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 fillfillment of asssumptions for efffective modeling. To learns regresssion 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 manufacting
5. Seller_Types-type of seller
6. Owner-owner type
7. KM_Driven-total driven
8. Ex_Showroom_Price-ex-showroom price

Import Library

```
import pandas as pd
```

```
import numpy as np
```

Import CSV as Dataframe

```
df = pd.read_csv(r'http://github.com/YBI-Foundation/Dataset/raw/main/Bike%20Prices.csv')
```

```
# df =pd.read_csv(r'C:\Users\YBI Foundation\Deskstop\Car Price.csv')
```

```
# df = pd.read_csv(r'/content/Car Price.csv')
```

Get the first five rows of datafame

```
df.head()
```

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Pric
0	TVS	TVS XL 100	30000	2017	Individual	1st owner	8000	30490.(
1	Bajaj	Bajaj ct 100	18000	2017	Individual	1st owner	35000	32000.(
2	Yo	Yo Style	20000	2011	Individual	1st owner	10000	37675.(

Get Information of Dataframe

```
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

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

Get the Summmary Statistics

```
df.describe()
```

	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

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

```
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
```

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

```
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
```

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

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

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

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

Get Column Names

```
df.columns
```

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

Get Shape of Dataframe

```
df.shape
```

```
(626, 8)
```

Get Encoding of Categorical Features

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

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

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

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

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

```
y.shape
```

```
(626,)
```

```
y
```

```
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
```

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

```
#X = df.drop(['Brand','Mode1','Selling_Price'],axix=1)
```

```
X.shape
```

```
(626, 5)
```

```
X
```

	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

Get Train Test Split

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

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

Get Model Train

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

```
lr = LinearRegression()
```

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

```
LinearRegression()
```

Get Model Prediction

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

