Diamond Price Prediction 2024 Project

Shai Training

1. Introduction

In the realm of data analytics, the predictive modeling of diamond prices stands as a quintessential exercise, offering a rich tapestry of data attributes to explore and analyze. This document encapsulates a comprehensive journey through the realms of data preprocessing, exploratory data analysis (EDA), model building, and submission preparation within the context of predicting diamond prices.

The dataset under scrutiny encompasses a diverse array of features, ranging from intrinsic characteristics such as carat weight, cut quality, color, and clarity to physical dimensions like length, width, and depth. With nearly 54,000 diamonds at our disposal, this dataset presents an invaluable opportunity for budding data scientists and seasoned analysts alike to delve deep into the nuances of predictive modeling.
This document serves as a roadmap, guiding the reader through each phase of the analytical process, from initial data exploration to the fine-tuning of predictive models. Along the way, we'll uncover insights, address challenges, and ultimately strive to construct a robust model capable of accurately predicting diamond prices.

1.1 Background

The diamond industry has long captivated both consumers and investors with its allure, representing not only a symbol of enduring love but also a lucrative market for investors and traders. Understanding the factors influencing diamond prices is of paramount importance in this industry, as it enables stakeholders to make informed decisions regarding pricing, marketing, and investment.
In this backdrop, the dataset containing prices and attributes of over 54,000 diamonds emerges as a treasure trove for data analysts and enthusiasts. By leveraging advanced analytics techniques, we aim to unravel the intricate relationships between diamond attributes and prices, providing actionable insights for industry stakeholders.

2. Objectives

The primary objectives of this document are as follows:

1. Explore the dataset: Conduct exploratory data analysis (EDA) to gain insights into the distribution, correlation, and significance of various diamond attributes.
2. Preprocess the data: Cleanse, transform, and engineer features to enhance the quality and relevance of input data for modeling.
3. Build predictive models: Utilize machine learning algorithms to develop predictive models capable of accurately estimating diamond prices.
4. Evaluate model performance: Assess the performance of the developed models using appropriate evaluation metrics and techniques.
5. Generate submissions: Prepare submissions in the required format for the competition, incorporating the insights gained and models developed throughout the analysis.

2- Dataset Overview

Our project revolves around a dataset comprising data on over 54,000 diamonds. It includes essential attributes such as carat weight, cut quality, color, clarity, and physical dimensions alongside their corresponding prices. This real-world dataset serves as the foundation for our data analytics efforts, aiming to extract insights and build predictive models for accurate diamond price estimation.

3- Framing the Problem

In the realm of diamond pricing, understanding the intricate interplay of various factors is paramount. Leveraging domain knowledge reveals that diamond prices are influenced by several key attributes, often encapsulated by the famous "4Cs" - Cut, Color, Clarity, and Carat Weight. Each of these factors contributes uniquely to the overall value and desirability of a diamond.

Utilizing this domain knowledge, the task at hand is to develop a predictive model capable of accurately estimating diamond prices based on these fundamental attributes. By framing the problem in this context, we aim to explore the relationships between the 4Cs and diamond prices, discerning patterns that can guide pricing strategies and market analysis.

Through data analytics techniques, we seek to uncover insights that elucidate the nuances of diamond pricing dynamics, ultimately empowering stakeholders to make informed decisions in the vibrant diamond market.

4- Selecting a Performance Measure

To gauge the effectiveness of our predictive models in estimating diamond prices, we opt for Root Mean Squared Error (RMSE) as our performance measure. RMSE calculates the average disparity between predicted and actual prices. By minimizing RMSE, we strive to enhance the accuracy of our models across various diamond attributes, including cut, color, clarity, and carat weight. This choice ensures our models meet the stringent standards demanded by the dynamic diamond market.

5- Exploratory Data Analysis (EDA)

Exploring the dataset is a fundamental step in understanding its characteristics and preparing for further analysis. Here's an overview of our exploratory analysis:

5.1 Previewing Data: We start by examining the first few rows of the dataset to get a glimpse of its structure and content.

5.2 Inspecting Tail End: Similarly, we review the last few rows to ensure data consistency and observe any potential patterns.

5.3 Data Summary: Utilizing the info() function, we obtain a summary of the dataset, including the data types and non-null counts for each column as in below:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43152 entries, 0 to 43151
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Id        43152 non-null  int64
 1   carat     43152 non-null  float64
 2   cut       43152 non-null  object
 3   color     43152 non-null  object
 4   clarity   43152 non-null  object
 5   depth     43152 non-null  float64
 6   table     43152 non-null  float64
 7   price     43152 non-null  int64
 8   x         43152 non-null  float64
 9   y         43152 non-null  float64
 10  z         43152 non-null  float64
dtypes: float64(6), int64(2), object(3
memory usage: 3.6+ MB
None
```

5.4 Number of Instances and Features: We ascertain the total number of instances and features in the dataset. In our dataset :

Number of instances: 43152, Number of features: 11

5.5 Statistical Summary: Employing describe(), we generate descriptive statistics for numerical features, revealing insights into central tendency, dispersion, and distribution as in below:

	Id	carat	depth	table	price
count	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000
mean	21576.500000	0.797855	61.747177	57.458347	3929.491912
std	12457.053745	0.473594	1.435454	2.233904	3985.527795
min	1.000000	0.200000	43.000000	43.000000	326.000000
25%	10788.750000	0.400000	61.000000	56.000000	947.750000
50%	21576.500000	0.700000	61.800000	57.000000	2401.000000
75%	32364.250000	1.040000	62.500000	59.000000	5312.000000
max	43152.000000	5.010000	79.000000	95.000000	18823.000000

	x	y	z
count	43152.000000	43152.000000	43152.000000
mean	5.731568	5.735018	3.538568
std	1.121279	1.148809	0.708238
min	0.000000	0.000000	0.000000
25%	4.710000	4.720000	2.910000
50%	5.700000	5.710000	3.530000
75%	6.540000	6.540000	4.040000
max	10.740000	58.900000	31.800000

5.6 Categorical Feature Counts: We explore the frequency distribution of categorical features such as cut, color, and clarity to discern any prevalent categories.

