Diamond_Price_Prediction

July 8, 2023

1 Problem Statements: Diamond Price Prediction

1.1 Description:

- **Diamonds are the hardest naturally occurring substance known to man. They are made of pure carbon, arranged in a crystal structure called diamond cubic.**
- **Diamonds are found in volcanic rock, called kimberlite, which is formed when the Earth's mantle melts and rises to the surface.**
- **Diamonds can be colorless, or they can have a variety of colors, including yellow, brown, pink, blue, green, and red. The color of a diamond is determined by the impurities that are present in the carbon atoms.**
- **Diamonds are used in jewelry and in industrial applications. In jewelry, diamonds are prized for their beauty, durability, and rarity. In industry, diamonds are used for cutting, drilling, and polishing.**
- **The four Cs of diamonds are color, cut, clarity, and carat. These are the factors that determine the value of a diamond.**

2 1.0. Importing Libraries

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

from sklearn.impute import SimpleImputer ## HAndling Missing Values
from sklearn.preprocessing import StandardScaler # HAndling Feature Scaling
from sklearn.preprocessing import OrdinalEncoder # Ordinal Encoding
## pipelines
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

3 2.0. The Datasets

3.1 2.1. Reading Datasets

[ ]:
```
## Data Ingestions step
df=pd.read_csv('data/gemstone.csv')
df.head()
```

[ ]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

3.2 2.2. Datasets Infromation:

3.2.1 Introduction About the Data :

The dataset The goal is to predict `price` of given diamond (Regression Analysis).

There are 10 independent variables (including `id`):

- `id` : unique identifier of each diamond
- `carat` : Carat (ct.) refers to the unique unit of weight measurement used exclusively to weigh gemstones and diamonds.
- `cut` : Quality of Diamond Cut
- `color` : Color of Diamond
- `clarity` : Diamond clarity is a measure of the purity and rarity of the stone, graded by the visibility of these characteristics under 10-power magnification.
- `depth` : The depth of diamond is its height (in millimeters) measured from the culet (bottom tip) to the table (flat, top surface)
- `table` : A diamond's table is the facet which can be seen when the stone is viewed face up.
- `x` : Diamond X dimension
- `y` : Diamond Y dimension
- `x` : Diamond Z dimension

Target variable: * `price`: Price of the given Diamond.

Dataset Source Link : https://www.kaggle.com/competitions/playground-series-s3e8/data?select=train.csv

4 3.0. Data Exploration

[ ]:
```
df.isnull().sum()
```

[ ]:
```
id         0
carat      0
cut        0
color      0
```

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

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193573 entries, 0 to 193572
Data columns (total 11 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   id       193573 non-null  int64
 1   carat    193573 non-null  float64
 2   cut      193573 non-null  object
 3   color    193573 non-null  object
 4   clarity  193573 non-null  object
 5   depth    193573 non-null  float64
 6   table    193573 non-null  float64
 7   x        193573 non-null  float64
 8   y        193573 non-null  float64
 9   z        193573 non-null  float64
 10  price    193573 non-null  int64
dtypes: float64(6), int64(2), object(3)
memory usage: 16.2+ MB
```

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

[ ]:

	id	carat	cut	color	clarity	depth	table	x	y	z	price
0	0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

- Lets drop the id column

[ ]:
```
df=df.drop(labels=['id'],axis=1)
df.head()
```

[ ]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772

```
3   0.32      Ideal     G     VS1   61.6   56.0  4.38  4.41  2.71    666
4   1.70    Premium     G     VS2   62.6   59.0  7.65  7.61  4.77  14453
```

- check for duplicated records

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
0
```

segregate numerical and categorical columns

[ ]:
```
numerical_columns=df.columns[df.dtypes!='object']
categorical_columns=df.columns[df.dtypes=='object']
print("Numerical columns:",numerical_columns)
print('Categorical Columns:',categorical_columns)
```

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

[ ]:
```
df[categorical_columns].describe()
```

[ ]:
```
           cut   color clarity
count   193573  193573  193573
unique       5       7       8
top      Ideal       G     SI1
freq     92454   44391   53272
```

[ ]:
```
df['cut'].value_counts()
```

[ ]:
```
Ideal        92454
Premium      49910
Very Good    37566
Good         11622
Fair          2021
Name: cut, dtype: int64
```

[ ]:
```
df['color'].value_counts()
```

[ ]:
```
G    44391
E    35869
F    34258
H    30799
D    24286
I    17514
J     6456
Name: color, dtype: int64
```

