

Subsetting Data in R

Reminder

Refresh the website and get the latest version of the labs and slides! We are constantly making improvements.

Recap

- Use `<-` to save (assign) values to objects
- Use `c()` to **combine** vectors
- `length()`, `class()`, and `str()` tell you information about an object
- The sequence `seq()` function helps you create numeric vectors (`from`, `to`, `by`, and `length.out` arguments)
- The repeat `rep()` function helps you create vectors with the `each` and `times` arguments
- Reproducible science makes everyone's life easier!
- `readr` has helpful functions like `read_csv()` that can help you import data into R

□ [Day 2 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.



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

We will work with data called `er` about heat-related ER visits between 2011 and 2022, as reported by the state of Colorado, specifically made available by the Colorado Environmental Public Health Tracking program website. Full dataset available at <https://coepht.colorado.gov/heat-related-illness>.

```
er <-  
  read_csv("https://daseh.org/data/CO_ER_heat_visits.csv")
```

Rows: 768 Columns: 6

— Column specification —————

Delimiter: ","

chr (1): county

dbl (5): rate, lower95cl, upper95cl, visits, year

- Use `spec()` to retrieve the full column specification for this data.
- Specify the column types or set `show_col_types = FALSE` to quiet this message

Checking the data `dim()`

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

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

```
[1] 768    6
```

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

```
[1] 768
```

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

```
[1] 6
```

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(er)
```

Rows: 768

Columns: 6

```
$ county    <chr> "Adams", "Adams", "Adams", "Adams", "Adams", "Adams", "Adams..."  
$ rate      <dbl> 6.729918, 4.843983, 6.836648, 3.080950, 3.356538, 8.848504, ...  
$ lower95cl <dbl> NA, 2.848937, 4.359735, 1.711087, 1.892912, 6.124961, 4.2920...  
$ upper95cl <dbl> 9.236776, NA, 9.313561, 4.846996, 5.232461, 11.572046, 8.977...  
$ visits    <dbl> 29, 23, 31, 15, 16, 42, 32, 37, 36, 24, 35, 45, 0, 0, NA, 0,...  
$ year      <dbl> 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, ...
```

This dataset has information about both the *rate* of ER visits for heat-related illness and the *number of visits*, broken out by both year and CO county.

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.

```
slice_sample(er, n = 2)
```

```
# A tibble: 2 × 6
  county      rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>     <dbl>    <dbl>   <dbl>
1 Teller       NA       NA       NA       NA   2021
2 Broomfield   NA       NA       NA       NA   2021
```

```
slice_sample(er, n = 2)
```

```
# A tibble: 2 × 6
  county      rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>     <dbl>    <dbl>   <dbl>
1 Mesa        8.88    4.53     14.7     12   2012
2 Huerfano    0        0         0        0    2020
```

Data frames and tibbles

Data frames

An older version of data in tables is called a data frame. The mtcars dataset is an example of this.

```
class(mtcars)
```

```
[1] "data.frame"
```

```
head(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

tibble

Tibbles are a **fancier** version of data frames:

- We don't have to use head to see a preview of it
- We see the dimensions
- We see the data types for each column

er

```
# A tibble: 768 × 6
  county   rate lower95cl upper95cl visits year
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>
1 Adams     6.73      NA       9.24     29  2011
2 Adams     4.84      2.85     NA        23  2012
3 Adams     6.84      4.36     9.31     31  2013
4 Adams     3.08      1.71     4.85     15  2014
5 Adams     3.36      1.89     5.23     16  2015
6 Adams     8.85      6.12    11.6      42  2016
7 Adams     6.63      4.29     8.98     32  2017
8 Adams     7.11      4.77     9.44     37  2018
9 Adams     6.76      4.53     8.99     36  2019
10 Adams    4.76      2.82     6.70     24  2020
# ... 758 more rows
```

Creating a **tibble**

If we wanted to create a **tibble** ("fancy" data frame), we can use the **tibble()** function on a data frame.

```
tbl_mtcars <- tibble(mtcars)
```

```
tbl_mtcars
```

```
# A tibble: 32 × 11
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	<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
# 22 more rows											

Note don't necessarily need to use **head()** with tibbles, as they conveniently print a portion of the data.

Summary of tibbles and data frames

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

Most functions work for both so you don't need to worry about it much!

It can be helpful to convert data frames to tibbles though just to see more about the data more easily. The `tibble()` function helps us do that.

Data frames vs tibbles - watch out for rownames

Note that this conversion can remove row names - which some data frames have. For example, `mtcars` (part of R) has row names. In this case we would want to make the rownames a new column first before making into a tibble.

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

```
# A tibble: 2 × 11
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <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

There is a function that specifically helps you do that.

```
head(rownames_to_column(mtcars), n = 2)
```

	rowname	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

```
head(tibble(rownames_to_column(mtcars)), n = 2)
```

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

Data for now

Let's stick with the tibble ER data for our next lesson

```
head(er)
```

```
# A tibble: 6 × 6
  county  rate lower95cl upper95cl visits year
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>
1 Adams    6.73      NA       9.24     29  2011
2 Adams    4.84      2.85     NA        23  2012
3 Adams    6.84      4.36     9.31     31  2013
4 Adams    3.08      1.71     4.85     15  2014
5 Adams    3.36      1.89     5.23     16  2015
6 Adams    8.85      6.12    11.6      42  2016
```

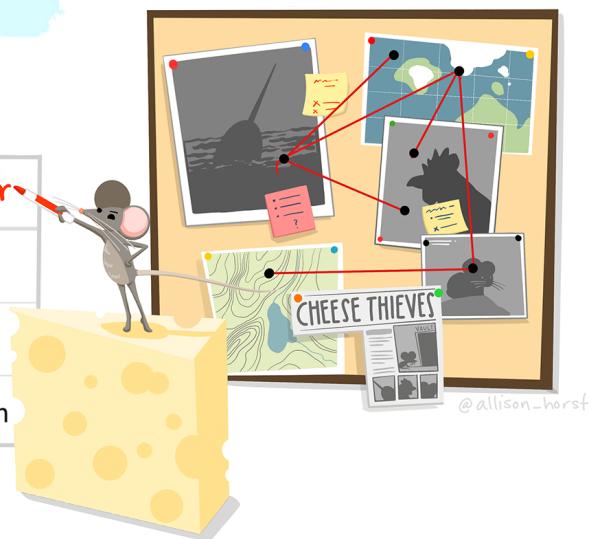
Renaming Columns

rename function

dplyr::rename()
RENAME COLUMNS*

df %>% rename(lair=site)

species nemesis	status	site lair
narwhal	unknown	ocean
chicken	active	coop
pika	active	mountain



*See `rename_with()` to rename using a function.

"Artwork by @allison_horst". <https://allisonhorst.com/>

checking names of columns, we can use the **colnames()** function

```
colnames(er)
```

```
[1] "county"      "rate"        "lower95cl"    "upper95cl"   "visits"      "year"
```

Renaming Columns of a data frame or tibble

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

Let's rename `lower95CI` to `lower_95_CI_limit`. Notice the new name is listed **first**, similar to how a new object is assigned on the left!

The `lower95CI` is the lower bound of the estimated rate of ER visits for a particular county and year.

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

```
renamed_er<- rename(er, lower_95_CI_limit = lower95cl)
head(renamed_er)
```

```
# A tibble: 6 × 6
  county   rate lower_95_CI_limit upper95cl  visits  year
  <chr>   <dbl>           <dbl>      <dbl>    <dbl> <dbl>
1 Adams     6.73            NA        9.24     29    2011
2 Adams     4.84            2.85       NA        23    2012
3 Adams     6.84            4.36       9.31     31    2013
4 Adams     3.08            1.71       4.85     15    2014
5 Adams     3.36            1.89       5.23     16    2015
6 Adams     8.85            6.12      11.6      42    2016
```

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://daseh.org/resources/quotes_vs_backticks.html for more guidance.

```
# this will cause an error
renamed_er <- rename(er, lower_95%_CI_limit = lower95cl)

# this will work
renamed_er <- rename(er, `lower_95%_CI_limit` = lower95cl)
head(renamed_er, 2)

# A tibble: 2 × 6
  county   rate `lower_95%_CI_limit` upper95cl visits year
  <chr>    <dbl>            <dbl>        <dbl>   <dbl> <dbl>
1 Adams     6.73             NA          9.24     29  2011
2 Adams     4.84             2.85         NA       23  2012
```

Unusual Column Names

It's best to avoid unusual column names where possible, as things get tricky later.

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 like % (besides "_" or "." and not at the beginning)

A solution!

Rename tricky column names so that you don't have to deal with them later!



Be careful about copy pasting code!

Curly quotes will not work!

```
# this will cause an error!
renamed_er <- rename(er, 'lower_95%_CI_limit' = lower95cl)
```

```
# this will work!
renamed_er <- rename(er, 'lower_95%_CI_limit' = lower95cl)
```

Also true for double quotes

```
# this will cause an error!
renamed_er <- rename(er, "lower_95%_CI_limit" = lower95cl)
```

```
# this will work!
renamed_er <- rename(er, "lower_95%_CI_limit" = lower95cl)
```

Rename multiple columns

A comma can separate different column names to change.

```
renamed_er <- rename(er,  
                      lower_95perc_CI_limit = lower95cl,  
                      upper_95perc_CI_limit = upper95cl)
```

```
head(renamed_er, 3)
```

```
# A tibble: 3 × 6  
  county   rate lower_95perc_CI_limit upper_95perc_CI_limit visits   year  
  <chr>   <dbl>            <dbl>                <dbl>    <dbl>    <dbl>  
1 Adams     6.73             NA                  9.24      29    2011  
2 Adams     4.84             2.85                 NA        23    2012  
3 Adams     6.84             4.36                 9.31      31    2013
```

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.

```
er_upper <- rename_with(er, toupper)
```

```
head(er_upper, 3)
```

```
# A tibble: 3 × 6
```

	COUNTY	RATE	LOWER95CL	UPPER95CL	VISITS	YEAR
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Adams	6.73	NA	9.24	29	2011
2	Adams	4.84	2.85	NA	23	2012
3	Adams	6.84	4.36	9.31	31	2013

```
rename_with(er_upper, tolower)
```

```
# A tibble: 768 × 6
```

	county	rate	lower95cl	upper95cl	visits	year
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Adams	6.73	NA	9.24	29	2011
2	Adams	4.84	2.85	NA	23	2012
3	Adams	6.84	4.36	9.31	31	2013
4	Adams	3.08	1.71	4.85	15	2014
5	Adams	3.36	1.89	5.23	16	2015
6	Adams	8.85	6.12	11.6	42	2016
7	Adams	6.63	4.29	8.98	32	2017
8	Adams	7.11	4.77	9.44	37	2018
9	Adams	6.76	4.53	8.99	36	2019
10	Adams	4.76	2.82	6.70	24	2020

```
# 758 more rows
```

janitor package

If you need to do lots of naming fixes - look into the janitor package!

```
#install.packages("janitor")
library(janitor)
```

janitor `clean_names`

The `clean_names` function can intuit what fixes you might need.