```
cut
Ideal          17203
Premium        11113
Very Good       9658
Good            3881
Fair            1297
Name: count, dtype: int64
color
G    9060
E    7832
F    7633
H    6651
D    5421
I    4265
J    2290
Name: count, dtype: int64
clarity
SI1     10428
VS2      9824
SI2      7432
VS1      6475
VVS2     4041
VVS1     2904
IF       1442
I1        606
Name: count, dtype: int64
```

5.7 Handling Missing Values: We verify the absence of missing values or null data within the dataset. Our dataset does not contain any missing values or null data.

5.8 Duplicate Entries Check: Lastly, we ensure data integrity by confirming the absence of duplicate entries. Our dataset is free from duplicate entries, ensuring that each record is unique and contributes distinct information for analysis and modeling.

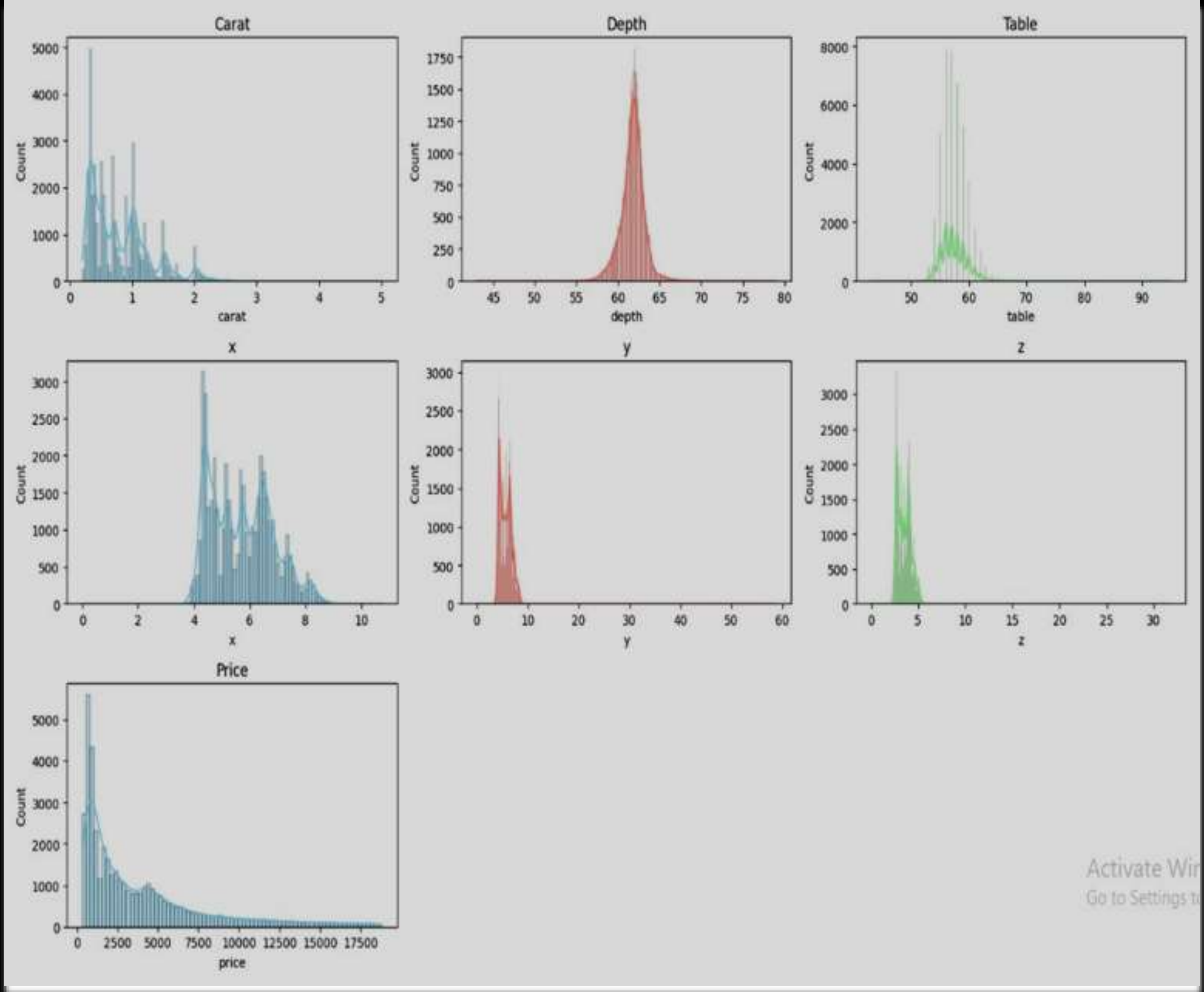
```
[9]
# Check for missing values
print(diamonds.isnull().sum())  # Our dataset does not contain any missing values or null data.

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

```
[10]
# Finding the duplicate data
diamonds.duplicated()

0        False
1        False
2        False
3        False
4        False
         ...
43147    False
43148    False
43149    False
43150    False
43151    False
Length: 43152, dtype: bool
```

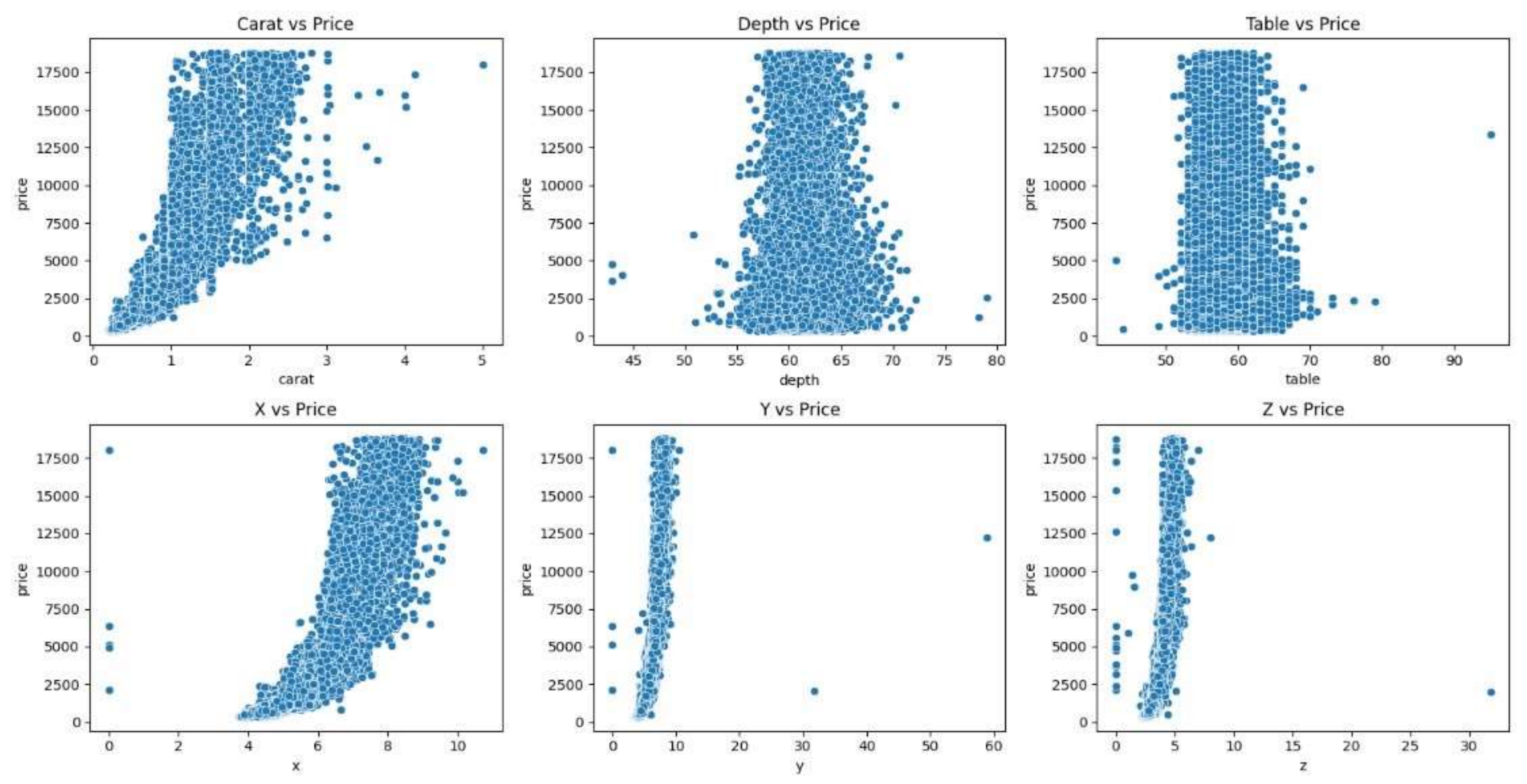
```
[11]
diamonds.duplicated().sum()

# Our dataset is free from duplicate entries, ensuring that each record is unique and contributes distinct information for analysis and modeling.

0
```

