What is Diamonds Prices Dataset?

This document explores a dataset containing prices and attributes for approximately 54,000 round-cut diamonds. There are 53,940 diamonds in the dataset with 10 features (carat, cut, color, clarity, depth, table, price, x, y, and z). Most variables are numeric in nature, but the variables cut, color, and clarity are ordered factor variables with the following levels.#

About the currency for the price column: it is Price ($)

And About the columns x,y, and z they are diamond measurements as ((x: length in mm, y: width in mm,z: depth in mm))

In [ ]:
```
# There are 10 columns
#carat:Weight of the diamond
#cut:Quality of the cut (Fair, Good, Very Good, Premium, Ideal)
#color:Diamond colour, from J (worst) to D (best)
#clarity:How clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))
#x:Length in mm
#y:Width in mm
#z:Depth in mm
#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)
#price:Price in US dollars (326--18,823)
```

importing the libraries

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

In [4]:
```
df = pd.read_csv("D:/Datasets/Diamonds Prices2022.csv")
df.head()
```

Out[4]:

	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

Data Preprocessing

In [5]:
```
df.shape
```

Out[5]:
```
(53943, 11)
```

In [6]:
```
#checking for null values
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53943 entries, 0 to 53942
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53943 non-null  int64
 1   carat       53943 non-null  float64
 2   cut         53943 non-null  object
 3   color       53943 non-null  object
 4   clarity     53943 non-null  object
 5   depth       53943 non-null  float64
 6   table       53943 non-null  float64
 7   price       53943 non-null  int64
 8   x           53943 non-null  float64
 9   y           53943 non-null  float64
 10  z           53943 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

In [7]:
```
#checking discrptive Values
df.describe()
```

Out[7]:

	Unnamed: 0	carat	depth	table	price	x	y	z
count	53943.000000	53943.000000	53943.000000	53943.000000	53943.000000	53943.000000	53943.000000	53943.000000
mean	26972.000000	0.797935	61.749322	57.457251	3932.734294	5.731158	5.734526	3.538730
std	15572.147122	0.473999	1.432626	2.234549	3989.338447	1.121730	1.142103	0.705679
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	13486.500000	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	26972.000000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	40457.500000	1.040000	62.500000	59.000000	5324.000000	6.540000	6.540000	4.040000
max	53943.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

In [9]:
```
#values count for categorical values
df["cut"].value_counts()
```

Out[9]:
```
Ideal        21551
Premium      13793
Very Good    12083
Good          4906
Fair          1610
Name: cut, dtype: int64
```

In [10]:
```
df["color"].value_counts()
```

Out[10]:
```
G    11292
E     9799
F     9543
H     8304
D     6775
I     5422
J     2808
Name: color, dtype: int64
```

In [11]:
```
df["clarity"].value_counts()
```

Out[11]:
```
SI1     13067
VS2     12259
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: clarity, dtype: int64
```

In [12]:
```
df.head(10)
```

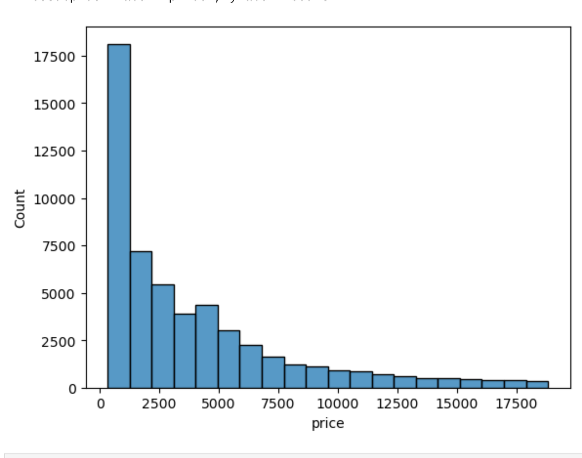
Out[12]:

	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
5	6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
6	7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
7	8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
8	9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
9	10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

Exploratory Data Analysis

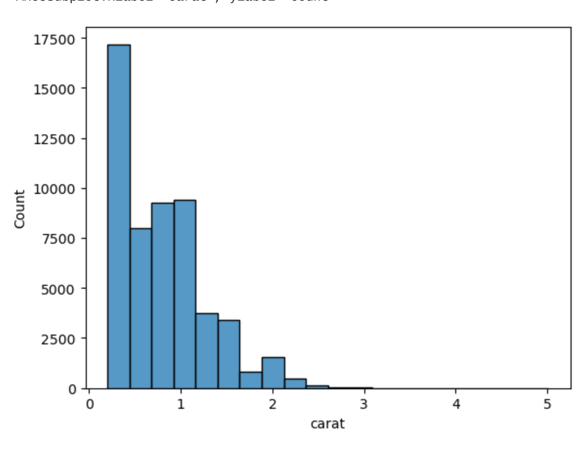
In [13]:
```
sns.histplot(df['price'],bins=20)
```

Out[13]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



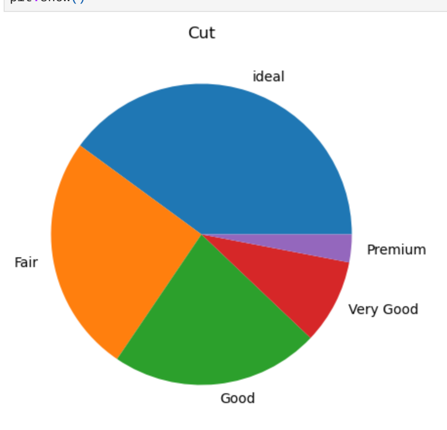
In [14]:
```
sns.histplot(df['carat'],bins=20)
```

Out[14]:
```
<AxesSubplot:xlabel='carat', ylabel='Count'>
```

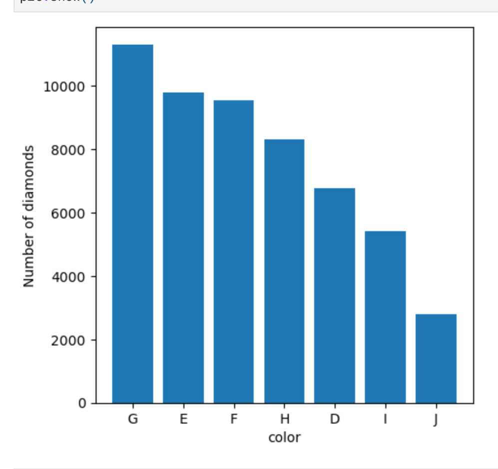


Most of the diamonds are less than 1 carat in weight

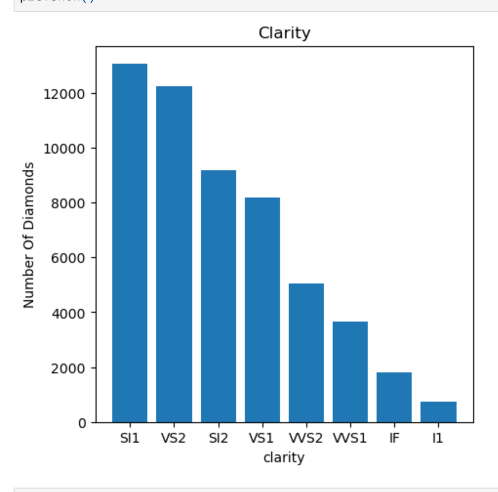
In [16]:
```
plt.figure(figsize=(5,5))
plt.pie(df["cut"].value_counts(),labels=["ideal","Fair","Good","Very Good","Premium"])
plt.title("Cut")
plt.show()
```



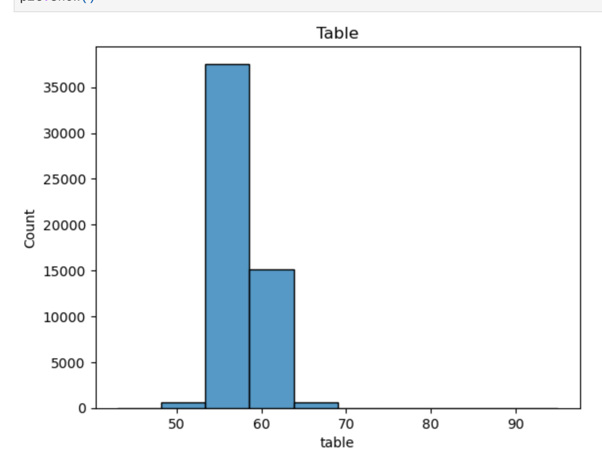
In [17]:
```
plt.figure(figsize=(5,5))
plt.bar(df["color"].value_counts().index,df["color"].value_counts())
plt.ylabel("Number of diamonds")
plt.xlabel("color")
plt.show()
```



In [18]:
```
plt.figure(figsize=(5,5))
plt.bar(df["clarity"].value_counts().index,df["clarity"].value_counts())
plt.title("Clarity")
plt.ylabel("Number Of Diamonds")
plt.xlabel("clarity")
plt.show()
```



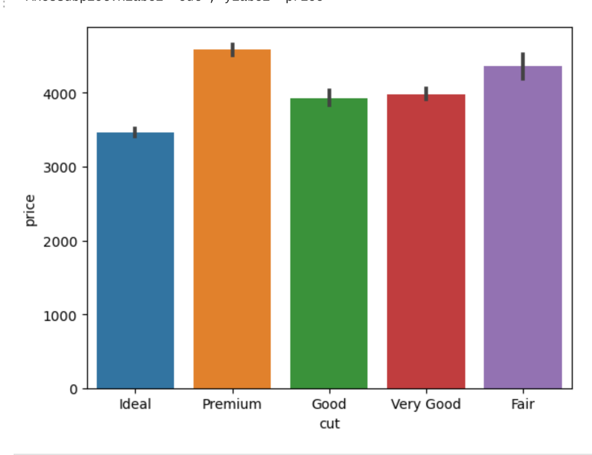
In [19]:
```
sns.histplot(df["table"],bins=10)
plt.title("Table")
plt.show()
```



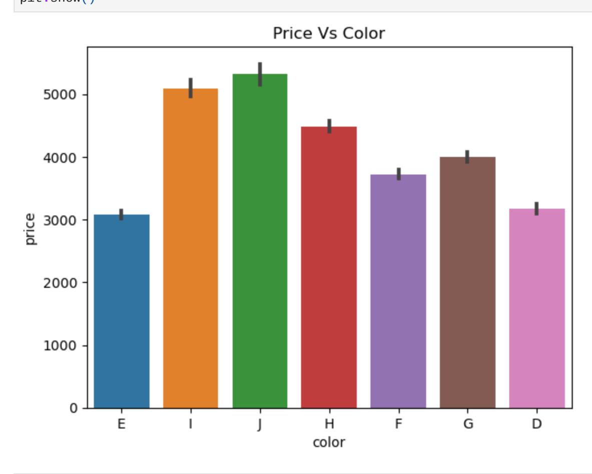
Compairing Diamond's Features With Price

In [20]:
```
sns.barplot(x="cut",y="price",data=df)
```

Out[20]:
```
<AxesSubplot:xlabel='cut', ylabel='price'>
```

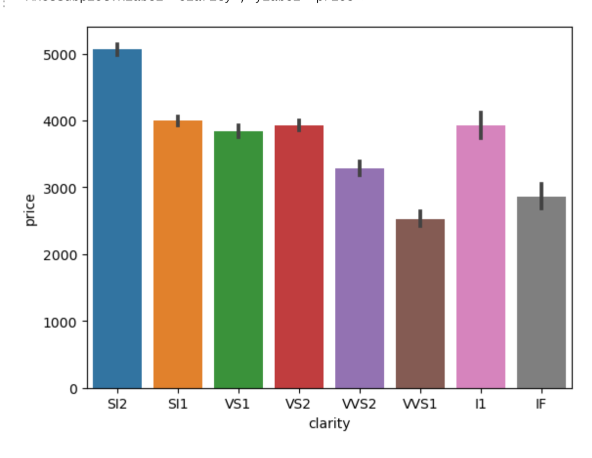


In [21]:
```
sns.barplot(x="color",y="price",data=df)
plt.title("Price Vs Color")
plt.show()
```



In [22]:
```
sns.barplot(x="clarity", y="price",data=df)
```

Out[22]:
```
<AxesSubplot:xlabel='clarity', ylabel='price'>
```



j color and I1 color are the worst features of the diamond,however when the data is plotted onthe bar graph.it seen the price of diamonds with J and I1 clarity is higher.

Than the price of diamonds of D color and IF clarity,which is opposite to each other what i expected

Data Preprocessing2

In [24]:
```
df["cut"] = df["cut"].map({"Ideal":5,"Premium":4,"Very Good":3,"Good":2,"Fair":1})
df["color"] = df["color"].map({"D":7,"E":6,"F":5,"G":4,"H":3,"I":2,"J":1})
df["clarity"] = df["clarity"].map({"IF":8,"VVS1":7,"VVS2":6,"VS1":5,"VS2":4,"SI1":3,"SI2":2,"I1":1})
```

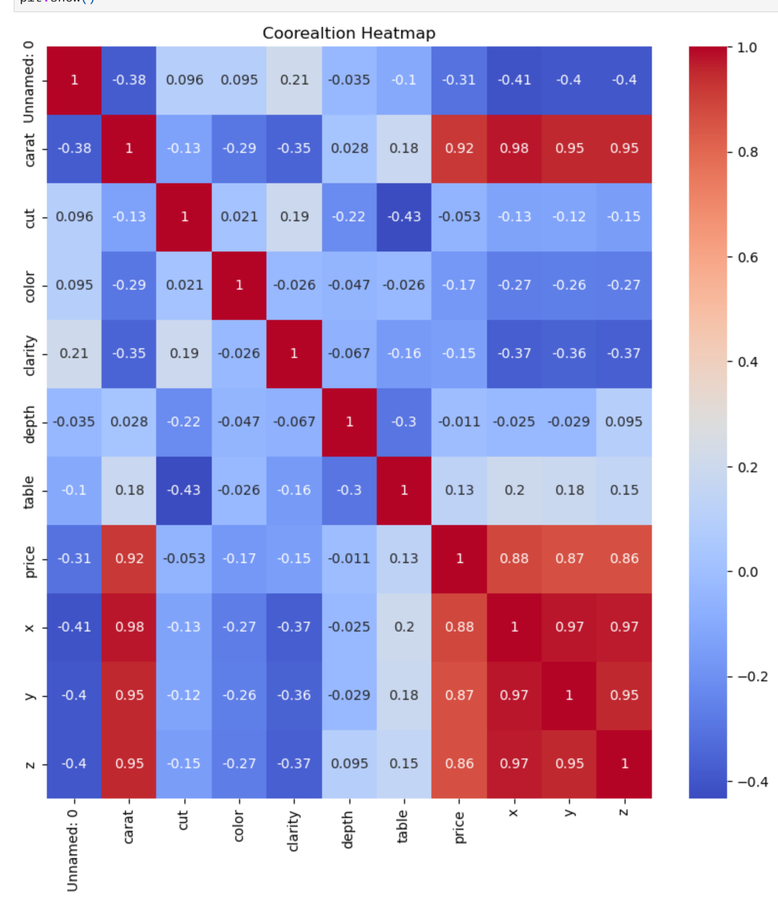
Coorealation

In [25]:
```
#coorealtion matrix
df.corr()
```

Out[25]:

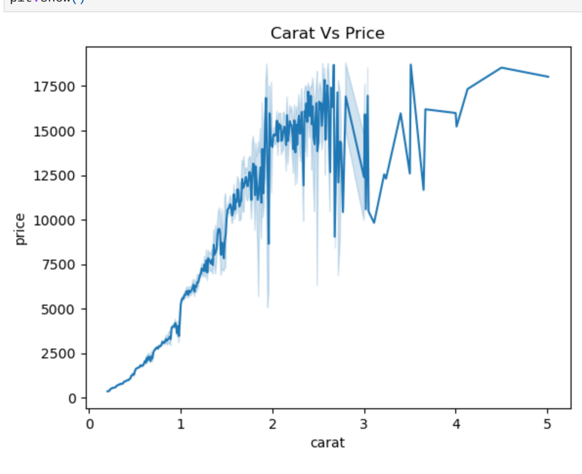
	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
Unnamed: 0	1.000000	-0.377970	0.096462	0.095160	0.208538	-0.034896	-0.100764	-0.306875	-0.405405	-0.395809	-0.399185
carat	-0.377970	1.000000	-0.134964	-0.291439	-0.352833	0.028234	0.181602	0.921591	0.975093	0.951721	0.953387
cut	0.096462	-0.134964	1.000000	0.020506	0.189171	-0.218037	-0.433387	-0.053487	-0.125564	-0.121461	-0.149320
color	0.095160	-0.291439	0.020506	1.000000	-0.025646	-0.047316	-0.026457	-0.172520	-0.270281	-0.263579	-0.268226
clarity	0.208538	-0.352833	0.189171	-0.025646	1.000000	-0.067355	-0.160328	-0.146791	-0.371996	-0.358417	-0.366946
depth	-0.034896	0.028234	-0.218037	-0.047316	-0.067355	1.000000	-0.295798	-0.010630	-0.025289	-0.029340	0.094927
table	-0.100764	0.181602	-0.433387	-0.026457	-0.160328	-0.295798	1.000000	0.127118	0.195333	0.183750	0.150915
price	-0.306875	0.921591	-0.053487	-0.172520	-0.146791	-0.010630	0.127118	1.000000	0.884433	0.865419	0.861249
x	-0.405405	0.975093	-0.125564	-0.270281	-0.371996	-0.025289	0.195333	0.884433	1.000000	0.974701	0.970771
y	-0.395809	0.951721	-0.121461	-0.263579	-0.358417	-0.029340	0.183750	0.865419	0.974701	1.000000	0.952005
