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, you can help diamond buyers and sellers make more informed decisions and gain insights into the factors that drive diamond prices.

1.Importing LIbraries :

- **Pandas:** Pandas is a powerful and widely used library for data manipulation and analysis. It provides data structures like DataFrames and Series, which allow you to store and manipulate tabular data. Pandas offers a wide range of functions for data cleaning, filtering, aggregation, merging, and more. It also supports reading and writing data from various file formats such as CSV, Excel, SQL databases, and more.
- **NumPy:** NumPy (Numerical Python) is a fundamental library for scientific computing in Python. It provides efficient data structures like arrays and matrices and a vast collection of mathematical functions. NumPy enables you to perform various numerical operations on large datasets, such as element-wise calculations, linear algebra, Fourier transforms, and random number generation. It also integrates well with other libraries for data analysis and machine learning.
- **Matplotlib:** Matplotlib is a popular plotting library that enables you to create a wide range of static, animated, and interactive visualizations. It provides a MATLAB-like interface and supports various types of plots, including line plots, scatter plots, bar plots, histograms, and more. Matplotlib gives you extensive control over plot customization, including labels, colors, legends, and annotations, allowing you to effectively communicate insights from your data
- **Seaborn:** Seaborn is a Python library for creating attractive and insightful statistical visualizations, particularly suited for categorical data, with built-in themes, improved aesthetics, and seamless integration with Pandas DataFrames.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

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

2.Importing Dataset :

The dataset meticulously presents attributes that intricately detail the characteristics of individual diamonds. Each diamond is defined by attributes like carat weight, influencing its mass, and cut quality, impacting visual brilliance. Color grade reveals hue, clarity detects imperfections, and depth provides proportion insight. Table width influences aesthetics. Price quantifies value, while dimensions ('x', 'y', 'z') offer precise measurements. These attributes collectively enable a predictive model for accurate diamond price forecasts, unveiling valuation insights.

```
dataframe = pd.read_csv('/content/diamonds.csv')
```

3.Exploratory Data Analysis :

Exploratory Data Analysis (EDA) in the "Predicting Diamond Price" project involves comprehensive attribute exploration, starting with statistics for central tendencies and distributions. Visualizations include histograms, box plots, and density plots for numeric attributes, while bar charts depict categorical attribute prevalence. Correlation analysis unveils numeric attribute connections via heatmaps, and scatter plots uncover interactions with the target "price." Grouping, aggregation, and box plots illustrate categorical attribute influences. EDA also covers dimensionality reduction like PCA, outlier handling, missing value checks, and feature transformations. Insights from EDA inform preprocessing, guiding model construction for accurate diamond price prediction.

```
dataframe.head()
```

	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

```
dataframe.tail()
```

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

columns

- Index counter
- **carat** - Carat weight of the diamond
- **cut** - Describe cut quality of the diamond. Quality in increasing order Fair, Good, Very Good, Premium, Ideal
- **color** -Color of the diamond, with D being the best and J the worst
- **clarity** -How obvious inclusions are within the diamond:(in order from best to worst, FL = flawless, I3= level 3 inclusions) FL,IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2, I3
- **depth** -depth % :The height of a diamond, measured from the culet to the table, divided by its average girdle diameter
- **table** - table%: The width of the diamond's table expressed as a percentage of its average diameter
- **price** -the price of the diamond
- **x** - length mm
- **y** - width mm

```
dataframe.shape
```

```
(53940, 11)
```

```
dataframe = dataframe.drop('Unnamed: 0', axis = 1)
```

```
dataframe.head()
```

	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

```
dataframe.shape
```

```
(53940, 10)
```

```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object 
 2   color    53940 non-null  object 
 3   clarity  53940 non-null  object 
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

```
dataframe.isna().sum()
```

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

```
dataframe.select_dtypes(include = 'object').head()
```

	cut	color	clarity
0	Ideal	E	SI2
1	Premium	E	SI1
2	Good	E	VS1
3	Premium	I	VS2
4	Good	J	SI2

```
print("__value_couts for CUT__",'\n'*2,dataframe['cut'].value_counts(),'\n', "___value_couts for COLOUR___ ",'\n'*2, dataframe['color'].value_
```

```
value_couts for CUT

 Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64 value_couts for COLOUR

 G    11292
E     9797
F     9542
H     8304
D     6775
I     5422
J     2808
Name: color, dtype: int64 value_couts for CLARITY

 SI1     13065
VS2     12258
SI2      9194
VS1      8171
VVS2     5066
VVS1     3655
IF       1790
I1        741
Name: clarity, dtype: int64
```

3.1.Data Visualization :

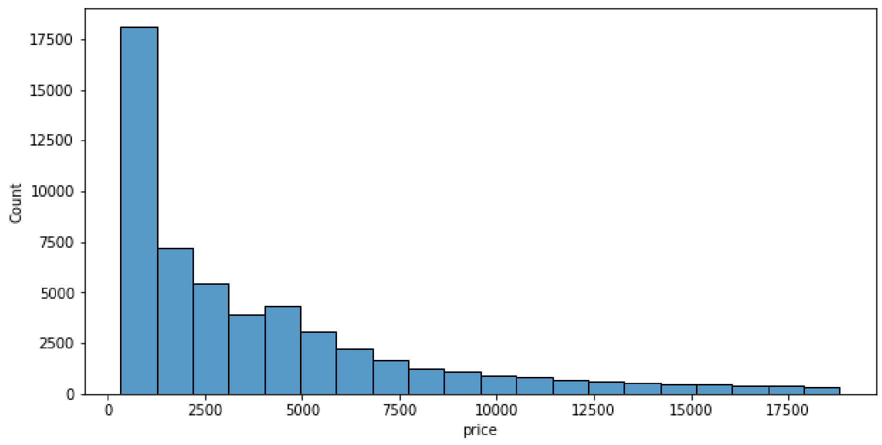
Data visualization is pivotal in the "Predicting Diamond Price" project's EDA, portraying attributes' distribution through histograms and density plots. Box plots reveal central tendencies and outliers, while bar charts depict categorical attributes like cut, color, and clarity. Correlation heatmaps identify strong relationships, while scatter plots unveil interactions with "price." Box plots by category enable price comparison, 3D scatter plots elucidate dimension relations, and violin plots capture price distributions. Interactive options offer deeper insights. These visualizations guide preprocessing and model construction for precise price predictions.

3.1.1 Single Visualization :

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.

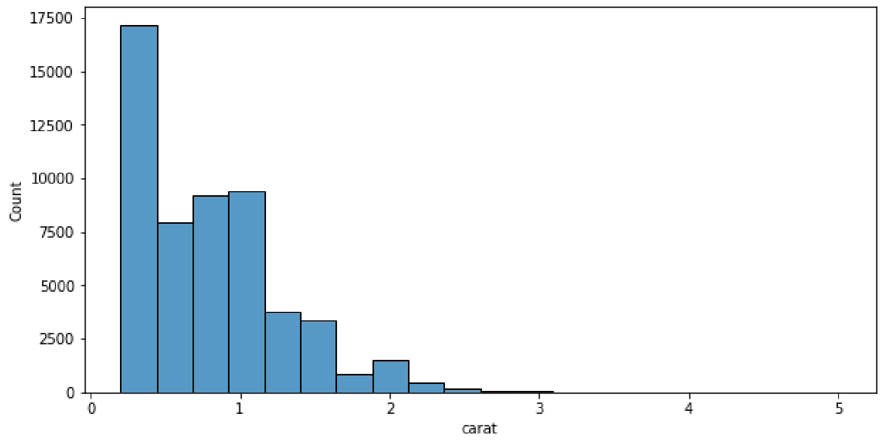
```
plt.figure(figsize = (10, 5))
sns.histplot(dataframe['price'], bins = 20)
```

```
<AxesSubplot:xlabel='price', ylabel='Count'>
```

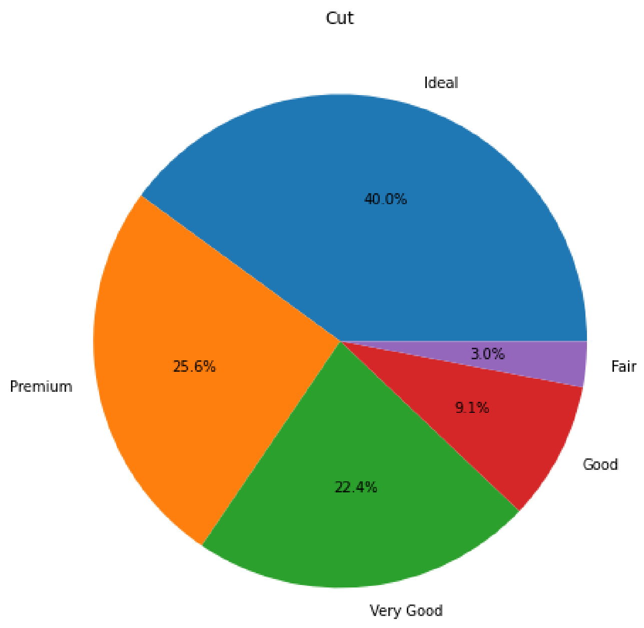


```
plt.figure(figsize = (10, 5))
sns.histplot(dataframe['carat'], bins=20)
```

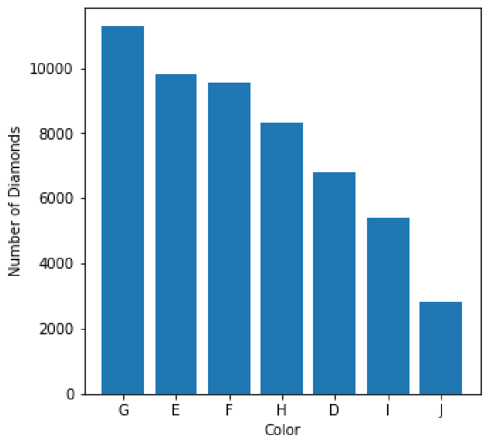
```
<AxesSubplot:xlabel='carat', ylabel='Count'>
```



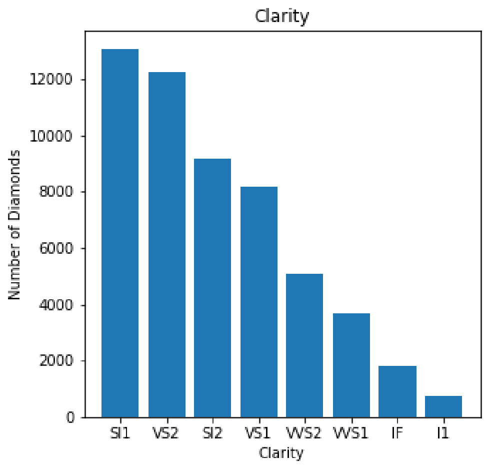
```
plt.figure(figsize=(8, 8))
plt.pie(dataframe['cut'].value_counts(),labels=['Ideal','Premium','Very Good','Good','Fair'],autopct='%1.1f%%')
plt.title('Cut')
plt.show()
```



```
plt.figure(figsize=(5,5))
plt.bar(dataframe['color'].value_counts().index, dataframe['color'].value_counts())
plt.ylabel("Number of Diamonds")
plt.xlabel("Color")
plt.show()
```

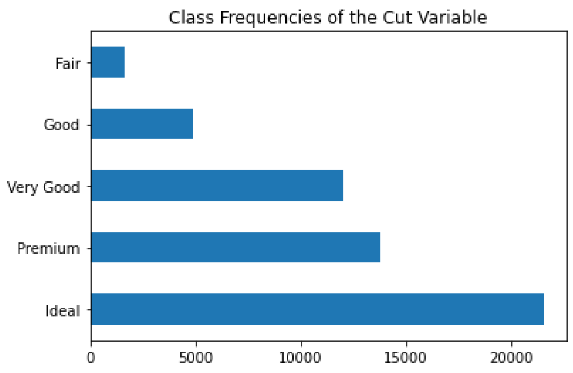


```
plt.figure(figsize=(5,5))
plt.bar(dataframe['clarity'].value_counts().index, dataframe['clarity'].value_counts())
plt.title('Clarity')
plt.ylabel("Number of Diamonds")
plt.xlabel("Clarity")
plt.show()
```

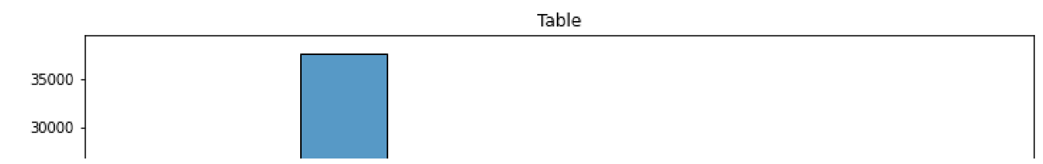


```
dataframe["cut"].value_counts().plot.barh().set_title("Class Frequencies of the Cut Variable")
```

```
Text(0.5, 1.0, 'Class Frequencies of the Cut Variable')
```



```
plt.figure(figsize = (12, 5))
sns.histplot(dataframe['table'], bins=10)
plt.title('Table')
plt.show()
```

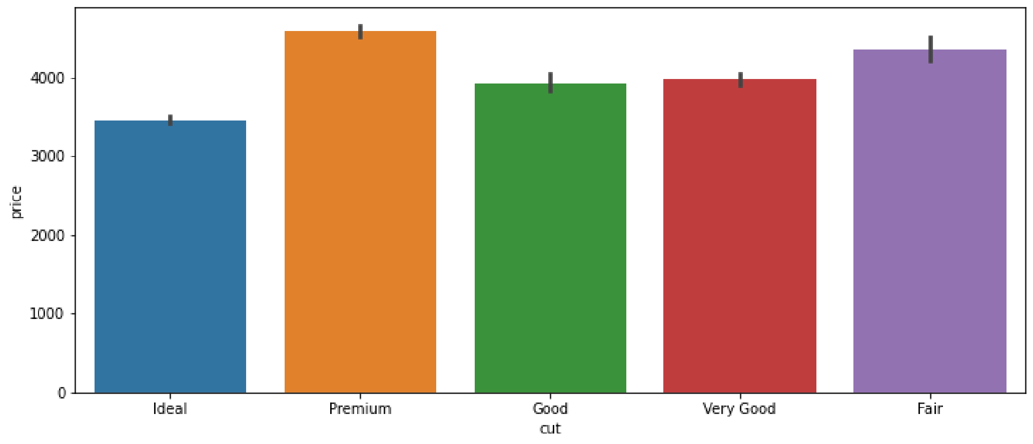


3.1.2 Grouped Visualization :

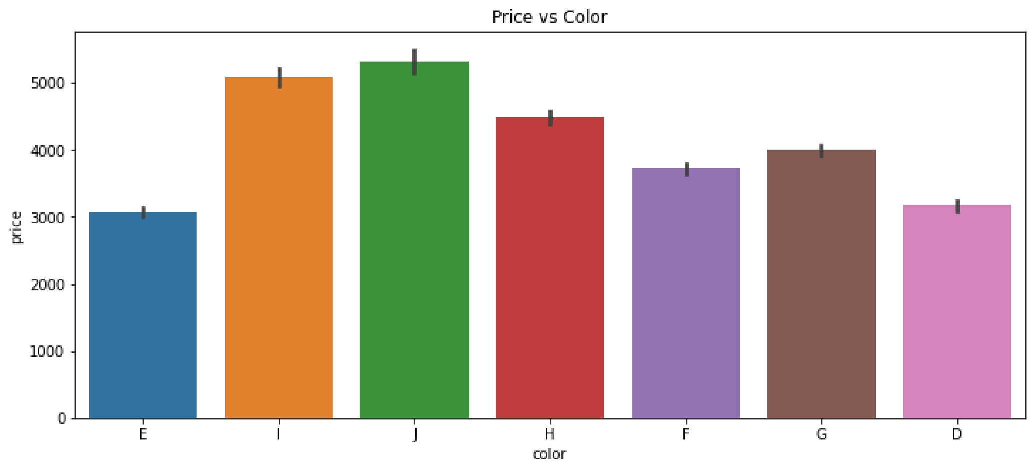
Grouped visualizations in the "Predicting Diamond Price" project's EDA unveil attribute interactions within categorical categories like "cut," "color," and "clarity." These visualizations, including box plots, bar charts, scatter plots with hues, and more, offer insights into how attributes influence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

```
plt.figure(figsize = (12, 5))
sns.barplot(x='cut',
            y='price',
            data = dataframe)
```

```
<AxesSubplot:xlabel='cut', ylabel='price'>
```

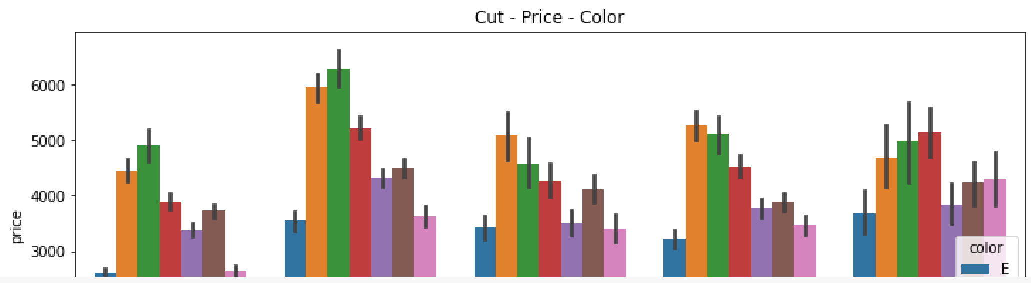


```
plt.figure(figsize = (12, 5))
sns.barplot(x='color',
            y='price',
            data = dataframe)
plt.title('Price vs Color')
plt.show()
```



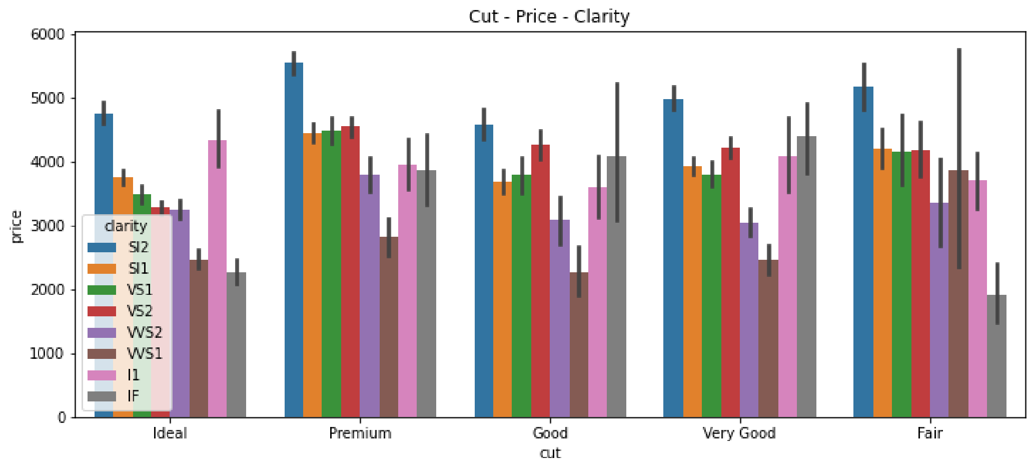
```
plt.figure(figsize = (12, 5))
sns.barplot(x="cut",
            y="price",
            hue="color",
            data=dataframe)
plt.title("Cut - Price - Color")
```

```
Text(0.5, 1.0, 'Cut - Price - Color')
```



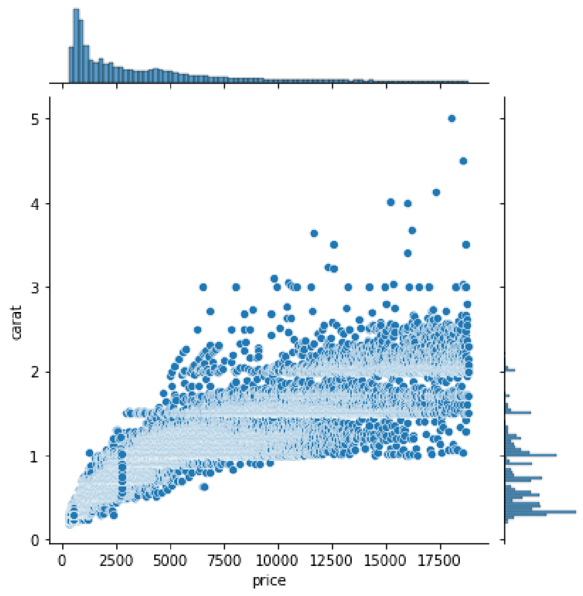
```
plt.figure(figsize = (12, 5))
sns.barplot(x="cut",
            y="price",
            hue="clarity",
            data = dataframe)
plt.title("Cut - Price - Clarity")
```

```
Text(0.5, 1.0, 'Cut - Price - Clarity')
```



```
sns.jointplot(x = "price",
              y = dataframe["carat"],
              data = dataframe)
```

```
<seaborn.axisgrid.JointGrid at 0x1749a4fb4f0>
```



3.2.Data Preprocessing :

Data preprocessing is a fundamental step in the "Predicting Diamond Price" project, involving measures like handling missing values, managing outliers, feature engineering, encoding categorical variables, and scaling features. This process enhances dataset quality, ensures compatibility with machine learning algorithms, and forms the basis for accurate predictive modeling.

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

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

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

```
dataframe.head()
```

	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

3.2.1 Statical Info :

Statistical information is pivotal in the "Predicting Diamond Price" project's EDA. Measures like mean, median, and standard deviation offer central tendencies and spread insights. Percentiles, skewness, and kurtosis illuminate data distribution and shape.Statistical insights guide subsequent analysis, visualization, and preprocessing steps.

```
dataframe.describe()
```

	carat	cut	color	clarity	depth	table	price	x	y	
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.00
mean	0.797940	3.904097	4.405803	4.051020	61.749405	57.457184	3932.799722	5.731157	5.734526	3.53
std	0.474011	1.116600	1.701105	1.647136	1.432621	2.234491	3989.439738	1.121761	1.142135	0.70
min	0.200000	1.000000	1.000000	1.000000	43.000000	43.000000	326.000000	0.000000	0.000000	0.00
25%	0.400000	3.000000	3.000000	3.000000	61.000000	56.000000	950.000000	4.710000	4.720000	2.91
50%	0.700000	4.000000	4.000000	4.000000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.53
75%	1.040000	5.000000	6.000000	5.000000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.04
max	5.010000	5.000000	7.000000	8.000000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.80

3.2.2 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.

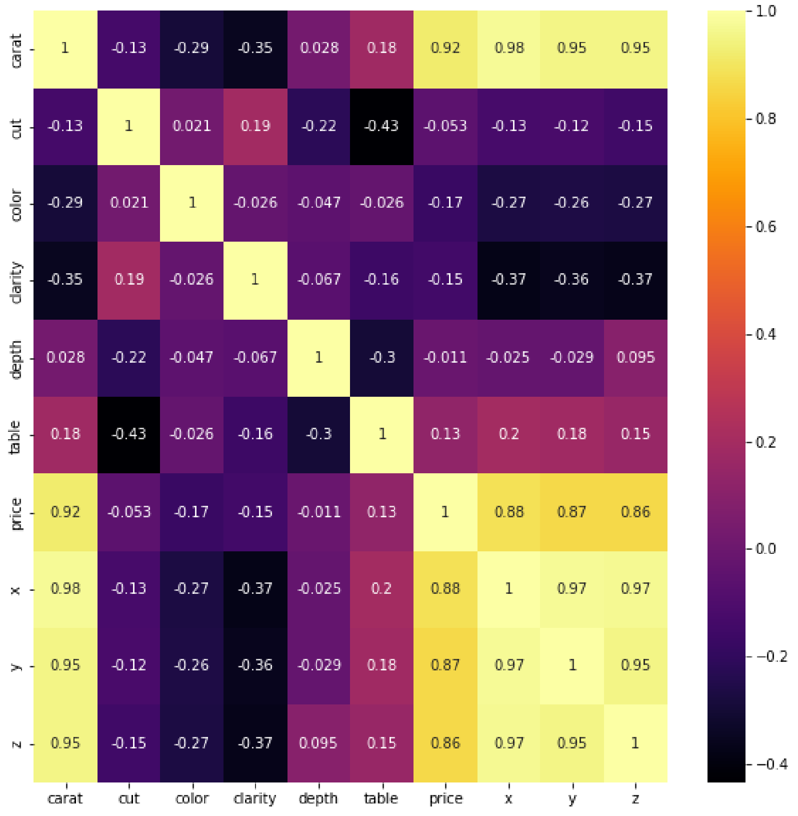
```
corr_dataframe = dataframe.corr()
```

```
corr_dataframe
```

	carat	cut	color	clarity	depth	table	price	x	y	z
carat	1.000000	-0.134967	-0.291437	-0.352841	0.028224	0.181618	0.921591	0.975094	0.951722	0.953387
cut	-0.134967	1.000000	0.020519	0.189175	-0.218055	-0.433405	-0.053491	-0.125565	-0.121462	-0.149323
color	-0.291437	0.020519	1.000000	-0.025631	-0.047279	-0.026465	-0.172511	-0.270287	-0.263584	-0.268227
clarity	-0.352841	0.189175	-0.025631	1.000000	-0.067384	-0.160327	-0.146800	-0.371999	-0.358420	-0.366952
depth	0.028224	-0.218055	-0.047279	-0.067384	1.000000	-0.295779	-0.010647	-0.025289	-0.029341	0.094924
table	0.181618	-0.433405	-0.026465	-0.160327	-0.295779	1.000000	0.127134	0.195344	0.183760	0.150929
price	0.921591	-0.053491	-0.172511	-0.146800	-0.010647	0.127134	1.000000	0.884435	0.865421	0.861249
x	0.975094	-0.125565	-0.270287	-0.371999	-0.025289	0.195344	0.884435	1.000000	0.974701	0.970772
y	0.951722	-0.121462	-0.263584	-0.358420	-0.029341	0.183760	0.865421	0.974701	1.000000	0.952006
z	0.953387	-0.149323	-0.268227	-0.366952	0.094924	0.150929	0.861249	0.970772	0.952006	1.000000

```
plt.figure(figsize = (10, 10))
sns.heatmap(corr_dataframe,
            annot = True,
            cmap = 'inferno')
```

```
<AxesSubplot:>
```



	carat	cut	color	clarity	depth	table	price	x	y	z
carat	1	-0.13	-0.29	-0.35	0.028	0.18	0.92	0.98	0.95	0.95
cut	-0.13	1	0.021	0.19	-0.22	-0.43	-0.053	-0.13	-0.12	-0.15
color	-0.29	0.021	1	-0.026	-0.047	-0.026	-0.17	-0.27	-0.26	-0.27
clarity	-0.35	0.19	-0.026	1	-0.067	-0.16	-0.15	-0.37	-0.36	-0.37
depth	0.028	-0.22	-0.047	-0.067	1	-0.3	-0.011	-0.025	-0.029	0.095
table	0.18	-0.43	-0.026	-0.16	-0.3	1	0.13	0.2	0.18	0.15
price	0.92	-0.053	-0.17	-0.15	-0.011	0.13	1	0.88	0.87	0.86
x	0.98	-0.13	-0.27	-0.37	-0.025	0.2	0.88	1	0.97	0.97
y	0.95	-0.12	-0.26	-0.36	-0.029	0.18	0.87	0.97	1	0.95
z	0.95	-0.15	-0.27	-0.37	0.095	0.15	0.86	0.97	0.95	1

```
dataframe.columns
```

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

```
dataset = dataframe.drop('price', axis = 1)
```

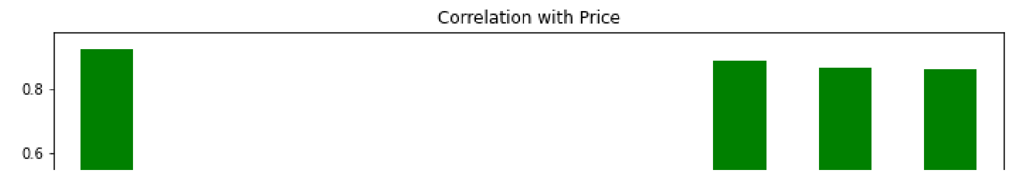
```
dataset.head()
```

	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

3.2.2.1 Correlation Of Diamond Price with Various Attributes :

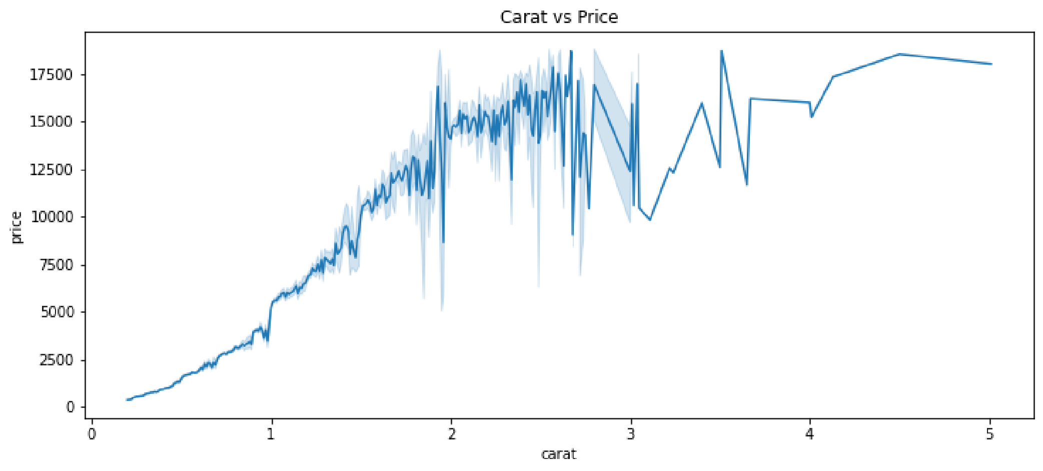
```
dataset.corrwith(dataframe['price']).plot.bar(
    figsize = (12, 5),
    title = 'Correlation with Price',
    cmap = 'ocean'
)
```

```
<AxesSubplot:title={'center':'Correlation with Price'}>
```



3.2.2.2 Relation Between Price And Caret:

```
plt.figure(figsize = (12, 5))
sns.lineplot(x='carat',
             y='price',
             data = dataframe)
plt.title('Carat vs Price')
plt.show()
```

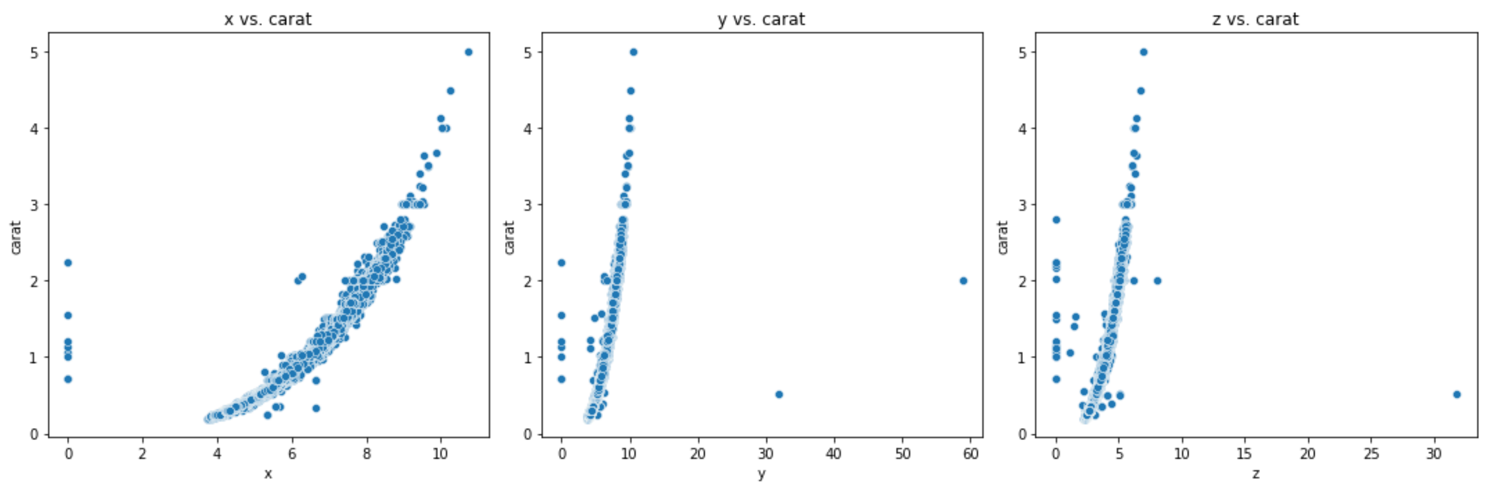


```
columns_to_plot = ['x', 'y', 'z']
titles = ['x vs. carat', 'y vs. carat', 'z vs. carat']

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

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

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

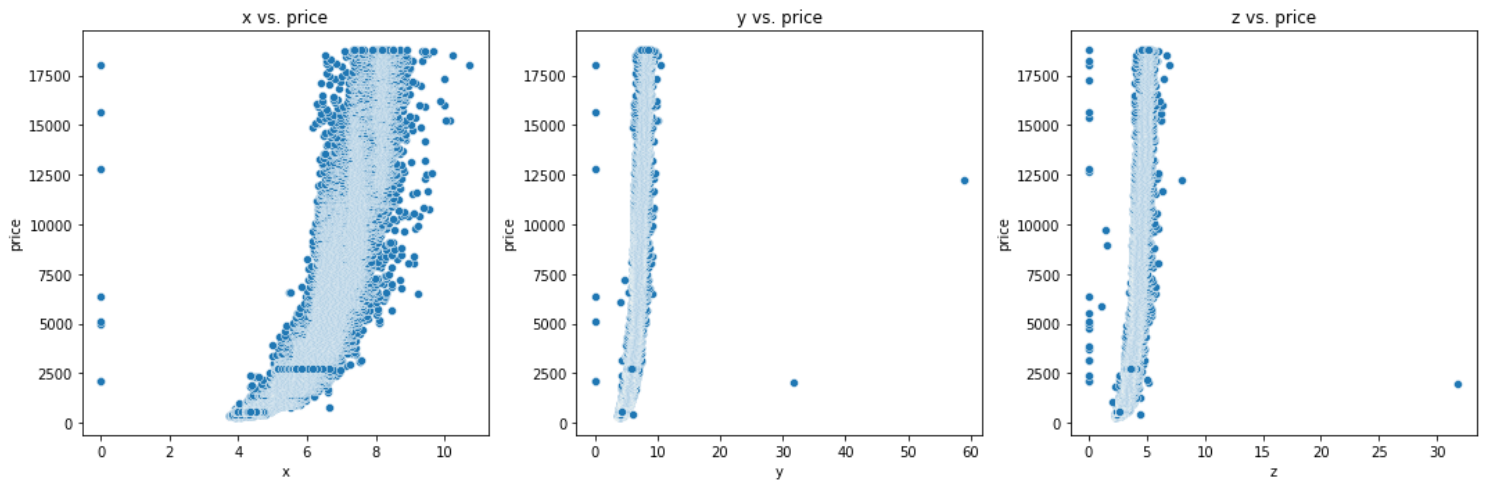


```
columns_to_plot = ['x', 'y', 'z']
titles = ['x vs. price', 'y vs. price', 'z vs. price']

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

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

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



```
dataframe.head()
```

	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

3.3.Splitting Dataset :

Splitting a dataset refers to the process of dividing a given dataset into two or more subsets for training and evaluation purposes. The most common type of split is between the training set and the testing (or validation) set. This division allows us to assess the performance of a machine learning model on unseen data and evaluate its generalization capabilities.

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 80% for training and 20% for testing.

```
x = dataframe.drop('price', axis = 1)
y = dataframe['price']
```

```
x.shape, y.shape
```

```
((53940, 9), (53940,))
```

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.2,
                                                    random_state = 42)
```

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((43152, 9), (10788, 9), (43152,), (10788,))
```

4. Model Selection And Training :

Model Selection:

Model selection involves choosing the best algorithm or model architecture for the given problem and dataset. This step requires careful consideration of various factors, such as the nature of the data (e.g., numerical or categorical), the problem type (e.g., regression, classification, clustering), the amount of available data, and the desired model performance. It is essential to select a model that can effectively capture the underlying patterns in the data and make accurate predictions.

Model Training:

Once the appropriate model has been selected, the next step is to train it on the dataset. Model training involves adjusting the model's parameters using the training data to make accurate predictions on unseen data. The goal is to minimize the difference between the model's predictions and the actual target values during training.

```
from sklearn.tree import DecisionTreeRegressor
```

```
DTR = DecisionTreeRegressor()
```

```
DTR.fit(x_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
DTR.score(x_test, y_test)
```

```
0.9666081086538498
```

```
DTR.score(x_train, y_train)
```

```
0.9999948355708338
```

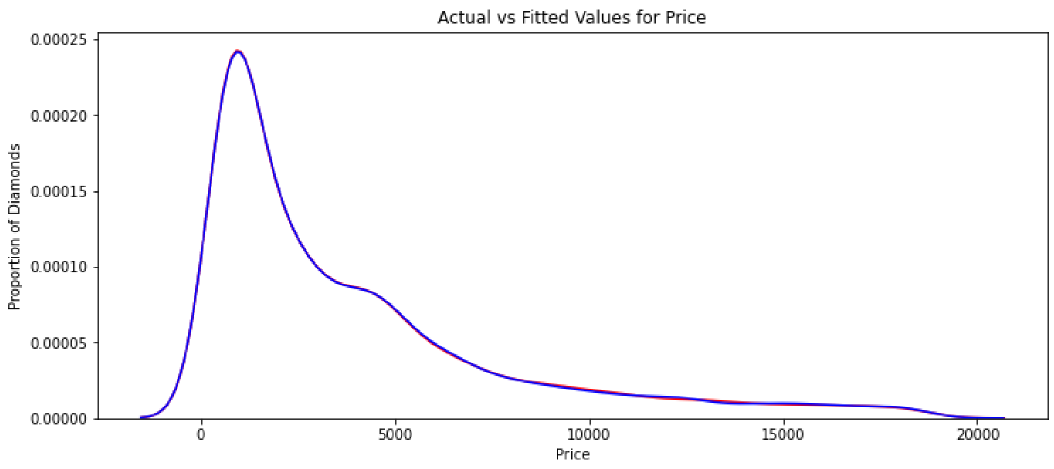
```
y_pred_DTR = DTR.predict(x_test)
```

```
y_pred_DTR
```

```
array([ 559., 2351., 1238., ...,  642., 9660., 4130.])
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

```
plt.figure(figsize = (12, 5))
ax = sns.distplot(y_test, hist = False, color = 'r', label = 'Actual Value')
sns.distplot(y_pred_DTR, 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()
```



```
from sklearn.ensemble import RandomForestRegressor
```

```
reg = RandomForestRegressor()
```

```
reg.fit(x_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
reg.score(x_test, y_test)
```

```
0.9817210885094707
```

```
reg.score(x_train, y_train)
```

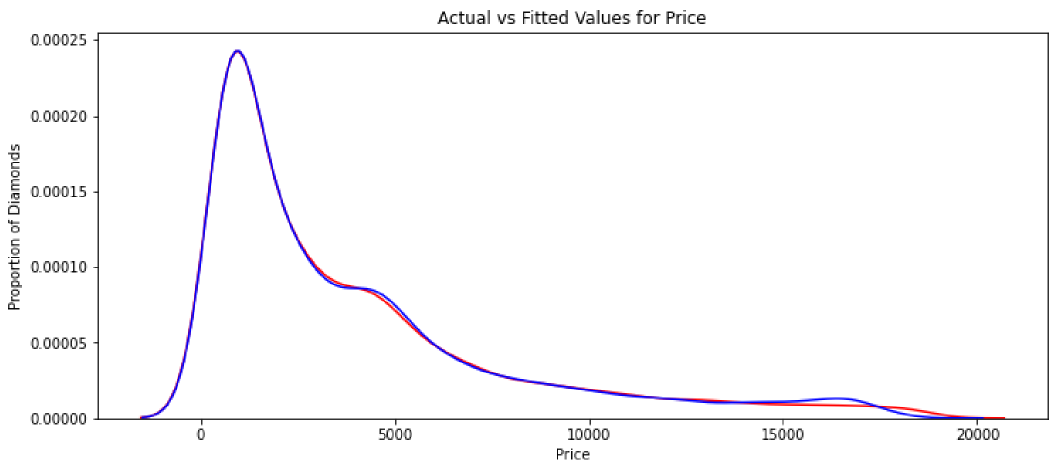
```
0.9974267892292097
```

```
y_pred_reg = reg.predict(x_test)
```

```
y_pred_reg
```

```
    array([ 557.3 , 2328.71, 1210.74, ...,  775.96, 9581.67, 3892.95])
```

```
plt.figure(figsize = (12, 5))
ax = sns.distplot(y_test, hist = False, color = 'r',label = 'Actual Value')
sns.distplot(y_pred_reg , 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()
```



Conclusion :

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

✓ 0s completed at 2:41 PM