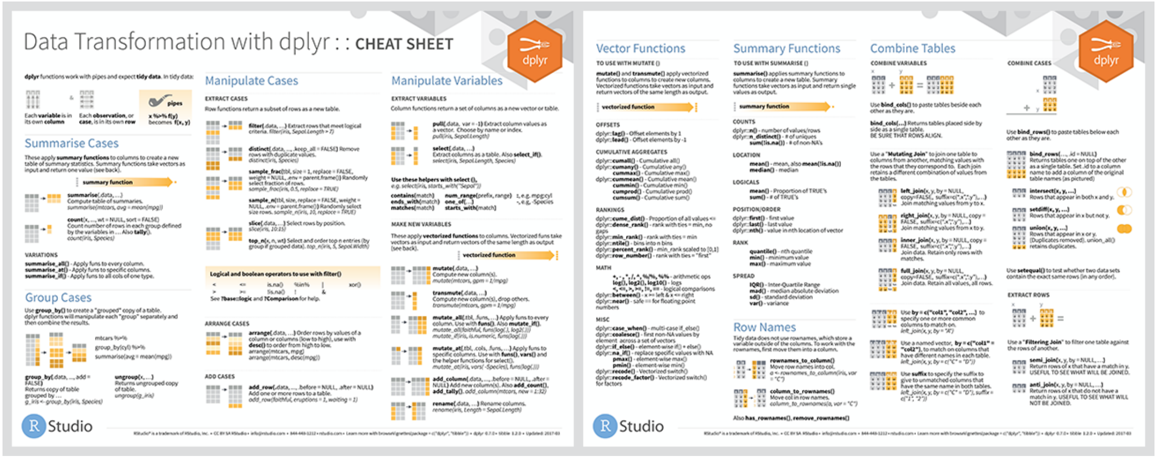
dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `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

Summarise Cases

Group Cases

Manipulate Cases

Manipulate Variables

Vector Functions

Summary Functions

Combine Tables

Row Names

RStudio

Section 2

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 x 14
##    name  height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>
##  1 Luke~    172    77 blond      fair       blue            19   male  mascu~ Tatooine  Human
##  2 C-3PO    167    75 <NA>       gold       yellow         112   none  mascu~ Tatooine  Droid
##  3 R2-D2     96    32 <NA>       white, bl~ red             33   none  mascu~ Naboo     Droid
##  4 Dart~    202   136 none       white      yellow          41.9 male  mascu~ Tatooine  Human
##  5 Leia~    150    49 brown      light      brown           19   fema~ femin~ Alderaan  Human
##  6 Owen~    178   120 brown, gr~ light      blue            52   male  mascu~ Tatooine  Human
##  7 Beru~    165    75 brown      light      blue            47   fema~ femin~ Tatooine  Human
##  8 R5-D4     97    32 <NA>       white, red red             NA   none  mascu~ Tatooine  Droid
##  9 Bigg~    183    84 black      light      brown           24   male  mascu~ Tatooine  Human
## 10 Obi-~    182    77 auburn, w~ fair       blue-gray       57   male  mascu~ Stewjon   Human
## # ... with 77 more rows, and 3 more variables: films <list>, vehicles <list>, starships <list>
```

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.

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

```
## # A tibble: 87 x 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]>
## # ... with 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 x 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]>
## # ... with 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

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

```
## # A tibble: 3 x 11
##   height  mass hair_color skin_color  sex   gender    homeworld species films     vehicles starships
##    <int> <dbl> <chr>      <chr>       <chr> <chr>     <chr>     <chr>   <list>    <list>   <list>
## 1    172    77 blond      fair        male  masculine Tatooine  Human   <chr [5]> <chr [2~ <chr [2]>
## 2    167    75 <NA>       gold        none  masculine Tatooine  Droid   <chr [6]> <chr [0~ <chr [0]>
## 3     96    32 <NA>       white, blue none  masculine Naboo     Droid   <chr [7]> <chr [0~ <chr [0]>
```

select() example

- Use colon notation to select a range of columns

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

```
## # A tibble: 3 x 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 columns ending with color
  head(3)
```

```
## # A tibble: 3 x 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 ends with "s"
starwars %>%
  select(name, matches("s$")) %>%
  head(3)
```

```
## # A tibble: 3 x 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()`

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

```
## # A tibble: 87 x 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
## # ... with 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.

