diamond_price_predication

August 10, 2023

1 Diamond Price Predication

The aim of diamond price prediction is to develop a model that can accurately estimate the price of a diamond based on its various attributes such as carat weight, cut quality, color, and clarity. By analyzing historical diamond data and utilizing machine learning techniques, this predictive model enables buyers, sellers, and enthusiasts to make informed decisions when buying or selling diamonds, considering factors that influence their value. The dataset we use has 219703 observations and 26 variables but we use only 13 vairables.

[1]:
```
#import neccessory libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Below is the column names and description that we using to predict the price of is listed.

[2]:
```
pd.read_csv("D:\Data Analysis\Projects\Diamond Price Predication\columns.csv")
```

[2]:
```
         Column Name                                         Description
0                cut  cut refers to one of the 10 or so most common …
1              color  Clear diamonds are graded D-Z. The higher lett…
2            clarity  clarity refers the inclusions (i.e., internal …
3       carat_weight  carat_weight Refers to the mass of the diamond…
4        cut_quality  cut_quality refers the GIA Cut Grading System …
5                lab  lab is the grading lab. The big three are GIA,…
6      table_percent  table_percent are the relative measurements of…
7           symmetry     polish and symmetry are what you would expect.
8             polish     polish and symmetry are what you would expect.
9        meas_length         the absolute length measurements of stone.
10        meas_width          the absolute width measurements of stone.
11        meas_depth          the absolute depth measurements of stone.
12  total_sales_price           total_sales_price is priced in dollars.
```

Importing the Diamond CSV data file that I downloaded from Kaggle

[3]:
```
diamond = pd.read_csv("D:\Data Analysis\Projects\Diamond Price␣
 ↪Predication\diamonds.csv")
```

[4]:
```
diamond
```

[4]:
```
        Unnamed: 0      cut    color clarity  carat_weight cut_quality  lab  \
0                0    Round        E    VVS2          0.09   Excellent  IGI
1                1    Round        E    VVS2          0.09   Very Good  IGI
2                2    Round        E    VVS2          0.09   Excellent  IGI
3                3    Round        E    VVS2          0.09   Excellent  IGI
4                4    Round        E    VVS2          0.09   Very Good  IGI
…              …      …      …      …           …         …   …
219698      219699    Round        E     VS1         10.65   Excellent  GIA
219699      219700  Radiant  unknown     VS2          5.17     unknown  GIA
219700      219701    Round        E     VS1         18.07   Excellent  GIA
219701      219702 Princess  unknown     SI2          0.90     unknown  GIA
219702      219703     Pear  unknown    VVS2         10.03     unknown  GIA

         symmetry     polish eye_clean  … meas_depth girdle_min  girdle_max  \
0       Very Good  Very Good   unknown  …       1.79          M           M
1       Very Good  Very Good   unknown  …       1.78        STK         STK
2       Very Good  Very Good   unknown  …       1.77         TN           M
3       Very Good  Very Good   unknown  …       1.78          M         STK
4       Very Good  Excellent   unknown  …       1.82        STK         STK
…             …        …     …   …      …        …          …
219698  Excellent  Excellent   unknown  …       8.66          M         STK
219699  Very Good  Very Good   unknown  …       5.71         TK         XTK
219700  Excellent  Excellent   unknown  …      10.20         TN           M
219701       Good       Good   unknown  …       3.47        XTN         VTK
219702  Very Good  Excellent   unknown  …       7.39    unknown     unknown

        fluor_color  fluor_intensity  fancy_color_dominant_color  \
0           unknown             None                     unknown
1           unknown             None                     unknown
2           unknown             None                     unknown
3           unknown             None                     unknown
4           unknown             None                     unknown
…             …              …                         …
219698      unknown             None                     unknown
219699      unknown             None                       Green
219700      unknown             None                     unknown
219701      unknown            Faint                         Red
219702      unknown             None                      Yellow

        fancy_color_secondary_color fancy_color_overtone  \
0                           unknown              unknown
1                           unknown              unknown
2                           unknown              unknown
3                           unknown              unknown
4                           unknown              unknown
```

```
…                                  …                    …
219698                       unknown              unknown
219699                       unknown                 None
219700                       unknown              unknown
219701                       unknown              unknown
219702                       unknown              unknown

       fancy_color_intensity total_sales_price
0                    unknown               200
1                    unknown               200
2                    unknown               200
3                    unknown               200
4                    unknown               200
…                        …                 …
219698               unknown           1210692
219699           Fancy Light           1292500
219700               unknown           1315496
219701                 Fancy           1350000
219702           Fancy Vivid           1449881

[219703 rows x 26 columns]
```

2 Data Preprocessing

Cheking the null values

[5]:
```
diamond.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219703 entries, 0 to 219702
Data columns (total 26 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Unnamed: 0                     219703 non-null  int64
 1   cut                            219703 non-null  object
 2   color                          219703 non-null  object
 3   clarity                        219703 non-null  object
 4   carat_weight                   219703 non-null  float64
 5   cut_quality                    219703 non-null  object
 6   lab                            219703 non-null  object
 7   symmetry                       219703 non-null  object
 8   polish                         219703 non-null  object
 9   eye_clean                      219703 non-null  object
 10  culet_size                     219703 non-null  object
 11  culet_condition                219703 non-null  object
 12  depth_percent                  219703 non-null  float64
 13  table_percent                  219703 non-null  float64
 14  meas_length                    219703 non-null  float64
```

```
 15  meas_width                    219703 non-null  float64
 16  meas_depth                    219703 non-null  float64
 17  girdle_min                    219703 non-null  object
 18  girdle_max                    219703 non-null  object
 19  fluor_color                   219703 non-null  object
 20  fluor_intensity               219703 non-null  object
 21  fancy_color_dominant_color    219703 non-null  object
 22  fancy_color_secondary_color   219703 non-null  object
 23  fancy_color_overtone          219703 non-null  object
 24  fancy_color_intensity         219703 non-null  object
 25  total_sales_price             219703 non-null  int64
dtypes: float64(6), int64(2), object(18)
memory usage: 43.6+ MB
```

[6]:
```
diamond.isnull().sum()
```

[6]:
```
Unnamed: 0                     0
cut                            0
color                          0
clarity                        0
carat_weight                   0
cut_quality                    0
lab                            0
symmetry                       0
polish                         0
eye_clean                      0
culet_size                     0
culet_condition                0
depth_percent                  0
table_percent                  0
meas_length                    0
meas_width                     0
meas_depth                     0
girdle_min                     0
girdle_max                     0
fluor_color                    0
fluor_intensity                0
fancy_color_dominant_color     0
fancy_color_secondary_color    0
fancy_color_overtone           0
fancy_color_intensity          0
total_sales_price              0
dtype: int64
```

Dropping the unnecessary columns that has unknown values more than 80% that will affect our model

[7]:
```
diamond.drop(['Unnamed: 0', 'eye_clean', 'culet_size',
       'culet_condition', 'depth_percent', 'girdle_min', 'girdle_max',␣
 ↪'fluor_color',
       'fluor_intensity', 'fancy_color_dominant_color',
       'fancy_color_secondary_color', 'fancy_color_overtone',
       'fancy_color_intensity'], axis=1, inplace=True)
```

[8]:
```
diamond.head()
```

[8]:
```
     cut color clarity  carat_weight cut_quality  lab  symmetry     polish  \
0  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
1  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Very Good
2  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
3  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
4  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Excellent

   table_percent  meas_length  meas_width  meas_depth  total_sales_price
0           59.0         2.85        2.87        1.79                200
1           59.0         2.84        2.89        1.78                200
2           59.0         2.88        2.90        1.77                200
3           59.0         2.86        2.88        1.78                200
4           58.5         2.79        2.83        1.82                200
```

[9]:
```
diamond.shape
```

[9]:
```
(219703, 13)
```

[10]:
```
diamond.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 219703 entries, 0 to 219702
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   cut                219703 non-null  object
 1   color              219703 non-null  object
 2   clarity            219703 non-null  object
 3   carat_weight       219703 non-null  float64
 4   cut_quality        219703 non-null  object
 5   lab                219703 non-null  object
 6   symmetry           219703 non-null  object
 7   polish             219703 non-null  object
 8   table_percent      219703 non-null  float64
 9   meas_length        219703 non-null  float64
 10  meas_width         219703 non-null  float64
 11  meas_depth         219703 non-null  float64
 12  total_sales_price  219703 non-null  int64
dtypes: float64(5), int64(1), object(7)
```

```
memory usage: 21.8+ MB
```

Checking how many unique values in main columns have

[11]:
```python
diamond['cut'].nunique()
```

[11]:
```
11
```

[12]:
```python
diamond['color'].nunique()
```

[12]:
```
11
```

[13]:
```python
diamond['clarity'].nunique()
```

[13]:
```
11
```

[14]:
```python
diamond['cut_quality'].nunique()
```

[14]:
```
6
```

[15]:
```python
diamond['lab'].nunique()
```

[15]:
```
3
```

[16]:
```python
diamond['symmetry'].nunique()
```

[16]:
```
5
```

[17]:
```python
diamond['polish'].nunique()
```

[17]:
```
5
```

Cheking descriptive statistics

[18]:
```python
diamond.describe()
```

[18]:
```
         carat_weight  table_percent    meas_length     meas_width  \
count   219703.000000  219703.000000  219703.000000  219703.000000
mean         0.755176      57.747585       5.548853       5.135626
std          0.845894       9.959928       1.763924       1.374529
min          0.080000       0.000000       0.000000       0.000000
25%          0.310000      57.000000       4.350000       4.310000
50%          0.500000      58.000000       5.060000       4.800000
75%          1.000000      60.000000       6.350000       5.700000
max         19.350000      94.000000      93.660000      62.300000

           meas_depth  total_sales_price
count   219703.000000       2.197030e+05
mean         3.285699       6.908062e+03
std          2.054822       2.595949e+04
```

```
min        0.000000    2.000000e+02
25%        2.680000    9.580000e+02
50%        3.030000    1.970000e+03
75%        3.630000    5.207000e+03
max       76.300000    1.449881e+06
```

Count the values of categorical variables

[19]:
```
diamond.cut.value_counts()
```

[19]:
```
Round               158316
Oval                 13857
Emerald              11091
Pear                  9860
Princess              7050
Radiant               5630
Heart                 4774
Cushion Modified      3984
Marquise              2916
Asscher               1696
Cushion                529
Name: cut, dtype: int64
```

[20]:
```
diamond.color.value_counts()
```

[20]:
```
E          33103
F          31566
D          30873
G          29184
H          26073
I          22364
J          16898
K          11750
unknown     9162
L           5683
M           3047
Name: color, dtype: int64
```

[21]:
```
diamond.clarity.value_counts()
```

[21]:
```
SI1     38627
VS2     38173
VS1     36956
SI2     31105
VVS2    28985
VVS1    27877
IF       9974
I1       6961
```

```
I2        944
I3         91
SI3        10
Name: clarity, dtype: int64
```

[22]:
```
diamond.cut_quality.value_counts()
```

[22]:
```
Excellent    124861
unknown       60607
Very Good     34201
Good             28
Fair              5
Ideal             1
Name: cut_quality, dtype: int64
```

[23]:
```
diamond.symmetry.value_counts()
```

[23]:
```
Excellent    131619
Very Good     83143
Good           4609
Fair            325
Poor              7
Name: symmetry, dtype: int64
```

[24]:
```
diamond.polish.value_counts()
```

[24]:
```
Excellent    175806
Very Good     42323
Good           1565
Fair              7
Poor              2
Name: polish, dtype: int64
```

[25]:
```
diamond.head(10)
```

[25]:
```
     cut color clarity  carat_weight cut_quality  lab   symmetry     polish  \
0  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
1  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Very Good
2  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
3  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
4  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Excellent
5  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Very Good
6  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Very Good
7  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good
8  Round     E    VVS2          0.09   Very Good  IGI  Very Good  Very Good
9  Round     E    VVS2          0.09   Excellent  IGI  Very Good  Very Good

   table_percent  meas_length  meas_width  meas_depth  total_sales_price
```

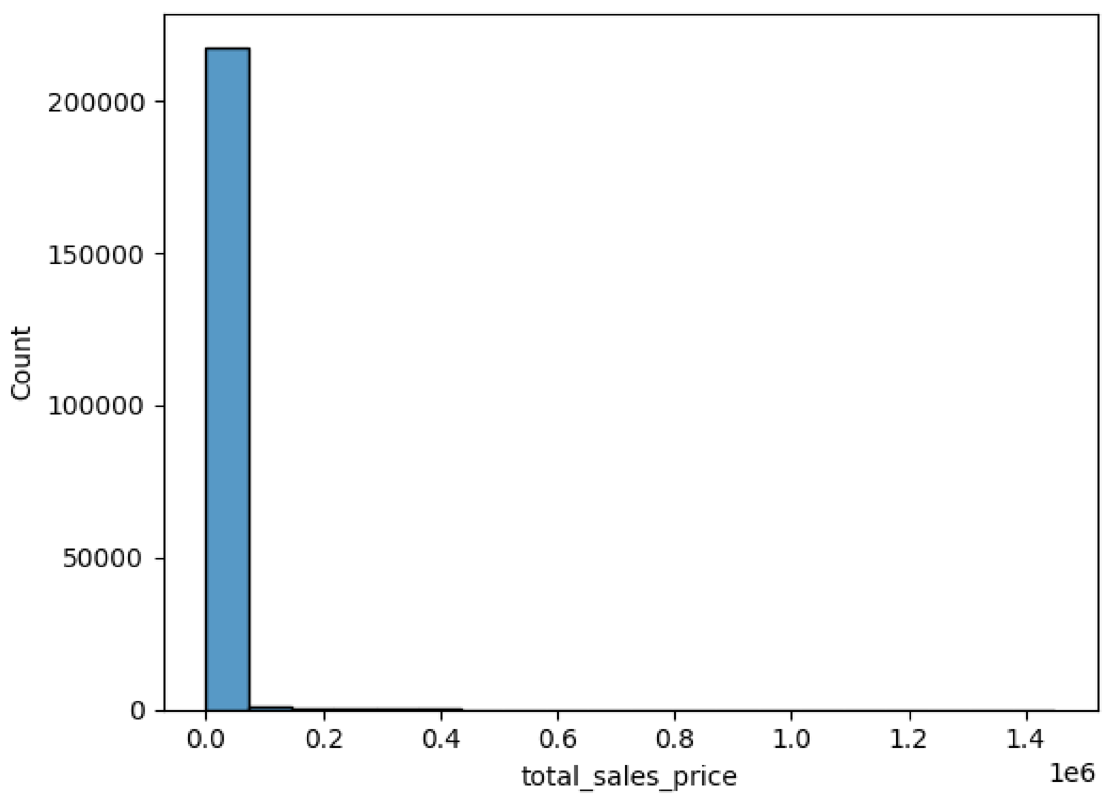
0	59.0	2.85	2.87	1.79	200
1	59.0	2.84	2.89	1.78	200
2	59.0	2.88	2.90	1.77	200
3	59.0	2.86	2.88	1.78	200
4	58.5	2.79	2.83	1.82	200
5	57.0	2.95	2.99	1.81	200
6	57.0	2.85	2.88	1.84	200
7	59.5	2.86	2.89	1.78	200
8	59.5	2.89	2.92	1.85	200
9	57.0	2.83	2.87	1.80	200

3 Exploratory Data Analysis

Plotting the graphs to get insights of data how can we make model

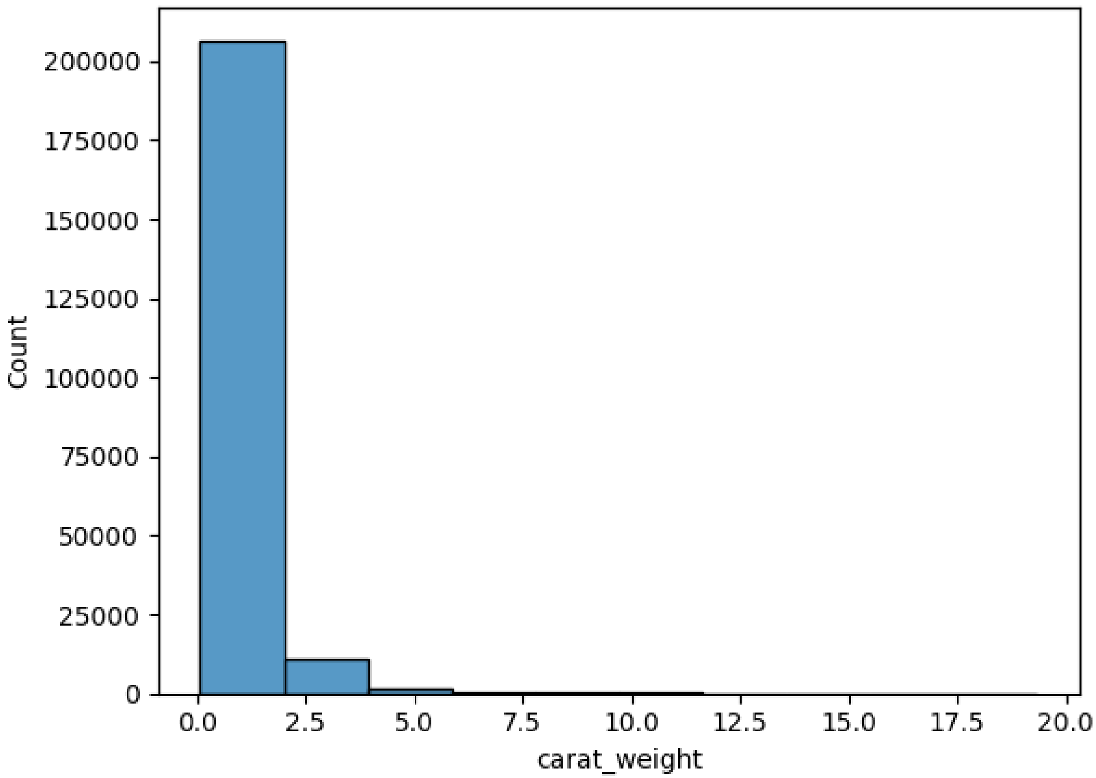
[26]:
```
sns.histplot(diamond['total_sales_price'], bins=20)
```

[26]: <AxesSubplot: xlabel='total_sales_price', ylabel='Count'>



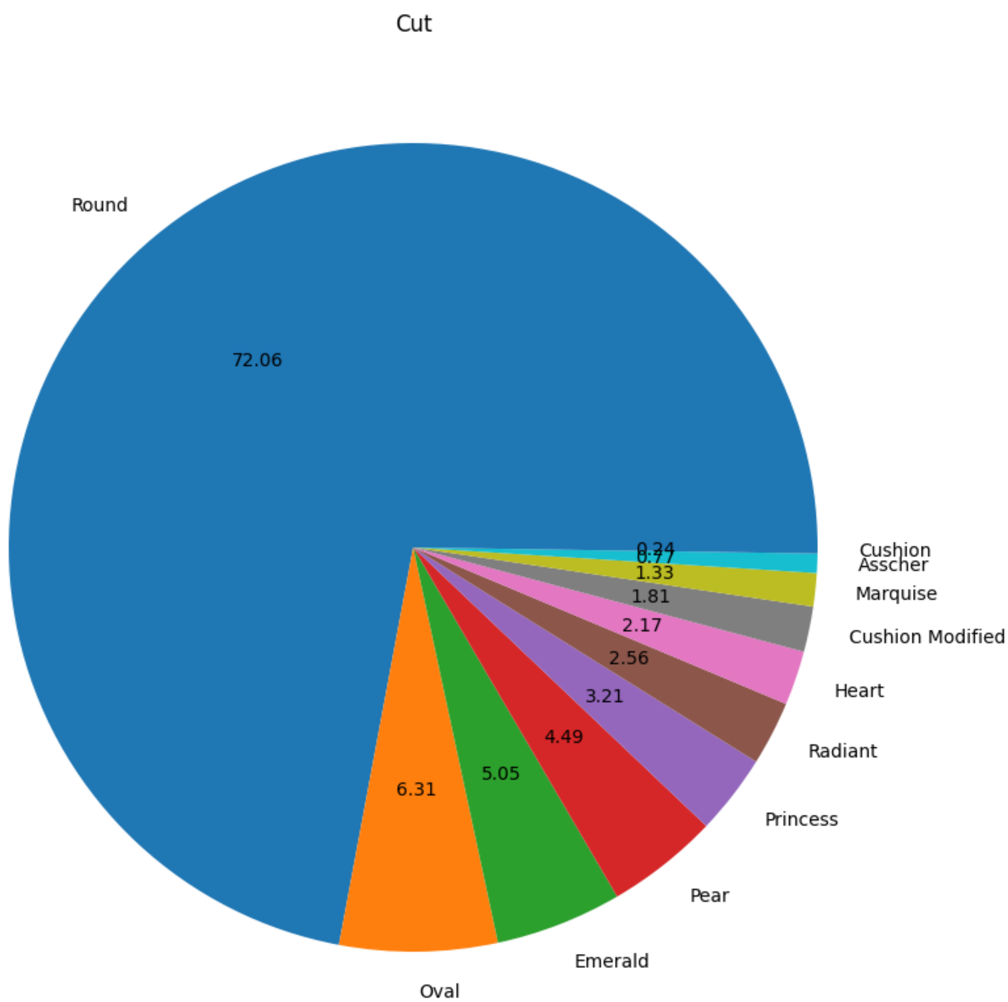
[27]:
```
sns.histplot(diamond['carat_weight'], bins=10)
```

[27]: <AxesSubplot: xlabel='carat_weight', ylabel='Count'>

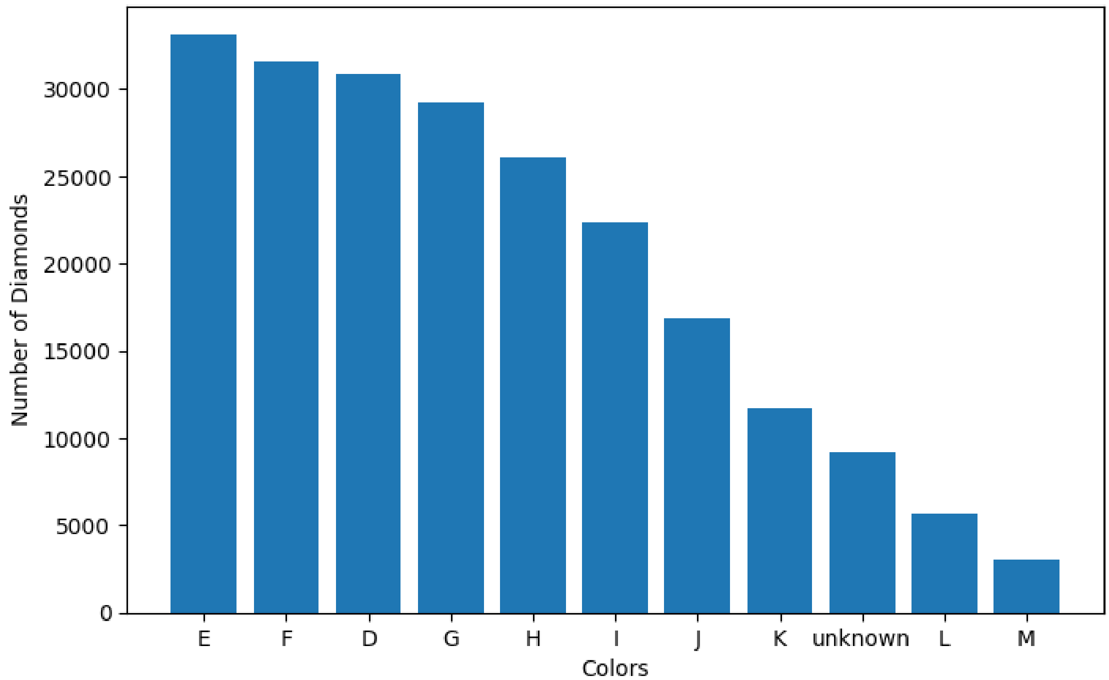


By this graph we can see that most of the diamonds are less then 2.5 Carat in weight

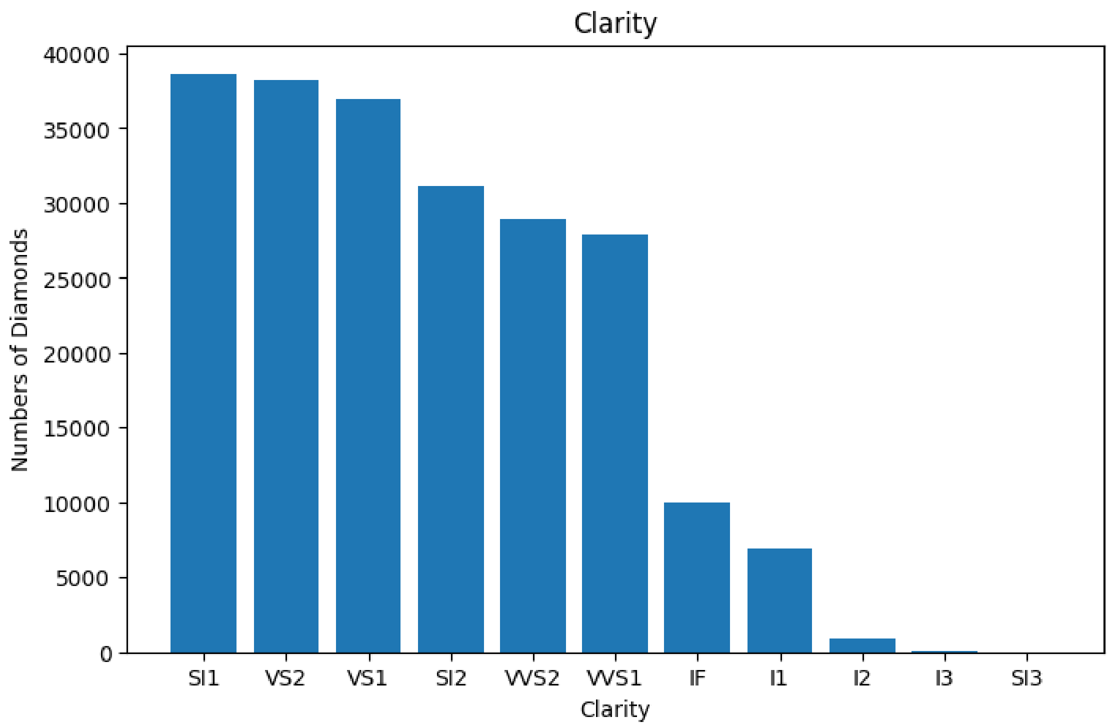
[28]:
```
plt.figure(figsize = (10, 10))
plt.pie(diamond['cut'].value_counts(), labels=['Round', 'Oval', 'Emerald',␣
 ↪'Pear', 'Princess', 'Radiant', 'Heart', 'Cushion Modified', 'Marquise',␣
 ↪'Asscher', 'Cushion'], autopct="%0.2f")
plt.title('Cut')
plt.show()
```



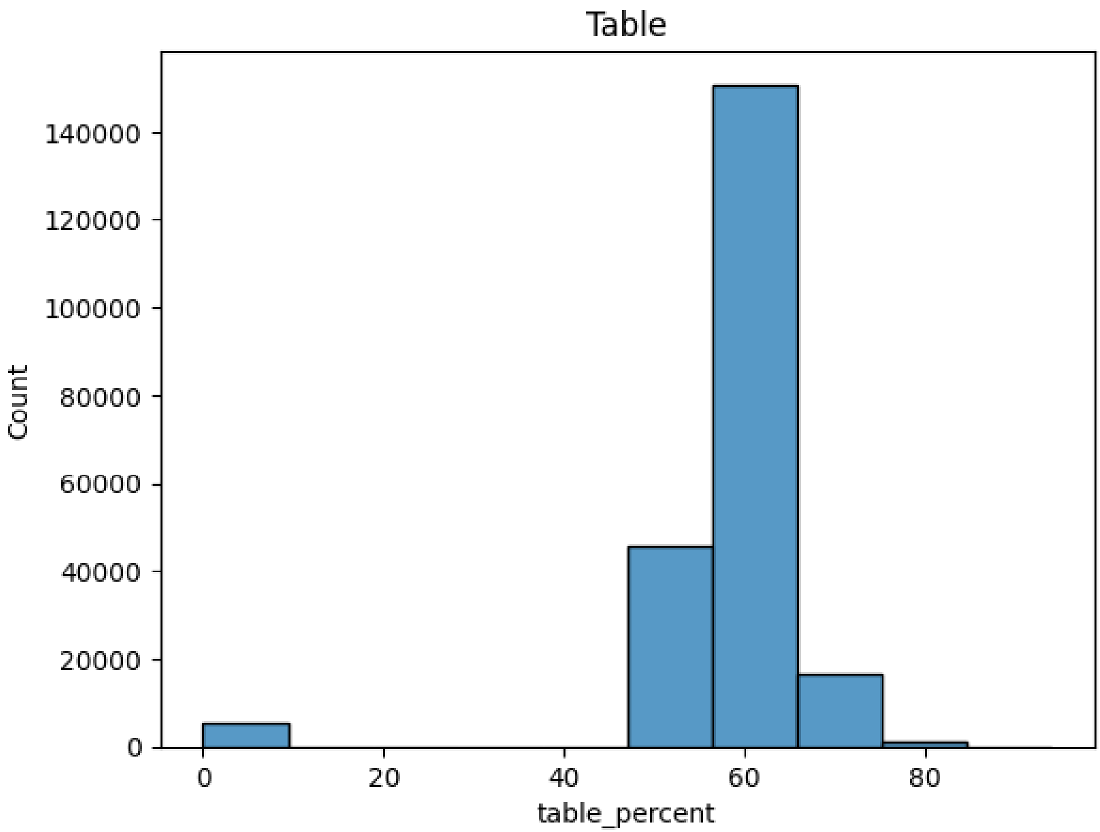
[29]:
```
plt.figure(figsize=(8, 5))
plt.bar(diamond['color'].value_counts().index, diamond['color'].value_counts())
plt.ylabel("Number of Diamonds")
plt.xlabel("Colors")
plt.show()
```



[30]:
```
plt.figure(figsize=(8,5))
plt.bar(diamond['clarity'].value_counts().index, diamond['clarity'].
    ↪value_counts())
plt.title("Clarity")
plt.ylabel("Numbers of Diamonds")
plt.xlabel("Clarity")
plt.show()
```



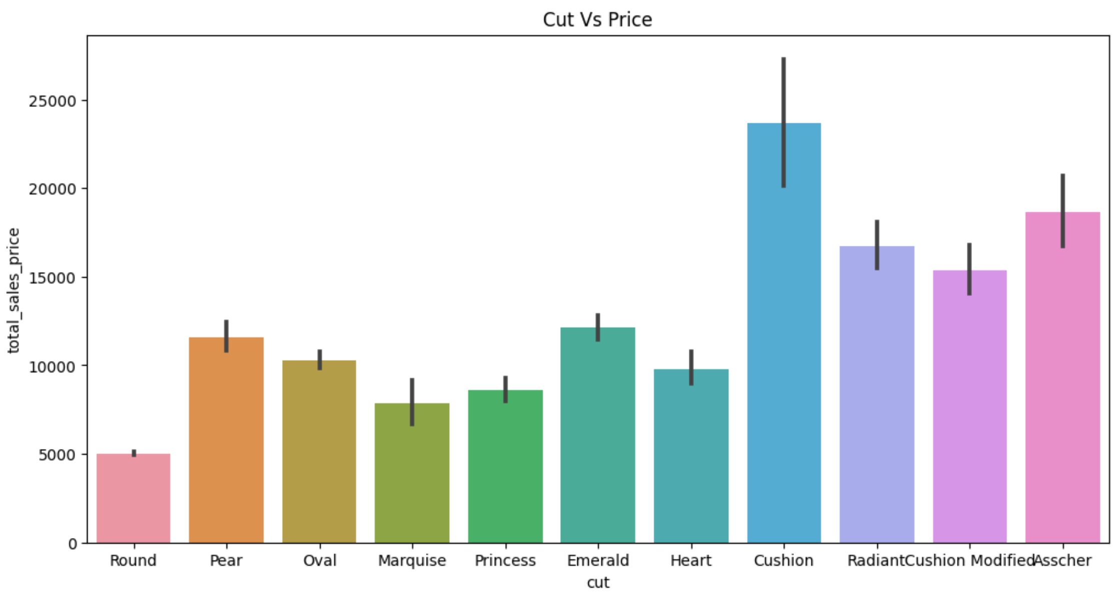
[31]:
```
sns.histplot(diamond['table_percent'], bins=10)
plt.title("Table")
plt.show()
```



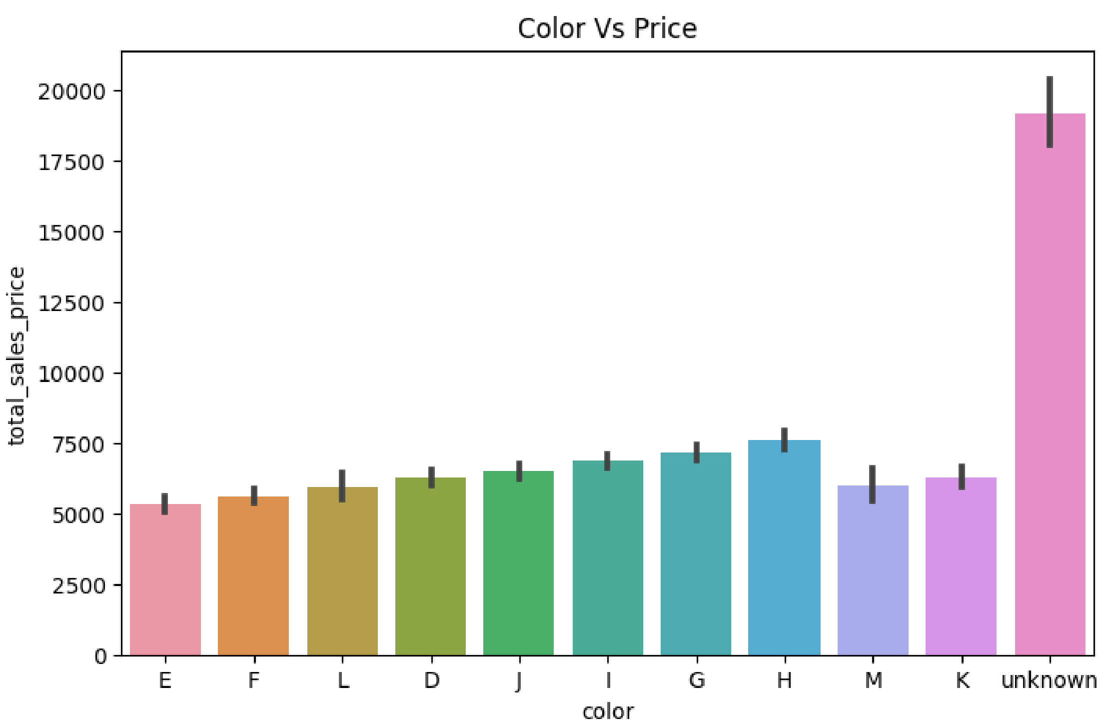
4 Comparing Diamond's Features with Price

[32]:

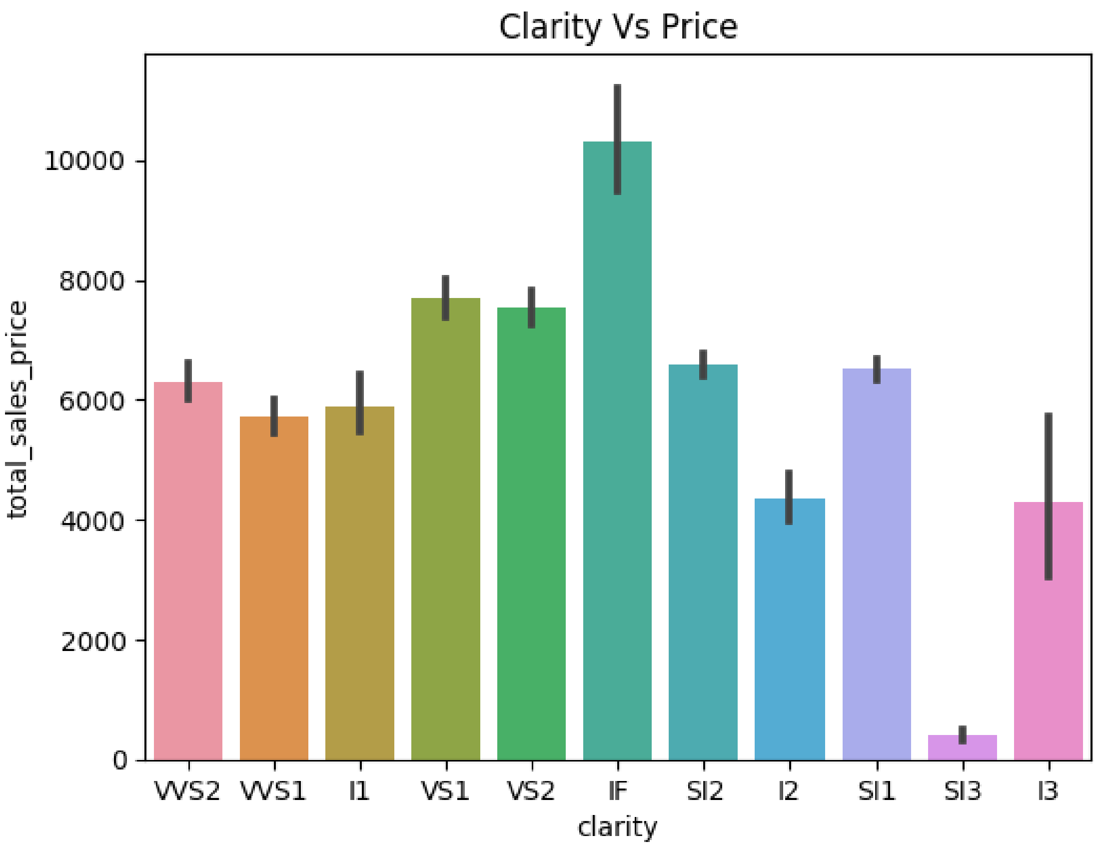
```
plt.figure(figsize=(12,6))
sns.barplot(x='cut', y='total_sales_price', data=diamond)
plt.title("Cut Vs Price")
plt.show()
```



[33]:
```
plt.figure(figsize=(8,5))
sns.barplot(x='color', y='total_sales_price', data=diamond)
plt.title("Color Vs Price")
plt.show()
```



[34]:
```
sns.barplot(x='clarity', y='total_sales_price', data=diamond)
plt.title("Clarity Vs Price")
plt.show()
```



M color and I3 clarity are worst features of diamond, however when the data is plotted on bar graph, it is seen that the price of M color and I3 is lower than the price of diamond of D color and IF clarity as they are the best feature of diamonds

5 Data Preprocessing to change categorical to numeric

[35]:
```
diamond['cut'] = diamond['cut'].map({'Round':11,'Oval':9,'Emerald':10, 'Pear':
 ↪8,'Princess':7,'Radiant':5,'Heart':3, 'Cushion Modified':1, 'Marquise':
 ↪2,'Asscher':4,'Cushion':6})
diamond['color'] = diamond['color'].map({'E':9, 'F':8, 'D':10, 'G':7, 'H':6,␣
 ↪'I':5, 'J':4, 'K':3, 'unknown':11, 'L':2,'M':1})
diamond['clarity'] = diamond['clarity'].map({'SI1':6, 'VS2':7,'VS1':8,'SI2':
 ↪5,'VVS2':9,'VVS1':10,'IF':11,'I1':3,'I2':2,'I3':1, 'SI3':4})
```

```
diamond['cut_quality'] = diamond['cut_quality'].map({'Excellent':5,'unknown':
↪0,'Very Good':4,'Good':3, 'Fair':2,'Ideal':1})
diamond['symmetry'] = diamond['symmetry'].map({'Excellent':4,'Very Good':
↪4,'Good':3, 'Fair':2,'Poor':1})
diamond['polish'] = diamond['polish'].map({'Excellent':5,'Very Good':4,'Good':
↪3, 'Fair':2,'Poor':1})
```

[36]:
```
diamond.drop('lab', axis=1, inplace=True)
```

[37]:
```
diamond.head()
```

[37]:
```
   cut  color  clarity  carat_weight  cut_quality  symmetry  polish  \
0   11      9        9          0.09            5         4       4
1   11      9        9          0.09            4         4       4
2   11      9        9          0.09            5         4       4
3   11      9        9          0.09            5         4       4
4   11      9        9          0.09            4         4       5

   table_percent  meas_length  meas_width  meas_depth  total_sales_price
0           59.0         2.85        2.87        1.79                200
1           59.0         2.84        2.89        1.78                200
2           59.0         2.88        2.90        1.77                200
3           59.0         2.86        2.88        1.78                200
4           58.5         2.79        2.83        1.82                200
```

6 Correlation

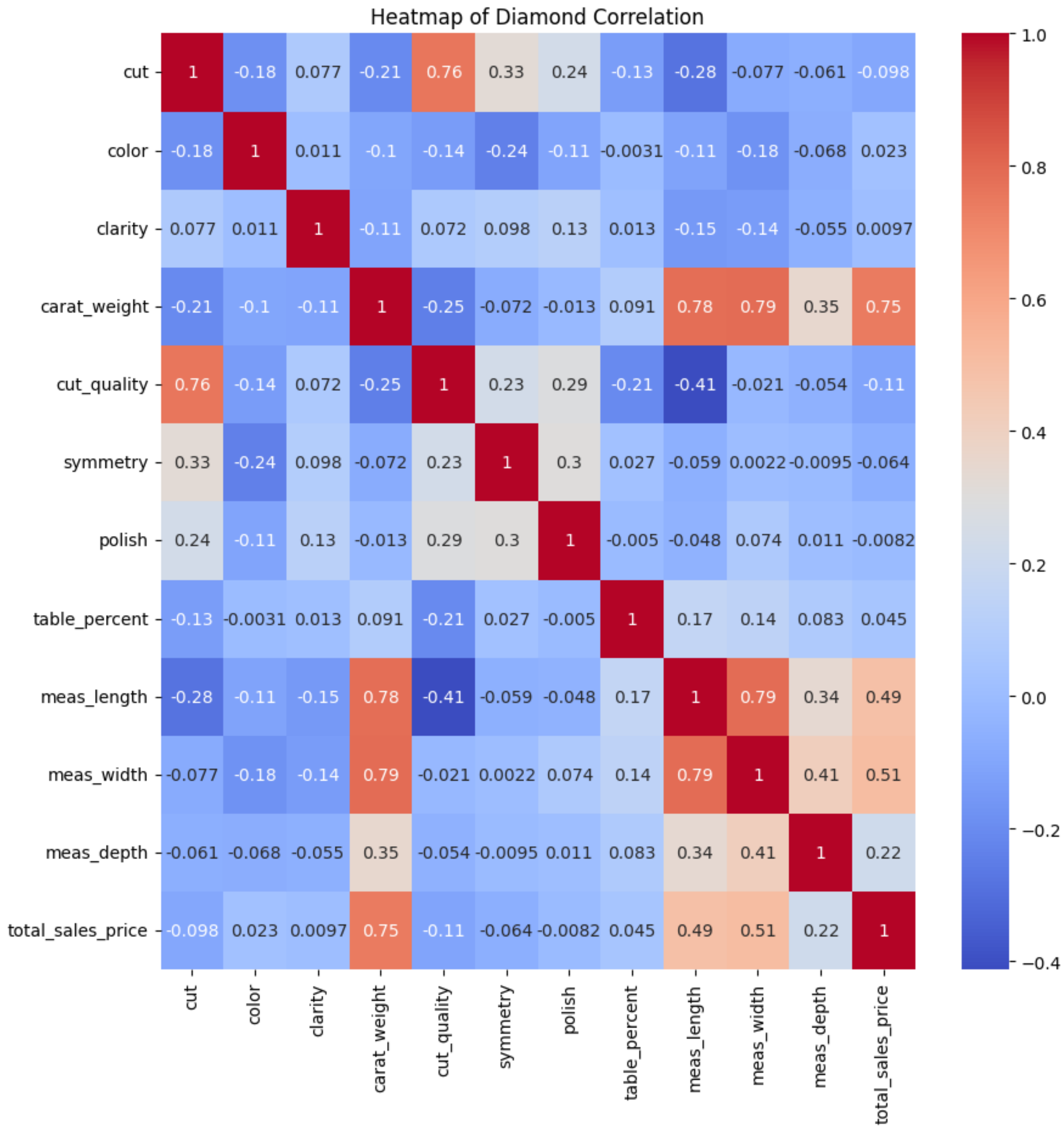
[38]:
```
diamond.corr()
```

[38]:
```
                        cut     color   clarity  carat_weight  cut_quality  \
cut                1.000000 -0.184921  0.076725     -0.208087     0.757709
color             -0.184921  1.000000  0.010645     -0.102915    -0.141346
clarity            0.076725  0.010645  1.000000     -0.105747     0.072141
carat_weight      -0.208087 -0.102915 -0.105747      1.000000    -0.247205
cut_quality        0.757709 -0.141346  0.072141     -0.247205     1.000000
symmetry           0.326102 -0.239464  0.097869     -0.071656     0.234340
polish             0.237122 -0.107078  0.126664     -0.012959     0.286634
table_percent     -0.134410 -0.003085  0.012751      0.090697    -0.208497
meas_length       -0.284844 -0.112455 -0.153244      0.782683    -0.413258
meas_width        -0.077052 -0.179592 -0.139439      0.788912    -0.020557
meas_depth        -0.061145 -0.068130 -0.055404      0.350719    -0.054241
total_sales_price -0.098456  0.023286  0.009665      0.745963    -0.107072

                   symmetry    polish  table_percent  meas_length  meas_width  \
cut                0.326102  0.237122      -0.134410    -0.284844   -0.077052
color             -0.239464 -0.107078      -0.003085    -0.112455   -0.179592
clarity            0.097869  0.126664       0.012751    -0.153244   -0.139439
```

```
carat_weight      -0.071656 -0.012959       0.090697      0.782683     0.788912
cut_quality        0.234340  0.286634      -0.208497     -0.413258    -0.020557
symmetry           1.000000  0.301732       0.027208     -0.058804     0.002165
polish             0.301732  1.000000      -0.004962     -0.048063     0.073572
table_percent      0.027208 -0.004962       1.000000      0.165742     0.141250
meas_length       -0.058804 -0.048063       0.165742      1.000000     0.788652
meas_width         0.002165  0.073572       0.141250      0.788652     1.000000
meas_depth        -0.009468  0.011241       0.082533      0.342209     0.412933
total_sales_price -0.063568 -0.008245       0.045192      0.489218     0.506403

                   meas_depth  total_sales_price
cut                 -0.061145          -0.098456
color               -0.068130           0.023286
clarity             -0.055404           0.009665
carat_weight         0.350719           0.745963
cut_quality         -0.054241          -0.107072
symmetry            -0.009468          -0.063568
polish               0.011241          -0.008245
table_percent        0.082533           0.045192
meas_length          0.342209           0.489218
meas_width           0.412933           0.506403
meas_depth           1.000000           0.216410
total_sales_price    0.216410           1.000000
```

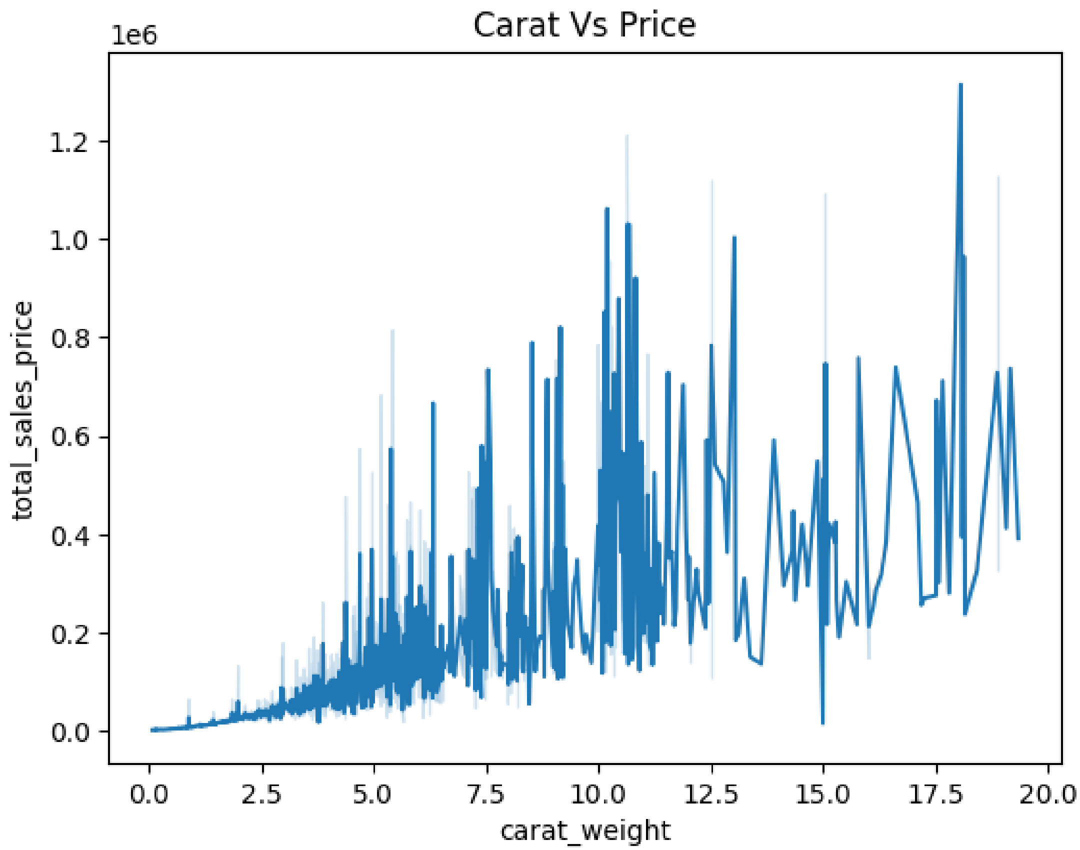
6.0.1 Plotting Correlation Heatmap

[39]:
```
plt.figure(figsize=(10, 10))
sns.heatmap(diamond.corr(), annot=True, cmap='coolwarm')
plt.title("Heatmap of Diamond Correlation")
plt.show()
```



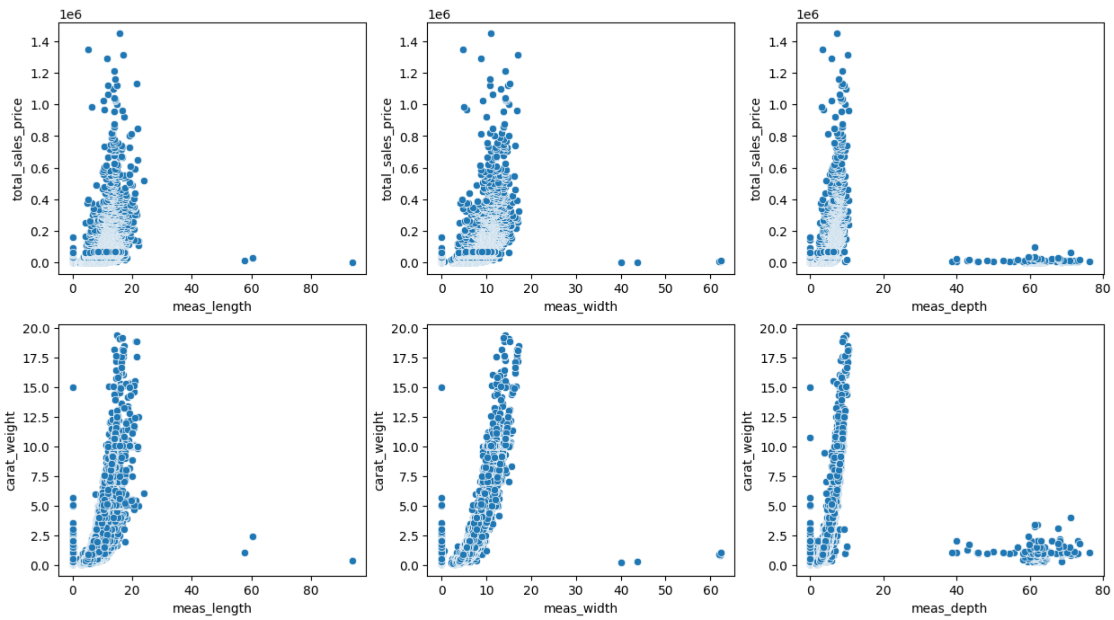
6.0.2 Plotting the relationship between Carat and Price

[40]:
```
sns.lineplot(x = 'carat_weight', y = 'total_sales_price', data=diamond)
plt.title("Carat Vs Price")
plt.show()
```



We can see that the price of diamond and not constant, it increasing with carat of the diamonds. We can see that diamonds with high carats are also low prices that is because of the other factors that affects the price of the diamond

[41]:
```
fig, ax = plt.subplots(2, 3, figsize=(15, 8))
sns.scatterplot(x='meas_length', y='total_sales_price', data=diamond,
    ax=ax[0,0])
sns.scatterplot(x='meas_width', y='total_sales_price', data=diamond, ax=ax[0,1])
sns.scatterplot(x='meas_depth', y='total_sales_price', data=diamond, ax=ax[0,2])
sns.scatterplot(x='meas_length', y='carat_weight', data=diamond, ax=ax[1,0])
sns.scatterplot(x='meas_width', y='carat_weight', data=diamond, ax=ax[1,1])
sns.scatterplot(x='meas_depth', y='carat_weight', data=diamond, ax=ax[1,2])
plt.show()
```



Majority of diamonds length is between 0 to 20, width is between 0 to 15 and depth is between 0 to 10 with other dimensions are very rare.

7 Train Test Split

[42]:
```python
from sklearn.model_selection import train_test_split
```

[43]:
```python
x_train,x_test,y_train,y_test=train_test_split(diamond.
↪drop('total_sales_price', axis=1), diamond['total_sales_price'], test_size=0.
↪2)
```

[44]:
```python
len(x_train)
```

[44]: 175762

[45]:
```python
len(x_test)
```

[45]: 43941

8 Model Building

8.0.1 Decision Tree Regressor

[46]:
```python
from sklearn.tree import DecisionTreeRegressor
```

[47]:
```python
dt = DecisionTreeRegressor()
```

[48]:
```
dt
```

[48]: DecisionTreeRegressor()

[49]:
```
dt.fit(x_train,y_train)
dt.score(x_train,y_train)
```

[49]: 0.9999642390601138

[50]:
```
dt_predict = dt.predict(x_test)
```

8.0.2 Random Forest Regressor

[51]:
```
from sklearn.ensemble import RandomForestRegressor
```

[52]:
```
rf = RandomForestRegressor()
```

[53]:
```
rf
```

[53]: RandomForestRegressor()

[54]:
```
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
```

[54]: 0.9774290220393043

[55]:
```
rf_predict = rf.predict(x_test)
```

9 Model Evalution

[56]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

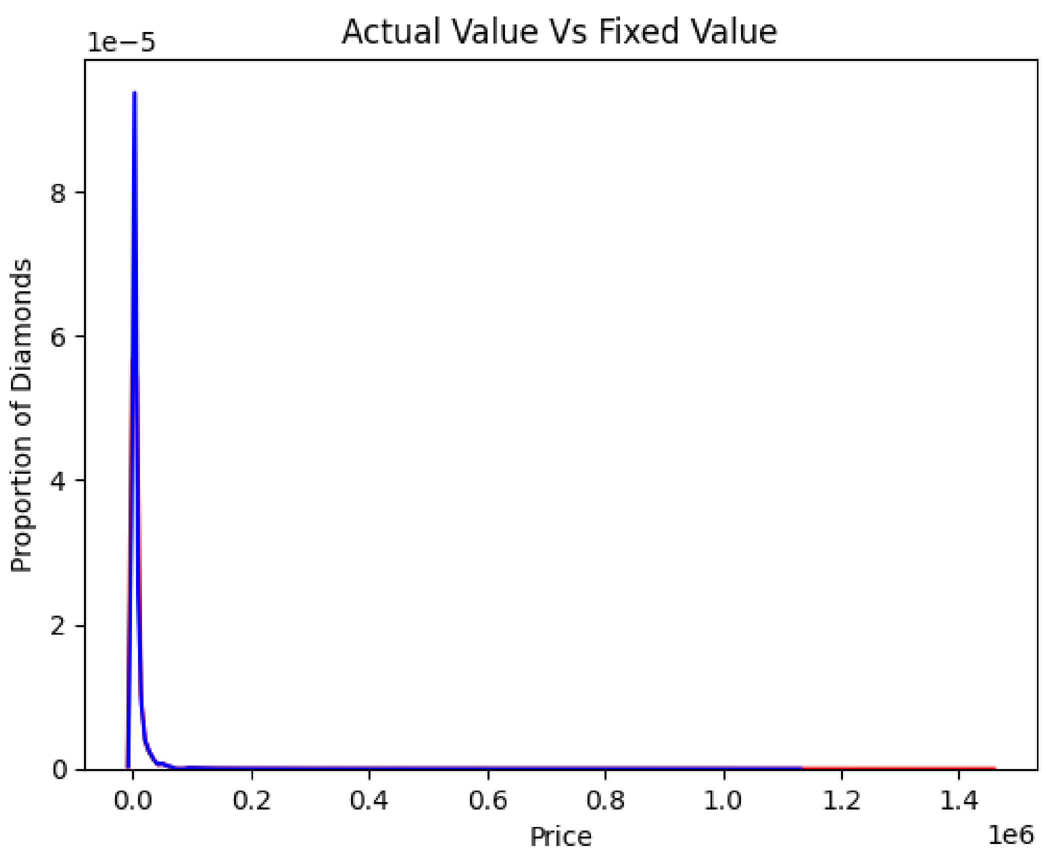
9.0.1 Decision Tree Regressor

Distribution plot between actual value and predicated value

[57]:
```
ax = sns.distplot(y_test,hist=False,color='r',label='Actual Value')
sns.distplot(dt_predict,hist=False,color='b',label='Fixed Value', ax=ax)
plt.title("Actual Value Vs Fixed Value")
plt.xlabel("Price")
plt.ylabel("Proportion of Diamonds")
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_19800\1158408235.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test,hist=False,color='r',label='Actual Value')
C:\Users\Admin\AppData\Local\Temp\ipykernel_19800\1158408235.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dt_predict,hist=False,color='b',label='Fixed Value', ax=ax)
```



```
[58]: print("Decision Tree Regressor RMSE:", np.
       ↪sqrt(mean_squared_error(y_test,dt_predict)))
      print("Decision Tree Regressor Accuracy:", dt.score(x_test,y_test))
      print("Decision Tree Regressor RME:", mean_absolute_error(y_test,dt_predict))
```

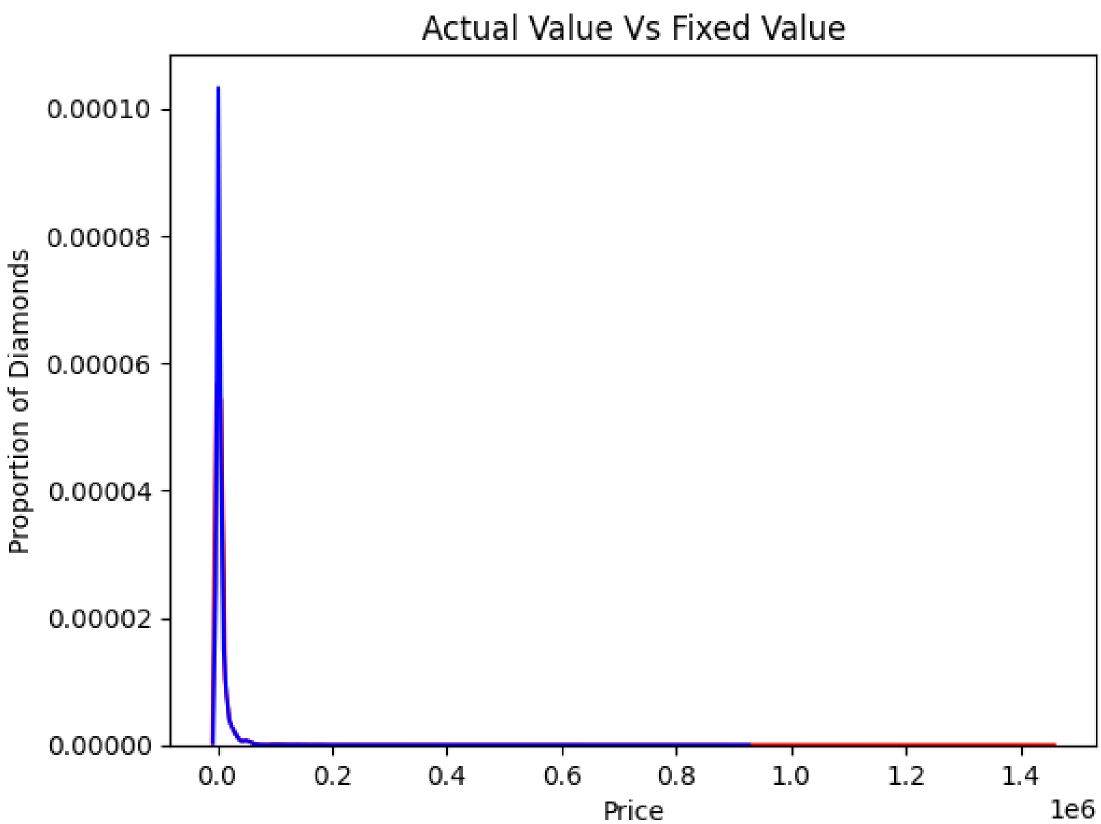
```
Decision Tree Regressor RMSE: 13394.731913001424
Decision Tree Regressor Accuracy: 0.7283413226916249
Decision Tree Regressor RME: 1561.3440225583877
```

9.0.2 Random Forest Regressor

Distribution plot between actual value and predicated value

```
[59]: ax = sns.distplot(y_test,hist=False,color='r',label="Actual Value")
      sns.distplot(rf_predict, hist=False,color='b',label="Fixed Value", ax=ax)
      plt.title("Actual Value Vs Fixed Value")
      plt.xlabel("Price")
      plt.ylabel("Proportion of Diamonds")
      plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_19800\399982518.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test,hist=False,color='r',label="Actual Value")
C:\Users\Admin\AppData\Local\Temp\ipykernel_19800\399982518.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density
plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(rf_predict, hist=False,color='b',label="Fixed Value", ax=ax)
```



[60]:
```
print("Random Forest Regressor RMSE:", np.
 ↪sqrt(mean_squared_error(y_test,rf_predict)))
print("Random Forest Regressor Accuracy:", rf.score(x_test,y_test))
print("Random Forest Regressor RME:", mean_absolute_error(y_test,rf_predict))
```

```
Random Forest Regressor RMSE: 10343.146227096493
Random Forest Regressor Accuracy: 0.8380201997008871
Random Forest Regressor RME: 1239.4787007140194
```

10 Predication

[62]:
```
def predication():
    carat_weight = input("Enter the value of Carat from 0.00 to 20.00: ")
    print("'Round':11,'Oval':9,'Emerald':10, 'Pear':8,'Princess':7,'Radiant':
 ↪5,'Heart':3, 'Cushion Modified':1, 'Marquise':2,'Asscher':4,'Cushion':6")
    cut  = input("Enter the Cut Value from Above List: ")
    print("'SI1':6, 'VS2':7,'VS1':8,'SI2':5,'VVS2':9,'VVS1':10,'IF':11,'I1':
 ↪3,'I2':2,'I3':1, 'SI3':4")
    clarity  = input("Enter the Clarity Value from Above List: ")
```

```python
    print("'E':9, 'F':8, 'D':10, 'G':7, 'H':6, 'I':5, 'J':4, 'K':3, 'unknown':
 ↪11, 'L':2,'M':1")
    color  = input("Enter the Color Value from Above List: ")
    print("'Excellent':5,'unknown':0,'Very Good':4,'Good':3, 'Fair':2,'Ideal':
 ↪1")
    cut_quality  = input("Enter the Cut Quality Value from Above List: ")
    print("'Excellent':4,'Very Good':4,'Good':3, 'Fair':2,'Poor':1")
    symmetry  = input("Enter the Symmetry Value from Above List: ")
    print("'Excellent':5,'Very Good':4,'Good':3, 'Fair':2,'Poor':1")
    polish  = input("Enter the Polish Value from Above List: ")
    table_percent  = input("Enter the Table Percent Value: ")
    meas_length  = input("Enter the Lenth Value: ")
    meas_width  = input("Enter the Width Value: ")
    meas_depth  = input("Enter the Depth Value: ")

    price = rf.predict([[cut, color, clarity, carat_weight, cut_quality,␣
 ↪symmetry, polish, table_percent, meas_length, meas_width, meas_depth]])

    print("Approximately Price of Diamond is: $ ", price)

predic = predication()

predic
```

```
Enter the value of Carat from 0.00 to 20.00: 0.09
'Round':11,'Oval':9,'Emerald':10, 'Pear':8,'Princess':7,'Radiant':5,'Heart':3,
'Cushion Modified':1, 'Marquise':2,'Asscher':4,'Cushion':6
Enter the Cut Value from Above List: 11
'SI1':6,
'VS2':7,'VS1':8,'SI2':5,'VVS2':9,'VVS1':10,'IF':11,'I1':3,'I2':2,'I3':1, 'SI3':4
Enter the Clarity Value from Above List: 9
'E':9, 'F':8, 'D':10, 'G':7, 'H':6, 'I':5, 'J':4, 'K':3, 'unknown':11,
'L':2,'M':1
Enter the Color Value from Above List: 9
'Excellent':5,'unknown':0,'Very Good':4,'Good':3, 'Fair':2,'Ideal':1
Enter the Cut Quality Value from Above List: 5
'Excellent':4,'Very Good':4,'Good':3, 'Fair':2,'Poor':1
Enter the Symmetry Value from Above List: 4
'Excellent':5,'Very Good':4,'Good':3, 'Fair':2,'Poor':1
Enter the Polish Value from Above List: 4
Enter the Table Percent Value: 59
Enter the Lenth Value: 2.85
Enter the Width Value: 2.87
Enter the Depth Value: 1.79
Approximately Price of Diamond is: $  [200.28]
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-
packages\sklearn\base.py:450: UserWarning: X does not have valid feature names,
```

```
but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

11 Conclusion

Both the model have very much different in accuracy. However the Random Forest Regressor is more better and accurate than the Decision Tree Regressor.