Diamond price prediction with Polynomial regression

In [1]:
```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn. metrics import r2_score, mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#reading dataset
df = pd.read_csv('data/diamonds.csv')
df.head()
```

Out[2]:

	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

In [3]:
```
df.shape
```

Out[3]:
```
(53940, 11)
```

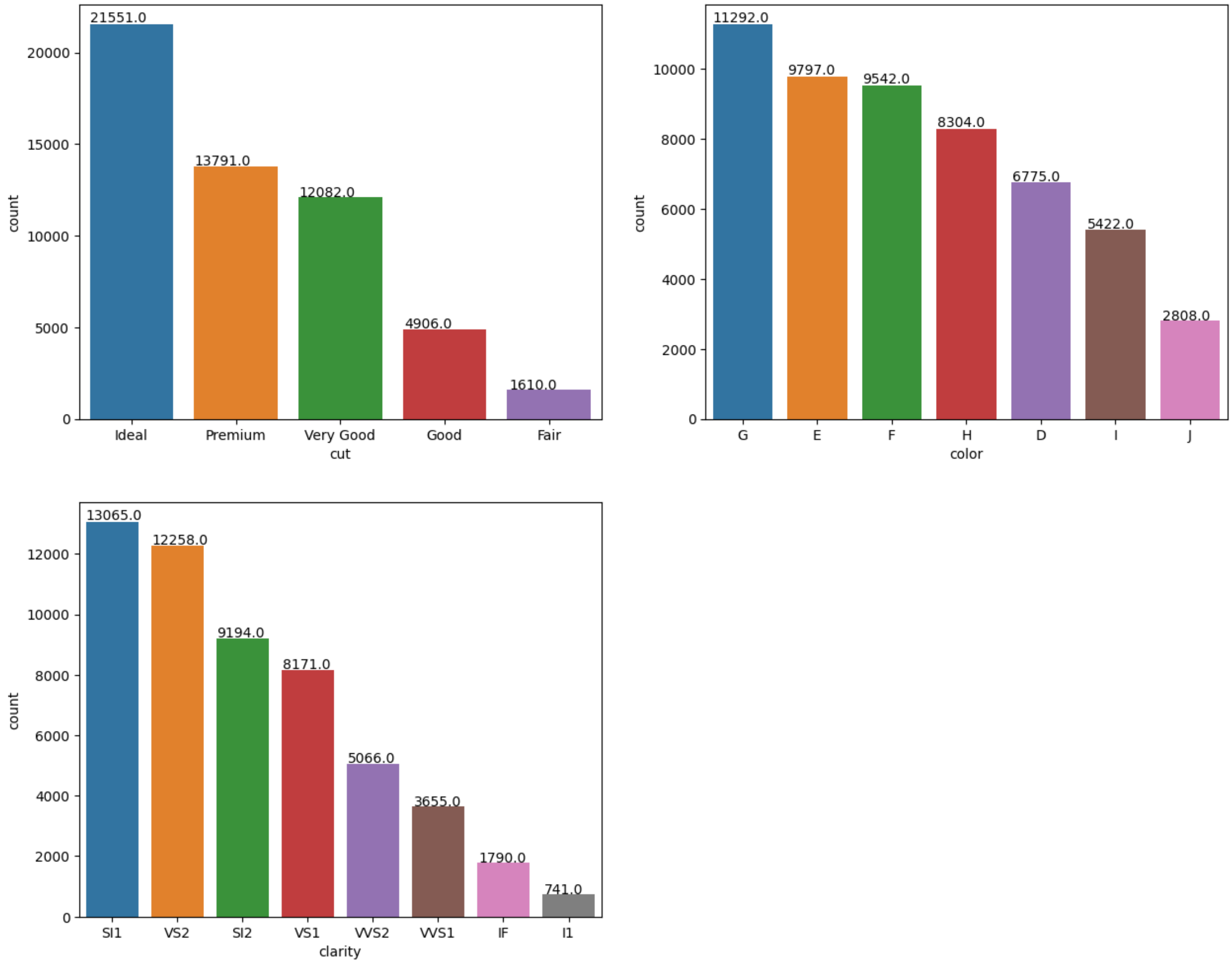
In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  53940 non-null  int64  
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object 
 3   color       53940 non-null  object 
 4   clarity     53940 non-null  object 
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64  
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