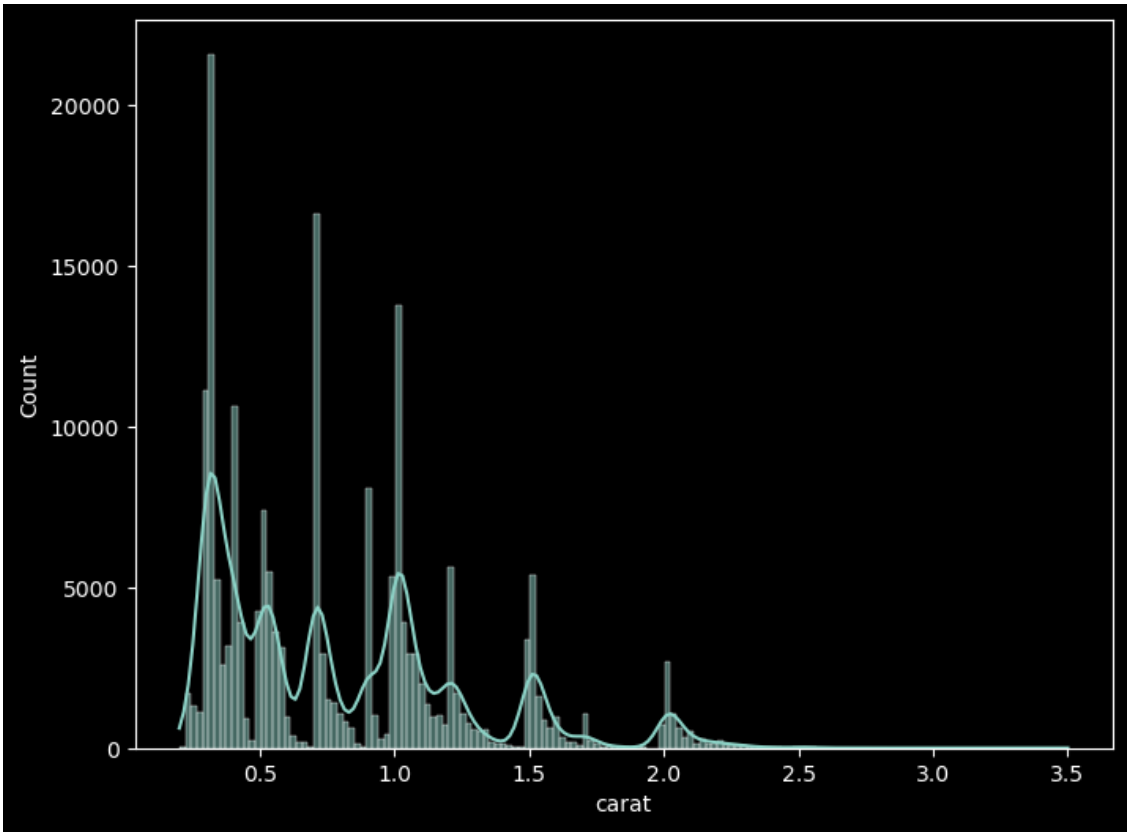
[ ]:
```
df['clarity'].value_counts()
```

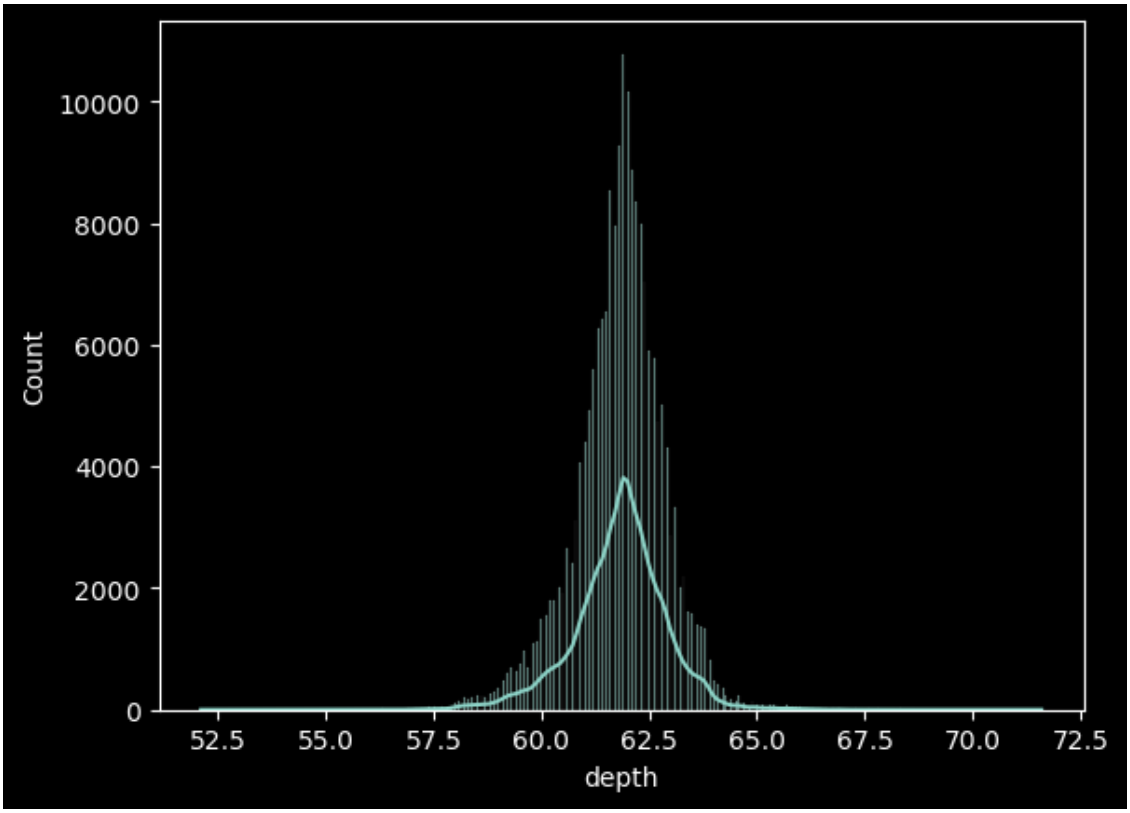
[ ]:
```
SI1     53272
VS2     48027
VS1     30669
SI2     30484
VVS2    15762
VVS1    10628
IF       4219
I1        512
Name: clarity, dtype: int64
```