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

```
## # A tibble: 1 x 14
##   name   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 R2-D2      96    32 <NA>       white, bl~ red               33 none  mascu~ Naboo     Droid   <chr~
## # ... with 2 more variables: 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: 16 x 14
##    name  height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species
##    <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>
##  1 Luke~    172    77 blond      fair       blue              19 male  mascu~ Tatooine  Human
##  2 C-3PO    167    75 <NA>       gold       yellow           112 none  mascu~ Tatooine  Droid
##  3 R2-D2     96    32 <NA>       white, bl~ red               33 none  mascu~ Naboo     Droid
##  4 Leia~    150    49 brown      light      brown             19 fema~ femin~ Alderaan  Human
##  5 Beru~    165    75 brown      light      blue              47 fema~ femin~ Tatooine  Human
##  6 R5-D4     97    32 <NA>       white, red red               NA none  mascu~ Tatooine  Droid
##  7 Wedg~    170    77 brown      fair       hazel             21 male  mascu~ Corellia  Human
##  8 Palp~    170    75 grey       pale       yellow            82 male  mascu~ Naboo     Human
##  9 Mon ~    150    NA auburn     fair       blue              48 fema~ femin~ Chandrila Human
## 10 Fini~    170    NA blond      fair       blue              91 male  mascu~ Coruscant Human
## 11 Shmi~    163    NA black      fair       brown             72 fema~ femin~ Tatooine  Human
## 12 Cordé    157    NA brown      light      brown             NA fema~ femin~ Naboo     Human
## 13 Dormé    165    NA brown      light      brown             NA fema~ femin~ Naboo     Human
## 14 Joca~    167    NA white      fair       blue              NA fema~ femin~ Coruscant Human
## 15 R4-P~     96    NA none       silver, r~ red, blue         NA none  femin~ <NA>      Droid
```

filter() is very powerful with regular expressions

We'll learn regular expressions in the next lecture. `str_detect()` returns a logical vector.

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

```
## # A tibble: 2 x 14
##   name  height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 Fini~    170    NA blond      fair       blue              91 male  mascu~ Coruscant Human   <chr~
## 2 Finn      NA    NA black      dark       dark              NA male  mascu~ <NA>      Human   <chr~
## # ... with 2 more variables: vehicles <list>, starships <list>
```

The dplyr functions can be piped into each other

- use | for 'OR'

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

```
## # A tibble: 38 x 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
## 10 Roos Tarpals   Gungan       Naboo          none       orange
## # ... with 28 more rows
```

Sort rows with arrange()

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

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

```
## # A tibble: 87 x 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
## # ... with 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 x 14
##   name   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 Leia~     150    49 brown      light      brown             19 fema~ femin~ Alderaan  Human   <chr~
## 2 Owen~     178   120 brown, gr~ light      blue              52 male  mascu~ Tatooine  Human   <chr~
## 3 Beru~     165    75 brown      light      blue              47 fema~ femin~ Tatooine  Human   <chr~
## 4 R5-D4      97    32 <NA>       white, red red               NA none  mascu~ Tatooine  Droid   <chr~
## 5 Bigg~     183    84 black      light      brown             24 male  mascu~ Tatooine  Human   <chr~
## 6 Obi-~     182    77 auburn, w~ fair       blue-gray         57 male  mascu~ Stewjon   Human   <chr~
## # ... with 2 more variables: vehicles <list>, starships <list>
```

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

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

```
## # A tibble: 5 x 14
##   name  height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 San ~    191    NA none       grey       gold              NA male  mascu~ Muunilin~ Muun    <chr~
## 2 Dart~    175    80 none       red        yellow            54 male  mascu~ Dathomir  Zabrak  <chr~
## 3 Ackb~    180    83 none       brown mot~ orange            41 male  mascu~ Mon Cala  Mon Ca~ <chr~
## 4 Luke~    172    77 blond      fair       blue              19 male  mascu~ Tatooine  Human   <chr~
## 5 Zam ~    168    55 blonde     fair, gre~ yellow            NA fema~ femin~ Zolan     Clawdi~ <chr~
## # ... with 2 more variables: vehicles <list>, starships <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()`.

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

