In [1]:
```
import pandas as pd
import numpy as np
```

In [6]:
```
d=pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/main/Bike%20Prices.c
```

In [7]:
```
d.head()
```

Out[7]:

	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

In [8]:
```
d.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
```

In [11]:
```
d=d.dropna()
```

In [12]:
```
d.describe()
```

Out[12]:

	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

Categories and Counts of Categorical data

In [13]:
```
d[['Brand']].value_counts()
```

Out[13]:
```
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 [15]:
```
d[['Model']].value_counts()
```

Out[15]:
```
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 [16]:
```
d[['Seller_Type']].value_counts()
```

Out[16]:
```
Seller_Type
Individual     623
Dealer           3
dtype: int64
```

In [17]:
```
d[['Owner']].value_counts()
```

Out[17]:
```
Owner
1st owner    556
2nd owner     66
3rd owner      3
4th owner      1
dtype: int64
```

Column names

In [18]:
```
d.columns
```

Out[18]:
```
Index(['Brand', 'Model', 'Selling_Price', 'Year', 'Seller_Type', 'Owner',
       'KM_Driven', 'Ex_Showroom_Price'],
      dtype='object')
```

In [19]:
```
d.shape
```

Out[19]:
```
(626, 8)
```

Encoding of Categorical Features

In [20]:
```
d.replace({'Seller_Type':{'Individual':0,'Dealer':1}},inplace=True)
```

In [49]:
```
d.replace({'Owner':{'1st owner':0,'2nd owner':1,'3rd owner':2,'4th owner':3}},inp
```

Define y and x variables

In [50]:
```
y=d['Selling_Price']
```

In [51]:
```
y.shape
```

Out[51]:
```
(626,)
```

In [52]:
```
y
```

Out[52]:
```
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 [53]:
```
x=d[['Year','Seller_Type','Owner','KM_Driven','Ex_Showroom_Price']]
```

In [54]:
```
x.shape
```

Out[54]:
```
(626, 5)
```

In [55]:
```
x
```

Out[55]:

	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

Train Test Split

In [56]:
```
from sklearn.model_selection import  train_test_split
```

In [57]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=252
```

In [58]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[58]:
```
((438, 5), (188, 5), (438,), (188,))
```

Model Train

In [59]:
```
from sklearn.linear_model import LinearRegression
```

In [60]:
```
l=LinearRegression()
l.fit(x_train,y_train)
```

Out[60]:
```
LinearRegression
LinearRegression()
```

Model Prediction

In [61]:
```
y_pred=l.predict(x_test)
```

In [62]:
```
y_pred.shape
```

Out[62]:
```
(188,)
```

In [63]:
```
y_pred
```

Out[63]:
```
array([ 27210.52271468,  56340.08335161,  63471.94672   ,  53627.63844782,
        55612.75744266,  53888.92259714,  33751.35275099,  60311.49501804,
       113713.05684464,  76639.49332948,  27826.73993814,  49919.83255839,
        65886.64311457,  26755.12664072,  48277.75426035, 127646.56079333,
        70047.10661637,  39350.67963649,  36081.03597878,  45360.79436333,
        48079.89470575,  44803.02464793,  55161.44026111,  71041.51821316,
        91689.22699154,  49301.5359465 ,  55988.19326246, 108171.54600296,
        32771.06897893,  25468.20073016,  17128.61806161, 179271.41130738,
        45698.99857631,  31371.09285074,  67886.52106728,  41492.49575812,
        56855.222386  ,  47820.47003471,  74682.14053955,  24984.21822744,
        55374.00513695,  41412.36775223,  67991.60287763,  26553.59421842,
        89788.69870685,  45764.83633684, 133888.03770386, 106988.11382497,
        71176.40667709,  25332.25485949,  79512.43778821,  63914.3808817 ,
        28632.12110983,  53656.13623931,  -5396.3713292 ,  70377.44571171,
        33313.03576471,  53994.92478412,  67509.85836358,  59735.05378843,
        22199.83644225,  15374.18984171,  44510.76819417,  30279.52476746,
       108243.77037516,  19291.88958741,  53614.31297599,  59230.2326913 ,
        60174.21081092,  45924.6346874 ,  25770.81883493,  63471.36257819,
       242123.45729789,  61387.72544543,  56510.98127069,  48123.2808721 ,
        51668.27442013,  90279.76190495,  14827.76533557, 112437.70820506,
        35066.88027402,  30902.41069168,  31441.48921435, 125593.75847161,
        27705.38813163, -11590.29205553,  15582.17108689,  75113.64511229,
       504085.44522283, 123545.42050112,  74770.89327694,  50747.47663248,
        44174.36182124,  25426.71561062,  30298.30524619,  47625.6783642 ,
        27850.37544803,  28845.23330927,  31580.38624695,  32309.63375626,
        47979.1678855 ,  65955.46375943,  13432.28218019,  15368.80064981,
        31973.23052409, 110353.92870541,  68181.49509145,  23143.49139794,
        53194.65732075,  34603.36376978,  56002.50967868,  62432.66994303,
       391470.77533196,   3558.29480883,  36019.18494312,  70876.34866549,
        72890.00667021, 137596.0138436 ,  27620.36308877, 135789.30486851,
        39674.40366792,  58367.09244526,  42401.2120262 ,  61864.43795667,
        42688.8965284 ,  63710.34571016,  10604.39360065,  38458.8282094 ,
       112251.84744221, 115403.0057753 ,  13658.41734787,  36196.83359583,
        54146.22998932,  97297.85724847,  55029.68137259,  22923.26533438,
       104569.97029681,  41965.75852017,  38759.68546479,  28930.61369013,
        45231.66612559,  48475.43422793,  26739.72257315,  53598.65972197,
        32558.54954519,  32212.22834943,  68172.98738417,  71839.47716456,
        32003.4669222 ,  40652.69995973,  39935.9221184 ,  63444.41846206,
        44545.58187707, 120873.38389615,  60926.58683174,  62641.82167495,
        60816.47379996,  27098.95433577,  26803.64749626,  48956.00468626,
        62032.8811871 ,  26471.97495722, 104937.23068756, 132903.35788466,
        37469.2040942 ,  57579.12080065,  40371.00915738,  -7039.40662501,
        26485.40030071,  90782.42554133,  52153.21149323,  56453.74542448,
        80440.59426   ,  31890.46870269,  49505.97985574,  24288.36959518,
        25540.47481571, 117708.26333952,  23399.66596747,  63678.40865459,
        70144.29372666,  33434.89010055,  60885.29444482,  58389.55370875,
        35118.70403476,  58729.45401958,  34627.9532246 ,  38583.46239725])
```

