hw2

learning objectives:

- manipulate data tables
- make quality data visualizations

student: Adrian Henle

```
my_name = "Adrian Henle"
```

```
my_name = "Adrian Henle"
```

```
using PyPlot, DataFrames, CSV, Statistics
```

gapminder data

goal: make a scatter plot of life expectancy (y-axis) against income per person (x-axis), where each point represents a country. use the data from the year 2011. make the x-axis on a log scale. be sure to include units and axis labels.

data:

- `GDP_per_capita.csv` has a list of countries with GDP in different years
- `life_expectancy.csv` has a list of countries with life expectancy in different years

source: **Gapminder**

you must read in the files as they are and do all computations in Julia (not in Excel, for example).

> **Hint**
>
> [illegible]

(1) read in `GDP_per_capita.csv` as `df_gdp`. keep only the `:country` and `Symbol("2011")` columns, since those are the only ones we need.

```
df_gdp =
```

275 rows × 2 columns

	country String	2011 Float64?
1	Sint Maarten (Dutch part)	missing
2	Germany	26206.5
3	Spain	15428.3
4	Bosnia and Herzegovina	2225.29
5	North Yemen (former)	missing
6	Somalia	missing
7	Congo, Rep.	1266.36
8	Christmas Island	missing
9	Malta	11213.5
10	New Caledonia	missing
11	Wake Island	missing
12	Abkhazia	missing
13	Mexico	6288.25
14	North Korea	missing
15	Holy See	missing
16	Barbados	missing
17	Netherlands Antilles	missing
18	Gambia	614.912
⋮	⋮	⋮

```
df_gdp = CSV.read("data/GDP_per_capita.csv")[:,[:country, Symbol("2011")]]
```

(2) read in `life_expectancy.csv` as `df_life`. keep only the `:country` and `Symbol("2011")` columns, since those are the only ones we need.

`df_life` =
260 rows × 2 columns

	country String	2011 Float64?
1	Syria	75.1
2	Congo, Dem. Rep.	58.8
3	Chad	56.1
4	Pitcairn	missing
5	Turkmenistan	68.5
6	Mali	59.6
7	Grenada	71.0

	country String	2011 Float64?
8	Svalbard	missing
9	Cyprus	81.1
10	Mauritania	68.8
11	South Korea	80.6
12	Thailand	74.3
13	Saudi Arabia	78.9
14	Anguilla	missing
15	Angola	58.1
16	Iceland	82.9
17	Georgia	72.2
18	Christmas Island	missing
⋮	⋮	⋮

```
df_life = CSV.read("data/life_expectancy.csv")[:,[:country, Symbol("2011")]]
```

(3) `join` the two data frames by `:country` so you have a new data frame, `df_j`, that appears:

you'll need to rename the 2011 column in both so that they are unique, e.g. change the column names to `life_expectancy` and `gdp_per_cap`, where it is understood that they correspond to 2011. try to join the data frames without changing the column names first if you don't believe me!

260 rows × 2 columns

	country String	life_expectancy Float64?
1	Syria	75.1
2	Congo, Dem. Rep.	58.8
3	Chad	56.1
4	Pitcairn	missing
5	Turkmenistan	68.5
6	Mali	59.6
7	Grenada	71.0
8	Svalbard	missing
9	Cyprus	81.1
10	Mauritania	68.8
11	South Korea	80.6
12	Thailand	74.3

	country String	life_expectancy Float64?
13	Saudi Arabia	78.9
14	Anguilla	missing
15	Angola	58.1
16	Iceland	82.9
17	Georgia	72.2
18	Christmas Island	missing
⋮	⋮	⋮

```
names!(df_life, [:country, :life_expectancy])
```

275 rows × 2 columns

	country String	gdp_per_cap Float64?
1	Sint Maarten (Dutch part)	missing
2	Germany	26206.5
3	Spain	15428.3
4	Bosnia and Herzegovina	2225.29
5	North Yemen (former)	missing
6	Somalia	missing
7	Congo, Rep.	1266.36
8	Christmas Island	missing
9	Malta	11213.5
10	New Caledonia	missing
11	Wake Island	missing
12	Abkhazia	missing
13	Mexico	6288.25
14	North Korea	missing
15	Holy See	missing
16	Barbados	missing
17	Netherlands Antilles	missing
18	Gambia	614.912
⋮	⋮	⋮

```
names!(df_gdp, [:country, :gdp_per_cap])
```

```
df_j =
```

260 rows × 3 columns

	country String	life_expectancy Float64⍰	gdp_per_cap Float64⍰
1	Syria	75.1	missing
2	Congo, Dem. Rep.	58.8	109.809
3	Chad	56.1	301.402
4	Pitcairn	missing	missing
5	Turkmenistan	68.5	1370.43
6	Mali	59.6	272.309
7	Grenada	71.0	6047.2
8	Svalbard	missing	missing
9	Cyprus	81.1	15378.2
10	Mauritania	68.8	623.374
11	South Korea	80.6	16684.2
12	Thailand	74.3	2699.12
13	Saudi Arabia	78.9	9913.76
14	Anguilla	missing	missing
15	Angola	58.1	629.955
16	Iceland	82.9	34706.2
17	Georgia	72.2	1334.65
18	Christmas Island	missing	missing
⋮	⋮	⋮	⋮

```
df_j = join(df_life, df_gdp, on=:country)
```

(4) drop all rows that contain `missing` values.

