Intro to R

Subsetting Data in R

Recap

- R functions as a calculator

- Use `c()` to **combine** vectors

- Use `<-` to save (assign) values to objects

- if you don't use `<-` to reassign objects that you want to modify, they will stay the same

- `length()`, `class()`, and `str()` tell you information about an object

- `head()` and `tail()` can also help you inspect an object

- `readr` has helpful functions like `read_csv()` that can help you import data into R

Cheatsheet

Overview

In this module, we will show you how to:

1. Look at your data in different ways
2. Create a data frame and a tibble
3. Create new variables/make rownames a column
4. Rename columns of a data frame
5. Subset rows of a data frame
6. Subset columns of a data frame
7. Add/remove new columns to a data frame
8. Order the columns of a data frame
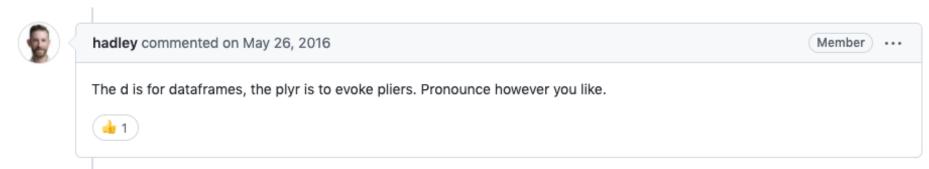9. Order the rows of a data frame

Setup

We will largely focus on the `dplyr` package which is part of the `tidyverse`.

Some resources on how to use `dplyr`:

- https://dplyr.tidyverse.org/
- https://cran.r-project.org/web/packages/dplyr/vignettes/dplyr.html
- https://www.opencasestudies.org/

Why dplyr?

hadley commented on May 26, 2016   Member   ...

The d is for dataframes, the plyr is to evoke pliers. Pronounce however you like.

👍 1

The `dplyr` package is one of the most helpful packages for altering your data to get it into a form that is useful for creating visualizations, summarizing, or more deeply analyzing.

So you can imagine using pliers on your data.

Loading in dplyr and tidyverse

See this website for a list of the packages included in the `tidyverse`:
https://www.tidyverse.org/packages/

```
library(tidyverse) # loads dplyr and other packages!
```

Getting data to work with

Here we use one of the datasets that comes with base R called `mtcars`. We will now create a toy data frame named `df` using this data. This way we can alter `df` without worrying about changing `mtcars`.

```r
df <- mtcars # df is a copy of mtcars
head(df) # changing df does **not** change mtcars!
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Checking the data `dim()`

The `dim()`, `nrow()`, and `ncol()` functions are good options to check the dimensions of your data before moving forward.

```
dim(df) # rows, columns
```

```
[1] 32 11
```

```
nrow(df) # number of rows
```

```
[1] 32
```

```
ncol(df) # number of columns
```

```
[1] 11
```

Checking the data: `glimpse()`

In addition to `head()` and `tail()`, the `glimpse()` function of the `dplyr` package is another great function to look at your data.

```
glimpse(df)
```

```
Rows: 32
Columns: 11
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,
$ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 16
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18
$ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
$ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3,
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2,
```

Checking your data: `slice_sample()`

What if you want to see the middle of your data? You can use the `slice_sample()` function of the `dplyr` package to see a random set of rows. You can specify the number of rows with the `n` argument or use a proportion with the `prop` argument.

```
slice_sample(df, n = 3)
```

```
                   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
Maserati Bora     15.0   8 301.0 335 3.54 3.57 14.60  0  1    5    8
Hornet Sportabout 18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
Merc 230          22.8   4 140.8  95 3.92 3.15 22.90  1  0    4    2
```

```
slice_sample(df, prop = .2)
```

```
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 450SLC    15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

skimr package

```
library(skimr)
skim(df)
```

```
> skim(df)
— Data Summary ─────────────────────
                         Values
Name                     df
Number of rows           32
Number of columns        11

────────────────────────
Column type frequency:
  numeric                11

────────────────────────
Group variables          None
```

```
— Variable type: numeric ──────────────────────────────────────────────
   skim_variable n_missing complete_rate    mean      sd    p0    p25    p50    p75   p100 hist
 1 mpg                   0             1    20.1    6.03  10.4   15.4   19.2   22.8   33.9 ▃▇▅▃▂
 2 cyl                   0             1    6.19    1.79   4      4      6      8      8   ▆▁▃▁▇
 3 disp                  0             1  231.    124.    71.1  121.   196.   326    472   ▇▃▃▃▂
 4 hp                    0             1  147.     68.6   52     96.5  123    180    335   ▇▇▆▃▁
 5 drat                  0             1    3.60    0.535  2.76   3.08   3.70   3.92   4.93 ▇▃▇▅▁
 6 wt                    0             1    3.22    0.978  1.51   2.58   3.32   3.61   5.42 ▃▃▇▁▂
 7 qsec                  0             1   17.8     1.79  14.5   16.9   17.7   18.9   22.9 ▃▇▇▂▁
 8 vs                    0             1    0.438   0.504  0      0      0      1      1   ▇▁▁▁▆
 9 am                    0             1    0.406   0.499  0      0      0      1      1   ▇▁▁▁▆
10 gear                  0             1    3.69    0.738  3      3      4      4      5   ▇▁▆▁▂
11 carb                  0             1    2.81    1.62   1      2      2      4      8   ▆▇▂▇▁
```

Making data frames (base R) and tibbles (tidyverse)

Creating data frames using base R data frame function

```
data.frame(df)
```

```
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
```

Keep in mind...

Need to assign the output of the function to keep the result!

```r
df_updated <- data.frame(df)
# this would overwrite the existing df object
df <- data.frame(df)
```

Or create a data frame when reading in the file

Or directly when reading in a csv with the `read.csv()` function (also base R)

```r
# function comes from base R - no package loading required
df_example_readr <- read.csv(file = "documents/data_analysis/data_file.csv")
```

tibble

We can create a **fancier** version of the previous data frame which can be really helpful.

Creating a `tibble`

If we would like to create a `tibble` ("fancy" data frame), we can using the `tibble()` function.

```
tbl <- tibble(df)
tbl
```

```
# A tibble: 32 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# … with 22 more rows
```

Note don't necessarily need to use `head()` - tibbles conveniently print a portion of the data.

tibbles form read_csv()

Alternatively we can read data files using the `tidyverse` with the `read_csv()` function of the `readr` package from the `tidyverse` to make a tibble.

```
df_example_readr <- read_csv(file = "documents/data_analysis/data_file.csv")
```

You may start to notice how the tidyverse package work well together!