Model Evaluation

In [64]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

In [65]:
```
mean_squared_error(y_test,y_pred)
```

Out[65]: 554715615.5045587

In [66]:
```
mean_absolute_error(y_test,y_pred)
```

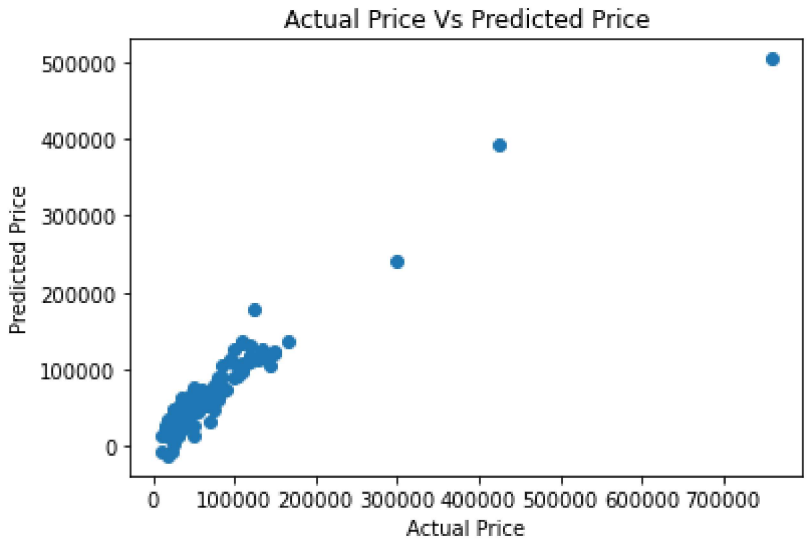
Out[66]: 12225.73701041481

In [67]:
```
r2_score(y_test,y_pred)
```

Out[67]: 0.8810414402984525

Visualisation of Actual and Predicted Results

In [68]:
```
import matplotlib.pyplot as p
p.scatter(y_test,y_pred)
p.xlabel('Actual Price')
p.ylabel('Predicted Price')
p.title("Actual Price Vs Predicted Price")
p.show()
```



Get Future Predictions

In [69]:
```
d_new=d.sample(1)
```

In [70]:
```
d_new
```

Out[70]:

	Brand	Model	Selling_Price	Year	Seller_Type	Owner	KM_Driven	Ex_Showroom_Price
38	Hero	Hero Maestro	25000	2015	0	1	40000	49412.0

In [71]:
```
x_new=d_new.drop(['Brand','Model','Selling_Price'],axis=1)
```

In [72]:
```
y_pred_new=l.predict(x_new)
```

In [73]:
```
y_pred_new
```

Out[73]:
```
array([30902.41069168])
```

In [ ]:
```

```