Diamond Price Prediction

Life cycle of Machine learning Project

- 1. Understanding the problem statement
- 1. Data Collection
- 1. Exploratory Data Analysis
- 1. Data Cleaning
- 1. Data Pre-Processing
- 1. Model Training

Introduction About the Data :

The dataset The goal is to predict `price` of given diamond (Regression Analysis).

There are 10 independent variables (including `id`):

- `id` : unique identifier of each diamond
- `carat` : Carat (ct.) refers to the unique unit of weight measurement used exclusively to weigh gemstones and diamonds.
- `cut` : Quality of Diamond Cut
- `color` : Color of Diamond
- `clarity` : Diamond clarity is a measure of the purity and rarity of the stone, graded by the visibility of these characteristics under 10-power magnification.
- `depth` : The depth of diamond is its height (in millimeters) measured from the culet (bottom tip) to the table (flat, top surface)
- `table` : A diamond's table is the facet which can be seen when the stone is viewed face up.
- `x` : Diamond X dimension
- `y` : Diamond Y dimension
- `x` : Diamond Z dimension

Target variable:

- `price`: Price of the given Diamond.

Dataset Source Link : https://www.kaggle.com/competitions/playground-series-s3e8/data?select=train.csv

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the Dataset

In [2]:

```
## Data Ingestions step
df=pd.read_csv('gemstone.csv')
```

Show the top 5 records

Show the top 5 records

In [3]:

```
df.head()
```

Out[3]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

Check Missing values

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
id          0
carat       0
cut         0
color       0
clarity     0
depth       0
table       0
x           0
y           0
z           0
price       0
dtype: int64
```

In [5]:

```
### No missing values present in the data
```

Summary of Dataset

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193573 entries, 0 to 193572
Data columns (total 11 columns):
 #   Column    Non-Null Count   Dtype  
---  ------    --------------   -----  
 0   id        193573 non-null  int64  
 1   carat     193573 non-null  float64
 2   cut       193573 non-null  object 
 3   color     193573 non-null  object 
 4   clarity   193573 non-null  object 
 5   depth     193573 non-null  float64
 6   table     193573 non-null  float64
 7   x         193573 non-null  float64
 8   y         193573 non-null  float64
 9   z         193573 non-null  float64
 10  price     193573 non-null  int64  
dtypes: float64(6), int64(2), object(3)
memory usage: 16.2+ MB
```

descriptive summary of the dataset

In [42]:

```
df.describe()
```

Out[42]:

	carat	cut	color	clarity	depth	table	x	
count	193573.000000	193573.000000	193573.000000	193573.000000	193573.000000	193573.000000	193573.000000	193573.000
mean	0.790688	4.132152	3.516157	3.975084	61.820574	57.227675	5.715312	5.720
std	0.462688	0.994157	1.623091	1.501776	1.081704	1.918844	1.109422	1.102
min	0.200000	1.000000	1.000000	1.000000	52.100000	49.000000	0.000000	0.000
25%	0.400000	3.000000	2.000000	3.000000	61.300000	56.000000	4.700000	4.710
50%	0.700000	4.000000	4.000000	4.000000	61.900000	57.000000	5.700000	5.720
75%	1.030000	5.000000	5.000000	5.000000	62.400000	58.000000	6.510000	6.510
max	3.500000	5.000000	7.000000	8.000000	71.600000	79.000000	9.650000	10.010

shape of the dataset

In [43]:

```
df.shape
```

Out[43]:

```
(193573, 10)
```

Check column names

In [41]:

```
df.columns
```

Out[41]:

```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z',
       'price'],
      dtype='object')
```

drop the id column

In [8]:

```
## Lets drop the id column
df=df.drop(labels=['id'],axis=1)
df.head()
```

Out[8]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

check the data type of every columns

In [44]:

```
df.dtypes
```

Out[44]:

```
carat      float64
cut          int64
color        int64
clarity      int64
depth      float64
table      float64
x          float64
y          float64
z          float64
price        int64
dtype: object
```

check for duplicated records

In [9]:

```
## check for duplicated records
df.duplicated().sum()
```

Out[9]:

```
0
```

segregate numerical and categorical columns

In [10]:

```
## segregate numerical and categorical columns

numerical_columns=df.columns[df.dtypes!='object']
categorical_columns=df.columns[df.dtypes=='object']
print("Numerical columns:",numerical_columns)
print('Categorical Columns:',categorical_columns)
```

```
Numerical columns: Index(['carat', 'depth', 'table', 'x', 'y', 'z', 'price'], dtype='obje
ct')
Categorical Columns: Index(['cut', 'color', 'clarity'], dtype='object')
```