z	-0.399185	0.953387	-0.149320	-0.268226	-0.366946	0.094927	0.150915	0.861249	0.970771	0.952005	1.000000

In [27]:
```
#plotting the coorealtion heatmap
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True,cmap="coolwarm")
plt.title("Coorealtion Heatmap")
plt.show()
```



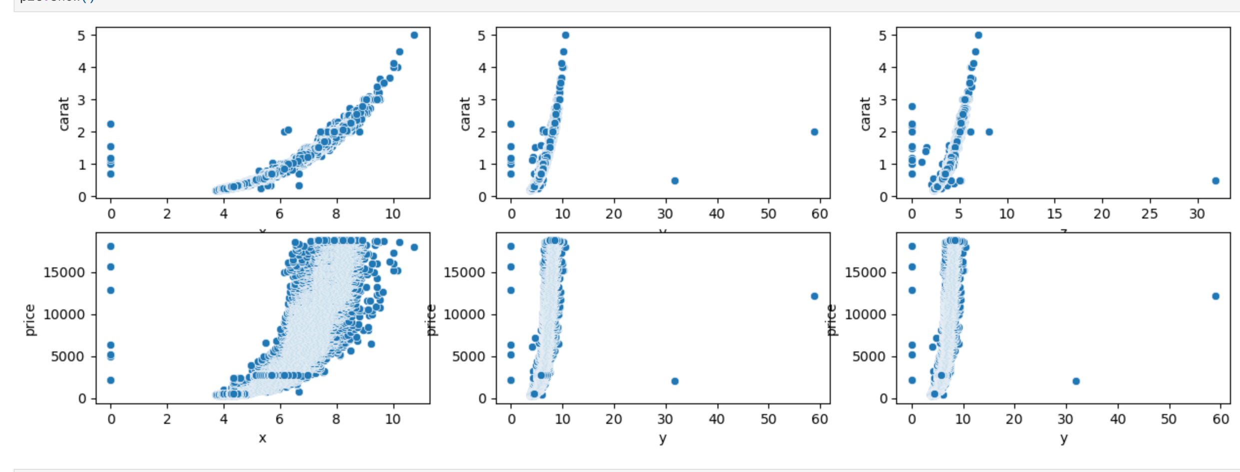
Plotting the relationship between price and carat

In [29]:
```
sns.lineplot(x="carat",y="price",data=df)
plt.title("Carat Vs Price")
plt.show()
```



From the lineplot it is quite to clear the price of diamonds is increasing with the increase of the carat of the diamonds.

In [31]:
```
fig,ax = plt.subplots(2,3,figsize=(15,5))
sns.scatterplot(x="x",y="carat",data=df,ax=ax[0,0])
sns.scatterplot(x="y",y="carat",data=df,ax=ax[0,1])
sns.scatterplot(x="z",y="carat",data=df,ax=ax[0,2])
sns.scatterplot(x="x",y="price",data=df,ax=ax[1,0])
sns.scatterplot(x="y",y="price",data=df,ax=ax[1,1])
sns.scatterplot(x="y",y="price",data=df,ax=ax[1,2])
plt.show()
```



In [ ]:
```
#Majority of the diamonds have x values between 4 and 8 ,y values between 4 and 10,and z values between 2 and 6,Diamonds with other Diamensions are very rare.
```

Train And testing

In [32]:
```
from sklearn.model_selection import train_test_split
x_test,x_train,y_test,y_train = train_test_split(df.drop('price',axis=1),df['price'],test_size=0.2,random_state=42)
```

Model Building

Decision Tree Regressor

In [33]:
```
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor()
dt
```

Out[33]:
```
DecisionTreeRegressor()
```

In [34]:
```
#training the model
dt.fit(x_train,y_train)
#train accuracy
dt.score(x_train,y_train)
```

Out[34]:
```
1.0
```

In [35]:
```
#predicting the test set
dt_pred = dt.predict(x_test)
```

Random Forest Regressor

In [36]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf
```

Out[36]:
```
RandomForestRegressor()
```

In [37]:
```
#training the model
rf.fit(x_train,y_train)
#train accuracy
rf.score(x_train,y_train)
```

Out[37]:
```
0.9998750994313683
```

In [38]:
```
#predicting the test set
rf_pred = rf.predict(x_test)
```

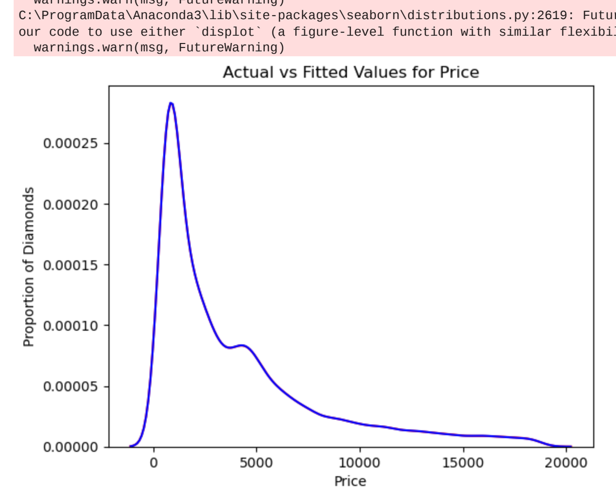
Model Evalution

In [39]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

Decision Tree Regressor

In [40]:
```
#distribution plot for actual and predicted values
ax = sns.distplot(y_test,hist=False,color='r',label='Actual Value')
sns.distplot(dt_pred,hist=False,color='b',label='Fitted Values',ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of Diamonds')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
```



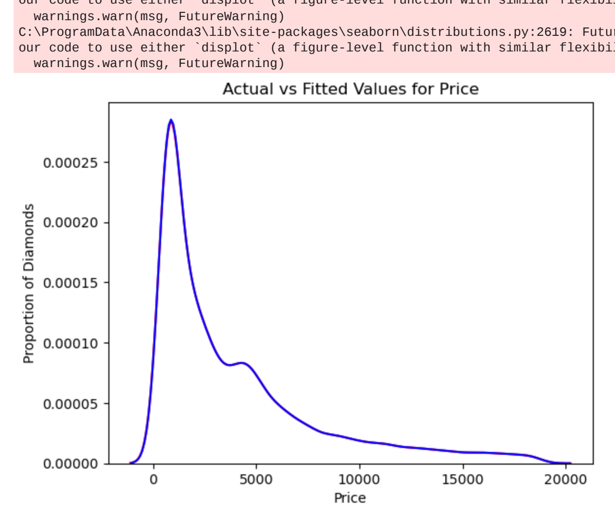
In [41]:
```
print('Decision Tree Regressor RMSE:',np.sqrt(mean_squared_error(y_test,dt_pred)))
print('Decision Tree Regressor Accuracy:',dt.score(x_test,y_test))
print('Decision Tree Regressor MAE:',mean_absolute_error(y_test,dt_pred))
```

```
Decision Tree Regressor RMSE: 144.69448278839047
Decision Tree Regressor Accuracy: 0.9986905599531942
Decision Tree Regressor MAE: 10.1044630856931
```

Random Forest Regressor

In [42]:
```
#distribution plot for actual and predicted values
ax = sns.distplot(y_test,hist=False,color='r',label='Actual Value')
sns.distplot(rf_pred,hist=False,color='b',label='Fitted Values',ax=ax)
plt.title('Actual vs Fitted Values for Price')
plt.xlabel('Price')
plt.ylabel('Proportion of Diamonds')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. 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).
  warnings.warn(msg, FutureWarning)
```



In [43]:
```
print('Random Forest Regressor RMSE:',np.sqrt(mean_squared_error(y_test,rf_pred)))
print('Random Forest Regressor Accuracy:',rf.score(x_test,y_test))
print('Random Forest Regressor MAE:',mean_absolute_error(y_test,rf_pred))
```

```
Random Forest Regressor RMSE: 84.844681294585
Random Forest Regressor Accuracy: 0.9995497736378737
Random Forest Regressor MAE: 8.241404968253235
```

Conclusion

Both the models have almost same accuracy. However, the Random Forest Regressor model is slightly better than the Decision Tree Regressor model.

There is something interesting about the data. The price of the diamonds with J color and I1 clarity is higher than the price of the diamonds with D color and IF clarity which couldn't be explained by the models. This could be because of the other factors that affect the price of the diamond.

In [ ]: