Data Wrangling with dplyr Part 1

Week 3 Friday

Miles Chen

Department of Statistics and Data Science

Section 1

dplyr

dplyr

`dplyr` is a powerful package for data manipulation and a core component of the tidyverse.

You can load `dplyr` specifically with:

```
library(dplyr)
```

Or load the entire tidyverse, which includes dplyr and other related packages:

```
library(tidyverse)
```

dplyr

dplyr is a package in the tidyverse that provides a consistent set of functions that help you solve data manipulation challenges.

cheat sheet

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

The starwars Dataset

The `starwars` dataset is included with `dplyr`. It contains information about various Star Wars characters from the first seven Star Wars movies. This dataset is structured as a tibble (a kind of data frame).

```r
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 Skyw~    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 Darth Vad~    202   136 none       white      yellow          41.9 male  mascu~ Tatooine  Human
 5 Leia Orga~    150    49 brown      light      brown           19   fema~ femin~ Alderaan  Human
 6 Owen Lars     178   120 brown, gr~ light      blue            52   male  mascu~ Tatooine  Human
 7 Beru Whit~    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 Biggs Dar~    183    84 black      light      brown           24   male  mascu~ Tatooine  Human
10 Obi-Wan K~    182    77 auburn, w~ fair       blue-gray       57   male  mascu~ Stewjon   Human
# i 77 more rows
# i 3 more variables: films <list>, vehicles <list>, starships <list>
```

glimpse()

The `glimpse()` function (part of dplyr) provides a structured overview of a dataset. It displays column names, data types, and a preview of values. Similar to `str()`.

```r
glimpse(starwars)
```

```
Rows: 87
Columns: 14
$ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Organa", "Owen Lars", ~
$ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 228, 180, 173, 175, 1~
$ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.0, 84.0, NA, 112.0, ~
$ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", NA, "black", "auburn,~
$ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "light", "white, red"~
$ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue", "red", "brown", "bl~
$ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, 41.9, 64.0, 200.0, 2~
$ sex        <chr> "male", "none", "none", "male", "female", "male", "female", "none", "male", "ma~
$ gender     <chr> "masculine", "masculine", "masculine", "masculine", "feminine", "masculine", "f~
$ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "Tatooine", "Tatooine"~
$ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Human", "Droid", "Human"~
$ films      <list> <"A New Hope", "The Empire Strikes Back", "Return of the Jedi", "Revenge of th~
$ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imperial Speeder Bike",~
$ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1", <>, <>, <>, <>, "X-~
```

The nycflights23 Dataset

Contains information from flights in New York City in the year 2023.

```r
library(nycflights23)
data(flights)
```

```r
glimpse(flights)
```

```
Rows: 435,352
Columns: 19
$ year           <int> 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 202~
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
$ dep_time       <int> 1, 18, 31, 33, 36, 503, 520, 524, 537, 547, 549, 551, 552, 554, 554, 558, 6~
$ sched_dep_time <int> 2038, 2300, 2344, 2140, 2048, 500, 510, 530, 520, 545, 559, 600, 559, 600, ~
$ dep_delay      <dbl> 203, 78, 47, 173, 228, 3, 10, -6, 17, 2, -10, -9, -7, -6, -6, -7, 0, 0, 0, ~
$ arr_time       <int> 328, 228, 500, 238, 223, 808, 948, 645, 926, 845, 905, 846, 857, 914, 725, ~
$ sched_arr_time <int> 3, 135, 426, 2352, 2252, 815, 949, 710, 818, 852, 901, 859, 911, 920, 735, ~
$ arr_delay      <dbl> 205, 53, 34, 166, 211, -7, -1, -25, 68, -7, 4, -13, -14, -6, -10, -31, -23,~
$ carrier        <chr> "UA", "DL", "B6", "B6", "UA", "AA", "B6", "AA", "UA", "NK", "B6", "B6", "AA~
$ flight         <int> 628, 393, 371, 1053, 219, 499, 996, 981, 206, 225, 800, 93, 518, 165, 445, ~
$ tailnum        <chr> "N25201", "N830DN", "N807JB", "N265JB", "N17730", "N925AN", "N2043J", "N918~
$ origin         <chr> "EWR", "JFK", "JFK", "JFK", "EWR", "EWR", "JFK", "EWR", "EWR", "EWR", "JFK"~
$ dest           <chr> "SMF", "ATL", "BQN", "CHS", "DTW", "MIA", "BQN", "ORD", "IAH", "FLL", "PBI"~
$ air_time       <dbl> 367, 108, 190, 108, 80, 154, 192, 119, 258, 157, 164, 143, 159, 169, 116, 1~
$ distance       <dbl> 2500, 760, 1576, 636, 488, 1085, 1576, 719, 1400, 1065, 1028, 950, 1096, 10~
$ hour           <dbl> 20, 23, 23, 21, 20, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6~
$ minute         <dbl> 38, 0, 44, 40, 48, 0, 10, 30, 20, 45, 59, 0, 59, 0, 0, 5, 0, 0, 0, 5, 0, 5,~
$ time_hour      <dttm> 2023-01-01 20:00:00, 2023-01-01 23:00:00, 2023-01-01 23:00:00, 2023-01-01 ~
```

```r
flights
```

```
# A tibble: 435,352 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
 1  2023     1     1        1           2038       203      328              3       205 UA
 2  2023     1     1       18           2300        78      228            135        53 DL
 3  2023     1     1       31           2344        47      500            426        34 B6
 4  2023     1     1       33           2140       173      238           2352       166 B6
 5  2023     1     1       36           2048       228      223           2252       211 UA
 6  2023     1     1      503            500         3      808            815        -7 AA
 7  2023     1     1      520            510        10      948            949        -1 B6
 8  2023     1     1      524            530        -6      645            710       -25 AA
 9  2023     1     1      537            520        17      926            818        68 UA
10  2023     1     1      547            545         2      845            852        -7 NK
# i 435,342 more rows
# i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

The Pipe

R has a pipe operator |> (older versions may appear as %>%)

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)` is equivalent to `f(x, y)`.

The easiest way to pronounce the pipe is “then”.

The Pipe

Instead of writing:

```
temp_data1 <- func1(original_data, options1)
temp_data2 <- func2(temp_data1, options2)
final_data <- func3(temp_data2, options3)
```

We can string them together with pipes:

```
final_data <- original_data |>
    func1(options1) |>
    func2(options2) |>
    func3(options3)
```

Shortcut to insert the pipe

Use this keyboard shortcut to quickly insert the pipe operator in RStudio:

Windows/Linux: `CTRL + SHIFT + M`
Mac: `CMD + SHIFT + M`

Depending on your settings, RStudio may insert the pipe function (%>%) from the package `magrittr`, which works very similarly.

Section 2

dplyr basics

dplyr basics

The functions in dplyr each perform a different task and all have the following in common:

- The first argument is always a data frame.
- The subsequent arguments usually describe which columns to operate on.
- The output is always a new data frame.

Each function does one thing, so many data wrangling problems will require combining multiple functions. We accomplish with the pipe.

Selecting Columns with select()

The select() function lets you choose columns from a dataset. When using select(), you do not need to use quotes around column names unless they contain spaces or special characters.

```
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]>
# i 77 more rows
```

Selecting Columns with select()

- Use a negative sign (−) to **deselect** columns.

```r
starwars |>
  select(-eye_color, -birth_year) |>
  print(n = 6)
```

```
# A tibble: 87 x 12
  name    height  mass hair_color skin_color sex   gender homeworld species films vehicles starships
  <chr>    <int> <dbl> <chr>      <chr>      <chr> <chr>  <chr>     <chr>   <lis> <list>   <list>
1 Luke S~    172    77 blond      fair       male  mascu~ Tatooine  Human   <chr> <chr>    <chr [2]>
2 C-3PO      167    75 <NA>       gold       none  mascu~ Tatooine  Droid   <chr> <chr>    <chr [0]>
3 R2-D2       96    32 <NA>       white, bl~ none  mascu~ Naboo     Droid   <chr> <chr>    <chr [0]>
4 Darth ~    202   136 none       white      male  mascu~ Tatooine  Human   <chr> <chr>    <chr [1]>
5 Leia O~    150    49 brown      light      fema~ femin~ Alderaan  Human   <chr> <chr>    <chr [0]>
6 Owen L~    178   120 brown, gr~ light      male  mascu~ Tatooine  Human   <chr> <chr>    <chr [0]>
# i 81 more rows
```

This removes the name, eye_color, and birth_year columns while keeping all others.

Selecting a Range of Columns

- Use **colon notation (:)** to select a continuous range of columns.