```
## # A tibble: 3 x 14
##   name  height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>  <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 Jabb~    175  1358 <NA>       green-tan~ orange           600 herm~ mascu~ Nal Hutta Hutt    <chr~
## 2 Grie~    216   159 none       brown, wh~ green, y~         NA male  mascu~ Kalee     Kaleesh <chr~
## 3 IG-88    200   140 none       metal      red               15 none  mascu~ <NA>      Droid   <chr~
## # ... with 2 more variables: 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 x 15
##   name   height  mass hair_color skin_color eye_color birth_year sex   gender homeworld species films
##   <chr>   <int> <dbl> <chr>      <chr>      <chr>          <dbl> <chr> <chr>  <chr>     <chr>   <lis>
## 1 Luke~     172    77 blond      fair       blue              19 male  mascu~ Tatooine  Human   <chr~
## # ... with 3 more variables: vehicles <list>, starships <list>, height_in <dbl>
```

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

```
## # A tibble: 1 x 3
##   name           height height_in
##   <chr>           <int>     <dbl>
## 1 Luke Skywalker    172      67.7
```

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.

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

```
## # A tibble: 87 x 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
## # ... with 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 so far
    ratio = mass / mean(mass, na.rm = TRUE), # we divide mass/by the col mean
    massyear_pmin = pmin(mass, birth_year), # pmin gives the element-wise min
    lag2 = lag(massyear_pmin, 2)) # lag offsets the column values
```

```
## # A tibble: 87 x 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
## # ... with 77 more rows
```

Summarize with summarise()

Hadley is from New Zealand where they spell it with an s. He later added summarize() to have the same functionality, but I'm accustomed to using the original function.

Summary functions take multiple values and summarize them with a single value. For example, mean() and var() are summary functions.

```r
starwars %>%
  select(height, mass) %>%
  summarise(
    avg_height = mean(height, na.rm = TRUE),
    var_height = var(height, na.rm = TRUE),
    avg_mass = mean(mass, na.rm = TRUE),
    min_height = min(height, na.rm = TRUE),
    max_mass = max(mass, na.rm = TRUE),
    count = n())
```

```
## # A tibble: 1 x 6
##   avg_height var_height avg_mass min_height max_mass count
##        <dbl>      <dbl>    <dbl>      <int>    <dbl> <int>
## 1       174.      1209.     97.3         66     1358    87
```

Create groups using group_by()

We can create groups using the group_by() function.

```
starwars %>%
  group_by(species) %>%
  select(name, height, mass, species)
```

```
## # A tibble: 87 x 4
## # Groups:   species [38]
##    name               height  mass species
##    <chr>               <int> <dbl> <chr>
##  1 Luke Skywalker        172    77 Human
##  2 C-3PO                 167    75 Droid
##  3 R2-D2                  96    32 Droid
##  4 Darth Vader           202   136 Human
##  5 Leia Organa           150    49 Human
##  6 Owen Lars             178   120 Human
##  7 Beru Whitesun lars    165    75 Human
##  8 R5-D4                  97    32 Droid
##  9 Biggs Darklighter     183    84 Human
## 10 Obi-Wan Kenobi        182    77 Human
## # ... with 77 more rows
```

group_by() + summarise()

The power of group_by() is realized when combined with summarise()

```r
starwars %>%
  group_by(species) %>%
  select(name, height, mass, species) %>%
  summarise(
    mean_ht = mean(height, na.rm = TRUE),
    sd_ht = sd(height, na.rm = TRUE),
    mean_mass = mean(mass, na.rm = TRUE),
    sd_mass = sd(mass, na.rm = TRUE),
    count = n())
```

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)

## # A tibble: 38 x 6
##    species   mean_ht sd_ht mean_mass sd_mass count
##    <chr>       <dbl> <dbl>     <dbl>   <dbl> <int>
##  1 Aleena        79   NA        15      NA       1
##  2 Besalisk     198   NA       102      NA       1
##  3 Cerean       198   NA        82      NA       1
##  4 Chagrian     196   NA       NaN      NA       1
##  5 Clawdite     168   NA        55      NA       1
##  6 Droid        131.  49.1      69.8    51.0     6
##  7 Dug          112   NA        40      NA       1
```

