In [101]:
```
!pip3 install pandas    #intalling pandas into the system
```

```
Requirement already satisfied: pandas in /home/placemnet/anaconda3/lib/python3.10/site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /home/placemnet/anaconda3/lib/python3.10/site-pac
kages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /home/placemnet/anaconda3/lib/python3.10/site-packages (fro
m pandas) (2022.7)
Requirement already satisfied: numpy>=1.21.0 in /home/placemnet/anaconda3/lib/python3.10/site-packages (fr
om pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in /home/placemnet/anaconda3/lib/python3.10/site-packages (from py
thon-dateutil>=2.8.1->pandas) (1.16.0)
```

In [102]:
```
!pip install numpy
#'intalling the numpy'
```

```
Requirement already satisfied: numpy in /home/placemnet/anaconda3/lib/python3.10/site-packages (1.23.5)
```

In [103]:
```
import pandas as y   #importing pandas
```

In [104]:
```
data=y.read_csv("/home/placemnet/Downloads/fiat500.csv")   #reading the file
data.describe()                                          #calculating the statistical details of data of datafi
```

Out[104]:

	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	769.500000	51.904421	1650.980494	53396.011704	1.123537	43.541361	11.563428	8576.003901
std	444.126671	3.988023	1289.522278	40046.830723	0.416423	2.133518	2.328190	1939.958641
min	1.000000	51.000000	366.000000	1232.000000	1.000000	36.855839	7.245400	2500.000000
25%	385.250000	51.000000	670.000000	20006.250000	1.000000	41.802990	9.505090	7122.500000
50%	769.500000	51.000000	1035.000000	39031.000000	1.000000	44.394096	11.869260	9000.000000
75%	1153.750000	51.000000	2616.000000	79667.750000	1.000000	45.467960	12.769040	10000.000000
max	1538.000000	77.000000	4658.000000	235000.000000	4.000000	46.795612	18.365520	11100.000000

In [105]:
```
data.head(100)     #to return the first n rows
```

Out[105]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.611560	8900
1	2	pop	51	1186	32500	1	45.666359	12.241890	8800
2	3	sport	74	4658	142228	1	45.503300	11.417840	4200
3	4	lounge	51	2739	160000	1	40.633171	17.634609	6000
4	5	pop	73	3074	106880	1	41.903221	12.495650	5700
...	...	...	...	...	...	...	...	...	...
95	96	sport	51	4292	165600	1	44.715408	11.308300	5950
96	97	pop	51	1066	28000	1	41.769051	12.662810	8500
97	98	sport	51	2009	86000	2	40.633171	17.634609	7800
98	99	lounge	51	456	18592	2	45.393600	10.482240	10900
99	100	pop	51	731	41558	2	45.571220	9.159140	8790

In [106]:
```
data.info()    #to print the information of the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1538 entries, 0 to 1537
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ID               1538 non-null   int64  
 1   model            1538 non-null   object 
 2   engine_power     1538 non-null   int64  
 3   age_in_days      1538 non-null   int64  
 4   km               1538 non-null   int64  
 5   previous_owners  1538 non-null   int64  
 6   lat              1538 non-null   float64
 7   lon              1538 non-null   float64
 8   price            1538 non-null   int64  
dtypes: float64(2), int64(6), object(1)
memory usage: 108.3+ KB
```

In [107]:
```
data.tail(8)    #to get last n no.of rows
```

Out[107]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
1530	1531	lounge	51	670	29000	1	45.764648	8.99450	10800
1531	1532	sport	73	4505	127000	1	45.528511	9.59323	4750
1532	1533	pop	51	1917	52008	1	45.548000	11.54947	9900
1533	1534	sport	51	3712	115280	1	45.069679	7.70492	5200
1534	1535	lounge	74	3835	112000	1	45.845692	8.66687	4600
1535	1536	pop	51	2223	60457	1	45.481541	9.41348	7500
1536	1537	lounge	51	2557	80750	1	45.000702	7.68227	5990
1537	1538	pop	51	1766	54276	1	40.323410	17.56827	7900

In [108]:
```
data.shape    #to know the no.of columns and rows
```

Out[108]:
```
(1538, 9)
```

In [109]:
```
data['ID'].unique()    # to find the unique values from a series
```