Data Analysis
&
Visualization

6.1 Visualize distributions of numerical features



6.2 Visualize relationships between numerical features and target variable

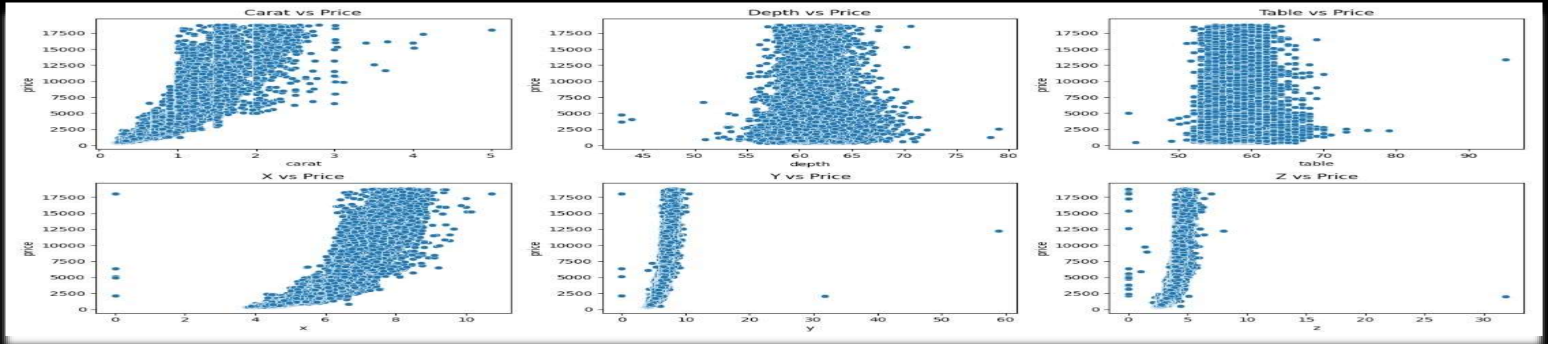


6.2 Visualize relationships between numerical features and target variable

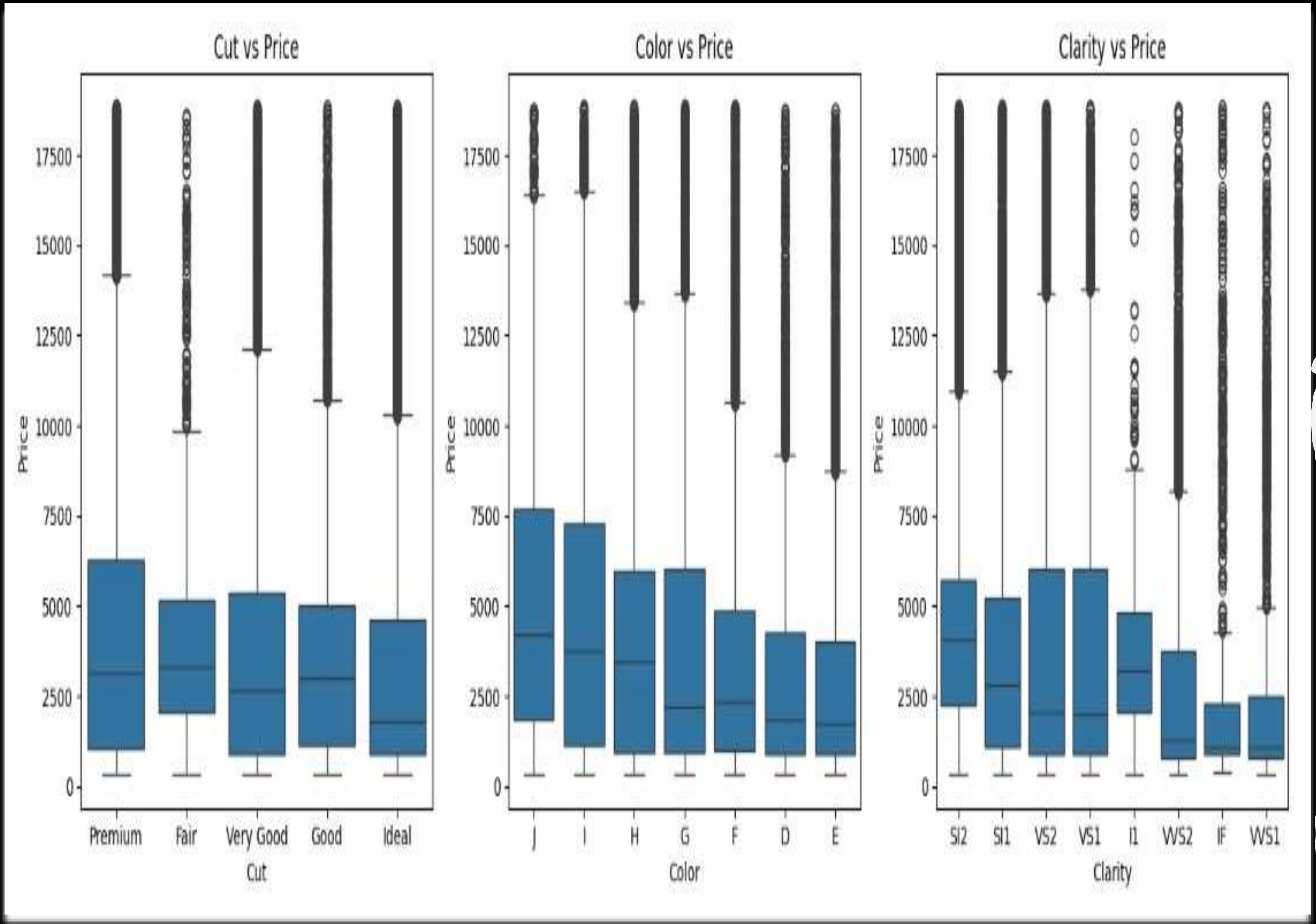
How These Insights Help in Model Training and Prediction:

- Feature Importance:
  - Carat and X (Length) are clearly the most important numerical predictors for price. These features should be given significant weight in the model.
  - Depth, Table, Y (Width), and Z (Depth) show less direct impact and may need to be combined with other features to extract meaningful patterns.

