diamond-price-prediction

December 5, 2023

1 ****PREDICTING DIAMOND PRICE****:

In this data science project, i will develop a predictive model to estimate the price of diamonds based on their characteristics. Diamonds are not only precious gemstones but also carry a wide range of attributes that influence their value, such as carat weight, cut quality, color, and clarity. By creating a predictive model, it can help diamond buyers and sellers make more informed decisions and gain insights into the factors that drive diamond prices.

LABELLED DIMENSIONS OF A DIAMOND

Introdution

this project focuses on preciously predicting diamond price using multiple regression models

Content

price :price in US dollars ($326–$18,823)

carat :weight of the diamond (0.2–5.01)

cut :quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color: diamond colour, from J (worst) to D (best)

clarity: a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x :length in mm (0–10.74)

y :width in mm (0–58.9)

z :depth in mm (0–31.8)

depth :total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43–79)

table :width of top of diamond relative to widest point (43–95)

#1.Importing libraries:

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import warnings
warnings.filterwarnings('ignore')
```

#2.Importing Dataset:

[3]:
```python
df=pd.read_csv('/content/diamonds.csv.zip')
df
```

[3]:
```
       Unnamed: 0  carat        cut color clarity  depth  table  price     x  \
0               1   0.23      Ideal     E     SI2   61.5   55.0    326  3.95
1               2   0.21    Premium     E     SI1   59.8   61.0    326  3.89
2               3   0.23       Good     E     VS1   56.9   65.0    327  4.05
3               4   0.29    Premium     I     VS2   62.4   58.0    334  4.20
4               5   0.31       Good     J     SI2   63.3   58.0    335  4.34
…             …    …          …     …       …    …      …    …
53935       53936   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75
53936       53937   0.72       Good     D     SI1   63.1   55.0   2757  5.69
53937       53938   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66
53938       53939   0.86    Premium     H     SI2   61.0   58.0   2757  6.15
53939       53940   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83

          y     z
0      3.98  2.43
1      3.84  2.31
2      4.07  2.31
3      4.23  2.63
4      4.35  2.75
…       …    …
53935  5.76  3.50
53936  5.75  3.61
53937  5.68  3.56
53938  6.12  3.74
53939  5.87  3.64

[53940 rows x 11 columns]
```

#3.Exploratory Data Analysis :

[4]:
```python
df.head()
```

[4]:
```
   Unnamed: 0  carat      cut color clarity  depth  table  price     x     y  \
0           1   0.23    Ideal     E     SI2   61.5   55.0    326  3.95  3.98
1           2   0.21  Premium     E     SI1   59.8   61.0    326  3.89  3.84
2           3   0.23     Good     E     VS1   56.9   65.0    327  4.05  4.07
3           4   0.29  Premium     I     VS2   62.4   58.0    334  4.20  4.23
4           5   0.31     Good     J     SI2   63.3   58.0    335  4.34  4.35

      z
```

```
0    2.43
1    2.31
2    2.31
3    2.63
4    2.75
```

[5]:
```
df.tail()
```

[5]:
```
       Unnamed: 0  carat        cut color clarity  depth  table  price     x  \
53935       53936   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75
53936       53937   0.72       Good     D     SI1   63.1   55.0   2757  5.69
53937       53938   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66
53938       53939   0.86    Premium     H     SI2   61.0   58.0   2757  6.15
53939       53940   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83

          y     z
53935  5.76  3.50
53936  5.75  3.61
53937  5.68  3.56
53938  6.12  3.74
53939  5.87  3.64
```

[6]:
```
df.columns
```

[6]:
```
Index(['Unnamed: 0', 'carat', 'cut', 'color', 'clarity', 'depth', 'table',
       'price', 'x', 'y', 'z'],
      dtype='object')
```

[7]:
```
df.dtypes
```

[7]:
```
Unnamed: 0      int64
carat         float64
cut            object
color          object
clarity        object
depth         float64
table         float64
price           int64
x             float64
y             float64
z             float64
dtype: object
```

[8]:
```
df.shape
```

[8]:
```
(53940, 11)
```

[9]:
```python
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
```

The first column is an index ("Unnamed: 0") and thus we are going to remove it.

[10]:
```python
df1=df.drop('Unnamed: 0', axis = 1)
```

[11]:
```python
df1.head()
```

[11]:

	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

Checking for missing values & categorical variables

[12]:
```python
df1.isna().sum()
```

[12]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
```

```
dtype: int64
```

[13]:
```
df1.isna().sum()
```

[13]:
```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

[14]:
```
print(df1['cut'].value_counts(), '\n', df1['color'].value_counts(), '\n',␣
↪df1['clarity'].value_counts())
```

```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
 G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
J     2808
Name: color, dtype: int64
 SI1     13065
VS2     12258
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: clarity, dtype: int64
```

checking unique value count

[15]:
```
for column_name in df1.columns:
  print(df1[column_name].value_counts())
  print('unique_values_count = ',df[column_name].nunique(),'-'*80)
```

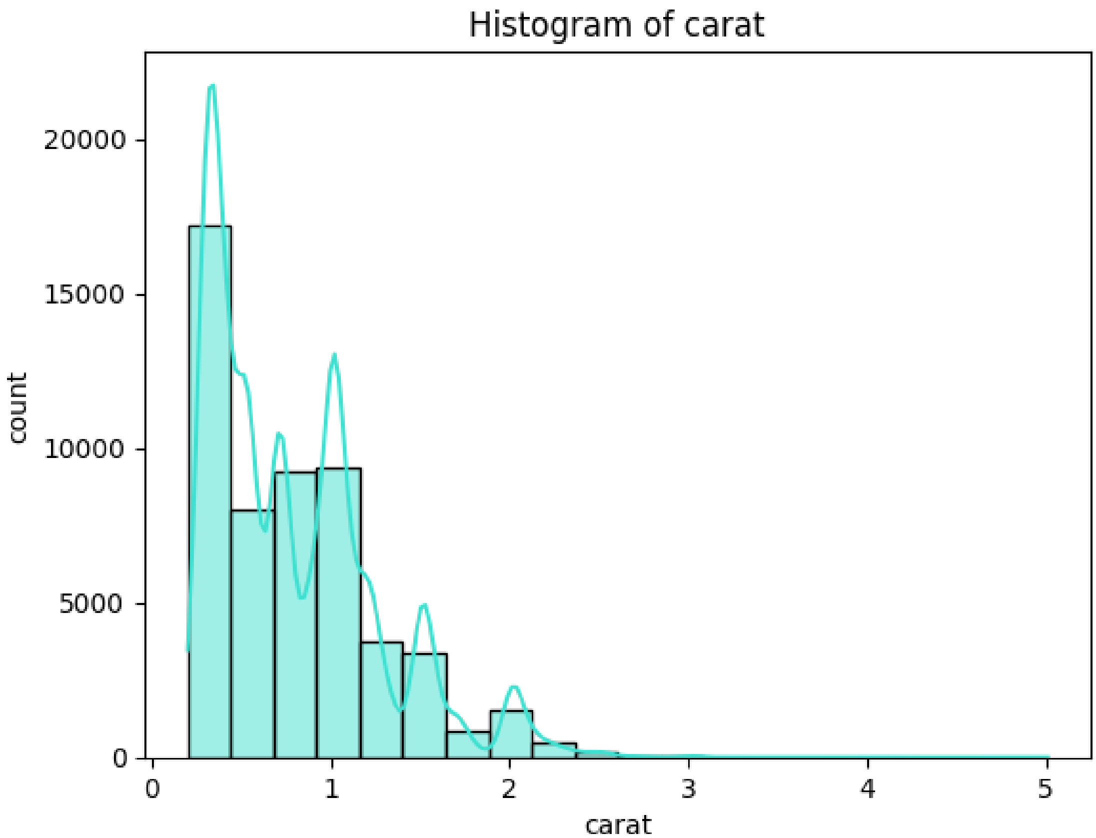
```
0.30    2604
0.31    2249
1.01    2242
0.70    1981
0.32    1840
        …
3.02       1
3.65       1
3.50       1
3.22       1
3.11       1
Name: carat, Length: 273, dtype: int64
unique_values_count =  273
--------------------------------------------------------------------------------
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
unique_values_count =  5
--------------------------------------------------------------------------------
G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
J     2808
Name: color, dtype: int64
unique_values_count =  7
--------------------------------------------------------------------------------
SI1     13065
VS2     12258
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: clarity, dtype: int64
unique_values_count =  8
--------------------------------------------------------------------------------
62.0    2239
61.9    2163
61.8    2077
62.2    2039
62.1    2020
```

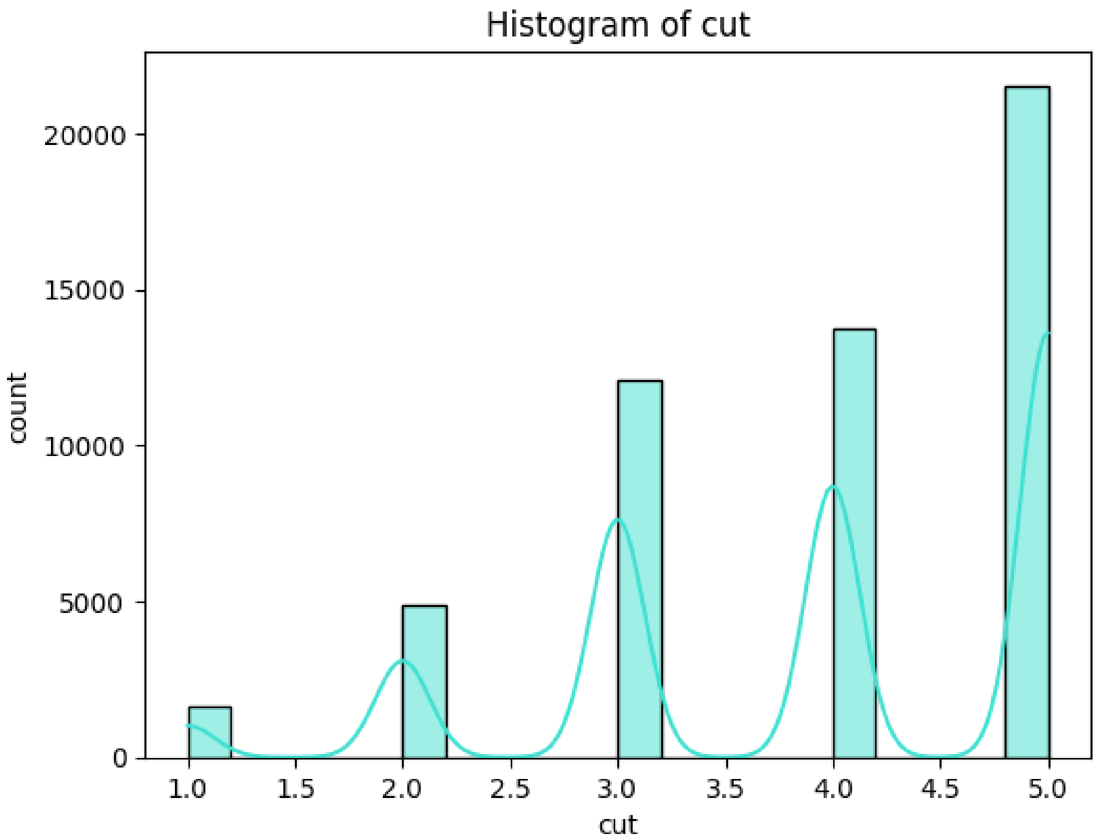
```
        …
71.3       1
44.0       1
53.0       1
53.1       1
54.7       1
Name: depth, Length: 184, dtype: int64
unique_values_count =  184
--------------------------------------------------------------------------------
56.0    9881
57.0    9724
58.0    8369
59.0    6572
55.0    6268
        …
51.6       1
63.5       1
43.0       1
62.4       1
61.6       1
Name: table, Length: 127, dtype: int64
unique_values_count =  127
--------------------------------------------------------------------------------
605      132
802      127
625      126
828      125
776      124
        …
8816       1
14704      1
14699      1
14698      1
9793       1
Name: price, Length: 11602, dtype: int64
unique_values_count =  11602
--------------------------------------------------------------------------------
4.37     448
4.34     437
4.33     429
4.38     428
4.32     425
        …
10.74      1
9.36       1
8.89       1
10.23      1
10.00      1
```

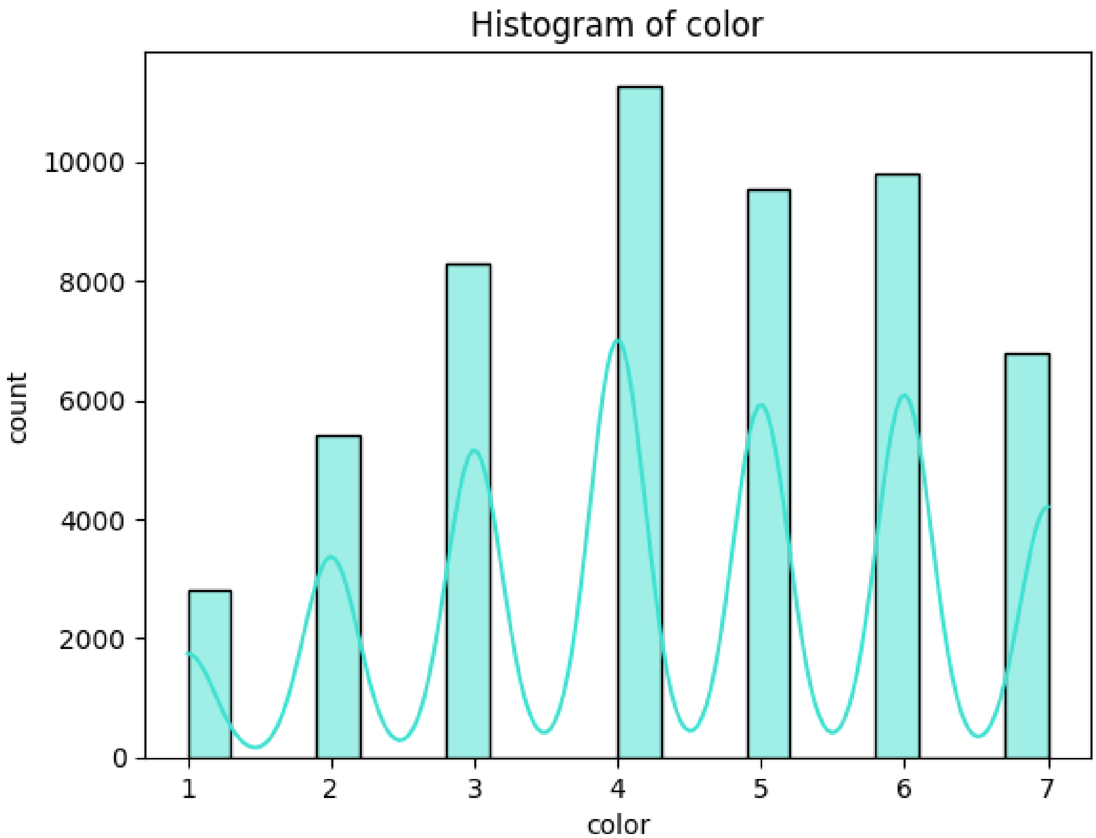
```
Name: x, Length: 554, dtype: int64
unique_values_count =  554
--------------------------------------------------------------------------------
4.34     437
4.37     435
4.35     425
4.33     421
4.32     414
        …
8.89       1
10.16      1
9.46       1
9.63       1
31.80      1
Name: y, Length: 552, dtype: int64
unique_values_count =  552
--------------------------------------------------------------------------------
2.70     767
2.69     748
2.71     738
2.68     730
2.72     697
        …
5.79       1
5.72       1
5.91       1
5.61       1
31.80      1
Name: z, Length: 375, dtype: int64
unique_values_count =  375
--------------------------------------------------------------------------------
```

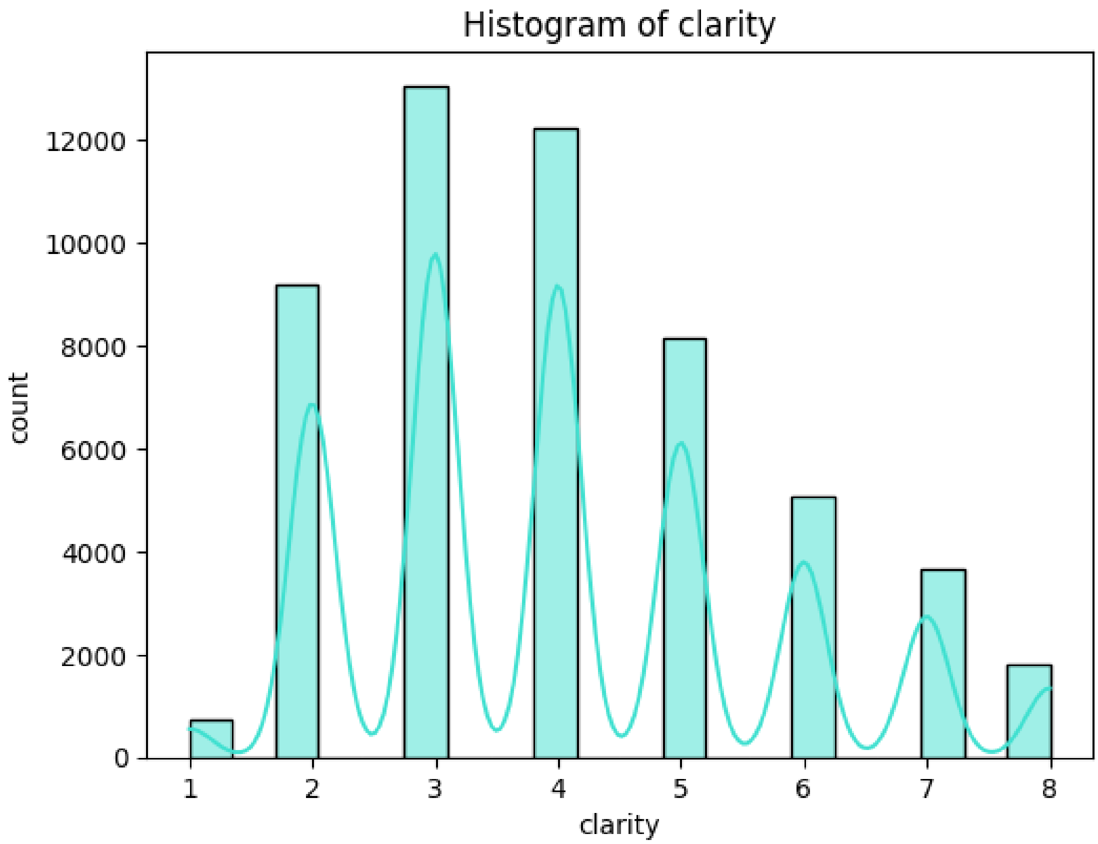
2 Data Visualization :

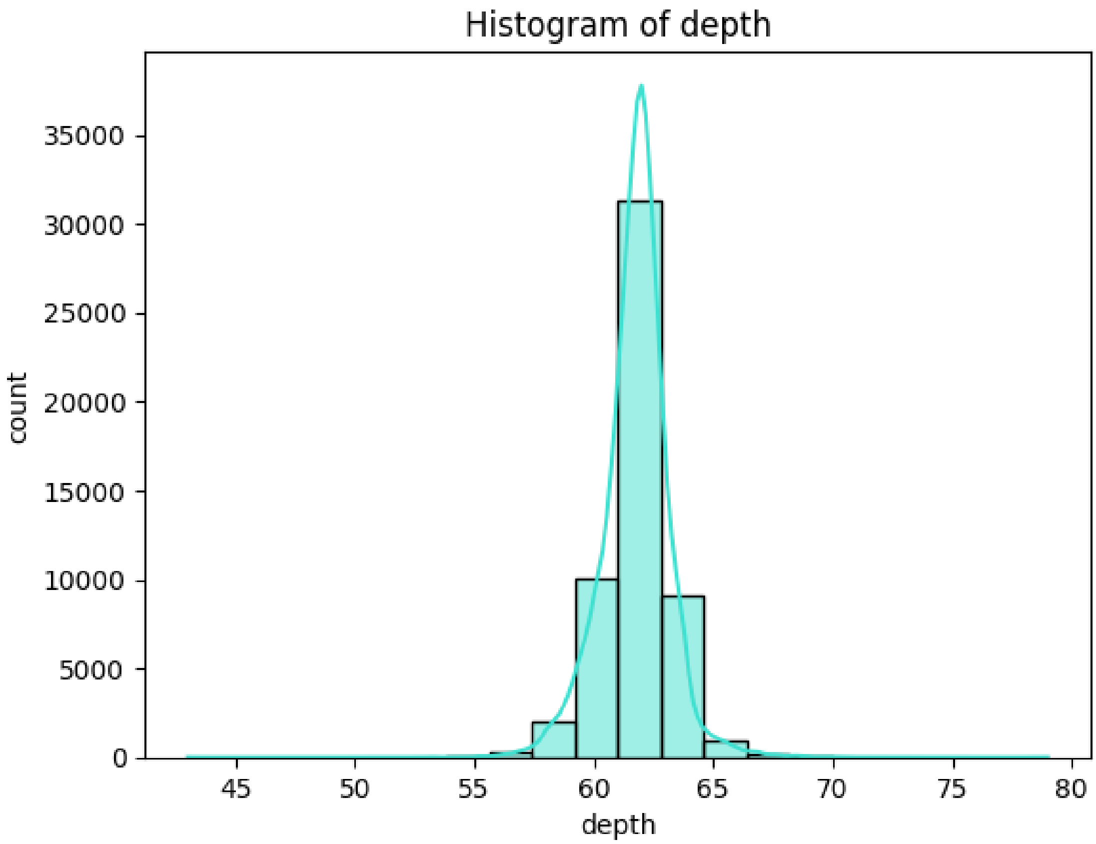
[73]:
```python
#histogram
lst1 = ['carat','cut','color','clarity','depth','table','price']

