```
In [110…: ##Project Objectives
#This project demonstrates the process of building a machine learning model
#to predict diamond prices based on their attributes. By performing exploratory data analysis,
#preprocessing, and model training, we achieved a robust predictive model. The insights gained from
#this project canbe used to make data-driven decisions in the diamond industry.
```

```
In [90]: # Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
```

```
In [75]: #Load the Dataset
data = pd.read_csv('https://raw.githubusercontent.com/dlab-berkeley/Python-Machine-Learning/refs/heads/main/data/diamonds.csv')
data.head()
```

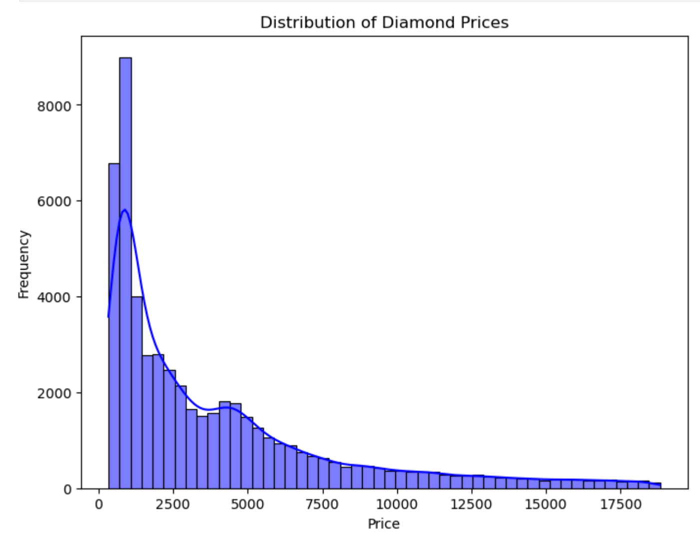
Out[75]:

	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

```
In [76]: #Data Cleaning
# Drop unnecessary columns
data = data.drop(columns=['Unnamed: 0'])
```

```
In [ ]: # Data Visualizations
```

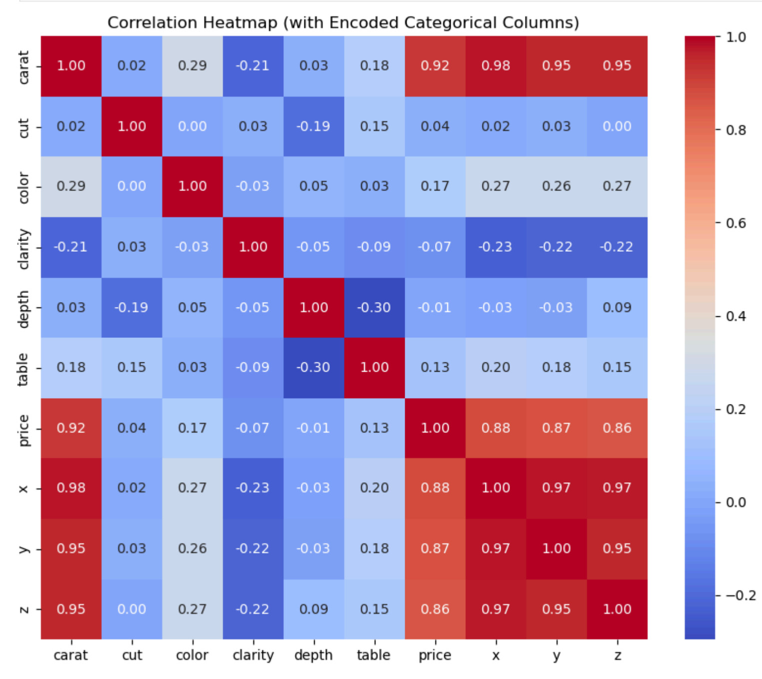
```
In [83]: #1 Distribution of the target variable (Price)
plt.figure(figsize=(8, 6))
sns.histplot(data['price'], bins=50, kde=True, color='blue')
plt.title('Distribution of Diamond Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



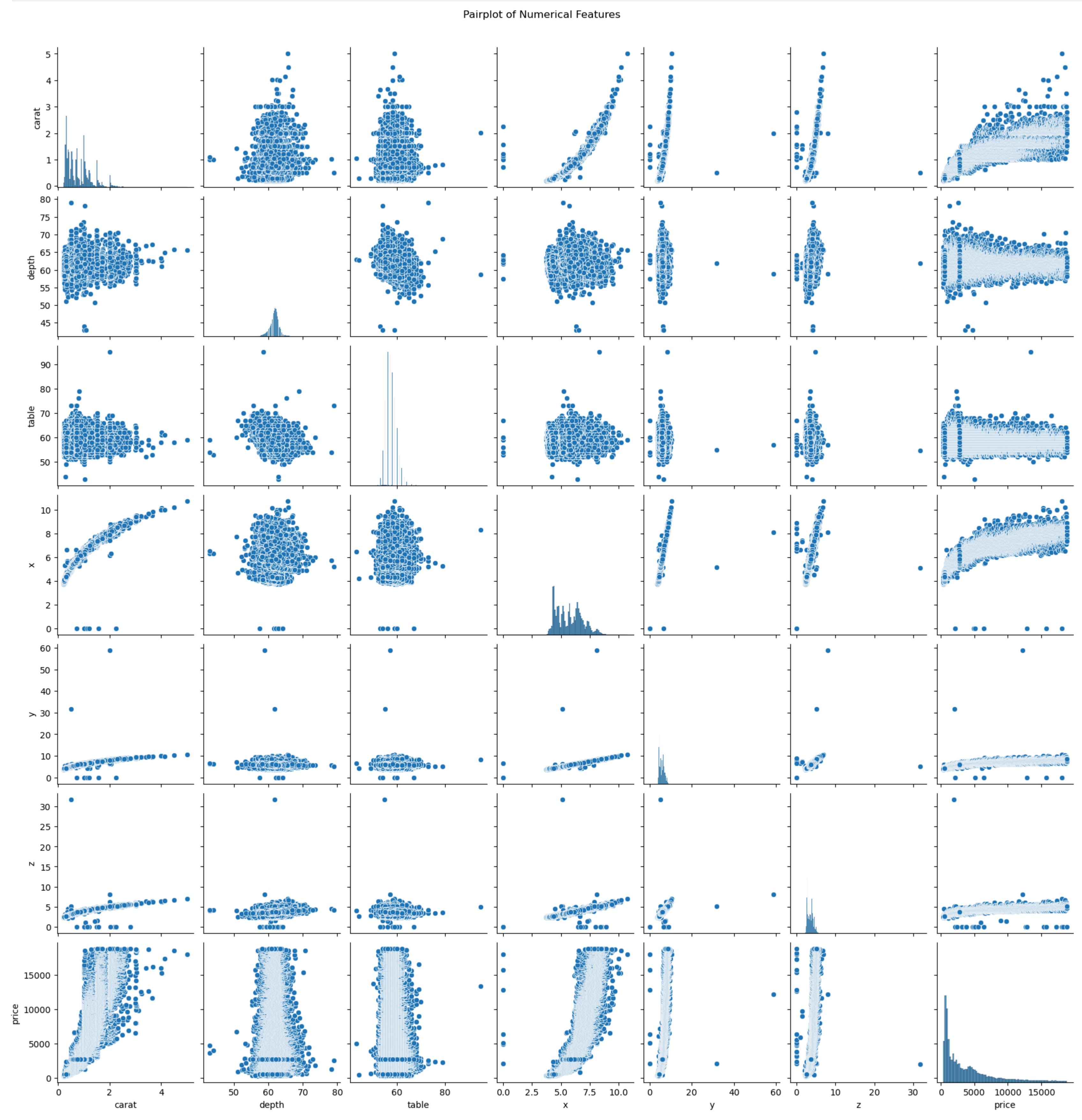

```
In [91]: # 2. Correlation heatmap
# Create a copy of the dataset to avoid modifying the original
data_encoded = data.copy()

# Encode categorical columns
label_encoders = {}
for col in ['cut', 'color', 'clarity']:
    le = LabelEncoder()
    data_encoded[col] = le.fit_transform(data_encoded[col])
    label_encoders[col] = le

# Correlation heatmap with encoded categorical columns
plt.figure(figsize=(10, 8))
corr = data_encoded.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap (with Encoded Categorical Columns)')
plt.show()
```




```
In [92]: # 3. Pairplot for numerical features
sns.pairplot(data[['carat', 'depth', 'table', 'x', 'y', 'z', 'price']])
plt.suptitle('Pairplot of Numerical Features', y=1.02)
plt.show()
```

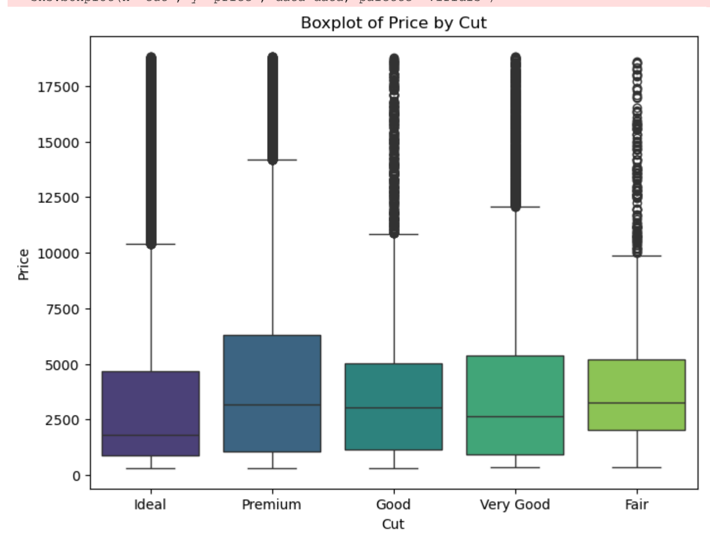



```
In [99]: # 4. Boxplot of price by cut
plt.figure(figsize=(8, 6))
sns.boxplot(x='cut', y='price', data=data, palette='viridis')
plt.title('Boxplot of Price by Cut')
plt.xlabel('Cut')
plt.ylabel('Price')
plt.show()
```

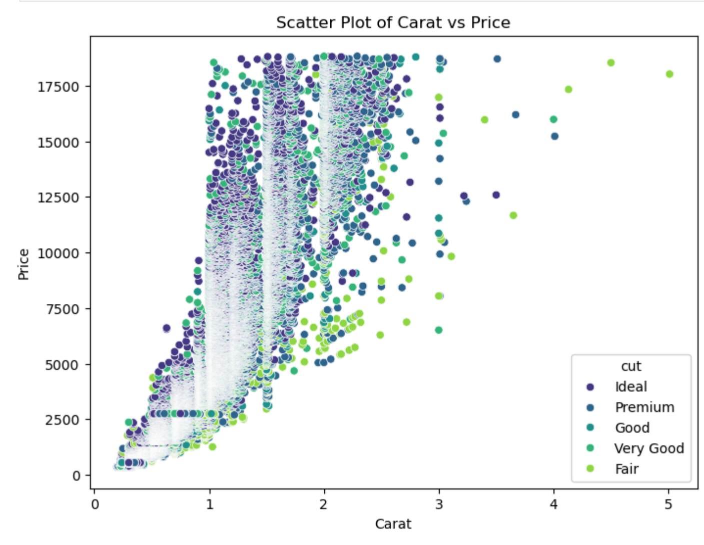
```
C:\Users\lsherpa\AppData\Local\Temp\ipykernel_1728\3423786036.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='cut', y='price', data=data, palette='viridis')
```




```
In [100…: # 5. Scatter plot of carat vs price
plt.figure(figsize=(8, 6))
sns.scatterplot(x='carat', y='price', data=data, hue='cut', palette='viridis')
plt.title('Scatter Plot of Carat vs Price')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.show()
```




```
In [77]: #Preparing Data Model
#Define features and target
X = data.drop(columns=['price'])
y = data['price']
```

