HW 2

This assignment covers several aspects of Linear Regresstion. **DO NOT ERASE MARKDOWN CELLS AND INSTRUCTIONS IN YOUR HW submission**

- **Q** - QUESTION
- **A** - Where to input your answer

Instructions

Keep the following in mind for all notebooks you develop:

- Structure your notebook.
- Use headings with meaningful levels in Markdown cells, and explain the questions each piece of code is to answer or the reason it is there.
- Make sure your notebook can always be rerun from top to bottom.
- Please start working on this assignment as soon as possible. If you are a beginner in Python this might take a long time. One of the objectives of this assignment is to help you learn python and scikit-learn package.
- Follow README.md for homework submission instructions

Tutorials

- scikit-learn linear model
- train-test-split
- least squares fitting
- Linear Regression
- Seaborn

REGRESSION TASK USING SKLEARN

In jupyter notebook environment, commands starting with the symbol % are magic commands or magic functions. `%%timeit` is one of such function. It basically gives you the speed of execution of certain statement or blocks of codes.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
```

Data Get the exploratory data and the folowwing files:
https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data
https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.names

or Use from our 2023Fall/data repository folder

- Link should automatically download the data
- copy them in your HW folder

- If you are using command line: `>> wget https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data`
  - If wget is not working
    - dowload it from link
    - follow steps

Q1 Read the data using pandas, and replace the ??? in the code cell below to accomplish this taks. Note that auto-mpg.data does not have the column headers. use auto-mpg.names file to provide column names to the dataframe.

A1

In [ ]:
```
# Replace ??? with code in the code cell below
column_names = ['mpg','cylinders','displacement','horsepower','weight','acceleratio
df = pd.read_csv('../data/auto-mpg.data', names=column_names, na_values = "?", comm
```

In [ ]:
```
# View head of the data to confirm the correctness of your answer
df.head()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	NaN
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	NaN
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	NaN
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	NaN
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	NaN

Data cleaning and manipulation

Use

Q2 Data cleaning and manipulation:

1. use `pandas.info()` method to find columns with large number of NaN values
2. remove the column with NaN values
3. Check if there are still NaN values in the dataframe using `isna()` method

A2 Replace ??? with code in the code cell below

In [ ]:
```
#1. use pandas.info() method to find columns with large number of NaN values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      0 non-null      float64
dtypes: float64(6), int64(3)
memory usage: 28.1 KB
```

In [ ]:
```
#2. remove the column with NaN values - replace ??? with code
df.drop(columns=['car name'], inplace=True)
# Print head
df.head()
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1

In [ ]:
```
#3. Check if there are still NaN values in the dataframe using ```isna()``` method
print(df.isna())
# drop if any left or replace Nan values
df['horsepower'] = df['horsepower'].fillna(130.0)
```

```
      mpg  cylinders  displacement  horsepower  weight  acceleration  \
0    False      False         False       False   False         False
1    False      False         False       False   False         False
2    False      False         False       False   False         False
3    False      False         False       False   False         False
4    False      False         False       False   False         False
..     ...        ...           ...         ...     ...           ...
393  False      False         False       False   False         False
394  False      False         False       False   False         False
395  False      False         False       False   False         False
396  False      False         False       False   False         False
397  False      False         False       False   False         False

     model year  origin
0         False   False
1         False   False
2         False   False
3         False   False
4         False   False
..          ...     ...
393       False   False
394       False   False
395       False   False
396       False   False
397       False   False

[398 rows x 8 columns]
```

In [ ]:

```
# Checking to see if NaN values from 'horsepower' are replaced
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
dtypes: float64(5), int64(3)
memory usage: 25.0 KB
```

In [ ]:

```
#Print Tail
df.tail(15)
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
383	38.0	4	91.0	67.0	1965.0	15.0	82	3
384	32.0	4	91.0	67.0	1965.0	15.7	82	3
385	38.0	4	91.0	67.0	1995.0	16.2	82	3
386	25.0	6	181.0	110.0	2945.0	16.4	82	1
387	38.0	6	262.0	85.0	3015.0	17.0	82	1
388	26.0	4	156.0	92.0	2585.0	14.5	82	1
389	22.0	6	232.0	112.0	2835.0	14.7	82	1
390	32.0	4	144.0	96.0	2665.0	13.9	82	3
391	36.0	4	135.0	84.0	2370.0	13.0	82	1
392	27.0	4	151.0	90.0	2950.0	17.3	82	1
393	27.0	4	140.0	86.0	2790.0	15.6	82	1
394	44.0	4	97.0	52.0	2130.0	24.6	82	2
395	32.0	4	135.0	84.0	2295.0	11.6	82	1
396	28.0	4	120.0	79.0	2625.0	18.6	82	1
397	31.0	4	119.0	82.0	2720.0	19.4	82	1

Q3:

1. Convert following columns 'cylinders', 'year', 'origin' to dummy variable using pandas get_dummies() function
2. Do data normalization on real value/continous columns
   - The formula for normalization is: (Col_value- Mean of the col)/ Standard Deviation of the col

A3 Replace ??? with code in the code cell below

In [ ]:

```
# 1. Convert following columns 'cylinders', 'year', 'origin'  to dummy variable usi
cols = ['cylinders','model year','origin']
df_dummies = pd.get_dummies(df, columns=cols, prefix=['cylinders','model year','ori

#show the head
df_dummies.head()
```

Out[ ]:

	mpg	displacement	horsepower	weight	acceleration	cylinders-3	cylinders-4	cylinders-5	cyl
0	18.0	307.0	130.0	3504.0	12.0	0	0	0	
1	15.0	350.0	165.0	3693.0	11.5	0	0	0	
2	18.0	318.0	150.0	3436.0	11.0	0	0	0	
3	16.0	304.0	150.0	3433.0	12.0	0	0	0	
4	17.0	302.0	140.0	3449.0	10.5	0	0	0	

5 rows × 26 columns

In [ ]:

```
# 2. Do data normalization on real value/continous columns
realcols = ['mpg','displacement','horsepower','weight','acceleration']

for col in realcols:
  mean = df[col].mean()
  std = df[col].std()
  df[col] = (df[col] - mean)/std
```

Regression Task

Given all the information we will try to predict mpg - miles per gallon. The First step toward predicting the mpg from the dataset is to find the correlation between the columns/features of the dataset.

Q4

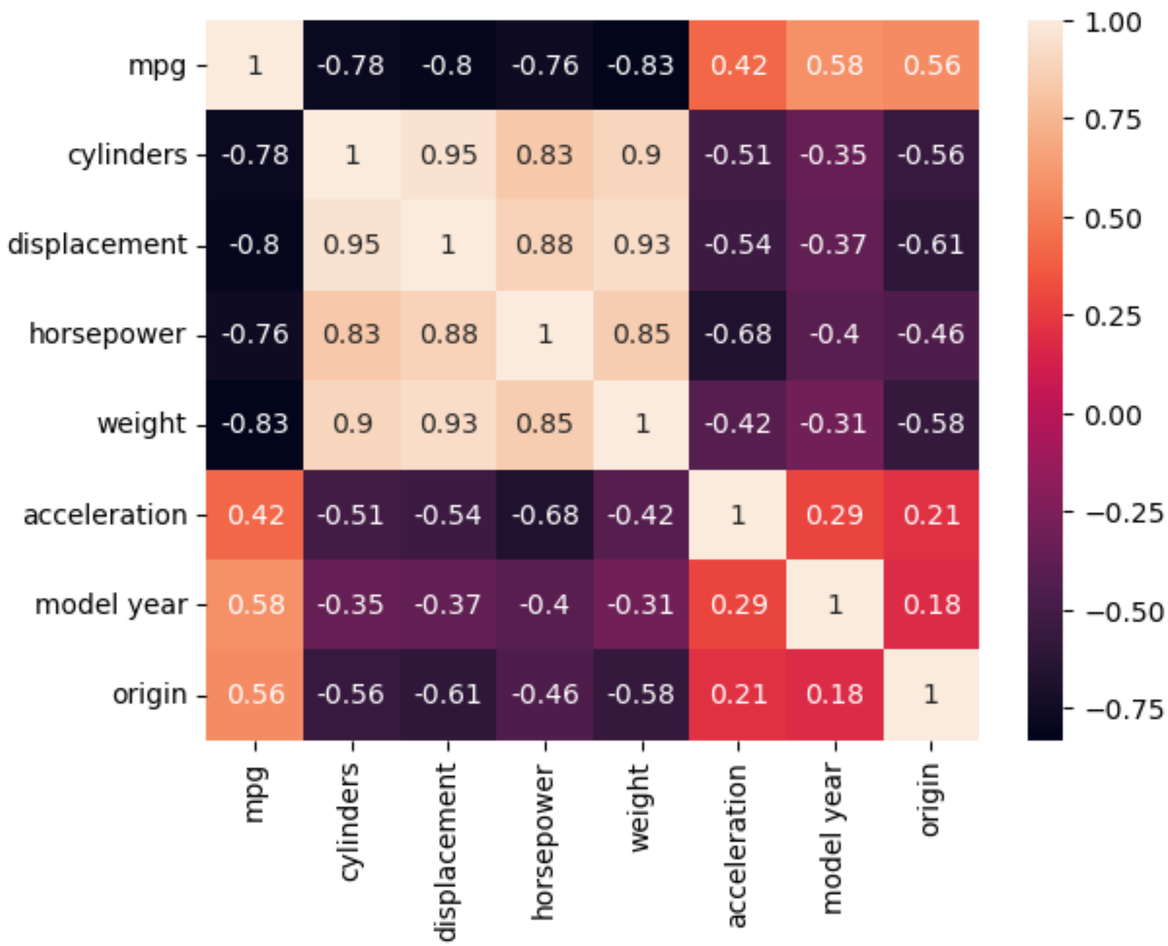
1. Use heatmap chart from seaborn library to findout the correlation between the columns.
2. Which of the columns is mostly related to mpg column and why?

In [ ]:

```
# A4 code goes below
corr = df.corr()
sns.heatmap(corr, annot=True)
```

Out[ ]:

```
<Axes: >
```



A4

The 'weight' class is the most correlated class to mpg , which is indicated by the highest absolute value (shown with annot=True).