171 rows × 3 columns

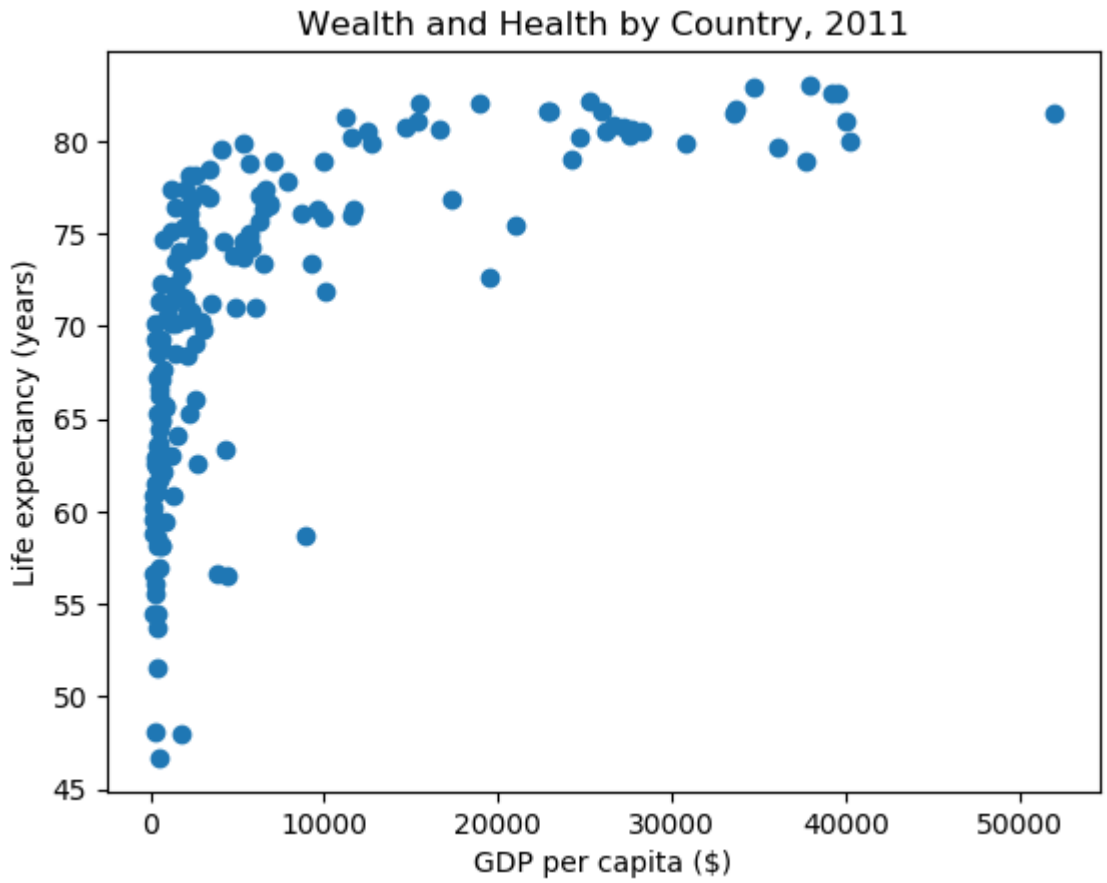
	country String	life_expectancy Float64	gdp_per_cap Float64
1	Congo, Dem. Rep.	58.8	109.809
2	Chad	56.1	301.402
3	Turkmenistan	68.5	1370.43
4	Mali	59.6	272.309
5	Grenada	71.0	6047.2
6	Cyprus	81.1	15378.2
7	Mauritania	68.8	623.374

	country String	life_expectancy Float64	gdp_per_cap Float64
8	South Korea	80.6	16684.2
9	Thailand	74.3	2699.12
10	Saudi Arabia	78.9	9913.76
11	Angola	58.1	629.955
12	Iceland	82.9	34706.2
13	Georgia	72.2	1334.65
14	Timor-Leste	71.3	451.983
15	Solomon Islands	63.0	1214.86
16	Senegal	64.4	559.968
17	Serbia	75.1	1223.64
18	Albania	77.4	1965.71
⋮	⋮	⋮	⋮

```
dropmissing!(df_j)
```

(5) make your scatter plot. using `df_j`, now that you have the life expectancy and GDP per capita matched up with each country properly.

> **Warning**
>
> think about why you needed to `join` the two data frames, don't just listen to my instructions! why not just:
>
> ```
> plot(df_life[:, Symbol("2011")], df_gdp[:, Symbol("2011")]) # wrong
> ```




```
begin
    figure()
    scatter(df_j.gdp_per_cap, df_j.life_expectancy)
    title("Wealth and Health by Country, 2011")
    xlabel("GDP per capita (\$)")
    ylabel("Life expectancy (years)")
    gcf()
end
```

💭 *ambitious Beavers*: a data viz challenge is to re-create **this Gapminder viz**. the size is proportional to the population of the country and the color indicates the continent. you can make an animation by looping over the years and doing what you did here.

note on causality: the independent variable is typically on the x-axis, whereas the dependent variable is typically on the y-axis. that I chose to put the income per person on the x-axis reflects my a priori belief that life expectancy is dependent in a causal sense on the income per person more so than the other way around. we can postulate mechanisms by which this could be true, e.g. since good health care costs money. however, we could also argue the reverse: if the life expectancy of folks in a population is low owing to disease, then they won't be able to be productive and earn a lot of money. so, this is a social science question, and it would be totally fine to switch the x- and y-axis in this problem. without further study, we can only say that we are looking at the *correlation* between these two variables. I like the idea of **these authors** in Fig. 5: putting one variable in the x-axis almost tells the audience what you perceive to be the independent variable.

Random

is GDP per capita = income per person? this is an economics question.

automobiles

(1) read in the CSV file `automobiles.csv` (**source of data**). the names of the columns are not in the `.csv` file but are described **here**. for your convenience, I made an array of the column names below. name your data frame `df_auto`.

```
automobile_col_names =  Symbol[:mpg, :cylinders, :displacement, :horsepower, :weight,
```

```
automobile_col_names = [:mpg, :cylinders, :displacement, :horsepower, :weight,
                        :acceleration, :model_year, :origin, :car_name]
```

`df_auto` =

398 rows × 9 columns (omitted printing of 1 columns)

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
	Float64	Int64	Float64	Float64?	Float64	Float64	Int64	In
1	18.0	8	307.0	130.0	3504.0	12.0	70	1
2	15.0	8	350.0	165.0	3693.0	11.5	70	1
3	18.0	8	318.0	150.0	3436.0	11.0	70	1
4	16.0	8	304.0	150.0	3433.0	12.0	70	1
5	17.0	8	302.0	140.0	3449.0	10.5	70	1
6	15.0	8	429.0	198.0	4341.0	10.0	70	1
7	14.0	8	454.0	220.0	4354.0	9.0	70	1

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
	Float64	Int64	Float64	Float64?	Float64	Float64	Int64	In
8	14.0	8	440.0	215.0	4312.0	8.5	70	1
9	14.0	8	455.0	225.0	4425.0	10.0	70	1
10	15.0	8	390.0	190.0	3850.0	8.5	70	1
11	15.0	8	383.0	170.0	3563.0	10.0	70	1
12	14.0	8	340.0	160.0	3609.0	8.0	70	1
13	15.0	8	400.0	150.0	3761.0	9.5	70	1
14	14.0	8	455.0	225.0	3086.0	10.0	70	1
15	24.0	4	113.0	95.0	2372.0	15.0	70	3
16	22.0	6	198.0	95.0	2833.0	15.5	70	1
17	18.0	6	199.0	97.0	2774.0	15.5	70	1
18	21.0	6	200.0	85.0	2587.0	16.0	70	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
df_auto = names!(CSV.read("data/automobiles.csv", copycols=true, header=0), automobile_col_names)
```

(2) how many automobiles are there in the data set?

398

```
size(df_auto, 1)
```

(3) are there any automobiles missing information about their horsepower? how many?

Yes, there are 6

