

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

□ [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!
```

```
— Attaching core tidyverse packages ————— tidyverse 2.0.0 —
 forcats 1.0.0    stringr  1.5.1
lubridate 1.9.3   tibble    3.2.1
purrr    1.0.2    tidyr    1.3.1
readr     2.1.5

— Conflicts ————— tidyverse_conflicts()
dplyr::filter() masks stats::filter()
dplyr::lag()   masks stats::lag()
Use the conflicted package (<http://conflicted.r-lib.org/>) to force all confl
```

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>.

```
library(dasehr)
ER <- CO_heat_ER_bygender
#or we could do this:
ER <-
  read_csv("https://daseh.org/data/Colorado_ER_heat_visits_by_county_gender.csv")
```

Rows: 240 Columns: 7

— Column specification —————

Delimiter: ","

chr (2): county, gender

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] 240    7
```

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

```
[1] 240
```

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

```
[1] 7
```

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: 240

Columns: 7

```
$ county    <chr> "Adams", "Adams", "Adams", "Adams", "Adams", "Adams", "Adams"...
$ rate      <dbl> 7.596958, NA, 6.218501, NA, NA, 6.161907, 4.685828, 6.386427...
$ lower95cl <dbl> 4.383039, NA, 3.367822, NA, NA, 3.350234, 2.314647, 3.619401...
$ upper95cl <dbl> 11.680261, NA, 9.928823, NA, NA, 9.816068, 7.879132, 9.92620...
$ visits    <dbl> 17, NA, 14, NA, NA, 14, 11, 16, 17, NA, 19, 18, 12, 13, 17, ...
$ year      <dbl> 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, ...
$ gender    <chr> "Female", "Female", "Female", "Female", "Female", "Female", ...
```

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 × 7
  county      rate lower95cl upper95cl visits year gender
  <chr>     <dbl>    <dbl>    <dbl>   <dbl> <dbl> <chr>
1 Arapahoe  7.18     4.04    10.3     21   2016 Male
2 Weld       NA       NA       NA       NA   2014 Female
```

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

```
# A tibble: 2 × 7
  county      rate lower95cl upper95cl visits year gender
  <chr>     <dbl>    <dbl>    <dbl>   <dbl> <dbl> <chr>
1 Cheyenne    0        0        0        0   2012 Male
2 Jefferson  4.15    2.01    7.04    11   2011 Female
```

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: 240 × 7
  county   rate lower95cl upper95cl visits   year gender
  <chr>   <dbl>    <dbl>     <dbl>   <dbl>   <dbl> <chr>
1 Adams    7.60     4.38     11.7     17    2011 Female
2 Adams    NA        NA       NA       NA    2012 Female
3 Adams    6.22     3.37     9.93     14    2013 Female
4 Adams    NA        NA       NA       NA    2014 Female
5 Adams    NA        NA       NA       NA    2015 Female
6 Adams    6.16     3.35     9.82     14    2016 Female
7 Adams    4.69     2.31     7.88     11    2017 Female
8 Adams    6.39     3.62     9.93     16    2018 Female
9 Adams    6.64     3.85     10.2     17    2019 Female
10 Adams   NA        NA       NA       NA   2020 Female
# ℹ 230 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 × 7
  county   rate lower95cl upper95cl visits year gender
  <chr>    <dbl>    <dbl>    <dbl>    <dbl> <dbl> <chr>
1 Adams     7.60     4.38    11.7      17  2011 Female
2 Adams     NA        NA       NA        NA  2012 Female
3 Adams     6.22     3.37    9.93      14  2013 Female
4 Adams     NA        NA       NA        NA  2014 Female
5 Adams     NA        NA       NA        NA  2015 Female
6 Adams     6.16     3.35    9.82      14  2016 Female
```

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 (or **names()**)

```
colnames(ER)
```

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

Renaming Columns of a data frame or tibble

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

For example, 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!

```
# 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 × 7
  county   rate lower_95_CI_limit upper95cl visits   year gender
  <chr>   <dbl>           <dbl>      <dbl>  <dbl> <dbl> <chr>
1 Adams     7.60            4.38      11.7    17   2011 Female
2 Adams     NA              NA        NA       NA   2012 Female
3 Adams     6.22            3.37      9.93    14   2013 Female
4 Adams     NA              NA        NA       NA   2014 Female
5 Adams     NA              NA        NA       NA   2015 Female
6 Adams     6.16            3.35      9.82    14   2016 Female
```

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 × 7
  county   rate `lower_95%_CI_limit` upper95cl visits year gender
  <chr>    <dbl>           <dbl>       <dbl>   <dbl> <chr>
1 Adams     7.60            4.38        11.7    17  2011 Female
2 Adams     NA              NA          NA      NA  2012 Female
```

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' = lower95c1)
```

```
# this will work!
renamed_ER <- rename(ER, 'lower_95%_CI_limit' = lower95c1)
```

Also true for double quotes

```
# this will cause an error!
renamed_ER <- rename(ER, "lower_95%_CI_limit" = lower95c1)
```

```
# this will work!
renamed_ER <- rename(ER, "lower_95%_CI_limit" = lower95c1)
```

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 × 7
  county   rate lower_95perc_CI_limit upper_95perc_CI_limit visits   year gender
  <chr>    <dbl>            <dbl>                <dbl>    <dbl> <dbl> <chr>
1 Adams     7.60             4.38                11.7      17  2011 Female
2 Adams     NA                  NA                  NA        NA  2012 Female
3 Adams     6.22             3.37                9.93      14  2013 Female
```

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 × 7
```

```
COUNTY RATE LOWER95CL UPPER95CL VISITS YEAR GENDER
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 Adams 7.60 4.38 11.7 17 2011 Female
2 Adams NA NA NA NA 2012 Female
3 Adams 6.22 3.37 9.93 14 2013 Female
```

```
rename_with(ER_upper, tolower)
```

```
# A tibble: 240 × 7
```

```
county rate lower95cl upper95cl visits year gender
<chr> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
1 Adams 7.60 4.38 11.7 17 2011 Female
2 Adams NA NA NA NA 2012 Female
3 Adams 6.22 3.37 9.93 14 2013 Female
4 Adams NA NA NA NA 2014 Female
5 Adams NA NA NA NA 2015 Female
6 Adams 6.16 3.35 9.82 14 2016 Female
7 Adams 4.69 2.31 7.88 11 2017 Female
8 Adams 6.39 3.62 9.93 16 2018 Female
9 Adams 6.64 3.85 10.2 17 2019 Female
10 Adams NA NA NA NA 2020 Female
# 230 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. Here it make sure year names aren't just a number, so that the colnames don't need ticks or quotes to be used.

```
#library(dasehr)
CO2 <- dasehr::yearly_co2_emissions
# or this:
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

```
head(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 ...
# □ 254 more variables: `1761` <dbl>, `1762` <dbl>, `1763` <dbl>, `1764` <dbl>,
#   `1765` <dbl>, `1766` <dbl>, `1767` <dbl>, `1768` <dbl>, `1769` <dbl>31/99
```

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
```

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](#)

Subsetting Columns

Let's get our data 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.

```
#library(dasehr)
#ER <- CO_heat_ER_bygender
#or this:
#read_csv("https://daseh.org/data/Colorado_ER_heat_visits_by_county_gender.csv")
set.seed(1234)
ER_30 <- slice_sample(ER, n = 30)
```

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

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

```
pull(ER_30, year)
```

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

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 2014
2 2018
3 2016
4 2015
5 2018
6 2013
7 2015
8 2011
9 2020
10 2022
# ... 20 more rows
```

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 2014     NA Arapahoe
2 2018     3.76 Denver
3 2016     8.13 Larimer
4 2015     3.94 El Paso
5 2018    11.1  Weld
6 2013     5.99 El Paso
7 2015     4.12 Jefferson
8 2011     5.76 Jefferson
9 2020     NA Larimer
10 2022    5.01 Jefferson
# ... 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 × 7
  county    rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>     <dbl>     <dbl>   <dbl> <dbl> <chr>
1 Arapahoe    NA       NA       NA      NA  2014 Female
2 Denver      3.76     1.97     6.12    13  2018 Female
```

```
select(ER_30, ends_with("c1"))
```

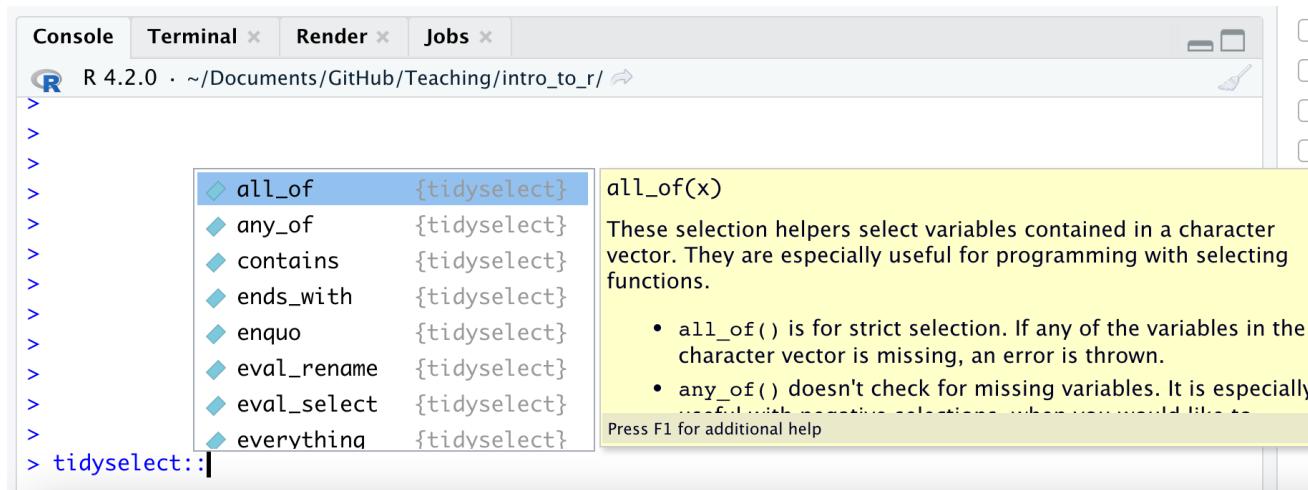
```
# A tibble: 30 × 2
  lower95cl upper95cl
  <dbl>     <dbl>
1 NA        NA
2 1.97      6.12
3 4.32      13.1
4 2.01      6.52
5 6.59      16.9
6 3.32      8.66
7 2.02      6.94
8 3.12      9.20
9 NA        NA
10 2.71     7.99
# ... 20 more rows
```

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 × 7
  county    rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>     <dbl>     <dbl>   <dbl> <dbl> <chr>
1 Arapahoe    NA       NA       NA      NA  2014 Female
2 Denver      3.76    1.97    6.12     13  2018 Female
```

```
select(ER_30, ends_with("c1"), year)
```

```
# A tibble: 30 × 3
  lower95cl upper95cl year
  <dbl>     <dbl> <dbl>
1 NA        NA    2014
2 1.97     6.12  2018
3 4.32     13.1   2016
4 2.01     6.52   2015
5 6.59     16.9   2018
6 3.32     8.66   2013
7 2.02     6.94   2015
8 3.12     9.20   2011
9 NA        NA    2020
10 2.71    7.99   2022
# ... 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 1.97        6.12       3.76
3 4.32        13.1       8.13
4 2.01        6.52       3.94
5 6.59        16.9       11.1
6 3.32        8.66       5.99
7 2.02        6.94       4.12
8 3.12        9.20       5.76
9 NA          NA          NA
10 2.71       7.99       5.01
# ... 20 more rows
```

Multiple patterns with tidyselect

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

```
select(ER_30, starts_with(c("r", "l")))
```

```
# A tibble: 30 × 2
  rate lower95cl
  <dbl>    <dbl>
1 NA        NA
2 3.76     1.97
3 8.13     4.32
4 3.94     2.01
5 11.1      6.59
6 5.99     3.32
7 4.12     2.02
8 5.76     3.12
9 NA        NA
10 5.01    2.71
# ... 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 × 7
  county      rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl> <chr>
1 Arapahoe     NA        NA        NA        NA  2014 Female
2 Denver      3.76     1.97     6.12     13  2018 Female
```

```
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  2014
2 3.76     1.97     6.12     13  2018
3 8.13     4.32     13.1     14  2016
4 3.94     2.01     6.52     12  2015
5 11.1     6.59     16.9     18  2018
6 5.99     3.32     8.66     20  2013
7 4.12     2.02     6.94     11  2015
8 5.76     3.12     9.20     14  2011
9 NA        NA        NA        NA  2020
10 5.01    2.71     7.99     14  2022
# ... 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: 2 × 7
  county      rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl> <chr>
1 Jefferson  5.01     2.71      7.99     14  2022 Male
2 Jefferson  4.56     2.43      7.35     14  2022 Female
```

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: 1 × 7
  county   rate lower95cl upper95cl visits  year gender
  <chr>    <dbl>     <dbl>     <dbl>   <dbl> <dbl> <chr>
1 Weld      12.8      6.96     20.3     15  2012 Male
```

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: 9 × 7
  county      rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl> <chr>
1 Weld       11.1     6.59    16.9      18    2018 Male 
2 El Paso    NA        NA       NA       NA    2012 Female
3 Statewide  7.56     6.48    8.65     193   2012 Male 
4 El Paso    9.07     5.74    12.4      31    2018 Male 
5 Adams      5.48     2.90    8.88      13    2012 Male 
6 Pueblo     18.5     9.91    29.7      14    2014 Male 
7 Cheyenne   0         0        0        0    2012 Male 
8 Weld       12.8     6.96    20.3      15    2012 Male 
9 Statewide  10.1     8.95    11.3     293   2018 Male
```

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", "Weld", "Arapahoe"))
```

```
# A tibble: 8 × 7
  county    rate lower95cl upper95cl visits year gender
  <chr>     <dbl>    <dbl>    <dbl>   <dbl> <dbl> <chr>
1 Arapahoe    NA       NA       NA      NA  2014 Female
2 Denver     3.76     1.97     6.12     13  2018 Female
3 Weld       11.1     6.59     16.9      18  2018 Male
4 Denver     8.51     5.30     11.7      30  2016 Male
5 Denver     3.56     1.82     5.88      13  2017 Female
6 Weld       12.8     6.96     20.3      15  2012 Male
7 Arapahoe    NA       NA       NA      NA  2014 Male
8 Weld       NA       NA       NA      NA  2013 Female
```

```
filter(ER_30, county == "Denver" | county == "Weld" | county == "Arapahoe") #equivalent
```

```
# A tibble: 8 × 7
  county    rate lower95cl upper95cl visits year gender
  <chr>     <dbl>    <dbl>    <dbl>   <dbl> <dbl> <chr>
1 Arapahoe    NA       NA       NA      NA  2014 Female
2 Denver     3.76     1.97     6.12     13  2018 Female
3 Weld       11.1     6.59     16.9      18  2018 Male
4 Denver     8.51     5.30     11.7      30  2016 Male
5 Denver     3.56     1.82     5.88      13  2017 Female
6 Weld       12.8     6.96     20.3      15  2012 Male
```

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, 2020), county %in% c("Denver", "Weld"))
```

```
# A tibble: 1 × 7
  county    rate lower95cl upper95cl visits   year gender
  <chr>    <dbl>     <dbl>     <dbl>   <dbl> <dbl> <chr>
1 Weld      NA        NA        NA      NA  2013 Female
```

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 × 7
  county      rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl> <chr>
1 Arapahoe     NA       NA       NA       NA     2014 Female
2 Denver      3.76     1.97     6.12     13     2018 Female
3 Larimer      8.13     4.32    13.1      14     2016 Female
4 El Paso      3.94     2.01     6.52     12     2015 Female
5 Weld        11.1     6.59    16.9      18     2018 Male 
6 El Paso      5.99     3.32     8.66     20     2013 Male 
7 Jefferson    4.12     2.02     6.94     11     2015 Male 
8 Jefferson    5.76     3.12     9.20     14     2011 Male 
9 Larimer      NA       NA       NA       NA     2020 Male 
10 Jefferson    5.01    2.71     7.99     14    2022 Male 
# ℹ 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 × 7  
  county      rate lower95cl upper95cl visits `year!` gender  
  <chr>     <dbl>    <dbl>    <dbl>    <dbl>    <dbl> <chr>  
1 Arapahoe    NA       NA       NA       NA       2014 Female  
2 Denver      3.76    1.97    6.12     13      2018 Female  
3 Larimer     8.13    4.32    13.1     14      2016 Female  
4 El Paso     3.94    2.01    6.52     12      2015 Female  
5 Weld        11.1    6.59    16.9     18      2018 Male  
6 El Paso     5.99    3.32    8.66     20      2013 Male  
7 Jefferson   4.12    2.02    6.94     11      2015 Male  
8 Jefferson   5.76    3.12    9.20     14      2011 Male  
9 Larimer     NA       NA       NA       NA       2020 Male  
10 Jefferson  5.01    2.71    7.99     14     2022 Male  
# ... 20 more rows
```

Be careful with column names and **filter**

Using backticks works!

```
filter(ER_30_rename, `year!` > 2013)
```

```
# A tibble: 21 × 7
  county      rate lower95cl upper95cl visits `year!` gender
  <chr>     <dbl>    <dbl>     <dbl>    <dbl>    <dbl> <chr>
1 Arapahoe     NA       NA        NA       NA     2014 Female
2 Denver      3.76     1.97     6.12     13     2018 Female
3 Larimer      8.13     4.32     13.1     14     2016 Female
4 El Paso      3.94     2.01     6.52     12     2015 Female
5 Weld         11.1     6.59     16.9     18     2018 Male
6 Jefferson    4.12     2.02     6.94     11     2015 Male
7 Larimer      NA       NA        NA       NA     2020 Male
8 Jefferson    5.01     2.71     7.99     14     2022 Male
9 Jefferson    4.56     2.43     7.35     14     2022 Female
10 El Paso      NA       NA        NA       NA    2017 Female
# ℹ 11 more rows
```

Be careful with column names and **filter**

```
filter(ER_30, "county" == "Denver") # this will not work  
  
# A tibble: 0 × 7  
#   county    rate lower95cl upper95cl visits year gender  
#   <chr>     <dbl>    <dbl>    <dbl>    <dbl>  <dbl> <chr>
```

Be careful with column names and **filter**

```
filter(ER_30, county == "Denver")# this works!  
  
# A tibble: 3 × 7  
  county    rate lower95cl upper95cl visits   year gender  
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>  
1 Denver    3.76     1.97      6.12     13  2018 Female  
2 Denver    8.51     5.30     11.7      30  2016 Male  
3 Denver    3.56     1.82      5.88     13  2017 Female
```

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>

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()`

Lab Part 2

- [Class Website](#)
- [Lab](#)

Get the data

```
#library(dasehr)
#ER <- CO_heat_ER_bygender
# or this:
#read_csv("https://daseh.org/data/Colorado_ER_heat_visits_by_county_gender.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: 24 × 1
```

```
  county  
  <chr>
```

```
1 Arapahoe  
2 Denver  
3 Larimer  
4 El Paso  
5 Weld  
6 El Paso  
7 Jefferson  
8 Larimer  
9 Jefferson  
10 Jefferson  
# ... 14 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 2014 Arapahoe
2 2018 Denver
```

Nesting can get confusing looking

```
select(filter(ER_30, year > 2000 & county == "Denver"), year, rate)
```

```
# A tibble: 3 × 2
```

```
  year    rate  
  <dbl> <dbl>  
1 2018    3.76  
2 2016    8.51  
3 2017    3.56
```

Assigning Temporary Objects

One can also create temporary objects and reassign them:

```
ER_30_CO <- filter(ER_30, year > 2000 & county == "Denver")  
ER_30_CO <- select(ER_30_CO, year, rate)
```

```
head(ER_30_CO)
```

```
# A tibble: 3 × 2  
  year    rate  
  <dbl>   <dbl>  
1 2018     3.76  
2 2016     8.51  
3 2017     3.56
```

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`:

```
ER_30 %>% filter(year > 2000 & county == "Denver") %>% select(year, rate)
```

```
# A tibble: 3 × 2
  year    rate
  <dbl>   <dbl>
1 2018    3.76
2 2016    8.51
3 2017    3.56
```

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 × 8
  county    rate lower95cl upper95cl visits   year gender newcol
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl> <chr>    <dbl>
1 Arapahoe    NA       NA        NA       NA   2014 Female    NA
2 Denver      3.76     1.97     6.12     13   2018 Female    7.53
3 Larimer     8.13     4.32     13.1     14   2016 Female   16.3
4 El Paso     3.94     2.01     6.52     12   2015 Female    7.89
```

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 × 8
  county    rate lower95cl upper95cl visits year gender newcol
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
1 Arapahoe    NA       NA        NA      NA  2014 Female    NA
2 Denver      3.76     1.97     6.12     13  2018 Female    3.76
3 Larimer     8.13     4.32    13.1      14  2016 Female    8.13
4 El Paso     3.94     2.01     6.52     12  2015 Female    3.94
```

You can pipe data into mutate

```
ER_30 <- ER_30 %>% mutate(newcol = newcol / 2)
head(ER_30, 4)
```

```
# A tibble: 4 × 8
  county      rate lower95cl upper95cl visits year gender newcol
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
1 Arapahoe     NA       NA        NA      NA  2014 Female    NA
2 Denver      3.76     1.97      6.12     13  2018 Female    1.88
3 Larimer     8.13     4.32     13.1      14  2016 Female    4.07
4 El Paso     3.94     2.01      6.52     12  2015 Female    1.97
```

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 × 7
  county    rate lower95cl upper95cl visits   year gender
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl> <chr>
1 Arapahoe    NA       NA        NA       NA    2014 Female
2 Denver      3.76     1.97     6.12     13    2018 Female
3 Larimer     8.13     4.32    13.1      14    2016 Female
4 El Paso     3.94     2.01     6.52     12    2015 Female
5 Weld        11.1     6.59    16.9      18    2018 Male
6 El Paso     5.99     3.32     8.66     20    2013 Male
```

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 × 6

	county	rate	lower95cl	upper95cl	visits	gender
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	Arapahoe	NA	NA	NA	NA	Female
2	Denver	3.76	1.97	6.12	13	Female
3	Larimer	8.13	4.32	13.1	14	Female
4	El Paso	3.94	2.01	6.52	12	Female
5	Weld	11.1	6.59	16.9	18	Male
6	El Paso	5.99	3.32	8.66	20	Male
7	Jefferson	4.12	2.02	6.94	11	Male
8	Jefferson	5.76	3.12	9.20	14	Male
9	Larimer	NA	NA	NA	NA	Male
10	Jefferson	5.01	2.71	7.99	14	Male
# 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 × 8
  county    rate lower95cl upper95cl visits   year gender newcol
  <chr>     <dbl>    <dbl>     <dbl>    <dbl> <dbl> <chr>    <dbl>
1 Arapahoe    NA       NA        NA       NA  2014 Female    NA
2 Denver      3.76    1.97      6.12      13  2018 Female    1.88
```

```
ER_30 %>% select(year, rate, county) %>%
head(2)
```

```
# A tibble: 2 × 3
  year    rate county
  <dbl>   <dbl> <chr>
1 2014    NA    Arapahoe
2 2018    3.76  Denver
```

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 × 8
  newcol county    rate lower95cl upper95cl visits   year gender
  <dbl> <chr>     <dbl>     <dbl>     <dbl>    <dbl> <dbl> <chr>
1    NA Arapahoe    NA        NA        NA      NA  2014 Female
2  1.88 Denver     3.76     1.97     6.12     13  2018 Female
3  4.07 Larimer    8.13     4.32    13.1      14  2016 Female
```

Ordering the columns of a data frame: dplyr

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

```
select(ER_30, !year, everything(), year)

# A tibble: 3 × 8
  county      rate lower95cl upper95cl visits gender newcol    year
  <chr>     <dbl>    <dbl>     <dbl>   <dbl> <chr>    <dbl> <dbl>
1 Arapahoe     NA       NA        NA      NA Female     NA    2014
2 Denver      3.76     1.97      6.12     13 Female    1.88  2018
3 Larimer     8.13     4.32     13.1      14 Female    4.07  2016
```

Ordering the column names of a data frame: alphabetically

Using the base R `order()` function.

```
order(colnames(ER_30))
```

```
[1] 1 7 3 8 2 4 5 6
```

```
ER_30 %>% select(order(colnames(ER_30)))
```

```
# A tibble: 30 × 8
  county    gender lower95cl newcol   rate upper95cl visits   year
  <chr>     <chr>    <dbl>    <dbl> <dbl>    <dbl>    <dbl>    <dbl>
1 Arapahoe Female     NA      NA      NA      NA      NA      2014
2 Denver    Female    1.97    1.88    3.76    6.12    13      2018
3 Larimer   Female    4.32    4.07    8.13    13.1    14      2016
4 El Paso   Female    2.01    1.97    3.94    6.52    12      2015
5 Weld      Male      6.59    5.57    11.1    16.9    18      2018
6 El Paso   Male      3.32    3.00    5.99    8.66    20      2013
7 Jefferson Male      2.02    2.06    4.12    6.94    11      2015
8 Jefferson Male      3.12    2.88    5.76    9.20    14      2011
9 Larimer   Male      NA      NA      NA      NA      NA      2020
10 Jefferson Male     2.71    2.50    5.01    7.99    14      2022
# ℒ 20 more rows
```

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 `year` to be before `BUYER_STATE`.

```
head(ER_30, 1)

# A tibble: 1 × 8
  county    rate lower95cl upper95cl visits   year gender newcol
  <chr>     <dbl>     <dbl>     <dbl> <dbl> <dbl> <chr>    <dbl>
1 Arapahoe     NA       NA       NA     NA   2014 Female     NA

tb_carb <- relocate(ER_30, year, .before = rate)

head(tb_carb, 1)

# A tibble: 1 × 8
  county    year    rate lower95cl upper95cl visits gender newcol
  <chr>     <dbl>   <dbl>     <dbl>     <dbl> <dbl> <chr>    <dbl>
1 Arapahoe  2014     NA       NA       NA     NA Female     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 × 8
  county      rate lower95cl upper95cl visits   year gender newcol
  <chr>     <dbl>    <dbl>    <dbl>    <dbl> <dbl> <chr>    <dbl>
1 Jefferson  5.76     3.12     9.20     14  2011 Male     2.88
2 El Paso    NA        NA       NA        NA  2012 Female   NA
3 Statewide   7.56     6.48     8.65     193 2012 Male     3.78
4 Adams      5.48     2.90     8.88     13  2012 Male     2.74
5 Cheyenne    0         0         0         0  2012 Male     0
6 Weld        12.8     6.96    20.3      15  2012 Male     6.38
7 El Paso    5.99     3.32     8.66     20  2013 Male     3.00
8 Weld        NA        NA       NA        NA  2013 Female   NA
9 Statewide   4.94     4.06     5.82     124 2013 Female   2.47
10 Arapahoe   NA        NA       NA        NA  2014 Female   NA
# ℹ 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 × 8
```

	county	rate	lower95cl	upper95cl	visits	year	gender	newcol
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
1	Jefferson	5.01	2.71	7.99	14	2022	Male	2.50
2	Jefferson	4.56	2.43	7.35	14	2022	Female	2.28
3	Larimer	NA	NA	NA	NA	2020	Male	NA
4	Statewide	6.60	5.66	7.53	197	2020	Male	3.30
5	Cheyenne	0	0	0	0	2020	Male	0
6	Denver	3.76	1.97	6.12	13	2018	Female	1.88
7	Weld	11.1	6.59	16.9	18	2018	Male	5.57
8	El Paso	9.07	5.74	12.4	31	2018	Male	4.54
9	Statewide	10.1	8.95	11.3	293	2018	Male	5.06
10	El Paso	NA	NA	NA	NA	2017	Female	NA
# 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 × 8
```

```
county      rate lower95cl upper95cl visits year gender newcol
<chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
1 Cheyenne     0        0         0       0   2020 Male      0
2 Cheyenne     0        0         0       0   2012 Male      0
```

```
arrange(ER_30, desc(year), rate) %>% head(n = 2)
```

```
# A tibble: 2 × 8
```

```
county      rate lower95cl upper95cl visits year gender newcol
<chr>     <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>    <dbl>
1 Jefferson  4.56     2.43     7.35    14   2022 Female   2.28
2 Jefferson  5.01     2.71     7.99    14   2022 Male    2.50
```

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

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 × 8  
  county    rate lower95cl upper95cl visits year gender newcol  
  <chr>   <dbl>    <dbl>     <dbl>   <dbl> <dbl> <chr>   <dbl>  
1 Arapahoe    NA       NA        NA      NA  2014 Female    NA  
2 Denver      3.76    1.97      6.12     13  2018 Female    1.71  
3 Larimer     8.13    4.32     13.1      14  2016 Female    3.70
```

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](#)



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(ER_30, year) == 2014)
```

```
[1] 1 22 24 27
```

```
select(ER_30, year) == 2014 %>% head(10)
```

```
      year
[1, ] TRUE
[2, ] FALSE
[3, ] FALSE
[4, ] FALSE
[5, ] FALSE
[6, ] FALSE
[7, ] FALSE
[8, ] FALSE
[9, ] FALSE
[10, ] FALSE
[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] "gender"      "newcol"
```

```
colnames(ER_30)[1:2] <- c("County", "Rate") # reassigned  
head(ER_30)
```

```
# A tibble: 6 × 8
```

	County	Rate	lower95cl	upper95cl	visits	year	gender	newcol
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
1	Arapahoe	NA	NA	NA	NA	2014	Female	NA
2	Denver	3.76	1.97	6.12	13	2018	Female	1.71
3	Larimer	8.13	4.32	13.1	14	2016	Female	3.70
4	El Paso	3.94	2.01	6.52	12	2015	Female	1.79
5	Weld	11.1	6.59	16.9	18	2018	Male	5.07
6	El Paso	5.99	3.32	8.66	20	2013	Male	2.72

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 × 8
```

	County	Rate	lower95cl	upper95cl	visits	year	gender	newcol
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
1	Arapahoe	NA	NA	NA	NA	2014	Female	NA
2	Larimer	8.13	4.32	13.1	14	2016	Female	3.70

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 2014
2 2018
3 2016
4 2015
5 2018
6 2013
7 2015
8 2011
9 2020
10 2022
# ... 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 Arapahoe    NA
2 Denver      3.76
3 Larimer     8.13
4 El Paso     3.94
5 Weld        11.1
6 El Paso     5.99
7 Jefferson   4.12
8 Jefferson   5.76
9 Larimer     NA
10 Jefferson   5.01
# ... 20 more rows
```