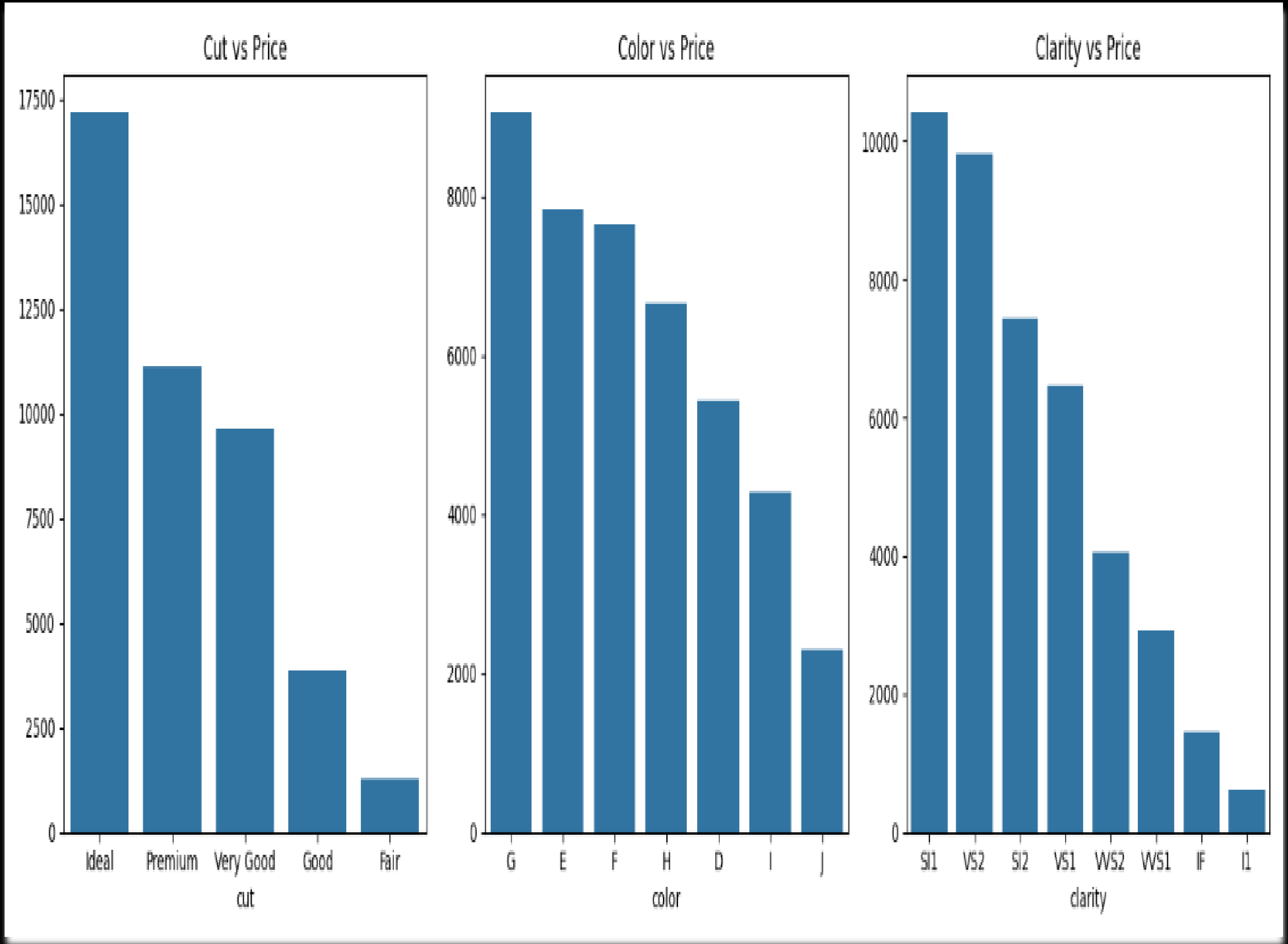
- The scatterplots reveal that while some features like carat and length (X) have a strong relationship with diamond price, others like depth and table do not. These insights guide us in prioritizing features, engineering new ones, and selecting appropriate models to improve the accuracy of price predictions.



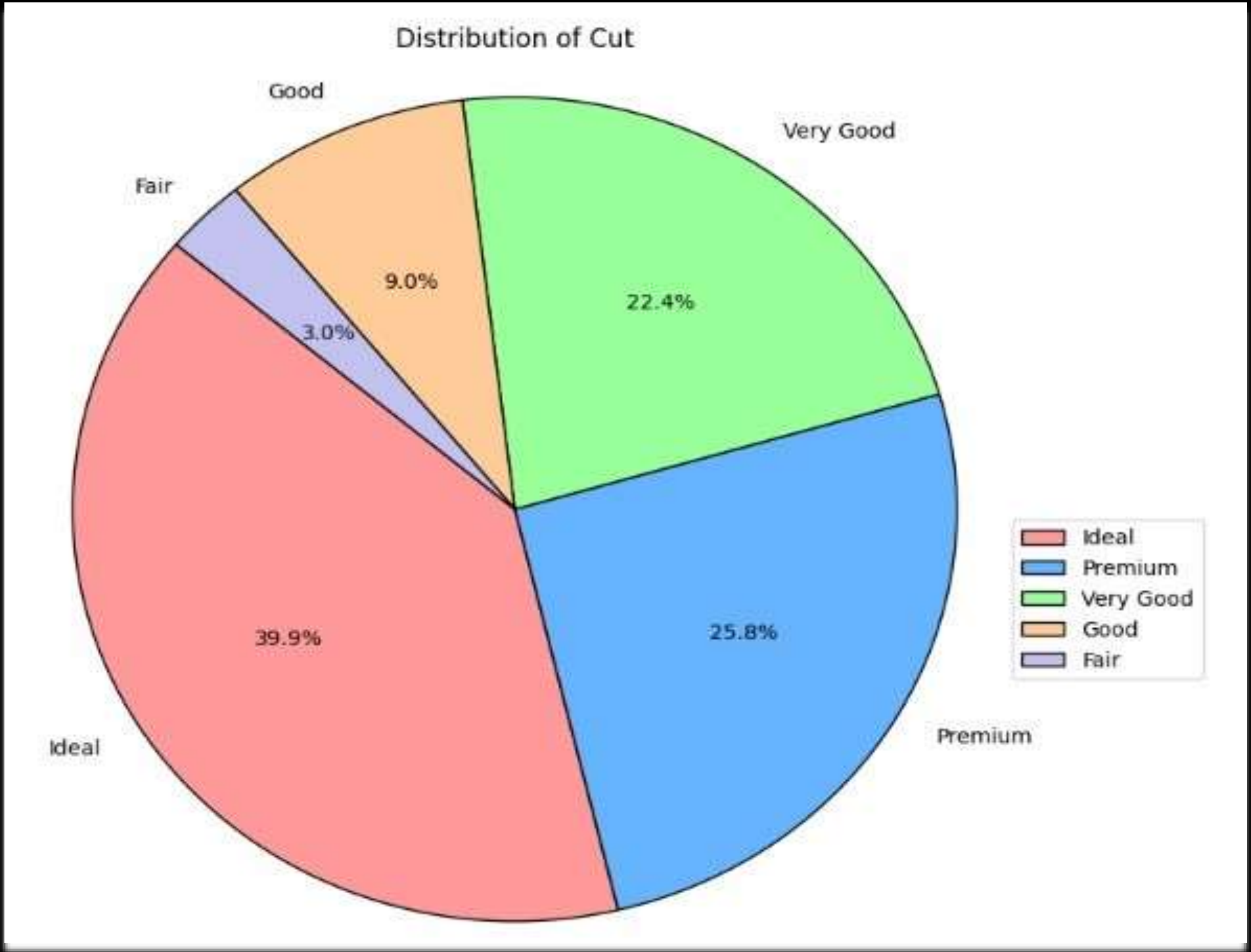
6.3 Visualize relationships between categorical features and target variable



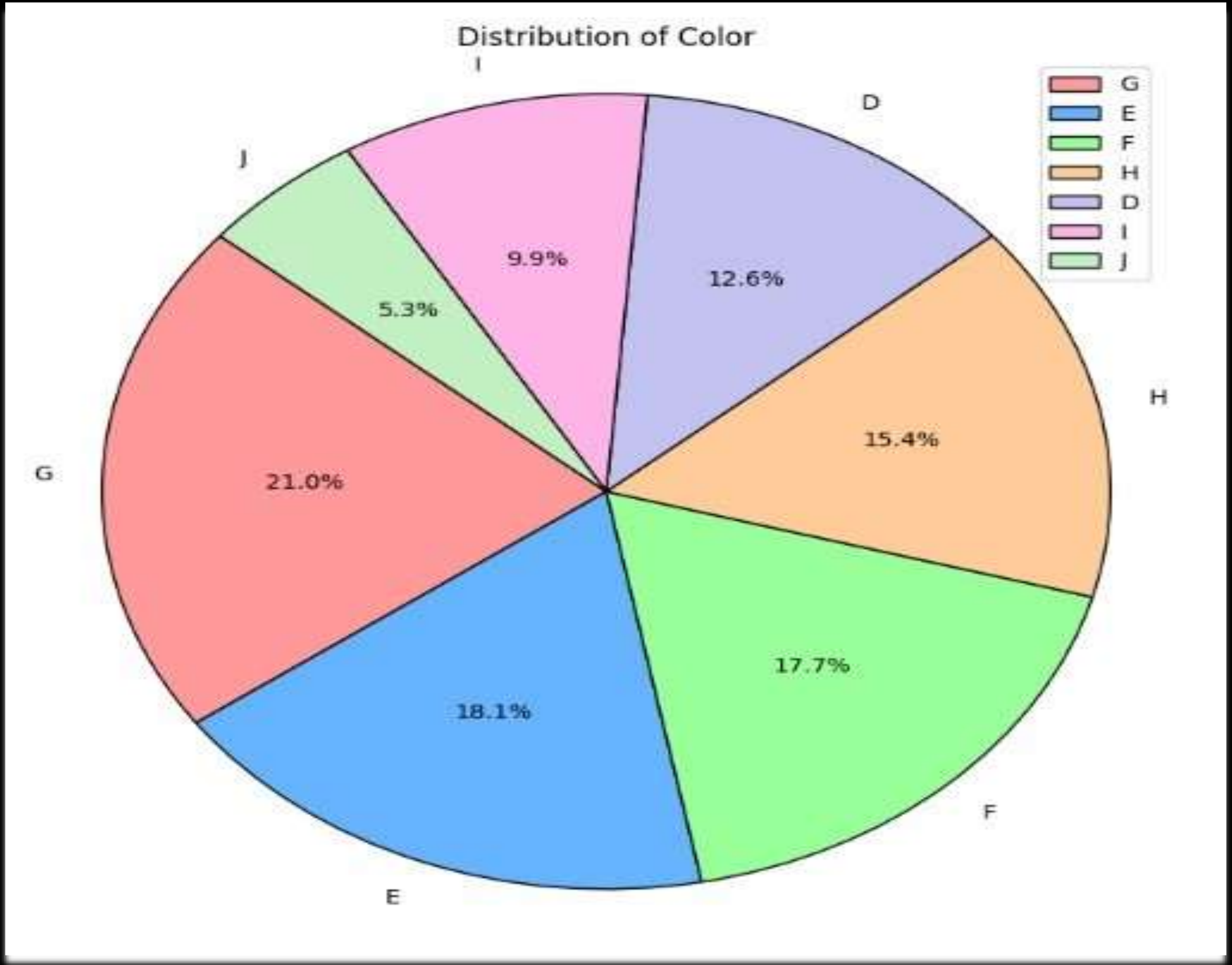
6.3.2 Visualize relationships between categorical features and target variable using bar plots



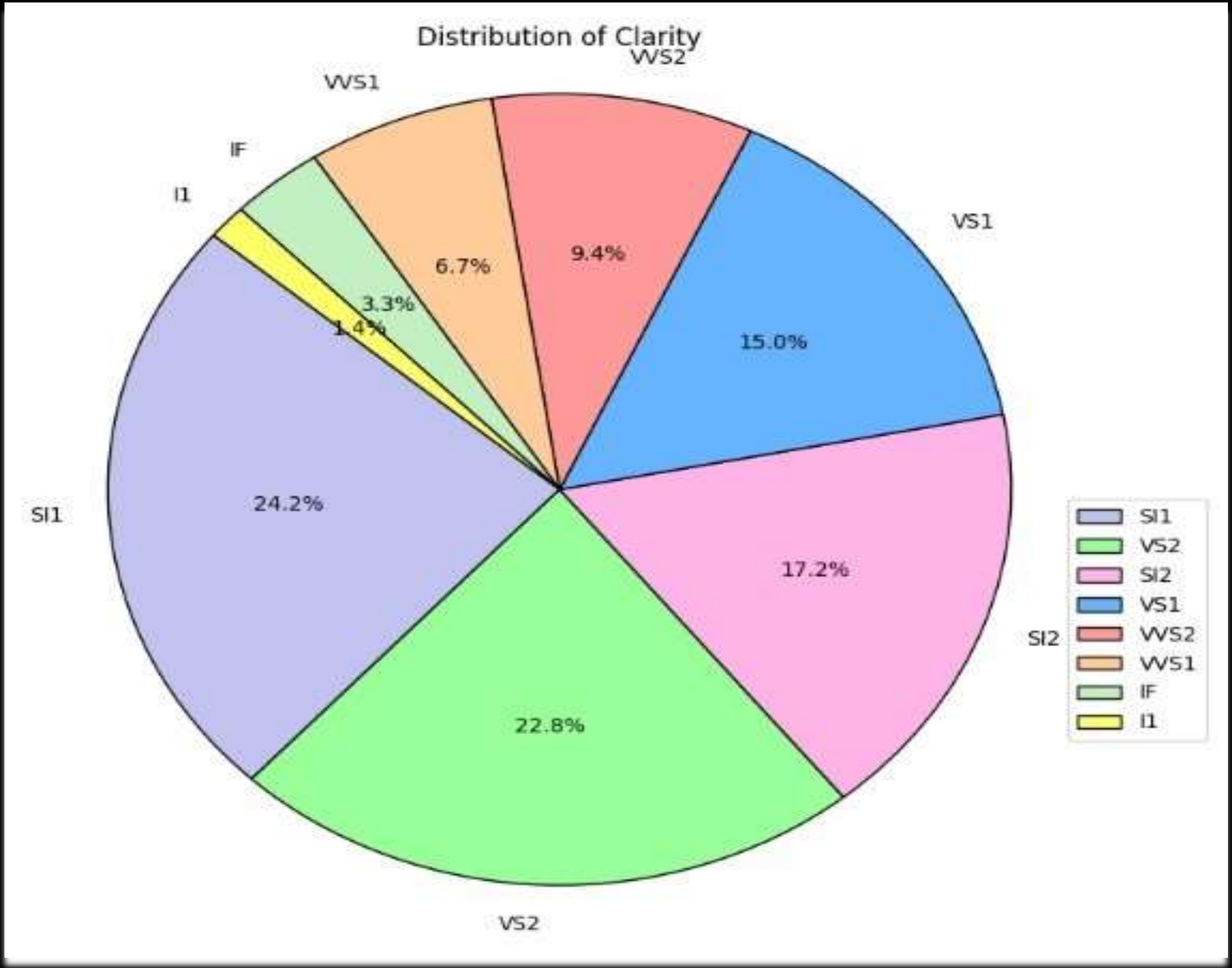
6.3.2 Visualize relationships between categorical features and target variable using pie charts



6.3.2 Visualize relationships between categorical features and target variable using pie charts



6.3.2 Visualize relationships between categorical features and target variable using pie charts



6.4 Descriptive Analytic Techniques - Statistics

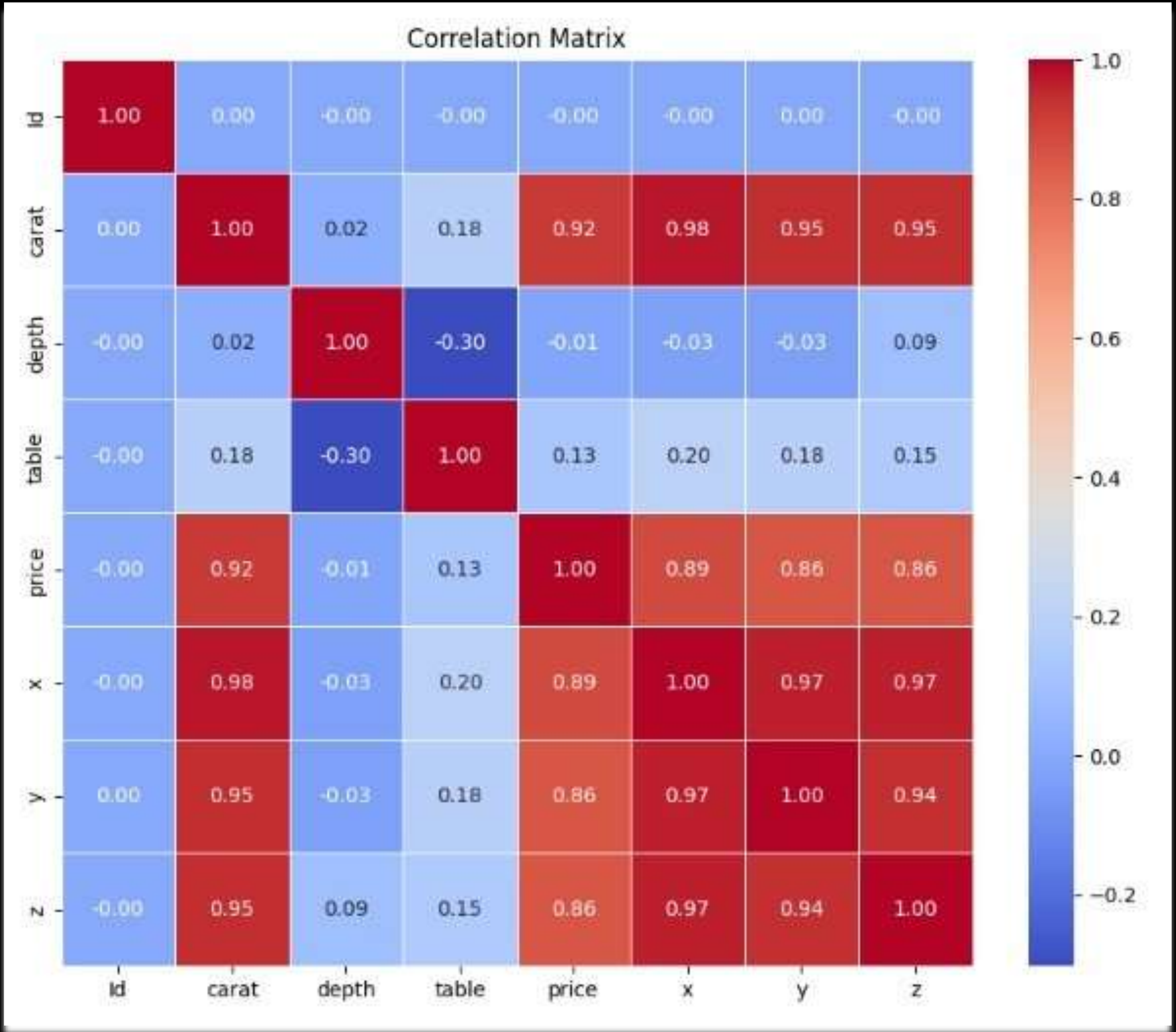
6.4.1 Measures of Position for 'x', 'y' and 'z' columns

In our analysis of the diamond's dataset, we applied various descriptive analytic techniques to summarize and understand the data. Here, we present the measures of position for the 'x', 'y', and 'z' columns, which represent the physical dimensions of the diamonds. These measures include the 25th percentile (Q1), the 75th percentile (Q3), the Interquartile Range (IQR), and the minimum and maximum values.

6.4.2 Outlier detections for 'x', 'y' and 'z' columns:

```
- Lower thresholds for 'x' column:
1.9649999999999999
- Upper thresholds for 'x' column: 9.285

- Lower thresholds for 'y' column:
1.9899999999999993
- Upper thresholds for 'y' column: 9.27

- Lower thresholds for 'z' column:
1.2150000000000003
```

6.5 Visualize correlation matrix



6.5 Visualize correlation matrix

Observations from the Correlation Matrix:

- Carat has a strong positive correlation with price, indicating that heavier diamonds tend to be more expensive.

- X, Y, and Z dimensions also show a positive correlation with price, as larger diamonds in physical dimensions are typically more valuable.

- Depth and Table have weaker correlations with price, suggesting that while they do

fect diamond value, their impact is le s
gnificant compared to carat weight an
ysical dimensions.