```
starwars |>
  select(name:eye_color) |>
  print(n = 6)
```

```
# A tibble: 87 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
4 Darth Vader       202   136 none        white       yellow
5 Leia Organa       150    49 brown       light       brown
6 Owen Lars         178   120 brown, grey light       blue
# i 81 more rows
```

This keeps only the columns from `name` through `eye_color`, in the order they appear in the dataset.

Special Selection Functions in dplyr

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

- starts_with("x"), ends_with("x"), contains("x") - picks columns that match a string
- matches(regex) picks columns that match a regular expression
- any_of(c("colA", "colB")) ignores missing names
- everything() Select all (remaining) columns
- num_range("x", 1:5) Select columns named something like x1, x2, x3, x4, x5
- one_of(name_vector) Select columns where the names are stored in a vector

Example of Special Selection Functions

Select `name` and all columns that end with `"color"`:

```
flights |>
  select(contains("time"))
```

```
# A tibble: 435,352 x 6
   dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
      <int>          <int>    <int>          <int>    <dbl> <dttm>
 1        1           2038      328              3      367 2023-01-01 20:00:00
 2       18           2300      228            135      108 2023-01-01 23:00:00
 3       31           2344      500            426      190 2023-01-01 23:00:00
 4       33           2140      238           2352      108 2023-01-01 21:00:00
 5       36           2048      223           2252       80 2023-01-01 20:00:00
 6      503            500      808            815      154 2023-01-01 05:00:00
 7      520            510      948            949      192 2023-01-01 05:00:00
 8      524            530      645            710      119 2023-01-01 05:00:00
 9      537            520      926            818      258 2023-01-01 05:00:00
10      547            545      845            852      157 2023-01-01 05:00:00
# i 435,342 more rows
```

Examples of Special Selection Functions

```
flights |>
  select(starts_with("dep"), starts_with("arr"), everything())
```

```
# A tibble: 435,352 x 19
   dep_time dep_delay arr_time arr_delay  year month   day sched_dep_time sched_arr_time carrier
      <int>     <dbl>    <int>     <dbl> <int> <int> <int>          <int>          <int> <chr>
 1        1       203      328       205  2023     1     1           2038              3 UA
 2       18        78      228        53  2023     1     1           2300            135 DL
 3       31        47      500        34  2023     1     1           2344            426 B6
 4       33       173      238       166  2023     1     1           2140           2352 B6
 5       36       228      223       211  2023     1     1           2048           2252 UA
 6      503         3      808        -7  2023     1     1            500            815 AA
 7      520        10      948        -1  2023     1     1            510            949 B6
 8      524        -6      645       -25  2023     1     1            530            710 AA
 9      537        17      926        68  2023     1     1            520            818 UA
10      547         2      845        -7  2023     1     1            545            852 NK
# i 435,342 more rows
# i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Rename while selecting

Rename while selecting (new = old):

```r
flights |>
  select(dep = dep_time, arr = arr_time, everything())
```

```
# A tibble: 435,352 x 19
     dep   arr  year month   day sched_dep_time dep_delay sched_arr_time arr_delay carrier flight
   <int> <int> <int> <int> <int>          <int>     <dbl>          <int>     <dbl> <chr>    <int>
 1     1   328  2023     1     1           2038       203              3       205 UA         628
 2    18   228  2023     1     1           2300        78            135        53 DL         393
 3    31   500  2023     1     1           2344        47            426        34 B6         371
 4    33   238  2023     1     1           2140       173           2352       166 B6        1053
 5    36   223  2023     1     1           2048       228           2252       211 UA         219
 6   503   808  2023     1     1            500         3            815        -7 AA         499
 7   520   948  2023     1     1            510        10            949        -1 B6         996
 8   524   645  2023     1     1            530        -6            710       -25 AA         981
 9   537   926  2023     1     1            520        17            818        68 UA         206
10   547   845  2023     1     1            545         2            852        -7 NK         225
# i 435,342 more rows
# i 8 more variables: tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Selecting Columns with a Variable

You can dynamically select columns stored in a character vector using `all_of()` or `any_of()`.

- **`all_of(vars)`** – Selects columns but **errors** out if any are missing.
- **`any_of(vars)`** – Selects columns without error if some are missing.

```
vars <- c("month", "day", "carrier", "flight", "origin", "dest")
flights |>
  select(all_of(vars))
```

```
# A tibble: 435,352 x 6
   month   day carrier flight origin dest
   <int> <int> <chr>    <int> <chr>  <chr>
 1     1     1 UA         628 EWR    SMF
 2     1     1 DL         393 JFK    ATL
 3     1     1 B6         371 JFK    BQN
 4     1     1 B6        1053 JFK    CHS
 5     1     1 UA         219 EWR    DTW
 6     1     1 AA         499 EWR    MIA
 7     1     1 B6         996 JFK    BQN
 8     1     1 AA         981 EWR    ORD
 9     1     1 UA         206 EWR    IAH
10     1     1 NK         225 EWR    FLL
# ℹ 435,342 more rows
```

Selecting a column and using the values with pull()

If you want a vector with the values, use pull()