The `yearly_co2_emissions` dataset contains estimated CO2 emissions for 265 countries between the years 1751 and 2014.

```
yearly_co2 <-  
  read_csv("https://daseh.org/data/Yearly_CO2_Emissions_1000_tonnes.csv")
```

Rows: 192 Columns: 265

— Column specification —

Delimiter: ","

chr (1): country

dbl (264): 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, ...

- Use ``spec()`` to retrieve the full column specification for this data.
- Specify the column types or set ``show_col_types = FALSE`` to quiet this message

yearly_co2 column names

```
head(yearly_co2, n = 2)
```

```
# A tibble: 2 × 265
  country `1751` `1752` `1753` `1754` `1755` `1756` `1757` `1758` `1759` `1760` ...
  <chr>   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> ...
1 Afghani...     NA     NA     NA     NA     NA     NA     NA     NA     NA     NA ...
2 Albania       NA     NA     NA     NA     NA     NA     NA     NA     NA     NA ...
# ℒ 254 more variables: `1761` <dbl>, `1762` <dbl>, `1763` <dbl>, `1764` <dbl>,
#   `1765` <dbl>, `1766` <dbl>, `1767` <dbl>, `1768` <dbl>, `1769` <dbl>,
#   `1770` <dbl>, `1771` <dbl>, `1772` <dbl>, `1773` <dbl>, `1774` <dbl>,
#   `1775` <dbl>, `1776` <dbl>, `1777` <dbl>, `1778` <dbl>, `1779` <dbl>,
#   `1780` <dbl>, `1781` <dbl>, `1782` <dbl>, `1783` <dbl>, `1784` <dbl>,
#   `1785` <dbl>, `1786` <dbl>, `1787` <dbl>, `1788` <dbl>, `1789` <dbl>,
#   `1790` <dbl>, `1791` <dbl>, `1792` <dbl>, `1793` <dbl>, `1794` <dbl>, ...
```

janitor `clean_names` can intuit fixes

The `clean_names` function can intuit what fixes you might need. Here it make sure year names aren't just a number, so that the colnames don't need ticks or quotes to be used.

```
clean_names(yearly_co2)
```

```
# A tibble: 192 × 265
  country      x1751  x1752  x1753  x1754  x1755  x1756  x1757  x1758  x1759  x1760  x1761
  <chr>       <dbl>  <dbl>
1 Afghanistan     NA     NA
2 Albania         NA     NA
3 Algeria          NA     NA
4 Andorra          NA     NA
5 Angola            NA    NA
6 Antigua and...   NA    NA
7 Argentina        NA    NA
8 Armenia           NA    NA
9 Australia         NA    NA
10 Austria          NA   NA
# ... 182 more rows
# ... 253 more variables: x1762 <dbl>, x1763 <dbl>, x1764 <dbl>, x1765 <dbl>,
# ... x1766 <dbl>, x1767 <dbl>, x1768 <dbl>, x1769 <dbl>, x1770 <dbl>,
# ... x1771 <dbl>, x1772 <dbl>, x1773 <dbl>, x1774 <dbl>, x1775 <dbl>,
# ... x1776 <dbl>, x1777 <dbl>, x1778 <dbl>, x1779 <dbl>, x1780 <dbl>,
# ... x1781 <dbl>, x1782 <dbl>, x1783 <dbl>, x1784 <dbl>, x1785 <dbl>,
# ... x1786 <dbl>, x1787 <dbl>, x1788 <dbl>, x1789 <dbl>, x1790 <dbl>, ...
```

more of clean_names

clean_names can also get rid of spaces and replace them with _.

```
test <- tibble(`col 1` = c(1,2,3), `col 2` = c(2,3,4))  
test
```

```
# A tibble: 3 × 2  
  `col 1` `col 2`  
  <dbl>   <dbl>  
1      1      2  
2      2      3  
3      3      4
```

```
clean_names(test)
```

```
# A tibble: 3 × 2  
  col_1 col_2  
  <dbl> <dbl>  
1      1      2  
2      2      3  
3      3      4
```

GUT CHECK: Which of the following would work well with a column

called `counties_of_seattle_with_population_over_10,000`?

- A. Renaming it using `rename` function to something simpler like `seattle_counties_over_10thous`.
- B. Keeping it as is and use backticks around the column name when you use it.
- C. Keeping it as is and use quotes around the column name when you use it.

Summary

- data frames are simpler version of a data table
- tibbles are fancier `tidyverse` version
- tibbles are made with `tibble()`
- 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 - typically use backticks around it. See https://daseh.org/resources/quotes_vs_backticks.html.
- avoid copy and pasting code from other sources - quotation marks will change!
- check out `janitor` if you need to make lots of column name changes

Lab Part 1

- [Class Website](#)
- [Lab](#)
- [Day 3 Cheatsheet](#)
- [Posit's dplyr Cheatsheet](#)

Subsetting Columns

Let's get our data again

We'll work with the CO heat-related ER visits dataset again.

This time lets also make it a smaller subset so it is easier for us to see the full dataset as we work through examples.

```
# er<-read_csv("https://daseh.org/data/CO_ER_heat_visits.csv")
set.seed(1234)
er_30 <- slice_sample(er, n = 30)
```

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

To grab a vector version (or “pull” out) the year column the **tidyverse** way we can use the `pull` function:

```
pull(er_30, year)
```

```
[1] 2018 2015 2021 2019 2014 2012 2017 2016 2012 2012 2017 2016 2020 2014 2020  
[16] 2014 2011 2022 2018 2013 2017 2021 2015 2020 2012 2016 2013 2014 2017 2022
```

Subset columns of a data frame: dplyr

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

```
select(er_30, year)
```

```
# A tibble: 30 × 1
```

```
  year
  <dbl>
1 2018
2 2015
3 2021
4 2019
5 2014
6 2012
7 2017
8 2016
9 2012
10 2012
# ... 20 more rows
```

GUT CHECK: What function would be useful for getting a vector version of a column?

