Mileage Prediction - Regression Analysis

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.

Data Set Information:

This dataset is a slightly modified version of the dataset provided in the StatLib library. In line with the use by Ross Quinlan (1983) in predicting the attribute "mpg",8 of the original instances were removed because they had unknown values for the "mpg" attribute. the original dataset is available in the file "auto-mpg.data-original". "The data concerned city-cycle fuel composition in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes."(Quinlan, 1993)

Attribute Information:

1.mpg: continuous

2.cylinders: multivalued discrete

3.displacement: continuous

4.horsepower: continuous

5.weight: continuous

6.acceleration: continuous

7.model year: multivalued discrete

8.origin: multivalued discrete

9.car name: string(unique for each instance)

Import Library

Double-click (or enter) to edit

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

Import Data

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

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origir
0	18.0	8	307.0	130.0	3504	12.0	70	usa
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
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

Next steps: Generate code with df View recommended plots

```
df.nunique()
```

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

Data Preprocessing

```
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
```

```
df.describe()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	mo
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	39
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	7
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	7
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	7
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	7
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	7
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	8

```
df.corr()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-115-2f6f6606aa2c> in <cell line: 1>()
----> 1 df.corr()

                              3 frames

/usr/local/lib/python3.10/dist-packages/pandas/core/internals/managers.py in
_interleave(self, dtype, na_value)
   1792             else:
   1793                 arr = blk.get_values(dtype)
-> 1794             result[rl.indexer] = arr
   1795             itemmask[rl.indexer] = 1
   1796

ValueError: could not convert string to float: 'usa'
```

Next steps: Explain error

Remove Missing Value

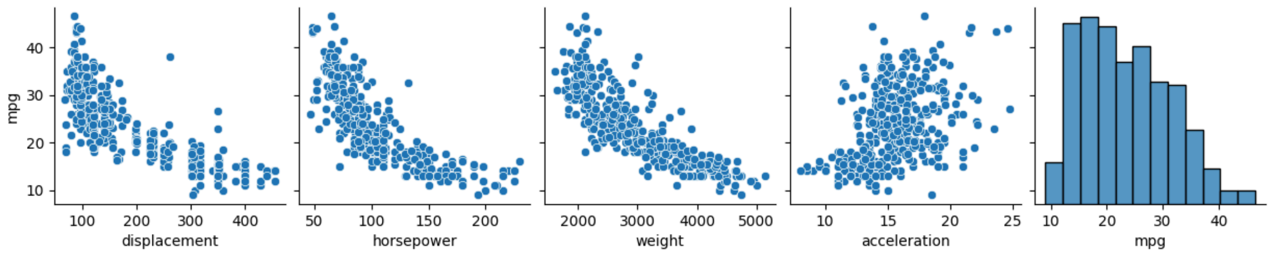
```
df = df.dropna()
```

```
df.info()
```

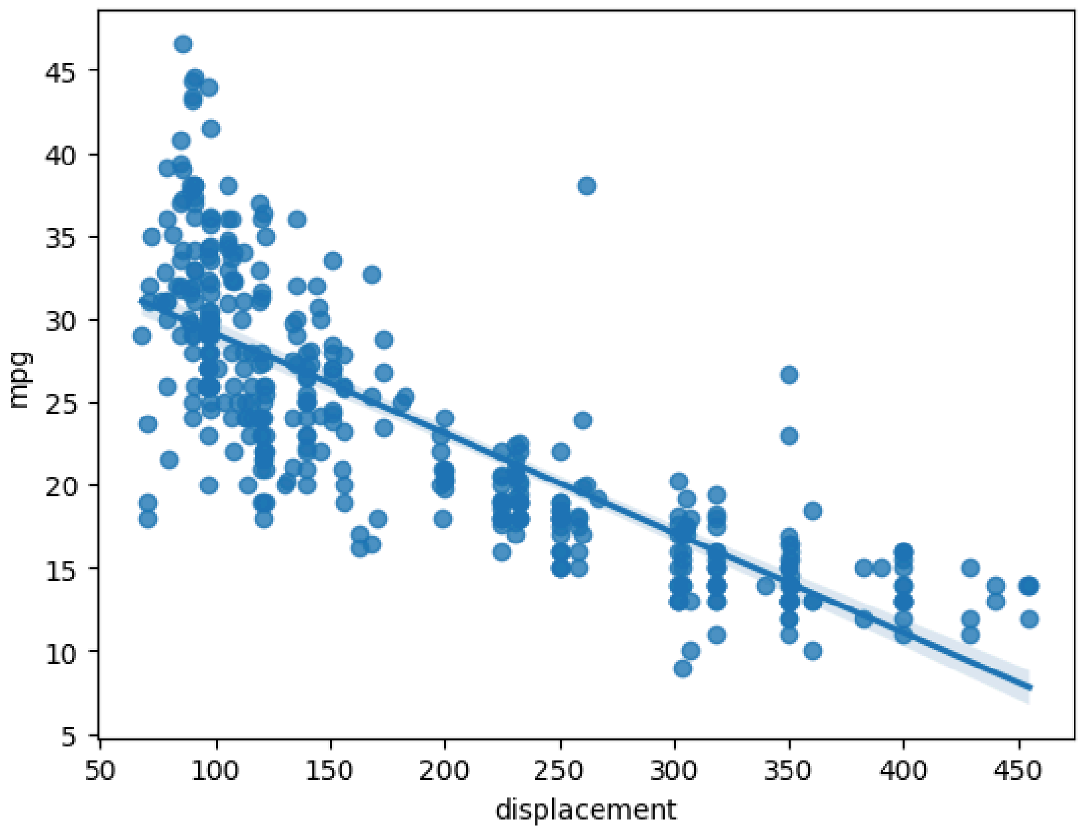
```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    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
```

Data Visualization

```
sns.pairplot(df, x_vars= ['displacement', 'horsepower', 'weight', 'acceleration', 'mpg'], y
```




```
sns.regplot(x = 'displacement', y ='mpg', data = df);
```



Define Target Variable y and Feature X

```
df.columns
```

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

```
y = df['mpg']
```

```
y.shape
```

```
(392,)
```

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

```
X.shape
```

```
(392, 4)
```

```
X
```

	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

Next steps: Generate code with X View recommended plots

Scaling Data

```
from sklearn.preprocessing import StandardScaler
```

```
ss = StandardScaler()
```

```
X = ss.fit_transform(X)
```

```
X
```

```
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]])
```

```
pd.DataFrame(X).describe()
```

	0	1	2	3
count	3.920000e+02	3.920000e+02	3.920000e+02	3.920000e+02
mean	-7.250436e-17	-1.812609e-16	-1.812609e-17	4.350262e-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

After Standardization Mean is Zero and Standard Deviation is One

Train Test Split Data

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, random_state =
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((274, 4), (118, 4), (274,), (118,))
```

Linear Regression Model

```
from sklearn.linear_model import LinearRegression
```

```
lr = LinearRegression()
```

```
lr.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lr.intercept_
```

```
23.485738559737584
```

Mileage = 23.4= 1.05Displacement- 1.68Horsepower- 4.10Weight- 0.115Acceleration + error

Predict Test Data

```
y_pred = lr.predict(X_test)
```

```
y_pred
```

```
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

```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, r2_score
```

```
mean_absolute_error(y_test, y_pred)
```

```
3.3286968643244106
```

```
mean_absolute_percentage_error(y_test, y_pred)
```

```
0.14713035779536746
```

Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
```

```
poly = PolynomialFeatures(degree=2, interaction_only = True, include_bias = False)
```

```
X_train2 = poly.fit_transform(X_train)
```

```
X_test2 = poly.fit_transform(X_test)
```

```
lr.fit(X_train2, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
lr.intercept_
```

```
21.27336450063766
```

```
lr.coef_
```

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

```
y_pred_poly = lr.predict(X_test2)
```

Model Accuracy

```
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error, r2_score
```

```
mean_absolute_error(y_test, y_pred_poly)
```

```
2.7887147720295977
```

```
mean_absolute_percentage_error(y_test, y_pred_poly)
```

```
0.12074018342938687
```

```
r2_score(y_test, y_pred_poly)
```