

Intro to R

Subsetting Data in R

Recap

- Use `<-` to save (assign) values to objects
- Need to assign the output of the function to keep the result!
- Functions (like verbs) perform specific tasks in R and are found within **packages**
- Use `c()` to combine elements in a vectors
- `length()`, `class()`, and `str()` tell you information about an object
- Install packages with `install.packages()`
- Load packages with `library()`
- Get help with `?` or help pane
- `readr` has helpful functions like `read_csv()` that can help you import data into R like so: `df_example_readr <- read_csv(file = "documents/data_analysis/data_file.csv")` [Cheatsheet](#)

recap continued

```
head(iris)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

summarize() or **summarise()**

These functions are the same (just language preference).

- It creates a new dataframe with stats about the data.
- The columns are named whatever you want.
- It allows us to apply mathematical functions on columns of a dataframe and make a new dataframe with just that information.

```
Data %>% summarize(any_name = math_function(column))
```

```
iris %>% summarize(Mean_Petal_Length = mean(Petal.Length))
```

```
  Mean_Petal_Length
1             3.758
```

group_by()

It sets up your code so that all subsequent steps will parse the data based on what you use in group_by.

```
iris %>% group_by(Species) %>%  
  summarize(Mean_Petal_Length = mean(Petal.Length))
```

```
# A tibble: 3 × 2  
  Species      Mean_Petal_Length  
  <fct>          <dbl>  
1 setosa          1.46  
2 versicolor      4.26  
3 virginica        5.55
```

Overview

In this module, we will show you how to:

1. Look at your data in different ways
2. Rename columns of a data frame
3. Subset rows of a data frame
4. Subset columns of a data frame
5. Add new columns to 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/>

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 can take a look at the top of it by using the `head()` function.

```
state_data <- read_csv("https://hutchdatascience.org/SeattleStatSummer_R/data/states.csv")
head(state_data)
```

```
# A tibble: 6 × 14
  entity      state_abb state_area_sq_mil... state_division state_region population
  <chr>      <chr>          <dbl> <chr>          <chr>          <dbl>
1 Alabama    AL              51609 East South Ce... South          4903185
2 Alaska     AK              589757 Pacific        West           731545
3 Arizona    AZ              113909 Mountain      West           7278717
4 Arkansas   AR              53104 West South Ce... South          3017804
5 California CA              158693 Pacific        West          39512223
6 Colorado   CO              104247 Mountain      West           5758736
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,
# cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,
# cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,
# Series_Complete_Pop_Pct <dbl>
```

To see the bottom of the data use **tail()**

```
tail(state_data) # looking at the bottom 6 rows of the mtcars data
```

```
# A tibble: 6 × 14
  entity      state_abb state_area_sq_m... state_division state_region population
  <chr>      <chr>          <dbl> <chr>          <chr>          <dbl>
1 Washington WA              68192 Pacific        West            76148
2 West Virgin... WV              24181 South Atlantic South            17921
3 Wisconsin   WI              56154 East North Ce... North Centr... 58224
4 Wyoming     WY              97914 Mountain       West            5787
5 District of... DC                68 South Atlantic South            7057
6 Puerto Rico  PR              3515 <NA>           <NA>            31936
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,
#   cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,
#   cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,
#   Series_Complete_Pop_Pct <dbl>
```

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(state_data)
```

```
Rows: 52
```

```
Columns: 14
```

```
$ entity      <chr> "Alabama", "Alaska", "Arizona", "Arkansas",  
$ state_abb   <chr> "AL", "AK", "AZ", "AR", "CA", "CO", "CT",  
$ state_area_sq_miles <dbl> 51609, 589757, 113909, 53104, 158693, 10424  
$ state_division <chr> "East South Central", "Pacific", "Mountain"  
$ state_region <chr> "South", "West", "West", "South", "West",  
$ population  <dbl> 4903185, 731545, 7278717, 3017804, 39512223  
$ births_in_2021 <dbl> 58054, 9367, 77916, 35965, 420608, 62949, 3  
$ fertility_rate_per_1000 <dbl> 59.5, 64.9, 55.5, 61.7, 52.8, 52.5, 52.1, 5  
$ cesarean_percent <dbl> 35.1, 24.2, 28.7, 34.3, 30.8, 27.3, 35.4, 3  
$ life_expect <dbl> 73.2, 76.6, 76.3, 73.8, 79.0, 78.3, 78.4, 7  
$ cancer_rate_per_100000 <dbl> 160.2, 156.0, 134.7, 168.2, 132.4, 126.5, 1  
$ cancer_mortality <dbl> 10429, 1093, 12813, 6516, 59503, 8058, 6526  
$ Administered_Dose1_Pop_Pct <dbl> 64.8, 72.8, 77.1, 69.6, 84.3, 83.3, 95.0, 8  
$ Series_Complete_Pop_Pct <dbl> 53.0, 64.9, 65.8, 56.7, 74.4, 73.2, 82.8, 7
```

Dimensions

`nrow` gives the number of rows

```
nrow(state_data)
```

```
[1] 52
```

Renaming Columns of a data frame or tibble

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

Notice the **new name** is listed **first**!

```
# general format! not code!
```

```
{data you are creating or changing} <- {data you are using} %>%  
  rename({New Name} = {Old name})
```

```
state_data_rename <- state_data %>% rename(location = entity)  
head(state_data_rename)
```

```
# A tibble: 6 × 14
```

	location	state_abb	state_area_sq_mil...	state_division	state_region	population
	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	Alabama	AL	51609	East South Ce...	South	4903185
2	Alaska	AK	589757	Pacific	West	731545
3	Arizona	AZ	113909	Mountain	West	7278717
4	Arkansas	AR	53104	West South Ce...	South	3017804
5	California	CA	158693	Pacific	West	39512223
6	Colorado	CO	104247	Mountain	West	5758736

```
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,  
# cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,  
# cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,  
# Series_Complete_Pop_Pct <dbl>
```

Reassign the data!

If we don't reassign the data it will just print the change to the screen

```
# this does not change the data! Just prints the result  
state_data %>% rename(location = entity)
```

```
# this makes a new data frame that is changed  
state_data_rename <- state_data %>% rename(location = entity)
```

```
# this updates the existing data  
state_data <- state_data %>% rename(location = entity)
```

Unusual Column Names

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

You can use ` backticks ` to refer to them . You may see people use quotes in certain situations.



Atypical column names are those with:

- spaces
- number without characters
- number starting the name
- other punctuation marks (besides “_” or “.” and not at the beginning)

A solution!

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



Summary

- `head()`, `tail()`, and `glimpse()` help us look at the data
- the `rename()` function of `dplyr` can help you rename columns
- avoid using punctuation in column names
- if you must refer to a nonstandard column name - use backticks around it.

Let's practice!

Practice

How can you look at the last row of a data frame?

```
state_data %>% _____
```

Practice

How can you see the data rotated to see the column names more easily?

```
state_data %>% _____
```

Practice

How can you rename the column "state_region" to be "region"?

```
state_data %>% rename(_____ = _____)
```

Subsetting Columns

Subset columns of a data frame: dplyr

The `select` command from `dplyr` allows you to subset part of the data

```
state_data %>% select(cancer_mortality)
```

```
# A tibble: 52 × 1
  cancer_mortality
      <dbl>
1      10429
2       1093
3      12813
4       6516
5      59503
6       8058
7       6526
8       2178
9      46937
10     18136
# ... with 42 more rows
```

Select multiple columns

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

```
state_data %>% select(entity, cancer_mortality)
```

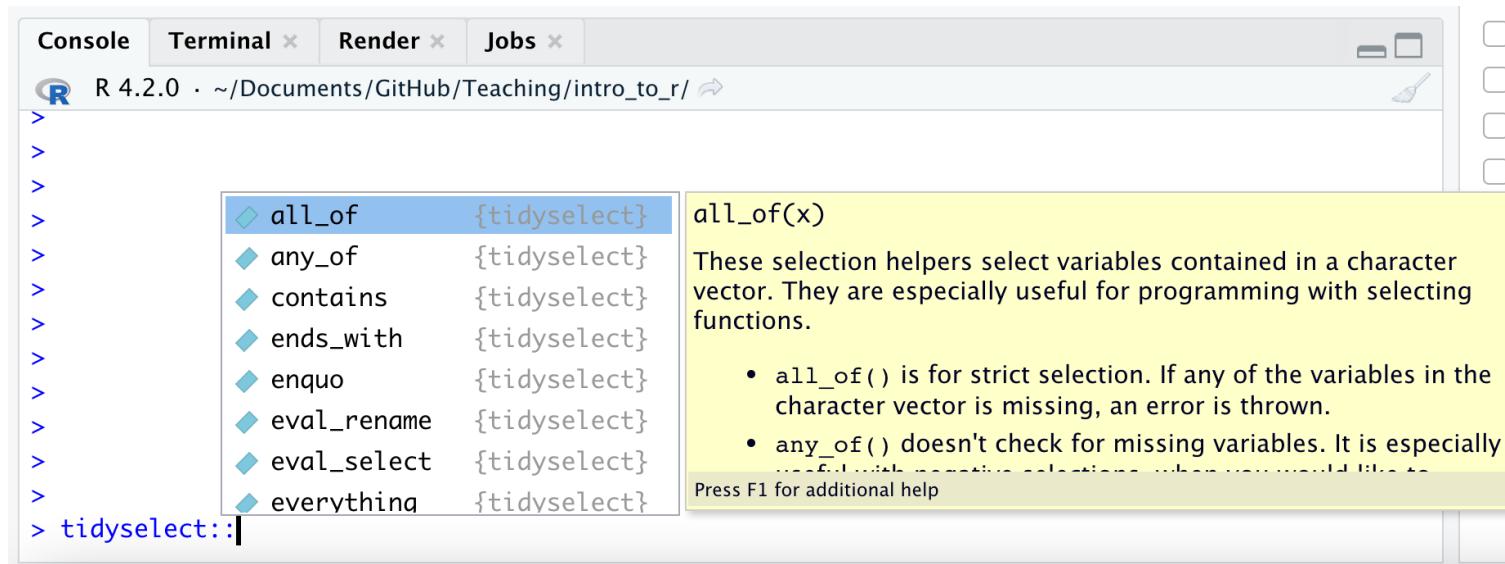
```
# A tibble: 52 × 2
  entity      cancer_mortality
  <chr>          <dbl>
1 Alabama      10429
2 Alaska        1093
3 Arizona     12813
4 Arkansas      6516
5 California   59503
6 Colorado      8058
7 Connecticut   6526
8 Delaware      2178
9 Florida     46937
10 Georgia     18136
# ... with 42 more rows
```


See the Select “helpers”

Here are a few:

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

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



Select columns of a data frame: dplyr

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

```
state_data %>% select(starts_with("cancer"))
```

```
# A tibble: 52 × 2
  cancer_rate_per_100000 cancer_mortality
      <dbl>          <dbl>
1      160.         10429
2      156          1093
3      135.        12813
4      168.         6516
5      132.        59503
6      126.         8058
7      134.         6526
8      153.         2178
9      142.        46937
10     152.        18136
# ... with 42 more rows
```

Combining tidyselect helpers with regular selection

```
state_data %>% select(entity, starts_with("cancer"))
```

```
# A tibble: 52 × 3
```

	entity <chr>	cancer_rate_per_100000 <dbl>	cancer_mortality <dbl>
1	Alabama	160.	10429
2	Alaska	156	1093
3	Arizona	135.	12813
4	Arkansas	168.	6516
5	California	132.	59503
6	Colorado	126.	8058
7	Connecticut	134.	6526
8	Delaware	153.	2178
9	Florida	142.	46937
10	Georgia	152.	18136

```
# ... with 42 more rows
```

Sometimes we want to preview the values

We can use `pull()` for that to “pull out” the values. This can give us a sense of the range, missing values, unusual values etc.

```
state_data %>% pull(cancer_rate_per_100000)
```

```
[1] 160.2 156.0 134.7 168.2 132.4 126.5 133.5 153.2 141.6 151.5 125.4 140.4  
[13] 150.0 169.7 150.9 150.8 181.1 163.9 161.3 139.2 137.4 160.1 143.2 181.8  
[25] 164.2 142.2 150.9 143.2 145.7 130.6 137.3 125.3 153.6 137.8 163.0 175.1  
[37] 155.2 152.9 142.0 155.2 154.8 166.3 143.3 121.0 154.0 150.5 149.3 184.7  
[49] 147.2 156.7      NA      NA
```

Practice

How can you subset the data to just have the “entity” and “fertility_rate_per_1000” columns?

```
state_data %>% _____(_____, _____)
```

Practice

How can I subset the data to just have the “entity” and “fertility_rate_per_1000” columns?

```
state_data %>% _____(_____, _____)
```

Subsetting Rows

Subset rows of a data frame: dplyr

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

```
# General format - Not the code!
```

```
{data object to update} <- {data to use} %>%  
  filter({variable name} {some condition})
```

```
state_data %>% filter(cancer_rate_per_100000 < 130)
```

```
# A tibble: 4 × 14
```

	entity	state_abb	state_area_sq_miles	state_division	state_region	population
	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	Colorado	CO	104247	Mountain	West	5758736
2	Hawaii	HI	6450	Pacific	West	1415872
3	New York	NY	49576	Middle Atlantic	Northeast	19453561
4	Utah	UT	84916	Mountain	West	3205958

```
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,  
#   cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,  
#   cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,  
#   Series_Complete_Pop_Pct <dbl>
```


Subset rows of a data frame: dplyr

You can have multiple **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

Subset rows of a data frame: dplyr

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

```
state_data %>% filter(cancer_rate_per_100000 < 130 & population > 4000000)
```

```
# A tibble: 2 × 14
  entity    state_abb state_area_sq_miles state_division state_region population
  <chr>      <chr>                <dbl> <chr>                <chr>         <dbl>
1 Colorado CO                104247 Mountain          West          5758736
2 New York NY                49576 Middle Atlantic Northeast    19453561
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,
#   cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,
#   cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,
#   Series_Complete_Pop_Pct <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:

```
state_data %>% filter(cancer_rate_per_100000 < 130 | population > 20000000)
```

```
# A tibble: 7 × 14
```

	entity	state_abb	state_area_sq_mil...	state_division	state_region	population
	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	California	CA	158693	Pacific	West	39512223
2	Colorado	CO	104247	Mountain	West	5758736
3	Florida	FL	58560	South Atlantic	South	21477737
4	Hawaii	HI	6450	Pacific	West	1415872
5	New York	NY	49576	Middle Atlant...	Northeast	19453561
6	Texas	TX	267339	West South Ce...	South	28995881
7	Utah	UT	84916	Mountain	West	3205958

```
# ... with 8 more variables: births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,  
# cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,  
# cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,  
# Series_Complete_Pop_Pct <dbl>
```

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. Use backticks if it isn't typical.

```
state_data %>% filter("cancer_rate_per_100000"< 130)
```

```
# A tibble: 0 × 14
# ... with 14 variables: entity <chr>, state_abb <chr>,
#   state_area_sq_miles <dbl>, state_division <chr>, state_region <chr>,
#   population <dbl>, births_in_2021 <dbl>, fertility_rate_per_1000 <dbl>,
#   cesarean_percent <dbl>, life_expect <dbl>, cancer_rate_per_100000 <dbl>,
#   cancer_mortality <dbl>, Administered_Dose1_Pop_Pct <dbl>,
#   Series_Complete_Pop_Pct <dbl>
```

Always good to check each step!

Did the filter work the way you expected? Did the number of rows change? Use nrow!



<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"))`
- `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()`

[Workshop Website](#)

Practice

How can I filter the data to only see the row about the state of California

```
state_data %>% filter(_____)
```


Adding/Modifying 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} <- {data to use} %>%  
  mutate({new variable name} = {new variable source})
```

```
state_updated <- state_data %>%  
  mutate(newcol = cancer_mortality / population)
```

Let's take a look

```
glimpse(state_updated)
```

```
Rows: 52
```

```
Columns: 15
```

\$ entity	<chr> "Alabama", "Alaska", "Arizona", "Arkansas", "
\$ state_abb	<chr> "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "
\$ state_area_sq_miles	<dbl> 51609, 589757, 113909, 53104, 158693, 10424
\$ state_division	<chr> "East South Central", "Pacific", "Mountain"
\$ state_region	<chr> "South", "West", "West", "South", "West", "
\$ population	<dbl> 4903185, 731545, 7278717, 3017804, 39512223
\$ births_in_2021	<dbl> 58054, 9367, 77916, 35965, 420608, 62949, 3
\$ fertility_rate_per_1000	<dbl> 59.5, 64.9, 55.5, 61.7, 52.8, 52.5, 52.1, 5
\$ cesarean_percent	<dbl> 35.1, 24.2, 28.7, 34.3, 30.8, 27.3, 35.4, 3
\$ life_expect	<dbl> 73.2, 76.6, 76.3, 73.8, 79.0, 78.3, 78.4, 7
\$ cancer_rate_per_100000	<dbl> 160.2, 156.0, 134.7, 168.2, 132.4, 126.5, 1
\$ cancer_mortality	<dbl> 10429, 1093, 12813, 6516, 59503, 8058, 6526
\$ Administered_Dose1_Pop_Pct	<dbl> 64.8, 72.8, 77.1, 69.6, 84.3, 83.3, 95.0, 8
\$ Series_Complete_Pop_Pct	<dbl> 53.0, 64.9, 65.8, 56.7, 74.4, 73.2, 82.8, 7
\$ newcol	<dbl> 0.002126985, 0.001494098, 0.001760338, 0.00

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} <- {data to use} %>%  
  mutate({variable name to change} = {variable modification})
```

```
state_updated<- state_updated %>%  
  mutate(newcol = newcol *1000000)
```

Let's take a look

```
glimpse(state_updated)
```

```
Rows: 52
```

```
Columns: 15
```

```
$ entity      <chr> "Alabama", "Alaska", "Arizona", "Arkansas",  
$ state_abb   <chr> "AL", "AK", "AZ", "AR", "CA", "CO", "CT",  
$ state_area_sq_miles <dbl> 51609, 589757, 113909, 53104, 158693, 10424  
$ state_division <chr> "East South Central", "Pacific", "Mountain"  
$ state_region <chr> "South", "West", "West", "South", "West",  
$ population  <dbl> 4903185, 731545, 7278717, 3017804, 39512223  
$ births_in_2021 <dbl> 58054, 9367, 77916, 35965, 420608, 62949,  
$ fertility_rate_per_1000 <dbl> 59.5, 64.9, 55.5, 61.7, 52.8, 52.5, 52.1,  
$ cesarean_percent <dbl> 35.1, 24.2, 28.7, 34.3, 30.8, 27.3, 35.4,  
$ life_expect <dbl> 73.2, 76.6, 76.3, 73.8, 79.0, 78.3, 78.4,  
$ cancer_rate_per_100000 <dbl> 160.2, 156.0, 134.7, 168.2, 132.4, 126.5,  
$ cancer_mortality <dbl> 10429, 1093, 12813, 6516, 59503, 8058, 6526  
$ Administered_Dose1_Pop_Pct <dbl> 64.8, 72.8, 77.1, 69.6, 84.3, 83.3, 95.0,  
$ Series_Complete_Pop_Pct <dbl> 53.0, 64.9, 65.8, 56.7, 74.4, 73.2, 82.8,  
$ newcol      <dbl> 2126.985, 1494.098, 1760.338, 2159.186, 150
```

Let's rename the new column

```
state_updated<- state_updated %>%  
  rename(cancer_mortality_rate_per_1000000 = newcol)
```

Reordering rows

The `arrange()` function can be a big help! It automatically does it in smallest to largest order.

```
state_updated %>% arrange(cancer_mortality_rate_per_1000000)
```

```
# A tibble: 52 × 15
```

	entity	state_abb	state_area_sq_m...	state_division	state_region	population
	<chr>	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	Utah	UT	84916	Mountain	West	3205958
2	Colorado	CO	104247	Mountain	West	5758736
3	Texas	TX	267339	West South Ce...	South	28995881
4	Alaska	AK	589757	Pacific	West	731545
5	California	CA	158693	Pacific	West	39512223
6	New York	NY	49576	Middle Atlant...	Northeast	19453561
7	North Dako...	ND	70665	West North Ce...	North Centr...	762062
8	Georgia	GA	58876	South Atlantic	South	10617423
9	Nevada	NV	110540	Mountain	West	3080156
10	New Jersey	NJ	7836	Middle Atlant...	Northeast	8882190

```
# ... with 42 more rows, and 9 more variables: births_in_2021 <dbl>,  
# fertility_rate_per_1000 <dbl>, cesarean_percent <dbl>, life_expect <dbl>,  
# cancer_rate_per_100000 <dbl>, cancer_mortality <dbl>,  
# Administered_Dose1_Pop_Pct <dbl>, Series_Complete_Pop_Pct <dbl>,  
# cancer_mortality_rate_per_1000000 <dbl>
```

A note about base R:

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

```
mtcars$carb
```

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

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

Practice

How can you create a new column that is the `fertility_rate_per_1000` multiplied by 2?

```
state_data %>% mutate(_____ = _____ *2)
```

How can you modify the column that you just made to divide it by the `population` value?

```
state_data %>% mutate(_____ = _____ / _____)
```

Summary

- can subset or remove rows with `filter()`
- can subset or remove (select what we want to keep) columns with `select()`
- `mutate()` can be used to create new variables or modify them

General format - Not the code!

```
{data object to update} <- {data to use} %>%  
  mutate({new variable name} = {new variable source})
```

[Workshop Website](#)