Diamond price prediction Analysis

In [ ]:
```
Dataset-1
dataset link :https://www.kaggle.com/code/karnikakapoor/diamond-price-prediction/in
```

In [3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
data=pd.read_csv("diamonds.csv")
```

In [5]:
```
data
```

Out[5]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

In [6]:
```
df=data.copy()
df
```

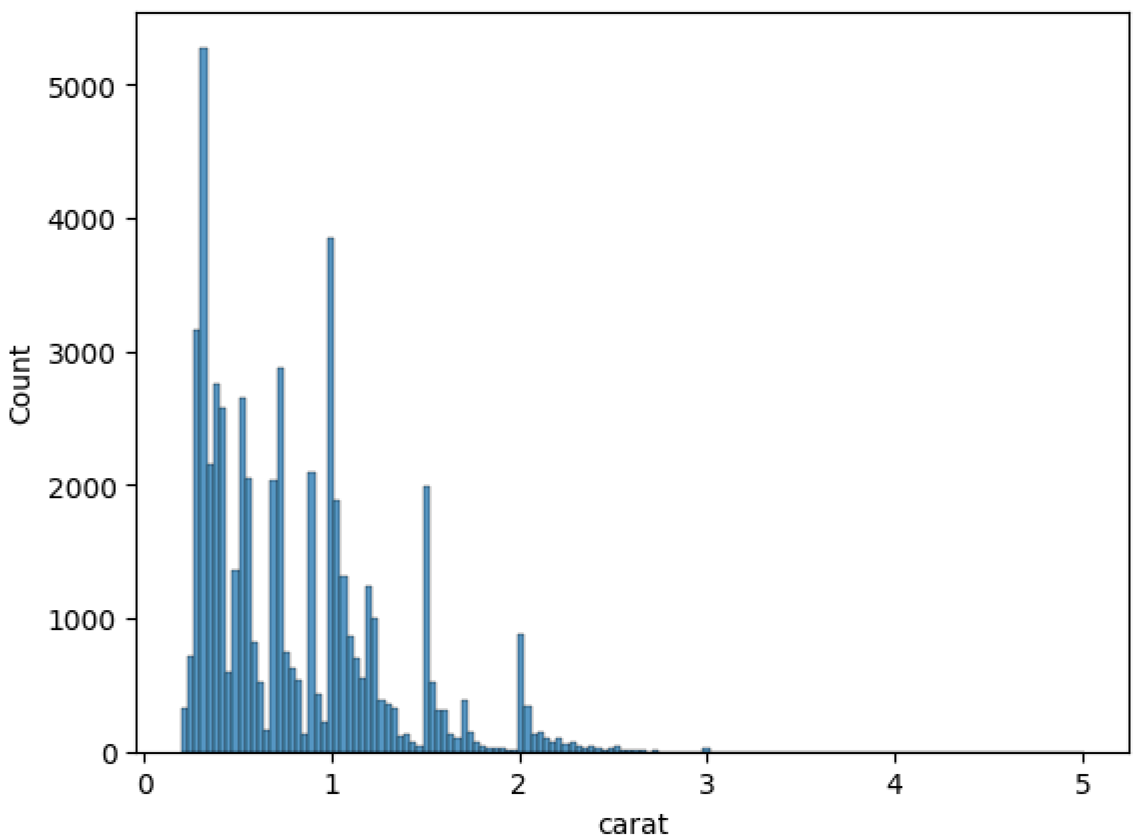
Out[6]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

In [7]:
```
sns.histplot(x=df.carat)
```

Out[7]: <Axes: xlabel='carat', ylabel='Count'>



In [8]:
```
q1=df["carat"].quantile(0.25)
q3=df["carat"].quantile(0.75)
iqr=q3-q1
iqr
lower=q1 - 1.5*iqr
upper=q3 + 1.5*iqr
upper
```

Out[8]: 2.0

In [9]:
```
# count=((df["carat"]<lower)|(df["carat"]>upper)).sum()
# per=(count/len(df["carat"]))*100
# per
num=df.select_dtypes(include="float64")
#num.drop("price", axis=1,inplace=True)
num
```

Out[9]:

	carat	depth	table	x	y	z
0	0.23	61.5	55.0	3.95	3.98	2.43
1	0.21	59.8	61.0	3.89	3.84	2.31
2	0.23	56.9	65.0	4.05	4.07	2.31
3	0.29	62.4	58.0	4.20	4.23	2.63
4	0.31	63.3	58.0	4.34	4.35	2.75
...	...	...	...	...	...	...
53935	0.72	60.8	57.0	5.75	5.76	3.50
53936	0.72	63.1	55.0	5.69	5.75	3.61
53937	0.70	62.8	60.0	5.66	5.68	3.56
53938	0.86	61.0	58.0	6.15	6.12	3.74
53939	0.75	62.2	55.0	5.83	5.87	3.64

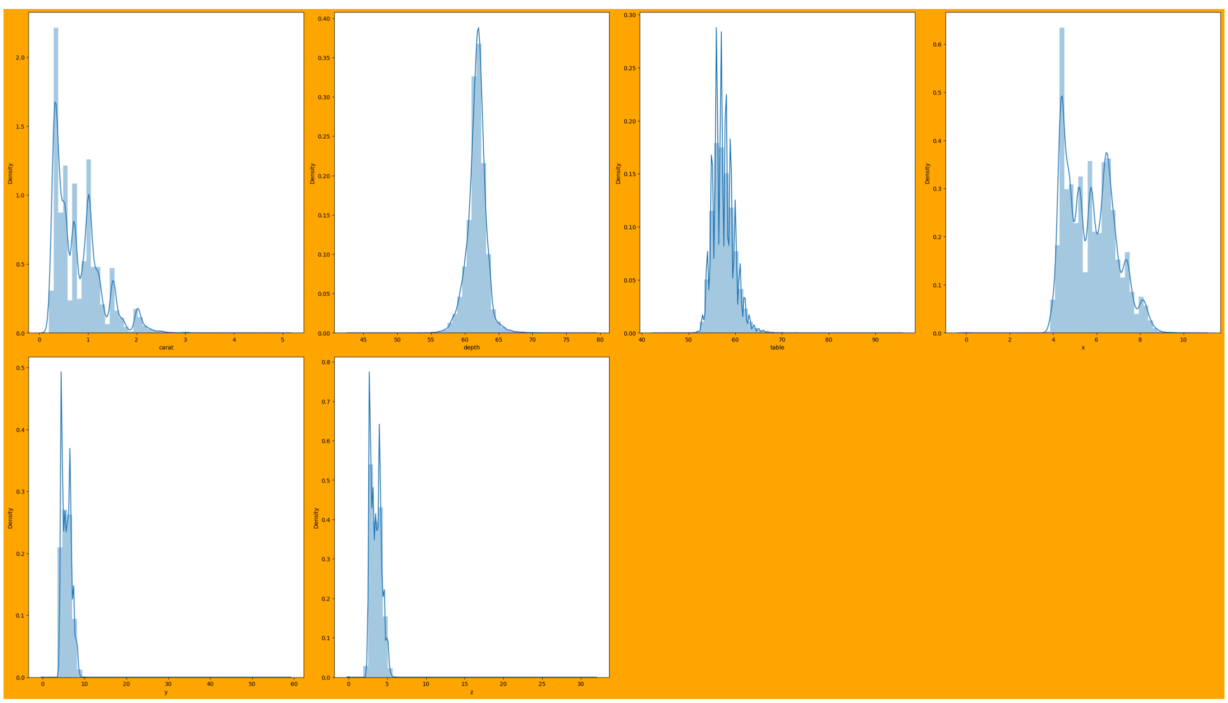
53940 rows × 6 columns

In [10]:
```
Q1=num.quantile(0.25)
Q3=num.quantile(0.75)
IQR=Q3-Q1
lower=Q1-1.5*IQR
upper=Q3+1.5*IQR
outliers_count=((num<lower)|(num>upper)).sum()
outliers_percentage=(outliers_count/len(num))*100
print(outliers_percentage)
print(outliers_count)
```

```
carat    3.502039
depth    4.718205
table    1.121617
x        0.059325
y        0.053763
z        0.090842
dtype: float64
carat    1889
depth    2545
table     605
x          32
y          29
z          49
dtype: int64
```

In [11]:

```python
#check the distribution
plt.figure(figsize=(30,25),facecolor='orange')
plotnum=1
for i in num.columns:
    if(plotnum<7):
        ax=plt.subplot(3,4,plotnum)
        sns.distplot(num[i])
    plotnum+=1
plt.tight_layout()
```



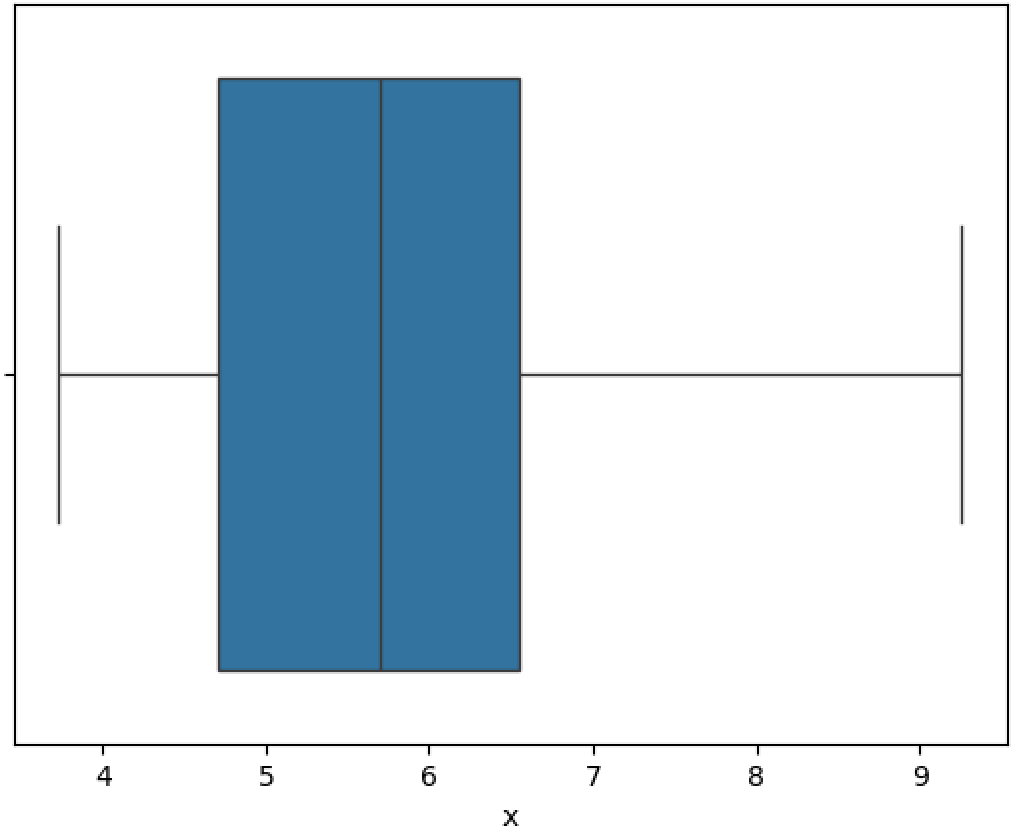
In [12]:

```python
# carat column
q1=df.carat.quantile(0.25)
q3=df.carat.quantile(0.75)
iqr=q3-q1
iqr
mini=q1-1.5*iqr
maxi=q3+1.5*iqr
df.loc[(df["carat"]<mini) |(df["carat"]>maxi),"carat"]=df["carat"].median()
```

In [13]:
```
#depth
q1=df.depth.quantile(0.25)
q3=df.depth.quantile(0.75)
iqr=q3-q1
iqr
mini=q1-1.5*iqr
maxi=q3+1.5*iqr
df.loc[(df["depth"]<mini) |(df["depth"]>maxi),"depth"]=df["depth"].mean()
df.loc[(df["depth"]<mini) |(df["depth"]>maxi),"depth"]
```
Out[13]:
```
Series([], Name: depth, dtype: float64)
```
In [14]:
```
df.loc[(df["depth"]<mini) |(df["depth"]>maxi),"depth"]
```
Out[14]:
```
Series([], Name: depth, dtype: float64)
```
In [15]:
```
#table
mean = np.mean(df.table)

std = np.std(df.table)

mini = mean-3*std
maxi = mean+3*std
df.loc[(df["table"]<mini)|(df["table"]>maxi),"table"]=df["table"].mean()
```
In [16]:
```
df.loc[(df["table"]<mini)|(df["table"]>maxi),"table"]
```
Out[16]:
```
Series([], Name: table, dtype: float64)
```
In [17]:
```
#column x
```
In [18]:
```
q1=df.x.quantile(0.25)
q3=df.x.quantile(0.75)
iqr=q3-q1
iqr
mini=q1-1.5*iqr
maxi=q3+1.5*iqr
df.loc[(df["x"]<mini) |(df["x"]>maxi),"x"]=df["x"].median()
```
In [19]:
```
df.loc[(df["x"]<mini) |(df["x"]>maxi),"x"]
```
Out[19]:
```
Series([], Name: x, dtype: float64)
```
In [20]:
```
sns.boxplot(data=df, x=df.x)
```
Out[20]:
```
<Axes: xlabel='x'>
```



In [21]:
```
df.drop("Unnamed: 0", axis=1,inplace=True)
df
```

Out[21]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.500000	55.000000	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.800000	61.000000	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	61.749405	57.457184	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.400000	58.000000	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.300000	58.000000	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.800000	57.000000	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.100000	55.000000	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.800000	60.000000	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.000000	58.000000	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.200000	55.000000	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [22]:
```
# plt.figure(figsize=(10,8))
# corr=(data.select_dtypes(exclude="object").corr())
```

```python
# corr
# sns.heatmap(corr, annot=True)
```

In [23]:
```python
corr=df.select_dtypes(exclude="object").corr()
sns.heatmap(corr,annot=True)
```

Out[23]: <Axes: >

In [24]:
```python
df.drop("z", axis=1, inplace=True)
df
```

Out[24]:

	carat	cut	color	clarity	depth	table	price	x	y
0	0.23	Ideal	E	SI2	61.500000	55.000000	326	3.95	3.98
1	0.21	Premium	E	SI1	59.800000	61.000000	326	3.89	3.84
2	0.23	Good	E	VS1	61.749405	57.457184	327	4.05	4.07
3	0.29	Premium	I	VS2	62.400000	58.000000	334	4.20	4.23
4	0.31	Good	J	SI2	63.300000	58.000000	335	4.34	4.35
...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.800000	57.000000	2757	5.75	5.76
53936	0.72	Good	D	SI1	63.100000	55.000000	2757	5.69	5.75
53937	0.70	Very Good	D	SI1	62.800000	60.000000	2757	5.66	5.68
53938	0.86	Premium	H	SI2	61.000000	58.000000	2757	6.15	6.12
53939	0.75	Ideal	D	SI2	62.200000	55.000000	2757	5.83	5.87

53940 rows × 9 columns

In [25]:

```
#label encoding
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
for i in df.select_dtypes(include="object"):
    df[i]=l.fit_transform(df[i])
```

In [26]:

```
df
```

Out[26]:

	carat	cut	color	clarity	depth	table	price	x	y
0	0.23	2	1	3	61.500000	55.000000	326	3.95	3.98
1	0.21	3	1	2	59.800000	61.000000	326	3.89	3.84
2	0.23	1	1	4	61.749405	57.457184	327	4.05	4.07
3	0.29	3	5	5	62.400000	58.000000	334	4.20	4.23
4	0.31	1	6	3	63.300000	58.000000	335	4.34	4.35
...	...	...	...	...	...	...	...	...	...
53935	0.72	2	0	2	60.800000	57.000000	2757	5.75	5.76
53936	0.72	1	0	2	63.100000	55.000000	2757	5.69	5.75
53937	0.70	4	0	2	62.800000	60.000000	2757	5.66	5.68
53938	0.86	3	4	3	61.000000	58.000000	2757	6.15	6.12
53939	0.75	2	0	3	62.200000	55.000000	2757	5.83	5.87

53940 rows × 9 columns

In [27]:
```
#train and split
X=df.drop("price" ,axis=1)
y=df.price
```

In [28]:
```
from sklearn.model_selection import train_test_split
x_train,x_test, y_train,y_test=train_test_split(X,y, random_state=13, test_size=0.2
```

In [29]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(43152, 8)
(10788, 8)
(43152,)
(10788,)
```

In [30]:
```
#scaling
from sklearn.preprocessing import MinMaxScaler
s=MinMaxScaler()
```

In [31]:
```
x_train[['carat', 'depth','cut','color','clarity','table', 'x','y']]=s.fit_transfor
x_train
```

Out[31]:

	carat	cut	color	clarity	depth	table	x	y
1715	0.066667	0.75	0.500000	0.714286	0.169492	0.615385	0.135624	0.076740
24957	0.277778	0.75	0.333333	0.428571	0.542373	0.692308	0.799277	0.137521
22696	0.055556	0.75	0.500000	0.714286	0.084746	0.615385	0.124774	0.074363
12972	0.450000	1.00	0.666667	0.714286	0.474576	0.461538	0.484629	0.109338
47735	0.222222	0.50	0.166667	0.285714	0.423729	0.384615	0.318264	0.092360
...	...	...	...	...	...	...	...	...
22260	0.727778	1.00	0.666667	0.285714	0.677966	0.307692	0.643761	0.124278
33634	0.094444	0.50	0.166667	0.285714	0.525424	0.384615	0.157324	0.077589
32842	0.088889	0.50	0.333333	0.714286	0.457627	0.307692	0.157324	0.077589
47280	0.055556	1.00	0.833333	0.285714	0.525424	0.769231	0.097649	0.072835
33106	0.105556	0.50	0.500000	0.714286	0.542373	0.461538	0.171790	0.080475

43152 rows × 8 columns

In [32]:

```
x_test[['carat','depth','cut','color','clarity','table', 'x','y']]=s.transform(x_te
x_test
```

Out[32]:

	carat	cut	color	clarity	depth	table	x	y
11622	0.555556	0.50	0.833333	0.428571	0.288136	0.538462	0.576854	0.115959
53686	0.333333	1.00	0.166667	0.428571	0.694915	0.230769	0.394213	0.102207
6479	0.455556	0.75	0.333333	0.428571	0.389831	0.846154	0.508137	0.110187
40391	0.183333	0.50	0.500000	0.428571	0.186441	0.461538	0.274864	0.089983
37663	0.122222	0.75	0.333333	0.285714	0.508475	0.538462	0.200723	0.081664
...	...	...	...	...	...	...	...	...
10910	0.466667	0.75	0.666667	0.285714	0.118644	0.538462	0.515371	0.113243
19279	0.444444	1.00	0.500000	0.142857	0.499899	0.846154	0.502712	0.111545
37775	0.116667	0.50	0.166667	0.714286	0.508475	0.384615	0.189873	0.080475
41250	0.055556	1.00	0.333333	0.285714	0.559322	0.692308	0.095841	0.072666
18461	0.727778	0.00	0.666667	0.285714	0.499899	0.769231	0.721519	0.129542

10788 rows × 8 columns

In [33]:

```
# model training
from sklearn.linear_model import LinearRegression
```

```
model=LinearRegression()
model.fit(x_train,y_train)
```

Out[33]:

```
▾ LinearRegression
LinearRegression()
```

In [34]:

```
y_pred=model.predict(x_test)
y_pred
```

Out[34]:

```
array([ 6889.04124634,  5089.4357925 ,  6120.62929429, ...,
        1480.69386083, -1526.22861944,  9059.72001347])
```

In [35]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
mse=mean_squared_error(y_test,y_pred)
mse
```

Out[35]:

```
2964242.1740486836
```

In [36]:

```
mae=mean_absolute_error(y_test, y_pred)
mae
```

Out[36]:

```
1289.5139887046507
```

In [37]:

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

Out[37]:

```
0.8149437115127462
```

In [38]:

```
# adjusted r2 score
adj_r2=1-(1-0.81)*(5394-1)/(5394-8-1)
adj_r2
```

Out[38]:

```
0.8097177344475395
```

In [ ]:

```
# adj r2 score should be less than a r2
```

In [ ]:

```
#Ridge
```

In [118…

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
ridge=Ridge(random_state=10)
parameters={'alpha':[0.01,0.001]}
ridge_regressor=GridSearchCV(ridge,parameters,scoring="neg_root_mean_squared_error"
ridge_regressor.fit(x_train,y_train)
```

Out[118…

```
▸ GridSearchCV
▸ estimator: Ridge
    ▸ Ridge
```

In [119…

```python
y_pred_r=ridge_regressor.predict(x_test)
y_pred_r
```

Out[119…

```
array([ 6889.29885425,  5089.1798881 ,  6120.72349834, ...,
        1480.81131885, -1526.20617813,  9059.8776627 ])
```

In [120…

```python
r2_score(y_test,y_pred_r)
```

Out[120…

```
0.8149330248416461
```

In [121…

```python
#lasso
from sklearn.linear_model import Lasso
lasso=Lasso(alpha=0.01,random_state=13)
lasso.fit(x_train,y_train)
```

Out[121…

```
▾ Lasso
Lasso(alpha=0.01, random_state=13)
```

In [122…

```python
y_pred_la=lasso.predict(x_test)
y_pred_la
```

Out[122…

```
array([ 6889.86068986,  5088.56727117,  6120.99903369, ...,
        1480.94152073, -1526.22133244,  9060.41268522])
```

In [76]:

```python
r2_score(y_test,y_pred_la)
```

Out[76]:

```
0.814916314339691
```

In [47]:

```python
conclusion:
# when i tried both Min max and standard scaler nothing improvement is there
# random state 13 transforms the accuracy to from 80 t0 81
# used ridge and apply GridsearchCV to control the overfitting
```

In [1]:

```python
pwd
```

Out[1]:

```
'C:\\Users\\User\\ML models'
```