Bike Price Prediction using Linear Regression

Use Linear Regression(Ordinary Least Square) to Predict Bike Price
This tutorial explains the necessary steps in coding. TO get a correct solution viewers must realise that OLS require the fullfilment of assumptions for effective modeling.To learn Regression technique join our free courses at ybifoundation.org

Get Understanding about Dataset

There are 8 variables in the dataset
1.Brand-manufacturing company
2.Model-model of bike
3.Selling_price-selling price of bike
4.Year-year of manufacturing
5.Seller_Type-type of seller
6.Owner-owner type
7.KM_Driven-total km driven
8.Ex_Showroom_Price-ex-showroom price

Import Library

In [1]:

```
import pandas as pd
import numpy as np
```

Import CSV as DataFrame

In [2]:

```
df = pd.read_csv("Dataset-main/Dataset-main/Bike Prices.csv")
```

Get the First Five Rows of Dataframe

In [3]:

```
df.head()
```

Out[3]:

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Price
0	TVS	TVS XL 100	30000	2017	Individual	1st owner	8000	30490.0
1	Bajaj	Bajaj ct 100	18000	2017	Individual	1st owner	35000	32000.0
2	Yo	Yo Style	20000	2011	Individual	1st owner	10000	37675.0
3	Bajaj	Bajaj Discover 100	25000	2010	Individual	1st owner	43000	42859.0
4	Bajaj	Bajaj Discover 100	24999	2012	Individual	2nd owner	35000	42859.0

Get Information of DataFrame

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1061 entries, 0 to 1060
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Brand              1061 non-null   object 
 1   Model              1061 non-null   object 
 2   Selling_Price      1061 non-null   int64  
 3   Year               1061 non-null   int64  
 4   Seller_Type        1061 non-null   object 
 5   Owner              1061 non-null   object 
 6   KM_Driven          1061 non-null   int64  
 7   Ex_Showroom_Price  626 non-null    float64
dtypes: float64(1), int64(3), object(4)
memory usage: 66.4+ KB
```

Get Missing Values Drop

In [5]:

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

Get the summary Statistics

In [6]:

```
df.describe()
```

Out[6]:

	Selling_Price	Year	KM_Driven	Ex_Showroom_Price
count	626.000000	626.000000	626.000000	6.260000e+02
mean	59445.164537	2014.800319	32671.576677	8.795871e+04
std	59904.350888	3.018885	45479.661039	7.749659e+04
min	6000.000000	2001.000000	380.000000	3.049000e+04
25%	30000.000000	2013.000000	13031.250000	5.485200e+04
50%	45000.000000	2015.000000	25000.000000	7.275250e+04
75%	65000.000000	2017.000000	40000.000000	8.703150e+04
max	760000.000000	2020.000000	585659.000000	1.278000e+06

Get Categories and Counts of Categorical Variables

In [7]:

```
df[['Brand']].value_counts()
```

Out[7]:

```
Brand
Honda         170
Bajaj         143
Hero          108
Yamaha         94
Royal          40
TVS            23
Suzuki         18
KTM             6
Mahindra        6
Kawasaki        4
UM              3
Activa          3
Harley          2
Vespa           2
BMW             1
Hyosung         1
Benelli         1
Yo              1
dtype: int64
```

In [8]:

```
df[['Model']].value_counts()
```

Out[8]:

```
Model
Honda Activa [2000-2015]                    23
Honda CB Hornet 160R                        22
Bajaj Pulsar 180                            20
Yamaha FZ S V 2.0                           16
Bajaj Discover 125                          16
                                            ..
Royal Enfield Thunderbird 500                1
Royal Enfield Continental GT [2013 - 2018]   1
Royal Enfield Classic Stealth Black          1
Royal Enfield Classic Squadron Blue          1
Yo Style                                     1
Length: 183, dtype: int64
```

In [9]:

```
df[['Seller_Type']].value_counts()
```

Out[9]:

```
Seller_Type
Individual     623
Dealer           3
dtype: int64
```

In [10]:

```
df[['Owner']].value_counts()
```

Out[10]:

```
Owner
1st owner    556
2nd owner     66
3rd owner      3
4th owner      1
dtype: int64
```

Get Column Names

In [11]:

```
df.columns
```

Out[11]:

```
Index(['Brand', 'Model', 'Selling_Price', 'Year', 'Seller_Type', 'Owner',
       'KM_Driven', 'Ex_Showroom_Price'],
      dtype='object')
```

Get Shape of Dataframe

In [12]:

```
df.shape
```

Out[12]:

```
(626, 8)
```

Get Encoding of Categorical Features

In [13]:

```
df.replace({'Seller_Type':{'Individual':0,'Dealer':1}},inplace=True)
```

In [14]:

```
f.replace({'Owner':{'1st owner':0,'2nd owner':1,'3rd owner':2,'4th owner':3}},inplace=True)
```

In [16]:

```
#X= pd.get_dummies(X,columns=['Seller_Type','Owner'],drop_first = True)
```

Its always recommended to use dummy variables in case of categorical features

Define y(dependent or label or target variable) and X(independent or features or attribute Variable

In [18]:

```
y = df['Selling_Price']
```

In [19]:

```
y.shape
```

Out[19]:

```
(626,)
```

In [20]:

```
y
```

Out[20]:

```
0       30000
1       18000
2       20000
3       25000
4       24999
        ...
621    330000
622    300000
623    425000
624    760000
625    750000
Name: Selling_Price, Length: 626, dtype: int64
```

In [21]:

```
X = df[['Year','Seller_Type','Owner','KM_Driven','Ex_Showroom_Price']]
```

or use drop function to define X

In [24]:

```
X = df.drop(['Brand','Model','Selling_Price'],axis=1)
```

In [25]:

```
X.shape
```

Out[25]:

```
(626, 5)
```

In [26]:

```
X
```

Out[26]:

	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Price
0	2017	0	0	8000	30490.0
1	2017	0	0	35000	32000.0
2	2011	0	0	10000	37675.0
3	2010	0	0	43000	42859.0
4	2012	0	1	35000	42859.0
...	...	...	...	...	...
621	2014	0	3	6500	534000.0
622	2011	0	0	12000	589000.0
623	2017	0	1	13600	599000.0
624	2019	0	0	2800	752020.0
625	2013	0	1	12000	1278000.0

626 rows × 5 columns

Get Train Test Split

In [27]:

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

In [28]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, random_state=2529)
```

In [29]:

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

Out[29]:

```
((438, 5), (188, 5), (438,), (188,))
```

Get Model Train

In [30]:

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

In [31]:

```
lr = LinearRegression()
```

In [32]:

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

Out[32]:

```
LinearRegression()
```

Get Model Prediction

In [33]:

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

In [34]:

```
y_pred.shape
```

Out[34]:

```
(188,)
```

In [35]:

```
y_pred
```

Out[35]:

```
array([ 27210.52271468,  56340.08335159,  63471.94671999,  53627.63844789,
        55612.75744271,  53888.9225972 ,  33751.35275101,  60311.4950189 ,
       113713.05684462,  76639.49332945,  27826.73993814,  49919.83255836,
        65886.64311458,  26755.12664068,  48277.75426041, 127646.56079321,
        70047.10661639,  39350.67963648,  36081.03597879,  45360.79436331,
        48079.89470578,  44803.02464799,  55161.44026112,  71041.51821317,
        91689.22699147,  49301.53594656,  55988.19326247, 108171.54600292,
        32771.06897895,  25468.2007301 ,  17128.61806156, 179271.4113071 ,
        45698.99857629,  31371.0928507 ,  67886.52106728,  41492.49575816,
        56855.22238604,  47820.47003474,  74682.14053958,  24984.21822742,
        55374.00513697,  41412.36775226,  67991.60287765,  26553.59421844,
        89788.69870685,  45764.83633685, 133888.03770374, 106988.11382486,
        71176.4066771 ,  25332.25485949,  79512.43778826,  63914.3808817 ,
        28632.12110986,  53656.13623938,  -5396.37132925,  70377.44571174,
        33313.03576473,  53994.92478404,  67509.8583636 ,  59735.05378851,
        22199.83644222,  15374.1898417 ,  44510.76819417,  30279.52476748,
       108243.77037514,  19291.88958741,  53614.31297603,  59230.23269131,
        60174.21081097,  45924.63468742,  25770.81883496,  63471.36257821,
       242123.45729766,  61387.7254455 ,  56510.98127077,  48123.28087215,
        51668.27442015,  90279.76190494,  14827.76533551, 112437.70820498,
        35066.88027407,  30902.4106917 ,  31441.48921438, 125593.75847151,
        27705.38813165, -11590.29205559,  15582.17108687,  75113.64511233,
       504085.44522221, 123545.42050108,  74770.89327697,  50747.47663244,
        44174.36182128,  25426.71561064,  30298.30524623,  47625.67836424,
        27850.37544807,  28845.23330931,  31580.38624691,  32309.63375628,
        47979.16788551,  65955.46375945,  13432.2821802 ,  15368.80064979,
        31973.23052407, 110353.9287054 ,  68181.49509144,  23143.49139796,
        53194.65732077,  34603.36376982,  56002.50967874,  62432.66994305,
       391470.77533145,   3558.29480881,  36019.18494308,  70876.34866549,
        72890.00667025, 137596.01384355,  27620.36308881, 135789.30486844,
        39674.40366796,  58367.09244534,  42401.2120262 ,  61864.43795671,
        42688.89652847,  63710.34571015,  10604.39360068,  38458.82820944,
       112251.84744213, 115403.00577519,  13658.41734785,  36196.83359584,
        54146.22998935,  97297.85724846,  55029.68137261,  22923.26533439,
       104569.9702967 ,  41965.7585202 ,  38759.68546474,  28930.61369013,
        45231.66612557,  48475.43422789,  26739.72257317,  53598.65972198,
        32558.54954523,  32212.22834946,  68172.98738422,  71839.47716452,
        32003.4669222 ,  40652.69995976,  39935.92211844,  63444.41846205,
        44545.58187712, 120873.38389606,  60926.58683176,  62641.82167492,
        60816.47379998,  27098.95433575,  26803.64749622,  48956.0046863 ,
        62032.88118716,  26471.97495717, 104937.23068752, 132903.35788461,
        37469.20409424,  57579.12080158,  40371.00915734,  -7039.40662507,
        26485.4003007 ,  90782.4255412 ,  52153.21149323,  56453.74542453,
        80440.59426002,  31890.46870272,  49505.97985576,  24288.36959517,
        25540.47481574, 117708.26333951,  23399.66596745,  63678.40865459,
        70144.2937267 ,  33434.89010058,  60885.29444484,  58389.55370881,
        35118.70403478,  58729.45401959,  34627.95322468,  38583.46239729])
```

Get Model Evaluation

In [36]:

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

In [37]:

```
mean_squared_error(y_test,y_pred)
```

Out[37]:

554715615.5062364

In [38]:

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

Out[38]:

12225.73701041818

In [39]:

```
r2_score(y_test,y_pred)
```

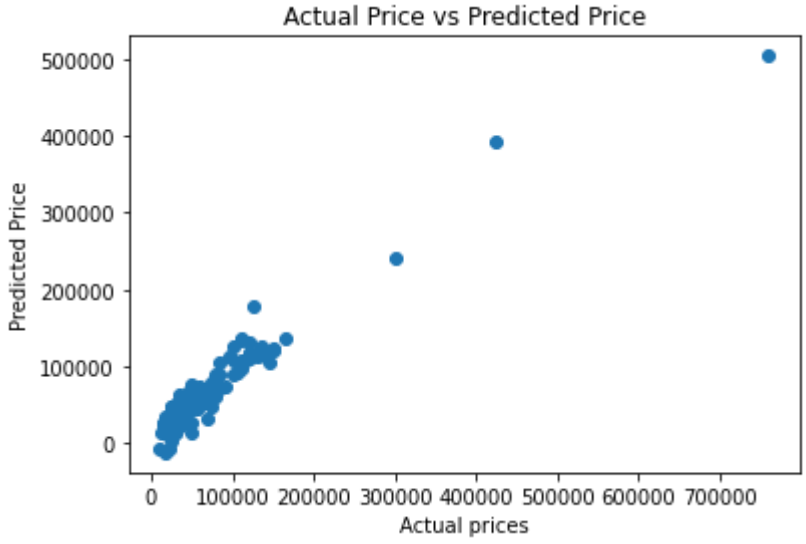
Out[39]:

0.8810414402980927

Get Visualization of Actual vs Predicted Results

In [40]:

```
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
plt.xlabel('Actual prices')
plt.ylabel('Predicted Price')
plt.title('Actual Price vs Predicted Price')
plt.show()
```



Get Future Predictions

Lets select a random from existing dataset as new value Steps to follow
1.Extract a random row using sample function
2.Separate X and y
3.Predict

In [42]:

```
df_new = df.sample(1)
```

In [43]:

```
df_new
```

Out[43]:

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Price
463	Hero	Hero Karizma 2014	30000	2014	0	1	51000	86744.0

In [44]:

```
df_new.shape
```

Out[44]:

```
(1, 8)
```

In [45]:

```
X_new = df_new.drop(['Brand','Model','Selling_Price'],axis=1)
```

In [46]:

```
y_pred_new = lr.predict(X_new)
```

In [47]:

```
y_pred_new
```

Out[47]:

```
array([49919.83255836])
```

In [ ]:

```

```