```
In [78]: #Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

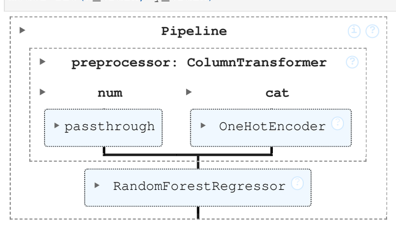
```
In [79]: #Data preprocessing
#Handle categorical variables
categorical_features = ['cut', 'color', 'clarity']
numerical_features = ['carat', 'depth', 'table', 'x', 'y', 'z']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numerical_features),
        ('cat', OneHotEncoder(), categorical_features)
    ])
```

```
In [80]: #Creat a pipeline with preprocessing and model
#create a data model
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(random_state=42))
])
```

```
In [81]: #train a model
model.fit(X_train, y_train)
```

Out[81]:




```
In [102…: #make predictions
y_pred = model.predict(X_test)
```

```
In [104…: # Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

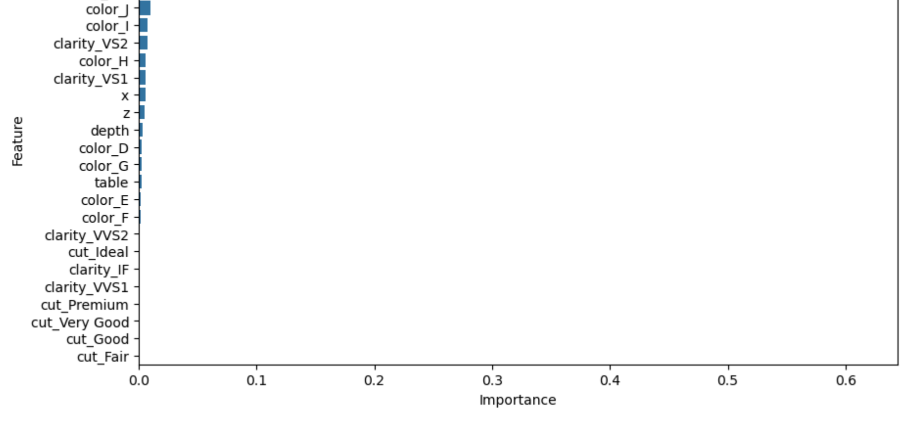
```
In [105…: print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 270.19445885713696
Mean Squared Error: 302594.57901389623
R-squared: 0.9809650891568692
```

```
In [107…: # Feature importance for Random Foreset model
if isinstance(model.named_steps['regressor'], RandomForestRegressor):
    importances = model.named_steps['regressor'].feature_importances_
    feature_names = numerical_features + list(model.named_steps['preprocessor'].named_transformers_['cat'].get_feature_names_out(categorical_features))
    feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
    print("\nFeature Importance:")
    print(feature_importance_df.sort_values(by='Importance', ascending=False))
```

```
Feature Importance:
          Feature  Importance
0           carat    0.613422
4               y    0.272781
21   clarity_SI2    0.019434
18    clarity_I1    0.014999
20   clarity_SI1    0.014462
17       color_J    0.010312
16       color_I    0.007273
23   clarity_VS2    0.007092
15       color_H    0.005649
22   clarity_VS1    0.005627
3               x    0.005504
5               z    0.005199
1           depth    0.003222
11       color_D    0.002837
14       color_G    0.002594
2           table    0.002185
12       color_E    0.001639
13       color_F    0.001427
25  clarity_VVS2    0.001117
8       cut_Ideal    0.001001
19    clarity_IF    0.000697
24  clarity_VVS1    0.000492
9     cut_Premium    0.000359
10  cut_Very Good    0.000282
7        cut_Good    0.000246
6        cut_Fair    0.000146
```

```
In [108…: # Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature',data=feature_importance_df.sort_values(by='Importance', ascending=False))
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```


Feature Importance
carat
y
clarity_SI2
clarity_I1
clarity_SI1
color_J
color_I
clarity_VS2
color_H
clarity_VS1
x
z
depth
color_D
color_G
table
color_E
color_F
clarity_VVS2
cut_Ideal
clarity_IF
clarity_VVS1
cut_Premium
cut_Very Good
cut_Good
cut_Fair
Feature
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Importance