7- Prepare the Data for Machine Learning Algorithms

7.1 Preprocess the Data

7.1.1 Encoding Categorical Features Ordinally

Preprocess the Data

[29]
```python
# Encoding Categorical Features in the train data
# Ordinal encoding
cut_order = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
color_order = ['J', 'I', 'H', 'G', 'F', 'E', 'D']
clarity_order = ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF']

diamonds['cut_ordinal'] = diamonds['cut'].map(lambda x: cut_order.index(x)+1)

diamonds['color_ordinal'] = diamonds['color'].map(lambda x: color_order.index(x)+1)

diamonds['clarity_ordinal'] = diamonds['clarity'].map(lambda x: clarity_order.index(x)+1)
```

7.1.2 Removing Redundant Categorical Features

Drop Irrelevant Columns:

[30]
```python
# Remove columns that are not useful for prediction.
diamonds.drop(['Id'], axis=1, inplace=True)

diamonds = diamonds.drop(columns=['cut','color','clarity'])
```

[31]
```python
# Check the shape of the DataFrame
print("Shape of DataFrame:", diamonds.shape)
```

```
Shape of DataFrame: (43152, 18)
```

7- Prepare the Data for Machine Learning Algorithms

7.2 Data Cleaning

7.2.1 Handling Outliers and Zero Values by removing them.

[140]

```
# Data Cleaning
# Handling Zero Values
diamonds = diamonds[(diamonds != 0).all(axis=1)]
zero_values = (diamonds == 0).any()
zero_values
```

```
carat               False
depth               False
table               False
price               False
x                   False
y                   False
z                   False
volume              False
diameter            False
density             False
surface_area        False
depth_percentage    False
length_ratio_xy     False
length_ratio_xz     False
length_ratio_yz     False
cut_ordinal         False
color_ordinal       False
clarity_ordinal     False
dtype: bool
```

```
# Data Cleaning
# Handling Outliers
q1 = diamonds[['x','y', 'z']].quantile(0.25)
q3 = diamonds[['x','y', 'z']].quantile(0.75)
iqr = q3 - q1
max_val = q3 + 1.5 * iqr
min_val = q1 - 1.5 * iqr
outliers = ((diamonds[['x','y', 'z']] < min_val) | (diamonds[['x','y', 'z']] > max_val))
diamonds = diamonds[~outliers.any(axis=1)]
```

7- Prepare the Data for Machine Learning Algorithms

7.3 Feature Engineering

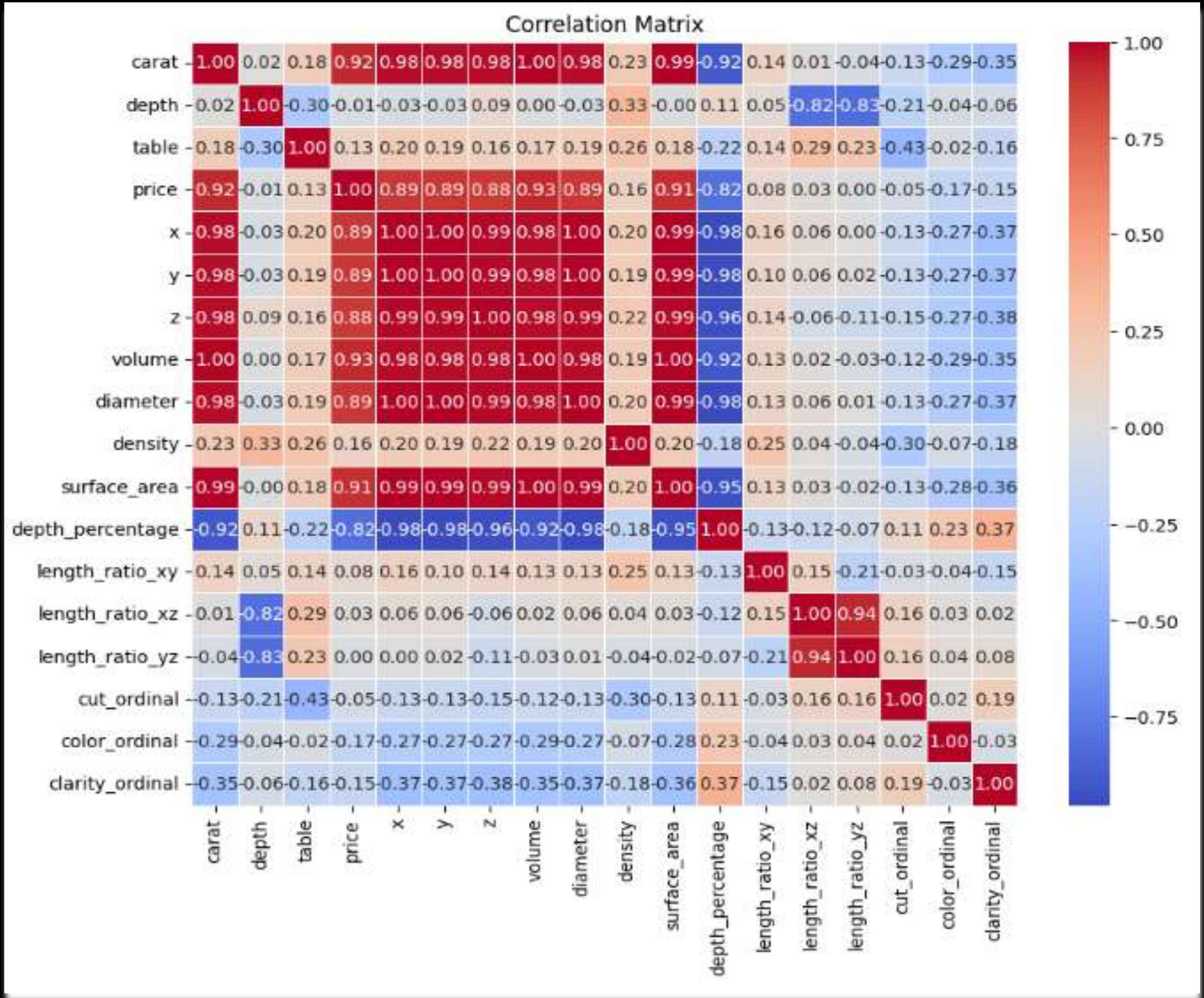
7.3.1 creating new features:

In data analytics, Feature Engineering step involves deriving new attributes from existing ones to improve predictive models. In this step, various geometric properties and ratios are calculated based on the dimensions of diamonds.

