YBI Foundation

Mileage_Prediction

In [103…
```
import pandas as pd
```

In [10]:
```
import numpy as np
```

In [13]:
```
# Step 2 : import data
df = pd.read_csv('https://github.com/ybifoundation/Dataset/raw/main/MPG.csv')
```

In [14]:
```
df.head()
```

Out[14]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	
0	18.0	8	307.0	130.0	3504	12.0	70	usa	c
1	15.0	8	350.0	165.0	3693	11.5	70	usa	
2	18.0	8	318.0	150.0	3436	11.0	70	usa	p
3	16.0	8	304.0	150.0	3433	12.0	70	usa	
4	17.0	8	302.0	140.0	3449	10.5	70	usa	

In [18]:
```
df.nunique()
```

Out[18]:
```
mpg             129
cylinders         5
displacement     82
horsepower       93
weight          351
acceleration     95
model_year       13
origin            3
name            305
dtype: int64
```

Data processig

In [19]:
```
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 [20]:
```
df.describe()
```

Out[20]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	mode
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.0
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.0
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.6
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.0
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.0
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.0
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.0
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.0

In [41]:
```
df_numeric = df.apply(pd.to_numeric, errors='coerce')
correlation_matrix = df_numeric.corr()

print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                    mpg  cylinders  displacement  horsepower    weight  \
mpg            1.000000  -0.777618     -0.805127   -0.778427 -0.832244
cylinders     -0.777618   1.000000      0.950823    0.842983  0.897527
displacement  -0.805127   0.950823      1.000000    0.897257  0.932994
horsepower    -0.778427   0.842983      0.897257    1.000000  0.864538
weight        -0.832244   0.897527      0.932994    0.864538  1.000000
acceleration   0.423329  -0.504683     -0.543800   -0.689196 -0.416839
model_year     0.580541  -0.345647     -0.369855   -0.416361 -0.309120
origin              NaN        NaN           NaN         NaN       NaN
name                NaN        NaN           NaN         NaN       NaN

              acceleration  model_year  origin  name
mpg               0.423329    0.580541     NaN   NaN
cylinders        -0.504683   -0.345647     NaN   NaN
displacement     -0.543800   -0.369855     NaN   NaN
horsepower       -0.689196   -0.416361     NaN   NaN
weight           -0.416839   -0.309120     NaN   NaN
acceleration      1.000000    0.290316     NaN   NaN
model_year        0.290316    1.000000     NaN   NaN
origin                 NaN         NaN     NaN   NaN
name                   NaN         NaN     NaN   NaN
```

In [25]:
```
df=df.dropna()
```

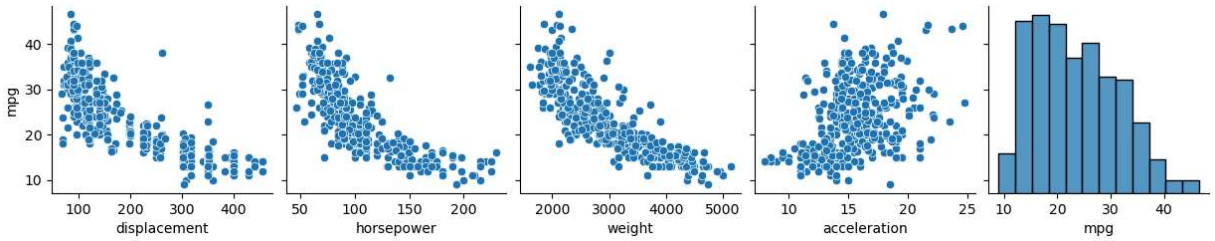
In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 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
```

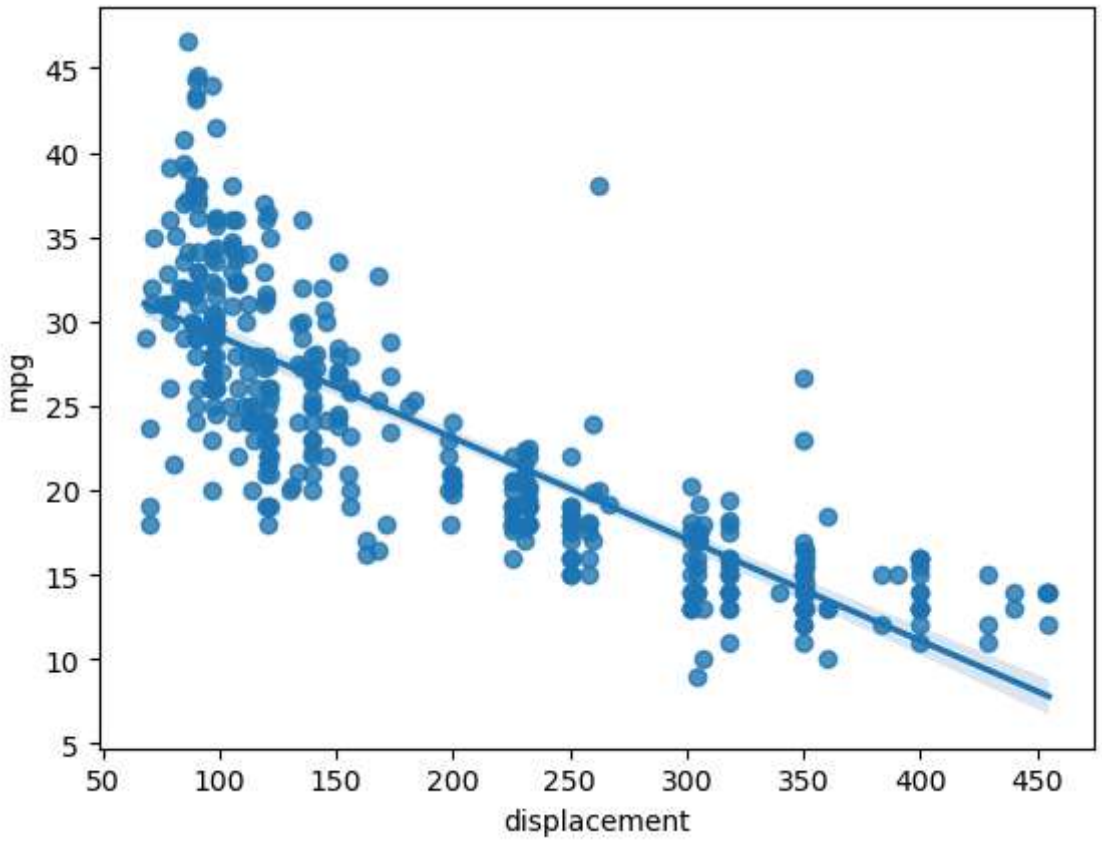
In [36]:
```
import seaborn as sns
```

In [37]:
```
sns.pairplot(df, x_vars=['displacement', 'horsepower', 'weight', 'acceleration', 'm
```

```
C:\Users\admin\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [44]:
```
sns.regplot(x ='displacement', y = 'mpg', data =df);
```



In [47]:
```
df.columns
```

Out[47]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

In [49]:
```
y=df['mpg']
```

In [50]:
```
y.shape
```

Out[50]:
```
(392,)
```

In [58]:
```
X=df[['displacement', 'horsepower', 'weight', 'acceleration']]
```

In [59]:
```
X.shape
```

Out[59]:
```
(392, 4)
```

In [60]:
```
X
```

Out[60]:

	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 Data

In [63]:
```
from sklearn.preprocessing import StandardScaler
```

In [65]:
```
ss = StandardScaler()
```

In [66]:
```
x = ss.fit_transform(X)
```

In [67]:
```
x
```

Out[67]:
```
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 [69]:
```
pd.DataFrame(X).describe()
```

Out[69]:

	displacement	horsepower	weight	acceleration
count	392.000000	392.000000	392.000000	392.000000
mean	194.411990	104.469388	2977.584184	15.541327
std	104.644004	38.491160	849.402560	2.758864
min	68.000000	46.000000	1613.000000	8.000000
25%	105.000000	75.000000	2225.250000	13.775000
50%	151.000000	93.500000	2803.500000	15.500000
75%	275.750000	126.000000	3614.750000	17.025000
max	455.000000	230.000000	5140.000000	24.800000

After Standardization Mean is Zero and Standard Deviation is One

Train Test Split Data

In [70]:
```
from sklearn.model_selection import train_test_split
```

In [71]:
```
X_train, X_test, y_train, y_test =  train_test_split(X, y, train_size = 0.7, random
```

In [72]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[72]:
```
((274, 4), (118, 4), (274,), (118,))
```

Linear Regression Model

In [73]:
```
from sklearn.linear_model import LinearRegression
```

In [74]:
```
lr=LinearRegression()
```

In [75]:
```
lr.fit(X_train, y_train)
```

Out[75]:
```
▾ LinearRegression
LinearRegression()
```

In [77]:
```
lr.intercept_
```

Out[77]:
```
45.105709426998416
```

In [78]:
```
lr.coef_
```

Out[78]:
```
array([-0.0101203 , -0.04389329, -0.00484238, -0.04171959])
```

Mileage=23.4 - 1.05 Displacement - 1.5 Horsepower - 4.10 Weight - 0.115 Acceleration + error

Predict Test Data

In [79]:
```
y_pred = lr.predict(X_test)
```

In [80]:
```
y_pred
```

Out[80]:
```
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 [81]:
```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error,r2_
```

In [83]:
```
mean_absolute_error(y_test, y_pred)
```

Out[83]:
```
3.3286968643244097
```

In [84]:
```
mean_absolute_percentage_error(y_test, y_pred)
```

Out[84]:
```
0.14713035779536743
```

In [85]:
```
r2_score(y_test, y_pred)
```

Out[85]:
```
0.7031250746717692
```

Polynomial Regression

In [86]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [90]:
```
poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
```

In [91]:
```
X_train2 = poly.fit_transform(X_train)
```

In [92]:
```
X_test2 = poly.fit_transform(X_test)
```

In [93]:
```
lr.fit(X_train2, y_train)
```

Out[93]:
```
LinearRegression
LinearRegression()
```

In [94]:
```
lr.intercept_
```

Out[94]:
```
83.65746234332917
```

In [95]:
```
lr.coef_
```

Out[95]:
```
array([-5.75868003e-03, -2.88386409e-01, -1.50870743e-02, -1.31734959e+00,
        3.10127340e-04, -1.40709212e-06, -3.14426164e-03,  4.14166062e-05,
       -1.63686388e-03,  6.06143149e-04])
```

In [97]:
```
y_pred_poly = lr.predict(X_test2)
```

Model Accuracy

In [99]:
```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error,r2_
```

In [101…
```
mean_absolute_percentage_error(y_test, y_pred_poly)
```

Out[101…
```
0.12074018342929002
```

In [102…
```
r2_score(y_test, y_pred_poly)
```

Out[102…
```
0.7461731314565869
```

In [ ]:
```

```