In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df=pd.read_csv(r'https://github.com/YBI-Foundation/Dataset/raw/main/MPG.csv')
```

In [3]:
```
df
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrole chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel ss
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	..
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang g
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford range
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s 10

398 rows × 9 columns

In [4]:
```
df.shape
```

Out[4]: (398, 9)

In [5]:
```
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    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    object 
 8   name          398 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

In [7]:
```
df.nunique() #mpg is milage per gallon
#we used it to find the nums of categorical cols, dont look for continuous ones
```

Out[7]:
```
mpg             129
cylinders         5
displacement     82
horsepower       93
weight          351
acceleration     95
model_year       13
origin            3
name            305
dtype: int64
```

Data preprocessing

generally to handle missing values

In [8]:
```
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    int64  
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64  
 7   origin        398 non-null    object 
 8   name          398 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

In [9]:
```
#above, horsepower has missing values
#since, only 6 values are missing, we shall go for dropping theminstead of replac
#displacement is volume of cylinder
```

In [10]:
```
df.describe() #gives summary stats
```

Out[10]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

In [11]:
```
#here mean varies too much for respective attributes so, we need to perform stand
```

In [12]:
```
df.corr() # correlation
```

Out[12]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

In [13]:
```
#cyl and disp have high co relation(95%), so we use either of two highly corelate
```

In [14]:
```
df=df.dropna()
```

In [15]:
```
df.info()
```

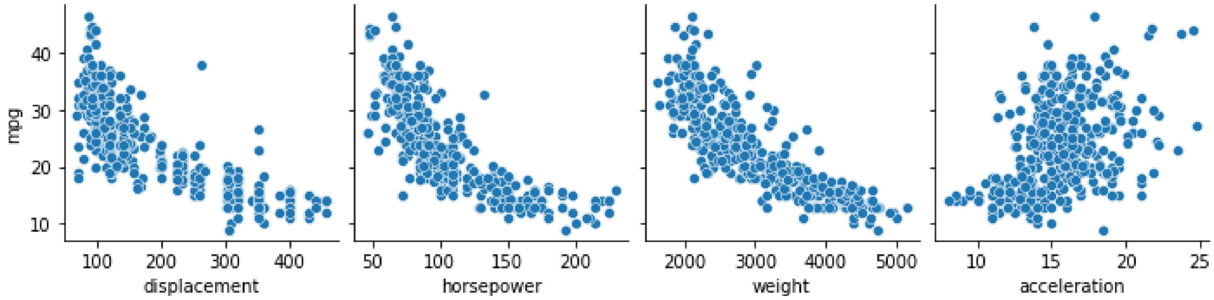
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64  
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    int64  
 5   acceleration  392 non-null    float64
 6   model_year    392 non-null    int64  
 7   origin        392 non-null    object 
 8   name          392 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 30.6+ KB
```

In [16]:
```
#disp(aka cc in automobile technical term)=no.of cyl * vol of cyl
```

In [17]:
```
import seaborn as sns
```

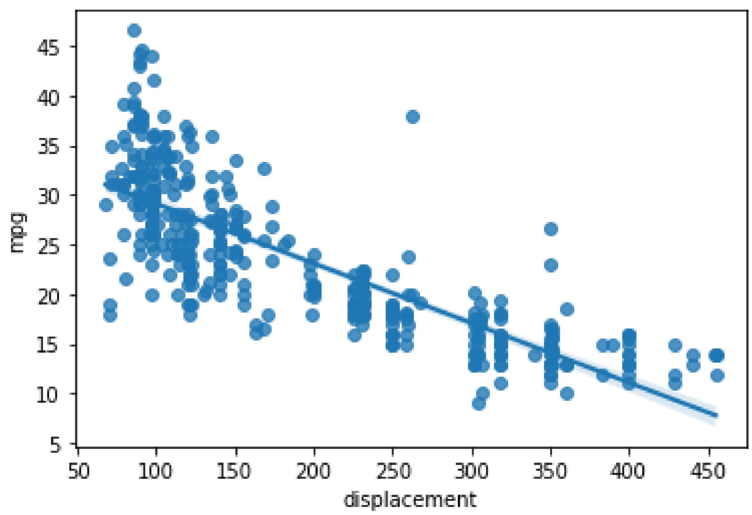
In [18]:
```
sns.pairplot(df,x_vars=['displacement','horsepower','weight','acceleration'],y_va
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x7fd58477d9d0>



In [19]:
```
# see above, all 3 have -ve correlation and acceleration has scattered correlatio
```

In [20]:
```
sns.regplot(x='displacement',y='mpg',data=df);
```



In [21]:
```
#the same can be observed from the above regression line plot

#there is a negative correlation, it is high, not linear but a curvature is obser
#let's first try to create a linear relation then go with polynomial and then con

#displacement aka size of engine or volume of engine or engine capacity or cc
```

Define target variables and features

In [22]:
```
df.columns
```

Out[22]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

In [23]:
```
y=df['mpg']
```

In [24]:
```
y.shape
```

Out[24]:
```
(392,)
```

In [25]:
```
X=df[['displacement', 'horsepower', 'weight',
       'acceleration']]
```

In [26]:
```
 X.shape
```

Out[26]:
```
(392, 4)
```

In [27]:
```
X
```

Out[27]:

	displacement	horsepower	weight	acceleration
0	307.0	130.0	3504	12.0
1	350.0	165.0	3693	11.5
2	318.0	150.0	3436	11.0
3	304.0	150.0	3433	12.0
4	302.0	140.0	3449	10.5
...	...	...	...	...
393	140.0	86.0	2790	15.6
394	97.0	52.0	2130	24.6
395	135.0	84.0	2295	11.6
396	120.0	79.0	2625	18.6
397	119.0	82.0	2720	19.4

392 rows × 4 columns

Scaling the Data

In [28]:
```
from sklearn.preprocessing import StandardScaler
```

In [29]:
```
ss=StandardScaler()
```

In [30]:
```
X=ss.fit_transform(X)
```

after scaling-

In [31]:
```
pd.DataFrame(X).describe()  #mean=0,standard_deviation=1 can also be observed-
```

Out[31]:

	0	1	2	3
count	3.920000e+02	3.920000e+02	3.920000e+02	3.920000e+02
mean	-2.537653e-16	-4.392745e-16	5.607759e-17	6.117555e-16
std	1.001278e+00	1.001278e+00	1.001278e+00	1.001278e+00
min	-1.209563e+00	-1.520975e+00	-1.608575e+00	-2.736983e+00
25%	-8.555316e-01	-7.665929e-01	-8.868535e-01	-6.410551e-01
50%	-4.153842e-01	-2.853488e-01	-2.052109e-01	-1.499869e-02
75%	7.782764e-01	5.600800e-01	7.510927e-01	5.384714e-01
max	2.493416e+00	3.265452e+00	2.549061e+00	3.360262e+00

features after scaling(mean=0,standard_deviation=1) -

In [32]:
```
X
```

Out[32]:
```
array([[ 1.07728956,  0.66413273,  0.62054034, -1.285258  ],
       [ 1.48873169,  1.57459447,  0.84333403, -1.46672362],
       [ 1.1825422 ,  1.18439658,  0.54038176, -1.64818924],
       ...,
       [-0.56847897, -0.53247413, -0.80463202, -1.4304305 ],
       [-0.7120053 , -0.66254009, -0.41562716,  1.11008813],
       [-0.72157372, -0.58450051, -0.30364091,  1.40043312]])
```

In [33]:
```
#standard scaler is returning -ve values can be verified through Z-Score formula
```

Train test split data

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,train_size=0.7,random_state=25
```

In [36]:
```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[36]:
```
((274, 4), (118, 4), (274,), (118,))
```