Exploratory data analysis

In [5]:
```
#ploting bar charts for categorical variables
```

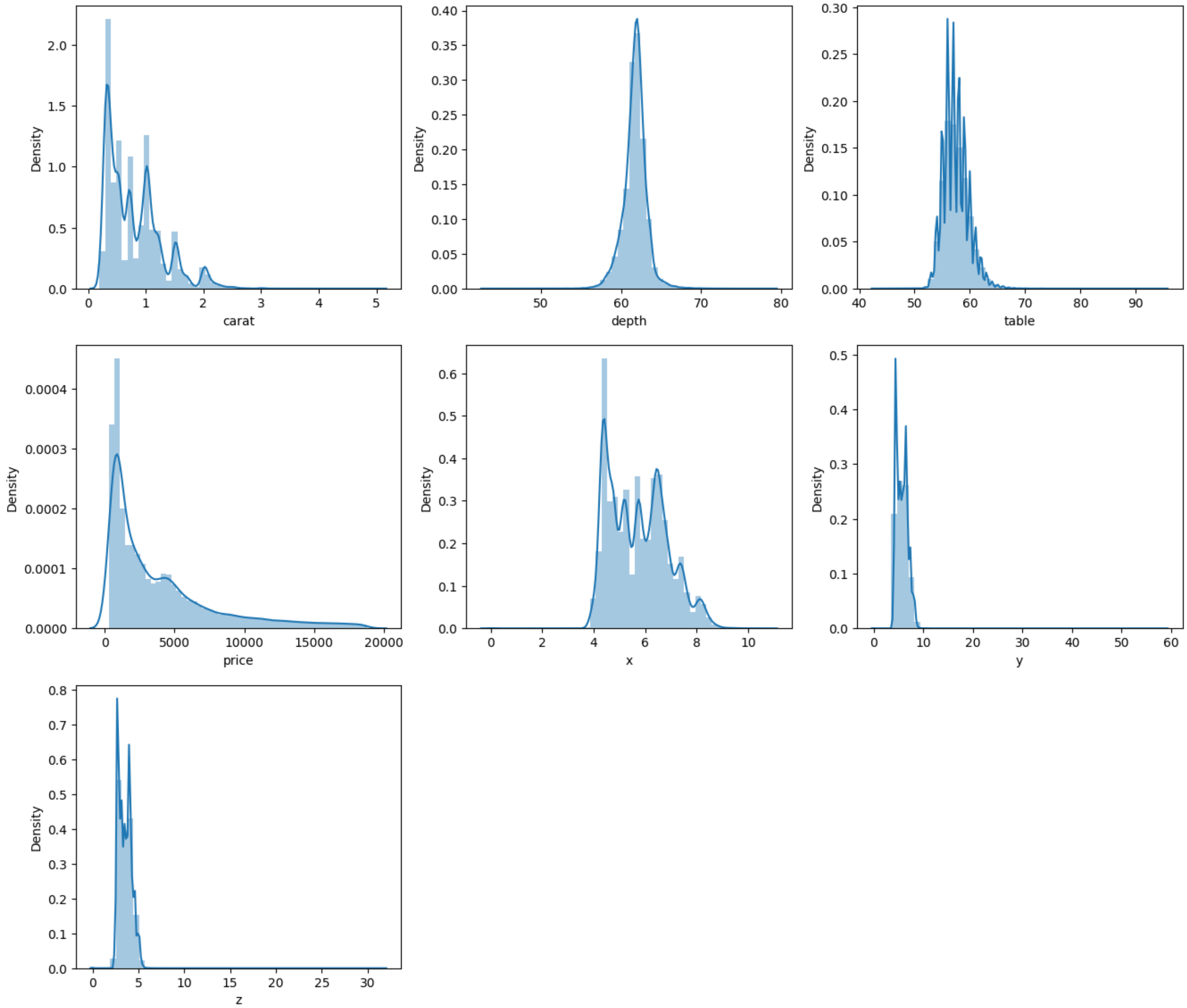
```python
categorical_cols = ['cut', 'color', 'clarity']

plt.figure(figsize=(15,12))
for i,col in enumerate(categorical_cols):
    plt.subplot(2, 2, i+1)
    ax = sns.countplot(data=df, x=categorical_cols[i], order= df[categorical_cols[i]].va

    for p in ax.patches:
       ax.annotate(p.get_height(), (p.get_x() * 1.005, p.get_height() * 1.005))
```



Inferences:

- Ideal cut diamonds more in numbers while fair cut is the least one
- The worst color 'J' is the rare one, however the bad colors 'G' and 'E' are more in numbers.
- Dimaonds with the best clairty 'IF' and worst clarity 'I1' are less in numbers while 'SI1' and 'VS2' have more number of diamonds.

In [6]:
```python
#plotting distirbution of numerical features
num_cols = ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']

plt.figure(figsize=(16,14))
for i, cols in enumerate(num_cols):
    plt.subplot(3,3,i+1)
    #sns.histplot(df[num_cols[i]], kde=True, stat='density', kde_kws=dict(cut=3))
    sns.distplot(df[num_cols[i]])
```



In [7]:
```
df.describe().T
```

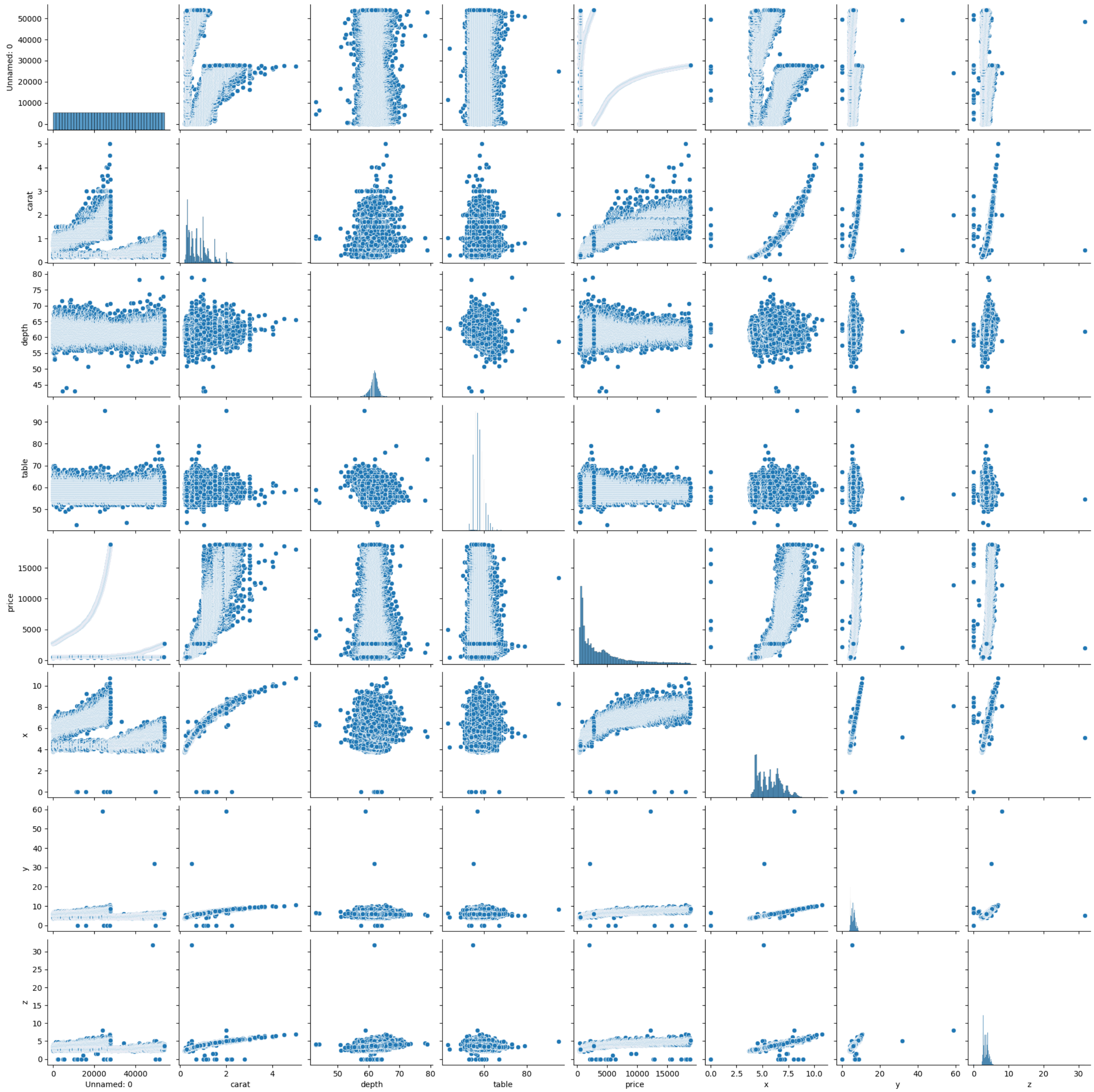
Out[7]:

	count	mean	std	min	25%	50%	75%	max
Unnamed: 0	53940.0	26970.500000	15571.281097	1.0	13485.75	26970.50	40455.25	53940.00
carat	53940.0	0.797940	0.474011	0.2	0.40	0.70	1.04	5.01
depth	53940.0	61.749405	1.432621	43.0	61.00	61.80	62.50	79.00
table	53940.0	57.457184	2.234491	43.0	56.00	57.00	59.00	95.00
price	53940.0	3932.799722	3989.439738	326.0	950.00	2401.00	5324.25	18823.00
x	53940.0	5.731157	1.121761	0.0	4.71	5.70	6.54	10.74
y	53940.0	5.734526	1.142135	0.0	4.72	5.71	6.54	58.90
z	53940.0	3.538734	0.705699	0.0	2.91	3.53	4.04	31.80

- Min values of x, y and z = 0 implies that there are errors in data (2D diamonds are meaningless)
- Price follows a right skewed distribution

In [8]:
```
#pair plot
sns.pairplot(df)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x27d3204bd90>
```



- There is an irrelevant column 'Unamed: 0'
- colums x,y and z seems to have outliers

Data preprocessing

In [9]:
```
#dropping unwanted columns
df = df.drop('Unnamed: 0', axis=1)
```

In [10]:
```
#checking for null values
df.isnull().sum()
```

Out[10]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
```

```
price      0
x          0
y          0
z          0
dtype: int64
```

In [11]:
```
#dropping datapoints with x, y and z have min value 0
df = df.drop(df[df['x']==0].index)
df = df.drop(df[df['y']==0].index)
df = df.drop(df[df['z']==0].index)
```

In [12]:
```
df.shape
```

Out[12]:
```
(53920, 10)
```

In [13]:
```
#removing outliers
df = df[(df["depth"]<75)&(df["depth"]>45)]
df = df[(df["table"]<80)&(df["table"]>40)]
df = df[(df["x"]<40)]
df = df[(df["y"]<40)]
df = df[(df["z"]<40)&(df["z"]>2)]
df.shape
```

Out[13]:
```
(53909, 10)
```

encoding categorical variales

In [14]:
```
df.head()
```

Out[14]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

In [15]:
```
df.cut.unique()
```

Out[15]:
```
array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)
```

In [16]:
```
df.color.unique()
```

Out[16]:
```
array(['E', 'I', 'J', 'H', 'F', 'G', 'D'], dtype=object)
```

In [17]:
```
df.clarity.unique()
```

Out[17]:
```
array(['SI2', 'SI1', 'VS1', 'VS2', 'VVS2', 'VVS1', 'I1', 'IF'],
      dtype=object)
```

In [18]:
```
cut_map = {'Fair':0, 'Good':1, 'Very Good':2, 'Premium':3, 'Ideal':4}
color_map = {'J':0, 'I':1, 'H':2, 'G':3, 'F':4, 'E':5, 'D':6}
clarity_map = {'SI2':0, 'SI1':1, 'VS1':2, 'VS2':3, 'VVS2':4, 'VVS1':5, 'I1':6, 'IF':7}

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

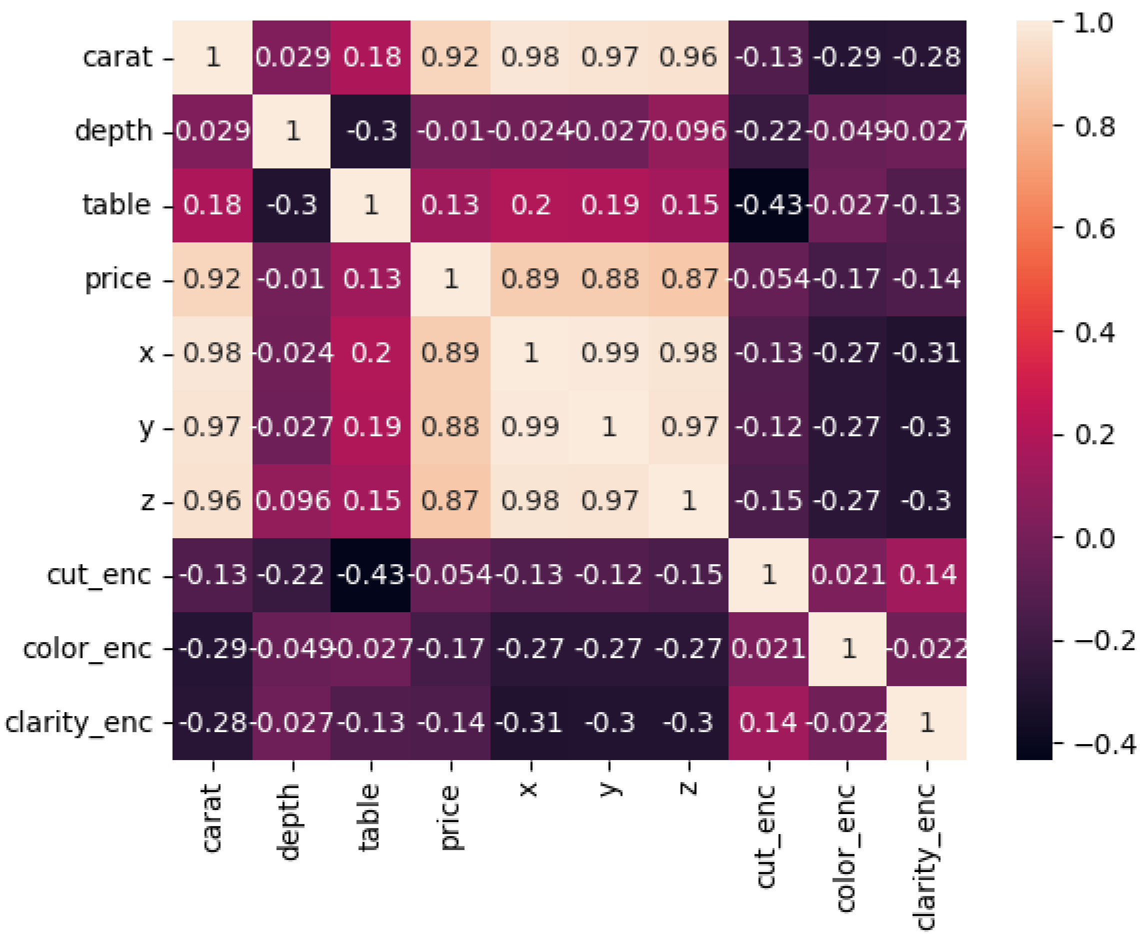
In [19]:
```python
df.head()
```

Out[19]:

	carat	cut	color	clarity	depth	table	price	x	y	z	cut_enc	color_enc	clarity_enc
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	4	5	0
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	3	5	1
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	1	5	2
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	3	1	3
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	1	0	0

In [20]:
```python
#correlation matrix
sns.heatmap(df.corr(), annot=True)
```

Out[20]:
```
<AxesSubplot: >
```



- price has strong positive correlations with carat, x, y and z
- carat, x, y and z columns have multicollenearity

In [21]:
```python
#Creating feature matrix and target vector
X = df[['carat', 'depth', 'table', 'cut_enc', 'color_enc', 'clarity_enc']]
y = df['price']
```

In [22]:
```python
X.shape
```

Out[22]:
```
(53909, 6)
```

In [23]:
```python
#splittng dataset into test and train splits
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Model training

1. Polynomial regression with degree = 2

In [24]:
```
#poly regression with degree = 2
poly_reg = PolynomialFeatures(degree=2)
poly_reg = poly_reg.fit(X_train)
X_train_poly = poly_reg.transform(X_train)
X_test_poly = poly_reg.transform(X_test)
```

In [25]:
```
X_train_poly.shape
```

Out[25]:
```
(37736, 28)
```

In [26]:
```
lr = LinearRegression()
lr.fit(X_train_poly, y_train)
```

Out[26]:
```
▾ LinearRegression
LinearRegression()
```

In [27]:
```
#model evaluation
y_pred = lr.predict(X_test_poly)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print('R2 score: ', r2)
print('Mean absolute: ', mae)
print('Mean squared error: ', mse)
print('Root mean squared error: ', rmse)
```

```
R2 score:  0.9055883760351068
Mean absolute:  739.7424590347676
Mean squared error:  1456273.0988075286
Root mean squared error:  1206.761409230312
```

2. Multiple Linear regression

In [30]:
```
mlr = LinearRegression()
mlr.fit(X_train, y_train)
```

Out[30]:
```
▾ LinearRegression
LinearRegression()
```

In [31]:
```
#model evaluation
y_pred = mlr.predict(X_test)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print('R2 score: ', r2)
print('Mean absolute error: ', mae)
```

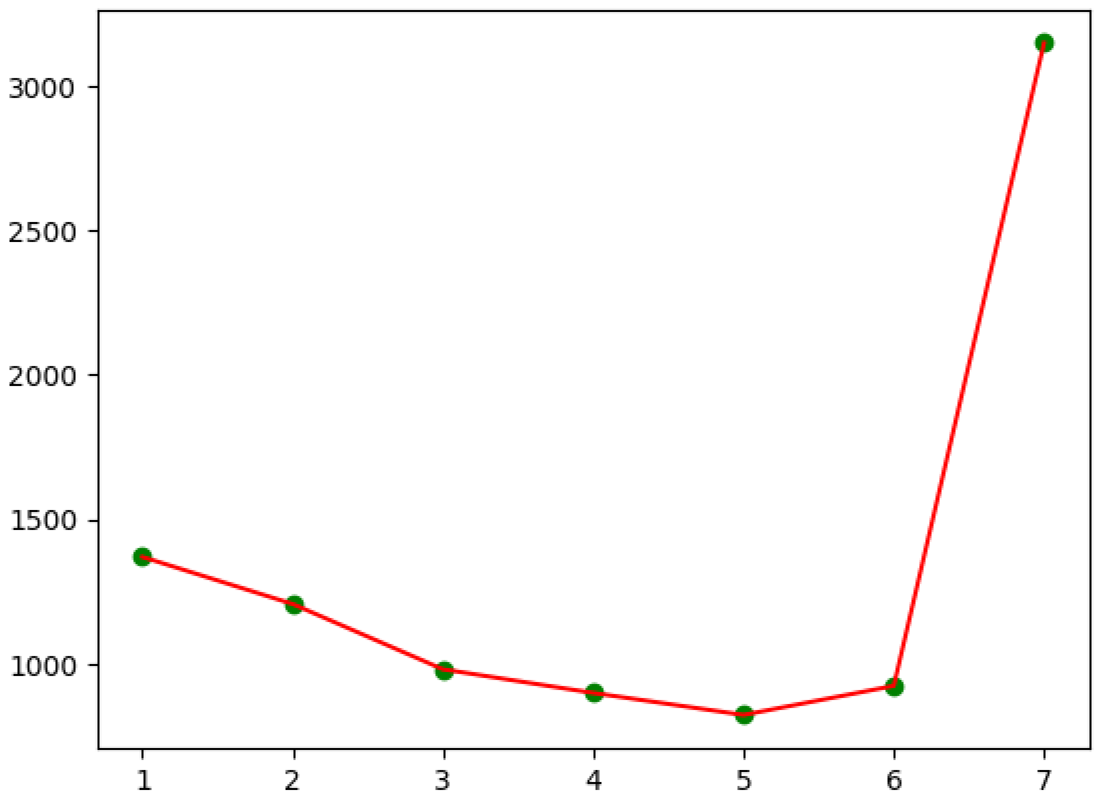
```
print('Mean squared error: ', mse)
print('Root mean squared error: ', rmse)
```

```
R2 score:  0.8783722897802407
Mean absolute error:  884.8190974160511
Mean squared error:  1876073.6763563755
Root mean squared error:  1369.6983888274
```

Finding best value for degree

In [32]:
```
number_degrees = [1,2,3,4,5,6,7]
plt_mean_squared_error = []
for degree in number_degrees:

   poly_model = PolynomialFeatures(degree=degree)

   poly_model = poly_model.fit(X_train)
   poly_X_train = poly_model.transform(X_train)
   poly_X_test = poly_model.transform(X_test)

   regression_model = LinearRegression()
   regression_model.fit(poly_X_train, y_train)
   y_pred = regression_model.predict(poly_X_test)

   plt_mean_squared_error.append(mean_squared_error(y_test, y_pred, squared=False))

plt.scatter(number_degrees,plt_mean_squared_error, color="green")
plt.plot(number_degrees,plt_mean_squared_error, color="red")
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x27d42307cd0>]
```



3. Polynomial regressoin with degree = 5

In [ ]:
```
```

In [33]:
```
#poly regression with degree = 5
```

```python
poly_reg = PolynomialFeatures(degree=5)
poly_reg = poly_reg.fit(X_train)
X_train_poly = poly_reg.transform(X_train)
X_test_poly = poly_reg.transform(X_test)

plr = LinearRegression()
plr.fit(X_train_poly, y_train)

#model evaluation
y_pred = plr.predict(X_test_poly)

r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print('R2 score: ', r2)
print('Mean absolute error: ', mae)
print('Mean squared error: ', mse)
print('Root mean squared error: ', rmse)
```

```
R2 score:  0.9561292323588778
Mean absolute error:  468.2426571275253
Mean squared error:  676694.415971052
Root mean squared error:  822.6143786556688
```

Conclusion

Polynomial regression with degree = 5 gives the best results

In [ ]:

```
!jupyter nbconvert --to webpdf --allow-chromium-download diamond_price_prediction.ipynb
```

In [ ]: