In [3]:

```python
import pandas as pd
```

In [4]:

```python
## Data Ingestions step
df=pd.read_csv('https://raw.githubusercontent.com/krishnaik06/FSDSRegression/main/notebooks/data/gemstone.csv')
df.head()
```

Out[4]:

	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

In [5]:

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

Out[5]:

```
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 [ ]:

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

In [6]:

```python
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
```

[7]:

```
df.head()
```

[7]:

	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

[8]:

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

[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

[9]:

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

[9]: 0

[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='object')
Categorical Columns: Index(['cut', 'color', 'clarity'], dtype='object')
```

[11]:

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

[11]:

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

2]:

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

2]:

```
cut
Ideal        92454
Premium      49910
Very Good    37566
Good         11622
Fair          2821
Name: count, dtype: int64
```

3]:

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

3]:

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

4]:

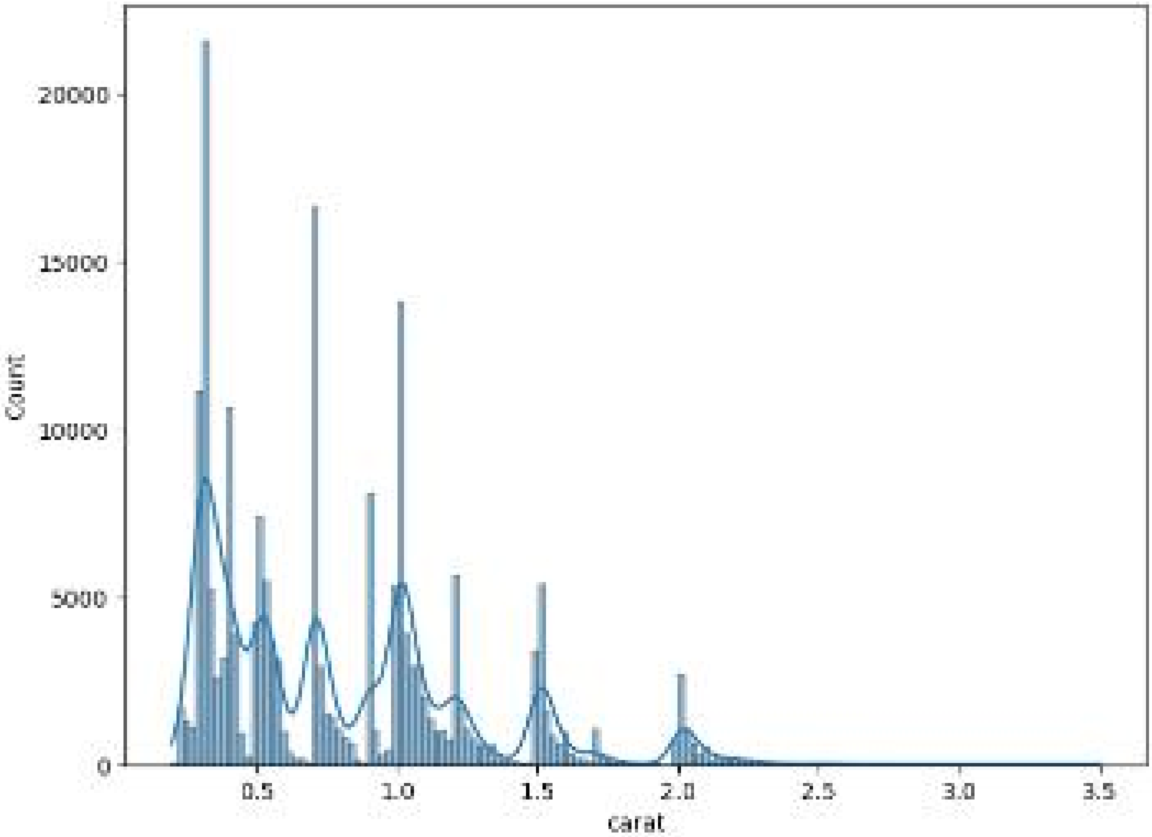
```
df['clarity'].value_counts()
```

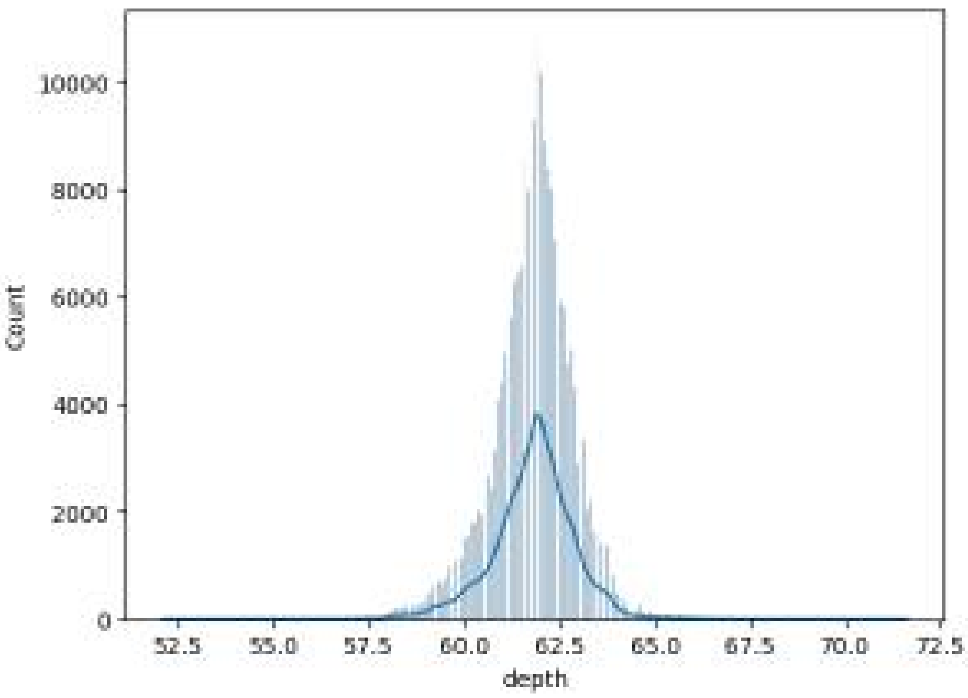
4]:

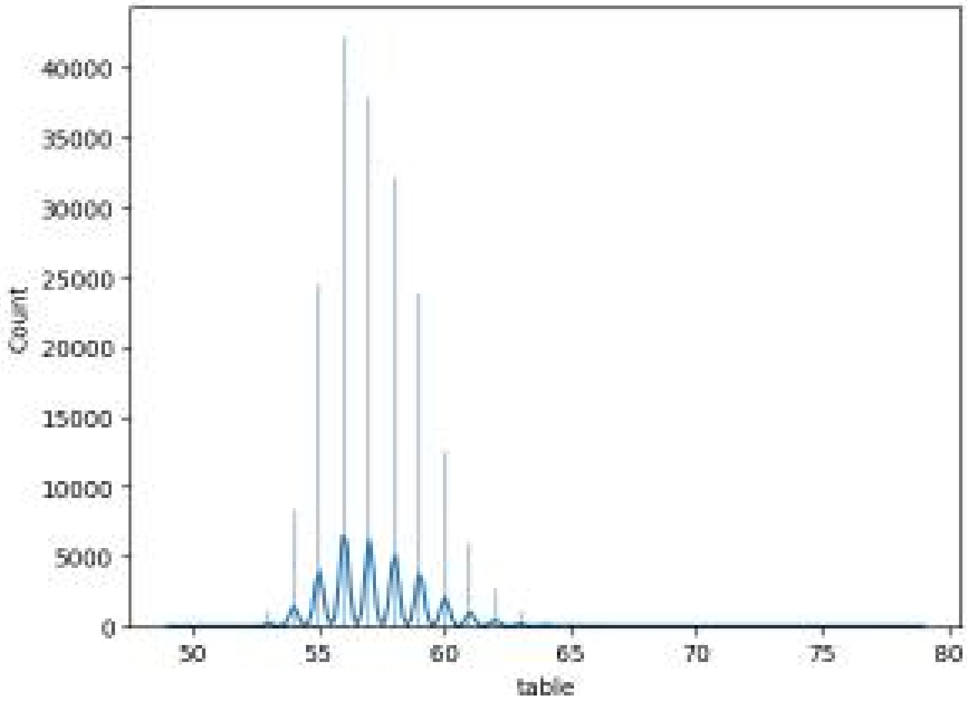
```
clarity
SI1     53272
VS2     48027
VS1     30669
SI2     30484
VVS2    15762
VVS1    18628
IF       4219
I1        512
Name: count, dtype: int64
```

Name: count, dtype: int64

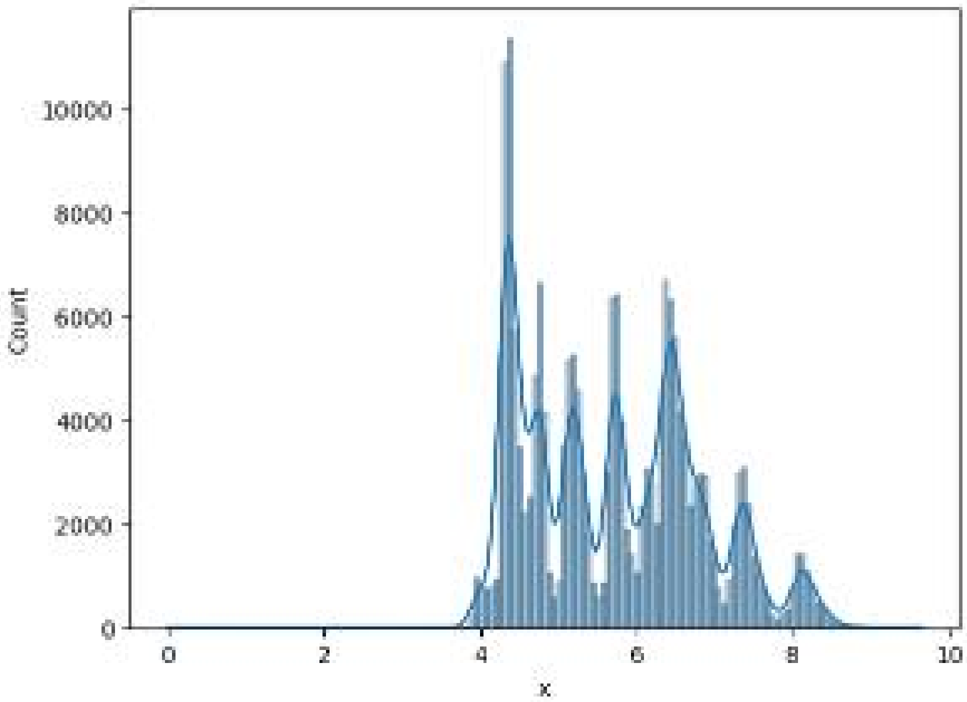
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
x=0
for i in numerical_columns:
    sns.histplot(data=df,x=i,kde=True)
    print('\n')
    plt.show()
```







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



Count
0
2000
4000
6000
8000
10000
0
2
4
6
8
10
x

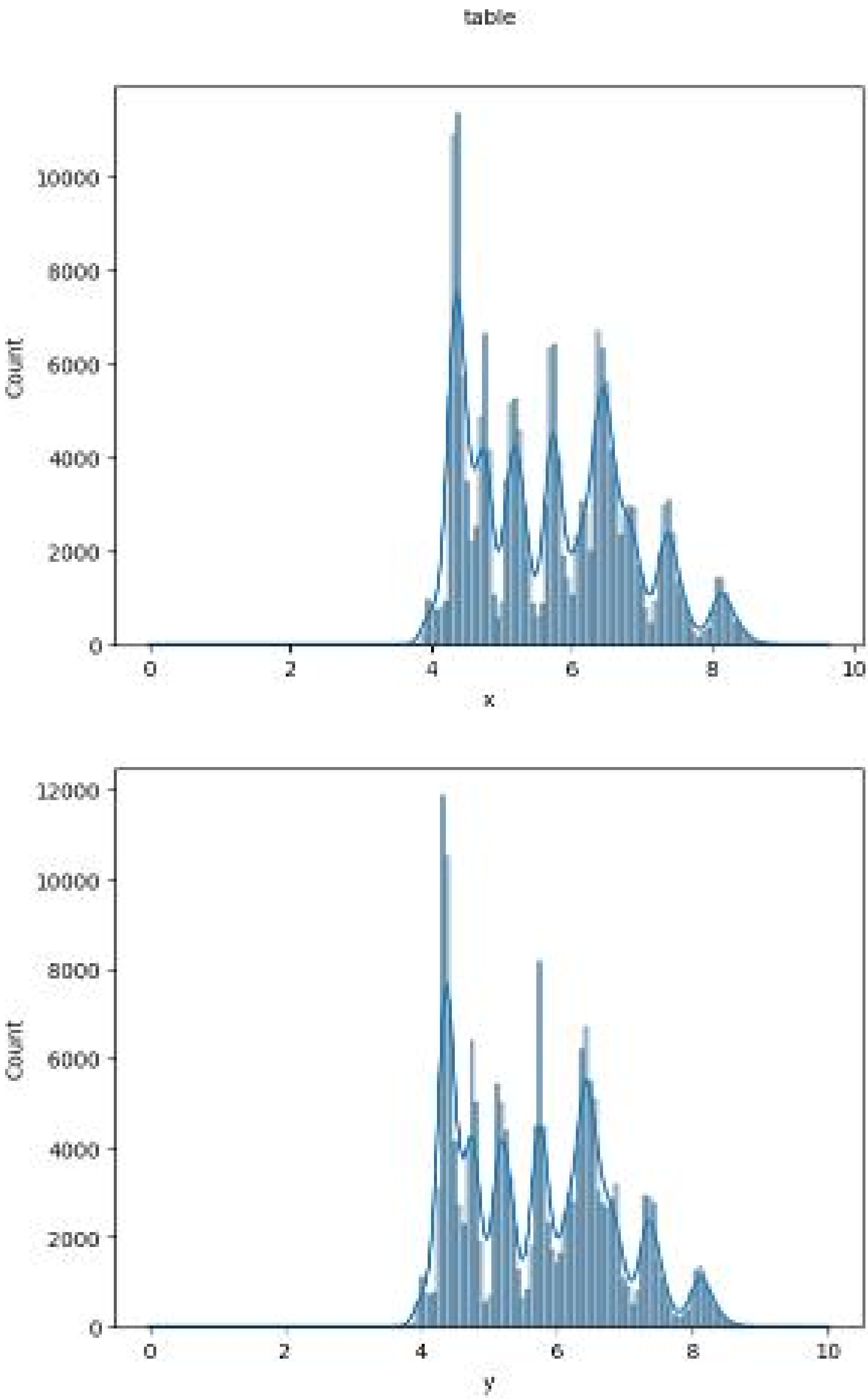
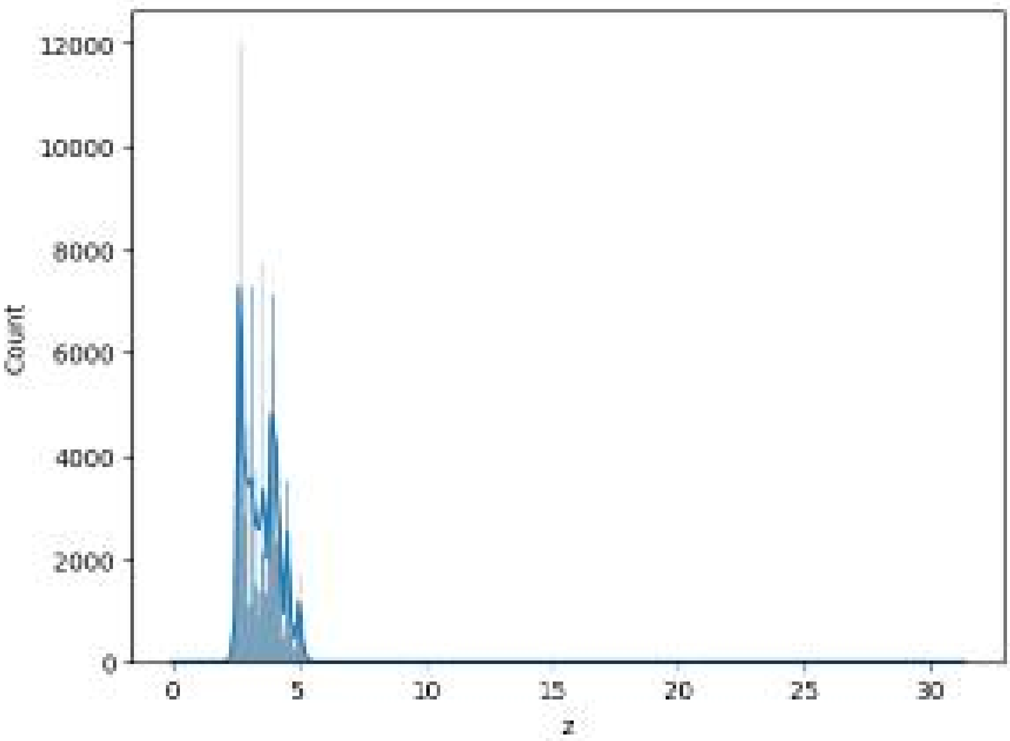
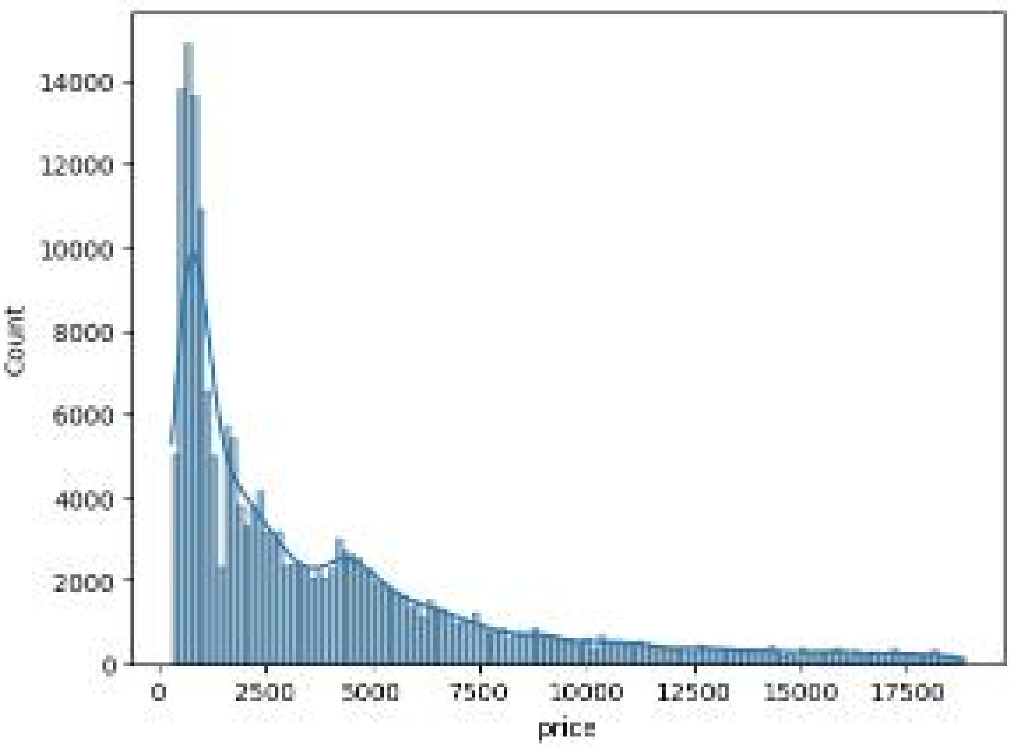


table
Count
x
Count
y

y

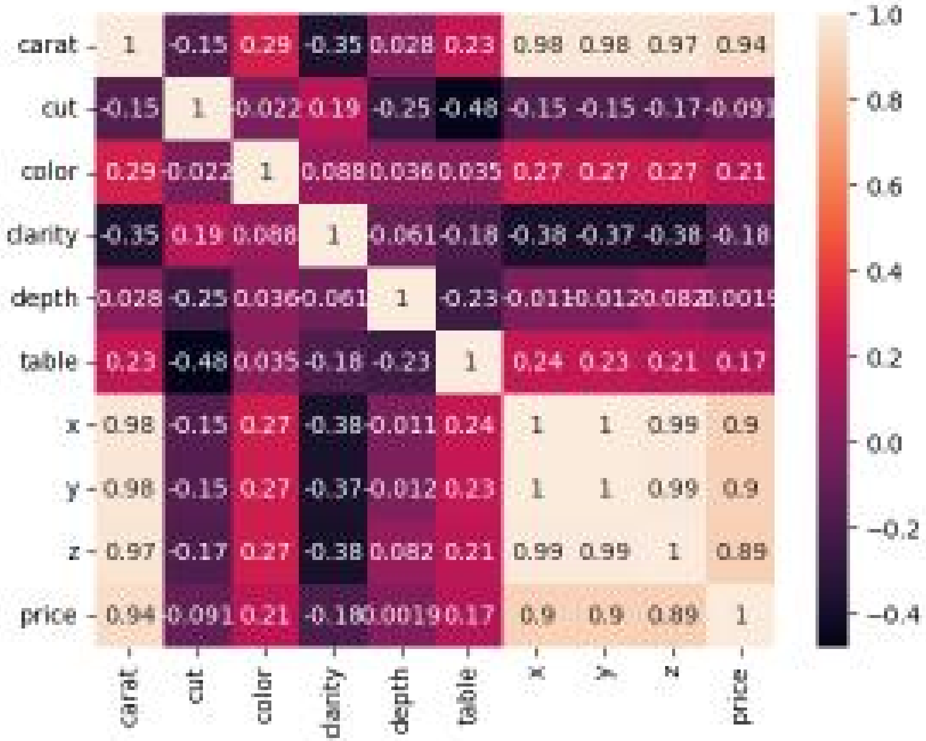




In [27]:

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

Out[27]: <Axes: >



In [18]:

```
df.head()
```

Out[18]:

	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

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

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

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

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

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

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

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

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

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

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

```
df.head()
```

	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 Trainning

[8]:

```
df=pd.read_csv('https://raw.githubusercontent.com/krishnaik06/FSDSRegression/main/notebooks/data/gemstone.csv')
df.head()
```

[8]:

	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

[9]:

```
df=df.drop(labels=['id'],axis=1)
```

[10]:

```
## Independent and dependent features
X = df.drop(labels=['price'],axis=1)
Y = df[['price']]
```

[11]:

```
Y
```

[11]:

	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

193573 rows × 9 columns

[12]:

```
X
```

[12]:

	carat	cut	color	clarity	depth	table	x	y	z
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50
3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71
4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77
...	...	...	...	...	...	...	...	...	...
193568	0.31	Ideal	D	VVS2	61.1	56.0	4.35	4.39	2.67
193569	0.70	Premium	G	VVS2	60.3	58.0	5.75	5.77	3.47
193570	0.73	Very Good	F	SI1	63.1	57.0	5.72	5.75	3.62
193571	0.34	Very Good	D	SI1	62.9	55.0	4.45	4.49	2.81
193572	0.71	Good	E	SI2	60.8	64.0	5.73	5.71	3.48

193573 rows × 9 columns

[13]:

```
# 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
```

[14]:

```
# 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']
```

[15]:

```
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
```

6]:

```python
## 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)
])
```

7]:

```python
## 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)
```

8]:

```python
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())
```

1]:

```python
X_train.head()
```

1]:

	num_pipeline__carat	num_pipeline__depth	num_pipeline__table	num_pipeline__x	num_pipeline__y	num_pipeline__z	cat_pipeline__cut
0	0.975439	0.849607	0.121531	1.042757	1.080970	1.123150	0.874076
1	0.235195	1.833637	0.121531	0.318447	0.279859	0.485354	2.144558
2	0.494617	0.815855	0.399800	0.570855	0.606458	0.673737	0.132136
3	1.018676	0.260701	0.921131	1.214034	1.244270	1.195605	0.132136
4	0.953821	0.664555	0.642862	1.069801	1.044681	1.094168	0.874076

```
X_test.head()
```

	num_pipeline__carat	num_pipeline__depth	num_pipeline__table	num_pipeline__x	num_pipeline__y	num_pipeline__z	cat_pipeline__cut
0	0.564688	0.942132	0.642862	0.429765	0.464061	0.500036	0.132136
1	0.175556	1.000906	0.121531	0.042137	0.028595	0.036132	1.138347
2	1.061913	0.260701	0.121531	1.304180	1.298703	1.268060	0.874076
3	0.970223	0.201927	1.963794	1.048629	0.996563	0.978049	0.132136
4	0.932202	1.312235	0.399800	1.006699	0.990248	1.065186	0.132136

```
## Model Training

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

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

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
regression.coef_
```

```
array([[ 6433.66803504,  -132.75843566,   -70.42922179, -1728.38971463,
         -499.29382619,   -63.39317848,    72.44537247,  -460.41684642,
          658.76431652]])
```

```
regression.intercept_
```

```
array([3970.76628955])
```

```
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
```

```python
## 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: 1013.9047894344082
MAE: 674.825511579685
R2 score 93.68988248567512
===================================


Lasso
Model Training Performance
RMSE: 1013.8784226767013
MAE: 675.071692336216
R2 score 93.68940971841704
===================================


Ridge
Model Training Performance
RMSE: 1013.9059272771631
MAE: 674.0555808798284
R2 score 93.68986732150594
===================================


Elasticnet
Model Training Performance
RMSE: 1533.4162456864048
MAE: 1060.7368759154729
R2 score 85.56494831165182
===================================
```

```python
model_list
```

```
['LinearRegression', 'Lasso', 'Ridge', 'Elasticnet']
```