for i in lst1:
  sns.histplot(df1[i],color='turquoise', bins=20, kde=True)
  plt.xlabel(i)
  plt.ylabel('count')
  plt.title(f'Histogram of {i}')
  plt.show()
```


Histogram of carat
count
carat
0
5000
10000
15000
20000
0
1
2
3
4
5

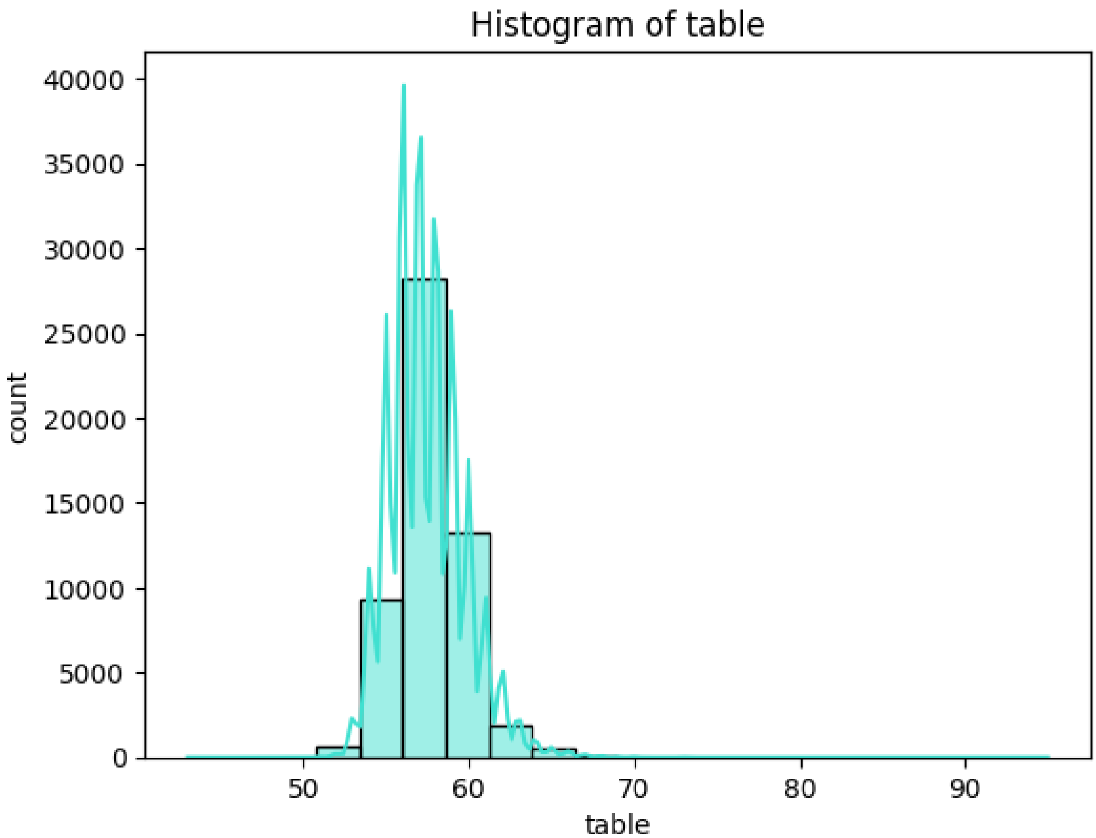

Histogram of cut
count
20000
15000
10000
5000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
cut

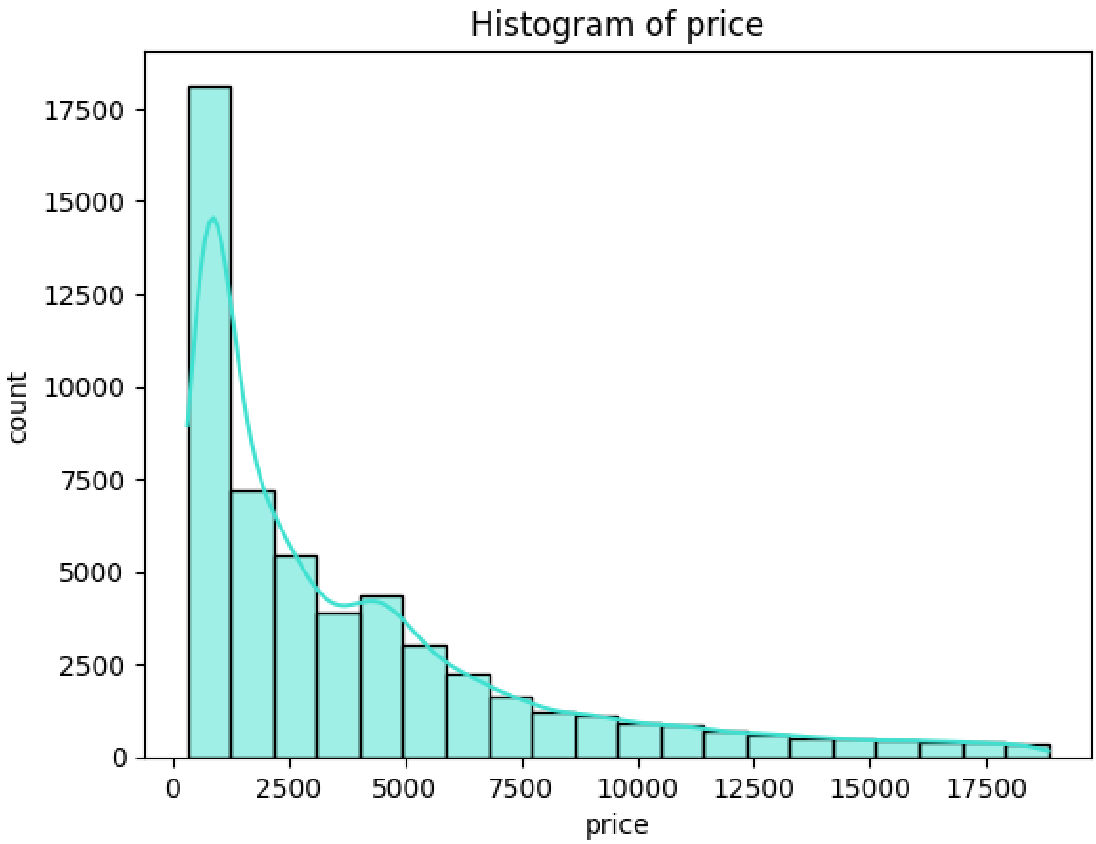

Histogram of color
count
color
0
2000
4000
6000
8000
10000
1
2
3
4
5
6
7


Histogram of clarity
count
clarity
0
2000
4000
6000
8000
10000
12000
1
2
3
4
5
6
7
8



Histogram of depth
count
35000
30000
25000
20000
15000
10000
5000
0
45
50
55
60
65
70
75
80
depth

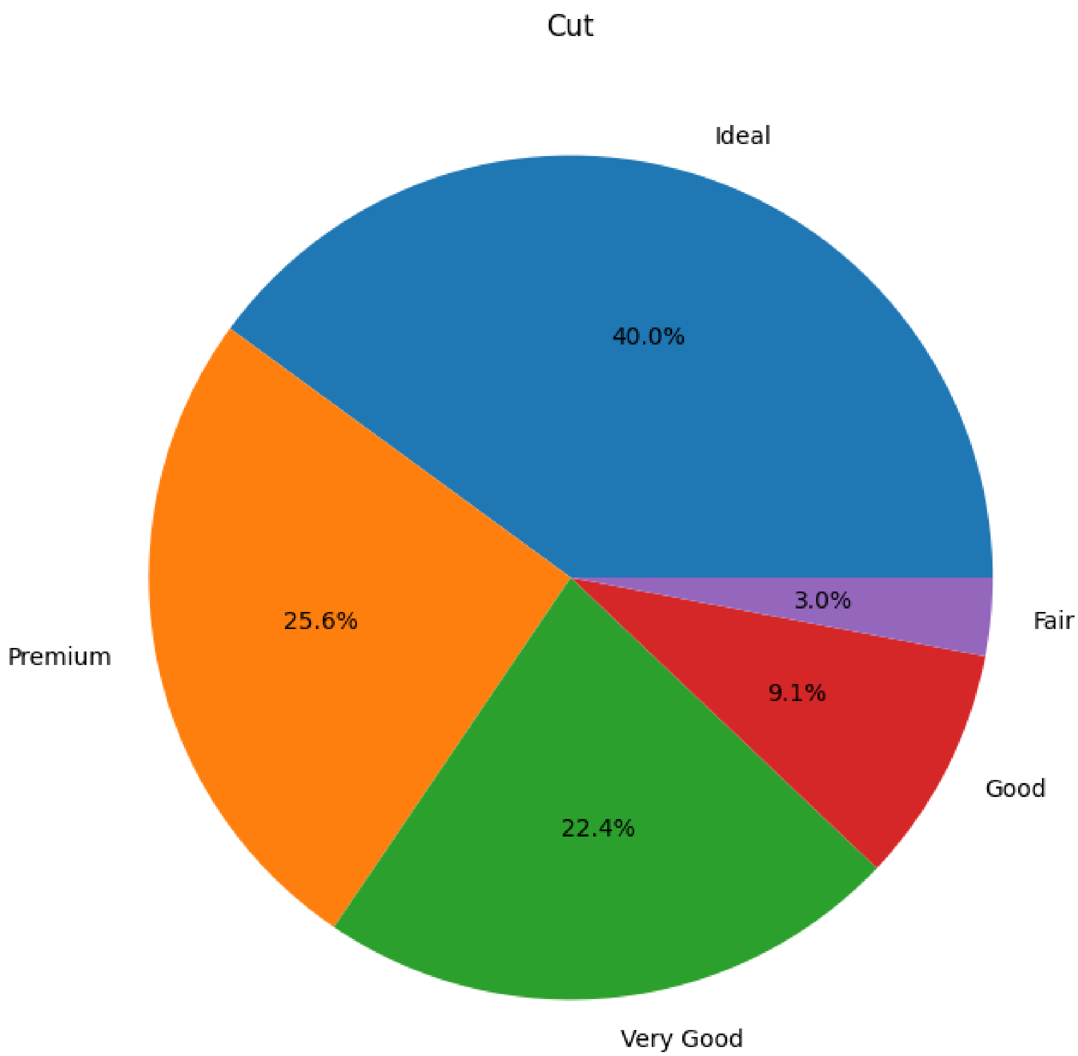

Histogram of table
40000
35000
30000
25000
20000
15000
10000
5000
0
count
50
60
70
80
90
table



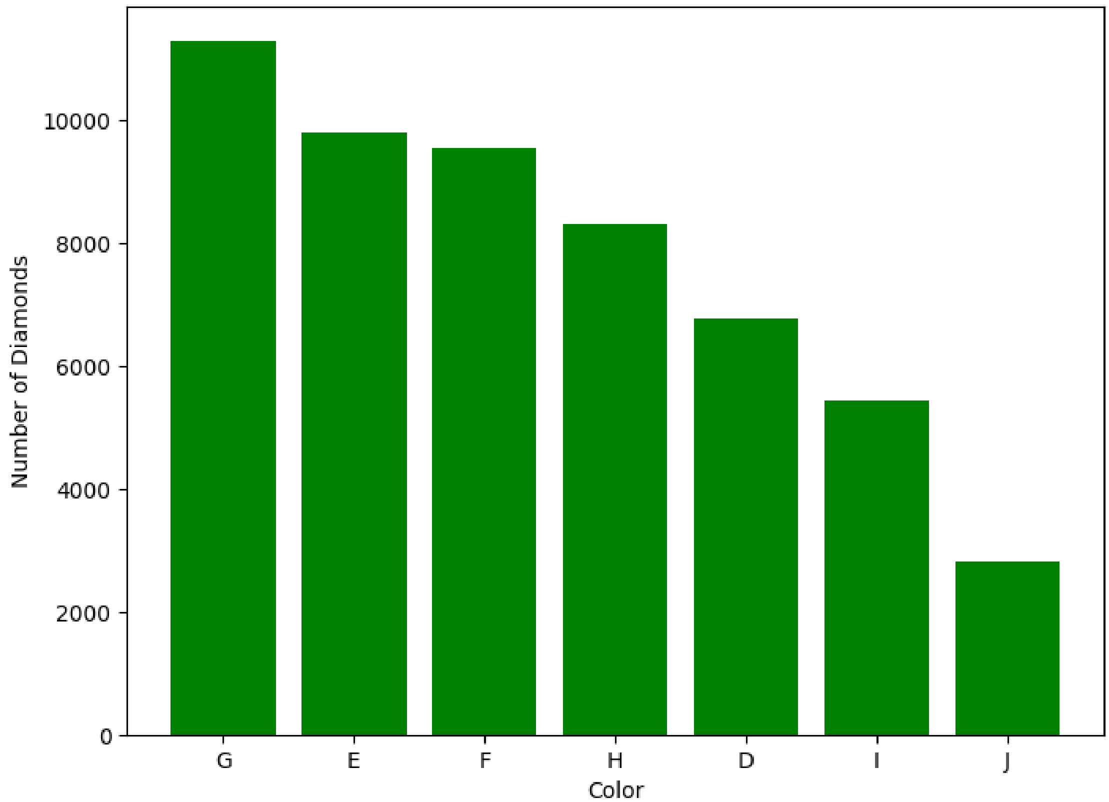
#**Single visulaization**

Single box plot is a powerful visualization for the "Predicting Diamond Price" project's EDA. It showcases how diamond prices differ across categories like "cut," "color," or "clarity." The plot displays median, quartiles, and outliers for each category, providing immediate insights into attribute impact on prices.

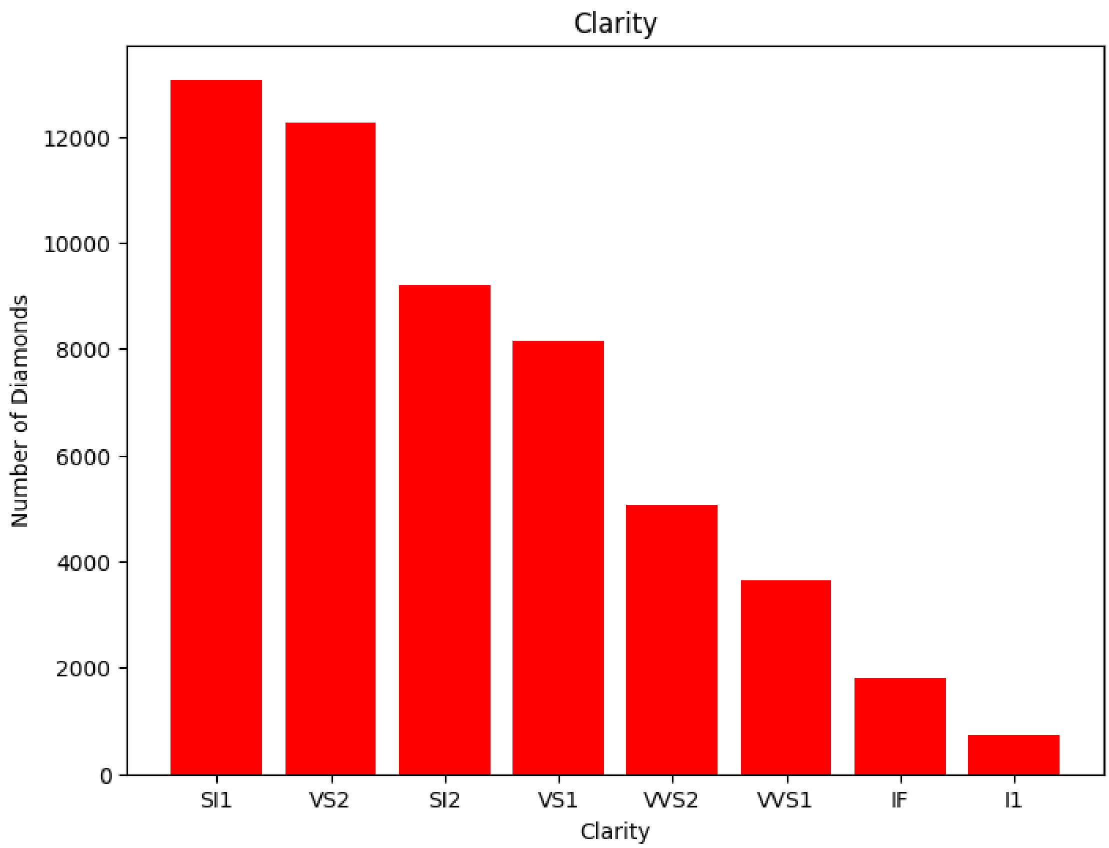
[17]:
```
plt.figure(figsize=(8, 8))
plt.pie(df1['cut'].value_counts(),labels=['Ideal','Premium','Very␣
↪Good','Good','Fair'],autopct='%1.1f%%')
plt.title('Cut')
plt.show()
```



[18]:
```
plt.figure(figsize=(8,6))
plt.bar(df1['color'].value_counts().index, df1['color'].
  ↪value_counts(),color='green')
plt.ylabel("Number of Diamonds")
plt.xlabel("Color")
plt.show()
```

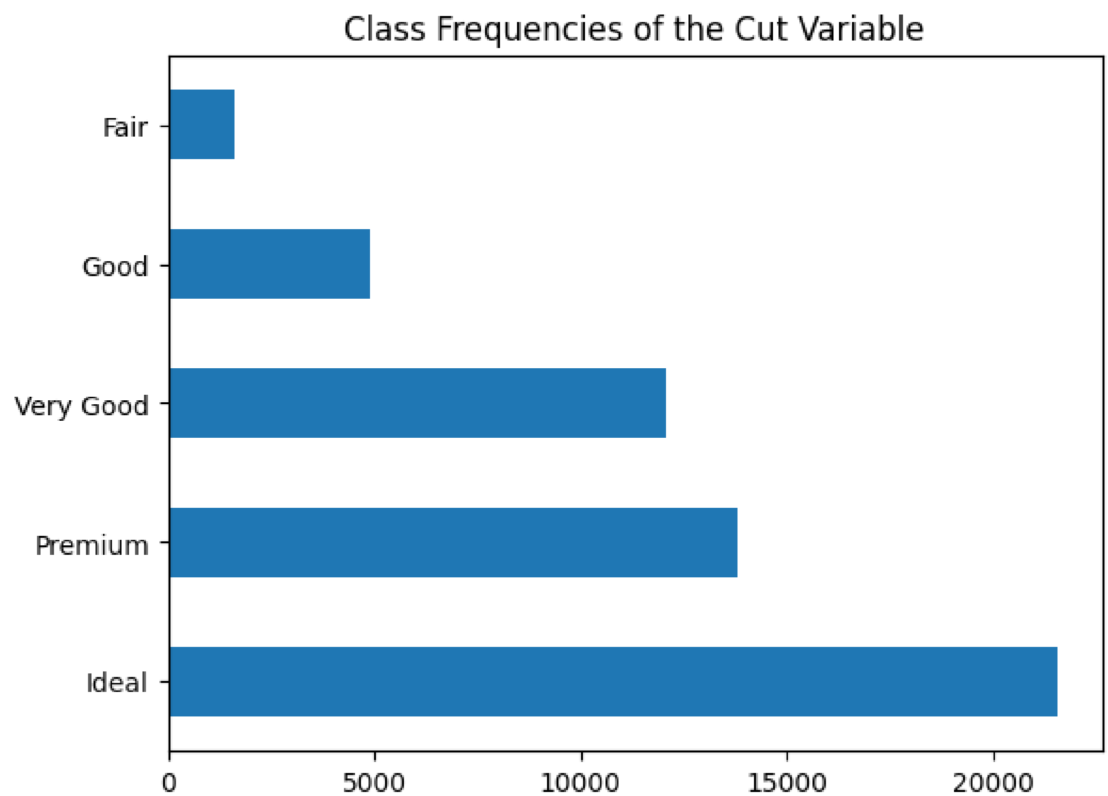


[19]:
```
plt.figure(figsize=(8,6))
plt.bar(df1['clarity'].value_counts().index, df1['clarity'].
 ↪value_counts(),color='red')
plt.title('Clarity')
plt.ylabel("Number of Diamonds")
plt.xlabel("Clarity")
plt.show()
```



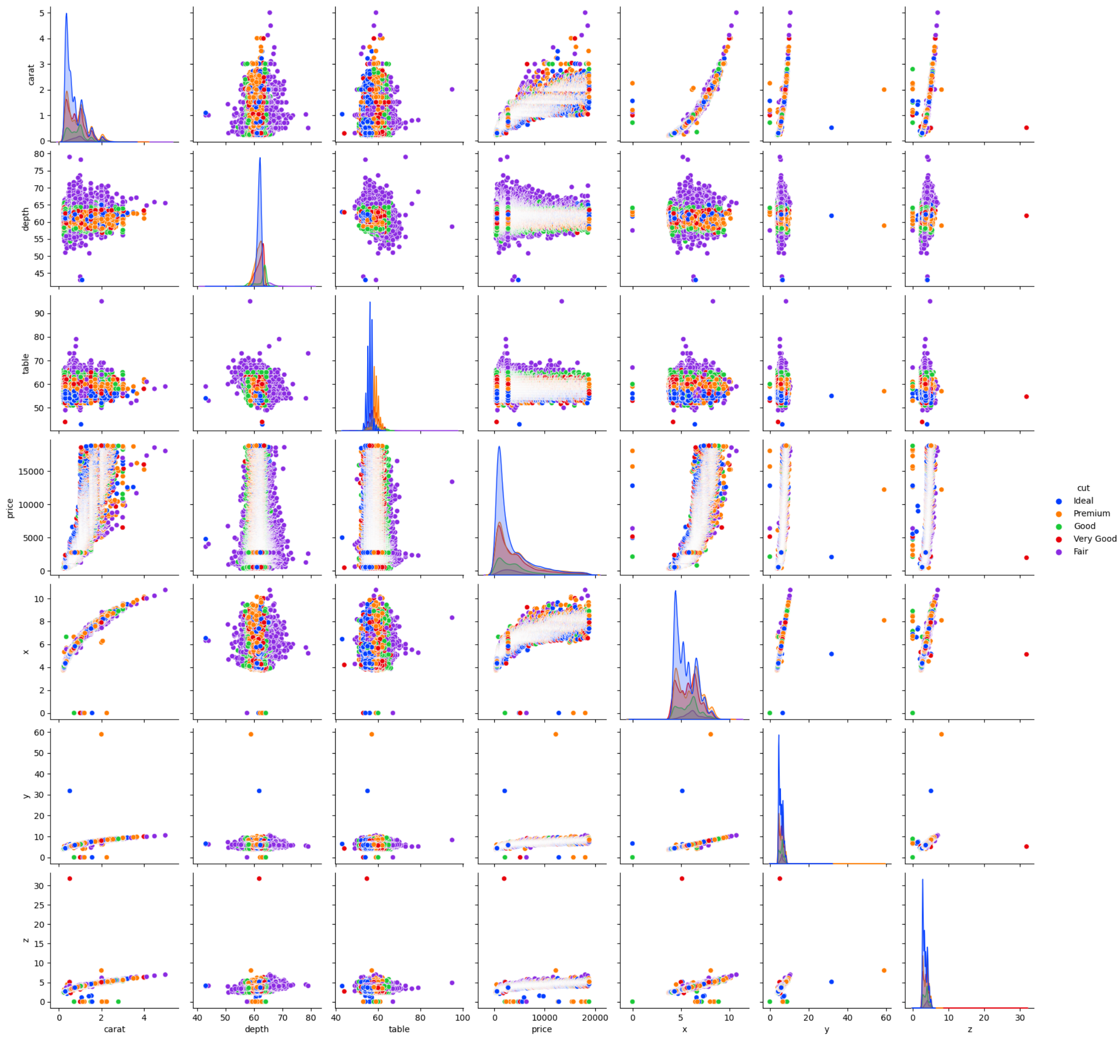
[20]:
```
df1["cut"].value_counts().plot.barh().set_title("Class Frequencies of the Cut␣
↪Variable")
```

[20]: Text(0.5, 1.0, 'Class Frequencies of the Cut Variable')



[21]:
```
sns.pairplot(df1,hue='cut',palette='bright')
```

[21]: <seaborn.axisgrid.PairGrid at 0x7a4b5d1f1c30>

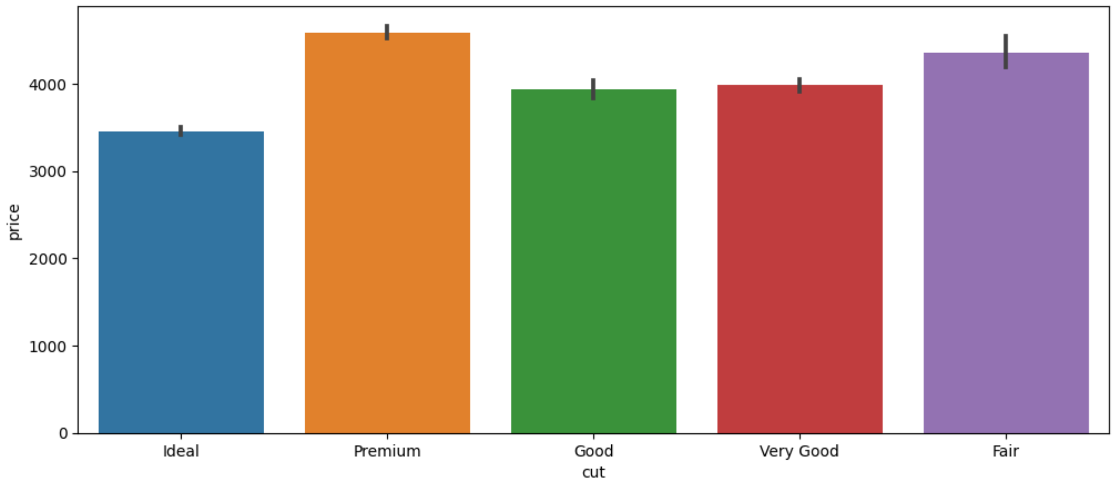


#Grouped visualization

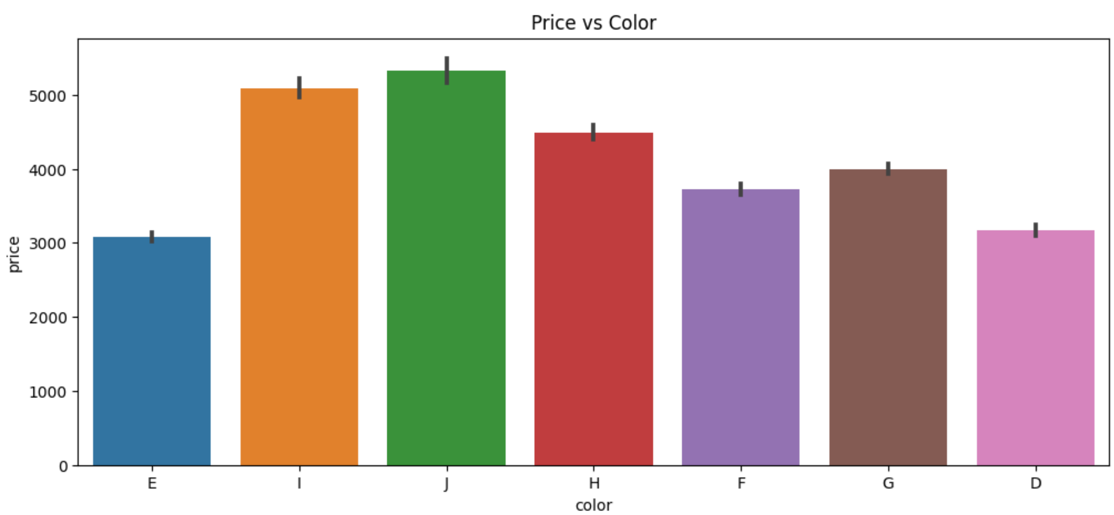
Grouped visualizations in the “Predicting Diamond Price” project’s EDA unveil attribute interactions within categorical categories like “cut,” “color,” and “clarity.” These visualizations, including bar charts,box plots ,scatter plots with hues, and more, offer insights into how attributes influence diamond prices across distinct groups. They provide a nuanced understanding of relationships, aiding data preprocessing and model construction decisions.

[22]:
```
plt.figure(figsize = (12, 5))
sns.barplot(x='cut',y='price',data = df1)
```

[22]: <Axes: xlabel='cut', ylabel='price'>

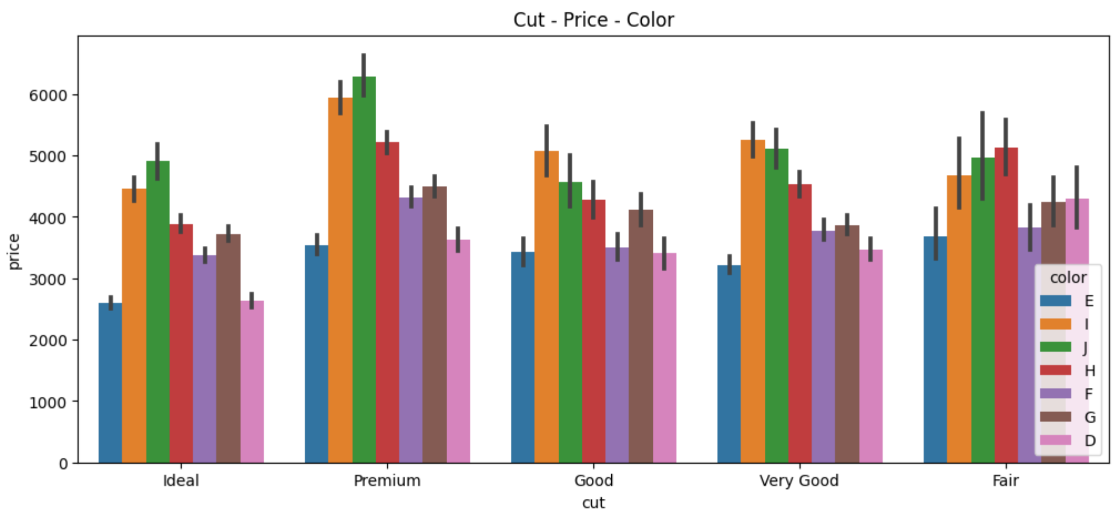


[23]:
```
plt.figure(figsize = (12, 5))
sns.barplot(x='color',y='price',data = df1)
plt.title('Price vs Color')
plt.show()
```



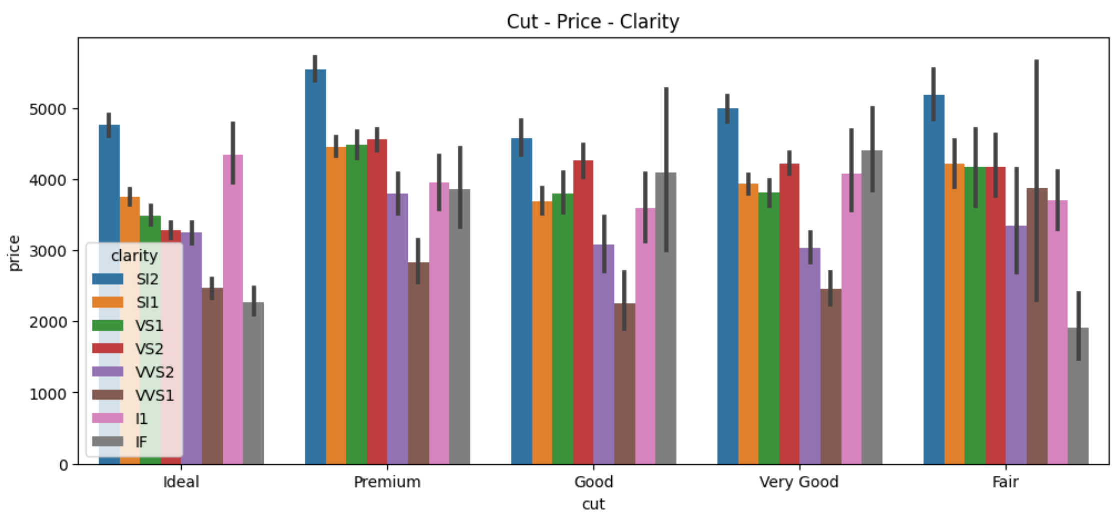
[24]:
```
plt.figure(figsize = (12, 5))
sns.barplot(x="cut",y="price",hue="color",data=df1)
plt.title("Cut - Price - Color")
```

[24]:
```
Text(0.5, 1.0, 'Cut - Price - Color')
```



[25]:
```
plt.figure(figsize = (12, 5))
sns.barplot(x="cut",
            y="price",
            hue="clarity",
            data = df1)
plt.title("Cut - Price - Clarity")
```

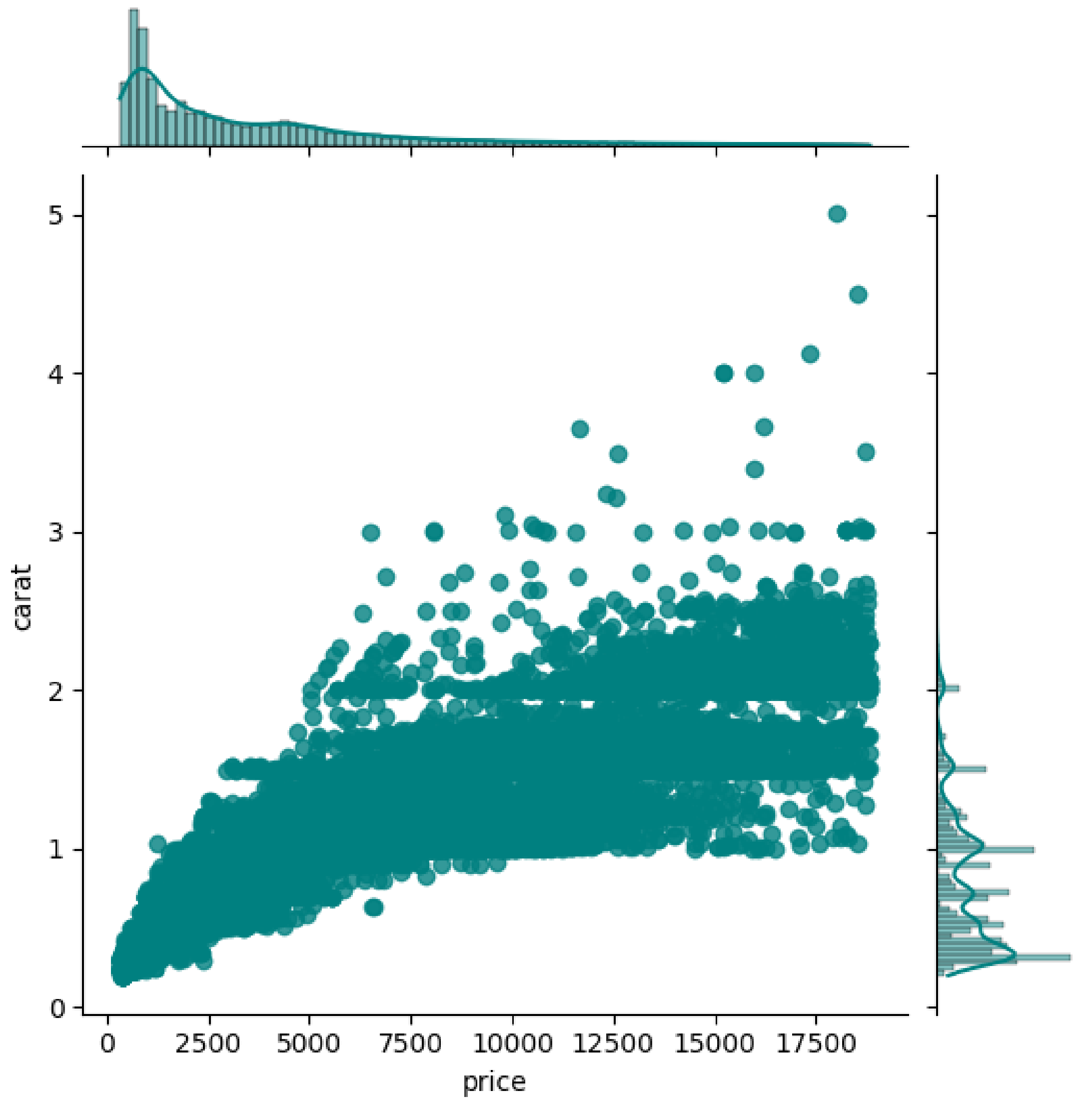
[25]: Text(0.5, 1.0, 'Cut - Price - Clarity')

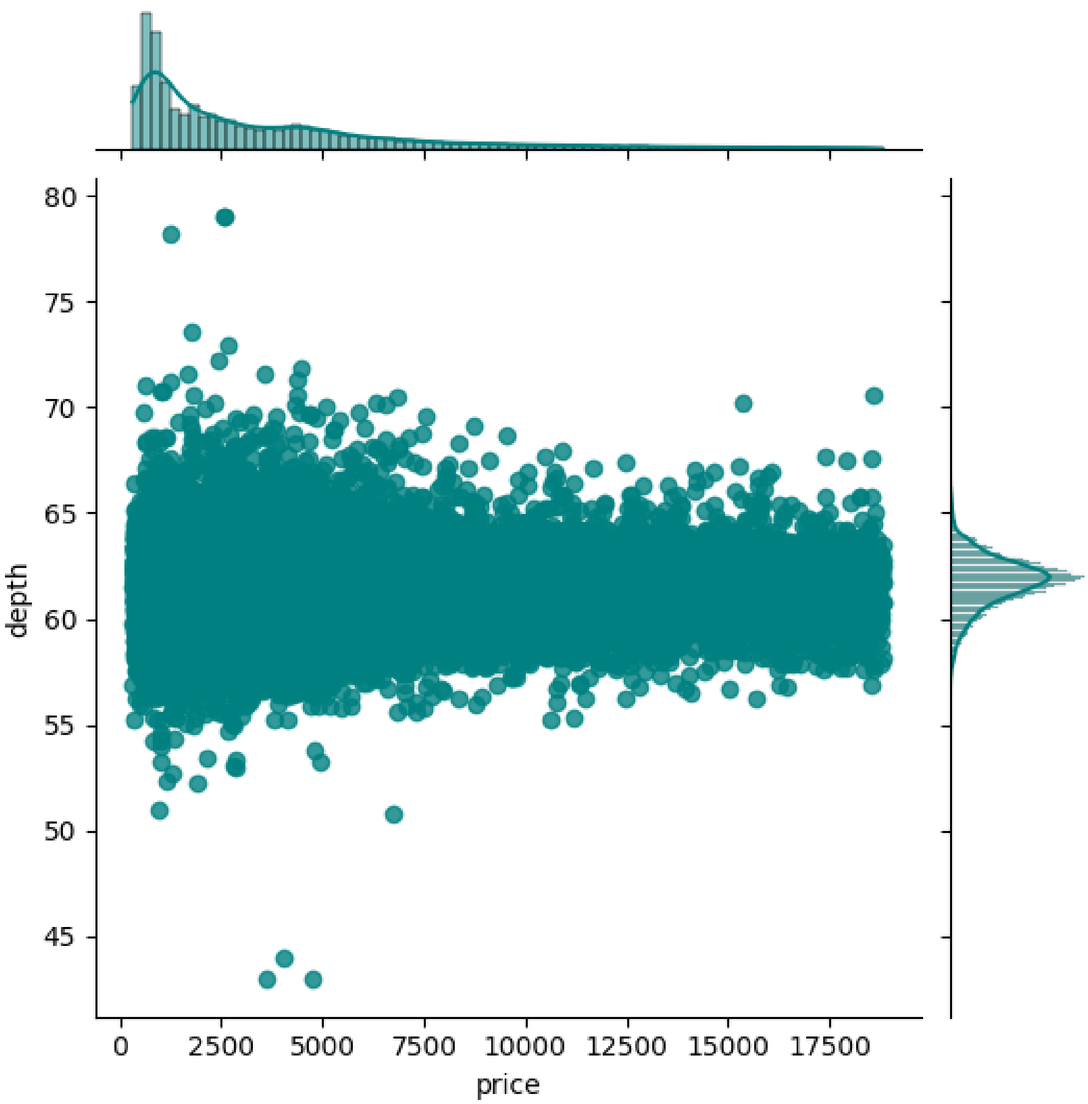


[72]:
```
lst2=['carat','depth','table']
for i in lst2:
  sns.jointplot(x = "price",y =i,data = df1,kind='reg',color='teal')
```

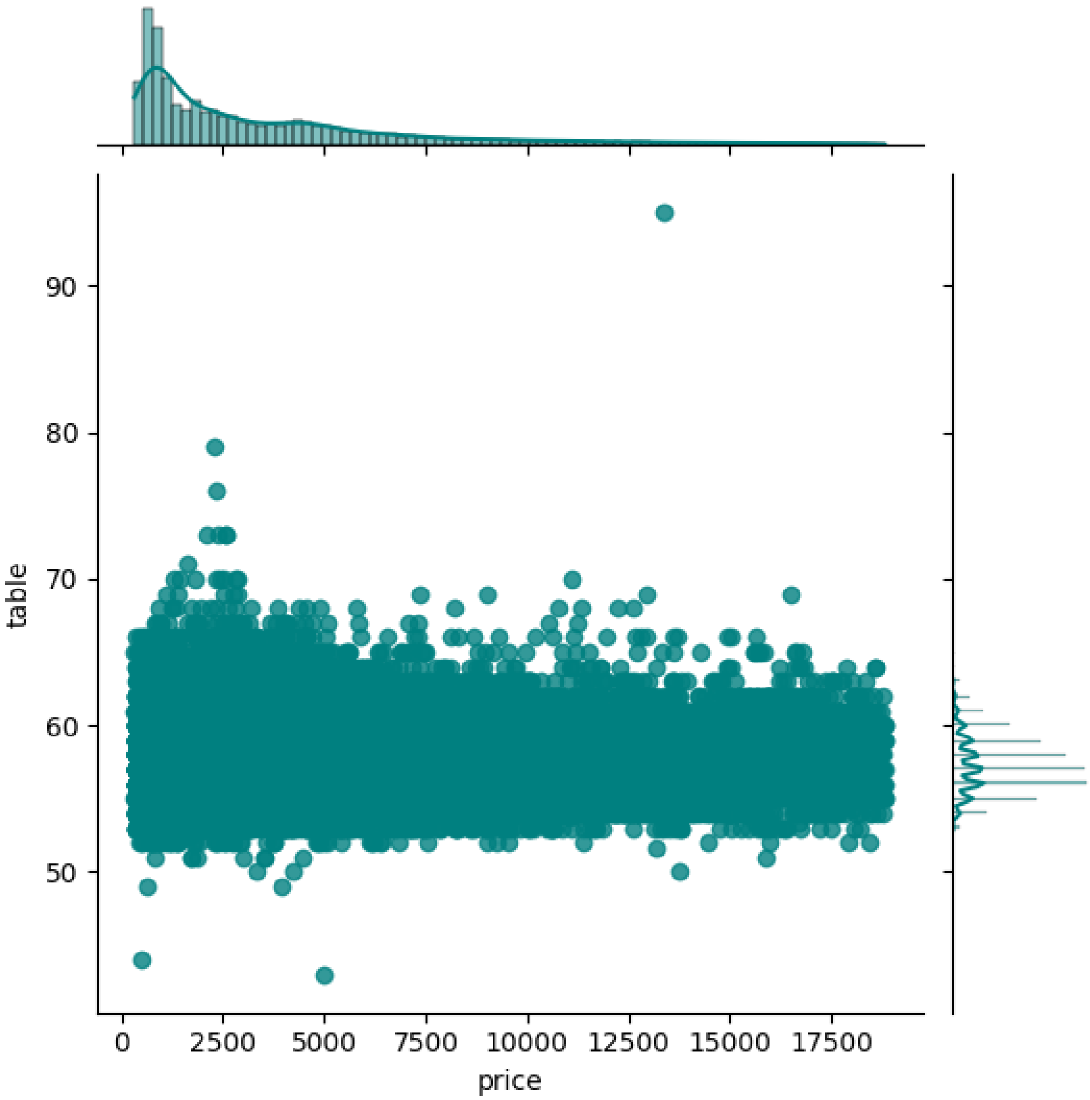
```
plt.show
```

[72]: <function matplotlib.pyplot.show(close=None, block=None)>



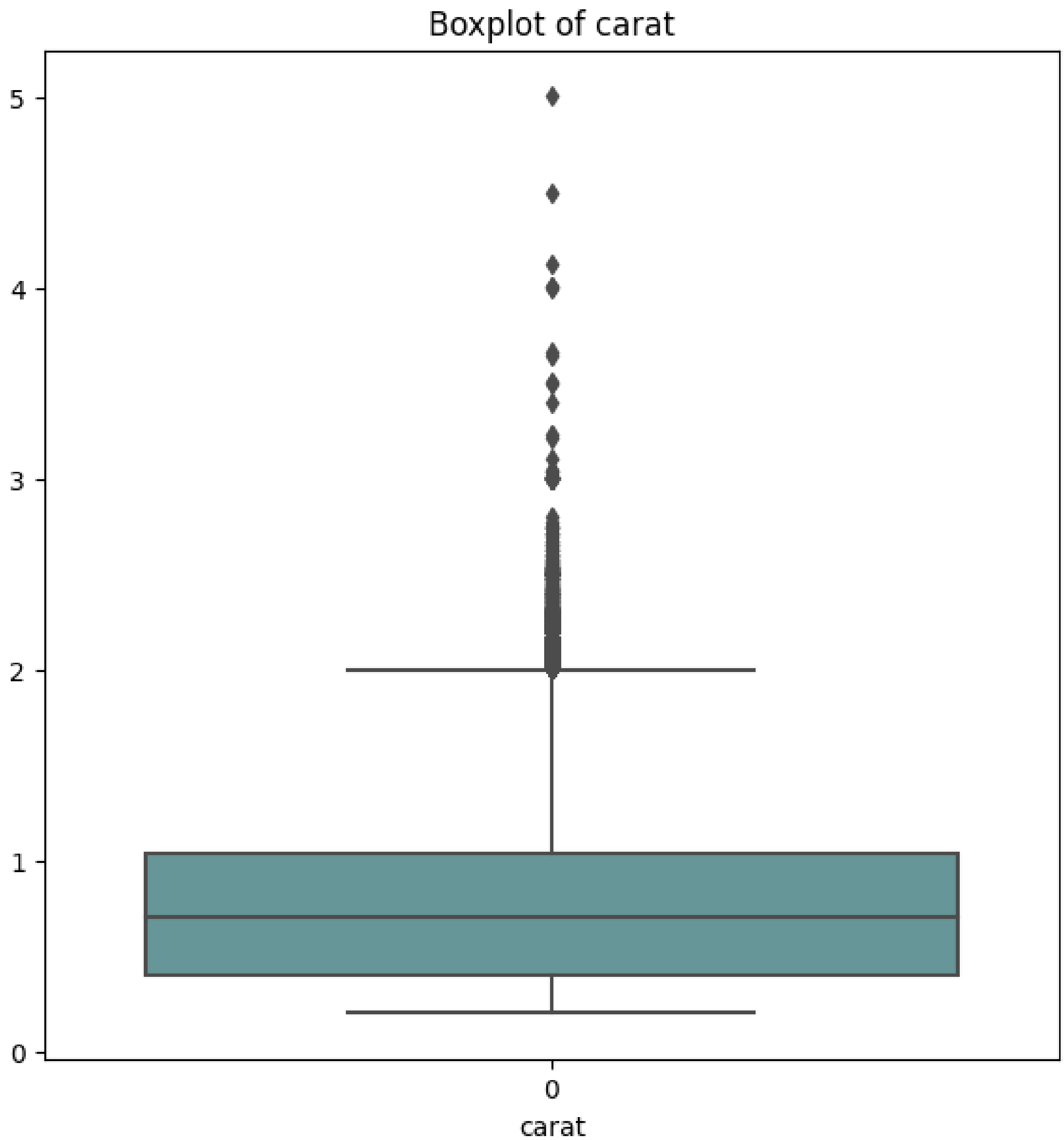


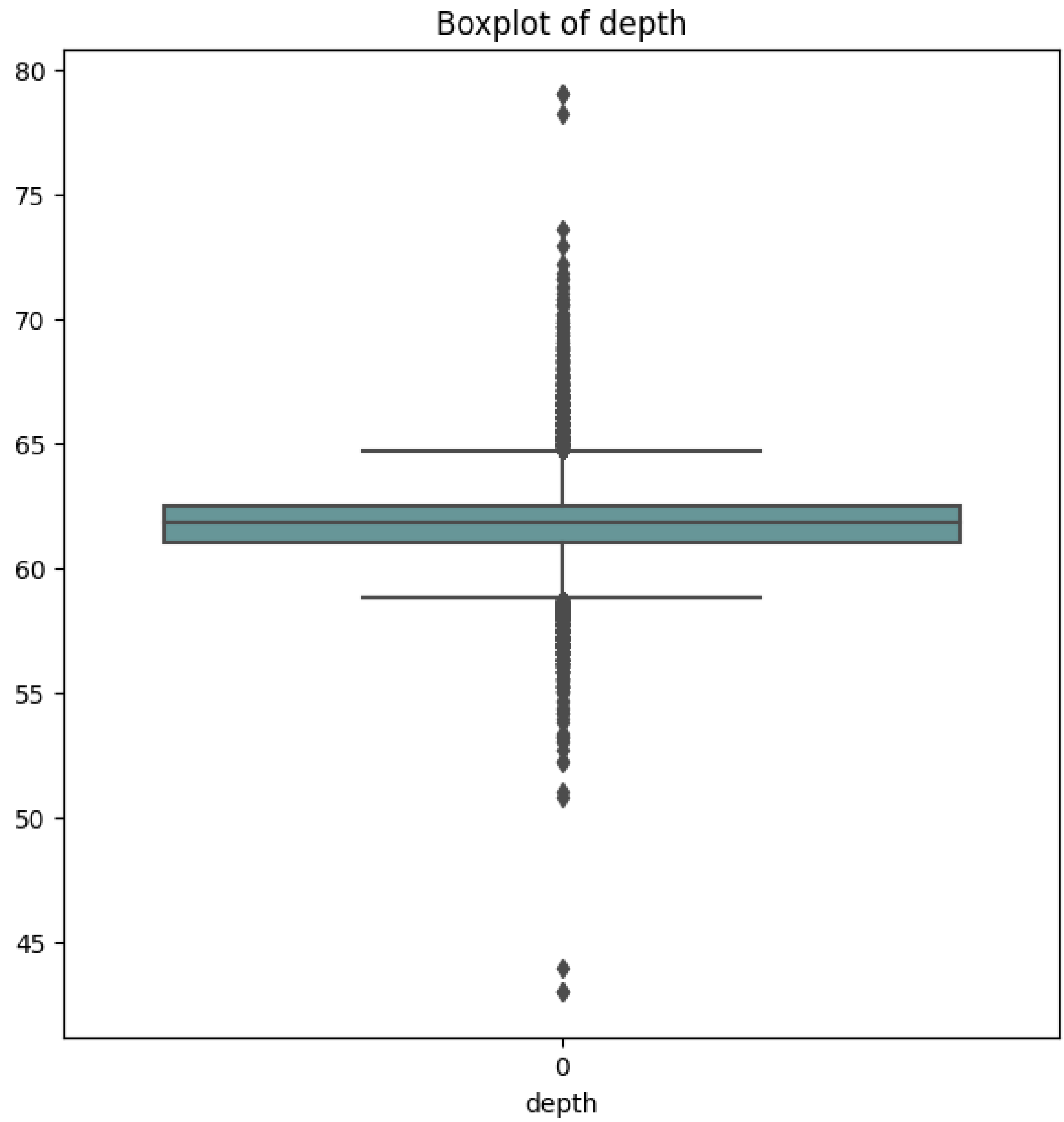
80
75
70
65
60
55
50
45
depth
0
2500
5000
7500
10000
12500
15000
17500
price



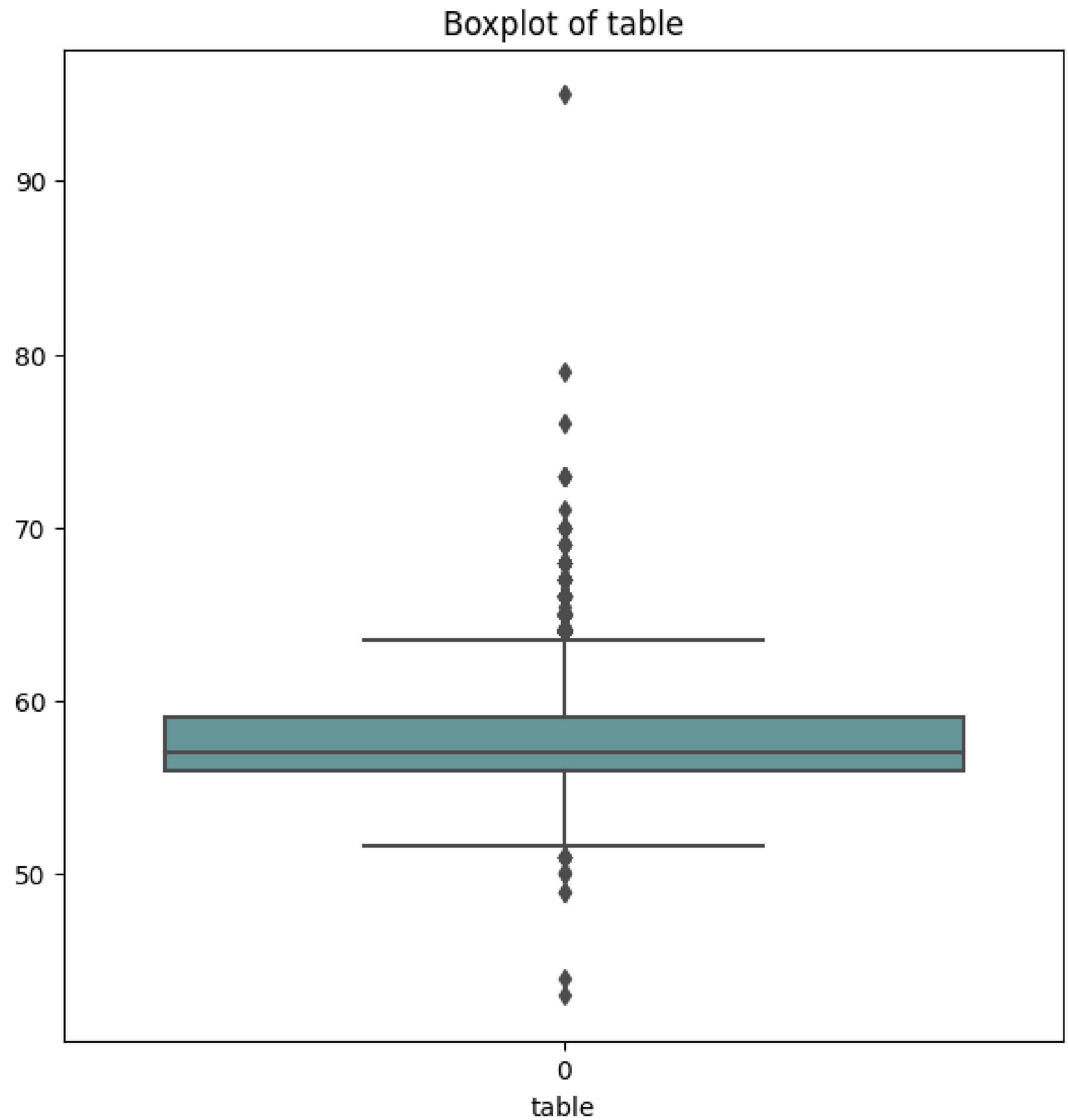
To check outliers

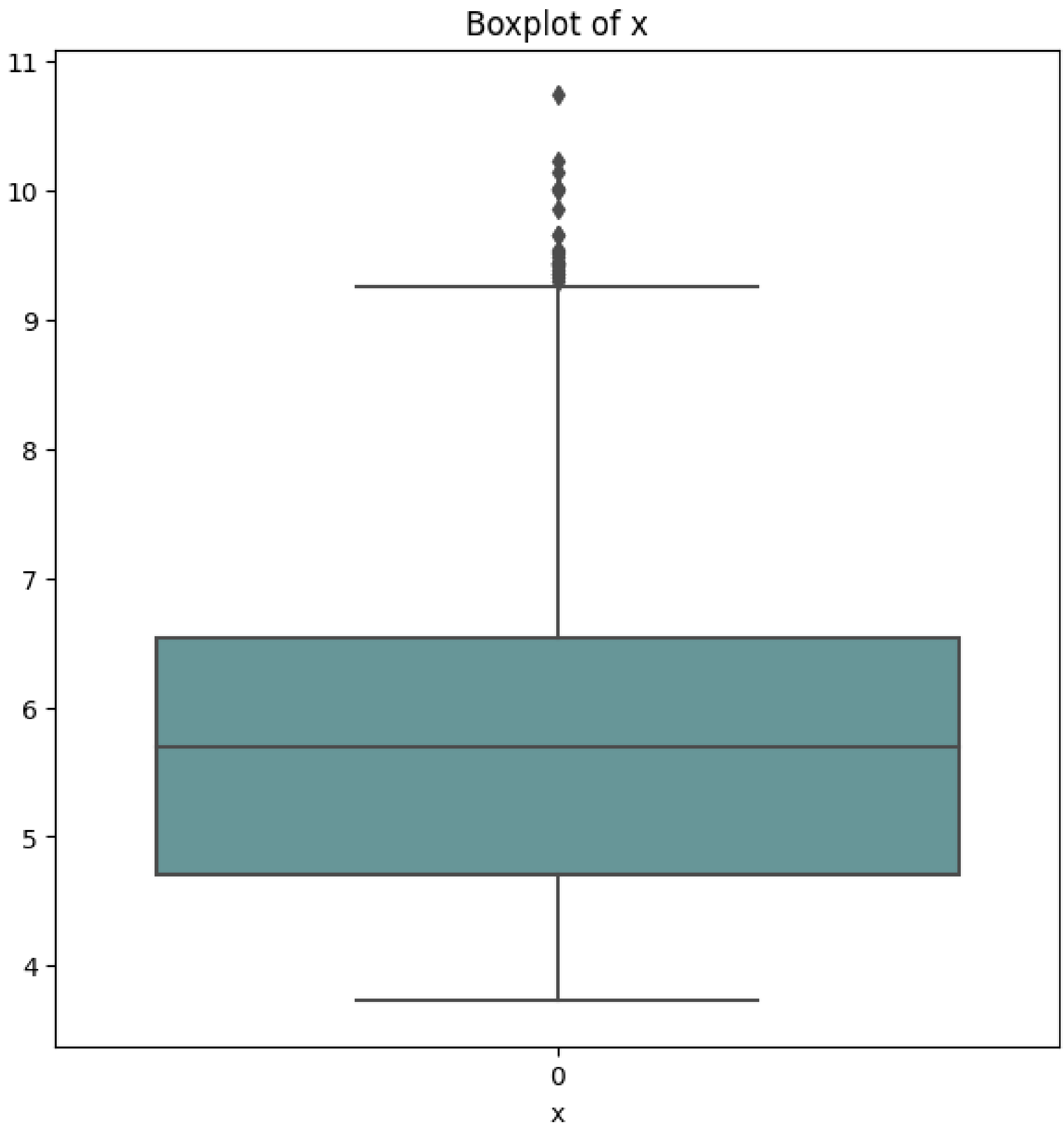
[71]:
```
lst3=['carat','depth','table','x','y','z']
for i in lst3:
  plt.figure(figsize=(7,7))
  sns.boxplot(df1[i],color='cadetblue')
  plt.xlabel(i)
  plt.title(f'Boxplot of {i}')