- A. `pull()`
- B. `select()`

Select multiple columns

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

```
select(er_30, year, rate, county)
```

```
# A tibble: 30 × 3
  year    rate county
  <dbl>   <dbl> <chr>
1 2018     NA Garfield
2 2015     NA Chaffee
3 2021  15.9 Pueblo
4 2019      0 Rio Grande
5 2014      0 Lake
6 2012     NA Chaffee
7 2017     NA Chaffee
8 2016      0 Teller
9 2012     NA Prowers
10 2012     0 Hinsdale
# ... 20 more rows
```

Select columns of a data frame: dplyr

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

```
head(er_30, 2)
```

```
# A tibble: 2 × 6
  county    rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl>
1 Garfield      NA        NA        NA      NA  2018
2 Chaffee       NA        NA        NA      NA  2015
```

```
select(er_30, ends_with("c1"))
```

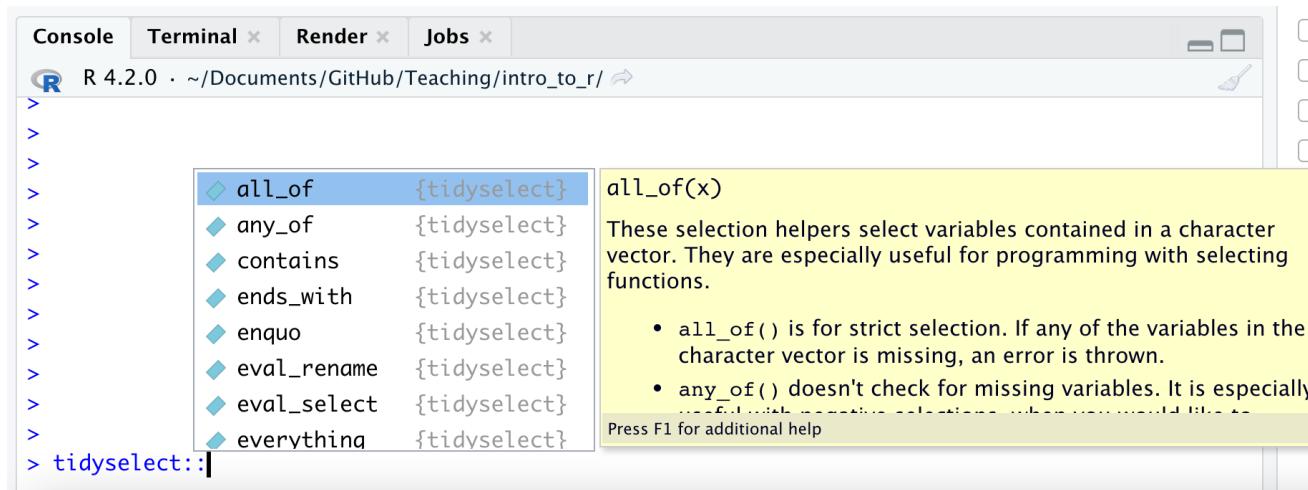
```
# A tibble: 30 × 2
  lower95cl upper95cl
  <dbl>     <dbl>
1 NA         NA
2 NA         NA
3 9.54      22.2
4 0          0
5 0          0
6 NA         NA
7 NA         NA
8 0          0
9 NA         NA
10 0         0
# ... 20 more rows
```

See the Select “helpers”

Here are a few:

```
last_col()  
starts_with()  
ends_with()  
contains()
```

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



Combining tidyselect helpers with regular selection

```
head(er_30, 2)
```

```
# A tibble: 2 × 6
  county    rate lower95cl upper95cl visits   year
  <chr>     <dbl>     <dbl>     <dbl>   <dbl> <dbl>
1 Garfield     NA        NA        NA      NA  2018
2 Chaffee      NA        NA        NA      NA  2015
```

```
select(er_30, ends_with("c1"), year)
```

```
# A tibble: 30 × 3
  lower95cl upper95cl year
  <dbl>     <dbl> <dbl>
1 NA        NA    2018
2 NA        NA    2015
3 9.54     22.2  2021
4 0         0    2019
5 0         0    2014
6 NA        NA    2012
7 NA        NA    2017
8 0         0    2016
9 NA        NA    2012
10 0        0    2012
# ... 20 more rows
```

Multiple tidyselect functions

Follows OR logic.

```
select(er_30, ends_with("cl"), starts_with("r"))
```

```
# A tibble: 30 × 3
```

	lower95cl	upper95cl	rate
	<dbl>	<dbl>	<dbl>
1	NA	NA	NA
2	NA	NA	NA
3	9.54	22.2	15.9
4	0	0	0
5	0	0	0
6	NA	NA	NA
7	NA	NA	NA
8	0	0	0
9	NA	NA	NA
10	0	0	0
# ... 20 more rows			

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

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

```
head(er_30, 2)
```

```
# A tibble: 2 × 6
  county    rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl>
1 Garfield     NA       NA       NA       NA  2018
2 Chaffee      NA       NA       NA       NA  2015
```

```
select(er_30, where(is.numeric))
```

```
# A tibble: 30 × 5
  rate lower95cl upper95cl visits   year
  <dbl>    <dbl>     <dbl>    <dbl> <dbl>
1 NA       NA       NA       NA  2018
2 NA       NA       NA       NA  2015
3 15.9     9.54    22.2     26  2021
4 0        0        0        0  2019
5 0        0        0        0  2014
6 NA       NA       NA       NA  2012
7 NA       NA       NA       NA  2017
8 0        0        0        0  2016
9 NA       NA       NA       NA  2012
10 0       0        0        0  2012
# ... 20 more rows
```

Subsetting Rows

filter function

dplyr:: filter()

KEEP ROWS THAT
s.a.t.i.s.f.y
your CONDITIONS

keep rows from... this data... ONLY IF... type is "otter" AND site is "bay"

```
filter(df, type == "otter" & site == "bay")
```

type	food	site
otter	urchin	bay
Shark	seal	channel
otter	abalone	bay
otter	crab	wharf

"Artwork by @allison_horst". <https://allisonhorst.com/>

Subset rows of a data frame: dplyr

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

```
filter(er_30, year > 2020)
```

```
# A tibble: 4 × 6
  county    rate lower95cl upper95cl visits   year
  <chr>    <dbl>    <dbl>     <dbl>    <dbl>   <dbl>
1 Pueblo    15.9     9.54     22.2      26    2021
2 Otero      NA       NA       NA       NA    2022
3 Mesa      12.0     7.12     18.2      19    2021
4 Chaffee    0        0        0        0    2022
```

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(er_30, rate > 9, year == 2012)

filter(er_30, rate > 9 & year == 2012) # same result

# A tibble: 0 × 6
#   county    rate lower95cl upper95cl visits year
#   <chr>     <dbl>     <dbl>     <dbl>    <dbl> <dbl>
```

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(er_30, rate > 9 | year == 2012)
```

```
# A tibble: 6 × 6
  county    rate lower95cl upper95cl visits  year
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>
1 Pueblo    15.9     9.54    22.2     26  2021
2 Chaffee    NA       NA       NA      NA  2012
3 Prowers    NA       NA       NA      NA  2012
4 Hinsdale    0        0        0       0  2012
5 Mesa      12.0     7.12    18.2     19  2021
6 Phillips    0        0        0       0  2012
```

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(er_30, county %in% c("Denver", "Larimer", "Pueblo"))
```

```
# A tibble: 3 × 6
  county    rate lower95cl upper95cl visits year
  <chr>    <dbl>    <dbl>    <dbl>   <dbl> <dbl>
1 Pueblo  15.9     9.54    22.2     26  2021
2 Denver   2.95    1.75    4.46     19  2013
3 Larimer  5.49    3.16    8.45     17  2014
```

```
filter(er_30, county == "Denver" | county == "Larimer" | county == "Pueblo") #equivalent
```

```
# A tibble: 3 × 6
  county    rate lower95cl upper95cl visits year
  <chr>    <dbl>    <dbl>    <dbl>   <dbl> <dbl>
1 Pueblo  15.9     9.54    22.2     26  2021
2 Denver   2.95    1.75    4.46     19  2013
3 Larimer  5.49    3.16    8.45     17  2014
```

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 with different columns.

```
filter(er_30, year %in% c(2013, 2021), county %in% c("Denver", "Pueblo"))
```

```
# A tibble: 2 × 6
  county   rate lower95cl upper95cl visits   year
  <chr>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>
1 Pueblo  15.9      9.54    22.2      26    2021
2 Denver   2.95     1.75     4.46      19    2013
```

Be careful with column names and **filter**

This will not work the way you might expect! Best to stick with nothing but the column name if it is a typical name.

```
filter(er_30, "year" > 2014)
```

```
# A tibble: 30 × 6
  county      rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>    <dbl>    <dbl>   <dbl>
1 Garfield     NA       NA       NA       NA    2018
2 Chaffee      NA       NA       NA       NA    2015
3 Pueblo      15.9     9.54    22.2     26    2021
4 Rio Grande    0        0        0        0    2019
5 Lake         0        0        0        0    2014
6 Chaffee      NA       NA       NA       NA    2012
7 Chaffee      NA       NA       NA       NA    2017
8 Teller        0        0        0        0    2016
9 Prowers      NA       NA       NA       NA    2012
10 Hinsdale     0        0        0        0    2012
# ℒ 20 more rows
```

Don't use quotes for atypical names

Atypical names are those with punctuation, spaces, start with a number, or are just a number.

```
er_30_rename <- rename(er_30, `year!` = year)  
filter(er_30_rename, "year!" > 2013) # will not work correctly
```

```
# A tibble: 30 × 6  
  county      rate lower95cl upper95cl visits `year!`  
  <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
1 Garfield     NA       NA       NA       NA     2018  
2 Chaffee      NA       NA       NA       NA     2015  
3 Pueblo      15.9     9.54    22.2     26     2021  
4 Rio Grande    0        0        0        0     2019  
5 Lake          0        0        0        0     2014  
6 Chaffee      NA       NA       NA       NA     2012  
7 Chaffee      NA       NA       NA       NA     2017  
8 Teller         0        0        0        0     2016  
9 Prowers       NA       NA       NA       NA     2012  
10 Hinsdale      0        0        0        0     2012  
# ... 20 more rows
```

Be careful with column names and **filter**

Using backticks works!

```
filter(er_30_rename, `year!` > 2013)
```

```
# A tibble: 23 × 6
  county      rate lower95cl upper95cl visits `year!`
  <chr>     <dbl>    <dbl>     <dbl>   <dbl>    <dbl>
1 Garfield     NA       NA       NA      NA    2018
2 Chaffee      NA       NA       NA      NA    2015
3 Pueblo      15.9     9.54    22.2     26    2021
4 Rio Grande    0        0        0       0    2019
5 Lake          0        0        0       0    2014
6 Chaffee      NA       NA       NA      NA    2017
7 Teller         0        0        0       0    2016
8 Boulder      3.51     1.77    5.83     12    2017
9 Fremont       NA       NA       NA      NA    2016
10 Kiowa        NA       NA       NA      NA    2020
# ℹ 13 more rows
```

Be careful with column names and **filter**

```
filter(er_30, "county" == "Denver") # this will not work  
  
# A tibble: 0 × 6  
#   county    rate lower95cl upper95cl visits year  
#   <chr>     <dbl>    <dbl>    <dbl>    <dbl>  <dbl>
```

Be careful with column names and **filter**

```
filter(er_30, county == "Denver")# this works!
```

```
# A tibble: 1 × 6
  county   rate lower95cl upper95cl visits   year
  <chr>    <dbl>    <dbl>     <dbl>   <dbl>   <dbl>
1 Denver    2.95     1.75     4.46     19    2013
```

filter() is tricky

Try not use anything special for the column names in `filter()`. This is why it is good to not use atypical column names. Then you can just use the column name!

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>

GUT CHECK: If we want to keep just rows that meet either or two conditions, what code should we use?

- A. `filter()` with `|`
- B. `filter()` with `&`

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"))))` (see extra slides at the end for more info!)
- `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()`

Lab Part 2

- [Class Website](#)
- [Lab](#)
- [Day 3 Cheatsheet](#)
- [Posit's dplyr Cheatsheet](#)

Get the data

```
#er <- read_csv("https://daseh.org/data/CO_ER_heat_visits.csv")
set.seed(1234)
er_30 <- slice_sample(er, n = 30)
```

Combining `filter` and `select`

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

```
select(filter(er_30, year > 2012), county)
```

```
# A tibble: 25 × 1
```

```
  county
```

```
  <chr>
```

```
1 Garfield
```

```
2 Chaffee
```

```
3 Pueblo
```

```
4 Rio Grande
```

```
5 Lake
```

```
6 Chaffee
```

```
7 Teller
```

```
8 Boulder
```

```
9 Fremont
```

```
10 Kiowa
```

```
# ... 15 more rows
```

Nesting

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

```
head(select(er_30, year, county), 2)
```

```
# A tibble: 2 × 2
  year county
  <dbl> <chr>
1 2018 Garfield
2 2015 Chaffee
```

Nesting can get confusing looking

```
select(filter(er_30, year > 2000 & county == "Denver"), year, rate)
```

```
# A tibble: 1 × 2
```

```
  year   rate  
  <dbl> <dbl>  
1 2013   2.95
```

Assigning Temporary Objects

One can also create temporary objects and reassign them:

```
er_30_Den <- filter(er_30, year > 2000 & county == "Denver")
er_30_Den <- select(er_30_Den, year, rate)

head(er_30_Den)

# A tibble: 1 × 2
  year    rate
  <dbl>   <dbl>
1 2013     2.95
```

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 `er_30` into `filter`, then pipe that into `select`:

```
er_30 %>% filter(year > 2000 & county == "Denver") %>% select(year, rate)
```

```
# A tibble: 1 × 2
  year    rate
  <dbl>   <dbl>
1 2013     2.95
```

Alternative Pipes

There are multiple ways to write a pipe and you might see these (they work the same!):

|>

%>%

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})
```

```
er_30 <- mutate(er_30, newcol = rate * 2)  
head(er_30, 4)
```

```
# A tibble: 4 × 7  
county      rate lower95cl upper95cl visits   year newcol  
<chr>     <dbl>    <dbl>     <dbl>  <dbl> <dbl>  <dbl>  
1 Garfield    NA       NA        NA      NA  2018    NA  
2 Chaffee     NA       NA        NA      NA  2015    NA  
3 Pueblo      15.9    9.54     22.2    26  2021   31.8  
4 Rio Grande   0        0         0       0  2019    0
```

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})
```

```
er_30 <- mutate(er_30, newcol = newcol / 2)
head(er_30, 4)
```

```
# A tibble: 4 × 7
  county      rate lower95cl upper95cl visits year newcol
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl>   <dbl>
1 Garfield     NA       NA        NA      NA  2018     NA
2 Chaffee      NA       NA        NA      NA  2015     NA
3 Pueblo      15.9     9.54     22.2     26  2021    15.9
4 Rio Grande    0        0         0       0  2019     0
```

You can pipe data into mutate

```
er_30 <- er_30 %>% mutate(newcol = newcol / 2)  
head(er_30, 4)
```

```
# A tibble: 4 × 7  
  county      rate lower95cl upper95cl visits year newcol  
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl>    <dbl>  
1 Garfield     NA       NA        NA      NA  2018     NA  
2 Chaffee      NA       NA        NA      NA  2015     NA  
3 Pueblo      15.9     9.54     22.2     26  2021     7.95  
4 Rio Grande    0        0         0       0  2019     0
```

mutate function



"Artwork by @allison_horst". <https://allisonhorst.com/>

Removing columns of a data frame: dplyr

The `NULL` method is still very common.

The `select` function can remove a column with exclamation mark (!) or using the minus sign (-):

```
select(er_30, !newcol)
```

```
# A tibble: 6 × 6
  county      rate lower95cl upper95cl visits   year
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl>
1 Garfield     NA       NA        NA       NA   2018
2 Chaffee      NA       NA        NA       NA   2015
3 Pueblo      15.9     9.54     22.2      26  2021
4 Rio Grande    0        0         0        0   2019
5 Lake          0        0         0        0   2014
6 Chaffee      NA       NA        NA       NA   2012
```

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 `year`:

```
select(er_30, !c(newcol, year))
```

A tibble: 30 × 5

	county	rate	lower95cl	upper95cl	visits
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	Garfield	NA	NA	NA	NA
2	Chaffee	NA	NA	NA	NA
3	Pueblo	15.9	9.54	22.2	26
4	Rio Grande	0	0	0	0
5	Lake	0	0	0	0
6	Chaffee	NA	NA	NA	NA
7	Chaffee	NA	NA	NA	NA
8	Teller	0	0	0	0
9	Prowers	NA	NA	NA	NA
10	Hinsdale	0	0	0	0
# 20 more rows					

Ordering columns

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns.