group_by() + summarise()

```r
starwars %>%
  group_by(species) %>%
  select(name, height, mass, species) %>%
  summarise(
    mean_ht = mean(height, na.rm = TRUE),
    sd_ht = sd(height, na.rm = TRUE),
    mean_mass = mean(mass, na.rm = TRUE),
    sd_mass = sd(mass, na.rm = TRUE),
    count = n()
    ) %>%
  filter(count > 1) %>%
  arrange(desc(count)) %>%
  head()
```

```
## ‘summarise()‘ ungrouping output (override with ‘.groups‘ argument)

## # A tibble: 6 x 6
##   species  mean_ht sd_ht mean_mass sd_mass count
##   <chr>      <dbl> <dbl>     <dbl>   <dbl> <int>
## 1 Human       177. 12.5       82.8    19.4    35
## 2 Droid       131. 49.1       69.8    51.0     6
## 3 <NA>        181.  2.89      48      NA       4
## 4 Gungan      209. 14.2       74      11.3     3
```

group_by() + mutate()

Note that C-3PO is above average when compared to other droids in the data set, but below average when compared to all characters in the data set.

```
starwars %>%
  filter(species %in% c("Human","Droid") |
       is.na(species)) %>%
  select(name, species, height) %>%
  group_by(species) %>%
  mutate(z_height = (height - mean(height, na.rm = TRUE))/sd(height, na.rm = TRUE)) %>%
  head()
```

```
## # A tibble: 6 x 4
## # Groups:   species [2]
##   name           species height z_height
##   <chr>          <chr>    <int>    <dbl>
## 1 Luke Skywalker Human      172   -0.371
## 2 C-3PO          Droid      167    0.728
## 3 R2-D2          Droid       96   -0.716
## 4 Darth Vader    Human      202    2.02
## 5 Leia Organa    Human      150   -2.13
## 6 Owen Lars      Human      178    0.108
```

Without group_by()

Note that C-3PO is above average when compared to other droids in the data set, but below average when compared to all characters in the data set.

```r
starwars %>%
  filter(species %in% c("Human","Droid") |
      is.na(species)) %>%
  select(name, species, height) %>%
  # group_by(species) %>%
  mutate(z_height = (height - mean(height, na.rm = TRUE))/sd(height, na.rm = TRUE)) %>%
  head()
```

```
## # A tibble: 6 x 4
##   name           species height z_height
##   <chr>          <chr>    <int>    <dbl>
## 1 Luke Skywalker Human      172   0.0329
## 2 C-3PO          Droid      167  -0.168
## 3 R2-D2          Droid       96  -3.02
## 4 Darth Vader    Human      202   1.24
## 5 Leia Organa    Human      150  -0.849
## 6 Owen Lars      Human      178   0.274
```

Multiple group_by() on some toy data

```r
toy_cases <- read_csv("https://raw.githubusercontent.com/rstudio/EDAWR/master/data-raw/toyb.csv")
print(toy_cases)
```

```
## # A tibble: 12 x 4
##    country      year sex    cases
##    <chr>       <dbl> <chr>  <dbl>
##  1 Afghanistan  1999 female     1
##  2 Afghanistan  1999 male       1
##  3 Afghanistan  2000 female     1
##  4 Afghanistan  2000 male       1
##  5 Brazil       1999 female     2
##  6 Brazil       1999 male       2
##  7 Brazil       2000 female     2
##  8 Brazil       2000 male       2
##  9 China        1999 female     3
## 10 China        1999 male       3
## 11 China        2000 female     3
## 12 China        2000 male       3
```

Multiple group_by() + summarise()

We can provide group_by() two variables and it will create a hierarchy of groups

```
summary1 <- toy_cases %>% group_by(country, year) %>%
  summarise(cases = sum(cases))
```

```
## ‘summarise()‘ regrouping output by ’country’ (override with ‘.groups‘ argument)
```

```
print(summary1)
```

```
## # A tibble: 6 x 3
## # Groups:   country [3]
##   country      year cases
##   <chr>       <dbl> <dbl>
## 1 Afghanistan  1999     2
## 2 Afghanistan  2000     2
## 3 Brazil       1999     4
## 4 Brazil       2000     4
## 5 China        1999     6
## 6 China        2000     6
```