plt.show()
```

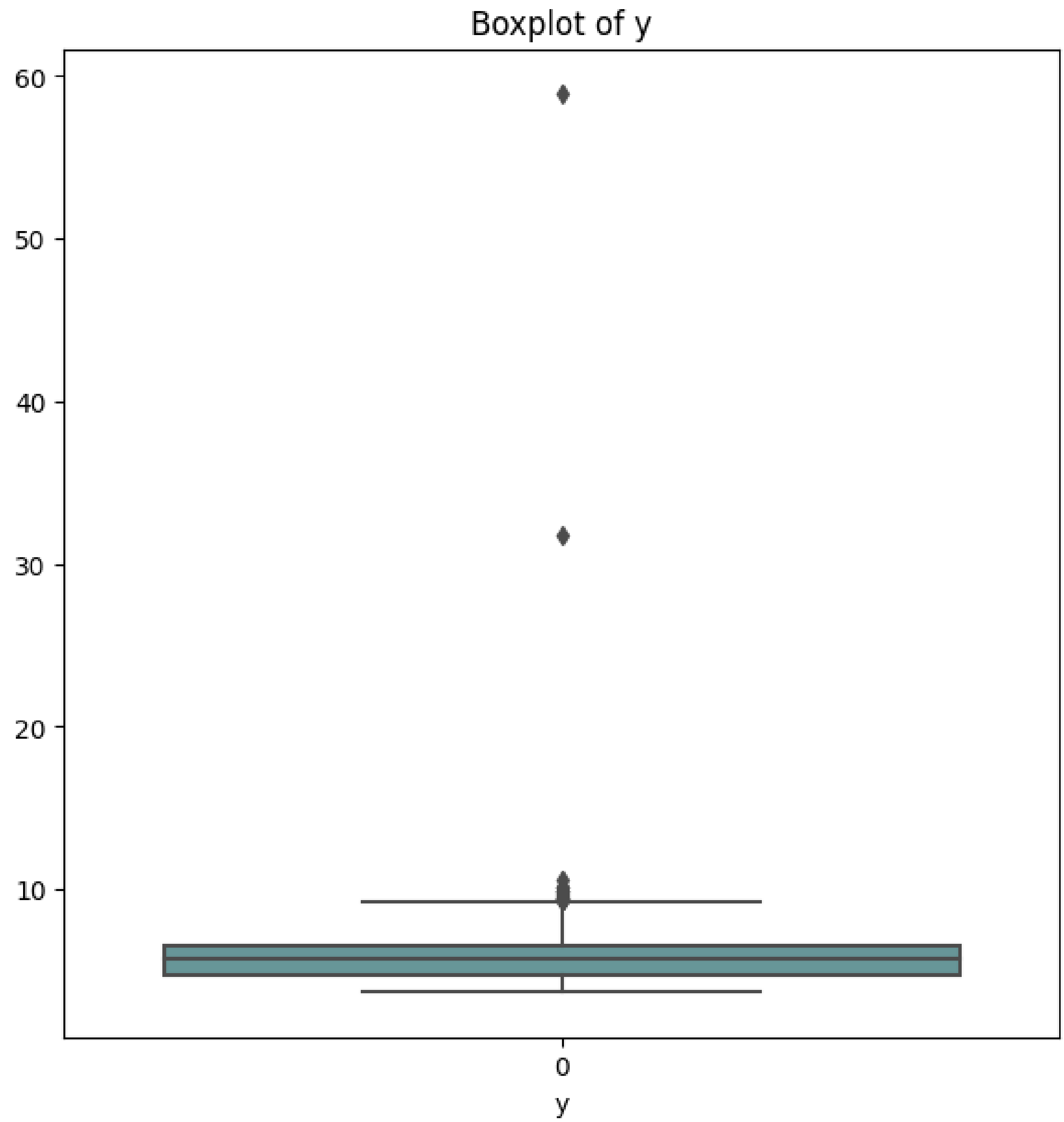

Boxplot of carat
5
4
3
2
1
0
0
carat

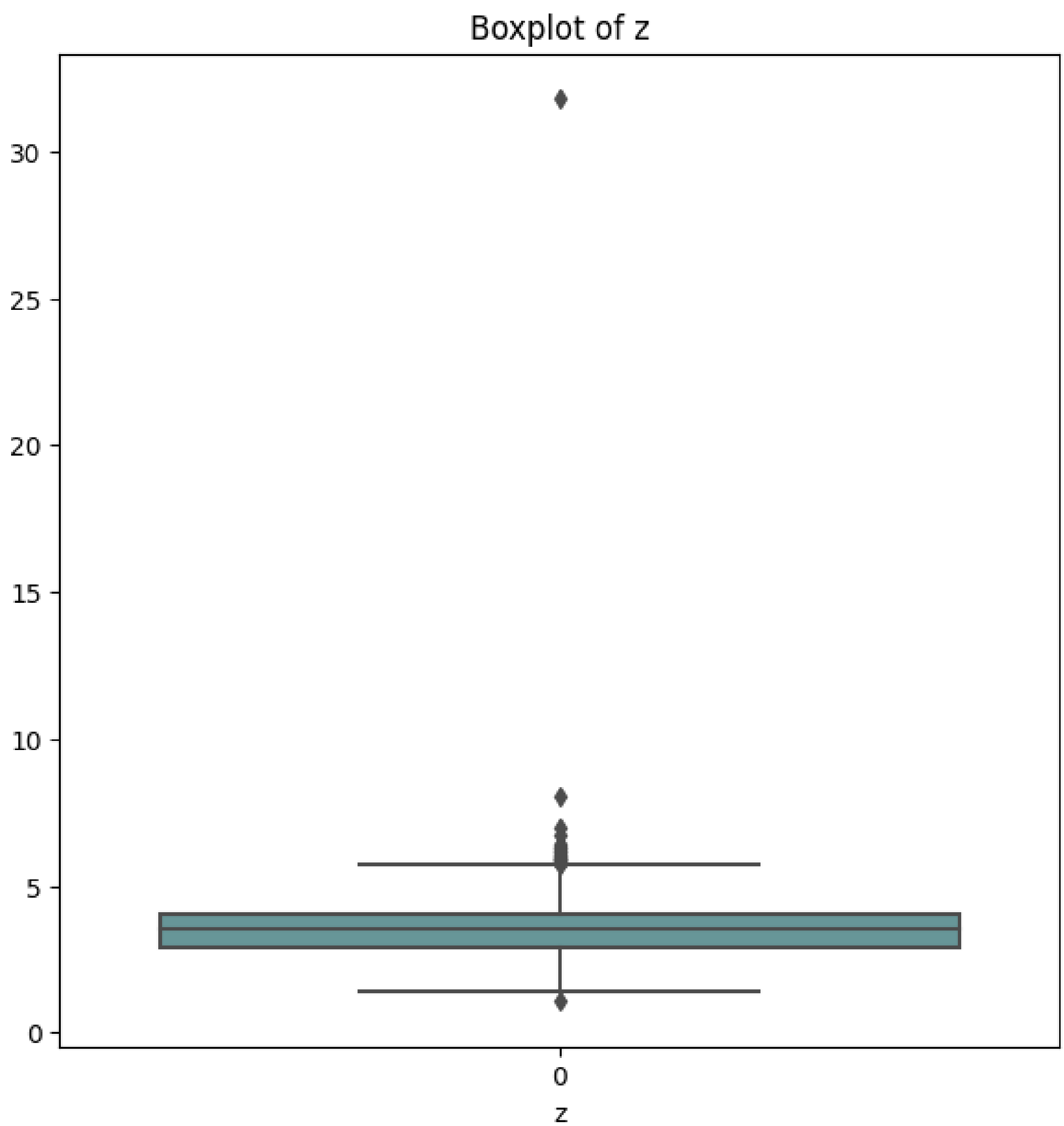


Boxplot of depth
80
75
70
65
60
55
50
45
0
depth


Boxplot of table
90
80
70
60
50
0
table


Boxplot of x
11
10
9
8
7
6
5
4
0
x


Boxplot of y
60
50
40
30
20
10
0
y

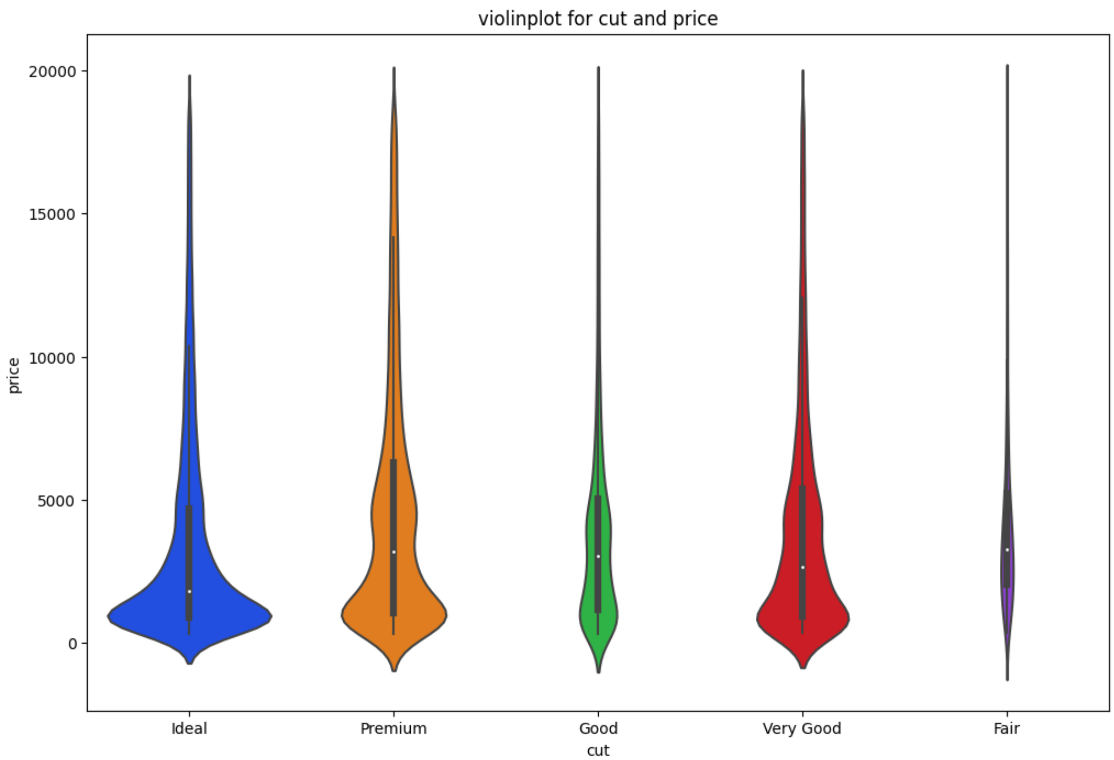


3 4. Data preprocessing

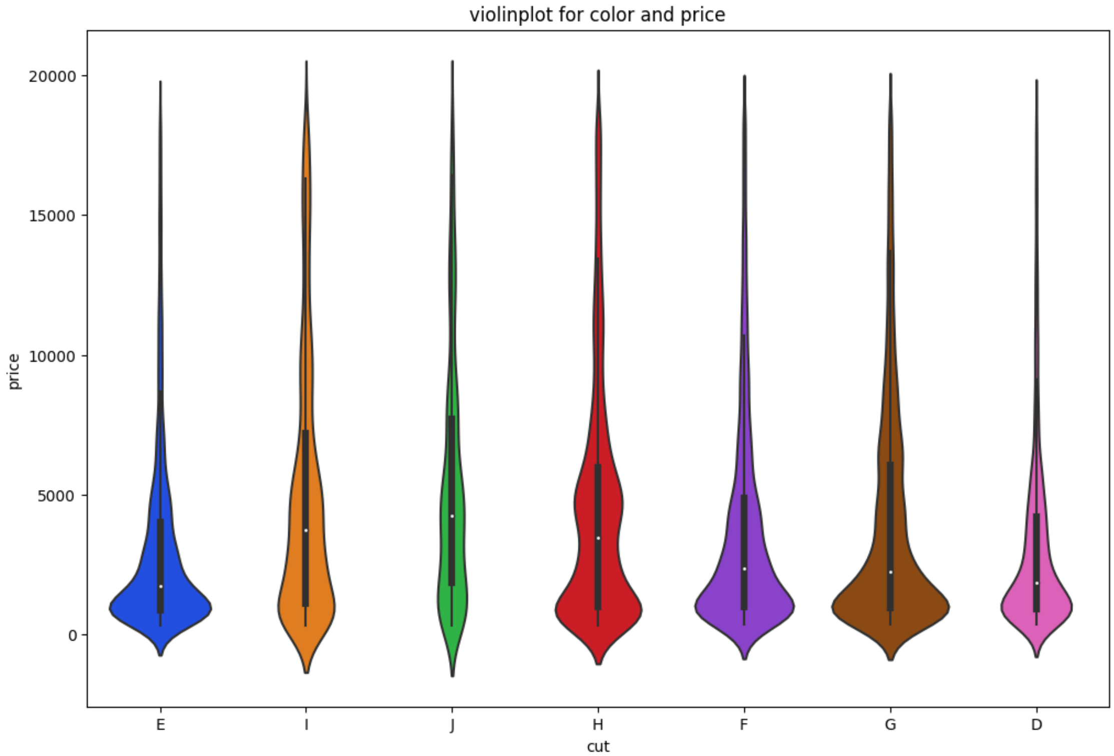
Datatype of features ‘cut’, ‘color’ & ‘clarity’ is “object” which needs to be converted into numerical variable (will be done in data preprocessing) before we feed the data to algorithms.

[28]:
```
plt.figure(figsize=(12,8))
sns.violinplot(x='cut',y='price',data=df1,palette='bright',scale='count')
plt.xlabel('cut')
plt.ylabel('price')
plt.title('violinplot for cut and price',color='black')
```

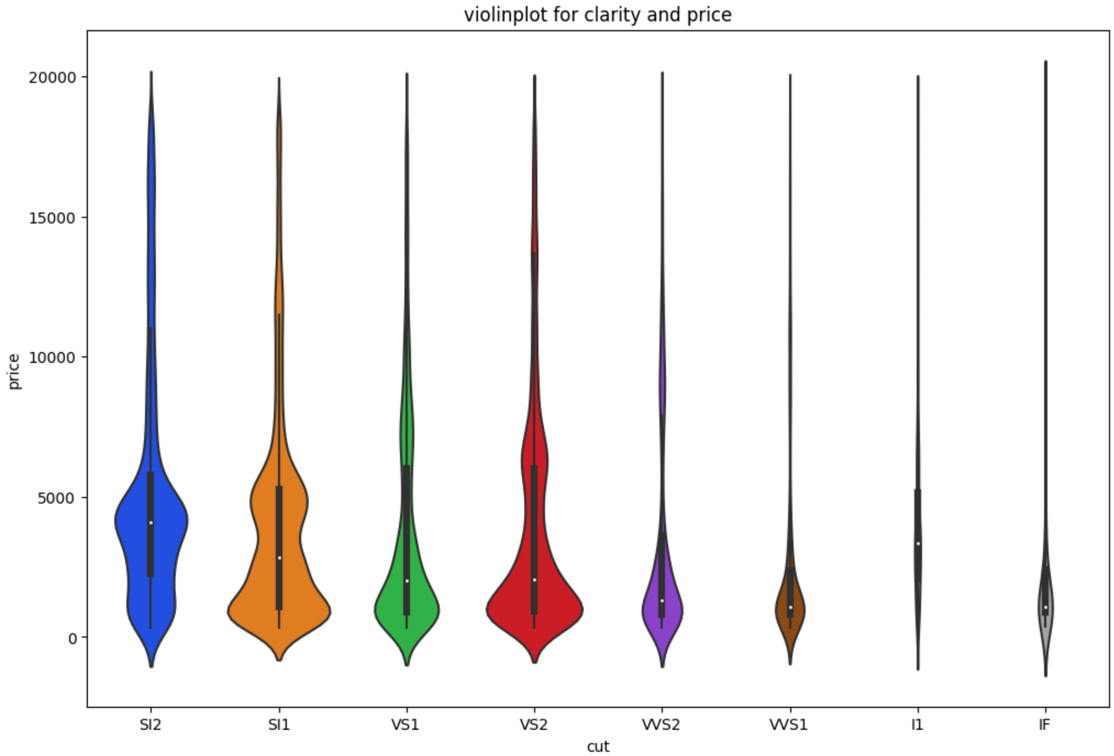
```
plt.show()
```



[29]:
```
plt.figure(figsize=(12,8))
sns.violinplot(x='color',y='price',data=df1,palette='bright',scale='count')
plt.xlabel('cut')
plt.ylabel('price')
plt.title('violinplot for color and price',color='black')
plt.show()
```



[30]:
```
plt.figure(figsize=(12,8))
sns.violinplot(x='clarity',y='price',data=df1,palette='bright',scale='count')
plt.xlabel('cut')
plt.ylabel('price')
plt.title('violinplot for clarity and price',color='black')
plt.show()
```



[31]:
```
df1['cut'] = df1['cut'].map({'Ideal':5,'Premium':4,'Very Good':3,'Good':
  ↪2,'Fair':1})

df1['color'] = df1['color'].map({'D':7,'E':6,'F':5,'G':4,'H':3,'I':2,'J':1})

df1['clarity'] = df1['clarity'].map({'IF':8,'VVS1':7,'VVS2':6,'VS1':5,'VS2':
  ↪4,'SI1':3,'SI2':2,'I1':1})
```

[32]:
```
df1.head()
```

[32]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	5	6	2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	4	6	3	59.8	61.0	326	3.89	3.84	2.31
2	0.23	2	6	5	56.9	65.0	327	4.05	4.07	2.31
3	0.29	4	2	4	62.4	58.0	334	4.20	4.23	2.63
4	0.31	2	1	2	63.3	58.0	335	4.34	4.35	2.75

[33]:
```
df1.dtypes
```

[33]:
```
carat       float64
cut           int64
color         int64
clarity       int64
```

```
depth       float64
table       float64
price         int64
x           float64
y           float64
z           float64
dtype: object
```

[34]:
```
df1.describe().T
```

[34]:
```
            count         mean          std    min     25%      50%      75%  \
carat     53940.0     0.797940     0.474011    0.2    0.40     0.70     1.04
cut       53940.0     3.904097     1.116600    1.0    3.00     4.00     5.00
color     53940.0     4.405803     1.701105    1.0    3.00     4.00     6.00
clarity   53940.0     4.051020     1.647136    1.0    3.00     4.00     5.00
depth     53940.0    61.749405     1.432621   43.0   61.00    61.80    62.50
table     53940.0    57.457184     2.234491   43.0   56.00    57.00    59.00
price     53940.0  3932.799722  3989.439738  326.0  950.00  2401.00  5324.25
x         53940.0     5.731157     1.121761    0.0    4.71     5.70     6.54
y         53940.0     5.734526     1.142135    0.0    4.72     5.71     6.54
z         53940.0     3.538734     0.705699    0.0    2.91     3.53     4.04

               max
carat         5.01
cut           5.00
color         7.00
clarity       8.00
depth        79.00
table        95.00
price     18823.00
x            10.74
y            58.90
z            31.80
```

[35]:
```
df1.shape
```

[35]:
```
(53940, 10)
```

*Min value of "x", "y", "z" are zero this indicates that there are faulty values in data that represents dimensionless or 2-dimensional diamonds. So we need to filter out those as it clearly faulty data points.

here have zero values and outliers

[36]:
```
# Removing the datapoints having min 0 value in either x, y or z features
#Dropping dimentionless diamonds
df1= df1.drop(df1[df1["x"]==0].index)
df1= df1.drop(df1[df1["y"]==0].index)
```

```python
df1= df1.drop(df1[df1["z"]==0].index)
df1.shape
```

[36]: (53920, 10)

Dropping the outliers (since we have huge dataset) by defining appropriate measures across features

[37]:
```python
lst4=['carat','depth','table','x','y','z']
for i in lst4:
  q1=df[i].quantile(0.25)
  q3=df[i].quantile(0.75)
  iqr=q3-q1
  lower=q1-(iqr*1.5)
  upper=q3+(iqr*1.5)
outlier_mask = (df[i]< lower)|(df[i] > upper)
df2= df1[~outlier_mask]
df2.shape
```

[37]: (53891, 10)

4 Correlation :

Correlation values highlight how attributes interact, aiding in feature selection, preprocessing, and even new feature creation. Heatmaps visualize correlations, guiding attribute choices and addressing multicollinearity. This analysis informs model development, ensuring accurate prediction of diamond prices.

[38]:
```python
df3=df2.corr()
df3
```

[38]:

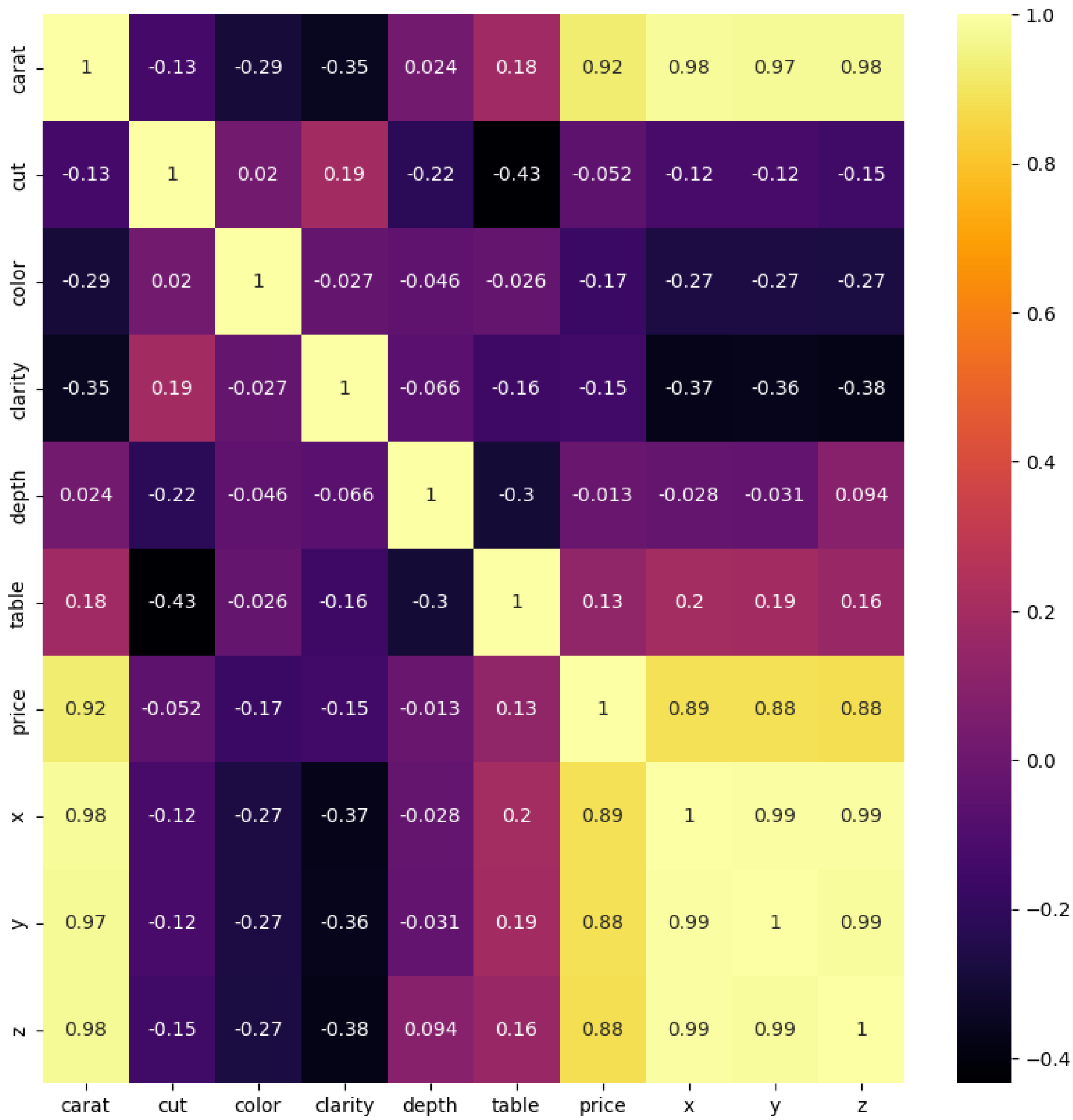
	carat	cut	color	clarity	depth	table	price
carat	1.000000	-0.132596	-0.290426	-0.351238	0.023728	0.182577	0.923195
cut	-0.132596	1.000000	0.019880	0.188394	-0.216637	-0.433977	-0.051958
color	-0.290426	0.019880	1.000000	-0.026754	-0.046455	-0.026329	-0.171248
clarity	-0.351238	0.188394	-0.026754	1.000000	-0.066099	-0.160337	-0.145119
depth	0.023728	-0.216637	-0.046455	-0.066099	1.000000	-0.296064	-0.012985
table	0.182577	-0.433977	-0.026329	-0.160337	-0.296064	1.000000	0.127213
price	0.923195	-0.051958	-0.171248	-0.145119	-0.012985	0.127213	1.000000
x	0.978860	-0.124641	-0.269578	-0.371451	-0.027922	0.196413	0.887008
y	0.972742	-0.123229	-0.267605	-0.363991	-0.030846	0.188777	0.883874
z	0.977318	-0.150527	-0.273814	-0.375032	0.093720	0.156187	0.882015

	x	y	z
carat	0.978860	0.972742	0.977318
cut	-0.124641	-0.123229	-0.150527
color	-0.269578	-0.267605	-0.273814
clarity	-0.371451	-0.363991	-0.375032

```
depth   -0.027922 -0.030846  0.093720
table    0.196413  0.188777  0.156187
price    0.887008  0.883874  0.882015
x        1.000000  0.993340  0.991198
y        0.993340  1.000000  0.986840
z        0.991198  0.986840  1.000000
```

[39]:
```python
plt.figure(figsize = (10, 10))
sns.heatmap(df3, annot = True, cmap = 'inferno')
```

[39]: `<Axes: >`



point of notice

x,y,and z show a high correlation to the target column. depth,cut and table show low correlation.we could consider dropping but let's keep it.

[40]:
```
df2.columns
```

[40]:
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
       'z'],
      dtype='object')
```

[41]:
```
dataset = df2.drop('price', axis = 1)
```

[42]:
```
dataset.head()
```

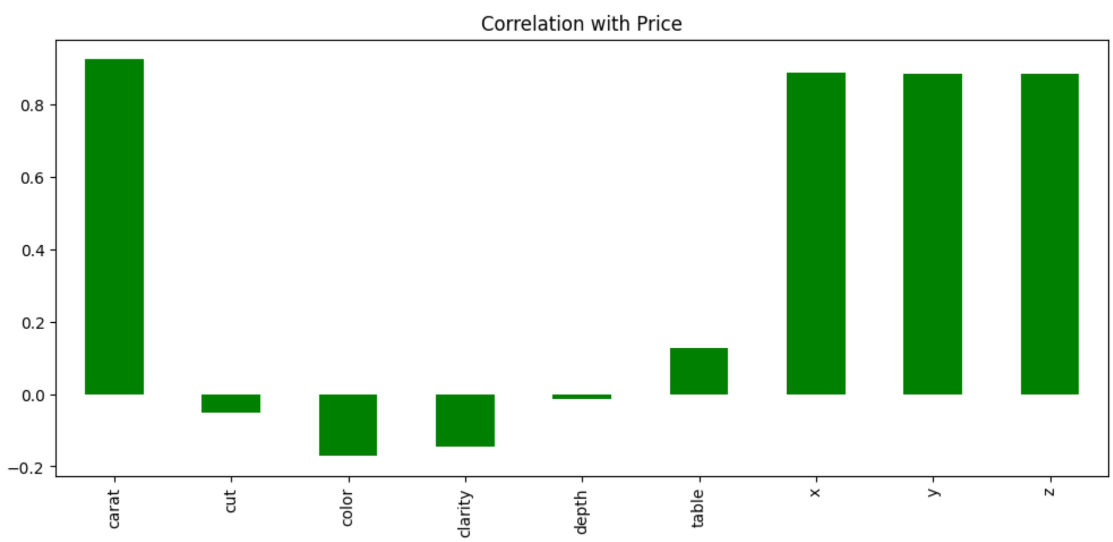
[42]:

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	5	6	2	61.5	55.0	3.95	3.98	2.43
1	0.21	4	6	3	59.8	61.0	3.89	3.84	2.31
2	0.23	2	6	5	56.9	65.0	4.05	4.07	2.31
3	0.29	4	2	4	62.4	58.0	4.20	4.23	2.63
4	0.31	2	1	2	63.3	58.0	4.34	4.35	2.75

Correlation Of Diamond Price with Various Attributes

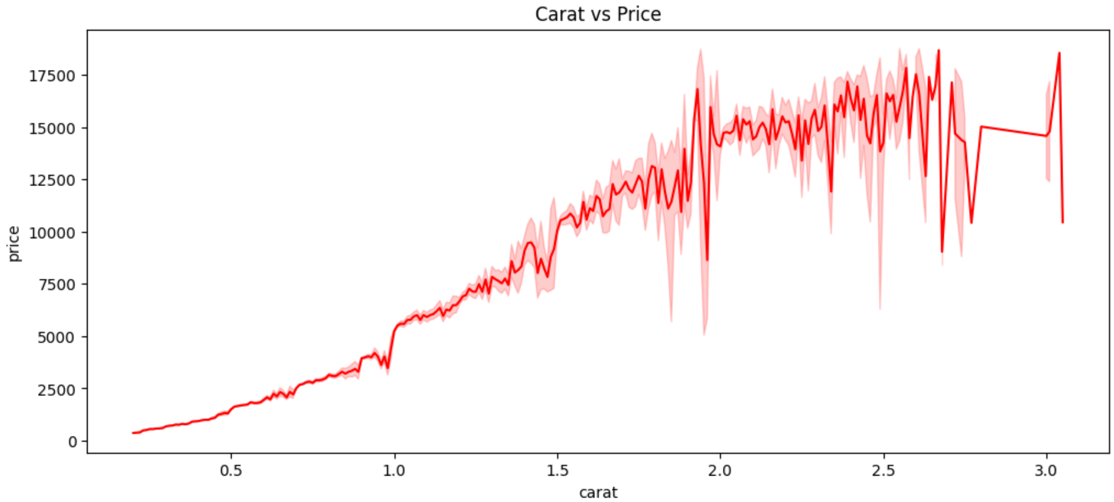
[43]:
```
dataset.corrwith(df2['price']).plot.bar(figsize = (12, 5),title = 'Correlation␣
↪with Price',cmap = 'ocean')
```

[43]:
```
<Axes: title={'center': 'Correlation with Price'}>
```

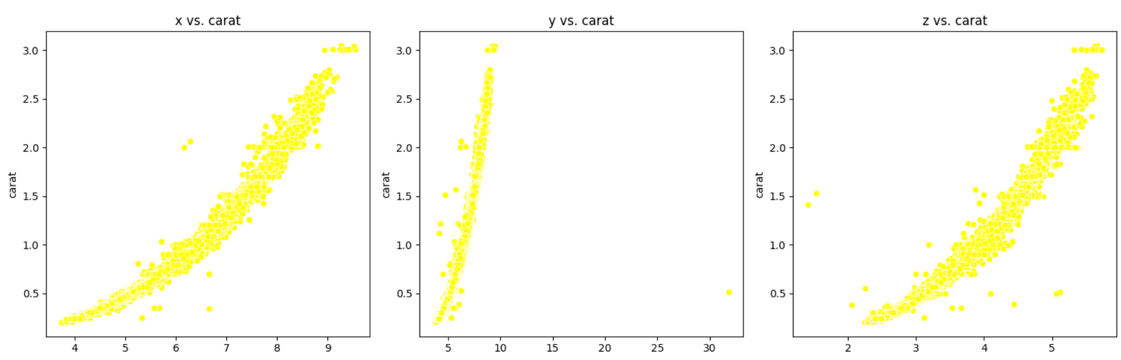


Relation Between Price And Caret

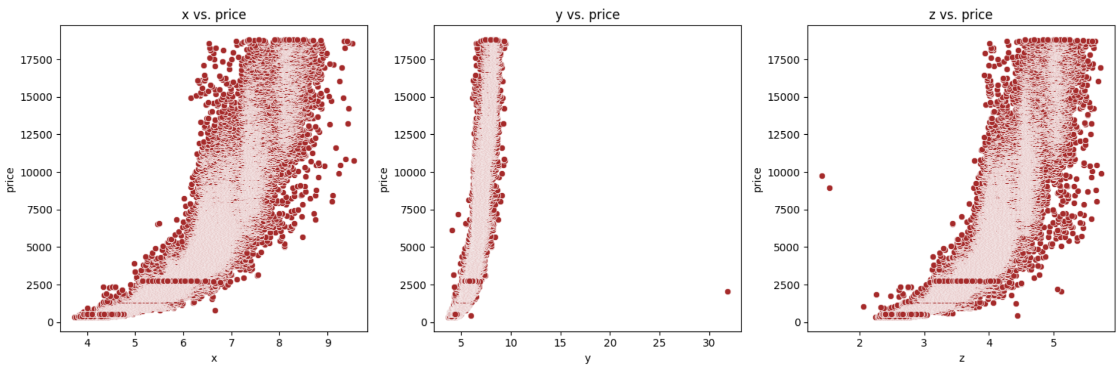
[44]:
```
plt.figure(figsize = (12, 5))
sns.lineplot(x='carat',y='price',data = df2,color='red')
plt.title('Carat vs Price')
plt.show()
```



[45]:
```
columns_to_plot = ['x', 'y', 'z']
titles = ['x vs. carat', 'y vs. carat', 'z vs. carat']

fig, axes = plt.subplots(1, 3, figsize = (15, 5))

for i, column in enumerate(columns_to_plot):
    sns.scatterplot(x = column, y = 'carat', data = df2, ax =␣
 ↪axes[i],color='yellow')
    axes[i].set_title(titles[i])

plt.tight_layout()
plt.show()
```



[46]:
```
columns_to_plot = ['x', 'y', 'z']
titles = ['x vs. price', 'y vs. price', 'z vs. price']

fig, axes = plt.subplots(1, 3, figsize = (15, 5))

for i, column in enumerate(columns_to_plot):
    sns.scatterplot(x = column, y='price', data = df2, ax =␣
↪axes[i],color='brown')
    axes[i].set_title(titles[i])

plt.tight_layout()
plt.show()
```



[47]:
```
df2.head()
```

[47]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	5	6	2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	4	6	3	59.8	61.0	326	3.89	3.84	2.31
2	0.23	2	6	5	56.9	65.0	327	4.05	4.07	2.31
3	0.29	4	2	4	62.4	58.0	334	4.20	4.23	2.63
4	0.31	2	1	2	63.3	58.0	335	4.34	4.35	2.75

5 Splitting dataset

Splitting a dataset refers to the process of dividing a given dataset into two or more subsets for training and evaluation purposes.

Train-Test Split: This is the most basic type of split, where the dataset is divided into a training set and a testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate its performance. The split is typically done using a fixed ratio, such as 70% for training and 30% for testing.

[48]:
```python
x = df2.drop('price', axis = 1)
y = df2['price']
```

[49]:
```python
x.shape,y.shape
```

[49]: ((53891, 9), (53891,))

[50]:
```python
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=42)
x_train
```

[50]:

	carat	cut	color	clarity	depth	table	x	y	z
36175	0.51	3	1	3	62.6	59.0	5.06	5.10	3.18
36332	0.32	4	4	6	61.6	60.0	4.40	4.37	2.70
18880	1.20	3	4	4	63.3	58.0	6.74	6.66	4.24
53265	0.70	4	5	4	63.0	61.0	5.64	5.59	3.54
33073	0.32	5	3	8	62.1	54.0	4.41	4.45	2.75
…	…	…	…	…	…	…	…	…	…
11290	1.29	3	5	2	63.4	57.0	6.92	6.77	4.35
44777	0.56	4	2	7	60.6	59.0	5.29	5.33	3.22
38203	0.43	4	3	4	61.9	55.0	4.95	4.80	3.02
860	0.90	4	1	3	62.8	59.0	6.13	6.03	3.82
15804	1.17	5	3	4	60.3	61.0	6.84	6.80	4.11

[37723 rows x 9 columns]

[51]:
```python
x_test
```

[51]:

	carat	cut	color	clarity	depth	table	x	y	z
32224	0.31	5	4	7	61.9	55.0	4.38	4.41	2.72
44764	0.53	5	5	4	61.9	55.0	5.20	5.23	3.23
18104	1.02	5	4	5	61.0	57.0	6.48	6.50	3.96
18315	1.00	3	5	5	61.9	58.0	6.34	6.37	3.94
33618	0.31	2	4	3	63.1	59.0	4.29	4.33	2.72
…	…	…	…	…	…	…	…	…	…
28510	0.30	5	5	4	63.0	55.0	4.29	4.28	2.70
34472	0.38	3	6	4	60.3	58.0	4.67	4.72	2.83
49812	0.71	2	2	3	59.9	65.0	5.71	5.74	3.43
14994	1.02	5	7	3	61.2	56.0	6.51	6.56	4.00
45811	0.69	5	4	2	62.7	54.0	5.71	5.64	3.56

[16168 rows x 9 columns]

[52]:
```python
y_train
```

```
[52]: 36175     930
      36332     936
      18880    7741
      53265    2648
      33073     814
               …
      11290    4977
      44777    1622
      38203    1016
      860      2871
      15804    6324
      Name: price, Length: 37723, dtype: int64
```

```
[53]: y_test
```

```
[53]: 32224     789
      44764    1621
      18104    7324
      18315    7450
      33618     462
               …
      28510     673
      34472     866
      49812    2165
      14994    6040
      45811    1711
      Name: price, Length: 16168, dtype: int64
```

Model selection and training

Linear Regression multiple linear regression

```
[54]: from sklearn.linear_model import LinearRegression
      model=LinearRegression()
      model.fit(x_train,y_train)
      y_pred=model.predict(x_test)
      y_pred
```

```
[54]: array([1433.13554112, 1847.69357385, 6204.00090085, …, 1422.05966109,
             6222.02818092, 1573.09340147])
```

```
[55]: df4=pd.DataFrame({'actual value':y_test,'predicted value':y_pred,'Difference':
        ↪y_test-y_pred})
      df4
```

```
[55]:        actual value  predicted value   Difference
      32224           789      1433.135541  -644.135541
      44764          1621      1847.693574  -226.693574
```

18104	7324	6204.000901	1119.999099
18315	7450	6117.120143	1332.879857
33618	462	-928.604022	1390.604022
…	…	…	…
28510	673	158.792722	514.207278
34472	866	882.746311	-16.746311
49812	2165	1422.059661	742.940339
14994	6040	6222.028181	-182.028181
45811	1711	1573.093401	137.906599

[16168 rows x 3 columns]

[56]:
```
print("slope is",model.coef_)
print("constant is",model.intercept_)
list(zip(x,model.coef_))
```

```
slope is [11685.60704703   126.5050641    328.97251853   492.88165444
    52.57044255   -20.35022654 -2190.95222314  2369.5884777
 -2260.9505913 ]
constant is -4426.566394531432
```

[56]:
```
[('carat', 11685.607047028516),
 ('cut', 126.5050640996011),
 ('color', 328.9725185309023),
 ('clarity', 492.88165444143533),
 ('depth', 52.570442550760674),
 ('table', -20.35022653610659),
 ('x', -2190.952223138188),
 ('y', 2369.588477696599),
 ('z', -2260.9505913004573)]
```

preformance evaluvation

[57]:
```
#r2 score
from sklearn.metrics import r2_score
r0=r2_score(y_test,y_pred)
print("score is",r0)
```

```
score is 0.9013864653961818
```

polynomial *regression*

[58]:
```
from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=3)
x_poly=poly.fit_transform(x)
x_poly
```

[58]:
```
array([[  1.        ,   0.23     ,   5.       , ...,  38.492172,  23.501502,
         14.348907],
       [  1.        ,   0.21     ,   4.       , ...,  34.062336,  20.490624,
         12.326391],
       [  1.        ,   0.23     ,   2.       , ...,  38.264919,  21.717927,
         12.326391],
       ...,
       [  1.        ,   0.7      ,   3.       , ..., 114.854144,  71.986048,
         45.118016],
       [  1.        ,   0.86     ,   4.       , ..., 140.079456,  85.604112,
         52.313624],
       [  1.        ,   0.75     ,   5.       , ..., 125.423116,  77.775152,
         48.228544]])
```

[59]:
```python
poly.fit(x_poly,y)
model1=LinearRegression()
model1.fit(x_poly,y)
y_poly=model1.predict(x_poly)
y_poly
```

[59]:
```
array([ 156.11876087,  443.59567646,  635.02385464, ..., 2457.69576323,
       2824.32683189, 2486.74148089])
```

[60]:
```python
x = df2.drop('price', axis = 1)
y = df2['price']
```

[61]:
```python
#r2 score
from sklearn.metrics import r2_score
r1=r2_score(y,y_poly)
print("score is",r2_score(y,y_poly))

from sklearn.metrics import mean_absolute_percentage_error
print('MAEP:',mean_absolute_percentage_error(y_poly,y))
```

```
score is 0.9780682887426088
MAEP: 0.12979788405525775
```

Decission Tree

[62]:
```python
from sklearn.tree import DecisionTreeRegressor
dtr= DecisionTreeRegressor()
dtr.fit(x_train, y_train)
dtr.score(x_test, y_test)
```

[62]: 0.9661267552710133

[63]:
```python
y_pred1=dtr.predict(x_test)
y_pred1
```

[63]: array([789., 1566., 7077., …, 1651., 5804., 1673.])

[64]:
```
#r2 score
from sklearn.metrics import r2_score
r2=r2_score(y_test,y_pred1)
print("score is",r1)
print('MAEP:',mean_absolute_percentage_error(y_pred1,y_test))
```

```
score is 0.9780682887426088
MAEP: 0.08585478313891322
```

Random Forest*

[65]:
```
from sklearn.ensemble import RandomForestRegressor
reg = RandomForestRegressor()
reg.fit(x_train, y_train)
reg.score(x_test, y_test)
```

[65]: 0.9818440865091461

[66]:
```
reg.score(x_train, y_train)
```

[66]: 0.9973870871659626

[67]:
```
y_pred2= reg.predict(x_test)
y_pred2
```

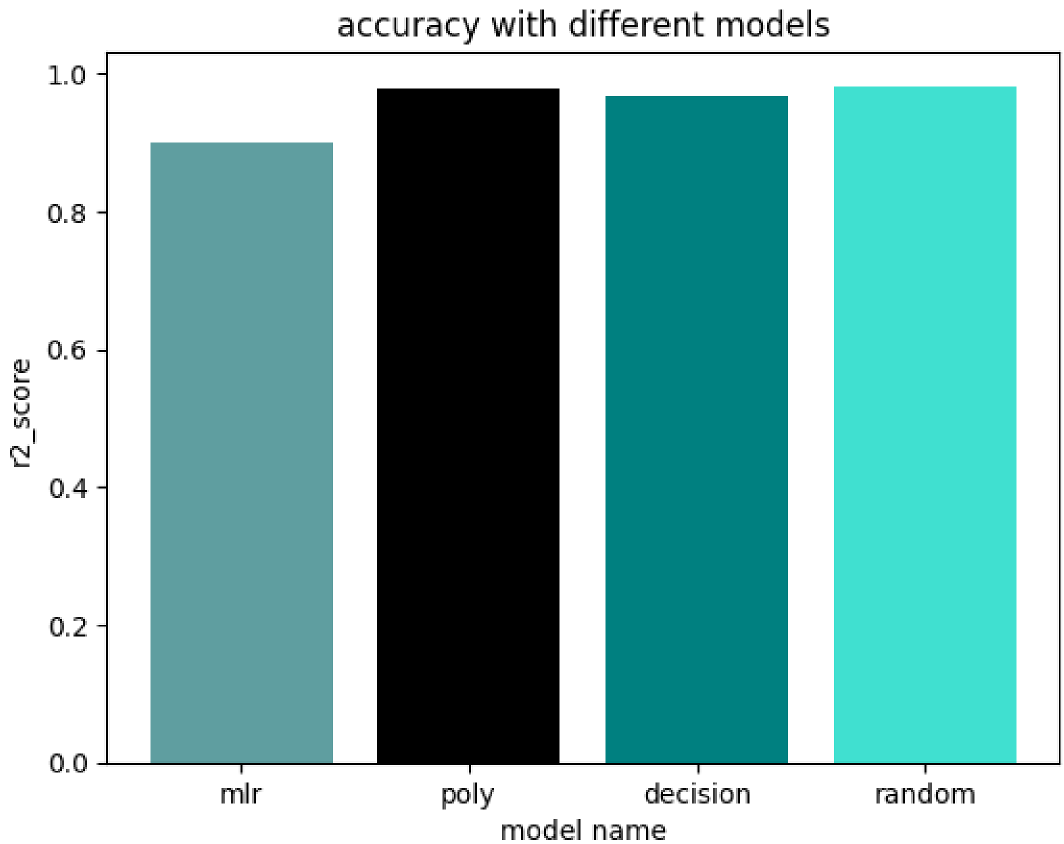
[67]: array([788.97, 1702.9 , 7170.91, …, 1836.43, 6119.01, 1750.99])

[68]:
```
#r2 score
from sklearn.metrics import r2_score
r3=r2_score(y_test,y_pred2)
print("score is",r2)
print('MAEP:',mean_absolute_percentage_error(y_pred2,y_test))
```

```
score is 0.9661267552710133
MAEP: 0.06296631254629519
```

[70]:
```
all=['mlr','poly','decision','random']
result=[r0,r1,r2,r3]
colors=['cadetblue','black','teal','turquoise']
plt.bar(all,result,color=colors)
plt.xlabel('model name')
plt.ylabel('r2_score')
plt.title('accuracy with different models')
```

[70]: Text(0.5, 1.0, 'accuracy with different models')



Conclusion :

All the models have almost same accuracy. However, the Random Forest Regression model is slightly better than the other three models.

6 Future Importance Of Diamond Price Prediction

Market Efficiency:

`ML models can contribute to making the diamond market more efficient by providing accurate and`

Customization and Personalization:

`ML models can be used to create personalized pricing models based on individual diamond charac`

Supply Chain Optimization:

`Predictive models can help optimize the diamond supply chain by predicting demand, identifying`

Risk Management:

`Predictive models can aid in assessing the risk associated with diamond investments and transa`

Fraud Detection:

ML algorithms can be employed to detect anomalies and potential fraud in the diamond trade. Thi