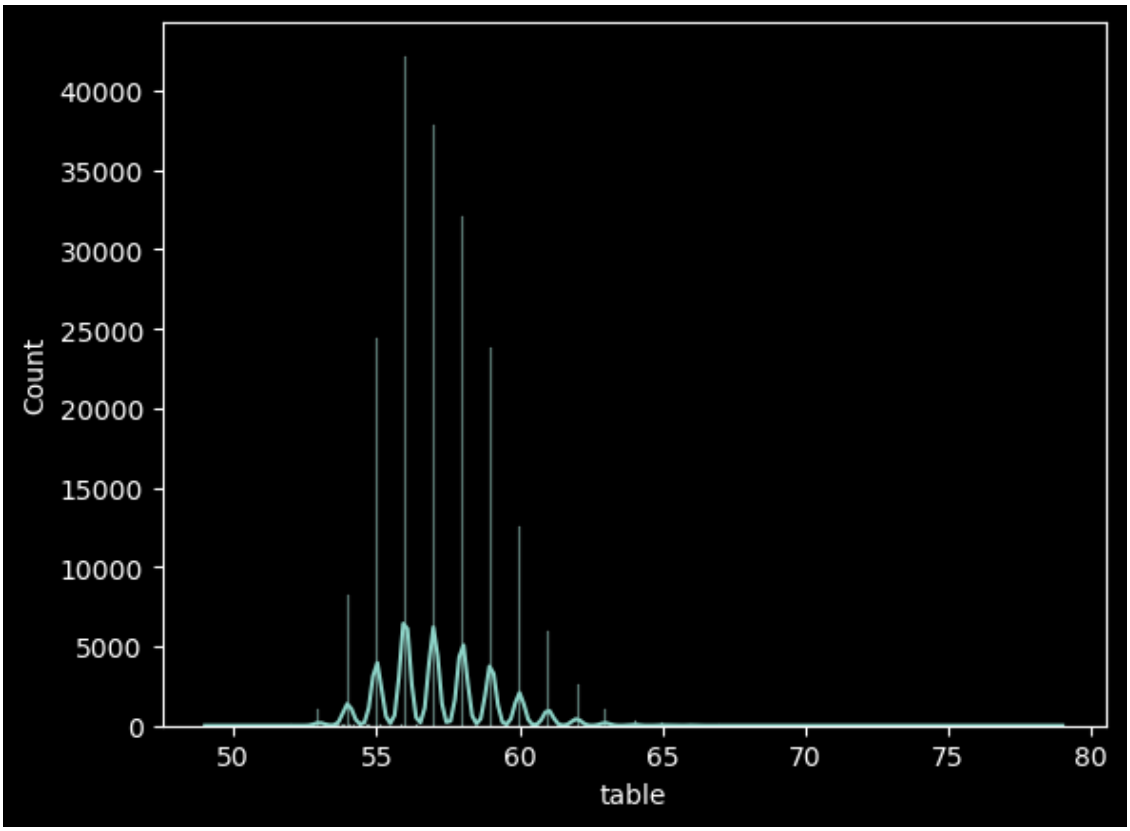
5 4.0. Data Visualization

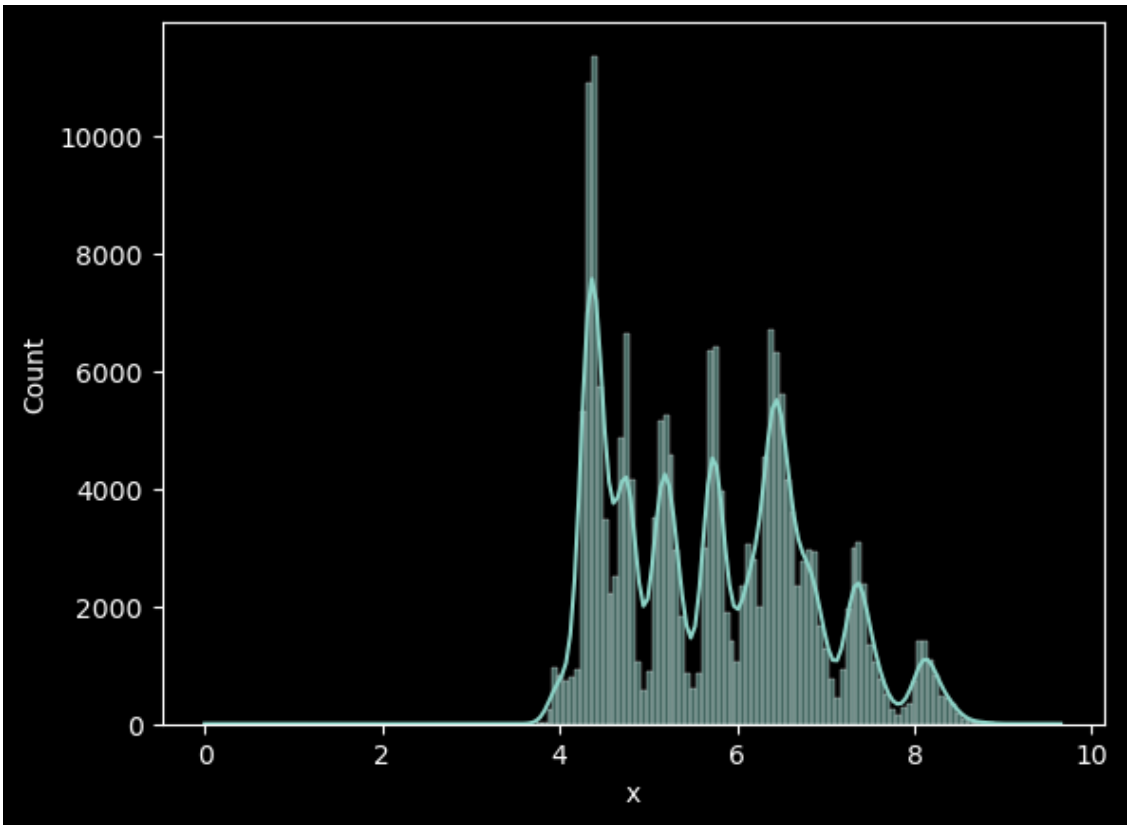
[ ]:
```
plt.style.use('dark_background')
```

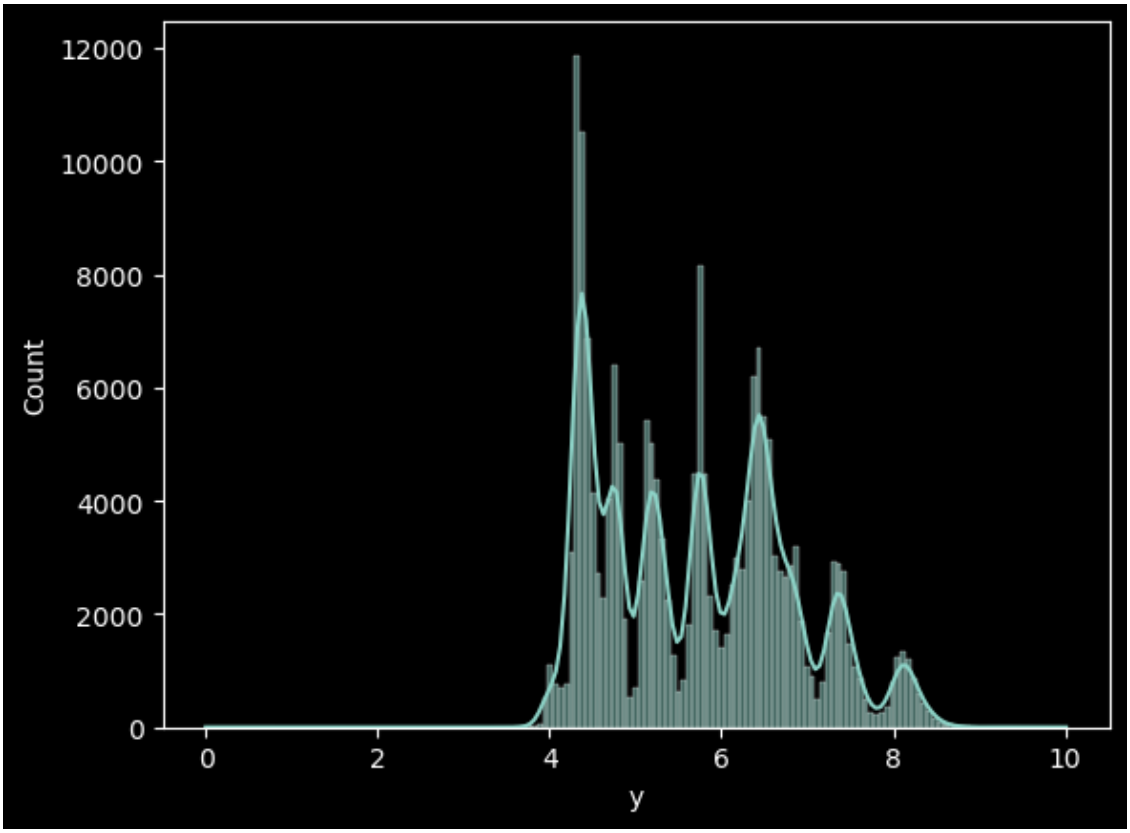
[ ]:
```
plt.figure(figsize=(8,6))
x=0
for i in numerical_columns:
    sns.histplot(data=df,x=i,kde=True)
    print('\n')
    plt.show()
```

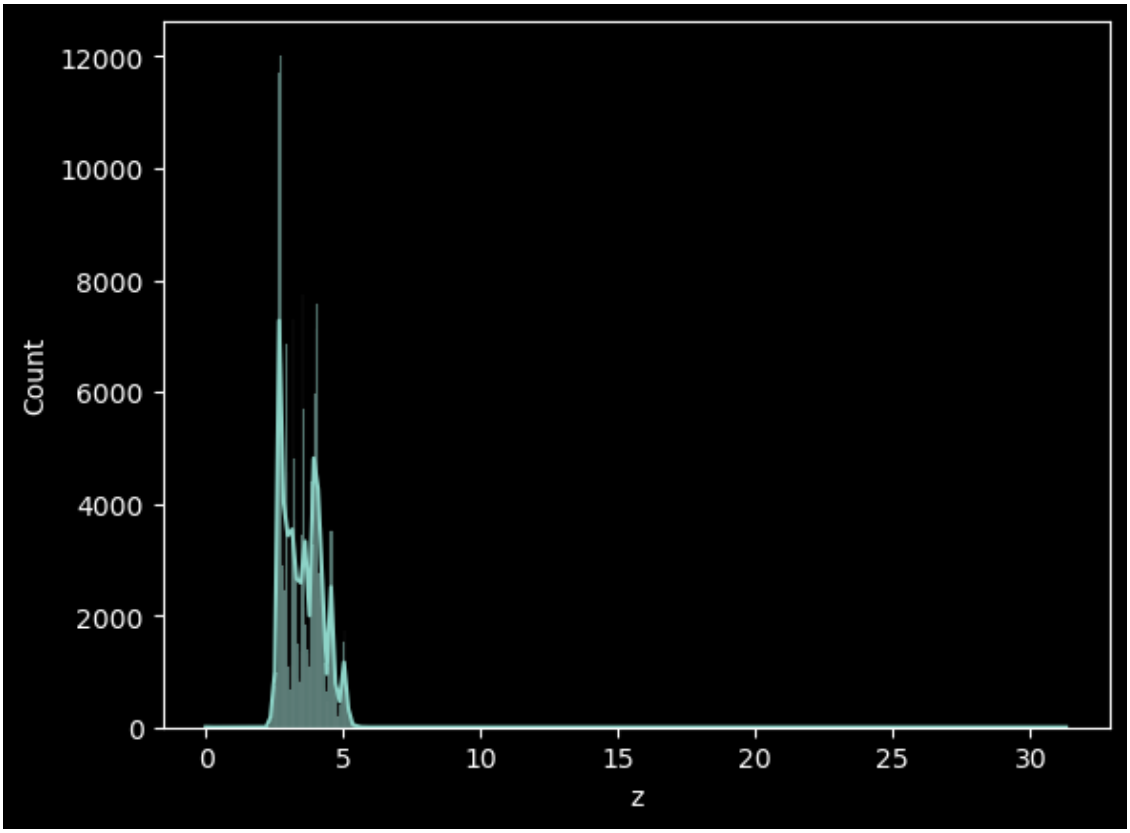


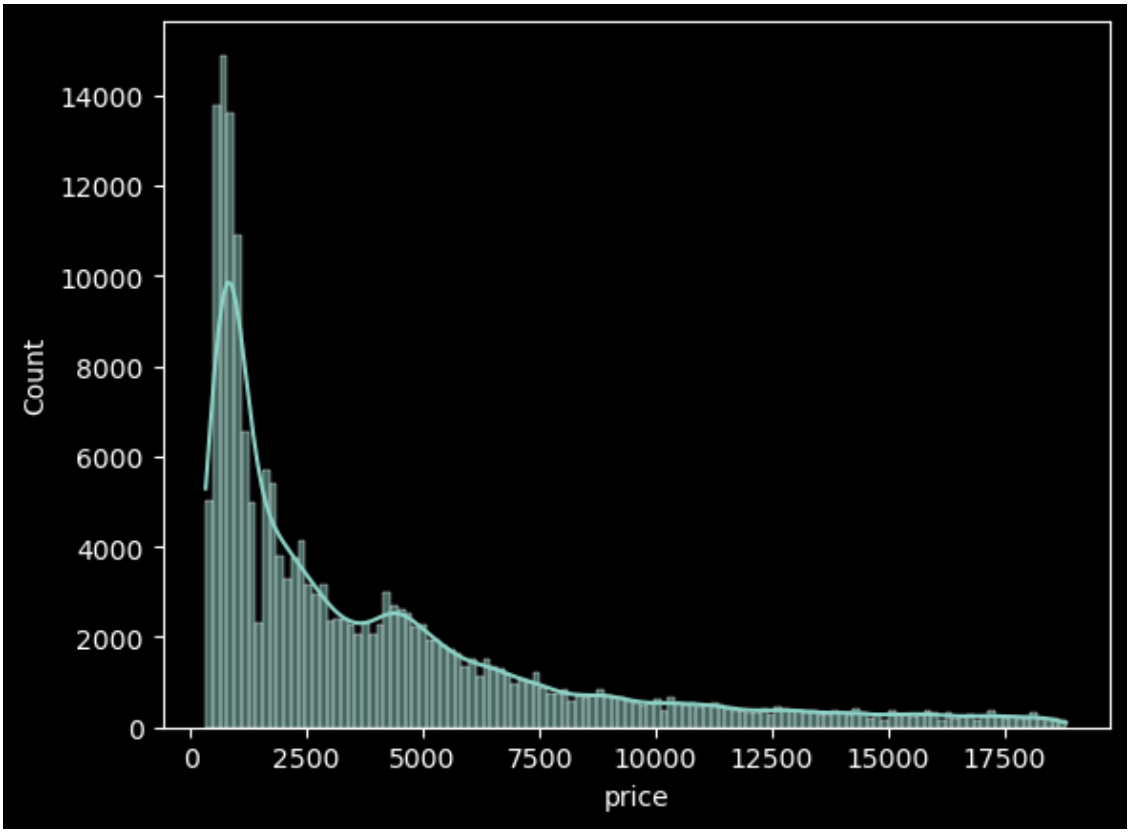

10000
8000
6000
4000
2000
0
Count
52.5
55.0
57.5
60.0
62.5
65.0
67.5
70.0
72.5
depth


40000
35000
30000
25000
20000
15000
10000
5000
0
Count
50
55
60
65
70
75
80
table


Count
10000
8000
6000
4000
2000
0
0
2
4
6
8
10
x

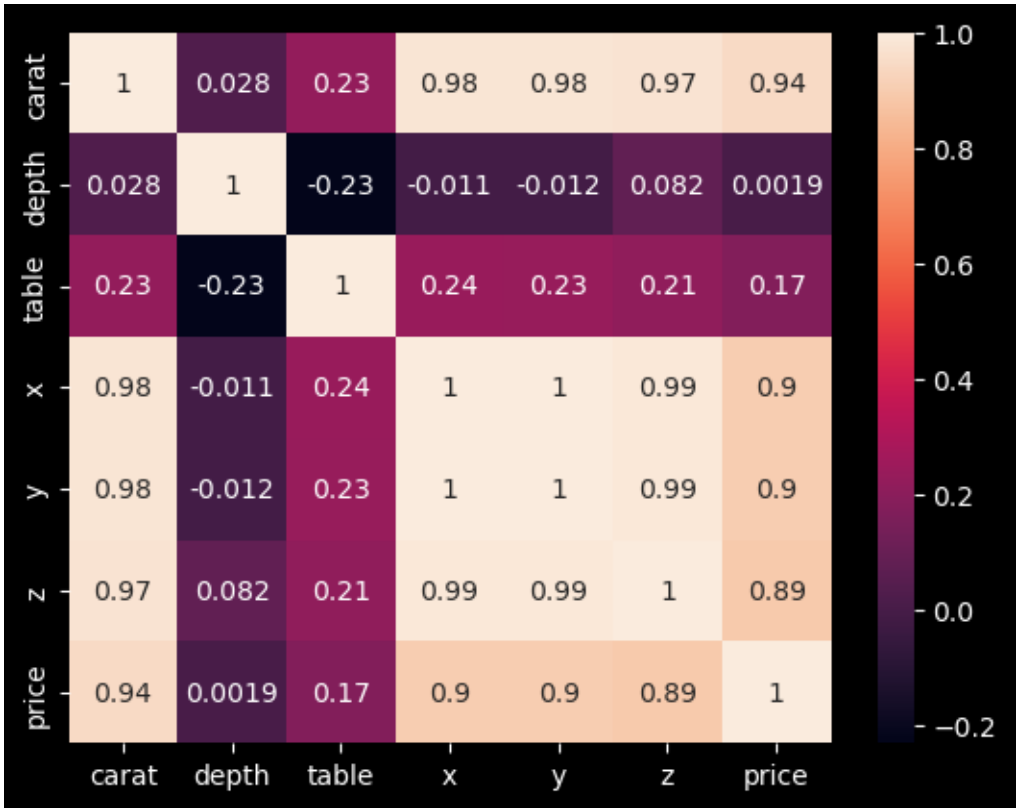

12000
10000
8000
6000
4000
2000
0
Count
0
2
4
6
8
10
y


12000
10000
8000
6000
4000
2000
0
Count
0
5
10
15
20
25
30
z



[ ]:
```
## correlation
sns.heatmap(df[numerical_columns].corr(),annot=True)
```

[ ]: <AxesSubplot: >



6 5.0. Feature Selection

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

[ ]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

[ ]:
```
df['cut'].unique()
```

[ ]:
```
array(['Premium', 'Very Good', 'Ideal', 'Good', 'Fair'], dtype=object)
```

[ ]:
```
cut_map={"Fair":1,"Good":2,"Very Good":3,"Premium":4,"Ideal":5}
```

[ ]:
```
df['clarity'].unique()
```

[ ]:
```
array(['VS2', 'SI2', 'VS1', 'SI1', 'IF', 'VVS2', 'VVS1', 'I1'],
      dtype=object)
```

[ ]:
```
df['clarity'].unique()
```

[ ]:
```
array(['VS2', 'SI2', 'VS1', 'SI1', 'IF', 'VVS2', 'VVS1', 'I1'],
      dtype=object)
```

[ ]:
```
df['clarity'].unique()
```

[ ]:
```
array(['VS2', 'SI2', 'VS1', 'SI1', 'IF', 'VVS2', 'VVS1', 'I1'],
      dtype=object)
```

[ ]:
```
df['color'].unique()
```

[ ]:
```
array(['F', 'J', 'G', 'E', 'D', 'H', 'I'], dtype=object)
```

[ ]:
```
color_map = {"D":1 ,"E":2 ,"F":3 , "G":4 ,"H":5 , "I":6, "J":7}
```

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

[ ]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	4	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	3	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	5	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	5	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	1.70	4	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

7 5.0. Model Training

[ ]:
```
df2 = pd.read_csv('./data/gemstone.csv')
df.head()
```

[ ]:

	carat	cut	color	clarity	depth	table	x	y	z	price
0	1.52	Premium	F	VS2	62.2	58.0	7.27	7.33	4.55	13619
1	2.03	Very Good	J	SI2	62.0	58.0	8.06	8.12	5.05	13387
2	0.70	Ideal	G	VS1	61.2	57.0	5.69	5.73	3.50	2772
3	0.32	Ideal	G	VS1	61.6	56.0	4.38	4.41	2.71	666
4	1.70	Premium	G	VS2	62.6	59.0	7.65	7.61	4.77	14453

[ ]:
```
df2=df2.drop(labels=['id'],axis=1)
## Independent and dependent features
X = df.drop(labels=['price'],axis=1)
Y = df[['price']]
Y
```

[ ]:

	price
0	13619
1	13387
2	2772

```
3          666
4        14453
…          …
193568    1130
193569    2874
193570    3036
193571     681
193572    2258

[193573 rows x 1 columns]
```

7.0.1 Segregating and categorical variables

[ ]:
```
categorical_cols = X.select_dtypes(include='object').columns
numerical_cols = X.select_dtypes(exclude='object').columns
```

7.0.2 Define the custom ranking for each ordinal variable

[ ]:
```
cut_categories = ['Fair', 'Good', 'Very Good','Premium','Ideal']
color_categories = ['D', 'E', 'F', 'G', 'H', 'I', 'J']
clarity_categories = ['I1','SI2','SI1','VS2','VS1','VVS2','VVS1','IF']
```

7.0.3 Numerical Pipeline

[ ]:
```
num_pipeline=Pipeline(
    steps=[
    ('imputer',SimpleImputer(strategy='median')),
    ('scaler',StandardScaler())

    ]

)

# Categorigal Pipeline
cat_pipeline=Pipeline(
    steps=[
    ('imputer',SimpleImputer(strategy='most_frequent')),
   ␣
 ↪('ordinalencoder',OrdinalEncoder(categories=[cut_categories,color_categories,clarity_catego
    ('scaler',StandardScaler())
    ]

)

preprocessor=ColumnTransformer([
('num_pipeline',num_pipeline,numerical_cols),
('cat_pipeline',cat_pipeline,categorical_cols)
```

```
])
```

7.0.4 Train test split

[ ]:
```
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=30)
```

[ ]:
```
X_train=pd.DataFrame(preprocessor.fit_transform(X_train),columns=preprocessor.
↪get_feature_names_out())
X_test=pd.DataFrame(preprocessor.transform(X_test),columns=preprocessor.
↪get_feature_names_out())
```

[ ]:
```
X_train.head()
```

[ ]:

	num_pipeline__carat	num_pipeline__depth	num_pipeline__table	\
0	-0.975439	-0.849607	-0.121531	
1	0.235195	1.833637	-0.121531	
2	0.494617	0.815855	0.399800	
3	-1.018676	0.260701	0.921131	
4	-0.953821	-0.664555	-0.642862	

	num_pipeline__x	num_pipeline__y	num_pipeline__z	cat_pipeline__cut	\
0	-1.042757	-1.080970	-1.123150	0.874076	
1	0.318447	0.279859	0.485354	-2.144558	
2	0.570855	0.606458	0.673737	-0.132136	
3	-1.214034	-1.244270	-1.195605	-0.132136	
4	-1.069801	-1.044681	-1.094168	0.874076	

	cat_pipeline__color	cat_pipeline__clarity
0	1.528722	1.352731
1	-0.935071	-0.646786
2	0.296826	0.686225
3	0.296826	0.019720
4	2.144670	1.352731

[ ]:
```
## Model Training

from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error
```

[ ]:
```
regression=LinearRegression()
regression.fit(X_train,y_train)
```

[ ]:
```
LinearRegression()
```

[ ]:
```
regression.coef_
```

[ ]:
```
array([[ 6433.66003594,  -132.75843566,   -70.42922179, -1720.30971463,
         -499.29302619,   -63.39317848,    72.44537247,  -460.41604642,
          650.76431652]])
```

[ ]:
```
regression.intercept_
```

[ ]:
```
array([3970.76628955])
```

[ ]:
```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

7.0.5 Train multiple models

7.0.6 Model Ecaluation

[ ]:
```
models={
    'LinearRegression':LinearRegression(),
    'Lasso':Lasso(),
    'Ridge':Ridge(),
    'Elasticnet':ElasticNet()
}
trained_model_list=[]
model_list=[]
r2_list=[]

for i in range(len(list(models))):
    model=list(models.values())[i]
    model.fit(X_train,y_train)

    #Make Predictions
    y_pred=model.predict(X_test)

    mae, rmse, r2_square=evaluate_model(y_test,y_pred)

    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model Training Performance')
    print("RMSE:",rmse)
    print("MAE:",mae)
    print("R2 score",r2_square*100)
```

```
    r2_list.append(r2_square)

    print('='*35)
    print('\n')
```

```
LinearRegression
Model Training Performance
RMSE: 1013.9047094344002
MAE: 674.025511579685
R2 score 93.68908248567512
==================================


Lasso
Model Training Performance
RMSE: 1013.8784226767013
MAE: 675.0716923362162
R2 score 93.68940971841704
==================================


Ridge
Model Training Performance
RMSE: 1013.9059272771628
MAE: 674.0555800798244
R2 score 93.6890673250594
==================================


Elasticnet
Model Training Performance
RMSE: 1533.4162456064048
MAE: 1060.7368759154729
R2 score 85.56494831165182
==================================
```

8 Reference:

PWSKILLS

9 Thank You!