[28]

```
# Feature Engineering

# Calculate volume for training dataset
diamonds['volume'] = diamonds['x'] * diamonds['y'] * diamonds['z']

# Calculate diameter for training dataset
diamonds['diameter'] = (diamonds['x'] + diamonds['y']) / 2

# Calculate density for training dataset
diamonds['density'] = diamonds['carat'] / diamonds['volume']

# Calculate surface area for training dataset
diamonds['surface_area'] = 2 * (diamonds['x'] * diamonds['y'] + diamonds['x'] * diamonds['z'] + diamonds['y'] * diamonds['z'])

# Calculate depth percentage for training dataset
diamonds['depth_percentage'] = (diamonds['depth'] / ((diamonds['x'] + diamonds['y'] + diamonds['z']) / 3)) * 100

# Calculate length ratios for training dataset
diamonds['length_ratio_xy'] = diamonds['x'] / diamonds['y']
diamonds['length_ratio_xz'] = diamonds['x'] / diamonds['z']
diamonds['length_ratio_yz'] = diamonds['y'] / diamonds['z']
```

7- Prepare the Data for Machine Learning Algorithms



7- Prepare the Data for Machine Learning Algorithms

- Sorting Correlation Coefficients with Price (the target variable)

```python
# Sorting Correlation Coefficients with Price (the target variable)
corr_matrix["price"].sort_values(ascending=False)
```

```
price                1.000000
volume               0.925347
carat                0.923658
surface_area         0.911311
y                    0.888624
diameter             0.888075
x                    0.886923
z                    0.882126
density              0.156364
table                0.127722
length_ratio_xy      0.084795
length_ratio_xz      0.032445
length_ratio_yz      0.000292
depth               -0.014940
cut_ordinal         -0.054094
clarity_ordinal     -0.145080
color_ordinal       -0.169901
depth_percentage    -0.817047
Name: price, dtype: float64
```

7- Prepare the Data for Machine Learning Algorithms

7.2 Data Cleaning

7.2.2 Handling duplicates values after dropping ‘Id’ Column by removing duplicate values.

Handling duplicates values

[36]
```python
diamonds.duplicated().sum()
```
```
97
```

[37]
```python
# Removing Duplicate Rows
diamonds = diamonds.drop_duplicates(keep='first')

# Checking for Duplicates
diamonds[diamonds.duplicated].shape
```
```
(0, 18)
```

[38]
```python
# Checking for Duplicates
diamonds.duplicated().sum()
```
```
0
```

7- Prepare the Data for Machine Learning Algorithms

```
[42]  1 print(diamonds.info())

<class 'pandas.core.frame.DataFrame'>
Index: 43011 entries, 0 to 43151
Data columns (total 18 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   carat               43011 non-null  float64
 1   depth               43011 non-null  float64
 2   table               43011 non-null  float64
 3   price               43011 non-null  int64  
 4   x                   43011 non-null  float64
 5   y                   43011 non-null  float64
 6   z                   43011 non-null  float64
 7   volume              43011 non-null  float64
 8   diameter            43011 non-null  float64
 9   density             43011 non-null  float64
 10  surface_area        43011 non-null  float64
 11  depth_percentage    43011 non-null  float64
 12  length_ratio_xy     43011 non-null  float64
 13  length_ratio_xz     43011 non-null  float64
 14  length_ratio_yz     43011 non-null  float64
 15  cut_ordinal         43011 non-null  int64  
 16  color_ordinal       43011 non-null  int64  
 17  clarity_ordinal     43011 non-null  int64  
dtypes: float64(14), int64(4)
memory usage: 6.2 MB
None
```

8- Model Selection and Training

1- Splitting Data into Features and Target Variable

2- Standardizing Features Using Robust Scaler

Split the Training Data

```
# Split data into features and target variable
X = diamonds.drop('price', axis=1)
y = diamonds['price']

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

Standardizing Features

```
# Standardizing Features using Robust Scaler
scaler = RobustScaler()

# Scaling the training data
X_train = scaler.fit_transform(X_train)

# Scaling the test data
X_val = scaler.transform(X_val)
```

3- Building and Evaluating Model Pipelines

```python
# Evaluate the performance of each pipeline
for idx, pipe in enumerate(pipelines):
    y_pred_pipe = pipe.predict(X_val)
    rmse_pipe = np.sqrt(mean_squared_error(y_val, y_pred_pipe))
    print(f"RMSE for {pipe_dict[idx]}: {rmse_pipe}")
```

```
RMSE for LinearRegression: 1126.923082633785
RMSE for DecisionTree: 784.5985391481325
RMSE for RandomForest: 571.392365056412
RMSE for KNeighbors: 772.3398205005454
RMSE for XGBRegressor: 564.4383292373294
RMSE for PolynomialRegression: 1524.5093360315373
RMSE for Ridge: 1132.986503820047
RMSE for Lasso: 1155.359079954855
RMSE for ElasticNet: 1491.002061422464
RMSE for GradientBoostingRegressor: 630.468800852954
```

4- Selecting the Best Model

Choose the model with the best RMSE

```python
# Dictionary to store RMSE values for each model
rmse_values = {
    "Polynomial Regression": rmse_poly,
    "Ridge Regression": rmse_ridge,
    "Lasso Regression": rmse_lasso,
    "ElasticNet Regression": rmse_elastic_net,
    "KNNR":rmse_knnr,
    "Decision Tree Regression":rmse_dtr,
    "Linear Regression":rmse_lr_Model,
    "XGBRegressor":rmse_xgb_Model,
    "GradientBoostingRegressor":rmse_Gradient_Model,
    "RandomForestRegressor":rmse_rf_Model
}

# Find the model with the lowest RMSE
best_model = min(rmse_values, key=rmse_values.get)

# Print the best model and its RMSE
print("Best Model:", best_model)
print("RMSE:", rmse_values[best_model])
```

```
Best Model: XGBRegressor
RMSE: 559.921144573028
```

9- Fine-Tune Your Model

Hyperparameter Tuning for XGBoost Regressor Using Grid Search

```python
# Hyperparameter Tuning for XGBoost Regressor Using Grid Search
# Define a custom scoring function for RMSE
def rmse(y_true, y_pred):
    return sqrt(mean_squared_error(y_true, y_pred))

# Make a scorer using the custom RMSE function
scorer = make_scorer(rmse, greater_is_better=False)

# Initialize an XGBoost regressor model
xgb_model = XGBRegressor()

# Define the grid of hyperparameters to search over
param_grid = {
    'n_estimators': [500],
    'learning_rate': [0.05],
    'max_depth': [5],
    'max_delta_step' : [0],
    'lambda' : [0],
    'alpha' : [1]
}

# Initialize GridSearchCV with the XGBoost regressor model, hyperparameter grid, and custom scoring function
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, n_jobs=-1, scoring=scorer)

grid_search.fit(X_train, y_train)

# Retrieve the best hyperparameters and the corresponding best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'alpha': 1, 'lambda': 0, 'learning_rate': 0.05, 'max_delta_step': 0, 'max_depth': 5, 'n_estimat
Best Score: -514.6121390834212
```

10- Evaluate Your System on the Test Set

```python
xgb_model = XGBRegressor(**best_params)
xgb_model.fit(X, y)

# Predict on the test data using the best model (XGBRegressor)
y_test_pred = xgb_model.predict(test)
```

```python
# Format the predictions according to the submission requirements
submission_df = pd.DataFrame({'price': y_test_pred})
```

```python
submission_df['Id'] = test['Id'] if 'Id' in test.columns else range(1, len(submission_df) + 1)
```

```python
# Reorder the columns to have 'Id' first
submission_df = submission_df[['Id', 'price']]
```

```python
# Save the predictions to a CSV file
submission_df.to_csv('submission.csv', index=False)
```

```python
best_params = {
    'alpha': 1,
    'lambda': 0,
    'learning_rate': 0.05,
    'max_delta_step': 0,
    'max_depth': 5,
    'n_estimators': 500
}
```

Team Names:

- Mays Moh’d Al-Fasfous
- Hesham Saad Alsaadi
- Sara Hasan
- Fahed Shadid

Thank You