In [11]:

```
df[categorical_columns].describe()
```

Out[11]:

	cut	color	clarity
count	193573	193573	193573
unique	5	7	8
top	Ideal	G	SI1
freq	92454	44391	53272

In [12]:

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

Out[12]:

```
Ideal        92454
Premium      49910
Very Good    37566
```

```
Good         11622
Fair          2021
Name: cut, dtype: int64
```

In [13]:

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

Out[13]:

```
G    44391
E    35869
F    34258
H    30799
D    24286
I    17514
J     6456
Name: color, dtype: int64
```

In [14]:

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

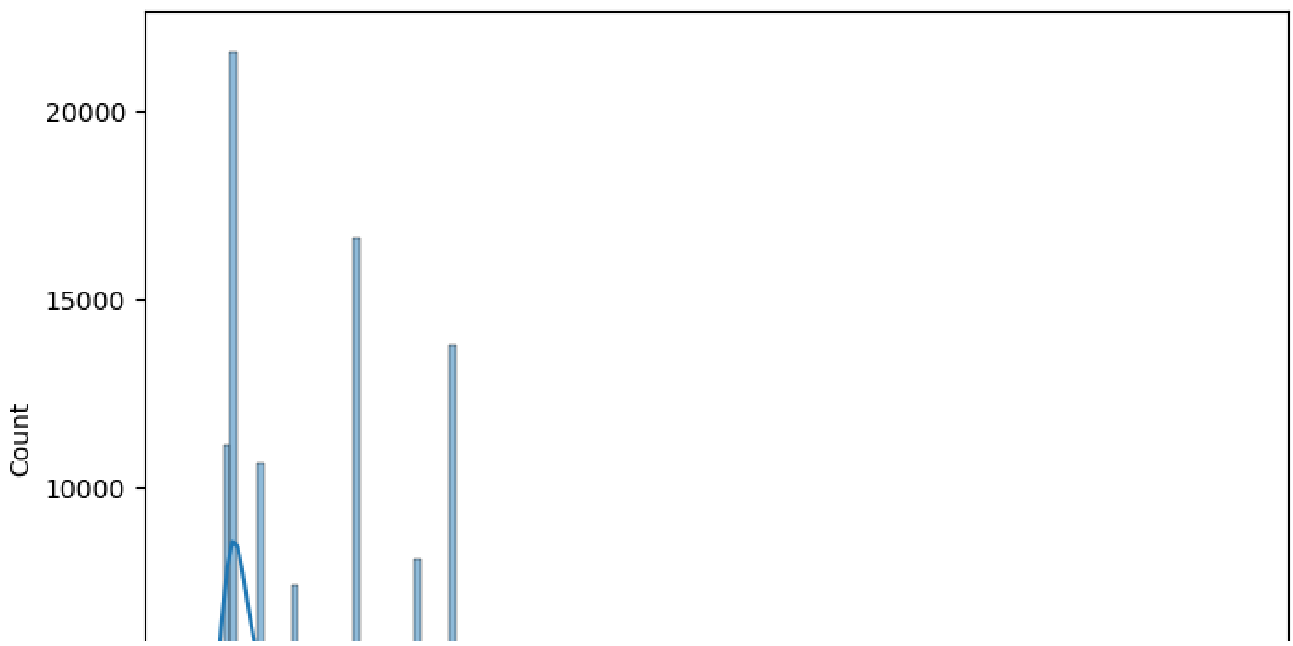
Out[14]:

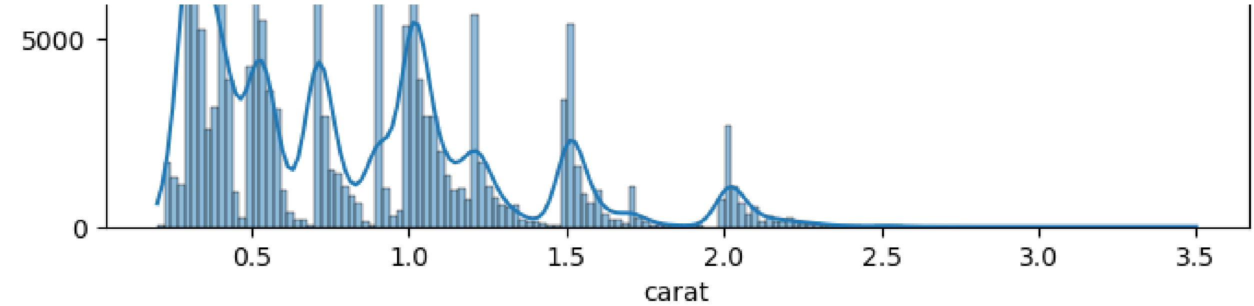
```
SI1     53272
VS2     48027
VS1     30669
SI2     30484
VVS2    15762
VVS1    10628
IF       4219
I1        512
Name: clarity, dtype: int64
```

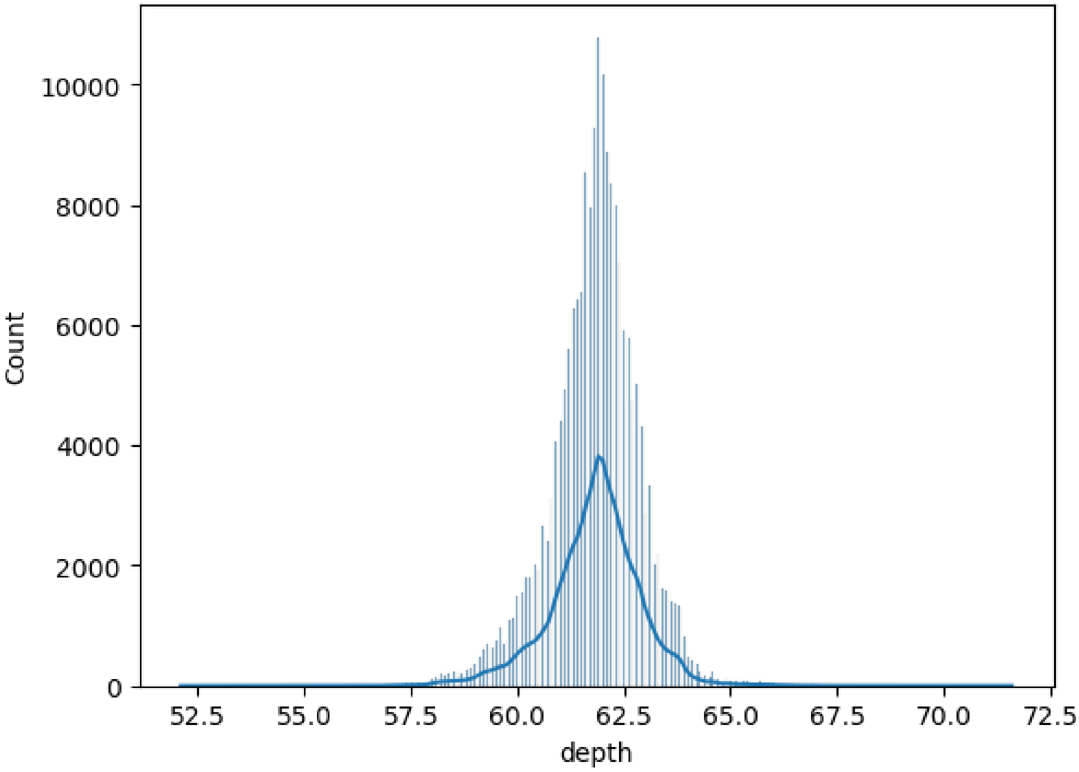
Histplot on Numerical columns

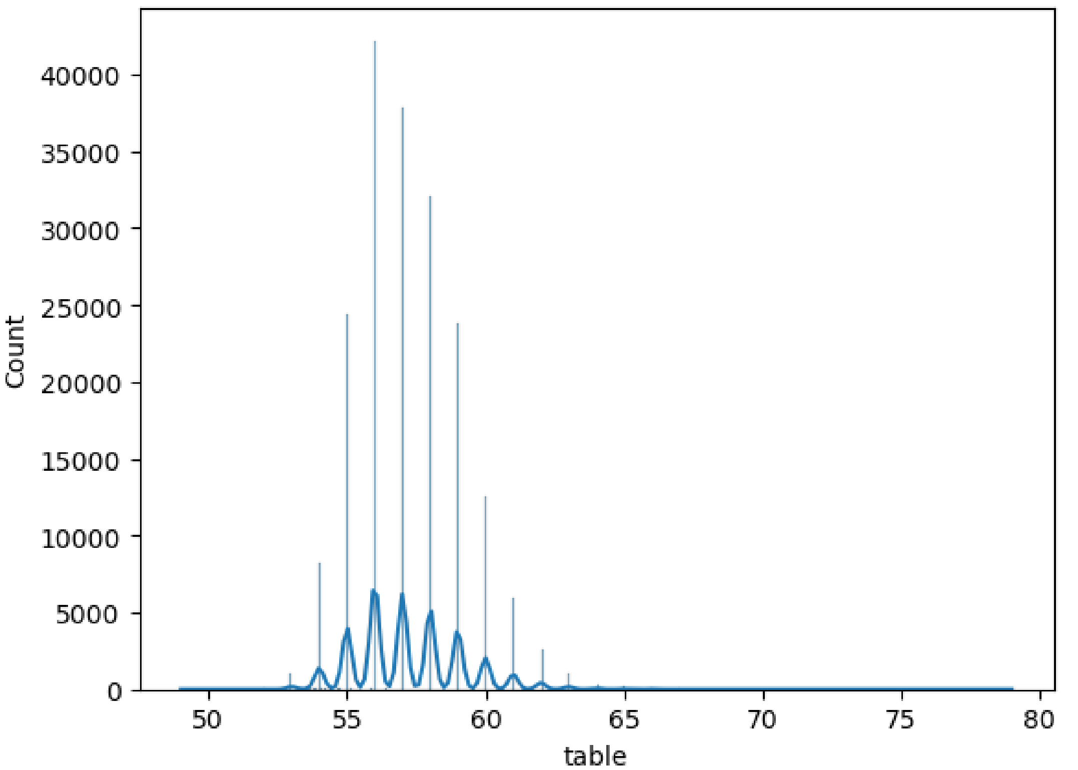
In [15]:

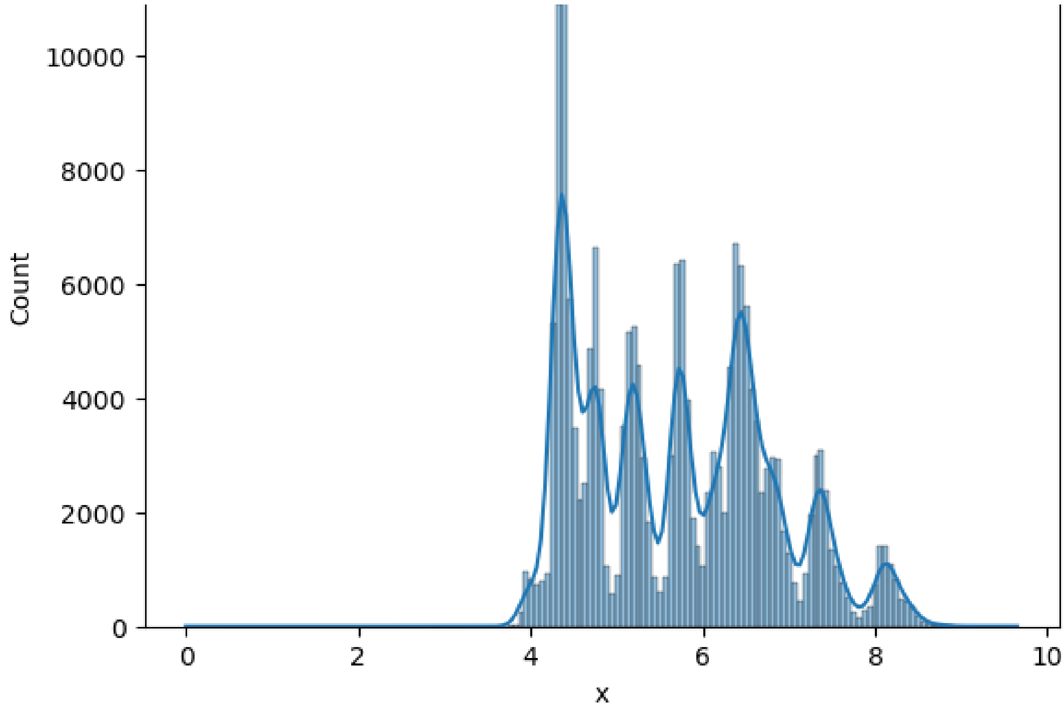
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
x=0
for i in numerical_columns:
    sns.histplot(data=df,x=i,kde=True)
    print('\n')
    plt.show()
```

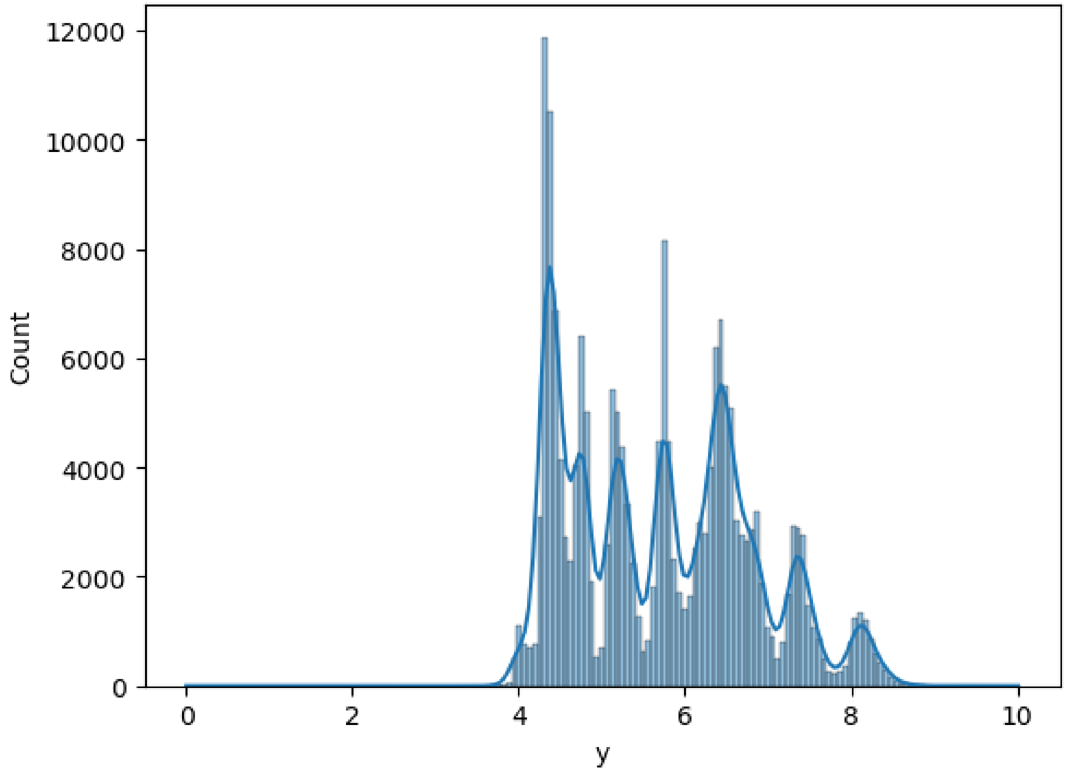


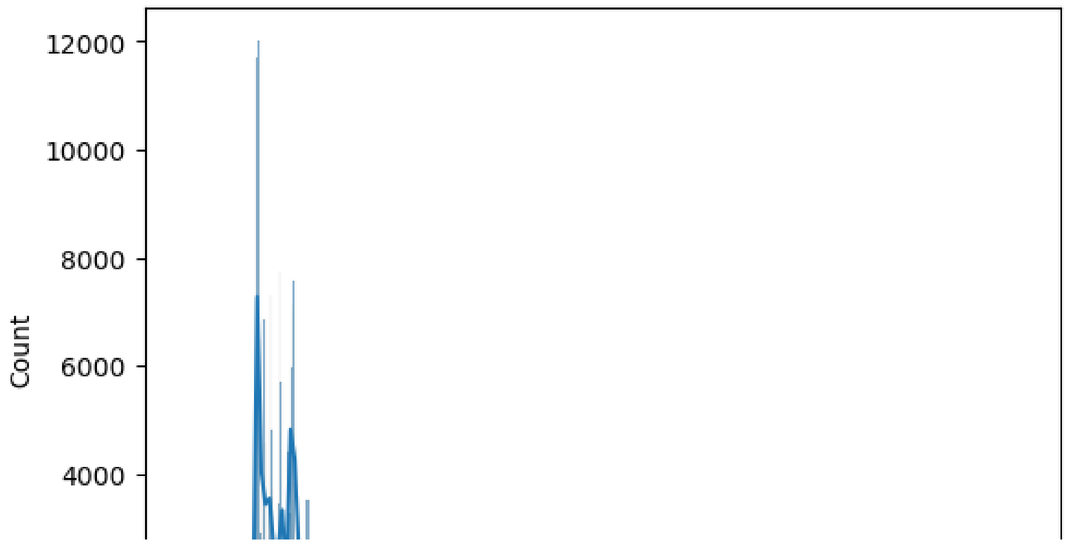

5000
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
carat

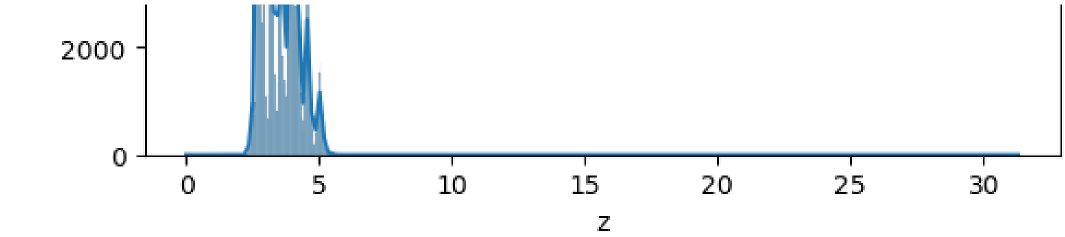

Count
10000
8000
6000
4000
2000
0
52.5
55.0
57.5
60.0
62.5
65.0
67.5
70.0
72.5
depth

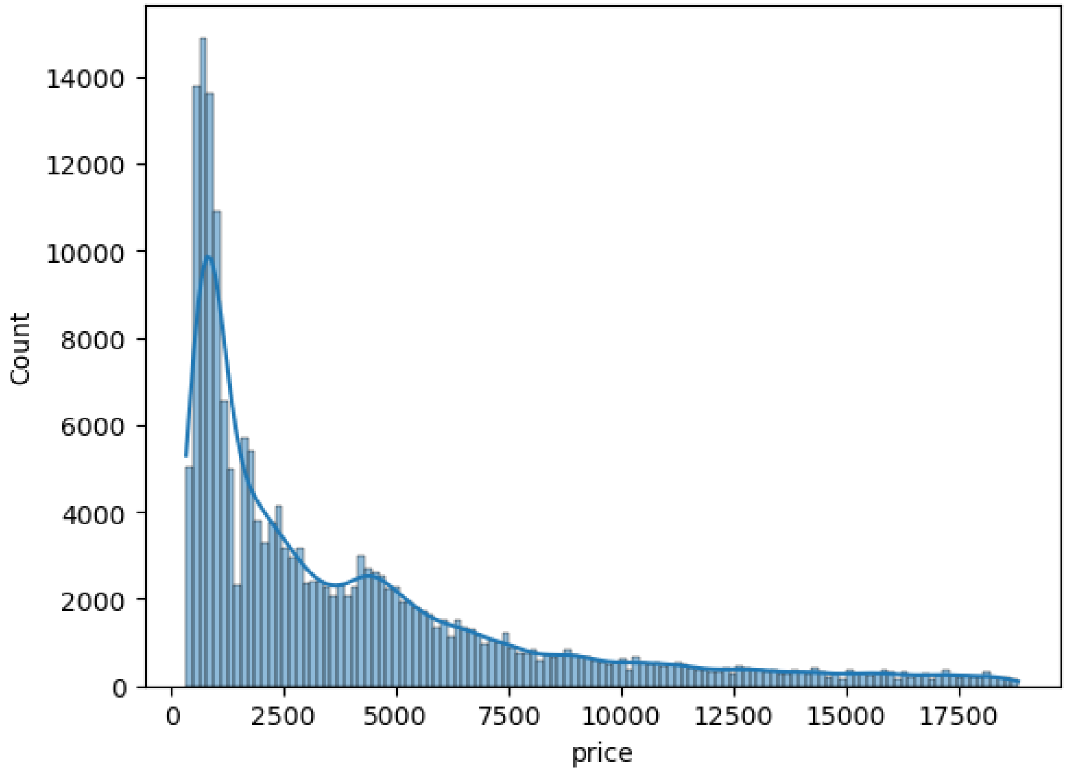

Count
40000
35000
30000
25000
20000
15000
10000
5000
0
50
55
60
65
70
75
80
table


Count
x


Count
y


Count





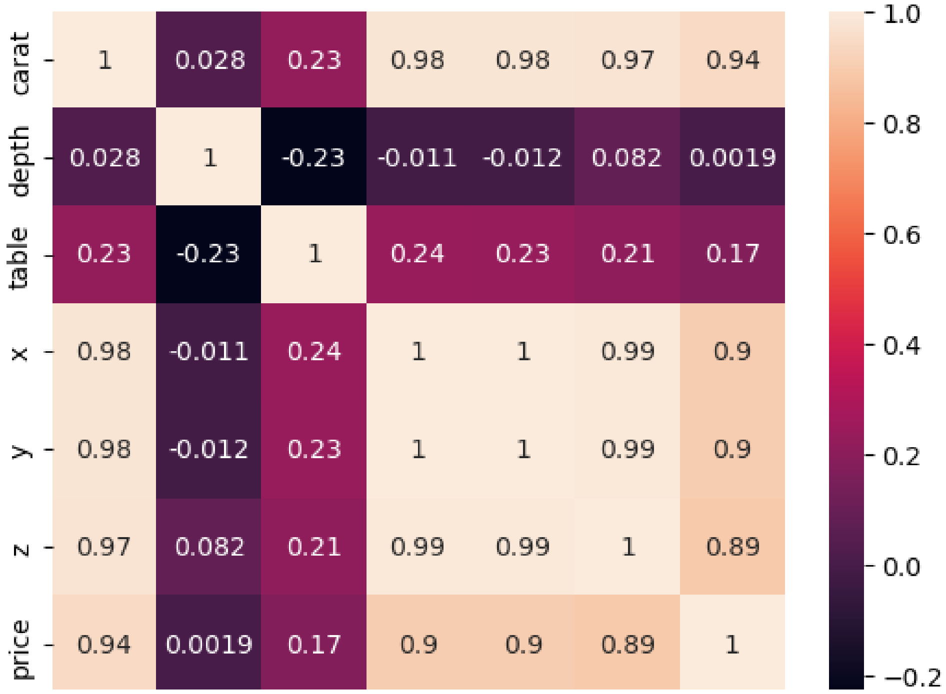
Check correlation

In [17]:

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

Out[17]:

```
<AxesSubplot:>
```



In [18]:

```
##Currently we will not execute this
## df.drop(labels=['x','y','z'],axis=1)
```

In [19]:

```
df.head()
```

Out[19]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

Check unique in cut column

In [20]:

```
df['cut'].unique()
```

Out[20]:

```
array(['Premium', 'Very Good', 'Ideal', 'Good', 'Fair'], dtype=object)
```

In [21]:

```
cut_map={"Fair":1,"Good":2,"Very Good":3,"Premium":4,"Ideal":5}
```

In [22]:

```
df['clarity'].unique()
```

Out[22]:

```
array(['VS2', 'SI2', 'VS1', 'SI1', 'IF', 'VVS2', 'VVS1', 'I1'],
      dtype=object)
```

In [23]:

```
clarity_map = {"I1":1,"SI2":2 ,"SI1":3 ,"VS2":4 , "VS1":5 , "VVS2":6 , "VVS1":7 ,"IF":8}
```

In [24]:

```
df['color'].unique()
```

Out[24]:

```
array(['F', 'J', 'G', 'E', 'D', 'H', 'I'], dtype=object)
```

In [25]:

```
color_map = {"D":1 ,"E":2 ,"F":3 , "G":4 ,"H":5 , "I":6, "J":7}
```

In [26]:

```
df['cut']=df['cut'].map(cut_map)
df['clarity'] = df['clarity'].map(clarity_map)
df['color'] = df['color'].map(color_map)
```

In [27]:

```
df.head()
```

Out[27]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	4	3	4	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	3	7	2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	5	4	5	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	5	4	5	61.6	56.0	4.38	4.41	2.71	666
4	1.70	4	4	4	62.6	59.0	7.65	7.61	4.77	14453

Model Training

In [28]:

```
X = df.drop(labels=['price'],axis=1)
Y = df[['price']]
```

In [29]:

```
Y
```

Out[29]:

	price
0	13619
1	13387
2	2772
3	666
4	14453
...	...
193568	1130
193569	2874
193570	3036
193571	681
193572	2258

193573 rows × 1 columns

In [30]:

```
# Define which columns should be ordinal-encoded and which should be scaled
categorical_cols = X.select_dtypes(include='object').columns
numerical_cols = X.select_dtypes(exclude='object').columns
```

In [31]:

```
# Define the custom ranking for each ordinal variable
cut_categories = ['Fair', 'Good', 'Very Good','Premium','Ideal']
color_categories = ['D', 'E', 'F', 'G', 'H', 'I', 'J']
clarity_categories = ['I1','SI2','SI1','VS2','VS1','VVS2','VVS1','IF']
```

In [32]:

```
from sklearn.impute import SimpleImputer ## HAndling Missing Values
from sklearn.preprocessing import StandardScaler # HAndling Feature Scaling
from sklearn.preprocessing import OrdinalEncoder # Ordinal Encoding
```

```
## pipelines
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

In [33]:

```
## Numerical Pipeline
num_pipeline=Pipeline(
    steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())

    ]

)

# Categorigal Pipeline
cat_pipeline=Pipeline(
    steps=[
    ('imputer',SimpleImputer(strategy='most_frequent')),
    ('ordinalencoder',OrdinalEncoder(categories=[cut_categories,color_categories,clarity_
categories])),
    ('scaler',StandardScaler())
    ]

)

preprocessor=ColumnTransformer([
('num_pipeline',num_pipeline,numerical_cols),
('cat_pipeline',cat_pipeline,categorical_cols)
])
```

Train test split

In [34]:

```
## 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=0.30,random_state=30)
```

X_train=pd.DataFrame(preprocessor.fit_transform(X_train),columns=preprocessor.get_feature_names_out())
X_test=pd.DataFrame(preprocessor.transform(X_test),columns=preprocessor.get_feature_names_out())

In [35]:

```
## Model Training

from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

In [36]:

```
regression=LinearRegression()
regression.fit(X_train,y_train)
```

Out[36]:

```
LinearRegression()
```

In [37]:

```
regression.coef_
```

Out[37]:

```
array([[13908.5914125 ,    72.89534931,  -283.5924305 ,   433.73822644,
         -122.8355985 ,   -36.71694327, -1550.78910628,  -452.96841314,
          -91.86321724]])
```

In [38]:

```
regression.intercept_
```

Out[38]:

```
array([13417.41178519])
```

In [39]:

```
import numpy as np
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

In [40]:

```
## Train multiple models

models={
    'LinearRegression':LinearRegression(),
    'Lasso':Lasso(),
    'Ridge':Ridge(),
    'Elasticnet':ElasticNet()
}
trained_model_list=[]
model_list=[]
r2_list=[]

for i in range(len(list(models))):
    model=list(models.values())[i]
    model.fit(X_train,y_train)

    #Make Predictions
    y_pred=model.predict(X_test)

    mae, rmse, r2_square=evaluate_model(y_test,y_pred)

    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model Training Performance')
    print("RMSE:",rmse)
    print("MAE:",mae)
    print("R2 score",r2_square*100)

    r2_list.append(r2_square)

    print('='*35)
    print('\n')
```

```
LinearRegression
Model Training Performance
RMSE: 1013.9047094344003
MAE: 674.025511579685
R2 score 93.68908248567512
===================================



Lasso
Model Training Performance
RMSE: 1013.9959392651111
MAE: 676.0374759508263
R2 score 93.68794673823564
===================================



Ridge
Model Training Performance
```

```
RMSE: 1013.9048469792065
MAE: 674.1622802042341
R2 score 93.68908077341561
===================================


Elasticnet
Model Training Performance
RMSE: 1668.5101542521966
MAE: 1147.899878068513
R2 score 82.90945424489222
===================================
```

In [ ]:

```
model_list
```

In [ ]:

```

```

In [ ]:

```

```