```
#Importing important libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.ensemble import RandomForestRegressor
import warnings
warnings.filterwarnings("ignore")


#Impoting Data from .csv and saving in variable TD
TD = pd.read_csv('/content/diamonds.csv')
TD.head()# to show 1st 5 rows of data
```

	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

Next steps: Generate code with TD View recommended plots

```
TD.shape # to show the rows and colmns of the data
```

```
(53940, 11)
```

```
TD.isnull().sum() # to show the sum of all null fields
```

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

```
TD = TD.dropna() # this removes all data whcih contains null values, since there are no null values the data remains the same
TD.drop(TD.columns[0], axis=1, inplace=True) # removing the 1st columnn since the its just numbering


TD.isnull().sum() # repeating previous step to recheck
```

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


TD.dtypes # to show the datatypes of the data
```

```
carat      float64
cut         object
color       object
clarity     object
depth      float64
table      float64
price        int64
x          float64
y          float64
z          float64
dtype: object
```

```
# EDA
TD.describe() # to show summary statistics of the data
```

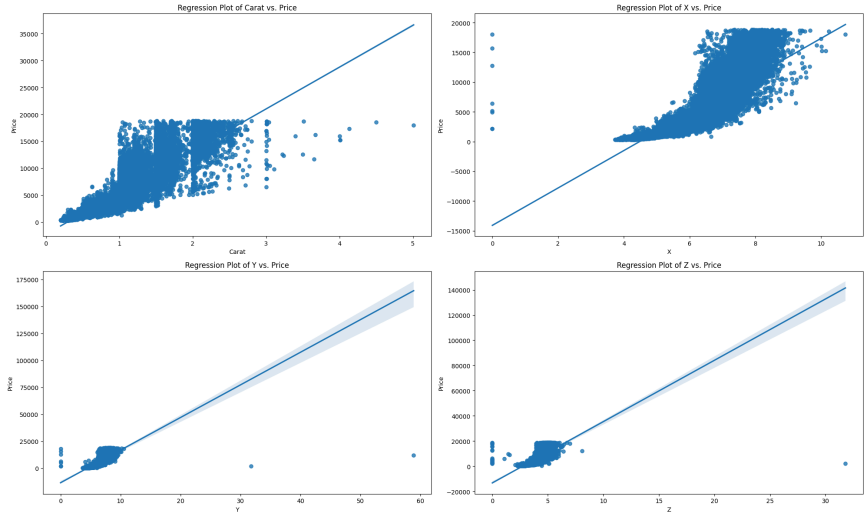
	carat	depth	table	price	x	
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000(
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734(
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142(
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000(
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720(
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710(
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540(
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900(

```
TD.describe(include='object') # to show summary of objects/categories which without specifying gives summary without objects
```

	cut	color	clarity
count	53940	53940	53940
unique	5	7	8
top	Ideal	G	SI1
freq	21551	11292	13065

```
#Regression plot against all the categories
features = ['carat', 'x', 'y', 'z']
plt.figure(figsize=(20, 12))
for i, feature in enumerate(features):
    plt.subplot(2, 2, i + 1)
    sns.regplot(x=feature, y='price', data=TD)
    plt.title(f'Regression Plot of {feature.capitalize()} vs. Price')
    plt.xlabel(feature.capitalize())
    plt.ylabel('Price')

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

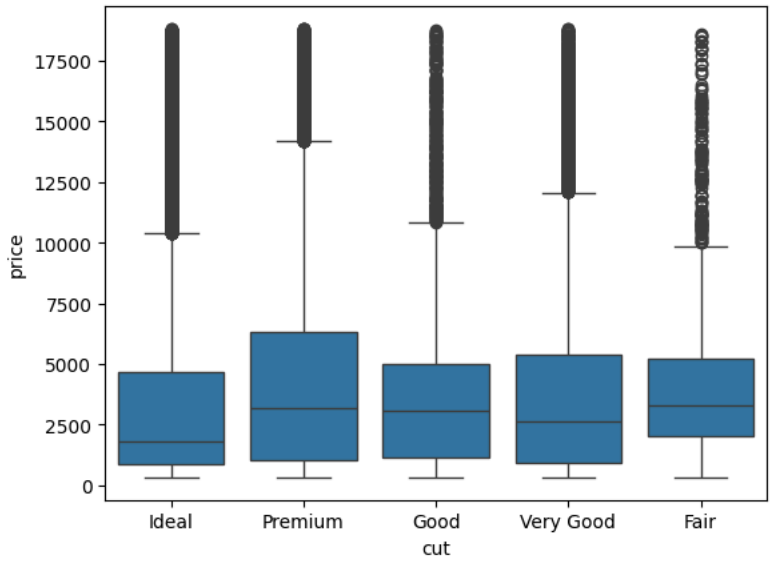



```
# Pearson Correlation Coefficient for all the categories
from scipy import stats
features = ['carat', 'x', 'y', 'z']
for feature in features:
    pc, pv = stats.pearsonr(TD[feature], TD['price'])
    print(f"The Pearson Correlation Coefficient for {feature} is {pc:.4f} with a P-value of P = {pv:.4g}")
```

```
The Pearson Correlation Coefficient for carat is 0.9216 with a P-value of P = 0
The Pearson Correlation Coefficient for x is 0.8844 with a P-value of P = 0
The Pearson Correlation Coefficient for y is 0.8654 with a P-value of P = 0
The Pearson Correlation Coefficient for z is 0.8612 with a P-value of P = 0
```

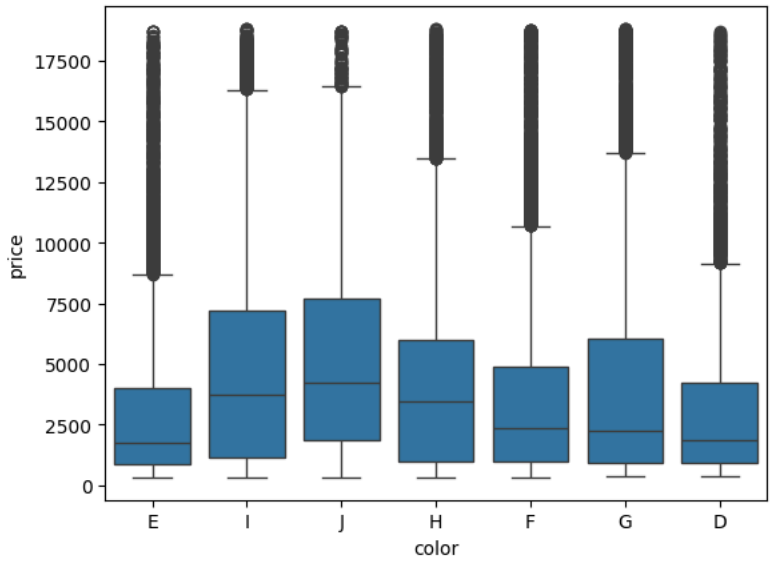
```
# box plot for the quality of cut
sns.boxplot(x='cut',y='price',data = TD)
```

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



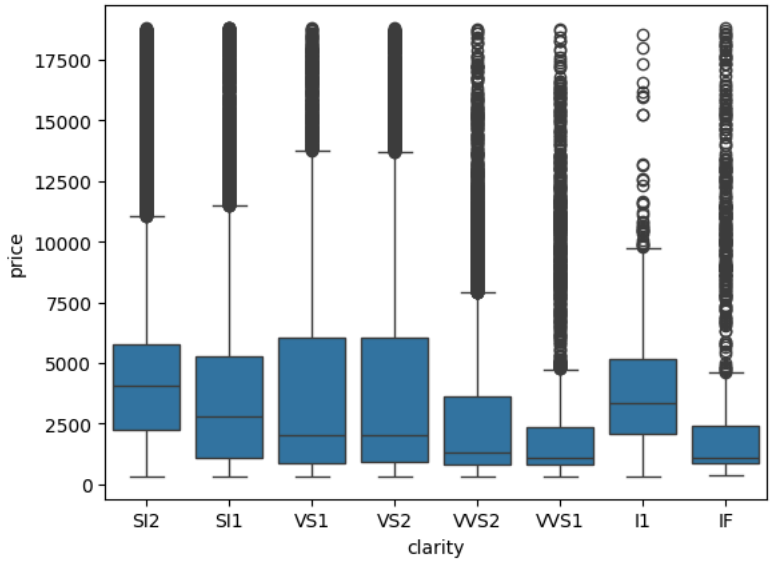

```
# box plot for color of the diamond
sns.boxplot(x='color',y='price',data = TD)
```

```
<Axes: xlabel='color', ylabel='price'>
```




```
# box plot for the clarity of the diamond
sns.boxplot(x='clarity',y='price',data = TD)
```

```
<Axes: xlabel='clarity', ylabel='price'>
```



```
#nothing to remove since all data is required and affects the price of the diamond
TD.shape
```

```
(53940, 10)
```

```
# Data Transformation
labelencoder = LabelEncoder()
TD.cut = labelencoder.fit_transform(TD.cut)
TD.color = labelencoder.fit_transform(TD.color)
TD.clarity = labelencoder.fit_transform(TD.clarity)
```

```
TD.head(10) # to check if label encoding is applied
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	2.31
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75
5	0.24	4	6	7	62.8	57.0	336	3.94	3.96	2.48
6	0.24	4	5	6	62.3	57.0	336	3.95	3.98	2.47
7	0.26	4	4	2	61.9	55.0	337	4.07	4.11	2.53
8	0.22	0	1	5	65.1	61.0	337	3.87	3.78	2.49
9	0.23	4	4	4	59.4	61.0	338	4.00	4.05	2.39

Next steps: Generate code with TD View recommended plots

```
# normalization
traindata = stats.zscore(TD)
traindata
```

	carat	cut	color	clarity	depth	table	price	
0	-1.198168	-0.538099	-0.937163	-0.484264	-0.174092	-1.099672	-0.904095	-1.5878
1	-1.240361	0.434949	-0.937163	-1.064117	-1.360738	1.585529	-0.904095	-1.6413
2	-1.198168	-1.511147	-0.937163	0.095589	-3.385019	3.375663	-0.903844	-1.4986
3	-1.071587	0.434949	1.414272	0.675442	0.454133	0.242928	-0.902090	-1.3649
4	-1.029394	-1.511147	2.002131	-0.484264	1.082358	0.242928	-0.901839	-1.2401
...	...	...	...	...	...	...	...	
53935	-0.164427	-0.538099	-1.525021	-1.064117	-0.662711	-0.204605	-0.294731	0.0167
53936	-0.164427	-1.511147	-1.525021	-1.064117	0.942753	-1.099672	-0.294731	-0.0366
53937	-0.206621	1.407998	-1.525021	-1.064117	0.733344	1.137995	-0.294731	-0.0634
53938	0.130927	0.434949	0.826413	-0.484264	-0.523105	0.242928	-0.294731	0.3733
53939	-0.101137	-0.538099	-1.525021	-0.484264	0.314528	-1.099672	-0.294731	0.0881

53940 rows × 10 columns

Next steps: Generate code with traindata View recommended plots

```
# dividing the data wherein xt retains all the features while yt retains the target (price)
xt = traindata.drop('price',axis =1)
yt = traindata['price']
```

```
xt.head() # checking if data is split properly
```

	carat	cut	color	clarity	depth	table	x	y
0	-1.198168	-0.538099	-0.937163	-0.484264	-0.174092	-1.099672	-1.587837	-1.536196
1	-1.240361	0.434949	-0.937163	-1.064117	-1.360738	1.585529	-1.641325	-1.658774
2	-1.198168	-1.511147	-0.937163	0.095589	-3.385019	3.375663	-1.498691	-1.457395
3	-1.071587	0.434949	1.414272	0.675442	0.454133	0.242928	-1.364971	-1.317305
4	-1.029394	-1.511147	2.002131	-0.484264	1.082358	0.242928	-1.240167	-1.212238

Next steps: Generate code with xt View recommended plots

```
yt.head() # checking if data is split properly
```

```
0   -0.904095
1   -0.904095
2   -0.903844
3   -0.902090
4   -0.901839
Name: price, dtype: float64
```

```
from sklearn.model_selection import train_test_split #importing important library for training a model, to split the data properly acco
Xt,Xtt,Yt,Ytt = train_test_split(xt,yt,test_size=0.3,random_state=42)
```

```
# Multiple Linear Regression fitting training data into model
model = LinearRegression()
model_mlr = model.fit(Xt,Yt)
```

```
# making prediction after fitting data into mdoel
YpMLR = model_mlr.predict(Xtt)
```

```
# calculating the mean square error of the model with actual data
mse_MLR = mean_squared_error(Ytt,YpMLR)
print('The mean squared error for Multiple Linear Regression: ',mse_MLR)
```

```
The mean squared error for Multiple Linear Regression:  0.11135551915732923
```

```
# calculating the mean absolute error of the model with actual data
mae_MLR = mean_absolute_error(Ytt,YpMLR)
print('The mean absolute error for Multiple Linear Regression: ',mae_MLR)
```

```
The mean absolute error for Multiple Linear Regression:  0.2146708101077064
```

```
# fitting data into Random forest
rfModel = RandomForestRegressor()
model_rf = rfModel.fit(Xt,Yt)
```

```
# making prediction after fitting data
Yprf = model_rf.predict(Xtt)
```

```
# calculating the mean square error of the model with actual data
mse_RF = mean_squared_error(Ytt,Yprf)
print('The mean squared error for Random Forest: ',mse_RF)
```

```
The mean squared error for Random Forest:  0.01842728160133064
```

```
# calculating the mean absolute error of the model with actual data
mae_RF = mean_absolute_error(Ytt,Yprf)
print('The mean absolute error for Random Forest: ',mae_RF)
```

```
The mean absolute error for Random Forest:  0.06785410202870568
```

```
# fitting data into LASSO model
LassoModel = Lasso()
model_ls = LassoModel.fit(Xt,Yt)
```

```
# making prediction using testing data
YpLs = model_ls.predict(Xtt)
```

```
# calculating the mean square error of the model with actual data
mse_Ls = mean_squared_error(Ytt,YpLs)
print('The mean squared error for Lasso: ',mse_Ls)
```

```
The mean squared error for Lasso:  0.9801771432345613
```

```
# calculating the mean absolute error of the model with actual data
mae_Ls = mean_absolute_error(Ytt,YpLs)
print('The mean absolute error for Lasso: ',mae_Ls)
```

```
The mean absolute error for Lasso:  0.7535247714449218
```

```
# saving all the scores of mean squared error
scoresSE = [('MLR',mse_MLR),('Random Forest',mse_RF),('LASSO',mse_Ls)]
# saving all the scores of mean absolute error
scoresAE = [('MLR',mae_MLR),('Random Forest',mae_RF),('LASSO',mae_Ls)]
```

```
# Visualizing MSE and MAE in table format
mse = pd.DataFrame(data = scoresSE,columns=['Model','MSE Score'])
mae = pd.DataFrame(data = scoresAE,columns=['Model','MAE Score'])
cdf = pd.merge(mse,mae,on='Model')
cdf
```

	Model	MSE Score	MAE Score
0	MLR	0.111356	0.214671
1	Random Forest	0.018427	0.067854
2	LASSO	0.980177	0.753525

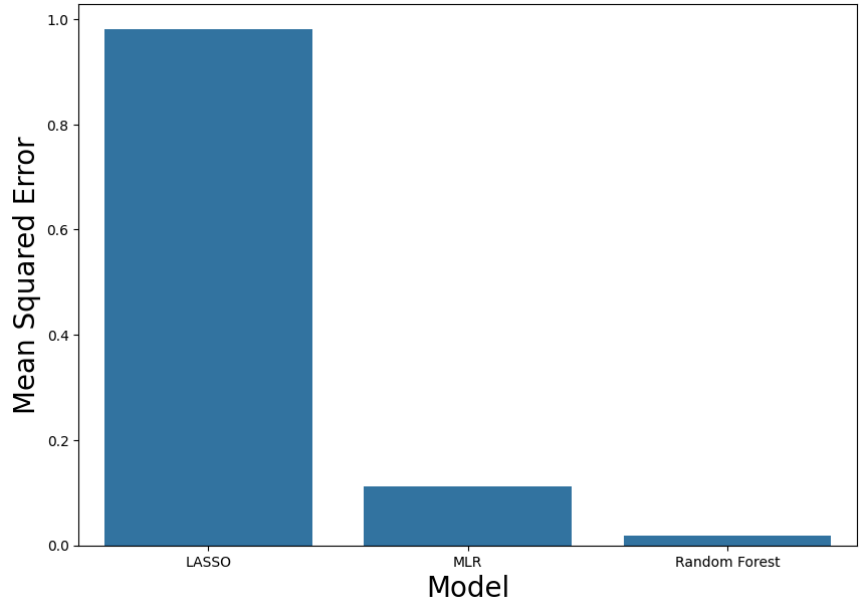
Next steps: Generate code with cdf | View recommended plots

```
# Visualizing findings in diagram
mse.sort_values(by=['MSE Score'], ascending=False, inplace=True)
f,axe = plt.subplots(1,1,figsize=(10,7))
sns.barplot(x=mse['Model'],y=mse['MSE Score'],ax = axe)
axe.set_xlabel('Model',size=20)
axe.set_ylabel('Mean Squared Error',size=20)
plt.show
```

```
matplotlib.pyplot.show
def show(*args, **kwargs)
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py
Display all open figures.

Parameters
----------
block : bool, optional
```



```
# Visualizing findings in diagram
mae.sort_values(by=(['MAE Score']), ascending=False, inplace=True)
f,axe = plt.subplots(1,1,figsize=(10,7))
```