```
starwars |>
  select(name) |>
  pull()
```

```
 [1] "Luke Skywalker"        "C-3PO"                 "R2-D2"                 "Darth Vader"
 [5] "Leia Organa"           "Owen Lars"             "Beru Whitesun Lars"    "R5-D4"
 [9] "Biggs Darklighter"     "Obi-Wan Kenobi"        "Anakin Skywalker"      "Wilhuff Tarkin"
[13] "Chewbacca"             "Han Solo"              "Greedo"                "Jabba Desilijic Tiure"
[17] "Wedge Antilles"        "Jek Tono Porkins"      "Yoda"                  "Palpatine"
[21] "Boba Fett"             "IG-88"                 "Bossk"                 "Lando Calrissian"
[25] "Lobot"                 "Ackbar"                "Mon Mothma"            "Arvel Crynyd"
[29] "Wicket Systri Warrick" "Nien Nunb"             "Qui-Gon Jinn"          "Nute Gunray"
[33] "Finis Valorum"         "Padmé Amidala"         "Jar Jar Binks"         "Roos Tarpals"
[37] "Rugor Nass"            "Ric Olié"              "Watto"                 "Sebulba"
[41] "Quarsh Panaka"         "Shmi Skywalker"        "Darth Maul"            "Bib Fortuna"
[45] "Ayla Secura"           "Ratts Tyerel"          "Dud Bolt"              "Gasgano"
[49] "Ben Quadinaros"        "Mace Windu"            "Ki-Adi-Mundi"          "Kit Fisto"
[53] "Eeth Koth"             "Adi Gallia"            "Saesee Tiin"           "Yarael Poof"
[57] "Plo Koon"              "Mas Amedda"            "Gregar Typho"          "Cordé"
[61] "Cliegg Lars"           "Poggle the Lesser"     "Luminara Unduli"       "Barriss Offee"
[65] "Dormé"                 "Dooku"                 "Bail Prestor Organa"   "Jango Fett"
[69] "Zam Wesell"            "Dexter Jettster"       "Lama Su"               "Taun We"
```

Filtering Rows with `filter()`

The `filter()` function allows you to **select rows** based on conditions. It keeps only the rows where the condition evaluates to `TRUE`.

Conditions must return a **logical vector** (`TRUE`/`FALSE`) with a length equal to the number of rows in the dataset.

```r
starwars |>
  filter(species == "Droid")
```

```
# 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 C-3PO    167    75 <NA>       gold       yellow           112 none  mascu~ Tatooine  Droid   <chr>
2 R2-D2     96    32 <NA>       white, bl~ red               33 none  mascu~ Naboo     Droid   <chr>
3 R5-D4     97    32 <NA>       white, red red               NA none  mascu~ Tatooine  Droid   <chr>
4 IG-88    200   140 none       metal      red               15 none  mascu~ <NA>      Droid   <chr>
5 R4-P~     96    NA none       silver, r~ red, blue         NA none  femin~ <NA>      Droid   <chr>
6 BB8       NA    NA none       none       black             NA none  mascu~ <NA>      Droid   <chr>
# i 2 more variables: vehicles <list>, starships <list>
```

Applying Multiple Conditions with `filter()`

You can **combine multiple conditions** using:

- A comma (`,`) – Equivalent to using & (logical AND).
- | – Logical OR, keeping rows that meet **at least one** condition.

```
starwars |>
  filter(species %in% c("Human", "Droid"), height < 175) |>
  print(n = 3)
```

```
# A tibble: 14 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 Luke~    172    77 blond      fair       blue              19 male  mascu~ Tatooine  Human   <chr>
2 C-3PO    167    75 <NA>       gold       yellow           112 none  mascu~ Tatooine  Droid   <chr>
3 R2-D2     96    32 <NA>       white, bl~ red               33 none  mascu~ Naboo     Droid   <chr>
# i 11 more rows
# i 2 more variables: vehicles <list>, starships <list>
```

This keeps only: Characters who are either Human or Droid And have a height less than 175 cm. Alternatively, using explicit logical operators:

```
starwars |> filter(species == "Human" | species == "Droid", height < 175)
```

`filter()` - common mistakes

- **= vs ==** for equality tests
  - Wrong: `filter(species = "Human")` (one equal sign leads to error)
  - Right: `filter(species = "Human")`
- **Using OR incorrectly**
  - Wrong: `filter(species == "Human" | "Droid")` (all rows will end up being selected)
  - Right: `filter(species %in% c("Human", "Droid"))` or `filter(species = "Human" | species = "Droid")`

Filtering with Regular Expressions

We will cover **regular expressions (regex)** in a future lecture. For now, note that `str_detect()` (from `stringr`) helps filter text-based patterns.