Summary of tibbles and data frames

Base R:

Using `read.csv()` and `data.frame()` you can make data frames

Tidyverse (fancier version):

Using `read_csv()` and `tibble()` you can make tibbles

We generally recommend using tibbles, but you are likely to run into lots of data frames with your work.

Data frames vs tibbles

In the "tidy" data format, rownames are removed. For example, `df` has each car name as a row name. Here we use the `head()` function to see the first 2 rows of each using the `n` argument. In this case we would want to make the rownames a new column first before making into a tibble.

```
head(df, n = 2)
```

```
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
head(tibble(df), n = 2)
```

```
# A tibble: 2 × 11
    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1    21     6   160   110   3.9  2.62  16.5     0     1     4     4
2    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

rownames_to_column function

If you run into losing a variable contained in your row names, you can also use `rownames_to_column` (of `tibble` package) to add it before turning it into a `tibble` to keep them:

```
# general format! not code!
{data you are creating or changing} <- # reassign if you want to keep changes
      rownames_to_column({data you are using},
                         {Name of column you are making from rownames})
```

```
head(df,  n = 2)
```

```
              mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21   6  160 110  3.9 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

```
df <- rownames_to_column(df, "car")
head(df, n = 2)
```

```
            car mpg cyl disp  hp drat    wt  qsec vs am gear carb
1     Mazda RX4  21   6  160 110  3.9 2.620 16.46  0  1    4    4
2 Mazda RX4 Wag  21   6  160 110  3.9 2.875 17.02  0  1    4    4
```

Let's stick with the tibble version

```
tb <- tibble(df)
tb
```

```
# A tibble: 32 × 12
   car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
 1 Mazda RX4     21      6  160    110  3.9   2.62  16.5     0     1     4
 2 Mazda RX4 …   21      6  160    110  3.9   2.88  17.0     0     1     4
 3 Datsun 710    22.8    4  108     93  3.85  2.32  18.6     1     1     4
 4 Hornet 4 D…   21.4    6  258    110  3.08  3.22  19.4     1     0     3
 5 Hornet Spo…   18.7    8  360    175  3.15  3.44  17.0     0     0     3
 6 Valiant       18.1    6  225    105  2.76  3.46  20.2     1     0     3
 7 Duster 360    14.3    8  360    245  3.21  3.57  15.8     0     0     3
 8 Merc 240D     24.4    4  147.    62  3.69  3.19  20       1     0     4
 9 Merc 230      22.8    4  141.    95  3.92  3.15  22.9     1     0     4
10 Merc 280      19.2    6  168.   123  3.92  3.44  18.3     1     0     4
# … with 22 more rows
```

Renaming Columns

Renaming Columns of a data frame or tibble

To rename columns in `dplyr`, you can use the `rename` function.

For example, let's rename mpg to MPG. Notice the new name is listed **first**!

```
# general format! not code!
{data you are creating or changing} <- rename({data you are using},
                                              {New Name} = {Old name})
```

```
tb <- rename(tb, MPG = mpg)
head(tb)
```

```
# A tibble: 6 × 12
  car            MPG   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
1 Mazda RX4       21     6   160   110  3.9   2.62  16.5     0     1     4
2 Mazda RX4 W…    21     6   160   110  3.9   2.88  17.0     0     1     4
3 Datsun 710    22.8     4   108    93  3.85  2.32  18.6     1     1     4
4 Hornet 4 Dr… 21.4      6   258   110  3.08  3.22  19.4     1     0     3
5 Hornet Spor… 18.7      8   360   175  3.15  3.44  17.0     0     0     3
6 Valiant      18.1      6   225   105  2.76  3.46  20.2     1     0     3
```
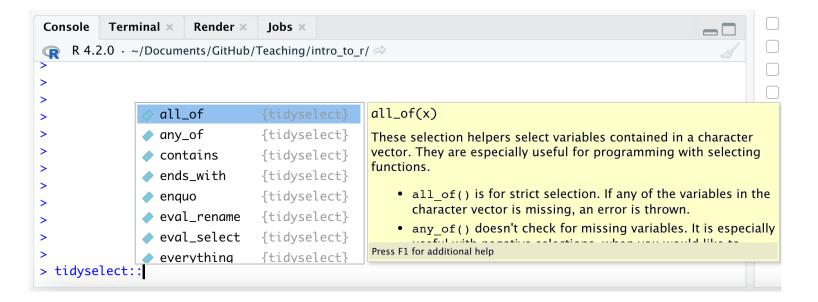
Take Care with Column Names

When you can, avoid spaces, special punctuation, or numbers in column names, as these require special treatment to refer to them.

See https://jhudatascience.org/intro_to_r/quotes_vs_backticks.html for more guidance.

```r
tb <- rename(tb, MPG! = MPG) # this will cause an error
```

```r
tb_rename <-rename(tb, `MPG!` = MPG) # this will work
head(tb_rename, 2)
```

```
# A tibble: 2 × 12
  car         `MPG!`   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4
2 Mazda RX4 …     21     6   160   110   3.9  2.88  17.0     0     1     4
```

You will need to refer to a column like this with most functions.

Unusual Column Names

It's best to avoid unusual column names where possible.

We just showed the use of `` `backticks` ``. You may see people use quotes as well.

Other atypical column names are those with:

· spaces

· number without characters

· number starting the name

· other punctuation marks besides "_"

–> –> –> –> –> –> –> –>

Be careful about copy pasting code!

Curly quotes will not work!

```r
tb_rename <-rename(tb, 'MPG!' = MPG) # this will cause an error!
```

```r
tb_rename <-rename(tb, `MPG!` = MPG) # this will work!
```

Also true for double quotes

```r
tb_rename <-rename(tb, "MPG!" = MPG) # this will cause an error!
```

```r
tb_rename <-rename(tb, "MPG!" = MPG) # this will work!
```

Renaming all columns of a data frame: dplyr

To rename all columns you use the `rename_with()`. In this case we will use `toupper()` to make all letters upper case. Could also use `tolower()` function.

```
tb_upper <- rename_with(tb, toupper)
head(tb_upper, 3)
```

```
# A tibble: 3 × 12
  CAR           MPG   CYL  DISP    HP  DRAT    WT  QSEC    VS    AM  GEAR  CARB
  <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4    21       6   160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 W… 21       6   160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710   22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
```

```
tb <- rename_with(tb, tolower)
head(tb, 3)
```

```
# A tibble: 3 × 12
  car           mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4    21       6   160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 W… 21       6   160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710   22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
```

Summary

- data frames are simpler version of a data table
- tibbles are fancier `tidyverse` version
- data frames are made with `data.frames()` and `read.csv()`
- tibbles are made with `tibble()` and `read_csv()` from `readr`
- if your original data has rownames, you need to use `rownames_to_column` before converting to tibble
- the `rename()` function of `dplyr` can help you rename columns
- avoid using punctuation (except underscores), spaces, and numbers (to start or alone) in column names
- if you must do a nonstandard column name, use backticks to refer to it
- quotes can be used for character values
- avoid copy and pasting code from other sources - quotation marks will change!

Lab Part 1

[Class Website](#)

[Lab](#)

Subsetting Columns

Subset columns of a data frame - `tidyverse` way:

To grab (or "pull" out) the `carb` column the `tidyverse` way we can use the `pull` function:

```
pull(tb, carb)
```

```
 [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
```

Subset columns of a data frame: dplyr

The `select` command from `dplyr` allows you to subset (still a `tibble`!)

```
select(tb, mpg)
```

```
# A tibble: 32 × 1
      mpg
    <dbl>
 1   21
 2   21
 3   22.8
 4   21.4
 5   18.7
 6   18.1
 7   14.3
 8   24.4
 9   22.8
10   19.2
# … with 22 more rows
```

Select multiple columns

We can use `select` to select for multiple columns.

```
select(tb, mpg, car, gear)
```

```
# A tibble: 32 × 3
      mpg car                 gear
    <dbl> <chr>              <dbl>
 1  21    Mazda RX4              4
 2  21    Mazda RX4 Wag          4
 3  22.8  Datsun 710             4
 4  21.4  Hornet 4 Drive         3
 5  18.7  Hornet Sportabout      3
 6  18.1  Valiant                3
 7  14.3  Duster 360             3
 8  24.4  Merc 240D              4
 9  22.8  Merc 230               4
10  19.2  Merc 280               4
# … with 22 more rows
```

Subset columns of a data frame: dplyr

Note that if you want the values (not a `tibble`), use `pull` - as it pulls out the data:

```
pull(tb, mpg)
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.
[31] 15.0 21.4
```

```
# pull with select works too!

pull(select(tb, mpg))
```

```
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.
[16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.
[31] 15.0 21.4
```

Select columns of a data frame: dplyr

The `select` command from `dplyr` allows you to subset columns matching patterns:

```
head(tb, 2)
```

```
# A tibble: 2 × 12
  car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4
```

```
select(tb, starts_with("c"))
```

```
# A tibble: 32 × 3
   car              cyl  carb
   <chr>          <dbl> <dbl>
 1 Mazda RX4          6     4
 2 Mazda RX4 Wag      6     4
 3 Datsun 710         4     1
 4 Hornet 4 Drive     6     1
 5 Hornet Sportabout  8     2
 6 Valiant            6     1
 7 Duster 360         8     4
 8 Merc 240D          4     2
 9 Merc 230           4     2
10 Merc 280           6     4
# … with 22 more rows
```

See the Select "helpers"

Here are a few:

```
last_col()
starts_with()
ends_with()
contains() # like searching
```

Type `tidyselect::` in the **console** and see what RStudio suggests:

Combining tidyselect helpers with regular selection

```
head(tb, 2)
```

```
# A tibble: 2 × 12
  car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4
```

```
select(tb, starts_with("d"), car)
```

```
# A tibble: 32 × 3
    disp  drat car
   <dbl> <dbl> <chr>
 1  160   3.9  Mazda RX4
 2  160   3.9  Mazda RX4 Wag
 3  108   3.85 Datsun 710
 4  258   3.08 Hornet 4 Drive
 5  360   3.15 Hornet Sportabout
 6  225   2.76 Valiant
 7  360   3.21 Duster 360
 8  147.  3.69 Merc 240D
 9  141.  3.92 Merc 230
10  168.  3.92 Merc 280
# … with 22 more rows
```

Multiple tidyselect functions

Follows OR logic.

```
select(tb, starts_with("c"), ends_with("r"))
```

```
# A tibble: 32 × 4
   car                cyl  carb  gear
   <chr>            <dbl> <dbl> <dbl>
 1 Mazda RX4            6     4     4
 2 Mazda RX4 Wag        6     4     4
 3 Datsun 710           4     1     4
 4 Hornet 4 Drive       6     1     3
 5 Hornet Sportabout    8     2     3
 6 Valiant              6     1     3
 7 Duster 360           8     4     3
 8 Merc 240D            4     2     4
 9 Merc 230             4     2     4
10 Merc 280             6     4     4
# … with 22 more rows
```

Multiple patterns with tidyselect

Need to combine the patterns with the `c()` function.

```
select(tb, starts_with(c("c", "d")))
```

```
# A tibble: 32 × 5
   car                 cyl  carb  disp  drat
   <chr>             <dbl> <dbl> <dbl> <dbl>
 1 Mazda RX4             6     4   160  3.9
 2 Mazda RX4 Wag         6     4   160  3.9
 3 Datsun 710            4     1   108  3.85
 4 Hornet 4 Drive        6     1   258  3.08
 5 Hornet Sportabout     8     2   360  3.15
 6 Valiant               6     1   225  2.76
 7 Duster 360            8     4   360  3.21
 8 Merc 240D             4     2   147.  3.69
 9 Merc 230              4     2   141.  3.92
10 Merc 280              6     4   168.  3.92
# … with 22 more rows
```

The `where()` function can help select columns of a specific class

`is.character()` and `is.numeric()` are often the most helpful

```
head(tb, 2)
```

```
# A tibble: 2 × 12
  car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

```
select(tb, where(is.numeric))
```

```
# A tibble: 32 × 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8   360   245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4   147.   62  3.69  3.19  20       1     0     4     2
 9  22.8     4   141.   95  3.92  3.15  22.9     1     0     4     2
10  19.2     6   168.  123  3.92  3.44  18.3     1     0     4     4
# … with 22 more rows
```

Subsetting Rows

Subset rows of a data frame: dplyr

The command in `dplyr` for subsetting rows is `filter`.