```
sum(ismissing.(df_auto.horsepower)) == 0 ? md"""No, there are none.""" : md"""Yes, there are $(sum(ismissing.(df_auto.horsepower)))"""
```

(4) drop the rows from the `DataFrame` that contain a `missing` attribute

392 rows × 9 columns (omitted printing of 1 columns)

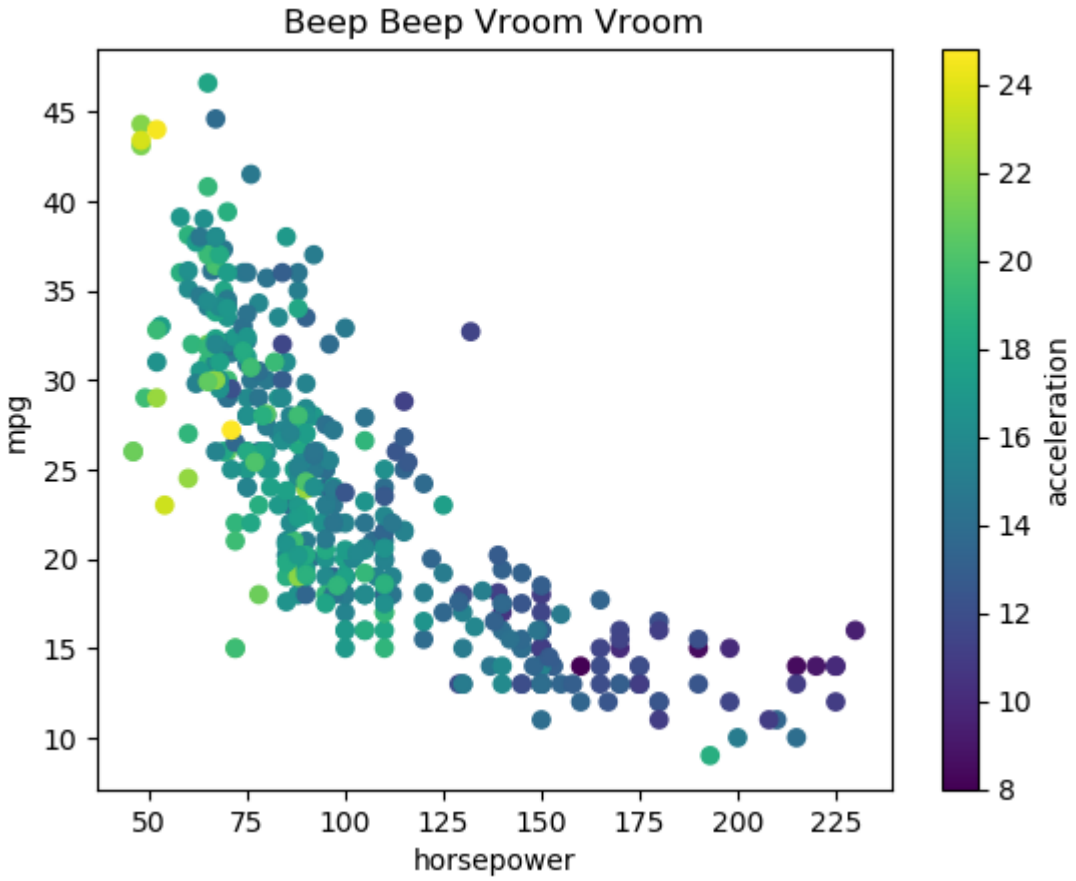
	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
	Float64	Int64	Float64	Float64	Float64	Float64	Int64	In
1	18.0	8	307.0	130.0	3504.0	12.0	70	1
2	15.0	8	350.0	165.0	3693.0	11.5	70	1
3	18.0	8	318.0	150.0	3436.0	11.0	70	1
4	16.0	8	304.0	150.0	3433.0	12.0	70	1
5	17.0	8	302.0	140.0	3449.0	10.5	70	1
6	15.0	8	429.0	198.0	4341.0	10.0	70	1

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
	Float64	Int64	Float64	Float64	Float64	Float64	Int64	In
7	14.0	8	454.0	220.0	4354.0	9.0	70	1
8	14.0	8	440.0	215.0	4312.0	8.5	70	1
9	14.0	8	455.0	225.0	4425.0	10.0	70	1
10	15.0	8	390.0	190.0	3850.0	8.5	70	1
11	15.0	8	383.0	170.0	3563.0	10.0	70	1
12	14.0	8	340.0	160.0	3609.0	8.0	70	1
13	15.0	8	400.0	150.0	3761.0	9.5	70	1
14	14.0	8	455.0	225.0	3086.0	10.0	70	1
15	24.0	4	113.0	95.0	2372.0	15.0	70	3
16	22.0	6	198.0	95.0	2833.0	15.5	70	1
17	18.0	6	199.0	97.0	2774.0	15.5	70	1
18	21.0	6	200.0	85.0	2587.0	16.0	70	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
dropmissing!(df_auto)
```

(5) make a scatter plot of the data such that:

- each point represents an automobile
- x-axis is horsepower
- y-axis is mpg (miles per gallon)
- color of point is used to illustrate acceleration (pass `c=x` into `scatter` with `x::Array{Float64, 1}` to color each point according to an array)
- a colorbar shows how the scale is depicted `colorbar(label="blah")`
- the colorbar, x-axis, and y-axis are labeled




```
begin
    figure()

    title("Beep Beep Vroom Vroom")

    scatter(df_auto.horsepower, df_auto.mpg, c=df_auto.acceleration)

    colorbar(label="acceleration")

    xlabel("horsepower")
    ylabel("mpg")

    gcf()
end
```

(6) remove from the `DataFrame` all automobiles that are not Fords. All Ford automobiles contain the string "ford" in the `:car_name` column. the function `occursin` will be useful here.

```
occursin("ford", "i love ford pintos") # true
occursin("ford", "i love subarus") # false
```

df_fords =

48 rows × 9 columns (omitted printing of 1 columns)

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
	Float64	Int64	Float64	Float64	Float64	Float64	Int64	In
1	17.0	8	302.0	140.0	3449.0	10.5	70	1
2	15.0	8	429.0	198.0	4341.0	10.0	70	1
3	21.0	6	200.0	85.0	2587.0	16.0	70	1
4	10.0	8	360.0	215.0	4615.0	14.0	70	1
5	19.0	6	250.0	88.0	3302.0	15.5	71	1
6	14.0	8	351.0	153.0	4154.0	13.5	71	1
7	13.0	8	400.0	170.0	4746.0	12.0	71	1
8	18.0	6	250.0	88.0	3139.0	14.5	71	1
9	21.0	4	122.0	86.0	2226.0	16.5	72	1
10	14.0	8	351.0	153.0	4129.0	13.0	72	1
11	13.0	8	302.0	140.0	4294.0	16.0	72	1
12	22.0	4	122.0	86.0	2395.0	16.0	72	1
13	14.0	8	302.0	137.0	4042.0	14.5	73	1
14	13.0	8	351.0	158.0	4363.0	13.0	73	1
15	18.0	6	250.0	88.0	3021.0	16.5	73	1
16	12.0	8	400.0	167.0	4906.0	12.5	73	1
17	19.0	4	122.0	85.0	2310.0	18.5	73	1
18	26.0	4	122.0	80.0	2451.0	16.5	74	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

```
df_fords = filter(row->occursin("ford",row.car_name), df_auto)
```

(7) of the Ford automobiles, how many unique numbers of cylinders are there?

3

```
length(unique(df_fords.cylinders))
```

(8) make a scatter plot for all Ford automobiles such that:

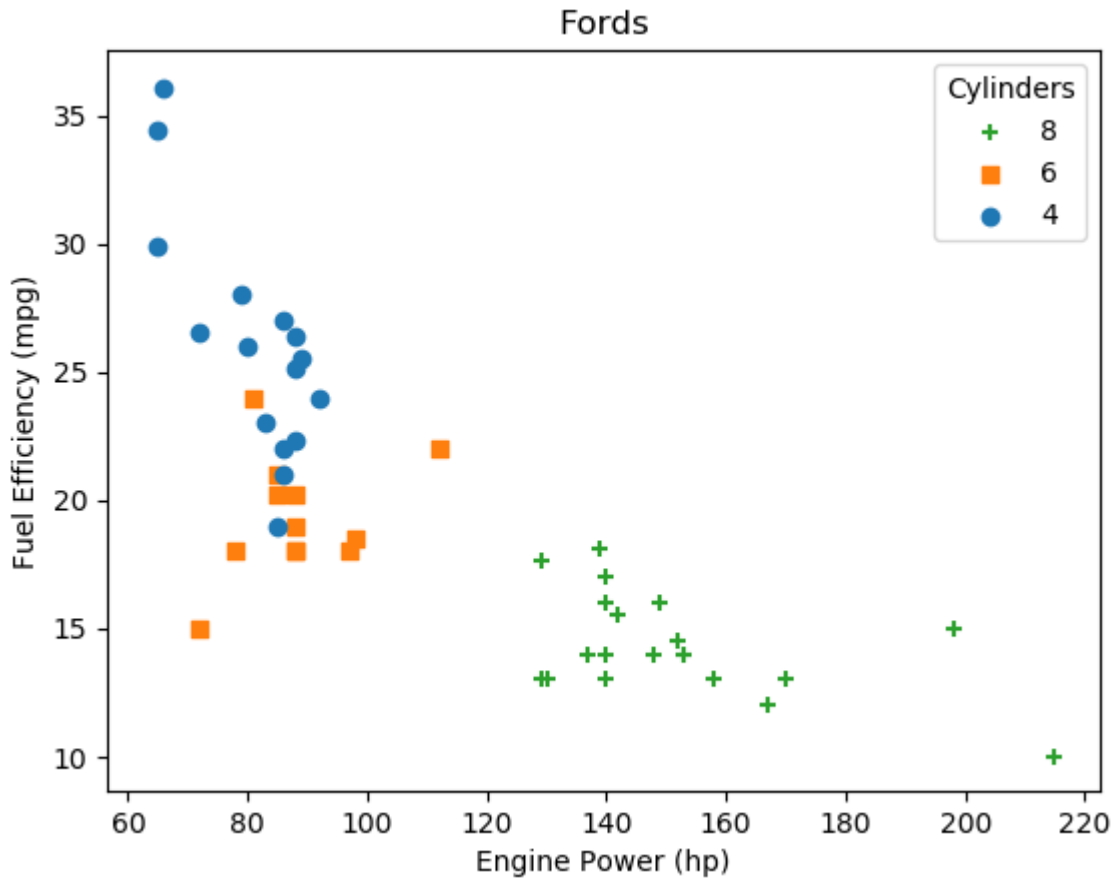
- each point represents an automobile
- x-axis is horsepower
- y-axis is mpg (miles per gallon)
- x-axis, and y-axis are labeled.
- title is "Fords"
- automobiles with different numbers of cylinders are depicted by different markers and different colors, and automobiles with the same number of cylinders share the same marker and color. for example:

    - 4 cylinders, use `marker="o"` and `color="C0"`
    - 6 cylinders, use `marker="s"` and `color="C1"`
    - 8 cylinders, use `marker="+"` and `color="C2"`

  I suggest a `Dict` to map the number of cylinders to a marker.
- a legend depicts the symbol and color for each number of cylinders

Hint

[illegible]




```
begin
    marker = Dict([4=>"o", 6=>"s", 8=>"+"])
    color = Dict([4=>"C0", 6=>"C1", 8=>"C2"])
    figure()
    for grp in groupby(df_fords, :cylinders)
        n = grp.cylinders[1]
        scatter(grp.horsepower, grp.mpg, color=color[n], marker=marker[n], label=n)
    end
    title("Fords")
    xlabel("Engine Power (hp)")
    ylabel("Fuel Efficiency (mpg)")
    legend(title="Cylinders")
    gcf()
end
```

ramen ratings

see `ramen-ratings.csv`.

> The Ramen Rater is a product review website for the hardcore ramen enthusiast, with over 2500 reviews to date. Each record in the dataset is a single ramen product review. Review numbers are contiguous: more recently reviewed ramen varieties have higher numbers. Brand, Variety (the product name), Country, and Style (Cup? Bowl? Tray?) are pretty self-explanatory. Stars indicate the ramen quality, as assessed by the reviewer, on a 5-point scale; this is the most important column in the dataset!

The `:top_ten` ramen attribute is `missing` if the ramen never placed in the top ten; if it is not `missing`, the `:top_ten` attribute gives us the year it placed in the top ten and its place.

source: **Kaggle**

(1) read in `ramen-ratings.csv` as a `DataFrame`, `df_ramen`, to work with.

`df_ramen =`
2,580 rows × 7 columns (omitted printing of 4 columns)

	review_id	brand	variety
	Int64	String	String
1	2580	New Touch	T's Restaurant Tantanmen
2	2579	Just Way	Noodles Spicy Hot Sesame Spicy Hot Sesame Guan-miao Noodles
3	2578	Nissin	Cup Noodles Chicken Vegetable
4	2577	Wei Lih	GGE Ramen Snack Tomato Flavor
5	2576	Ching's Secret	Singapore Curry
6	2575	Samyang Foods	Kimchi song Song Ramen
7	2574	Acecook	Spice Deli Tantan Men With Cilantro
8	2573	Ikeda Shoku	Nabeyaki Kitsune Udon
9	2572	Ripe'n'Dry	Hokkaido Soy Sauce Ramen
10	2571	KOKA	The Original Spicy Stir-Fried Noodles
11	2570	Tao Kae Noi	Creamy tom Yum Kung Flavour
12	2569	Yamachan	Yokohama Tonkotsu Shoyu
13	2568	Nongshim	Mr. Bibim Stir-Fried Kimchi Flavor
14	2567	Nissin	Deka Buto Kimchi Pork Flavor
15	2566	Nissin	Demae Ramen Bar Noodle Aka Tonkotsu Flavour Instant Noodle

	review_id	brand	variety
	Int64	String	String
16	2565	KOKA	Mushroom Flavour Instant Noodles
17	2564	TRDP	Mario Masala Noodles
18	2563	Yamachan	Tokyo Shoyu Ramen
⋮	⋮	⋮	⋮

```
df_ramen = CSV.read("data/ramen-ratings.csv", copycols=true)
```

(1.5) what are the names of the columns?

```
Symbol[:review_id, :brand, :variety, :style, :country, :stars, :top_ten]
```

```
names(df_ramen)
```

(2) how many ratings are in the data set?

```
2580
```

```
length(df_ramen.stars)
```

(3) oddly, the `:stars` column appears as an array of `String`s, even though they should be numbers!

why? because `"Unrated"` appears as the `:stars` attribute in a few rows and prevents `CSV.jl` from reading in the `:stars` column as all `Float64`s.

remove all rows of the `DataFrame` that have `"Unrated"` as the `:stars` attribute.

Hint

[redacted]

2,577 rows × 7 columns (omitted printing of 4 columns)

	review_id	brand	variety
	Int64	String	String
1	2580	New Touch	T's Restaurant Tantanmen
2	2579	Just Way	Noodles Spicy Hot Sesame Spicy Hot Sesame Guan-miao Noodles
3	2578	Nissin	Cup Noodles Chicken Vegetable
4	2577	Wei Lih	GGE Ramen Snack Tomato Flavor
5	2576	Ching's Secret	Singapore Curry
6	2575	Samyang Foods	Kimchi song Song Ramen

	review_id	brand	variety
	Int64	String	String
7	2574	Acecook	Spice Deli Tantan Men With Cilantro
8	2573	Ikeda Shoku	Nabeyaki Kitsune Udon
9	2572	Ripe'n'Dry	Hokkaido Soy Sauce Ramen
10	2571	KOKA	The Original Spicy Stir-Fried Noodles
11	2570	Tao Kae Noi	Creamy tom Yum Kung Flavour
12	2569	Yamachan	Yokohama Tonkotsu Shoyu
13	2568	Nongshim	Mr. Bibim Stir-Fried Kimchi Flavor
14	2567	Nissin	Deka Buto Kimchi Pork Flavor
15	2566	Nissin	Demae Ramen Bar Noodle Aka Tonkotsu Flavour Instant Noodle
16	2565	KOKA	Mushroom Flavour Instant Noodles
17	2564	TRDP	Mario Masala Noodles
18	2563	Yamachan	Tokyo Shoyu Ramen
⋮	⋮	⋮	⋮

```
filter!(row-> row.stars ≠ "Unrated", df_ramen)
```

(4) the `:stars` column still appears as `String`s. Convert it to `Float64`s by using `parse`, which works as follows:

```
3.76
```

```
parse(Float64, "3.76") # String -> Float64
```

use `parse`, operated element-wise on the `:stars` column, to convert the ratings to `Float64`s

Hint

[redacted]

```
Float64[3.75, 1.0, 2.25, 2.75, 3.75, 4.75, 4.0, 3.75, 0.25, 2.5, 5.0, 5.0, 4.25
```

```
df_ramen[!, :stars] = [parse(Float64, s) for s in df_ramen.stars]
```

```
Float64[3.75, 1.0, 2.25, 2.75, 3.75, 4.75, 4.0, 3.75, 0.25, 2.5, 5.0, 5.0, 4.25
```

```
df_ramen[:, :stars]
```

(5) What is the highest rated variety of ramen that satisfies all of the following attributes:

- "Nissin" brand
- mentions "Beef" in the variety

- "Pack" style

> **Hint**
>
> [illegible]

```
"Premium Instant Noodles Roasted Beef Flavour"
```

```
sort(filter(row -> row.brand == "Nissin" && occursin("Beef", row.variety) && row.style == "Pack", df_ramen), cols=:stars, rev=true).variety[1]
```

(6) ramen from how many different countries is rated?

```
37
```

```
length(unique(df_ramen.country))
```

(7) construct a new `DataFrame`, `df_by_country`, from the data, whose rows are the countries and whose two columns, `:avg_rating` and `nb_ratings`, give the average and number of ratings of ramen, respectively, from that country.

```
    country avg_rating  nb_ratings
    String  Float64 Int64
1   Japan   3.98161 352
2   Taiwan  3.6654  224
...
```

> **Hint**
>
> [illegible]

`df_by_country` =
37 rows × 3 columns

	country	avg_rating	nb_ratings
	String	Float64	Int64
1	Japan	3.98161	352
2	Taiwan	3.6654	224

	country String	avg_rating Float64	nb_ratings Int64
3	USA	3.45795	324
4	India	3.39516	31
5	South Korea	3.79055	307
6	Singapore	4.12615	109
7	Thailand	3.38482	191
8	Hong Kong	3.80182	137
9	Vietnam	3.18796	108
10	Ghana	3.5	2
11	Malaysia	4.15419	155
12	Indonesia	4.06746	126
13	China	3.42189	169
14	Nigeria	1.5	1
15	Germany	3.63889	27
16	Hungary	3.61111	9
17	Mexico	3.73	25
18	Fiji	3.875	4
⋮	⋮	⋮	⋮

```
df_by_country = by(df_ramen, :country, avg_rating=:stars=>mean,
nb_ratings=:stars=>length)
```

(8) some countries do not have many ramens with ratings. let's drop countries from the data frame that have fewer than 100 ratings.

11 rows × 3 columns

	country String	avg_rating Float64	nb_ratings Int64
1	Japan	3.98161	352
2	Taiwan	3.6654	224
3	USA	3.45795	324
4	South Korea	3.79055	307
5	Singapore	4.12615	109
6	Thailand	3.38482	191
7	Hong Kong	3.80182	137
8	Vietnam	3.18796	108
9	Malaysia	4.15419	155

	country String	avg_rating Float64	nb_ratings Int64
10	Indonesia	4.06746	126
11	China	3.42189	169

```
filter!(row -> row.nb_ratings ≥ 100, df_by_country)
```

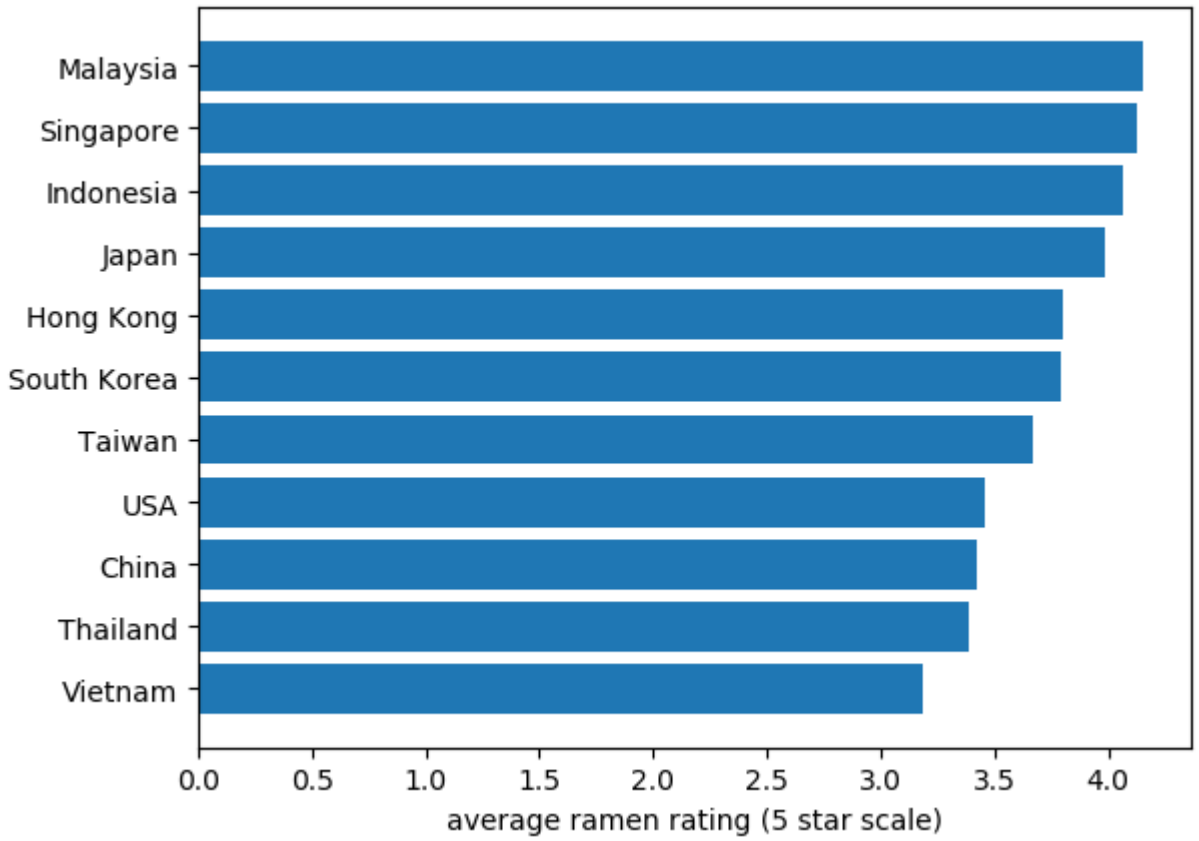
(9) `sort! df_by_country` by the average rating.

11 rows × 3 columns

	country String	avg_rating Float64	nb_ratings Int64
1	Vietnam	3.18796	108
2	Thailand	3.38482	191
3	China	3.42189	169
4	USA	3.45795	324
5	Taiwan	3.6654	224
6	South Korea	3.79055	307
7	Hong Kong	3.80182	137
8	Japan	3.98161	352
9	Indonesia	4.06746	126
10	Singapore	4.12615	109
11	Malaysia	4.15419	155

```
sort!(df_by_country, :avg_rating)
```

(10) make a bar plot. each country gets a bar to represent it. the length of the bar is proportional to the average rating of ramen in that country. use `df_by_country` to make the bar plot, which should only have 11 countries in it.




```
begin
    figure()
    xlabel("average ramen rating (5 star scale)")
    barh(df_by_country.country, df_by_country.avg_rating)
    gcf()
end
```