```r
library(stringr)
starwars |>
  filter(str_detect(name, "^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>
# i 2 more variables: vehicles <list>, starships <list>
```

- `str_detect(name, "^F")` returns `TRUE` for names that start with “F” (^ means “start of the string”).
- `filter()` keeps only rows where this condition holds.

Combining dplyr Functions with the Pipe (|>)

You can **chain multiple** dplyr **functions** using the pipe (|>), making your code more readable and efficient.

The following code keeps only:

- Characters with **no hair (hair_color == "none")**
- OR characters with **black eyes (eye_color == "black")**

Then, it selects only the specified columns.

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

```
# A tibble: 39 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
```

Sorting Rows with arrange()

The arrange() function sorts rows based on a variable. By default, it sorts in ascending order. Use desc() to sort in descending order.

The following first sorts characters by birth year (newest to oldest), then by mass (ascending) within each birth year.

```
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
# i 77 more rows
```

Selecting Rows by Position with `slice()`

The `slice()` function selects rows based on **row number**.

The following line keeps only **rows 5 through 10**, maintaining the original order.

```
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>
# i 2 more variables: vehicles <list>, starships <list>
```

Randomly Sampling Rows with slice_sample()

The `slice_sample()` function selects a **random sample** of rows.

```r
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 Shaa~    178    57 none       red, blue~ black             NA fema~ femin~ Shili     Togruta <chr>
2 Dormé    165    NA brown      light      brown             NA fema~ femin~ Naboo     Human   <chr>
3 Finn      NA    NA black      dark       dark              NA male  mascu~ <NA>      Human   <chr>
4 Mon ~    150    NA auburn     fair       blue              48 fema~ femin~ Chandrila Human   <chr>
5 Obi-~    182    77 auburn, w~ fair       blue-gray         57 male  mascu~ Stewjon   Human   <chr>
# i 2 more variables: vehicles <list>, starships <list>
```

This is useful when: - Previewing **random** portions of a dataset. - Generating a random subset for **testing or analysis**.

Selecting with slice_min() and slice_max()

The slice_min() and slice_max() functions select rows with the **smallest** or **largest** values for a given column.

```
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>
# i 2 more variables: vehicles <list>, starships <list>
```

This is equivalent to:

```
starwars |> arrange(desc(mass)) |> head(3)
```

- slice_max(mass, n = 3) directly finds the top 3 characters with highest mass values.
- slice_min() works the same way but finds the smallest values.

distinct() - unique rows or combos

distinct() removes duplicate rows and returns only unique rows. If multiple columns are selected, then it returns only unique combos of specified columns.

```
# Remove duplicate rows (if any)
starwars |>
  distinct()
```

```
# 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 Skyw~     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 Darth Vad~     202   136 none       white      yellow          41.9 male  mascu~ Tatooine  Human
 5 Leia Orga~     150    49 brown      light      brown           19   fema~ femin~ Alderaan  Human
 6 Owen Lars      178   120 brown, gr~ light      blue            52   male  mascu~ Tatooine  Human
 7 Beru Whit~     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 Biggs Dar~     183    84 black      light      brown           24   male  mascu~ Tatooine  Human
10 Obi-Wan K~     182    77 auburn, w~ fair       blue-gray       57   male  mascu~ Stewjon   Human
# i 77 more rows
# i 3 more variables: films <list>, vehicles <list>, starships <list>
```

```r
starwars |>
  distinct(species)
```

```
# A tibble: 38 x 1
   species
   <chr>
 1 Human
 2 Droid
 3 Wookiee
 4 Rodian
 5 Hutt
 6 <NA>
 7 Yoda's species
 8 Trandoshan
 9 Mon Calamari
10 Ewok
# i 28 more rows
```

```r
starwars |>
  distinct(sex, gender)
```

```
# A tibble: 6 x 2
  sex            gender
  <chr>          <chr>
1 male           masculine
2 none           masculine
3 female         feminine
4 hermaphroditic masculine
5 <NA>           <NA>
6 none           feminine
```

Keep other columns for the first occurrence:

```
starwars |>
  distinct(sex, gender, .keep_all = TRUE)
```

```
# 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 Luke~    172    77 blond      fair       blue              19 male  mascu~ Tatooine  Human   <chr>
2 C-3PO    167    75 <NA>       gold       yellow           112 none  mascu~ Tatooine  Droid   <chr>
3 Leia~    150    49 brown      light      brown             19 fema~ femin~ Alderaan  Human   <chr>
4 Jabb~    175  1358 <NA>       green-tan~ orange           600 herm~ mascu~ Nal Hutta Hutt    <chr>
5 Jek ~    180   110 brown      fair       blue              NA <NA>  <NA>   Bestine ~ <NA>    <chr>
6 R4-P~     96    NA none       silver, r~ red, blue         NA none  femin~ <NA>      Droid   <chr>
# i 2 more variables: vehicles <list>, starships <list>
```

Section 3

making new columns

mutate() - add or transform columns

mutate() computes new columns from existing ones. It keeps all existing columns by default, and puts the newly created column at the end.