Out[109]:
```
array([   1,    2,    3, ..., 1536, 1537, 1538])
```

In [110]:
```
data["previous_owners"].unique()
```

Out[110]:
```
array([1, 2, 3, 4])
```

In [111]:
```
list(data.columns)    #to print columns in a list
```

Out[111]:
```
['ID',
 'model',
 'engine_power',
 'age_in_days',
 'km',
 'previous_owners',
 'lat',
 'lon',
 'price']
```

In [112]:
```
data.groupby(['previous_owners']).count()    #counts the number of not empty values for a particular row or
```

Out[112]:

previous_owners	ID	model	engine_power	age_in_days	km	lat	lon	price
1	1389	1389	1389	1389	1389	1389	1389	1389
2	117	117	117	117	117	117	117	117
3	23	23	23	23	23	23	23	23
4	9	9	9	9	9	9	9	9

In [113]:
```
data.groupby(['model']).count()
```

Out[113]:

model	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
lounge	1094	1094	1094	1094	1094	1094	1094	1094
pop	358	358	358	358	358	358	358	358
sport	86	86	86	86	86	86	86	86

In [114]:
```
data.groupby(['price']).count()
```

Out[114]:

price	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon
2500	1	1	1	1	1	1	1	1
2900	1	1	1	1	1	1	1	1
3390	1	1	1	1	1	1	1	1
3500	1	1	1	1	1	1	1	1
3600	1	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...
10990	9	9	9	9	9	9	9	9
10999	5	5	5	5	5	5	5	5
11000	13	13	13	13	13	13	13	13
11090	2	2	2	2	2	2	2	2
11100	1	1	1	1	1	1	1	1

222 rows × 8 columns

In [115]:
```
data1=data.drop('lon',axis=1)    #to delete the column
```

In [116]:
```
data1.head(5)
```

Out[116]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	price
0	1	lounge	51	882	25000	1	44.907242	8900
1	2	pop	51	1186	32500	1	45.666359	8800
2	3	sport	74	4658	142228	1	45.503300	4200
3	4	lounge	51	2739	160000	1	40.633171	6000
4	5	pop	73	3074	106880	1	41.903221	5700

In [117]:
```
data['ID'].sum()   #to add the data in the column
```

Out[117]:
```
1183491
```

In [118]:
```
data2=data.loc[(data.model=='lounge')|(data.model=='pop')]
data2
```

Out[118]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.611560	8900
1	2	pop	51	1186	32500	1	45.666359	12.241890	8800
3	4	lounge	51	2739	160000	1	40.633171	17.634609	6000
4	5	pop	73	3074	106880	1	41.903221	12.495650	5700
5	6	pop	74	3623	70225	1	45.000702	7.682270	7900
...	...	...	...	...	...	...	...	...	...
1532	1533	pop	51	1917	52008	1	45.548000	11.549470	9900
1534	1535	lounge	74	3835	112000	1	45.845692	8.666870	4600
1535	1536	pop	51	2223	60457	1	45.481541	9.413480	7500
1536	1537	lounge	51	2557	80750	1	45.000702	7.682270	5990
1537	1538	pop	51	1766	54276	1	40.323410	17.568270	7900

1452 rows × 9 columns

In [119]:
```
data2=data.loc[(data.km<=50000)]
data2
```

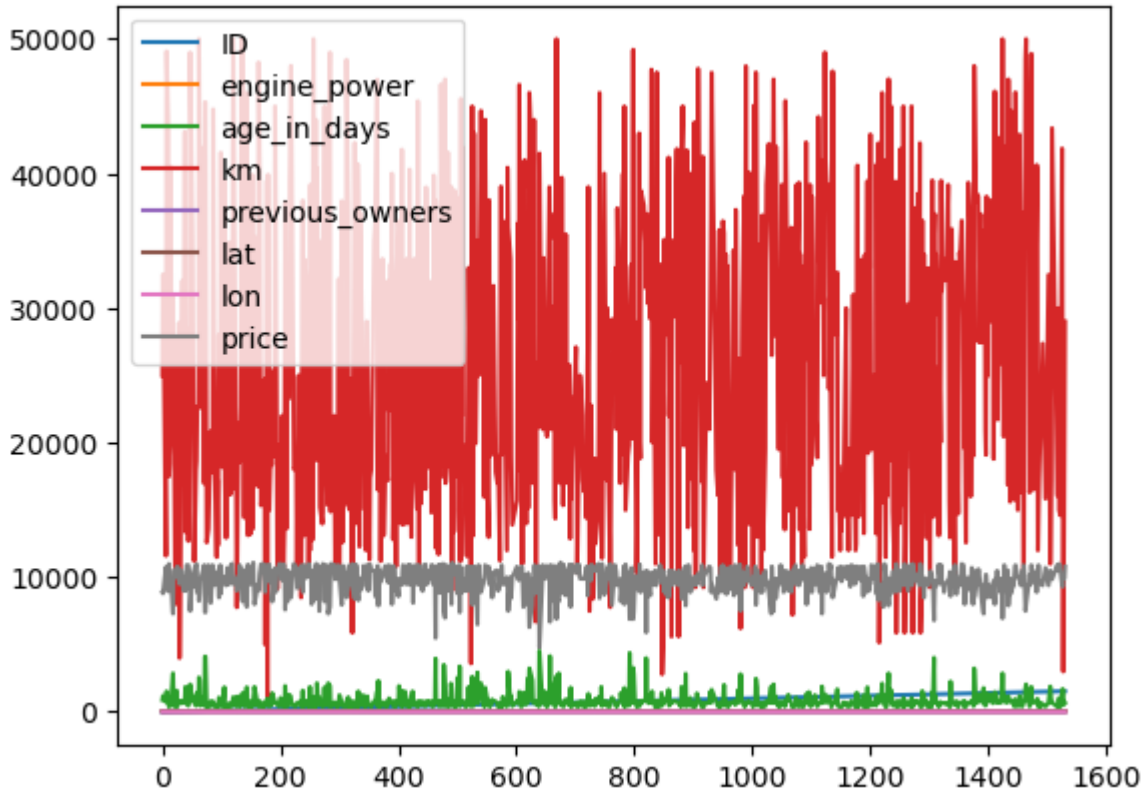
Out[119]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	lounge	51	882	25000	1	44.907242	8.61156	8900
1	2	pop	51	1186	32500	1	45.666359	12.24189	8800
6	7	lounge	51	731	11600	1	44.907242	8.61156	10750
7	8	lounge	51	1521	49076	1	41.903221	12.49565	9190
10	11	pop	51	790	43286	1	40.871429	14.43896	8950
...	...	...	...	...	...	...	...	...	...
1525	1526	lounge	51	790	41870	1	45.707249	11.47760	9500
1526	1527	lounge	51	1705	23600	1	38.122070	13.36112	9300
1527	1528	pop	51	517	3000	1	40.748241	14.52835	9999
1529	1530	lounge	51	731	22551	1	38.122070	13.36112	9900
1530	1531	lounge	51	670	29000	1	45.764648	8.99450	10800

907 rows × 9 columns

In [120]:
```
data2.groupby(['model']).count()
data2.plot()                              #create graph of the data
```

Out[120]: <Axes: >



CORRELATION

In [121]:
```
a=data.corr()
a
```

```
/tmp/ipykernel_6447/1834144657.py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  a=data.corr()
```

Out[121]:

	ID	engine_power	age_in_days	km	previous_owners	lat	lon	price
ID	1.000000	-0.034059	-0.060753	-0.006537	0.007803	-0.058207	0.058941	0.028516
engine_power	-0.034059	1.000000	0.319190	0.285495	-0.005030	0.005721	-0.005032	-0.277235
age_in_days	-0.060753	0.319190	1.000000	0.833890	0.075775	0.062982	-0.042667	-0.893328
km	-0.006537	0.285495	0.833890	1.000000	0.097539	0.035519	0.004839	-0.859373
previous_owners	0.007803	-0.005030	0.075775	0.097539	1.000000	0.001697	-0.026836	-0.076274
lat	-0.058207	0.005721	0.062982	0.035519	0.001697	1.000000	-0.766646	-0.011733
lon	0.058941	-0.005032	-0.042667	0.004839	-0.026836	-0.766646	1.000000	-0.003541
price	0.028516	-0.277235	-0.893328	-0.859373	-0.076274	-0.011733	-0.003541	1.000000

In [122]:
```
data2['model']=data['model'].map({'lounge':1,'pop':2,'sport':3})   #change the data in a column
data2
```

```
/tmp/ipykernel_6447/2086736257.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data2['model']=data['model'].map({'lounge':1,'pop':2,'sport':3})   #change the data in a column
```

Out[122]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
0	1	1	51	882	25000	1	44.907242	8.61156	8900
1	2	2	51	1186	32500	1	45.666359	12.24189	8800
6	7	1	51	731	11600	1	44.907242	8.61156	10750
7	8	1	51	1521	49076	1	41.903221	12.49565	9190
10	11	2	51	790	43286	1	40.871429	14.43896	8950
...	...	...	...	...	...	...	...	...	...
1525	1526	1	51	790	41870	1	45.707249	11.47760	9500
1526	1527	1	51	1705	23600	1	38.122070	13.36112	9300
1527	1528	2	51	517	3000	1	40.748241	14.52835	9999
1529	1530	1	51	731	22551	1	38.122070	13.36112	9900
1530	1531	1	51	670	29000	1	45.764648	8.99450	10800

907 rows × 9 columns

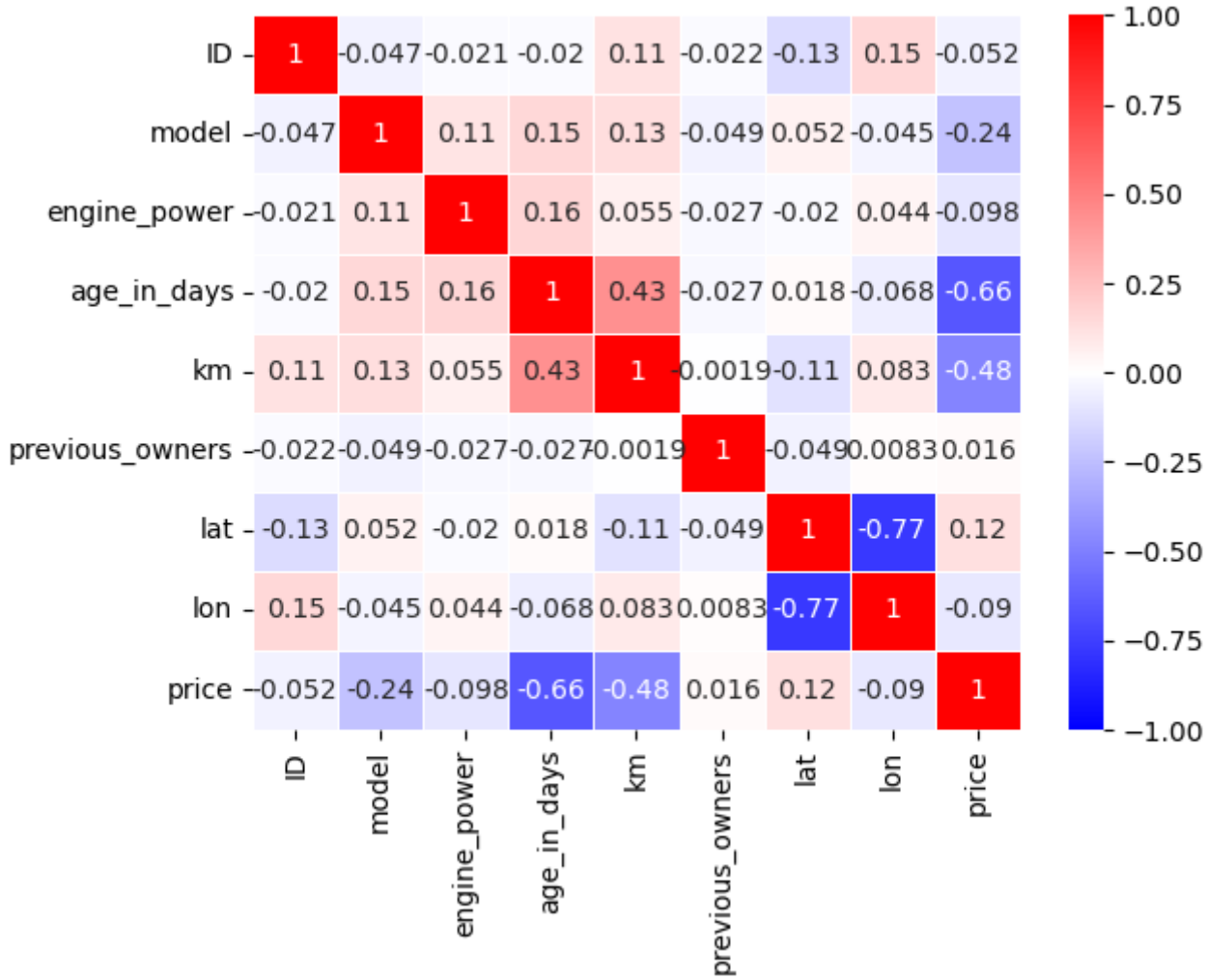
In [123]:
```
corr=data2.corr()    #correlation
corr
```

Out[123]:

	ID	model	engine_power	age_in_days	km	previous_owners	lat	lon	price
ID	1.000000	-0.047113	-0.021308	-0.019672	0.112097	-0.021821	-0.134745	0.153563	-0.051750
model	-0.047113	1.000000	0.107483	0.152722	0.126514	-0.049441	0.051640	-0.044861	-0.238326
engine_power	-0.021308	0.107483	1.000000	0.160405	0.055262	-0.026521	-0.019823	0.043889	-0.097790
age_in_days	-0.019672	0.152722	0.160405	1.000000	0.430566	-0.027217	0.017777	-0.067735	-0.656945
km	0.112097	0.126514	0.055262	0.430566	1.000000	-0.001910	-0.109633	0.083076	-0.479849
previous_owners	-0.021821	-0.049441	-0.026521	-0.027217	-0.001910	1.000000	-0.049327	0.008286	0.015958
lat	-0.134745	0.051640	-0.019823	0.017777	-0.109633	-0.049327	1.000000	-0.774363	0.120258
lon	0.153563	-0.044861	0.043889	-0.067735	0.083076	0.008286	-0.774363	1.000000	-0.090349
price	-0.051750	-0.238326	-0.097790	-0.656945	-0.479849	0.015958	0.120258	-0.090349	1.000000

In [124]:
```
import seaborn as sns                                                    #importing the seaborn
sns.heatmap(corr,vmax=1,vmin=-1,annot=True,linewidth=.5,cmap='bwr')      #heatmap
```

Out[124]: <Axes: >



In [125]:
```
d1=data.drop(['lat','lon','ID'],axis=1)
d1.describe()
```

Out[125]:

	engine_power	age_in_days	km	previous_owners	price
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	51.904421	1650.980494	53396.011704	1.123537	8576.003901
std	3.988023	1289.522278	40046.830723	0.416423	1939.958641
min	51.000000	366.000000	1232.000000	1.000000	2500.000000
25%	51.000000	670.000000	20006.250000	1.000000	7122.500000
50%	51.000000	1035.000000	39031.000000	1.000000	9000.000000
75%	51.000000	2616.000000	79667.750000	1.000000	10000.000000
max	77.000000	4658.000000	235000.000000	4.000000	11100.000000

In [126]:
```
d1=y.get_dummies(d1)
d1.describe()
```

Out[126]:

	engine_power	age_in_days	km	previous_owners	price	model_lounge	model_pop	model_sport
count	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000	1538.000000
mean	51.904421	1650.980494	53396.011704	1.123537	8576.003901	0.711313	0.232770	0.055917
std	3.988023	1289.522278	40046.830723	0.416423	1939.958641	0.453299	0.422734	0.229836
min	51.000000	366.000000	1232.000000	1.000000	2500.000000	0.000000	0.000000	0.000000
25%	51.000000	670.000000	20006.250000	1.000000	7122.500000	0.000000	0.000000	0.000000
50%	51.000000	1035.000000	39031.000000	1.000000	9000.000000	1.000000	0.000000	0.000000
75%	51.000000	2616.000000	79667.750000	1.000000	10000.000000	1.000000	0.000000	0.000000
max	77.000000	4658.000000	235000.000000	4.000000	11100.000000	1.000000	1.000000	1.000000

In [127]:
```
a=d1['price']
b=d1.drop('price',axis=1)
b
```

Out[127]:

	engine_power	age_in_days	km	previous_owners	model_lounge	model_pop	model_sport
0	51	882	25000	1	1	0	0
1	51	1186	32500	1	0	1	0
2	74	4658	142228	1	0	0	1
3	51	2739	160000	1	1	0	0
4	73	3074	106880	1	0	1	0
...	...	...	...	...	...	...	...
1533	51	3712	115280	1	0	0	1
1534	74	3835	112000	1	1	0	0
1535	51	2223	60457	1	0	1	0
1536	51	2557	80750	1	1	0	0
1537	51	1766	54276	1	0	1	0

1538 rows × 7 columns

In [128]:
```
#pip install -U scikit-learn
```

In [129]:
```
from sklearn.model_selection import train_test_split
a_train,a_test,b_train,b_test=train_test_split(a,b,test_size=0.33,random_state=42)
```

In [130]:
```
b_test.head()
```

Out[130]:

	engine_power	age_in_days	km	previous_owners	model_lounge	model_pop	model_sport
481	51	3197	120000	2	0	1	0
76	62	2101	103000	1	0	1	0
1502	51	670	32473	1	1	0	0
669	51	913	29000	1	1	0	0
1409	51	762	18800	1	1	0	0

In [131]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()   #creating object of linear regression
reg.fit(b_train,a_train)   #training and fitting LR object using training data
```

Out[131]:

▾ LinearRegression

LinearRegression()

In [132]:
```
apred=reg.predict(b_test)
apred
```

Out[132]:
```
array([ 5867.6503378 ,  7133.70142341,  9866.35776216,  9723.28874535,
       10039.59101162,  9654.07582608,  9673.14563045, 10118.70728123,
        9903.85952664,  9351.55828437, 10434.34963575,  7732.26255693,
        7698.67240131,  6565.95240435,  9662.90103518, 10373.20344286,
        9599.94844451,  7699.34400418,  4941.33017994, 10455.2719478 ,
       10370.51555682, 10391.60424404,  7529.06622456,  9952.37340054,
        7006.13845729,  9000.1780961 ,  4798.36770637,  6953.10376491,
        7810.39767825,  9623.80497535,  7333.52158317,  5229.18705519,
        5398.21541073,  5157.65652129,  8948.63632836,  5666.62365159,
        9822.1231461 ,  8258.46551788,  6279.2040404 ,  8457.38443276,
        9773.86444066,  6767.04074749,  9182.99904787, 10210.05195479,
        8694.90545226, 10328.43369248,  9069.05761443,  8866.7826029 ,
        7058.39787506,  9073.33877162,  9412.68162121, 10293.69451263,
       10072.49011135,  6748.5794244 ,  9785.95841801,  9354.09969973,
        9507.9444386 , 10443.01608254,  9795.31884316,  7197.84932877,
       10108.31707235,  7009.6597206 ,  9853.90699412,  7146.87414965,
        6417.69133992,  9996.97382441,  9781.18795953,  8515.83255277,
        8456.30006203,  6499.76668237,  7768.57829985,  6832.86406122,
        8347.96113362, 10439.02404036,  7356.43463051,  8562.56562053,
```

In [133]:
```
from sklearn.metrics import r2_score
r2_score(a_test,apred)     #to check the efficiency
```

Out[133]:
```
0.8415526986865394
```

In [134]:
```
from sklearn.metrics import mean_squared_error   #calculating MSE
mean_squared_error(apred,a_test)
```

Out[134]:
```
581887.727391353
```

In [135]:
```
print(mean_squared_error(apred,a_test)**(1/2))
```

```
762.8156575420782
```

In [136]:
```
results=y.DataFrame(columns=['Price','Predicted'])
results['Price']=a_test
results['Predicted']=apred
results.head()
#results=results.reset_index()
#results['Id']=results.index
```

Out[136]:

	Price	Predicted
481	7900	5867.650338
76	7900	7133.701423
1502	9400	9866.357762
669	8500	9723.288745
1409	9700	10039.591012

In [137]:
```
results['Difference']=results.apply(lambda row:row.Price-row.Predicted,axis=1)
results
```

Out[137]:

	Price	Predicted	Difference
481	7900	5867.650338	2032.349662
76	7900	7133.701423	766.298577
1502	9400	9866.357762	-466.357762
669	8500	9723.288745	-1223.288745
1409	9700	10039.591012	-339.591012
...	...	...	...
291	10900	10032.665135	867.334865
596	5699	6281.536277	-582.536277
1489	9500	9986.327508	-486.327508
1436	6990	8381.517020	-1391.517020
575	10900	10371.142553	528.857447

508 rows × 3 columns