```
head(er_30, 2)
```

```
# A tibble: 2 × 7
  county    rate lower95cl upper95cl visits   year newcol
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl>   <dbl>
1 Garfield    NA       NA        NA      NA  2018    NA
2 Chaffee     NA       NA        NA      NA  2015    NA
```

```
er_30 %>% select(year, rate, county) %>%
head(2)
```

```
# A tibble: 2 × 3
  year    rate county
  <dbl>   <dbl> <chr>
1 2018     NA Garfield
2 2015     NA Chaffee
```

Ordering the columns of a data frame: dplyr

The `select` function can reorder columns. Put `newcol1` first, then select the rest of columns:

```
select(er_30, newcol, everything())
```

```
# A tibble: 3 × 7
  newcol county    rate lower95cl upper95cl visits year
  <dbl> <chr>     <dbl>     <dbl>     <dbl>   <dbl> <dbl>
1 NA     Garfield  NA       NA       NA       NA     2018
2 NA     Chaffee   NA       NA       NA       NA     2015
3 7.95   Pueblo    15.9     9.54    22.2     26    2021
```

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 with the `.before` and `.after` arguments.

For example, let say we just wanted `year` to be before `'rate'`.

```
head(er_30, 1)

# A tibble: 1 × 7
  county    rate lower95cl upper95cl visits   year newcol
  <chr>     <dbl>     <dbl>     <dbl>   <dbl> <dbl>   <dbl>
1 Garfield     NA       NA       NA      NA  2018     NA
```

```
er_year_fix <- relocate(er_30, year, .before = rate)
```

```
head(er_year_fix, 1)

# A tibble: 1 × 7
  county    year    rate lower95cl upper95cl visits newcol
  <chr>     <dbl>   <dbl>     <dbl>     <dbl>   <dbl>   <dbl>
1 Garfield  2018     NA       NA       NA      NA     NA
```

Ordering rows

Ordering the rows of a data frame: dplyr

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

```
arrange(er_30, year)
```

```
# A tibble: 30 × 7
  county      rate lower95cl upper95cl visits   year newcol
  <chr>     <dbl>    <dbl>    <dbl>    <dbl> <dbl>    <dbl>
1 Saguache     0        0        0        0    2011      0
2 Chaffee      NA       NA       NA       NA    2012     NA
3 Prowers      NA       NA       NA       NA    2012     NA
4 Hinsdale     0        0        0        0    2012      0
5 Phillips      0        0        0        0    2012      0
6 Denver       2.95    1.75    4.46    19    2013    1.47
7 San Miguel   NA       NA       NA       NA    2013     NA
8 Lake          0        0        0        0    2014      0
9 Delta          0        0        0        0    2014      0
10 Adams        3.08   1.71    4.85    15    2014    1.54
# ... 20 more rows
```

Ordering the rows of a data frame: dplyr

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

```
arrange(er_30, desc(year))
```

```
# A tibble: 30 × 7
  county      rate lower95cl upper95cl visits   year newcol
  <chr>     <dbl>    <dbl>    <dbl>    <dbl> <dbl>    <dbl>
1 Otero        NA       NA       NA       NA  2022     NA
2 Chaffee      0        0        0        0  2022     0
3 Pueblo      15.9     9.54    22.2     26  2021    7.95
4 Mesa         12.0     7.12    18.2     19  2021    6.01
5 Kiowa        NA       NA       NA       NA  2020     NA
6 Park          NA       NA       NA       NA  2020     NA
7 Rio Blanco   NA       NA       NA       NA  2020     NA
8 Rio Grande    0        0        0        0  2019     0
9 Garfield     NA       NA       NA       NA  2018     NA
10 Dolores     0        0        0        0  2018     0
# ℹ 20 more rows
```

Ordering the rows of a data frame: dplyr

You can combine increasing and decreasing orderings:

```
arrange(er_30, rate, desc(year)) %>% head(n = 2)
```

```
# A tibble: 2 × 7
```

	county	rate	lower95cl	upper95cl	visits	year	newcol
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Chaffee	0	0	0	0	2022	0
2	Rio Grande	0	0	0	0	2019	0

```
arrange(er_30, desc(year), rate) %>% head(n = 2)
```

```
# A tibble: 2 × 7
```

	county	rate	lower95cl	upper95cl	visits	year	newcol
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Chaffee	0	0	0	0	2022	0
2	Otero	NA	NA	NA	NA	2022	NA

GUT CHECK: What function would be useful for changing a column to be a percentage instead of a ratio?

- A. `filter()`
- B. `select()`
- C. `mutate()`

GUT CHECK: How would we interpret `er_30 %>% filter(year > 2020)`

`%>% select(year, rate)?`

- A. Get the `er_30` data, then filter it for rows with `year` values over 2020, then select only the `year` and `rate` columns.
- B. Get the `er_30` data, then filter it rows with `year` values over 2020, then select for rows that have values for `year` and `rate`.

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 exclamation mark in front of column name(s)
 2. not selecting it (without exclamation mark)

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})
```

```
er_30 <- mutate(ER_30, newcol = count/2.2)
```

A note about base R:

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

```
er_30$year
```

```
[1] 2018 2015 2021 2019 2014 2012 2017 2016 2012 2012 2017 2016 2020 2014 2020  
[16] 2014 2011 2022 2018 2013 2017 2021 2015 2020 2012 2016 2013 2014 2017 2022
```

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.

```
er_30$newcol <- er_30$rate/2.2  
head(er_30, 3)
```

```
# A tibble: 3 × 7  
  county    rate lower95cl upper95cl visits year newcol  
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>   <dbl>  
1 Garfield    NA       NA        NA      NA  2018     NA  
2 Chaffee     NA       NA        NA      NA  2015     NA  
3 Pueblo     15.9     9.54     22.2     26  2021    7.22
```

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](#)
- [Day 3 Cheatsheet](#)
- [Posit's dplyr Cheatsheet](#)



Image by [Gerd Altmann](#) from [Pixabay](#)

Extra Slides

Multiple patterns with tidyselect

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

```
select(er_30, ends_with("cl"), starts_with("r"))
```

```
# A tibble: 30 × 3
  lower95cl upper95cl   rate
  <dbl>     <dbl>    <dbl>
1      NA       NA     NA
2      NA       NA     NA
3     9.54     22.2   15.9
4      0        0      0
5      0        0      0
6      NA       NA     NA
7      NA       NA     NA
8      0        0      0
9      NA       NA     NA
10     0        0      0
# ... 20 more rows
```

```
select(er_30, starts_with(c("r", "l"))) # here we combine two patterns
```

```
# A tibble: 30 × 2
  rate lower95cl
  <dbl>    <dbl>
1     NA      NA
2     NA      NA
3    15.9     9.54
4      0       0
5      0       0
```

Nuances about `filter()`

```
test <- tibble(A = c(1, 2, 3, 4), B = c(1, 2, 3, 4))  
test
```

```
# A tibble: 4 × 2
```

	A	B
	<dbl>	<dbl>
1	1	1
2	2	2
3	3	3
4	4	4

These are technically the same but `>=` is easier to read

Separating can cause issues
`filter(test, B > 2 | B==2)`

```
# A tibble: 3 × 2
```

	A	B
	<dbl>	<dbl>
1	2	2
2	3	3
3	4	4

```
filter(test, B >= 2)
```

```
# A tibble: 3 × 2
```

	A	B
	<dbl>	<dbl>
1	2	2

Order of operations for `filter()`

Order can matter. Think of individual statements separately first.

```
filter(test, A>3 | B==2 & B>2) # A is greater than 3 or B is equal to 2 AND (th
```

```
# A tibble: 1 × 2
  A     B
  <dbl> <dbl>
1     4     4
```

```
filter(test, A>3 & B>2 | B==2) # A is greater than 3 AND B is greater than 2 OR B
```

```
# A tibble: 2 × 2
  A     B
  <dbl> <dbl>
1     2     2
2     4     4
```

Ordering the column names of a data frame: alphabetically

Using the base R `order()` function.

```
order(colnames(er_30))
```

```
[1] 1 3 7 2 4 5 6
```

```
er_30 %>% select(order(colnames(er_30)))
```

```
# A tibble: 30 × 7
  county    lower95cl newcol   rate upper95cl visits   year
  <chr>      <dbl>   <dbl> <dbl>     <dbl>   <dbl> <dbl>
1 Garfield     NA     NA     NA       NA     NA  2018
2 Chaffee      NA     NA     NA       NA     NA  2015
3 Pueblo        9.54   7.22  15.9     22.2   26  2021
4 Rio Grande    0      0      0       0      0  2019
5 Lake          0      0      0       0      0  2014
6 Chaffee      NA     NA     NA       NA     NA  2012
7 Chaffee      NA     NA     NA       NA     NA  2017
8 Teller         0      0      0       0      0  2016
9 Prowers       NA     NA     NA       NA     NA  2012
10 Hinsdale     0      0      0       0      0  2012
# ... 20 more rows
```

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(er_30, year) == 2012)
```

```
[1] 6 9 10 25
```

```
select(er_30, year) == 2012 %>% head(10)
```

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

Remove a column in base R

```
er_30$year <- 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(er_30) # just prints
```

```
[1] "county"      "rate"        "lower95cl"    "upper95cl"   "visits"      "year"  
[7] "newcol"
```

```
colnames(er_30)[1:2] <- c("County", "Rate") # reassigned  
head(er_30)
```

```
# A tibble: 6 × 7  
  County      Rate lower95cl upper95cl visits year newcol  
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl>    <dbl>  
1 Garfield     NA       NA        NA       NA  2018     NA  
2 Chaffee      NA       NA        NA       NA  2015     NA  
3 Pueblo      15.9     9.54     22.2     26  2021     7.22  
4 Rio Grande    0        0         0        0  2019      0  
5 Lake          0        0         0        0  2014      0  
6 Chaffee      NA       NA        NA       NA  2012     NA
```

Subset rows of a data frame with indices:

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

```
er_30[ c(1, 3), ]
```

```
# A tibble: 2 × 7
  County    Rate lower95cl upper95cl visits year newcol
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl>   <dbl>
1 Garfield    NA       NA        NA      NA  2018     NA
2 Pueblo     15.9     9.54     22.2     26  2021    7.22
```

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 ("year")

```
er_30[, 6]
```

```
# A tibble: 30 × 1
  year
  <dbl>
1 2018
2 2015
3 2021
4 2019
5 2014
6 2012
7 2017
8 2016
9 2012
10 2012
# ... 20 more rows
```

```
er_30[, "year"]
```

```
# A tibble: 30 × 1
  year
  <dbl>
```

Subset columns of a data frame:

We can select multiple columns using multiple column names:

```
er_30[, c("County", "Rate")]
```

```
# A tibble: 30 × 2
  County      Rate
  <chr>     <dbl>
1 Garfield    NA
2 Chaffee     NA
3 Pueblo     15.9
4 Rio Grande   0
5 Lake        0
6 Chaffee     NA
7 Chaffee     NA
8 Teller      0
9 Prowers     NA
10 Hinsdale    0
# ... 20 more rows
```