```r
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed_mph = distance / (air_time / 60)
  )
```

```
# A tibble: 435,352 x 21
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
 1  2023     1     1        1           2038       203      328              3       205 UA
 2  2023     1     1       18           2300        78      228            135        53 DL
 3  2023     1     1       31           2344        47      500            426        34 B6
 4  2023     1     1       33           2140       173      238           2352       166 B6
 5  2023     1     1       36           2048       228      223           2252       211 UA
 6  2023     1     1      503            500         3      808            815        -7 AA
 7  2023     1     1      520            510        10      948            949        -1 B6
 8  2023     1     1      524            530        -6      645            710       -25 AA
 9  2023     1     1      537            520        17      926            818        68 UA
10  2023     1     1      547            545         2      845            852        -7 NK
# i 435,342 more rows
# i 11 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

option .before will place the new columns before the specified columns. In general, it's fine to have the newly created column appear as the last column in the data, but for the lecture, it's helpful to see the new columns up front.

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed_mph = distance / (air_time / 60),
    .before = everything())
```

```
# A tibble: 435,352 x 21
    gain speed_mph  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl>     <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1    -2      409.  2023     1     1        1           2038       203      328              3
 2    25      422.  2023     1     1       18           2300        78      228            135
 3    13      498.  2023     1     1       31           2344        47      500            426
 4     7      353.  2023     1     1       33           2140       173      238           2352
 5    17      366   2023     1     1       36           2048       228      223           2252
 6    10      423.  2023     1     1      503            500         3      808            815
 7    11      492.  2023     1     1      520            510        10      948            949
 8    19      363.  2023     1     1      524            530        -6      645            710
 9   -51      326.  2023     1     1      537            520        17      926            818
10     9      407.  2023     1     1      547            545         2      845            852
# i 435,342 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Use newly created columns within the same mutate() call:

```r
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    gain_per_hour = gain / (air_time / 60),
    .before = everything()
  )
```

```
# A tibble: 435,352 x 21
    gain gain_per_hour  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl>         <dbl> <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1    -2        -0.327  2023     1     1        1           2038       203      328              3
 2    25        13.9    2023     1     1       18           2300        78      228            135
 3    13         4.11   2023     1     1       31           2344        47      500            426
 4     7         3.89   2023     1     1       33           2140       173      238           2352
 5    17        12.8    2023     1     1       36           2048       228      223           2252
 6    10         3.90   2023     1     1      503            500         3      808            815
 7    11         3.44   2023     1     1      520            510        10      948            949
 8    19         9.58   2023     1     1      524            530        -6      645            710
 9   -51       -11.9    2023     1     1      537            520        17      926            818
10     9         3.44   2023     1     1      547            545         2      845            852
# i 435,342 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
#   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Vectorized conditionals - use `if_else(condition, result_if_true, result_if_false)`:

```r
flights |>
  mutate(
    long_delay = if_else(arr_delay > 60, "> 1 hr", "  1 hr"),
    .before = everything()
    )
```

```
# A tibble: 435,352 x 20
   long_delay  year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <chr>      <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1 > 1 hr      2023     1     1        1           2038       203      328              3       205
 2   1 hr     2023     1     1       18           2300        78      228            135        53
 3   1 hr     2023     1     1       31           2344        47      500            426        34
 4 > 1 hr      2023     1     1       33           2140       173      238           2352       166
 5 > 1 hr      2023     1     1       36           2048       228      223           2252       211
 6   1 hr     2023     1     1      503            500         3      808            815        -7
 7   1 hr     2023     1     1      520            510        10      948            949        -1
 8   1 hr     2023     1     1      524            530        -6      645            710       -25
 9 > 1 hr      2023     1     1      537            520        17      926            818        68
10   1 hr     2023     1     1      547            545         2      845            852        -7
# i 435,342 more rows
# i 10 more variables: carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Multiple cases, use `case_when()`:

```
flights |>
  mutate(
    delay_band = case_when(
      arr_delay <= 0 ~ "early/on time",
      arr_delay <= 30 ~ "0-30",
      arr_delay <= 120 ~ "31-120",
      TRUE ~ ">120"
    ) ,
    .before = everything()
  )
```

```
# A tibble: 435,352 x 20
   delay_band    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <chr>        <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl>
 1 >120          2023     1     1        1           2038       203      328              3       205
 2 31-120        2023     1     1       18           2300        78      228            135        53
 3 31-120        2023     1     1       31           2344        47      500            426        34
 4 >120          2023     1     1       33           2140       173      238           2352       166
 5 >120          2023     1     1       36           2048       228      223           2252       211
 6 early/on t~   2023     1     1      503            500         3      808            815        -7
 7 early/on t~   2023     1     1      520            510        10      948            949        -1
 8 early/on t~   2023     1     1      524            530        -6      645            710       -25
 9 31-120        2023     1     1      537            520        17      926            818        68
10 early/on t~   2023     1     1      547            545         2      845            852        -7
# i 435,342 more rows
```

New Columns will vave the same number of rows

Important: When using `mutate()`, the new column will have an entry for every row in the data. This means that summary functions like mean will result in the same value duplicated across all rows.

Example: Adding a column for the mean of `mass` duplicates the value for all rows

```r
flights |>
  filter(month == 1, day == 1) |>
  select(carrier, flight, dep_time:dep_delay) |>
  mutate(mean_dep_delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 865 x 6
   carrier flight dep_time sched_dep_time dep_delay mean_dep_delay
   <chr>    <int>    <int>          <int>     <dbl>          <dbl>
 1 UA         628        1           2038       203           17.8
 2 DL         393       18           2300        78           17.8
 3 B6         371       31           2344        47           17.8
 4 B6        1053       33           2140       173           17.8
 5 UA         219       36           2048       228           17.8
 6 AA         499      503            500         3           17.8
 7 B6         996      520            510        10           17.8
 8 AA         981      524            530        -6           17.8
 9 UA         206      537            520        17           17.8
```

Useful Functions for `mutate()`

Here are some handy functions that work well inside `mutate()`:

- Row-wise (Element-wise) Functions
  - `pmin()` / `pmax()` – Find the minimum/maximum **for each row** across multiple columns.
- Cumulative Functions
  - `cummin()` / `cummax()` – Keep track of the **minimum/maximum seen so far**.
  - `cumsum()` / `cumprod()` – Compute the **cumulative sum/product**.
  - `cummean()` – Compute the **cumulative mean**.
- Logical Functions
  - `between(x, a, b)` – Checks if `x` is **between** `a` **and** `b`.
- Offset Functions
  - `lead(x, n)` / `lag(x, n)` – Shift values **forward (`lead()`)** or **backward (`lag()`)**.
  - `ntile(x, n)` – Splits a column into `n` **equal-sized bins**.

mutate() Examples

```
flights |>
  filter(month == 1, day == 1) |>
  select(carrier, dep_delay, arr_delay) |>
  mutate(
    cummin_arr_del = cummin(arr_delay), # Keeps track of the smallest mass seen so far.
    pmin_delay = pmin(dep_delay, arr_delay), # Element-wise min between mass & birth_year
    lag2 = lag(pmin_delay, 2) # Shift massyear_pmin values back by 2 rows
  )
```

```
# A tibble: 865 x 6
   carrier dep_delay arr_delay cummin_arr_del pmin_delay  lag2
   <chr>       <dbl>     <dbl>          <dbl>      <dbl> <dbl>
 1 UA            203       205            205        203    NA
 2 DL             78        53             53         53    NA
 3 B6             47        34             34         34   203
 4 B6            173       166             34        166    53
 5 UA            228       211             34        211    34
 6 AA              3        -7             -7         -7   166
 7 B6             10        -1             -7         -1   211
 8 AA             -6       -25            -25        -25    -7
 9 UA             17        68            -25         17    -1
10 NK              2        -7            -25         -7   -25
# i 855 more rows
```

transmute() - mutate + keep only new columns

transmute() is like mutate(), but drops the original columns unless you explicitly keep them.

```r
flights |>
  transmute(
    tailnum,
    gain = dep_delay - arr_delay,
    speed_mph = distance / (air_time / 60)
  )
```

```
# A tibble: 435,352 x 3
   tailnum  gain speed_mph
   <chr>   <dbl>     <dbl>
 1 N25201     -2      409.
 2 N830DN     25      422.
 3 N807JB     13      498.
 4 N265JB      7      353.
 5 N17730     17      366
 6 N925AN     10      423.
 7 N2043J     11      492.
 8 N918AN     19      363.
 9 N13113    -51      326.
10 N912NK      9      407.
# i 435,342 more rows
```

relocate() - move columns without dropping

relocate() moves existing columns relative to others. The .before option in the earlier examples work by making a call to relocate

relocate() is great when you want to keep all columns but improve readability.

```r
# Move dep/arr columns to the front
flights |>
  relocate(dep_time, arr_time)
```

```
# A tibble: 435,352 x 19
   dep_time arr_time  year month   day sched_dep_time dep_delay sched_arr_time arr_delay carrier
      <int>    <int> <int> <int> <int>          <int>     <dbl>          <int>     <dbl> <chr>
 1        1      328  2023     1     1           2038       203              3       205 UA
 2       18      228  2023     1     1           2300        78            135        53 DL
 3       31      500  2023     1     1           2344        47            426        34 B6
 4       33      238  2023     1     1           2140       173           2352       166 B6
 5       36      223  2023     1     1           2048       228           2252       211 UA
 6      503      808  2023     1     1            500         3            815        -7 AA
 7      520      948  2023     1     1            510        10            949        -1 B6
 8      524      645  2023     1     1            530        -6            710       -25 AA
 9      537      926  2023     1     1            520        17            818        68 UA
10      547      845  2023     1     1            545         2            852        -7 NK
# i 435,342 more rows
```

```r
# Move after a reference column
flights |>
  relocate(dep_delay, arr_delay, .after = arr_time)
```

```
# A tibble: 435,352 x 19
    year month   day dep_time sched_dep_time arr_time dep_delay arr_delay sched_arr_time carrier
   <int> <int> <int>    <int>          <int>    <int>     <dbl>     <dbl>          <int> <chr>
 1  2023     1     1        1           2038      328       203       205              3 UA
 2  2023     1     1       18           2300      228        78        53            135 DL
 3  2023     1     1       31           2344      500        47        34            426 B6
 4  2023     1     1       33           2140      238       173       166           2352 B6
 5  2023     1     1       36           2048      223       228       211           2252 UA
 6  2023     1     1      503            500      808         3        -7            815 AA
 7  2023     1     1      520            510      948        10        -1            949 B6
 8  2023     1     1      524            530      645        -6       -25            710 AA
 9  2023     1     1      537            520      926        17        68            818 UA
10  2023     1     1      547            545      845         2        -7            852 NK
# i 435,342 more rows
# i 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```r
# Move to the end
flights |>
  relocate(contains("delay"), .after = last_col())
```

```
# A tibble: 435,352 x 19
    year month   day dep_time sched_dep_time arr_time sched_arr_time carrier flight tailnum origin
   <int> <int> <int>    <int>          <int>    <int>          <int> <chr>    <int> <chr>   <chr>
 1  2023     1     1        1           2038      328              3 UA         628 N25201  EWR
 2  2023     1     1       18           2300      228            135 DL         393 N830DN  JFK
 3  2023     1     1       31           2344      500            426 B6         371 N807JB  JFK
 4  2023     1     1       33           2140      238           2352 B6        1053 N265JB  JFK
 5  2023     1     1       36           2048      223           2252 UA         219 N17730  EWR
 6  2023     1     1      503            500      808            815 AA         499 N925AN  EWR
 7  2023     1     1      520            510      948            949 B6         996 N2043J  JFK
 8  2023     1     1      524            530      645            710 AA         981 N918AN  EWR
 9  2023     1     1      537            520      926            818 UA         206 N13113  EWR
10  2023     1     1      547            545      845            852 NK         225 N912NK  EWR
# i 435,342 more rows
# i 8 more variables: dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, dep_delay <dbl>, arr_delay <dbl>
```

Putting it together

`lubridate::make_date()` combines year, month, day information into a date-type variable, which R understands as dates.

Create derived variables, trim columns, and reorder for reporting:

```
flights |>
  mutate(
    gain = dep_delay - arr_delay,
    speed_mph = distance / (air_time / 60)
  ) |>
  transmute(
    date = lubridate::make_date(year, month, day),
    dep_time, arr_time, origin, dest, carrier,
    gain, speed_mph
  ) |>
  relocate(date, carrier, origin, dest, .before = dep_time)
```

```
# A tibble: 435,352 x 8
   date       carrier origin dest  dep_time arr_time  gain speed_mph
   <date>     <chr>   <chr>  <chr>    <int>    <int> <dbl>     <dbl>
 1 2023-01-01 UA      EWR    SMF          1      328    -2      409.
 2 2023-01-01 DL      JFK    ATL         18      228    25      422.
 3 2023-01-01 B6      JFK    BQN         31      500    13      498.
 4 2023-01-01 B6      JFK    CHS         33      238     7      353.
 5 2023-01-01 UA      EWR    DTW         36      223    17      366
 6 2023-01-01 AA      EWR    MIA        503      808    10      423.
 7 2023-01-01 B6      JFK    BQN        520      948    11      492.
 8 2023-01-01 AA      EWR    ORD        524      645    19      363.
 9 2023-01-01 UA      EWR    IAH        537      926   -51      326.
10 2023-01-01 NK      EWR    FLL        547      845     9      407.
```