

# Mileage Prediction

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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

Out[2]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet 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 sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

```
In [4]: d.nunique()
```

```
Out[4]: mpg          129
cylinders      5
displacement   82
horsepower     93
weight         351
acceleration   95
model_year     13
origin          3
name            305
dtype: int64
```

# Data processing

In [5]: d.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]: d.describe()

Out[6]:

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

In [7]: d.corr()

Out[7]:

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

## Missing Values

```
In [16]: d=d.dropna()
```

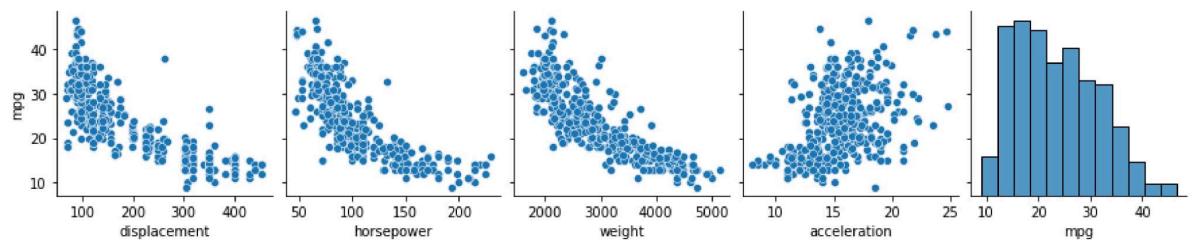
```
In [17]: d.info()
```

```
<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
```

## Data Visualization

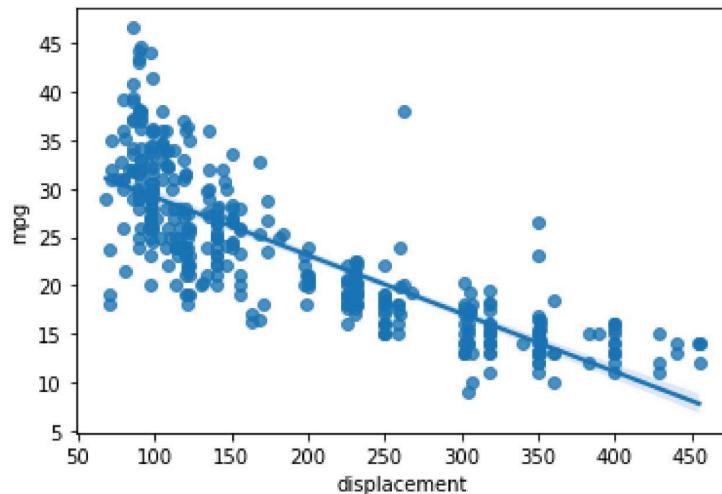
```
In [100]: sns.pairplot(d,x_vars=['displacement','horsepower','weight','acceleration','mpg'])
```

```
Out[100]: <seaborn.axisgrid.PairGrid at 0x141e5a92520>
```



```
In [30]: sns.regplot(x='displacement',y='mpg',data=d)
```

```
Out[30]: <AxesSubplot:xlabel='displacement', ylabel='mpg'>
```



## Define Target variable y and x

```
In [48]: d.columns
```

```
Out[48]: Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
   'acceleration', 'model_year', 'origin', 'name'],
  dtype='object')
```

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

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

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

```
In [52]: x=d[['displacement','horsepower','weight','acceleration']]
x.shape
```

```
Out[52]: (392, 4)
```

## Scaling Data

```
In [56]: from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x=ss.fit_transform(x)
x
```

```
Out[56]: 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 [103]: pd.DataFrame(x).describe()
```

```
Out[103]:
```

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

## TrainTest split data

```
In [105]: 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=25)
```

```
In [62]: x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
Out[62]: ((274, 4), (118, 4), (274,), (118,))
```

## Linear Regression Model

```
In [101]: from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

```
Out[101]: LinearRegression()
```

In [106]: lr.intercept\_

Out[106]: 23.6889216106858

In [83]: lr.coef\_

Out[83]: array([-0.13510042, -1.4297211 , -5.23891463, 0.22436094])

## Equation:

Mileage=23.4-0.13 Displacement-1.42 Horsepower-5.28 weight-0.224Acceleration+error

## Predict Test Data

In [79]: y\_pred=lr.predict(x\_test)

In [86]: y\_pred

Out[86]: array([25.24954801, 26.85525431, 26.58882904, 29.48052754, 23.91216916, 14.9529791 , 30.0607685 , 34.07634195, 30.550342 , 11.31024173, 18.14067535, 18.75305197, 29.80678264, 33.19954312, 17.23635872, 16.06983768, 25.94812038, 21.15777548, 29.92508087, 25.05587641, 22.85575427, 30.96630956, 22.82202336, 24.04513247, 25.95102384, 26.21136844, 14.91805111, 31.85928917, 21.95227216, 26.85446824, 8.94214825, 26.21244694, 30.20552304, 7.15733458, 26.31771126, 30.54356872, 14.13603243, 31.02810818, 33.19140036, 31.74995879, 11.07428823, 30.50398808, 29.36195486, 31.022648 , 23.53384962, 22.87821543, 11.03531446, 14.3757476 , 31.44484893, 26.64255441, 27.96470623, 21.80486111, 20.32272978, 31.27632871, 24.83127389, 19.13391479, 28.2786737 , 25.21468804, 26.89045676, 28.76603057, 19.03600671, 29.49310219, 28.42147856, 26.6112997 , 7.384747 , 20.13152225, 22.77931428, 20.50765035, 32.81875326, 27.92430623, 13.34341223, 8.03767139, 25.34229398, 17.23635872, 33.03710336, 31.07878627, 21.58700058, 24.53266643, 30.38829664, 17.84737111, 31.30622407, 30.1021144 , 22.81248978, 20.01904445, 9.12644754, 24.50457451, 29.57695629, 29.45235437, 31.59169567, 26.49442535, 30.32795983, 12.36145993, 16.48933189, 15.27329229, 32.77989962, 27.25863029, 11.07878871, 25.72147567, 12.57968624, 30.4363069 , 27.56306784, 24.92600083, 16.21791725, 23.89776551, 18.63499966, 10.21748386, 21.60970196, 23.01257072, 27.30850629, 30.45961552, 29.43254102, 27.21176721, 24.2365775 , 28.87030773, 21.16703179, 27.97152628, 24.54560958, 32.23487944])

## Model Accuracy

```
In [91]: from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,rmse
mean_absolute_error(y_test,y_pred)
```

```
Out[91]: 3.417654680078562
```

```
In [88]: mean_absolute_percentage_error(y_test,y_pred)
```

```
Out[88]: 0.16282215595698366
```

```
In [92]: r2_score(y_test,y_pred)
```

```
Out[92]: 0.6767436309121446
```

## Polynomial Regression

```
In [107]: from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=2,interaction_only=True,include_bias=False)
```

```
In [108]: x_train2=poly.fit_transform(x_train)
```

```
In [109]: x_test2=poly.fit_transform(x_test)
```

```
In [110]: lr.fit(x_train2,y_train)
```

```
Out[110]: LinearRegression()
```

```
In [111]: lr.intercept_
```

```
Out[111]: 21.45712035519168
```

```
In [112]: lr.coef_
```

```
Out[112]: array([-1.97594907e+00, -5.50639326e+00, -1.82341405e+00, -8.04049934e-01,
       1.55534517e+00, -4.40583099e-01, -5.33735335e-01,  1.29466895e+00,
       2.61553723e-03,  5.86761939e-01])
```

```
In [114]: y_pred_poly=lr.predict(x_test2)
```

## Model Accuracy

```
In [115]: from sklearn.metrics import mean_absolute_error,mean_absolute_percentage_error,rmse
mean_absolute_error(y_test,y_pred_poly)
```

```
Out[115]: 2.924007242447458
```

```
In [116]: mean_absolute_percentage_error(y_test,y_pred_poly)
```

```
Out[116]: 0.12874881331071994
```

```
In [117]: r2_score(y_test,y_pred_poly)
```

```
Out[117]: 0.7198303534964863
```

## Hand Written Digit Classification

```
In [118]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt
```

### importing data from sklearn library

```
In [119]: from sklearn.datasets import load_digits
```

```
In [120]: df=load_digits()
```

```
In [126]: _,axes=plt.subplots(nrows=1,ncols=4,figsize=(10,3))  
for ax,image,label in zip(axes,df.images,df.target):  
    ax.set_axis_off()  
    ax.imshow(image,cmap=plt.cm.gray_r,interpolation='nearest')  
    ax.set_title("Training: %i" % label)
```



### Flatten image

```
In [128]: df.images.shape
```

```
Out[128]: (1797, 8, 8)
```

```
In [131]: df.images[0]
```

```
Out[131]: array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
   [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
   [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
   [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
   [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
   [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
   [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
   [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

```
In [132]: df.images[0].shape
```

```
Out[132]: (8, 8)
```

```
In [133]: n_samples=len(df.images)
data=df.images.reshape((n_samples,-1))
```

```
In [134]: data[0]
```

```
Out[134]: array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0., 13., 15., 10.,
   15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  0.,  4.,
   12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
   0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
   10., 12.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
In [135]: data[0].shape
```

```
Out[135]: (64,)
```

```
In [136]: data.shape
```

```
Out[136]: (1797, 64)
```

## Scaling data

```
In [137]: data.min()
```

```
Out[137]: 0.0
```

```
In [138]: data.max()
```

```
Out[138]: 16.0
```

```
In [139]: data=data/16
```

```
In [140]: data.shape
```

```
Out[140]: (1797, 64)
```

## Train Test split Data

```
In [141]: from sklearn.model_selection import train_test_split  
x_train,x_test,y_train,y_test=train_test_split(data,df.target,train_size=0.7,rand
```

```
In [142]: x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
Out[142]: ((1257, 64), (540, 64), (1257,), (540,))
```

## Random forest model

```
In [148]: from sklearn.ensemble import RandomForestClassifier  
rf=RandomForestClassifier()
```

```
In [149]: rf.fit(x_train,y_train)
```

```
Out[149]: RandomForestClassifier()
```

## Predict test data

```
In [150]: y_pred=rf.predict(x_test)  
y_pred
```

```
Out[150]: array([9, 5, 9, 6, 9, 8, 3, 1, 4, 0, 1, 2, 9, 1, 7, 1, 5, 6, 8, 6, 7, 5,  
5, 2, 1, 4, 0, 5, 7, 8, 4, 2, 1, 2, 0, 4, 2, 1, 9, 9, 2, 7, 5, 3,  
9, 2, 4, 7, 8, 4, 0, 8, 7, 7, 8, 7, 8, 0, 7, 3, 3, 1, 9, 2, 4, 0,  
5, 2, 5, 1, 4, 1, 4, 9, 7, 3, 3, 6, 9, 6, 9, 7, 9, 2, 7, 2, 1, 5,  
2, 4, 1, 3, 4, 9, 7, 1, 8, 1, 3, 1, 5, 2, 5, 8, 3, 8, 2, 7, 1, 1,  
1, 1, 6, 2, 2, 3, 4, 1, 4, 6, 8, 5, 4, 0, 0, 2, 7, 0, 3, 2, 1, 6,  
9, 3, 5, 9, 6, 7, 1, 3, 8, 6, 7, 5, 8, 3, 9, 0, 1, 9, 7, 3, 2, 6,  
7, 4, 8, 1, 4, 0, 6, 3, 6, 6, 5, 5, 2, 8, 9, 4, 9, 5, 2, 1, 6, 6,  
7, 1, 0, 3, 0, 2, 8, 5, 3, 3, 5, 2, 1, 8, 4, 3, 5, 2, 7, 2, 9, 9,  
2, 4, 6, 0, 1, 6, 1, 5, 4, 4, 5, 7, 9, 8, 3, 9, 3, 5, 0, 2, 6, 4,  
1, 3, 1, 9, 6, 5, 7, 5, 1, 6, 4, 2, 1, 7, 2, 0, 2, 1, 9, 5, 7, 3,  
0, 3, 9, 9, 4, 0, 1, 2, 4, 5, 1, 0, 7, 5, 3, 7, 8, 4, 8, 3, 0, 3,  
8, 9, 3, 6, 3, 1, 0, 3, 8, 2, 2, 6, 5, 5, 6, 6, 5, 7, 0, 9, 8, 8,  
4, 0, 7, 6, 1, 8, 0, 1, 7, 0, 2, 4, 7, 8, 6, 8, 3, 2, 4, 5, 6, 0,  
7, 6, 1, 5, 5, 3, 4, 7, 0, 7, 0, 8, 3, 8, 2, 9, 8, 5, 2, 1, 2, 5,  
6, 5, 9, 1, 3, 8, 1, 7, 8, 1, 7, 8, 0, 3, 7, 5, 5, 4, 7, 3, 8, 6,  
9, 5, 7, 7, 5, 6, 5, 3, 1, 2, 7, 4, 4, 6, 5, 5, 9, 5, 2, 7, 2, 5,  
1, 9, 7, 9, 6, 8, 0, 4, 6, 9, 9, 6, 8, 2, 7, 6, 7, 4, 7, 5, 5, 6,  
5, 2, 3, 3, 0, 3, 4, 9, 3, 4, 3, 9, 2, 4, 2, 0, 5, 0, 1, 6, 8, 4,  
4, 2, 9, 0, 8, 4, 6, 9, 0, 9, 6, 0, 5, 1, 5, 5, 2, 5, 7, 9, 1, 3,  
6, 5, 8, 4, 3, 7, 5, 8, 1, 8, 2, 3, 0, 0, 0, 3, 1, 9, 0, 0, 2, 4,  
5, 0, 4, 0, 6, 0, 1, 9, 4, 8, 3, 8, 5, 9, 0, 5, 0, 5, 0, 6, 3, 5,  
2, 9, 7, 4, 6, 5, 0, 9, 2, 0, 8, 2, 9, 0, 0, 1, 0, 4, 3, 3, 9, 8,  
1, 1, 6, 3, 8, 8, 5, 9, 5, 4, 1, 7, 1, 3, 4, 5, 1, 4, 3, 2, 9, 7,  
4, 9, 0, 2, 3, 8, 1, 5, 6, 3, 7, 1])
```

```
In [151]: from sklearn.metrics import confusion_matrix,classification_report
```

```
In [152]: confusion_matrix(y_test,y_pred)
```

```
Out[152]: array([[52,  0,  0,  0,  0,  0,  0,  0,  0,  0],  
[ 0, 58,  0,  0,  0,  0,  0,  0,  0,  0],  
[ 1,  1, 54,  0,  0,  0,  0,  0,  0,  0],  
[ 0,  0,  0, 52,  0,  0,  0,  0,  0,  0],  
[ 0,  0,  0,  0, 51,  0,  0,  1,  1,  0],  
[ 0,  0,  0,  1,  0, 65,  1,  0,  0,  3],  
[ 0,  0,  0,  0,  1, 46,  0,  1,  0,  0],  
[ 0,  0,  0,  0,  0,  0, 50,  0,  1,  0],  
[ 0,  1,  0,  1,  0,  0,  0,  0, 46,  0],  
[ 0,  0,  0,  2,  0,  0,  0,  1,  0, 49]], dtype=int64)
```

```
In [153]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.98	1.00	0.99	52
1	0.97	1.00	0.98	58
2	1.00	0.96	0.98	56
3	0.93	1.00	0.96	52
4	1.00	0.96	0.98	53
5	0.98	0.93	0.96	70
6	0.98	0.96	0.97	48
7	0.96	0.98	0.97	51
8	0.96	0.96	0.96	48
9	0.92	0.94	0.93	52
accuracy			0.97	540
macro avg	0.97	0.97	0.97	540
weighted avg	0.97	0.97	0.97	540

```
In [ ]:
```