```
filter(tb, mpg > 20)
```

```
# A tibble: 14 × 12
     car          mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear   ca
     <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
 1 Mazda RX4    21       6 160    110  3.9   2.62  16.5     0     1     4
 2 Mazda RX4 …  21       6 160    110  3.9   2.88  17.0     0     1     4
 3 Datsun 710   22.8     4 108     93  3.85  2.32  18.6     1     1     4
 4 Hornet 4 D…  21.4     6 258    110  3.08  3.22  19.4     1     0     3
 5 Merc 240D    24.4     4 147.    62  3.69  3.19  20       1     0     4
 6 Merc 230     22.8     4 141.    95  3.92  3.15  22.9     1     0     4
 7 Fiat 128     32.4     4  78.7   66  4.08  2.2   19.5     1     1     4
 8 Honda Civic  30.4     4  75.7   52  4.93  1.62  18.5     1     1     4
 9 Toyota Cor…  33.9     4  71.1   65  4.22  1.84  19.9     1     1     4
10 Toyota Cor…  21.5     4 120.    97  3.7   2.46  20.0     1     0     3
11 Fiat X1-9    27.3     4  79     66  4.08  1.94  18.9     1     1     4
12 Porsche 91…  26       4 120.    91  4.43  2.14  16.7     0     1     5
13 Lotus Euro…  30.4     4  95.1  113  3.77  1.51  16.9     1     1     5
14 Volvo 142E   21.4     4 121    109  4.11  2.78  18.6     1     1     4
```

Subset rows of a data frame: dplyr

You can have multiple logical conditions using the following:

- == : equals to

- !=: not equal to (! : not/negation)

- > / <: greater than / less than

- >= or <=: greater than or equal to / less than or equal to

- & : AND

- | : OR

Common error for filter

If you try to filter for a column that does not exist it will not work:

- misspelled column name
- column that was already removed

Subset rows of a data frame: dplyr

You can filter by two conditions using & or commas (must meet both conditions):

```
filter(tb, mpg > 20, cyl == 4) # same result
```

```
filter(tb, mpg > 20 & cyl == 4)
```

```
# A tibble: 11 × 12
   car          mpg   cyl disp    hp drat    wt  qsec    vs    am  gear  carb
   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Datsun 710  22.8     4 108     93 3.85  2.32  18.6     1     1     4     1
 2 Merc 240D   24.4     4 147.    62 3.69  3.19  20       1     0     4     2
 3 Merc 230    22.8     4 141.    95 3.92  3.15  22.9     1     0     4     2
 4 Fiat 128    32.4     4  78.7   66 4.08  2.2   19.5     1     1     4     1
 5 Honda Civic 30.4     4  75.7   52 4.93  1.62  18.5     1     1     4     2
 6 Toyota Cor… 33.9     4  71.1   65 4.22  1.84  19.9     1     1     4     1
 7 Toyota Cor… 21.5     4 120.    97 3.7   2.46  20.0     1     0     3     1
 8 Fiat X1-9   27.3     4  79     66 4.08  1.94  18.9     1     1     4     1
 9 Porsche 91… 26       4 120.    91 4.43  2.14  16.7     0     1     5     2
10 Lotus Euro… 30.4     4  95.1  113 3.77  1.51  16.9     1     1     5     2
11 Volvo 142E  21.4     4 121    109 4.11  2.78  18.6     1     1     4     2
```

Subset rows of a data frame: dplyr

If you want OR statements (meaning the data can meet either condition does not need to meet both), you need to use | between conditions:

```
filter(tb, mpg > 20 | cyl == 4)
```

```
# A tibble: 14 × 12
     car           mpg   cyl  disp     hp  drat     wt  qsec    vs    am  gear    ca
     <chr>       <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <db
 1 Mazda RX4      21      6 160     110  3.9   2.62  16.5     0     1     4
 2 Mazda RX4 …    21      6 160     110  3.9   2.88  17.0     0     1     4
 3 Datsun 710     22.8    4 108      93  3.85  2.32  18.6     1     1     4
 4 Hornet 4 D…    21.4    6 258     110  3.08  3.22  19.4     1     0     3
 5 Merc 240D      24.4    4 147.     62  3.69  3.19  20       1     0     4
 6 Merc 230       22.8    4 141.     95  3.92  3.15  22.9     1     0     4
 7 Fiat 128       32.4    4  78.7    66  4.08  2.2   19.5     1     1     4
 8 Honda Civic    30.4    4  75.7    52  4.93  1.62  18.5     1     1     4
 9 Toyota Cor…    33.9    4  71.1    65  4.22  1.84  19.9     1     1     4
10 Toyota Cor…    21.5    4 120.     97  3.7   2.46  20.0     1     0     3
11 Fiat X1-9      27.3    4  79      66  4.08  1.94  18.9     1     1     4
12 Porsche 91…    26      4 120.     91  4.43  2.14  16.7     0     1     5
13 Lotus Euro…    30.4    4  95.1   113  3.77  1.51  16.9     1     1     5
14 Volvo 142E     21.4    4 121     109  4.11  2.78  18.6     1     1     4
```

Subset rows of a data frame: dplyr

The `%in%` operator can be used find values from a pre-made list (using `c()`) for a **single column** at a time.

```
filter(tb, mpg %in% c(20,21,22))
```

```
# A tibble: 2 × 12
  car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

```
filter(tb, mpg == 20 | mpg == 21 | mpg == 22) #equivalent
```

```
# A tibble: 2 × 12
  car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4       21     6   160   110   3.9  2.62  16.5     0     1     4     4
2 Mazda RX4 W…    21     6   160   110   3.9  2.88  17.0     0     1     4     4
```

Subset rows of a data frame: dplyr

The `%in%` operator can be used find values from a pre-made list (using `c()`) for a **single column** at a time.

```
filter(tb, gear %in% c(4,5), cyl %in% c(6,5))
```

```
# A tibble: 5 × 12
  car           mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4     21      6  160    110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 W…  21      6  160    110  3.9   2.88  17.0     0     1     4     4
3 Merc 280      19.2    6  168.   123  3.92  3.44  18.3     1     0     4     4
4 Merc 280C     17.8    6  168.   123  3.92  3.44  18.9     1     0     4     4
5 Ferrari Dino  19.7    6  145    175  3.62  2.77  15.5     0     1     5     6
```

Be careful with column names and `filter`

This will not work the way you might expect!

```
filter(iris,"Sepal.Length" > 5)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa

Be careful with column names and `filter`

Using backticks works!

```
filter(iris,`Sepal.Length`> 7.7)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.9         3.8          6.4           2 virginica
```

Be careful with column names and `filter`

Using nothing works!

```
filter(iris, Sepal.Length > 7.7)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.9         3.8          6.4           2 virginica
```

Be careful with column names and `filter`

```
filter(iris, "Species" == "virginica") # this will give an error
```

Be careful with column names and `filter`

```
filter(iris, Species == "virginica") # this works!
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1           6.3         3.3          6.0         2.5 virginica
2           5.8         2.7          5.1         1.9 virginica
3           7.1         3.0          5.9         2.1 virginica
4           6.3         2.9          5.6         1.8 virginica
5           6.5         3.0          5.8         2.2 virginica
6           7.6         3.0          6.6         2.1 virginica
7           4.9         2.5          4.5         1.7 virginica
8           7.3         2.9          6.3         1.8 virginica
9           6.7         2.5          5.8         1.8 virginica
10          7.2         3.6          6.1         2.5 virginica
11          6.5         3.2          5.1         2.0 virginica
12          6.4         2.7          5.3         1.9 virginica
13          6.8         3.0          5.5         2.1 virginica
14          5.7         2.5          5.0         2.0 virginica
15          5.8         2.8          5.1         2.4 virginica
16          6.4         3.2          5.3         2.3 virginica
17          6.5         3.0          5.5         1.8 virginica
18          7.7         3.8          6.7         2.2 virginica
19          7.7         2.6          6.9         2.3 virginica
20          6.0         2.2          5.0         1.5 virginica
21          6.9         3.2          5.7         2.3 virginica
22          5.6         2.8          4.9         2.0 virginica
23          7.7         2.8          6.7         2.0 virginica
24          6.3         2.7          4.9         1.8 virginica
25          6.7         3.3          5.7         2.1 virginica
26          7.2         3.2          6.0         1.8 virginica
27          6.2         2.8          4.8         1.8 virginica
```

`filter()` is tricky

Try not use anything special for the column names in `filter()`

Always good to check each step!

Did the filter work the way you expected? Did the dimensions change?

https://media.giphy.com/media/5b5OU7aUekfdSAER5I/giphy.gif

`distinct()` function

To filter for distinct values from a variable, multiple variables, or an entire tibble you can use the `distinct()` function from the `dplyr` package.

```
distinct(tb, cyl)
```

```
# A tibble: 3 × 1
    cyl
  <dbl>
1     6
2     4
3     8
```

```
distinct(tb, cyl, gear)
```

```
# A tibble: 8 × 2
    cyl  gear
  <dbl> <dbl>
1     6     4
2     4     4
3     6     3
4     8     3
5     4     3
6     4     5
7     8     5
8     6     5
```

Summary

- `pull()` to get values out of a data frame/tibble

- `select()` is the `tidyverse` way to get a tibble with only certain columns

- you can `select()` based on patterns in the column names

- you can also `select()` based on column class with the `where()` function

- you can combine multiple tidyselect functions together like
  `select(starts_with("C"), ends_with("state"))`

- you can combine multiple patterns with the `c()` function like
  `select(starts_with(c("A", "C")))`

- `filter()` can be used to filter out rows based on logical conditions

- avoid using quotes when referring to column names with `filter()`

Summary Continued

- `==` is the same as equivalent to

- `&` means both conditions must be met to remain after `filter()`

- `|` means either conditions needs to be met to remain after `filter()`

- `distinct()` helps you filter for unique values

Lab Part 2

[Class Website](Lab)

Combining `filter` and `select`

You can combine `filter` and `select` to subset the rows and columns, respectively, of a data frame:

```
select(filter(tb, mpg > 20), cyl)
```

```
# A tibble: 14 × 1
     cyl
   <dbl>
 1     6
 2     6
 3     4
 4     6
 5     4
 6     4
 7     4
 8     4
 9     4
10     4
11     4
12     4
13     4
14     4
```

Nesting

In R, the common way to perform multiple operations is to wrap functions around each other in a "nested" form.

```
head(select(tb, car, cyl), 2)
```

```
# A tibble: 2 × 2
  car            cyl
  <chr>        <dbl>
1 Mazda RX4        6
2 Mazda RX4 Wag    6
```

Nesting can get confusing looking

```
select(filter(tb, mpg > 20 & cyl == 4), cyl, hp)
```

```
# A tibble: 11 × 2
      cyl    hp
    <dbl> <dbl>
 1      4    93
 2      4    62
 3      4    95
 4      4    66
 5      4    52
 6      4    65
 7      4    97
 8      4    66
 9      4    91
10      4   113
11      4   109
```

Assigning Temporary Objects

One can also create temporary objects and reassign them:

```
tb2 <- filter(tb, mpg > 20 & cyl == 4)
tb2 <- select(tb2, cyl, hp)

head(tb2,4)
```

```
# A tibble: 4 × 2
    cyl    hp
  <dbl> <dbl>
1     4    93
2     4    62
3     4    95
4     4    66
```

Using the **pipe** (comes with `dplyr`):

The pipe `%>%` makes this much more readable. It reads left side "pipes" into right side. RStudio `CMD/Ctrl + Shift + M` shortcut. Pipe `tb` into `filter`, then pipe that into `select`:

```
tb %>% filter(mpg > 20 & cyl == 4) %>% select(cyl, hp)
```

```
# A tibble: 11 × 2
     cyl    hp
   <dbl> <dbl>
 1     4    93
 2     4    62
 3     4    95
 4     4    66
 5     4    52
 6     4    65
 7     4    97
 8     4    66
 9     4    91
10     4   113
11     4   109
```

Adding/Removing Columns

Adding columns to a data frame: dplyr (**tidyverse** way)

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                         {new variable name} = {new variable source})
```

```
tb <- mutate(tb, newcol = wt / 2.2)
head(tb, 4)
```

```
# A tibble: 4 × 13
  car     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newcol
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd…  21      6   160   110  3.9   2.62  16.5     0     1     4     4   1.19
2 Mazd…  21      6   160   110  3.9   2.88  17.0     0     1     4     4   1.31
3 Dats…  22.8    4   108    93  3.85  2.32  18.6     1     1     4     1   1.05
4 Horn…  21.4    6   258   110  3.08  3.22  19.4     1     0     3     1   1.46
```

Use mutate to modify existing columns

The `mutate` function in `dplyr` allows you to add or modify columns of a data frame.

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                        {variable name to change} = {variable modification})
```

```
tb <- mutate(tb, wt = wt * 2)
head(tb, 4)
```

```
# A tibble: 4 × 13
   car     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newcol
   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd…  21       6   160   110  3.9   5.24  16.5     0     1     4     4   1.19
2 Mazd…  21       6   160   110  3.9   5.75  17.0     0     1     4     4   1.31
3 Dats…  22.8     4   108    93  3.85  4.64  18.6     1     1     4     1   1.05
4 Horn…  21.4     6   258   110  3.08  6.43  19.4     1     0     3     1   1.46
```

You can pipe data into mutate

```r
tb <- tb %>% mutate(wt = wt / 2)
head(tb,4)
```

```
# A tibble: 4 × 13
   car      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newcol
   <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd…   21       6   160   110  3.9   2.62  16.5     0     1     4     4   1.19
2 Mazd…   21       6   160   110  3.9   2.88  17.0     0     1     4     4   1.31
3 Dats…   22.8     4   108    93  3.85  2.32  18.6     1     1     4     1   1.05
4 Horn…   21.4     6   258   110  3.08  3.22  19.4     1     0     3     1   1.46
```

Removing columns of a data frame: dplyr

The `NULL` method is still very common.

The `select` function can remove a column with minus (`-`):

```
select(tb, - newcol)
```

```
# A tibble: 6 × 12
  car           mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>       <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4      21     6   160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 W…   21     6   160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710   22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
4 Hornet 4 Dr… 21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
5 Hornet Spor… 18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
6 Valiant      18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

Or, you can simply select the columns you want to keep, ignoring the ones you want to remove.

Removing columns in a data frame: dplyr

You can use `c()` to list the columns to remove.

Remove `newcol` and `drat`:

```
select(tb, -c("newcol", "drat"))
```

```
# A tibble: 32 × 11
   car                mpg   cyl  disp    hp    wt  qsec    vs    am  gear  carb
   <chr>            <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Mazda RX4         21       6   160   110  2.62  16.5     0     1     4     4
 2 Mazda RX4 Wag     21       6   160   110  2.88  17.0     0     1     4     4
 3 Datsun 710        22.8     4   108    93  2.32  18.6     1     1     4     1
 4 Hornet 4 Drive    21.4     6   258   110  3.22  19.4     1     0     3     1
 5 Hornet Sportabout 18.7     8   360   175  3.44  17.0     0     0     3     2
 6 Valiant           18.1     6   225   105  3.46  20.2     1     0     3     1
 7 Duster 360        14.3     8   360   245  3.57  15.8     0     0     3     4
 8 Merc 240D         24.4     4   147.   62  3.19  20       1     0     4     2
 9 Merc 230          22.8     4   141.   95  3.15  22.9     1     0     4     2
10 Merc 280          19.2     6   168.  123  3.44  18.3     1     0     4     4
# … with 22 more rows
```

Ordering columns

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns.

```
head(tb, 2)
```

```
# A tibble: 2 × 13
  car      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newc
  <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <db
1 Mazd…    21     6   160   110   3.9  2.62  16.5     0     1     4     4   1.
2 Mazd…    21     6   160   110   3.9  2.88  17.0     0     1     4     4   1.
```

```
select(tb, cyl, mpg, wt, car) %>%
head(2)
```

```
# A tibble: 2 × 4
    cyl   mpg    wt car
  <dbl> <dbl> <dbl> <chr>
1     6    21  2.62 Mazda RX4
2     6    21  2.88 Mazda RX4 Wag
```

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns. Put `newcol` first, then select the rest of columns:

```
select(tb, newcol, everything())
```

```
# A tibble: 3 × 13
  newcol car     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   1.19 Mazd…   21      6   160   110  3.9   2.62  16.5     0     1     4     4
2   1.31 Mazd…   21      6   160   110  3.9   2.88  17.0     0     1     4     4
3   1.05 Dats… 22.8      4   108    93  3.85  2.32  18.6     1     1     4     1
```

Ordering the columns of a data frame: dplyr

Put `newcol` at the end ("remove, everything, then add back in"):

```
select(tb, -newcol, everything(), newcol)
```

```
# A tibble: 3 × 13
  car     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newcol
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd…  21      6   160   110   3.9  2.62  16.5     0     1     4     4   1.19
2 Mazd…  21      6   160   110   3.9  2.88  17.0     0     1     4     4   1.31
3 Dats…  22.8    4   108    93   3.85 2.32  18.6     1     1     4     1   1.05
```

Ordering the column names of a data frame: alphabetically

Using the base R `order()` function.

```
order(colnames(tb))
```

```
[1] 10  1 12  3  4  6 11  5  2 13  8  9  7
```

```
tb %>% select(order(colnames(tb)))
```

```
# A tibble: 32 × 13
      am car         carb   cyl  disp  drat  gear    hp   mpg newcol  qsec    vs
   <dbl> <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>
 1     1 Mazda RX4      4     6   160  3.9      4   110  21     1.19  16.5     0
 2     1 Mazda RX4…     4     6   160  3.9      4   110  21     1.31  17.0     0
 3     1 Datsun 710     1     4   108  3.85     4    93  22.8   1.05  18.6     1
 4     0 Hornet 4 …     1     6   258  3.08     3   110  21.4   1.46  19.4     1
 5     0 Hornet Sp…     2     8   360  3.15     3   175  18.7   1.56  17.0     0
 6     0 Valiant        1     6   225  2.76     3   105  18.1   1.57  20.2     1
 7     0 Duster 360     4     8   360  3.21     3   245  14.3   1.62  15.8     0
 8     0 Merc 240D      2     4   147.  3.69    4    62  24.4   1.45  20       1
 9     0 Merc 230       2     4   141.  3.92    4    95  22.8   1.43  22.9     1
10     0 Merc 280       4     6   168.  3.92    4   123  19.2   1.56  18.3     1
# … with 22 more rows, and 1 more variable: wt <dbl>
```

Ordering the columns of a data frame: dplyr

In addition to `select` we can also use the `relocate()` function of dplyr to rearrange the columns for more complicated moves.

For example, let say we just wanted `wt` to be before `cyl`.

```
head(tb, 1)
```

```
# A tibble: 1 × 13
  car      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb newcol
  <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd...   21     6   160   110   3.9  2.62  16.5     0     1     4     4   1.19
```

```
tb_carb <- relocate(tb, wt, .before = cyl)

head(tb_carb, 1)
```

```
# A tibble: 1 × 13
  car      mpg    wt   cyl  disp    hp  drat  qsec    vs    am  gear  carb newcol
  <chr>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
1 Mazd...   21  2.62     6   160   110   3.9  16.5     0     1     4     4   1.19
```

Ordering rows

Ordering the rows of a data frame: dplyr

The `arrange` function can reorder rows By default, `arrange` orders in increasing order:

```
arrange(tb, mpg)
```

```
# A tibble: 32 × 13
   car          mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear   ca
   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <db
 1 Cadillac F…  10.4     8   472   205  2.93  5.25  18.0     0     0     3
 2 Lincoln Co…  10.4     8   460   215  3     5.42  17.8     0     0     3
 3 Camaro Z28   13.3     8   350   245  3.73  3.84  15.4     0     0     3
 4 Duster 360   14.3     8   360   245  3.21  3.57  15.8     0     0     3
 5 Chrysler I…  14.7     8   440   230  3.23  5.34  17.4     0     0     3
 6 Maserati B…  15       8   301   335  3.54  3.57  14.6     0     1     5
 7 Merc 450SLC  15.2     8   276.  180  3.07  3.78  18       0     0     3
 8 AMC Javelin  15.2     8   304   150  3.15  3.44  17.3     0     0     3
 9 Dodge Chal…  15.5     8   318   150  2.76  3.52  16.9     0     0     3
10 Ford Pante…  15.8     8   351   264  4.22  3.17  14.5     0     1     5
# … with 22 more rows, and 1 more variable: newcol <dbl>
```

Ordering the rows of a data frame: dplyr

Use the `desc` to arrange the rows in descending order:

```
arrange(tb, desc(mpg))
```

```
# A tibble: 32 × 13
   car            mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
   <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
 1 Toyota Cor…   33.9     4  71.1    65  4.22  1.84  19.9     1     1     4
 2 Fiat 128      32.4     4  78.7    66  4.08  2.2   19.5     1     1     4
 3 Honda Civic   30.4     4  75.7    52  4.93  1.62  18.5     1     1     4
 4 Lotus Euro…   30.4     4  95.1   113  3.77  1.51  16.9     1     1     5
 5 Fiat X1-9     27.3     4  79      66  4.08  1.94  18.9     1     1     4
 6 Porsche 91…   26       4 120.     91  4.43  2.14  16.7     0     1     5
 7 Merc 240D     24.4     4 147.     62  3.69  3.19  20       1     0     4
 8 Datsun 710    22.8     4 108      93  3.85  2.32  18.6     1     1     4
 9 Merc 230      22.8     4 141.     95  3.92  3.15  22.9     1     0     4
10 Toyota Cor…   21.5     4 120.     97  3.7   2.46  20.0     1     0     3
# … with 22 more rows, and 1 more variable: newcol <dbl>
```

Ordering the rows of a data frame: dplyr

You can combine increasing and decreasing orderings:

```
arrange(tb, mpg, desc(hp))
```

```
# A tibble: 32 × 13
   car          mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  ca
   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <db
 1 Lincoln Co… 10.4     8   460   215  3     5.42  17.8     0     0     3
 2 Cadillac F… 10.4     8   472   205  2.93  5.25  18.0     0     0     3
 3 Camaro Z28  13.3     8   350   245  3.73  3.84  15.4     0     0     3
 4 Duster 360  14.3     8   360   245  3.21  3.57  15.8     0     0     3
 5 Chrysler I… 14.7     8   440   230  3.23  5.34  17.4     0     0     3
 6 Maserati B… 15       8   301   335  3.54  3.57  14.6     0     1     5
 7 Merc 450SLC 15.2     8   276.  180  3.07  3.78  18       0     0     3
 8 AMC Javelin 15.2     8   304   150  3.15  3.44  17.3     0     0     3
 9 Dodge Chal… 15.5     8   318   150  2.76  3.52  16.9     0     0     3
10 Ford Pante… 15.8     8   351   264  4.22  3.17  14.5     0     1     5
# … with 22 more rows, and 1 more variable: newcol <dbl>
```

Summary

- `select()` and `filter()` can be combined together
- you can do sequential steps in a few ways:
  1. nesting them inside one another using parentheses `()`

  2. creating intermediate data objects in between

  3. using pipes `%>%` (like "then" statements)

- `select()` and `relocate()` can be used to reorder columns

- `arrange()` can be used to reorder rows

- can remove rows with `filter()`

- can remove a column in a few ways:
  1. using `select()` with negative sign in front of column name(s)

  2. not selecting it (without negative sign)

Summary cont...

- `mutate()` can be used to create new variables or modify them

```
# General format - Not the code!
{data object to update} <- mutate({data to use},
                                  {new variable name} = {new variable source})
```

```
tb <- mutate(tb, newcol = wt/2.2)
```

A note about base R:

The `$` operator is similar to `pull()`. This is the base R way to do this:

```
tb$carb
```

```
 [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
```

Although it is easier (for this one task), mixing and matching the `$` operator with tidyverse functions usually doesn't work. Therefore, we want to let you know about it in case you see it, but we suggest that you try working with the tidyverse way.

Adding new columns to a data frame: base R

You can add a new column (or modify an existing one) using the `$` operator instead of `mutate`.

Just want you to be aware of this as it is very common.

```
tb$newcol <- tb$wt/2.2
head(tb,3)
```

```
# A tibble: 3 × 13
  car     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb  newc
  <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <db
1 Mazd…  21      6   160   110  3.9   2.62  16.5     0     1     4     4   1.
2 Mazd…  21      6   160   110  3.9   2.88  17.0     0     1     4     4   1.
3 Dats…  22.8    4   108    93  3.85  2.32  18.6     1     1     4     1   1.
```

Even though `$` is easier for creating new columns, `mutate` is really powerful, so it's worth getting used to.

Lab Part 3

Class Website

[Lab](Lab)

Image by Gerd Altmann from Pixabay

Extra Slides

which() function

Instead of removing rows like filter, `which()` simply shows where the values occur if they pass a specific condition. We will see that this can be helpful later when we want to select and filter in more complicated ways.

```
which(select(tb,carb) == 4)
```

```
 [1]  1  2  7 10 11 15 16 17 24 29
```

```
select(tb, carb) == 4
```

```
          carb
 [1,]    TRUE
 [2,]    TRUE
 [3,]   FALSE
 [4,]   FALSE
 [5,]   FALSE
 [6,]   FALSE
 [7,]    TRUE
 [8,]   FALSE
 [9,]   FALSE
[10,]    TRUE
[11,]    TRUE
[12,]   FALSE
[13,]   FALSE
[14,]   FALSE
[15,]    TRUE
[16,]    TRUE
[17,]    TRUE
[18,]   FALSE
```

base R subsetting

Subset columns of a data frame:

We can grab the `carb` column using the `$` operator. This is the base R way to do this:

```
df$carb
```

```
 [1]  4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
```

Remove a column in base R

```r
df$mpg <- NULL
```

Renaming Columns of a data frame: base R

We can use the `colnames` function to extract and/or directly reassign column names of `df`:

```
colnames(df) # just prints
```

```
 [1] "car"  "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"
[11] "gear" "carb"
```

```
colnames(df)[1:3] <- c("MPG", "CYL", "DISP") # reassigns
head(df)
```

```
                  MPG CYL DISP disp  hp drat    wt  qsec vs am gear carb
1         Mazda RX4 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
2     Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
3        Datsun 710 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
4    Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
6           Valiant 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
colnames(df)[1:3] <- c("mpg", "cyl", "disp") #reset - just to keep consistent
```

Renaming Columns of a data frame: base R

We can assign the column names, change the ones we want, and then re-assign the column names:

```
cn <- colnames(df)
cn[ cn == "drat"] <-"DRAT"
colnames(df) <- cn
head(df)
```

```
                  mpg  cyl disp disp   hp DRAT    wt  qsec vs am gear carb
1           Mazda RX4 21.0    6  160 110 3.90 2.620 16.46  0  1    4    4
2       Mazda RX4 Wag 21.0    6  160 110 3.90 2.875 17.02  0  1    4    4
3          Datsun 710 22.8    4  108  93 3.85 2.320 18.61  1  1    4    1
4      Hornet 4 Drive 21.4    6  258 110 3.08 3.215 19.44  1  0    3    1
5   Hornet Sportabout 18.7    8  360 175 3.15 3.440 17.02  0  0    3    2
6             Valiant 18.1    6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
colnames(df)[ colnames(df) == "DRAT"] <- "drat" #reset
```

Subset rows of a data frame with indices:

Let's select **rows** 1 and 3 from `df` using brackets:

```
df[ c(1, 3), ]
```

```
           mpg  cyl disp disp  hp drat   wt  qsec vs am gear carb
1  Mazda RX4 21.0    6  160 110 3.90 2.62 16.46  0  1    4    4
3 Datsun 710 22.8    4  108  93 3.85 2.32 18.61  1  1    4    1
```

Subset columns of a data frame:

We can also subset a data frame using the bracket `[, ]` subsetting.

For data frames and matrices (2-dimensional objects), the brackets are `[rows, columns]` subsetting. We can grab the `x` column using the index of the column or the column name ("`carb`")

```
df[, 11]
```

```
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
```

```
df[, "carb"]
```

```
 [1] 4 4 1 1 2 1 4 2 2 4 4 3 3 3 4 4 4 1 2 1 1 2 2 4 2 1 2 2 4 6 8 2
```

Another difference between `tbl` and data frame:

Mostly, `tbl` (tibbles) are the same as data frames, except they don't print all lines. When subsetting only one column using brackets, a data frame will return the values, but a `tbl` will return a `tbl`

```
df[, 1]
```

```
 [1] "Mazda RX4"          "Mazda RX4 Wag"      "Datsun 710"
 [4] "Hornet 4 Drive"     "Hornet Sportabout"  "Valiant"
 [7] "Duster 360"         "Merc 240D"          "Merc 230"
[10] "Merc 280"           "Merc 280C"          "Merc 450SE"
[13] "Merc 450SL"         "Merc 450SLC"        "Cadillac Fleetwood"
[16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"
[19] "Honda Civic"        "Toyota Corolla"     "Toyota Corona"
[22] "Dodge Challenger"   "AMC Javelin"        "Camaro Z28"
[25] "Pontiac Firebird"   "Fiat X1-9"          "Porsche 914-2"
[28] "Lotus Europa"       "Ford Pantera L"     "Ferrari Dino"
[31] "Maserati Bora"      "Volvo 142E"
```

```
tbl[, 1]
```

```
# A tibble: 32 × 1
    mpg
  <dbl>
1  21
2  21
3  22.8
4  21.4
5  18.7
6  18.1
```

Subset columns of a data frame:

We can select multiple columns using multiple column names:

```
df[, c("mpg", "cyl")]
```

```
                      mpg  cyl
1             Mazda RX4 21.0
2         Mazda RX4 Wag 21.0
3            Datsun 710 22.8
4        Hornet 4 Drive 21.4
5     Hornet Sportabout 18.7
6               Valiant 18.1
7            Duster 360 14.3
8             Merc 240D 24.4
9              Merc 230 22.8
10             Merc 280 19.2
11            Merc 280C 17.8
12            Merc 450SE 16.4
13            Merc 450SL 17.3
14           Merc 450SLC 15.2
15    Cadillac Fleetwood 10.4
16   Lincoln Continental 10.4
17     Chrysler Imperial 14.7
18              Fiat 128 32.4
19            Honda Civic 30.4
20         Toyota Corolla 33.9
21          Toyota Corona 21.5
22       Dodge Challenger 15.5
23            AMC Javelin 15.2
24             Camaro Z28 13.3
25         Pontiac Firebird 19.2
```