Multiple group_by() + summarise()

```
summary2 <- summary1 %>% summarise(cases = sum(cases))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
summary2
```

```
## # A tibble: 3 x 2
##   country     cases
##   <chr>       <dbl>
## 1 Afghanistan     4
## 2 Brazil          8
## 3 China          12
```

```
summary3 <- summary2 %>% summarise(cases = sum(cases))
summary3
```

```
## # A tibble: 1 x 1
##   cases
##   <dbl>
## 1    24
```

Chanign the order of the multiple group_by()

```
summary_a <- toy_cases %>% group_by(year, country) %>%
  summarise(cases = sum(cases))
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
print(summary_a)
```

```
## # A tibble: 6 x 3
## # Groups:   year [2]
##    year country     cases
##   <dbl> <chr>       <dbl>
## 1  1999 Afghanistan     2
## 2  1999 Brazil          4
## 3  1999 China           6
## 4  2000 Afghanistan     2
## 5  2000 Brazil          4
## 6  2000 China           6
```

Multiple group_by() + summarise()

```
summary_b <- summary_a %>% summarise(cases = sum(cases))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
summary_b
```

```
## # A tibble: 2 x 2
##    year cases
##   <dbl> <dbl>
## 1  1999    12
## 2  2000    12
```

```
summary_c <- summary_b %>% summarise(cases = sum(cases))
summary_c
```

```
## # A tibble: 1 x 1
##   cases
##   <dbl>
## 1    24
```

Section 3

Two-table verbs

Two-table verbs

dplyr also comes with some two-table verbs that allow you to combine tables.

Mutating joins add new variables to one table from matching rows in another

Not covered here, but you can read more at https://dplyr.tidyverse.org/articles/two-table.html

- **Filtering joins** filter observations from one table based on whether or not they match an observation in the other.
- **Set operations**, which combine the observations in the data sets as if they were set elements.

Toy tables

```
people <- tibble(
  name = c("Adam", "Betty", "Carl", "Doug"),
  state = c("CA", "CA", "NY", "TX")
  )
states <- tibble(
  abbreviation = c("CA", "NY", "WA"),
  state_name = c("California", "New York", "Washington")
  )
```

left_join()

left_join() takes all the values in the left table and adds variables from the right table by matching values using a column that exists in both tables. Values that do not exist in the other table have NA returned.

```
people %>% left_join(states, by = c("state" = "abbreviation"))
```

```
## # A tibble: 4 x 3
##   name  state state_name
##   <chr> <chr> <chr>
## 1 Adam  CA    California
## 2 Betty CA    California
## 3 Carl  NY    New York
## 4 Doug  TX    <NA>
```

right_join()

`right_join()` is similar to `left_join` except it keeps all the rows in the right table.

```r
people %>% right_join(states, by = c("state" = "abbreviation"))
```

```
## # A tibble: 4 x 3
##   name  state state_name
##   <chr> <chr> <chr>
## 1 Adam  CA    California
## 2 Betty CA    California
## 3 Carl  NY    New York
## 4 <NA>  WA    Washington
```

inner_join()

`inner_join()` keeps only rows that have values that exist in both tables. You can think of this as the intersection.

```
people %>% inner_join(states, by = c("state" = "abbreviation"))
```

```
## # A tibble: 3 x 3
##   name  state state_name
##   <chr> <chr> <chr>
## 1 Adam  CA    California
## 2 Betty CA    California
## 3 Carl  NY    New York
```

full_join()

full_join() keeps all rows from both tables. You can think of this as the union. (in SQL this is called a full outer join)

```r
people %>% full_join(states, by = c("state" = "abbreviation"))
```

```
## # A tibble: 5 x 3
##   name  state state_name
##   <chr> <chr> <chr>
## 1 Adam  CA    California
## 2 Betty CA    California
## 3 Carl  NY    New York
## 4 Doug  TX    <NA>
## 5 <NA>  WA    Washington
```

Controlling how the tables are matched

Depending on the tables, the join operation can match tables on different variables.

In the previous examples, we used a named character vector `by = c("state" = "abbreviation")` specifying the name in the left table that matches the name in the right table.

Options for joining tables

- `by = NULL` (or don't specify anything): `dplyr` will use all variables that have the same name in both tables.
- `by = "x"`: dplyr will use only some of the variables that have the same name in both tables
- `by = c("x" = "y")`: this is the form that must be used if the matching columns do not have the same name in both tables.