```
y_pred.shape
```

```
(188,)
```

```
y_pred
```

```
array([ 27210.52271465,  56340.08335163,  63471.94671996,  53627.63844785,
        55612.75744268,  53888.92259719,  33751.35275102,  60311.4950183 ,
       113713.05684467,  76639.49332954,  27826.7399381 ,  49919.83255841,
        65886.64311457,  26755.12664064,  48277.75426038, 127646.56079335,
        70047.10661635,  39350.67963653,  36081.03597878,  45360.79436339,
        48079.89470577,  44803.02464799,  55161.44026111,  71041.51821318,
        91689.22699159,  49301.53594645,  55988.19326252, 108171.54600296,
        32771.06897901,  25468.20072996,  17128.61806164, 179271.41130746,
        45698.99857622,  31371.09285079,  67886.52106737,  41492.49575815,
        56855.22238602,  47820.47003468,  74682.14053958,  24984.21822736,
        55374.00513699,  41412.36775222,  67991.60287764,  26553.59421844,
        89788.69870689,  45764.83633686, 133888.03770389, 106988.113825  ,
        71176.40667714,  25332.25485946,  79512.43778826,  63914.38088173,
        28632.12110986,  53656.13623937,  -5396.37132904,  70377.44571174,
        33313.03576476,  53994.92478411,  67509.85836352,  59735.05378847,
        22199.83644217,  15374.18984158,  44510.76819427,  30279.52476752,
       108243.77037514,  19291.8895874 ,  53614.312976  ,  59230.23269131,
        60174.2108109 ,  45924.63468736,  25770.81883496,  63471.36257814,
       242123.45729792,  61387.72544548,  56510.98127074,  48123.28087213,
        51668.27442011,  90279.76190495,  14827.76533556, 112437.70820504,
        35066.88027405,  30902.41069172,  31441.48921433, 125593.75847157,
        27705.38813164, -11590.29205553,  15582.17108685,  75113.64511232,
       504085.44522282, 123545.42050116,  74770.89327697,  50747.47663245,
        44174.3618212 ,  25426.7156106 ,  30298.3052462 ,  47625.67836414,
        27850.37544807,  28845.23330928,  31580.38624692,  32309.63375635,
        47979.16788554,  65955.46375944,  13432.28218017,  15368.80064986,
        31973.23052409, 110353.92870546,  68181.49509136,  23143.49139797,
        53194.65732076,  34603.36376989,  56002.50967868,  62432.66994305,
       391470.77533201,   3558.29480891,  36019.18494305,  70876.34866549,
        72890.00667025, 137596.01384364,  27620.36308877, 135789.30486854,
        39674.40366791,  58367.0924453 ,  42401.21202624,  61864.4379567 ,
        42688.89652842,  63710.34571021,  10604.39360071,  38458.82820943,
       112251.84744225, 115403.00577536,  13658.41734785,  36196.83359584,
        54146.22998932,  97297.85724851,  55029.68137265,  22923.26533437,
       104569.97029689,  41965.75852017,  38759.68546491,  28930.61369011,
        45231.66612551,  48475.43422775,  26739.7225731 ,  53598.65972203,
        32558.54954524,  32212.22834942,  68172.98738422,  71839.47716461,
        32003.46692215,  40652.69995971,  39935.92211843,  63444.41846202,
        44545.5818771 , 120873.38389616,  60926.58683174,  62641.82167496,
        60816.47379994,  27098.95433573,  26803.64749618,  48956.00468627,
        62032.88118713,  26471.97495723, 104937.23068766, 132903.3578847 ,
        37469.2040942 ,  57579.12080094,  40371.00915736,  -7039.40662503,
        26485.40030077,  90782.42554145,  52153.21149321,  56453.74542453,
        80440.59426003,  31890.46870273,  49505.97985573,  24288.36959514,
        25540.47481573, 117708.26333955,  23399.66596746,  63678.40865459,
        70144.29372668,  33434.89010059,  60885.29444481,  58389.55370878,
        35118.7040348 ,  58729.4540196 ,  34627.9532246 ,  38583.4623973 ])
```

Get Model Evaluation

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

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

    554715615.5043668

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

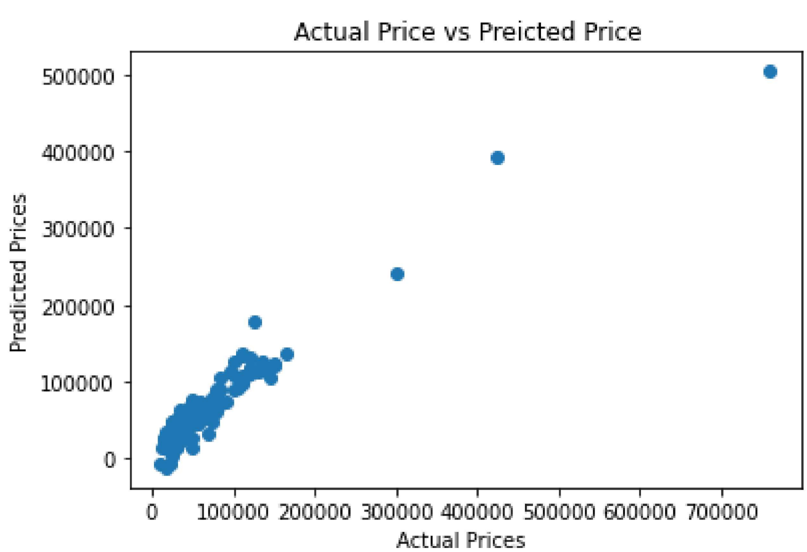
    12225.7370104107

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

    0.8810414402984937

Get Visualization of Actual Vs Predicted Results

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



Get Future Predictions

*Lets selects a random sample from existing dataset as new value *

Steps to follow

1. Extract a random row using sample function
2. Separate X and y
3. Predict

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

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

```
df_new.shape
```

```
(1, 8)
```

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

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

```
y_pred_new
```

```
array([165689.63963219])
```

0s completed at 10:19 AM