Linear Regression Model

In [37]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
lr=LinearRegression()
```

In [39]:
```
lr.fit(X_train,y_train)
```

Out[39]:
```
LinearRegression()
```

In [40]:
```
lr.intercept_
```

Out[40]:
```
23.485738559737584
```

In [41]:
```
lr.coef_
```

Out[41]:
```
array([-1.05767743, -1.68734727, -4.10787617, -0.11495177])
```

In [42]:
```
#-ve coeff also verifies the -ve correlation observed before
```

Thus, regression equation: 23.4(intercept) -1.05(displacement) -1.68(horsepower) -4.10(weight) -0.115(acceleration) +error

i.e. keeping all vars constant, a unit change in displacement leads to decrement of mileage by 1.05

Predict Test Data

In [43]:
```
y_pred=lr.predict(X_test)
```

In [44]:
```
y_pred
```

Out[44]:
```
array([18.51865637, 15.09305675, 14.30128789, 23.6753321 , 29.7546115 ,
       23.68796629, 26.61066644, 24.56692437, 15.06260986, 11.94312046,
       24.08050053, 27.96518468, 31.66130278, 31.01309132, 18.32428976,
       19.32795009, 28.08847536, 32.1506879 , 31.15859692, 27.15792144,
       18.82433097, 22.54580176, 26.15598115, 32.36393869, 20.74377679,
        8.78027518, 22.19699435, 18.20614294, 25.00052718, 15.26421552,
       23.13441082, 17.10542257,  9.87180062, 30.00790415, 20.41204655,
       29.11860245, 24.4305187 , 21.72601835, 10.51174626, 13.12426391,
       21.41938406, 19.96113872,  6.19146626, 17.79025345, 22.5493033 ,
       29.34765021, 13.4861847 , 25.88852083, 29.40406946, 22.41841964,
       22.07684766, 16.46575802, 24.06290693, 30.12890046, 10.11318121,
        9.85011438, 28.07543852, 23.41426617, 20.08501128, 30.68234133,
       20.92026393, 26.78370281, 22.9078744 , 14.15936872, 24.6439883 ,
       26.95515832, 15.25709393, 24.11272087, 30.80980589, 14.9770217 ,
       27.67836372, 24.2372919 , 10.92177228, 30.22858779, 30.88687365,
       27.33992044, 31.18447082, 10.8873597 , 27.63510608, 16.49231363,
       25.63229888, 29.49776285, 14.90393439, 32.78670687, 30.37325244,
       30.9262743 , 14.71702373, 27.09633246, 26.69933806, 29.06424799,
       32.45810182, 29.44846898, 31.61239999, 31.57891837, 21.46542321,
       31.76739191, 26.28605476, 28.96419915, 31.09628395, 24.80549594,
       18.76490961, 23.28043777, 23.04466919, 22.14143162, 15.95854367,
       28.62870918, 25.58809869, 11.4040908 , 25.73334842, 30.83500051,
       21.94176255, 15.34532941, 30.37399213, 28.7620624 , 29.3639931 ,
       29.10476703, 20.44662365, 28.11466839])
```

Model Accuracy

In [45]:
```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error,r
```

In [46]:
```
mean_absolute_error(y_test,y_pred)
```

Out[46]:
```
3.3286968643244106
```

In [47]:
```
mean_absolute_percentage_error(y_test,y_pred)
```

Out[47]:
```
0.14713035779536746
```

In [48]:
```
r2_score(y_test,y_pred)
```

Out[48]:
```
0.7031250746717692
```

In [49]:
```
#our model able to explain 70% of variance in model or R-square. Yayyy!!
```

In [49]:
```
```

Polynomial Regression

In [50]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [51]:
```
poly=PolynomialFeatures(degree=2,interaction_only=True,include_bias=False)
```

In [52]:
```
X_train2=poly.fit_transform(X_train)
```

In [53]:
```
X_test2=poly.fit_transform(X_test)
```

In [54]:
```
lr.fit(X_train2,y_train)
```

Out[54]:
```
LinearRegression()
```

In [55]:
```
lr.intercept_
```

Out[55]:
```
21.27336450063766
```

In [56]:
```
lr.coef_
```

Out[56]:
```
array([-2.76070596, -5.00559628, -1.36884133, -0.81225214,  1.24596571,
       -0.12475017, -0.90542822,  1.35064048, -0.17337823,  1.41680398])
```

In [57]:
```
y_pred_poly=lr.predict(X_test2)
```

Model Accuracy:

In [58]:
```
from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,r2
```

In [59]:
```
mean_absolute_error(y_test,y_pred_poly)
```

Out[59]:
```
2.7887147720295977
```

In [60]:
```
mean_absolute_percentage_error(y_test,y_pred_poly)
```

Out[60]:
```
0.1207401834293869
```

In [61]:
```
r2_score(y_test,y_pred_poly)
```

Out[61]:
```
0.7461731314563803
```

In [62]:
```
#MAE was 3.2 but now, 2.7
#MAPE was 14% but now, 12%
#r2- increased by around 5%
```

In [ ]: