Lecture 1 - Python Review

CMSE 381 - Spring 2024

In today's assignment, we are going to go over some of the python commands you've (hopefully!) seen before.
In order to successfully complete this assignment you need to participate both individually and in groups during class. This lab loosely follows the lab content from the the ISLP textbook, Ch 2.3.

IMPORTANT: If you are looking at this notebook prior to class with the intention of doing it early PLEASE DON'T! The intention of these notebooks is for group work in class, so it will be much more interesting if you don't have it done already and can talk with the rest of the group members.

1. The Dataset

In this module, we will be using the `Auto.csv` data set from the textbook website. I have included the file in the data set folder on the repo but you can also download the file directly.

Info about the data set

- Info on R version
- Info on python version

Auto: Auto Data Set

Description

Gas mileage, horsepower, and other information for 392 vehicles. Usage

Format

A data frame with 392 observations on the following 9 variables.

- `mpg` : miles per gallon
- `cylinders` : Number of cylinders between 4 and 8
- `displacement` : Engine displacement (cu. inches)
- `horsepower` : Engine horsepower
- `weight` : Vehicle weight (lbs.)
- `acceleration` : Time to accelerate from 0 to 60 mph (sec.)
- `year` : Model year (modulo 100)
- `origin` : Origin of car (1. American, 2. European, 3. Japanese)
- `name` : Vehicle name

The orginal data contained 408 observations but 16 observations with missing values were removed.

Source

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University. The dataset was used in the 1983 American Statistical Association Exposition.

---

2. Load the data set

In [1]:
```
# As always, we start with our favorite standard imports.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

First, load in your csv file as a pandas data frame. Save this data frame as `auto`.

In [2]:
```
# If you are pulling straight from the course's git repo,
# this command will open the file already on your system.
# However, if you are doing something else to access the
# notebook, such as downloading from the github website,
# you might need to modify this command to point to the
# right place.
auto = pd.read_csv('/Users/siony/OneDrive/바탕 화면/MSU_SS_24/CMSE 381/CMSE381SS24/D
```

If that worked and you managed to load the file, the following command should show you the top of your data frame.

In [3]:
```
auto.head()
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

...and the following command show show you the column labels

In [4]:
```
auto.columns
```

Out[4]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'year', 'origin', 'name'],
      dtype='object')
```

The `shape` command tells us about the size of the dataframe.

In [5]:
```
auto.shape
```

Out[5]:
```
(397, 9)
```

✅ **Q:** How many data points do we have? How many variables do we have?

we have 9 columns and 397 variables

3. Cleaning up the data set

Here's one thing this class won't really show you..... real data is MESSY. You almost never are handed a data set that's ready to go for analysis off the bat. You'll have to spend a bit of time cleaning up your data before you can use the awesome tools we have in this class. So, to that end, let's do some careful checking of this data set before we get started. My favorite place to start with any data set is the `describe` command.

In [6]:
```
auto.describe()
```

Out[6]:

	mpg	cylinders	displacement	weight	acceleration	year	origin
count	397.000000	397.000000	397.000000	397.000000	397.000000	397.000000	397.000000
mean	23.515869	5.458438	193.532746	2970.261965	15.555668	75.994962	1.574307
std	7.825804	1.701577	104.379583	847.904119	2.749995	3.690005	0.802549
min	9.000000	3.000000	68.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.000000	2223.000000	13.800000	73.000000	1.000000
50%	23.000000	4.000000	146.000000	2800.000000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	3609.000000	17.100000	79.000000	2.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	82.000000	3.000000

✅ **Q:** What columns are missing from the `describe` output?

horsepower and the name is missing in the describe output.

The next thing to check is to see if there is any missing data. Usually, this is an entry in your dataframe that shows up as `np.nan`, which is a special value from numpy that just means the data is missing from that cell.

We can use the `isna()` command to see if there are any null values around.

In [7]:
```
auto[auto.isna().any(axis=1)]
```

Out[7]:

mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name

Hey cool, no NaN's to be found! You know what, maybe it's a good idea just to take a second look. Check out the first 40 rows of the data set:

In [8]:
```
auto.head(40)
```

Out[8]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190	3850	8.5	70	1	amc ambassador dpl
10	15.0	8	383.0	170	3563	10.0	70	1	dodge challenger se
11	14.0	8	340.0	160	3609	8.0	70	1	plymouth 'cuda 340
12	15.0	8	400.0	150	3761	9.5	70	1	chevrolet monte carlo
13	14.0	8	455.0	225	3086	10.0	70	1	buick estate wagon (sw)
14	24.0	4	113.0	95	2372	15.0	70	3	toyota corona mark ii
15	22.0	6	198.0	95	2833	15.5	70	1	plymouth duster
16	18.0	6	199.0	97	2774	15.5	70	1	amc hornet
17	21.0	6	200.0	85	2587	16.0	70	1	ford maverick
18	27.0	4	97.0	88	2130	14.5	70	3	datsun pl510
19	26.0	4	97.0	46	1835	20.5	70	2	volkswagen 1131 deluxe sedan
20	25.0	4	110.0	87	2672	17.5	70	2	peugeot 504
21	24.0	4	107.0	90	2430	14.5	70	2	audi 100 ls

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
22	25.0	4	104.0	95	2375	17.5	70	2	saab 99e
23	26.0	4	121.0	113	2234	12.5	70	2	bmw 2002
24	21.0	6	199.0	90	2648	15.0	70	1	amc gremlin
25	10.0	8	360.0	215	4615	14.0	70	1	ford f250
26	10.0	8	307.0	200	4376	15.0	70	1	chevy c20
27	11.0	8	318.0	210	4382	13.5	70	1	dodge d200
28	9.0	8	304.0	193	4732	18.5	70	1	hi 1200d
29	27.0	4	97.0	88	2130	14.5	71	3	datsun pl510
30	28.0	4	140.0	90	2264	15.5	71	1	chevrolet vega 2300
31	25.0	4	113.0	95	2228	14.0	71	3	toyota corona
32	25.0	4	98.0	?	2046	19.0	71	1	ford pinto
33	19.0	6	232.0	100	2634	13.0	71	1	amc gremlin
34	16.0	6	225.0	105	3439	15.5	71	1	plymouth satellite custom
35	17.0	6	250.0	100	3329	15.5	71	1	chevrolet chevelle malibu
36	19.0	6	250.0	88	3302	15.5	71	1	ford torino 500
37	18.0	6	232.0	100	3288	15.5	71	1	amc matador
38	14.0	8	350.0	165	4209	12.0	71	1	chevrolet impala
39	14.0	8	400.0	175	4464	11.5	71	1	pontiac catalina

✅ **Q:** What symbol(s) is this data set using to represent missing data?

horsepower have a missing data

✅ **DO THIS:** Well, it's not the end of the world, but there are lots of built in commands in pandas that make life easier when we have `np.nan` used as the missing value entry. So, use the `replace` command to swap out the missing value they used for `np.nan` everywhere it shows up.

In [9]:
```
#---- your code goes in here! ---#
auto[auto.horsepower == "?"] = np.nan

auto.head(40)
```

Out[9]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8.0	307.0	130	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150	3436.0	11.0	70.0	1.0	plymouth satellite
3	16.0	8.0	304.0	150	3433.0	12.0	70.0	1.0	amc rebel sst
4	17.0	8.0	302.0	140	3449.0	10.5	70.0	1.0	ford torino
5	15.0	8.0	429.0	198	4341.0	10.0	70.0	1.0	ford galaxie 500
6	14.0	8.0	454.0	220	4354.0	9.0	70.0	1.0	chevrolet impala
7	14.0	8.0	440.0	215	4312.0	8.5	70.0	1.0	plymouth fury iii
8	14.0	8.0	455.0	225	4425.0	10.0	70.0	1.0	pontiac catalina
9	15.0	8.0	390.0	190	3850.0	8.5	70.0	1.0	amc ambassador dpl
10	15.0	8.0	383.0	170	3563.0	10.0	70.0	1.0	dodge challenger se
11	14.0	8.0	340.0	160	3609.0	8.0	70.0	1.0	plymouth 'cuda 340
12	15.0	8.0	400.0	150	3761.0	9.5	70.0	1.0	chevrolet monte carlo
13	14.0	8.0	455.0	225	3086.0	10.0	70.0	1.0	buick estate wagon (sw)
14	24.0	4.0	113.0	95	2372.0	15.0	70.0	3.0	toyota corona mark ii
15	22.0	6.0	198.0	95	2833.0	15.5	70.0	1.0	plymouth duster
16	18.0	6.0	199.0	97	2774.0	15.5	70.0	1.0	amc hornet
17	21.0	6.0	200.0	85	2587.0	16.0	70.0	1.0	ford maverick
18	27.0	4.0	97.0	88	2130.0	14.5	70.0	3.0	datsun pl510
19	26.0	4.0	97.0	46	1835.0	20.5	70.0	2.0	volkswagen 1131 deluxe sedan
20	25.0	4.0	110.0	87	2672.0	17.5	70.0	2.0	peugeot 504
21	24.0	4.0	107.0	90	2430.0	14.5	70.0	2.0	audi 100 ls

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
22	25.0	4.0	104.0	95	2375.0	17.5	70.0	2.0	saab 99e
23	26.0	4.0	121.0	113	2234.0	12.5	70.0	2.0	bmw 2002
24	21.0	6.0	199.0	90	2648.0	15.0	70.0	1.0	amc gremlin
25	10.0	8.0	360.0	215	4615.0	14.0	70.0	1.0	ford f250
26	10.0	8.0	307.0	200	4376.0	15.0	70.0	1.0	chevy c20
27	11.0	8.0	318.0	210	4382.0	13.5	70.0	1.0	dodge d200
28	9.0	8.0	304.0	193	4732.0	18.5	70.0	1.0	hi 1200d
29	27.0	4.0	97.0	88	2130.0	14.5	71.0	3.0	datsun pl510
30	28.0	4.0	140.0	90	2264.0	15.5	71.0	1.0	chevrolet vega 2300
31	25.0	4.0	113.0	95	2228.0	14.0	71.0	3.0	toyota corona
32	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
33	19.0	6.0	232.0	100	2634.0	13.0	71.0	1.0	amc gremlin
34	16.0	6.0	225.0	105	3439.0	15.5	71.0	1.0	plymouth satellite custom
35	17.0	6.0	250.0	100	3329.0	15.5	71.0	1.0	chevrolet chevelle malibu
36	19.0	6.0	250.0	88	3302.0	15.5	71.0	1.0	ford torino 500
37	18.0	6.0	232.0	100	3288.0	15.5	71.0	1.0	amc matador
38	14.0	8.0	350.0	165	4209.0	12.0	71.0	1.0	chevrolet impala
39	14.0	8.0	400.0	175	4464.0	11.5	71.0	1.0	pontiac catalina

If you did that right, the following command should show you the 5 rows that now have a `np.nan` entry somewhere.

In [10]:
```
# Find the rows with a NaN somewhere
auto[auto.isna().any(axis=1)]
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
32	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
126	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
330	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
336	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
354	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

✅ **DO THIS:** Finally, let's just get rid of those rows from the data set entirely. Overwrite the `auto` data frame with the version that deletes those 5 rows using the `dropna` command.

In [11]:
```
#---- your code goes in here! ---#
auto = auto.dropna()
```

In [12]:
```
# If that worked, you now have 392 rows
auto.shape
```

Out[12]:
```
(392, 9)
```

Fixing horsepower

One last weird data cleanup for us to do on this data set. Check out the `horsepower` column.

In [13]:
```
auto['horsepower']
```

Out[13]:
```
0      130
1      165
2      150
3      150
4      140
      ...
392     86
393     52
394     84
395     79
396     82
Name: horsepower, Length: 392, dtype: object
```

Compare that to, for example, the `weight` and `name` columns.

In [14]:
```
auto['weight']
```

Out[14]:
```
0      3504.0
1      3693.0
2      3436.0
3      3433.0
4      3449.0
        ...
392    2790.0
393    2130.0
394    2295.0
395    2625.0
396    2720.0
Name: weight, Length: 392, dtype: float64
```

In [15]:
```
auto['name']
```

Out[15]:
```
0      chevrolet chevelle malibu
1              buick skylark 320
2             plymouth satellite
3                  amc rebel sst
4                    ford torino
                 ...
392              ford mustang gl
393                    vw pickup
394                dodge rampage
395                  ford ranger
396                   chevy s-10
Name: name, Length: 392, dtype: object
```

The `dtype` tells us what kind of data pandas thinks is contained in there. `int64` is for numbers, like horsepower, but for some reason* pandas is treating it like object data, which is what pandas uses for basically anything else, like data inputs that are strings. This makes sense for the `name` column, but we'd like to fix it for the `horsepower` column now that we've fixed the `np.nan` issue. The code below returns the `horsepower` column with `dtype: int64`.

*The reason is related to the weird choice of null entry for this data set. It's also why `describe` above didn't have the `horsepower` column.

In [22]:
```
auto['horsepower'].astype('int')
```

Out[22]:
```
0      130
1      165
2      150
3      150
4      140
      ...
392     86
393     52
394     84
395     79
396     82
Name: horsepower, Length: 392, dtype: int32
```

✅ **DO THIS:** Overwrite the `horsepower` column in the `auto` data frame with this fixed version.

In [ ]:
```
#---- Your code here! ----#
```

2. Extracting data from a frame

Ok, I know everyone needs a reminder on this (It's me. I can never remember any of this without googling it.....), let's just do a quick refresh on how to get out portions of your data table.

First, you can get a whole column (which is known as a pandas `Series`) like this:

In [16]:
```
auto['weight']
```

Out[16]:
```
0      3504.0
1      3693.0
2      3436.0
3      3433.0
4      3449.0
        ...
392    2790.0
393    2130.0
394    2295.0
395    2625.0
396    2720.0
Name: weight, Length: 392, dtype: float64
```

For more fine-grained control, there are two commands that are used: `loc`, and `iloc`.

In [17]:
```
auto.head()
```

Out[17]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8.0	307.0	130	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150	3436.0	11.0	70.0	1.0	plymouth satellite
3	16.0	8.0	304.0	150	3433.0	12.0	70.0	1.0	amc rebel sst
4	17.0	8.0	302.0	140	3449.0	10.5	70.0	1.0	ford torino

In [18]:
```
# `loc` takes the labels as inputs to find a particular point
auto.loc[3,'weight']
```

Out[18]:
```
3433.0
```

In [19]:
```
# 'iloc' takex the indices. Here's how to get the same number
auto.iloc[3,4]
```

Out[19]:
```
3433.0
```

In this case, the row entry is the same for both (`3`) because the rows happen to be labeled with their number. However, for `.loc`, we need the name of the column we want (`weight`), while for `.iloc` we need the number of the column (`4` because we count from zero...).

✅ **DO THIS:** Extract a data frame with rows 3,4,5, and 6, and with information on displacement, horsepower, and weight.

In [ ]:
```
#---- Your code here!----#
```

3. Plotting

The third-ish thing I do with a new data set that I'm trying to understand is to just start plotting random things. This is great for getting a sense of ranges for values, as well as to start looking for simple correlations.

In this class, we will use two python modules for plotting, depending on which has the tools we want:

- `matplotlib`. This is basically the standard plotting tool. It does basically anything you want (albeit with a bit of pain and suffering and a few choice four letter words along the way). You've already seen this package in CMSE 201 at least.
- `seaborn`. This is helpful for some prepackaged figure generation that we will make use of. It's actually built on top of matplotlib, but has often simplified syntax.

In [20]:
```
# Make sure you run this to import seaborn. If this doesn't run
# for some reason, check that it's installed.
import seaborn as sns
```

In [21]:
```
auto.head()
```

Out[21]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8.0	307.0	130	3504.0	12.0	70.0	1.0	chevrolet chevelle malibu
1	15.0	8.0	350.0	165	3693.0	11.5	70.0	1.0	buick skylark 320
2	18.0	8.0	318.0	150	3436.0	11.0	70.0	1.0	plymouth satellite
3	16.0	8.0	304.0	150	3433.0	12.0	70.0	1.0	amc rebel sst
4	17.0	8.0	302.0	140	3449.0	10.5	70.0	1.0	ford torino

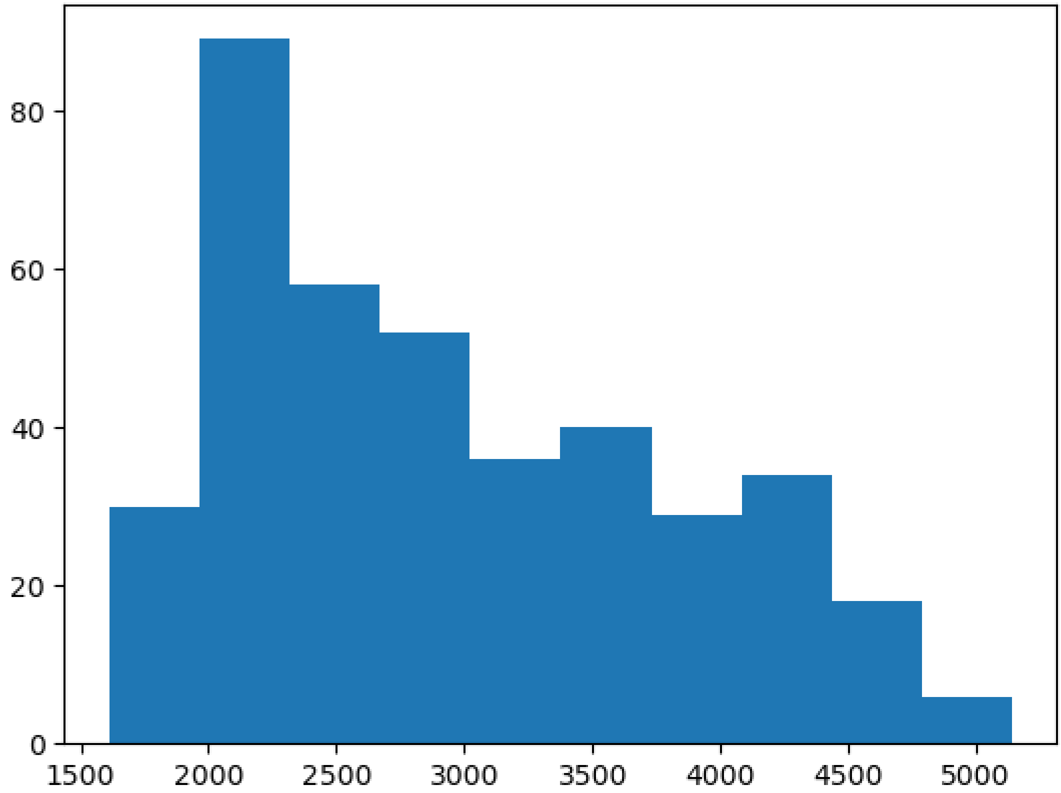
✅ **DO THIS:** First, use matplotlib's `hist` command to show a histogram of the `weight` data.

Hint if you ever forget how to use a command, you can of course google it, but you can also type `?` before the name of the command to see the help info from inside the jupyter notebook, e.g.:

```
?plt.hist
```

In [24]:
```
#----- Your code here!-----#
plt.hist(auto['weight'])
```

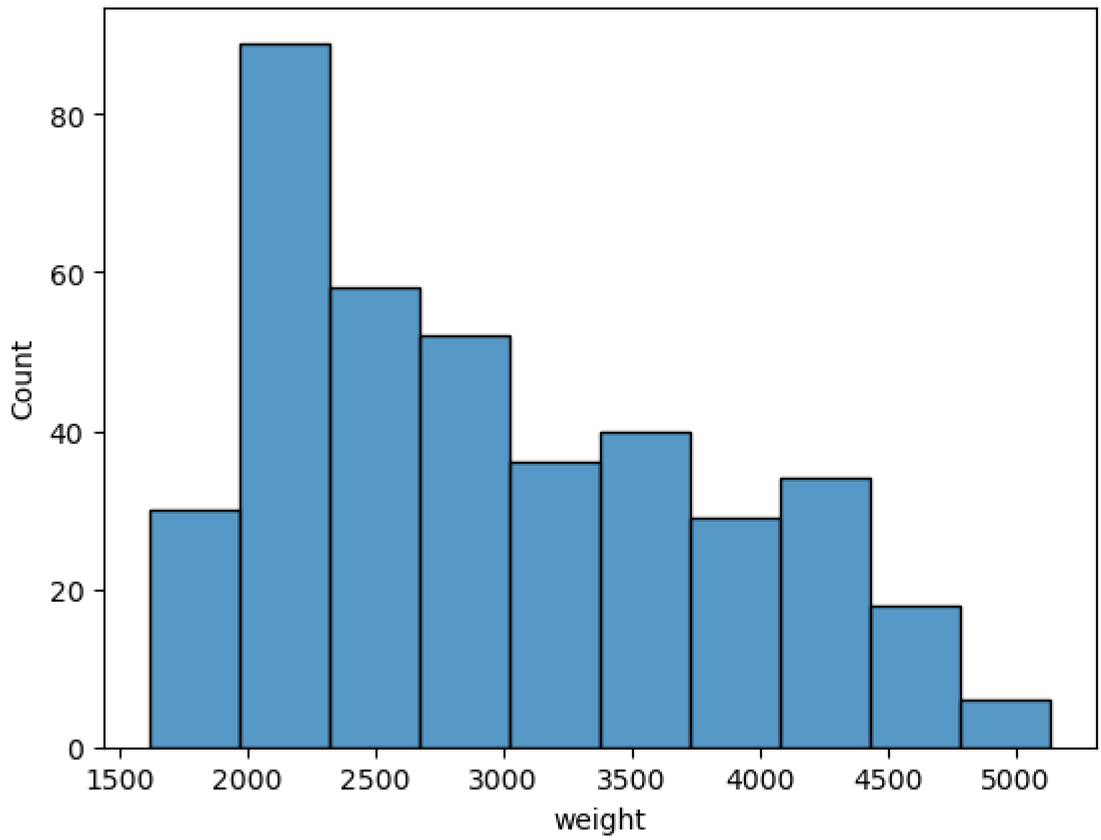
Out[24]:
```
(array([30., 89., 58., 52., 36., 40., 29., 34., 18.,  6.]),
 array([1613. , 1965.7, 2318.4, 2671.1, 3023.8, 3376.5, 3729.2, 4081.9,
        4434.6, 4787.3, 5140. ]),
 <BarContainer object of 10 artists>)
```



Plot the same histogram using seaborn's `histplot`.

In [26]:
```
#---- Your code here-----_#
sns.histplot(auto['weight'])
```

Out[26]:
```
<Axes: xlabel='weight', ylabel='Count'>
```

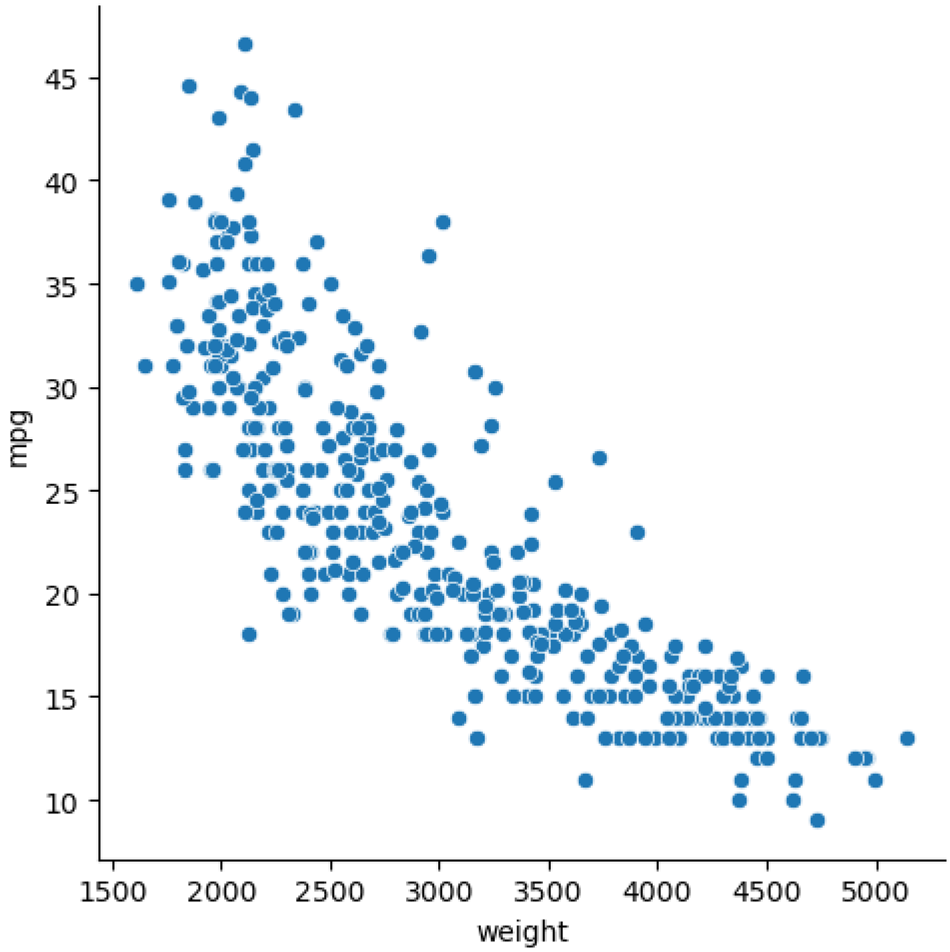


The next useful tool is to see data points scattered with each other in 2 dimensions.

✅ **DO THIS:** Draw a scatter plot of the `weight` variable vs `mpg` variable.

In [27]:
```
# This command should get you approximately the same thing,
# but with the added perk of automatically labeling axes
sns.relplot(x="weight", y="mpg", data=auto);
```

```
c:\Users\siony\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

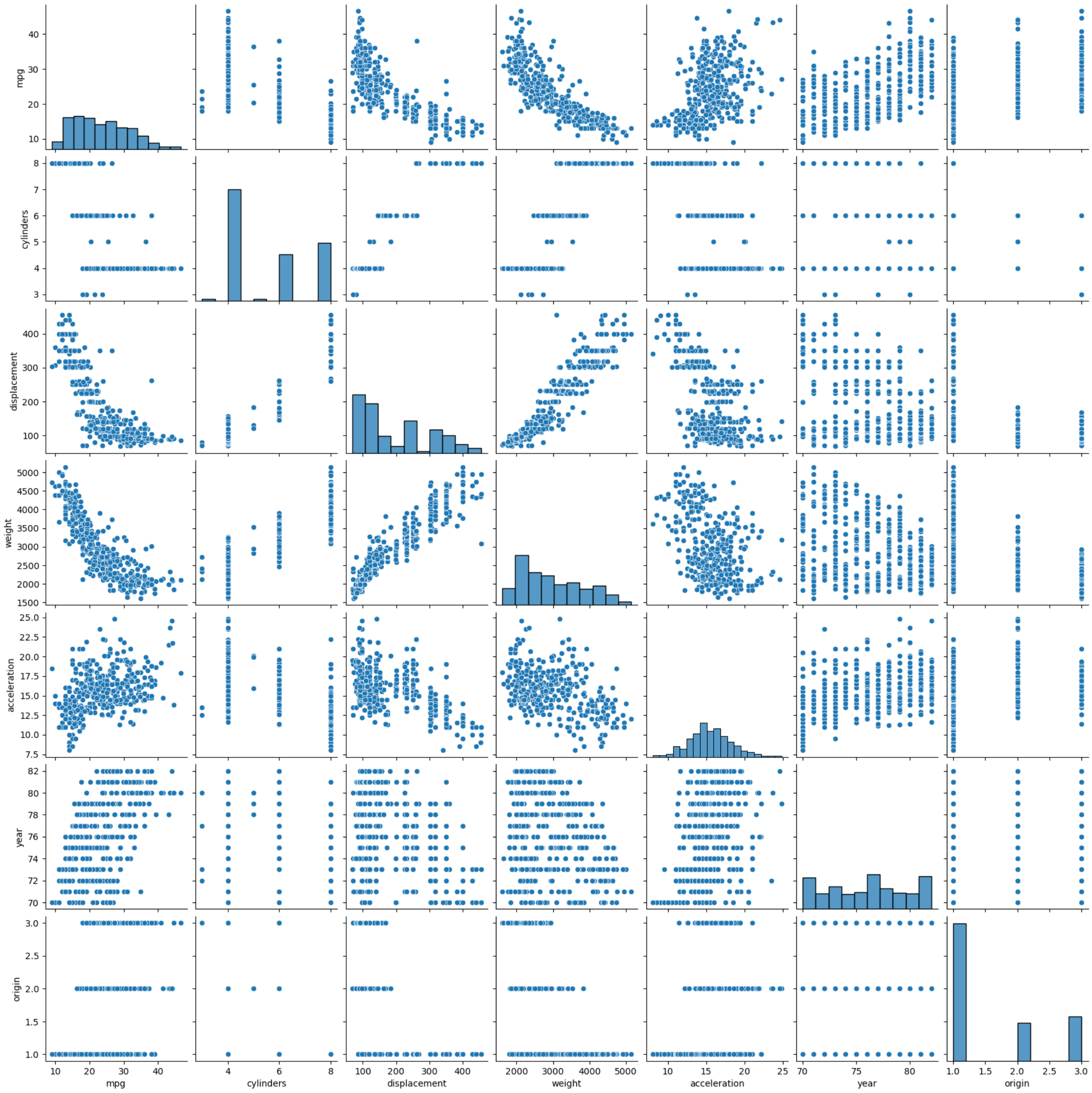


Now here is what I think is one of the most useful tools in seaborn for use when you're starting to understand a dataset....

In [28]:
```
sns.pairplot(auto)
```

```
c:\Users\siony\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[28]:
```
<seaborn.axisgrid.PairGrid at 0x1d02e4c6e50>
```



✅ **Q:** What is each graph on this giant grid showing you?

comareing each with the best fit graph

Put your answer to the above question here.

A note on the `ISLP` package

The new version of the textbook has been setup in python, and with it they have setup a python package with the stuff needed for the labs.

- Documentation of ISLP
- Installation Instructions

In theory, you should be able to get it running using the command

```
{bash}
pip install ISLP
```

however this caused a bunch of headaches when I tried to do it on my machine.

For the moment, I'm not going to try to set up all labs without using this package. The issue is that their package is VERY restrictive about versions of dependent packages (`numpy` , `matplotlib` , etc), which means you would need to downgrade many standard packages on your system, or be comfortable with using conda environments. If that is something you are comfortable with, you can still follow the Installation Instructions to install the package.

Congratulations, we're done!

Written by Dr. Liz Munch, Michigan State University

In [ ]:

```

```