Q5

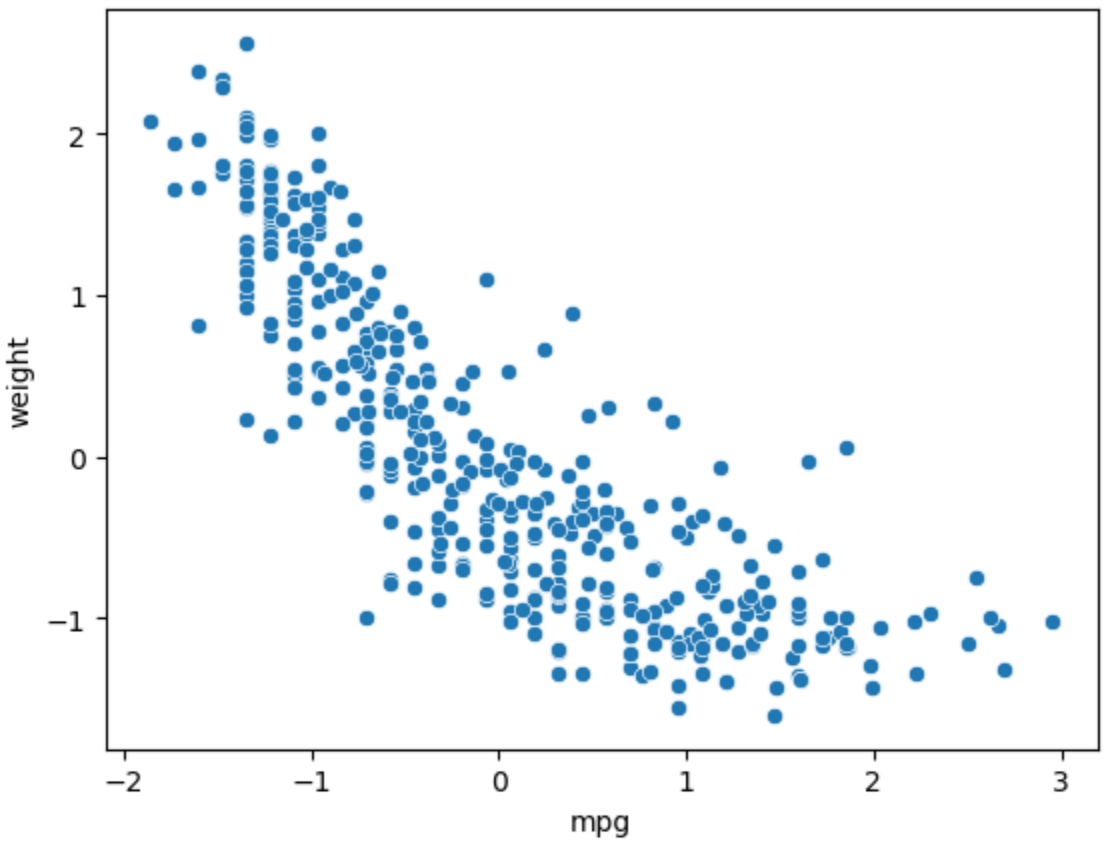
1. Draw a lineplot or scattered plot between mpg and your answer from the above cell.
2. Use pairplot from sns to plot our data frame df for better understanding of your selection
   - NOTE: 2. should inform 3.
3. Choose a set of columns/ features based on pairplot and heatmap for the mpg prediction.

- Justify your answer using some explanation from the heatmap and pairplot graph formulated from the dataset.

A5 For 1. and 2. replace ??? with code in the code cell below.

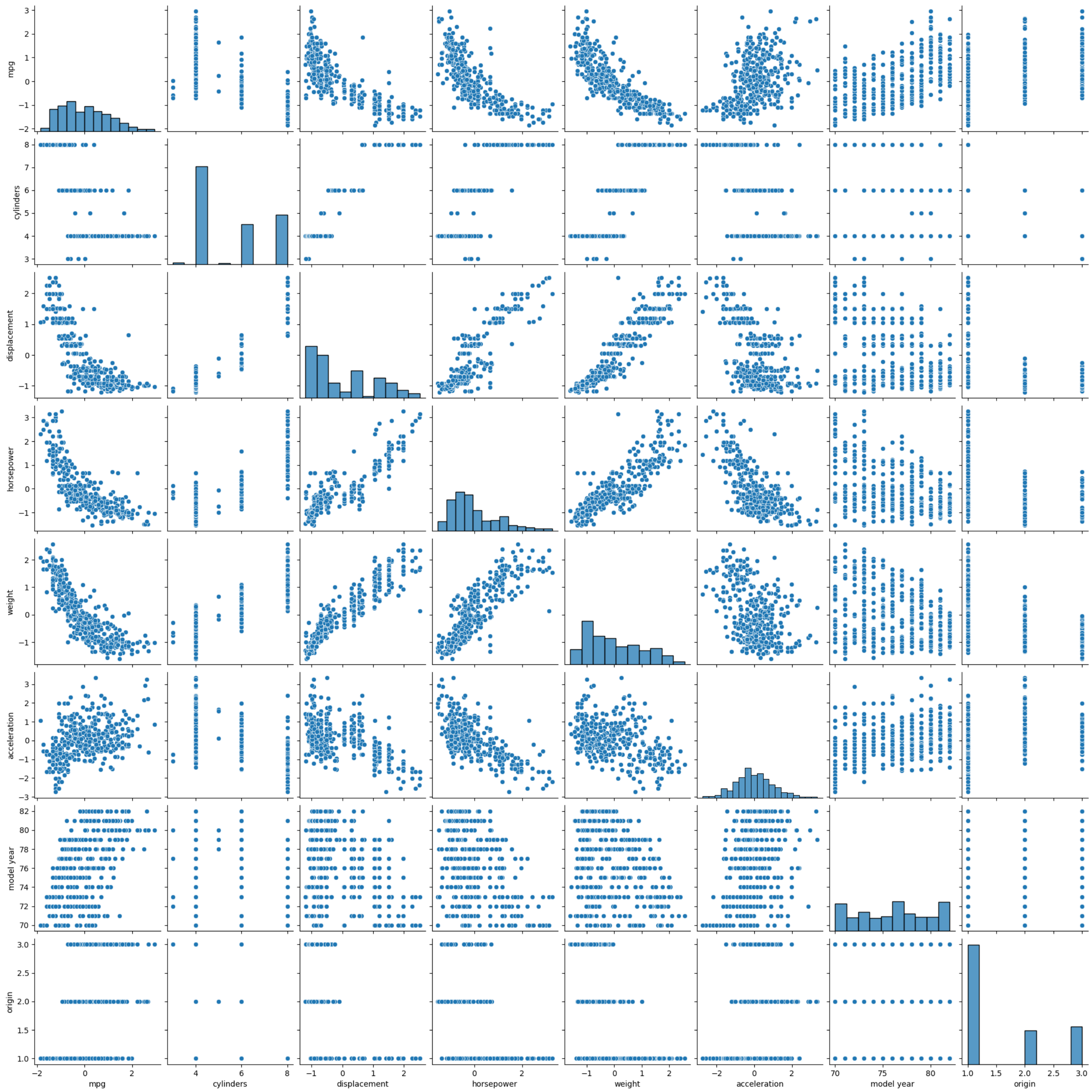
In [ ]:
```
# 1. Draw a lineplot or scattered plot between mpg and your answer from the above c
sns.scatterplot(df, x=df['mpg'], y=df['weight'])
```

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



In [ ]:
```
# 2. Use pairplot from sns to plot our data frame df for better understanding of yo
sns.pairplot(df)
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x203182f7290>



A5
'horsepower' also appears to be a contender for best, but 'weight' looks a bit more linear so I still believe it would be best.

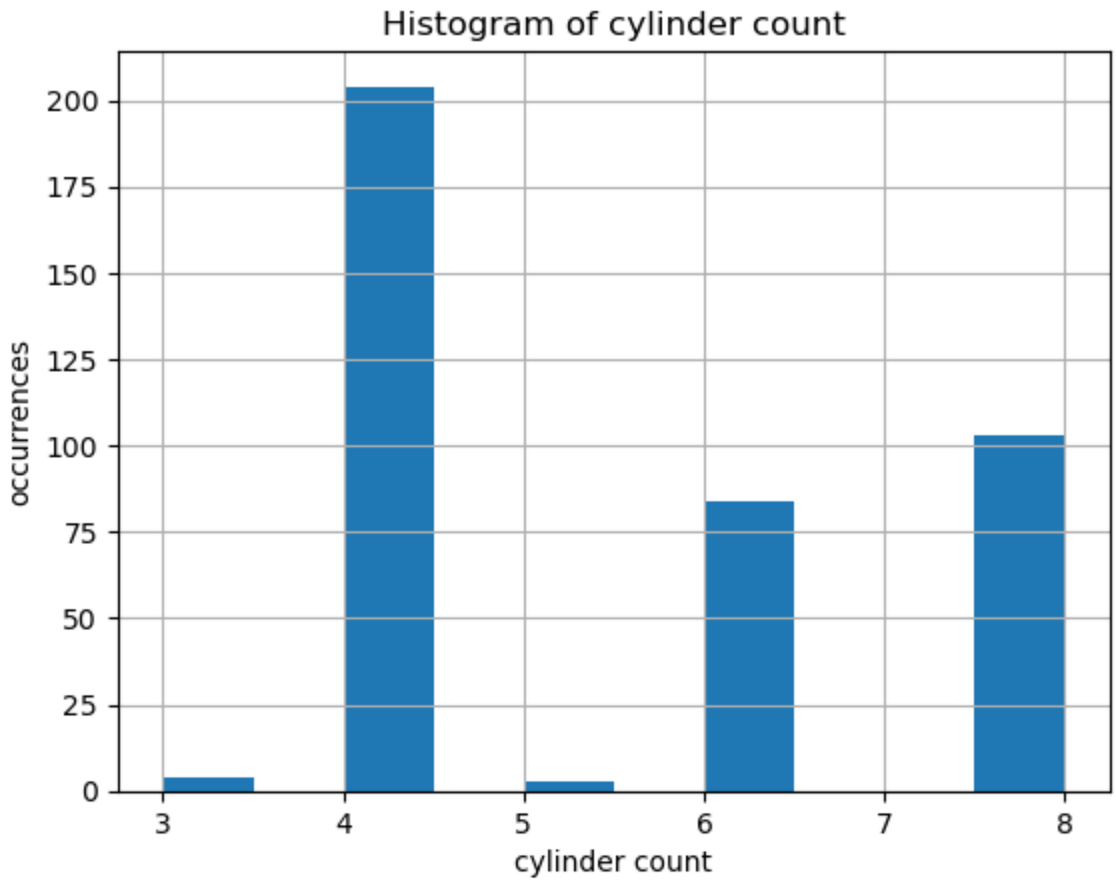
Q6 Data Visualization:

1. Now, create a histogram which represents number items with per cylinder class

A6 Replace ??? with code in the code cell below

In [ ]:
```
import matplotlib.pyplot as plt

df.hist(column='cylinders')
plt.title("Histogram of cylinder count")
plt.xlabel('cylinder count')
plt.ylabel('occurrences')
plt.show()
```



Data Preparation

Q7 Assign mpg column value to y and rest columns to x, remember x shouldn't have mpg

A7 Replace ??? with code in the code cell below

In [ ]:
```
y = df.mpg.values
df.drop(columns=['mpg'], inplace=True)
x = df.values
```

Q8 Use train_test_split to split the data set as train:test=(80%:20%) ratio.

A8 Replace ??? with code in the code cell below

In [ ]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.20)
# View the shape of your data set
xtrain.shape, xtest.shape, ytrain.shape, ytest.shape
```

Out[ ]:
```
((318, 7), (80, 7), (318,), (80,))
```

Q9 Follow examples from references given in the top of this notebook

- Note:Use linear model to fit regression line and plot
- Our linear model will be of following type
- Y = b + coef0*x0* + *coef1*x1+coef2*x2+...........

A9: Replace ??? with code in the code cell below

In [ ]:
```
from sklearn import linear_model

reg = linear_model.LinearRegression()
reg.fit(xtrain, ytrain)

#Now view the coefficient use .coef_ and shape of .coef_
print(reg.coef_)
print(reg.intercept_)
print(reg.coef_.shape[0])
```

```
[-0.0674672   0.22894077  0.00965163 -0.73855098  0.03137155  0.09595259
  0.17804421]
-7.2052069786155375
7
```

Q10 Relates to the code in the cell below. Why the printed values the same?

In [ ]:
```
# Now if you view
print(f'{reg.coef_.shape[0]},{xtrain.shape[1]}, ', f'are equal? {reg.coef_.shape[0]
```

```
7,7,  are equal? True
```

A10 The printed values are the same because the LinearRegression model has a coefficient for every column of the xtrain training set

Model Scoring

In [ ]:
```
# Model Score
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(xtrain, ytrain)
reg.score(xtest,ytest)

# Calculate the score on train and test sets
# Your code goes below
reg.score(xtrain,ytrain), reg.score(xtest,ytest)
```

Out[ ]:
```
(0.832277997368469, 0.7792078194462327)
```

Q11 Each of the sklearn models have different model evaluations core value.

- LinearRegression documentation
- More on model_evaluation

Explain what's the meaning of reg.score return value in this notebook.

A11 When we call score in this notebook, the model predicts ytrain/test using xtrain/test and then compares the accuracy of its prediction to the ytrain/test parameter.

In [ ]:
```
# A custom function to calculate r2 score
# Details on the custom scorers: https://scikit-learn.org/stable/modules/model_eval

def r2score_(ytrue, ypred):
  rss = ((ytrue - ypred)**2).sum()
  tss = ((ytrue - ytrue.mean()) ** 2).sum()
  r2 = 1 - rss/tss
  return r2

# Now do prediction on xtrain and xtest and check your r2 score by printing rescore
trainpredict = reg.predict(xtrain)
testpredict = reg.predict(xtest)

print(r2score_(ytrain, trainpredict), r2score_(ytest, testpredict))

# You can see that reg.score values and your custom function output are matched
```

```
0.832277997368469 0.7792078194462327
```

One way of achieving linear regression is by minimizing the error between actual y and predicted y. The method is known as least square method. We will make our custom least square optimize to calculate model parameters that minimizes output error.

Q12 Write a function which takes weights(or params), x and y and do following

- 1. calculate dot product between x and params , which is ypredicted
- 1. calculate difference between actual y and ypredicted
- 1. return the difference

A12 complete the code below

In [ ]:
```
import scipy.optimize as optimization
from sklearn.metrics import r2_score

def constraint(params, x, y):
  ypred = x@params
  return y-ypred

# Our initial params is a vector of size equal to dimension of x, or you can say nu
# You can create zeros vector using np.zeros(size)

# complete code
params = np.zeros(x.shape[1])
```

In [ ]:
```
# Now study the documentation and complete following code
params, _ = optimization.leastsq(constraint, params[:], args=(x,y))
```

In [ ]:
```
# Now we have parameter or weight we can now create our model
model = lambda x:np.dot(x,params)
```

In [ ]:
```
# Now predict ytrain using model and see first 5 predicted and actual values
ypred_train = model(xtrain)
# see first 5 predicted values
print(ypred_train[:5])
# see first 5 actual values
print(ytrain[:5])
```

```
[ 0.18072655  0.65001268 -0.34028627 -1.14018253 -0.81904368]
[ 1.64859941  1.20079913 -0.38569331 -1.34526535 -1.08937947]
```

In [ ]:
```
# Now predict ytest using model and see first 5 predicted and actual values
ypred_test = model(xtest)
print(ypred_test[:5])
print(ytest[:5])
```

```
[ 0.10217579 -0.1071802   0.92351151  0.50938303  0.4400872 ]
[-0.1937789  -0.42407619  0.70182167  0.44593579  0.19004991]
```

In [ ]:
```
# Now use custom made r2score calculator to calculate r2 score on both train and te
print(r2score_(ytest, ypred_test), r2score_(ytrain, ypred_train))
```

```
0.7094476734696706 0.7439375008506243
```

In [ ]:
```
# Now use sklearn build-in r2score calculator to calculate r2 score on both train a
print(r2_score(ytrain, ypred_train)), print(r2_score(ytest, ypred_test))
```

```
0.7439375008506243
0.7094476734696706
```

Out[ ]:
```
(None, None)
```