CHAPTER 1

INTRODUCTION

The diamond market plays a vital role in the global economy, encompassing multiple sectors, including mining, processing, retail, and investment. Diamonds are valued for their rarity, durability, and symbolic significance, making them a cornerstone of luxury markets. However, pricing diamonds is a complex task influenced by a multitude of factors. The "4 Cs" of diamonds - carat, cut, color, and clarity - are fundamental to valuation, but external factors such as market demand, geopolitical conditions, and currency fluctuations also play a significant role. Consequently, determining diamond prices with precision remains a challenge for stakeholders, ranging from miners and jewelers to consumers and investors.

Traditionally, diamond pricing has relied on manual evaluations by experts or simple statistical models, such as linear regression, which are often limited in their ability to capture the nuances of the data. These methods may work well for broad trends but struggle to account for nonlinear interactions between variables or adapt to dynamic market conditions. Inaccurate price forecasts can lead to inefficiencies, such as overpricing that deters customers or underpricing that reduces profit margins. As the diamond market continues to grow and evolve, there is an increasing need for more advanced and reliable methods to predict prices and analyze market trends.

Machine learning (ML), a branch of artificial intelligence, offers a promising solution to these challenges. ML techniques are particularly adept at processing and analyzing large datasets, identifying hidden patterns, and making precise predictions. Unlike traditional models, ML algorithms can accommodate complex relationships among variables and adapt to changing patterns in real-time. With the growing availability of detailed datasets on diamond characteristics and pricing, applying ML to forecast diamond market trends has become both feasible and highly advantageous.

This research focuses on leveraging ML to improve the accuracy of diamond price predictions and uncover insights into market trends. By analyzing a dataset that includes critical attributes such as carat weight, cut, color, clarity, and historical price data, this study applies a range of ML algorithms, including Linear Regression, Decision Trees, Random Forest, and Gradient Boosting. Advanced feature engineering techniques are employed to identify the most influential factors in pricing, while cross-validation methods are used to ensure model robustness.

In addition to improving predictive accuracy, this research aims to bridge the gap between traditional valuation methods and modern computational techniques. The findings will provide stakeholders with a transparent and data-driven framework for decision-making, enhancing efficiency and reducing uncertainty in the diamond market.

The broader implications of this study extend beyond the diamond industry. It serves as a case study for how ML can transform traditional industries by enabling more accurate forecasting and facilitating data-driven strategies. In an era where data is becoming increasingly central to decision-making processes, this work highlights the value of integrating ML technologies into established markets to foster innovation, transparency, and sustainability.

Through this research, we aim to not only improve the state of diamond price forecasting but also contribute to the growing body of knowledge on the application of machine learning in complex, real-world domains. The insights generated here have the potential to shape the future of the diamond industry and inspire similar advancements in other luxury markets.

1.1. OBJECTIVES :

The primary goal of this research is to develop and implement a machine learning-based framework for accurately forecasting diamond market trends and prices. The specific objectives of the study include :

1. **Develop a Predictive Model for Diamond Prices**
    - Utilize advanced machine learning algorithms to create a robust model capable of predicting diamond prices based on key attributes such as carat, cut, color, and clarity.

2. **Identify Key Factors Influencing Diamond Pricing**
   - Conduct feature selection and importance analysis to determine the most influential attributes and external factors affecting diamond prices.

3. **Enhance Predictive Accuracy**
   - Compare the performance of various machine learning models, including Linear Regression, Decision Trees, Random Forest, and Gradient Boosting, to identify the best-performing algorithm for price forecasting

4. **Analyze Market Trends**
   - Leverage historical price data to uncover patterns and trends in the diamond market, providing insights into demand fluctuations and seasonal effects.

5. **Enable Data-Driven Decision Making**
   - Develop a framework that stakeholders in the diamond industry - such as retailers, investors, and consumers -ncan use to make informed pricing and purchasing decisions.

6. **Validate Model Robustness**
   - Employ cross-validation techniques to ensure the reliability and generalizability of the predictive model across diverse datasets and market scenarios.

7. **Bridge the Gap Between Traditional and Modern Approaches**
   - Integrate machine learning into traditional diamond valuation practices, demonstrating how advanced computational tools can enhance the accuracy and efficiency of price predictions.

8. **Support Sustainable Practices in the Diamond Industry**
   - Provide insights that can help optimize supply chain operations, reduce inefficiencies, and promote fair pricing, contributing to a more sustainable diamond market.

9. **Promote Innovation in the Luxury Goods Market**
    - Showcase the potential of machine learning as a transformative tool in luxury markets, setting a precedent for similar applications in other high-value industries.

1.1. PROBLEM STATEMENT :

The diamond market's complexity, driven by multiple factors like the "4 Cs," market demand, and economic conditions, makes accurate price forecasting challenging. Traditional valuation methods often fail to capture the nonlinear relationships between these factors, leading to inefficiencies, pricing inaccuracies, and reduced transparency. This study aims to address these challenges by leveraging machine learning to develop a robust, data-driven framework for precise diamond price prediction and market trend analysis, empowering stakeholders with actionable insights and informed decision-making.

CHAPTER 2

LITERATURE SURVEY

The application of machine learning (ML) in price prediction has gained significant attention in recent years, especially in industries dealing with complex pricing structures such as real estate, automotive markets, and luxury goods. The diamond market, however, remains an underexplored area for ML applications despite its dependence on multiple interacting factors like carat, cut, color, and clarity, collectively known as the "4 Cs." This literature survey reviews existing research in related domains to establish the foundation for leveraging ML in forecasting diamond market trends.

1. Traditional Approaches to Diamond Valuation

Historically, diamond pricing has relied on manual assessments and linear statistical models. Rapaport pricing tables, widely used in the diamond industry, provide benchmark prices based on the "4 Cs." However, such methods are static, subjective, and fail to account for dynamic market conditions or interdependencies among variables. Studies have highlighted the limitations of these approaches in addressing nonlinear relationships between diamond attributes and pricing (e.g., high carat weights amplifying the effect of clarity).

2. Machine Learning in Price Prediction

Machine learning has been extensively applied in other industries for price prediction, demonstrating its potential to address challenges in the diamond market. For instance:

- **Real Estate** : Studies have used algorithms like Random Forest, Support Vector Machines (SVM), and Gradient Boosting to predict property prices based on multiple features (location, size, amenities). These techniques are highly transferable to diamond pricing due to the multivariable nature of both markets.
- **Luxury Goods** : Research on predicting prices of luxury handbags and watches has utilized ML models to uncover the impact of brand, condition, and rarity, drawing parallels with diamonds' unique valuation factors.

3. Application of ML in Diamond Price Prediction

Few studies have explored ML directly for diamond valuation:

- **Regression Models**: Linear regression has been applied to predict diamond prices, showing the importance of features like carat weight and cut quality. However, its simplicity limits its accuracy when capturing complex interactions.
- **Ensemble Models**: Research using Decision Trees and Random Forest has shown improved accuracy by accommodating nonlinear relationships.
- **Deep Learning**: Neural networks have been tested in limited contexts for predicting prices based on diamond datasets, though computational complexity and overfitting remain challenges.

4. Feature Engineering and Selection

Research emphasizes the significance of feature engineering in ML models for price prediction. Studies suggest that preprocessing techniques like standardization, feature scaling, and removal of multicollinearity enhance model performance. For diamonds, additional features such as fluorescence, symmetry, and polish are identified as important variables.

5. Challenges in Applying ML to Diamonds

While ML offers promise, challenges include:

- **Data quality and availability**: The diamond market lacks large, publicly available datasets compared to industries like real estate.
- **Interpretability**: Stakeholders often require models that not only predict accurately but also provide explanations for price variations.
- **Generalizability**: Models trained on specific datasets may fail to adapt to varying market conditions or regional pricing norms.

CHAPTER 3

SYSTEM DESIGN

3.1. ARCHITECTURE :

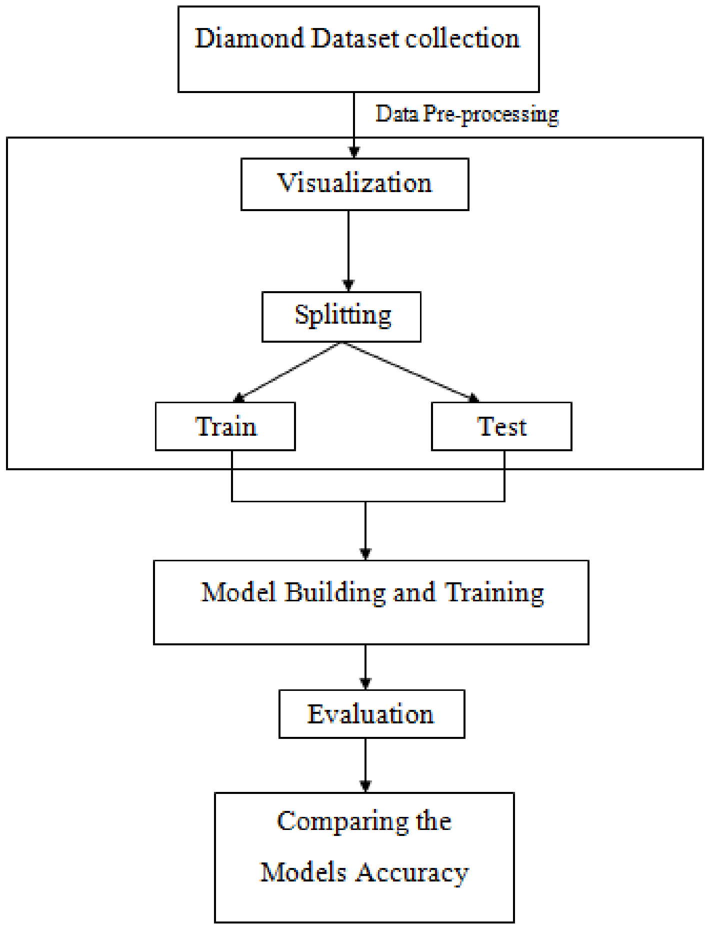

Diamond Dataset collection
Data Pre-processing
Visualization
Splitting
Train
Test
Model Building and Training
Evaluation
Comparing the Models Accuracy

Fig 3.1. Architecture of the proposed model

3.2. DATAFLOW DIAGRAM :

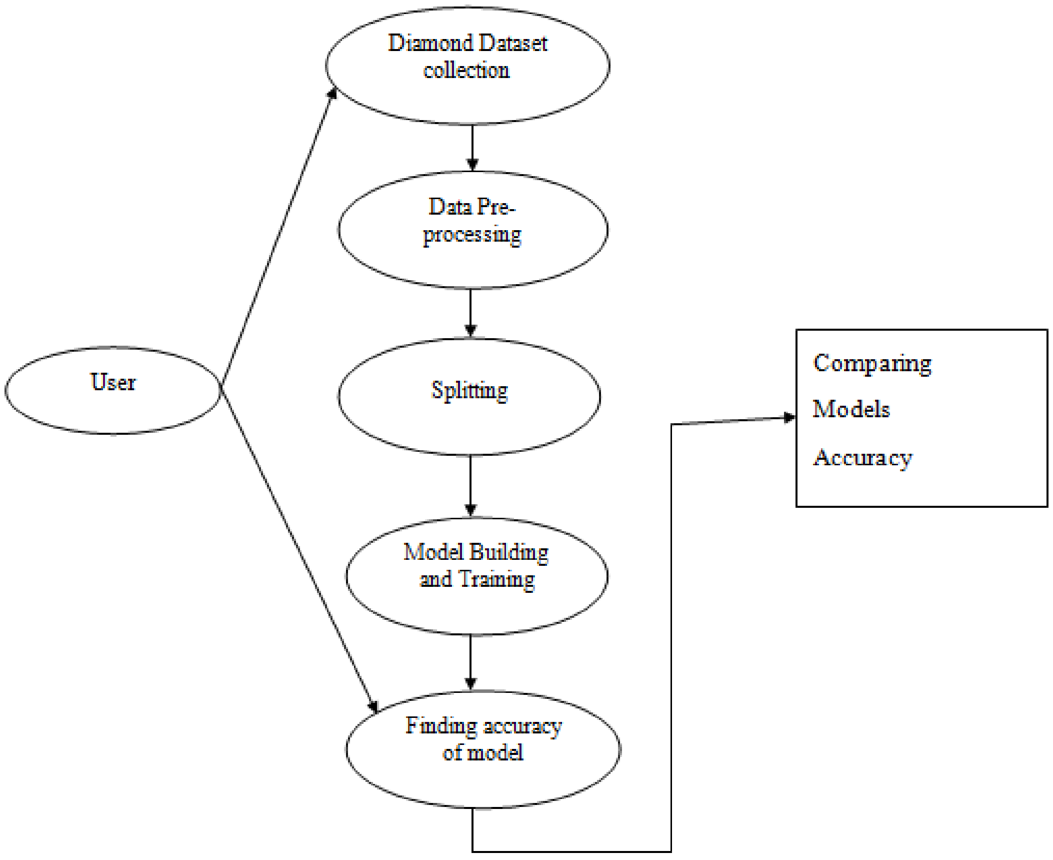

Diamond Dataset collection
Data Pre-processing
Splitting
Model Building and Training
Finding accuracy of model
User
Comparing
Models
Accuracy

Fig 3.2. Dataflow diagram of proposed system

3.3. USE CASE DIAGRAM :

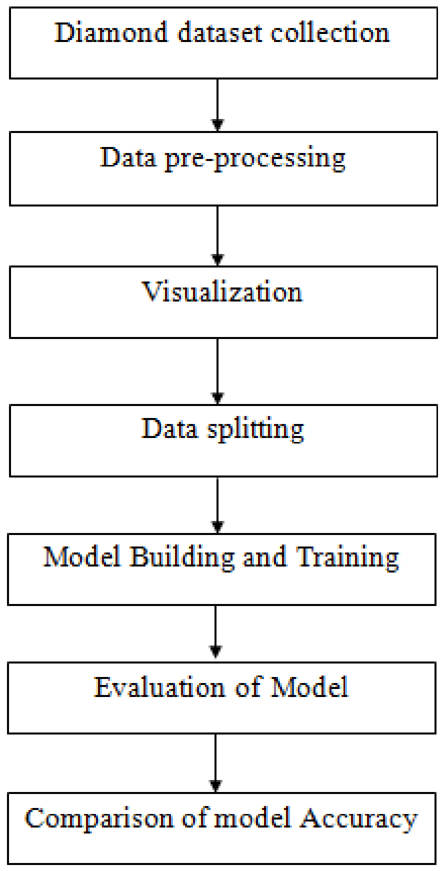

Diamond dataset collection
Data pre-processing
Visualization
Data splitting
Model Building and Training
Evaluation of Model
Comparison of model Accuracy

Fig 3.3. use case diagram of proposed system

3.4. SEQUENCE DIAGRAM :

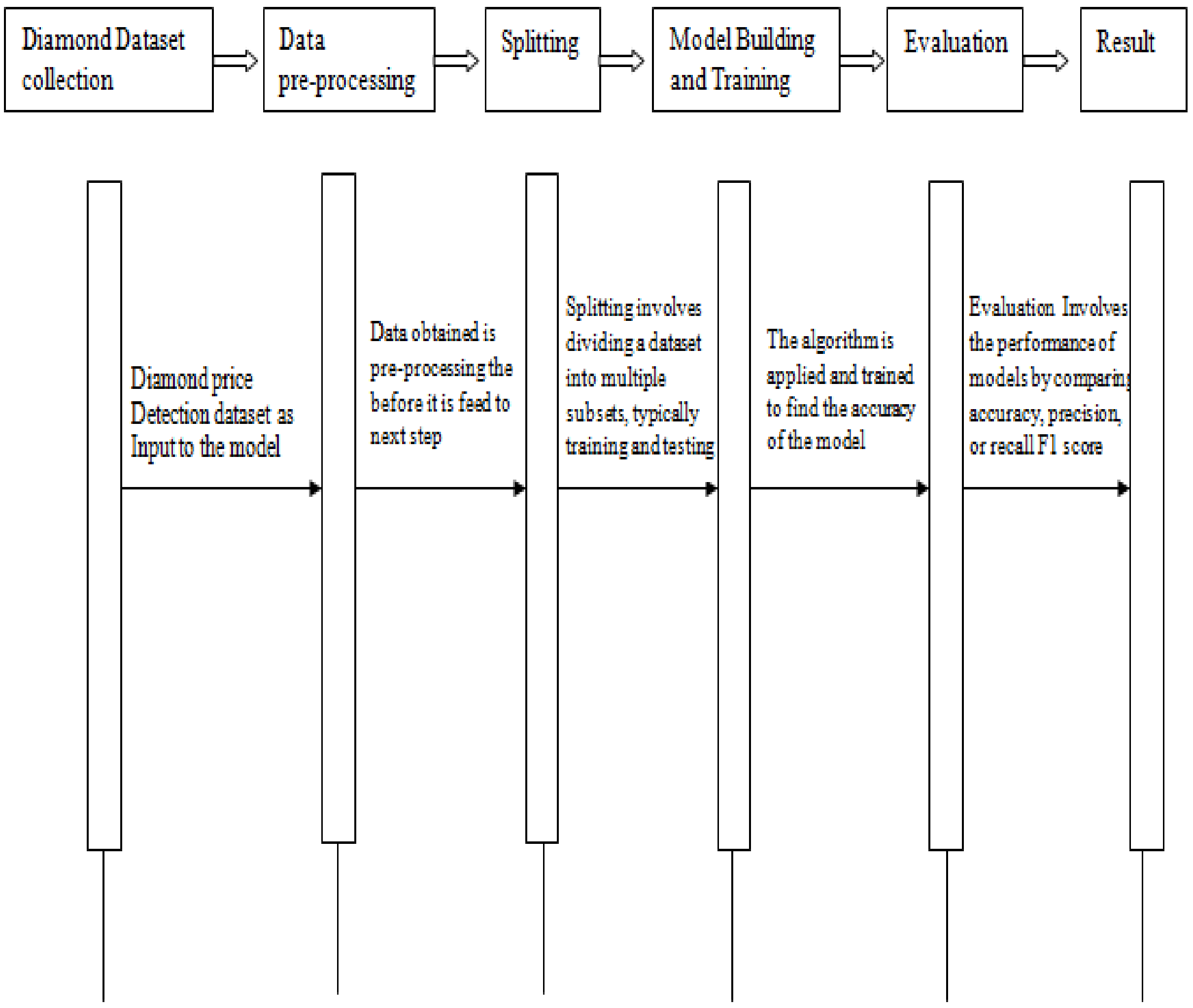

Diamond Dataset collection
Data pre-processing
Splitting
Model Building and Training
Evaluation
Result
Diamond price Detection dataset as Input to the model
Data obtained is pre-processing the before it is feed to next step
Splitting involves dividing a dataset into multiple subsets, typically training and testing
The algorithm is applied and trained to find the accuracy of the model
Evaluation Involves the performance of models by comparing accuracy, precision, or recall F1 score

Fig 3.4. Sequence diagram of proposed system

CHAPTER 4

METHODOLOGY

The proposed methodology comprises of following phases:

4.1. Data Collection :

The dataset is downloaded from Kaggle. The “Diamond Price Prediction.csv file” contains diamond quality metrics. The dataset consists of 53941 rows and 10 columns. The 10 columns are divided into Carat (Weight of Daimond), Cut (Quality), Color, Clarity, Depth, Table, Price (in US dollars), X(length), Y(width) and Z(Depth).

4.2. Data Preprocessing :

Steps involved in Data Preprocessing

- Data cleaning
- Identifying and removing outliers
- Encoding categorical variables

Here's a breakdown of the data processing:

---

1. Importing Data

```
df = pd.read_csv('DPP.csv')
```

- Reads the dataset from a CSV file into a pandas DataFrame (df).

---

2. Understanding the Dataset

```
print(df.head())
```

- Displays the first few rows of the dataset to understand its structure and variables.

---

3. Encoding Categorical Variables

df_encoded = pd.get_dummies(df, columns=['Cut(Quality)', 'Color', 'Clarity'], drop_first=True)

- **Purpose**: Converts categorical variables (Cut(Quality), Color, Clarity) into numerical representations using one-hot encoding.
- **drop_first=True**: Avoids multicollinearity by dropping the first category of each variable.
- **Result**: Adds new binary columns for each category in these variables.

---

4. Splitting Data into Features and Target

X = df_encoded.drop(columns=['Price(in US dollars)'])

y = df_encoded['Price(in US dollars)']

- **X**: Contains independent variables (features) after removing the target column (Price(in US dollars)).
- **y**: Contains the dependent variable (target), which is the price.

---

5. Splitting 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)

- Splits the dataset into training (80%) and testing (20%) sets.
- Ensures the model is trained and tested on different subsets to evaluate performance.

---

6. Normalizing the Data (Optional but Recommended)

Normalization is not explicitly performed here. However, it is good practice to scale numerical features like Depth, Table, X(length), Y(width), and Z(Depth) to improve model performance, especially for distance-based algorithms.

---

7. Handling New Data for Prediction

```
new_data_encoded = pd.get_dummies(new_data, columns=["Cut(Quality)", "Color", "Clarity"])
new_data_encoded = new_data_encoded.reindex(columns=X_train.columns, fill_value=0)
```

- **Purpose**: Prepares new data for prediction by ensuring it matches the feature set used during training.
- **Steps**:
  - ➢ Apply one-hot encoding to the new data.
  - ➢ Reindex columns to match X_train (training features), filling missing columns with zeros.

Key Notes

1. **Categorical Encoding**: Converts non-numeric variables to numeric forms for machine learning models, ensuring compatibility.
2. **Data Splitting**: Separates the dataset into training and testing sets for unbiased model evaluation.
3. **Feature Engineering**: Maintains consistency between training features and new data features during prediction.

These processing steps ensure that the data is clean, compatible with machine learning algorithms, and aligned with the requirements of both training and prediction stages.

4.3. Visualization :

Visualization is a powerful technique used to represent data visually, allowing patterns, trends, and relationships to be easily understood and interpreted. There are various types of visualizations, each suited to different types of data and analytical tasks.
Common types of visualizations include:

- Bar charts and histograms for displaying frequency distributions or comparing categories. Like for bar chart Hemoglobin vs Result and for histogram for gender or hemoglobin etc.
- Line charts for showing trends or changes over time.
- Scatter plots for visualizing relationships between two continuous variables.

4.4. Data Splitting :

When a machine learning model fits its training data too well and cannot reliably fit fresh data, it is said to be overfit. In order to avoid overfitting, data splitting is widely utilized in machine learning. A machine learning model often divides the initial data into two three. The three sets that are usually utilized are the training set, the validation set, and the testing set. The piece of data used to train the model is known as the training set. In order to improve any of its parameters, the model must observe and learn from the training set. The validation set is a data collection of examples used to alter the settings for the learning process. The objective of this data set is to rate the model's accuracy, which can aid in model selection. The data set that is tested in the final model and contrasted with the earlier data sets is known as the testing set. The testing set serves as an assessment of the chosen algorithm and mode.

The dataset into consists of the 10 parameters of the data [Carat (Weight of Daimond), Cut (Quality), Color, Clarity, Depth, Table, Price (in US dollars), X(length), Y(width) and Z(Depth)]. In this study, the data were split into two parts: train and test with a ratio 80:20

4.5. Feature Selection :

Finding the optimum collection of features that can be utilized to build practical models is the goal of employing feature selection strategies in machine learning. It includes determining each input variable's link to the target variable using a set of evaluation criteria before choosing the input variables with the strongest relationships. Feature selection helps to increase decision accuracy, and shorten the time needed to complete the ML training process.

4.6. Model used :

The sklearn.linear_model module in scikit-learn provides a set of classes for implementing linear models in machine learning. These models are used for both regression and classification tasks, where the target variable is expected to be a linear combination of the input features.

Key Classes in sklearn.linear_model:

- **Linear Regression :** This class implements ordinary least squares linear regression, which is the most basic form of linear regression.
- **Ridge :** This class implements ridge regression, which adds L2 regularization to the linear regression model. L2 regularization helps to prevent overfitting by penalizing large coefficients.
- **Lasso :** This class implements lasso regression, which adds L1 regularization to the linear regression model. L1 regularization encourages sparsity in the coefficients, meaning it tends to select a subset of features that are most important for the prediction.
- **Elastic Net :** This class implements elastic net regression, which combines L1 and L2 regularization.
- **Logistic Regression :** This class implements logistic regression, which is a classification algorithm that models the probability of a binary outcome as a linear combination of the input features.
- **SGD Regressor :** This class implements stochastic gradient descent (SGD) for linear regression. SGD is an iterative optimization algorithm that is often used for large-scale datasets.
- **SGD Classifier :** This class implements SGD for classification tasks.

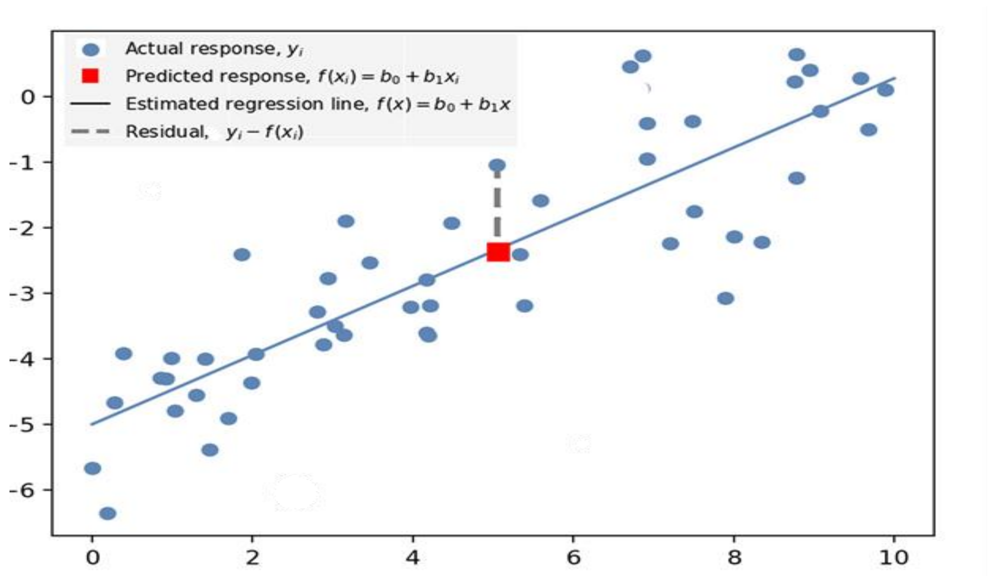


Fig 4.6.1. scikit-learn's linear regression method

ALGORITHM

Linear Regression :

1. Mean Absolute Error (MAE) :

It is commonly used because it is easy to interpret and directly measures the average magnitude of errors.

$$\mathbf{MAE} = \frac{1}{N}\sum_{i=1}^{n} |y^i - \hat{y}^i|$$

Where:

- N : The number of data points
- y^i : The actual (observed) value for the i th data point
- y^i : The predicted value for the i th data point
- $|y^i - \hat{y}^i|$: The absolute error for each data point

2. Mean Squared Error (MSE) :

The most common cost function used in linear regression is Mean Squared Error (MSE), which calculates the squared difference between the predicted and actual values.

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{n}(\mathbf{y}^{\mathbf{i}} - \hat{\mathbf{y}}^{\mathbf{i}})^2$$

Where:

- N = number of training examples
- y^i = actual value for the i-th training example
- $\hat{y}^i$ = predicted value for the i-th training example

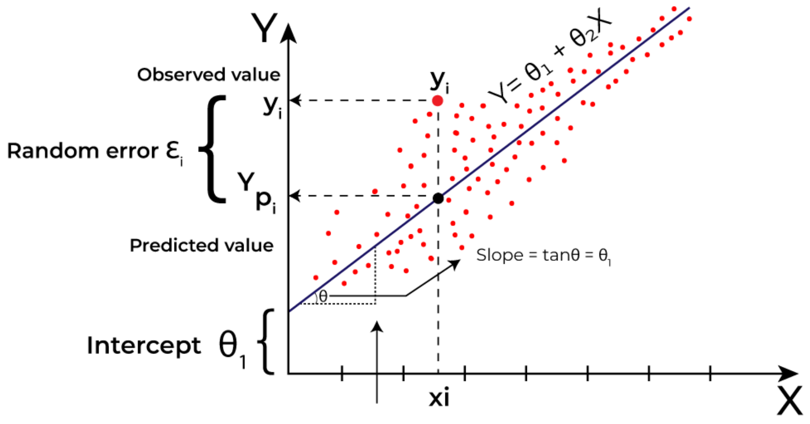

Y
Observed value
y_i
y_i
Random error ε_i
Y_{p_i}
Predicted value
$Y = \theta_1 + \theta_2 X$
Slope = tanθ = θ_1
θ
Intercept θ_1
xi
X

Fig 4.6.2. Linear Regression Algorithm

Random Forest Regressor :

Random Forest is an ensemble method based on the concept of **bagging** (Bootstrap Aggregating), where multiple decision trees are trained on different random subsets of the data, and the final prediction is made by aggregating the individual predictions of all trees. For regression tasks, this aggregation is typically done by averaging the predictions from all the trees.

1. **Initialize the Random Forest Regressor**

- **RandomForestRegressor**: A machine learning model that combines multiple decision trees to make predictions, improving accuracy and robustness.
- **random_state**: Ensures reproducibility by fixing the randomness.
- **n_estimators**: The number of trees in the forest.

2. **Train the model**

- **`fit()`**: Trains the model using the training dataset, `X_train` (features) and `y_train` (target values).

3. **Making Predictions**

- **`predict()`**: Uses the trained model to predict target values (`y_pred_rf`) for unseen data (`X_test`).

4. **Calculate Performance Metrics**

- Measures the average absolute difference between predicted and actual values.

$$\mathbf{MAE} = \frac{1}{N}\sum_{i=1}^{n} |y^{i} - \hat{y}^{i}|$$

Where:

- y^i : Actual values.
- $\hat{y}^i$: Predicted values.
- n : Number of samples.

5. **Mean Squared Error (MSE):**

   - Measures the average squared difference between predicted and actual values. It gives higher weight to larger errors.

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{n}(\mathrm{y}^{\mathrm{i}} - \hat{\mathrm{y}}^{\mathrm{i}})^2$$

6. **Root Mean Squared Error (RMSE):**

   - The square root of MSE, giving an error metric in the same unit as the target variable.

$$\mathbf{RMSE} = \sqrt{\mathbf{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(y^{i} - \hat{\mathrm{y}}^{i})^2}$$

7. **R² Score:**

   - Measures the proportion of variance in the target variable explained by the model.

$$\mathbf{R^2} = 1 - \frac{\sum_{i=1}^{n}(y^{i} - \hat{\mathrm{y}}^{i})^2}{\sum_{i=1}^{n}(y^{i} - \bar{\mathrm{y}})^2}$$

Where:

- y¯ : Mean of the actual target values.
- A value of $R^2=1$ indicates perfect predictions, while $R^2=0$ means the model explains no variance.

8. Print Metrics

```
python
Copy code
print(f"Mean Absolute Error (MAE): {mae_rf:.2f}")
print(f"Mean Squared Error (MSE): {mse_rf:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_rf:.2f}")
print(f"R² Score: {r2_rf:.2f}")
```

- Outputs the calculated performance metrics for easy interpretation of the model's accuracy and error characteristics.

Decision Tree Regressor :

A **decision tree** is a tree-like structure used to make decisions or predictions based on data. In the context of regression, a decision tree splits the data at each node based on a feature, and at the leaves, it provides a predicted value (the mean of the target variable in that leaf).

1. **Initialize the Decision Tree Regressor**

- This is the machine learning algorithm used for regression tasks. A decision tree splits the dataset into subsets based on feature values, forming a tree-like structure.
- random_state = 42: This sets the seed for the random number generator, ensuring that the results can be reproduced. It's useful for consistent results when splitting data or shuffling.

2. **Training the Model:**

- .fit(): This method trains the model on the training data.
- X_train: Features of the training dataset (independent variables).
- y_train: Target values of the training dataset (dependent variable).
- The model learns patterns in the data (how X_train influences y_train) during this phase.

3. **Making Predictions:**

- predict(): After training the model, it is used to make predictions on the X_test data, which is unseen by the model during training.

- X_test: Features of the test dataset, used to evaluate how well the model generalizes to new dat
- y_pred_dt: The predicted target values based on the X_test data.

4. Evaluating Model Performance:

- These performance metrics evaluate how well the model's predictions match the actual values (y_test).

a. Mean Absolute Error (MAE)

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{n} |y_{\mathrm{true},i} - y_{\mathrm{predict},i}|$$

Where :

- y_test: The true values of the target variable.
- y_pred_dt: The predicted values from the model.

b. Mean Squared Error (MSE)

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{n} (y_{\mathrm{true},i} - y_{\mathrm{predict},i})^2$$

c. Root Mean Squared Error (RMSE)

$$\sqrt{\mathbf{RMSE}} = \sqrt{\mathbf{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{n} |y_{\mathrm{true},i} - y_{\mathrm{predict},i}|}$$

d. R^2 Score (Coefficient of Determination):

$$\boldsymbol{R^2} = \mathbf{1} - \frac{\sum_{i=1}^{n}(\mathbf{y}_{\mathbf{true,i}} - \mathbf{y}_{\mathbf{predict,i}})^2}{\sum_{i=1}^{n}(\mathbf{y}_{\mathbf{true,i}} - \bar{\mathbf{y}}_{\mathbf{true}})^2}$$

4.7. Model Evaluation

Accuracy:

This is the proportion of correct predictions out of all the predictions made by the model. It is a commonly used metric for binary classification problems like Diamond Price prediction. However, accuracy can be misleading if the data is imbalanced (i.e., one class is much more prevalent than the other), as a model that always predicts the majority class can have a high accuracy but not be useful. Mathematically, accuracy is calculated as the ration of the number of correctly predicted instances to the total number of instances:

$$Accuracy = \frac{Number\ of\ Correct\ Prediction}{Total\ Number\ of\ Prediction} \times 100\%$$

Precision:

- Precision, also known as positive predictive value, measures the proportion of true positive predictions (correctly predicted positive instances) out of all instances predicted as positive by the model.
- It focuses on the accuracy of positive predictions and answers the question: "Of all instances predicted as positive, how many are actually positive?"
- Mathematically, precision is calculated as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall :

- Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions (correctly predicted positive instances) out of all actual positive instances in the dataset.
- It focuses on the model's ability to capture all positive instances and answers the question: "Of all actual positive instances, how many did the model correctly predict as positive?"
- Mathematically, recall is calculated as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

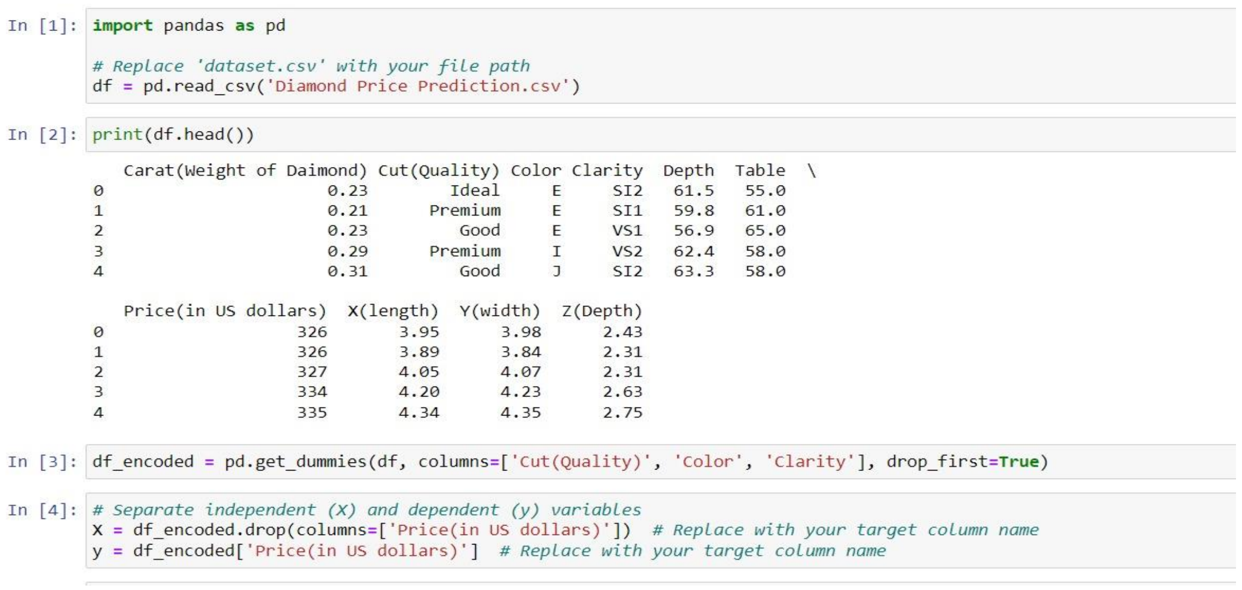
F1-score:

This is a harmonic mean of precision and recall, and can be useful for imbalanced datasets where both precision and recall need to be considered. True Positives + False Positives True Positives True Positives + False Negatives

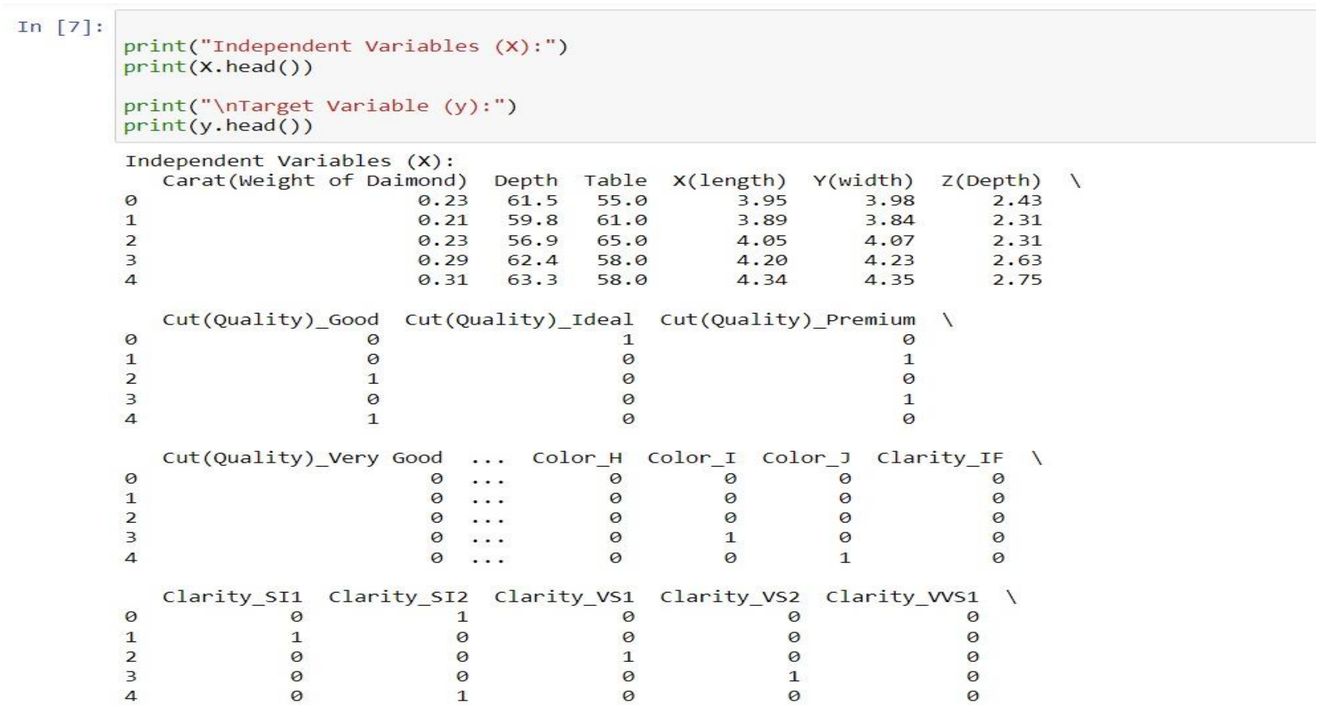
CHAPTER 5

RESULTS

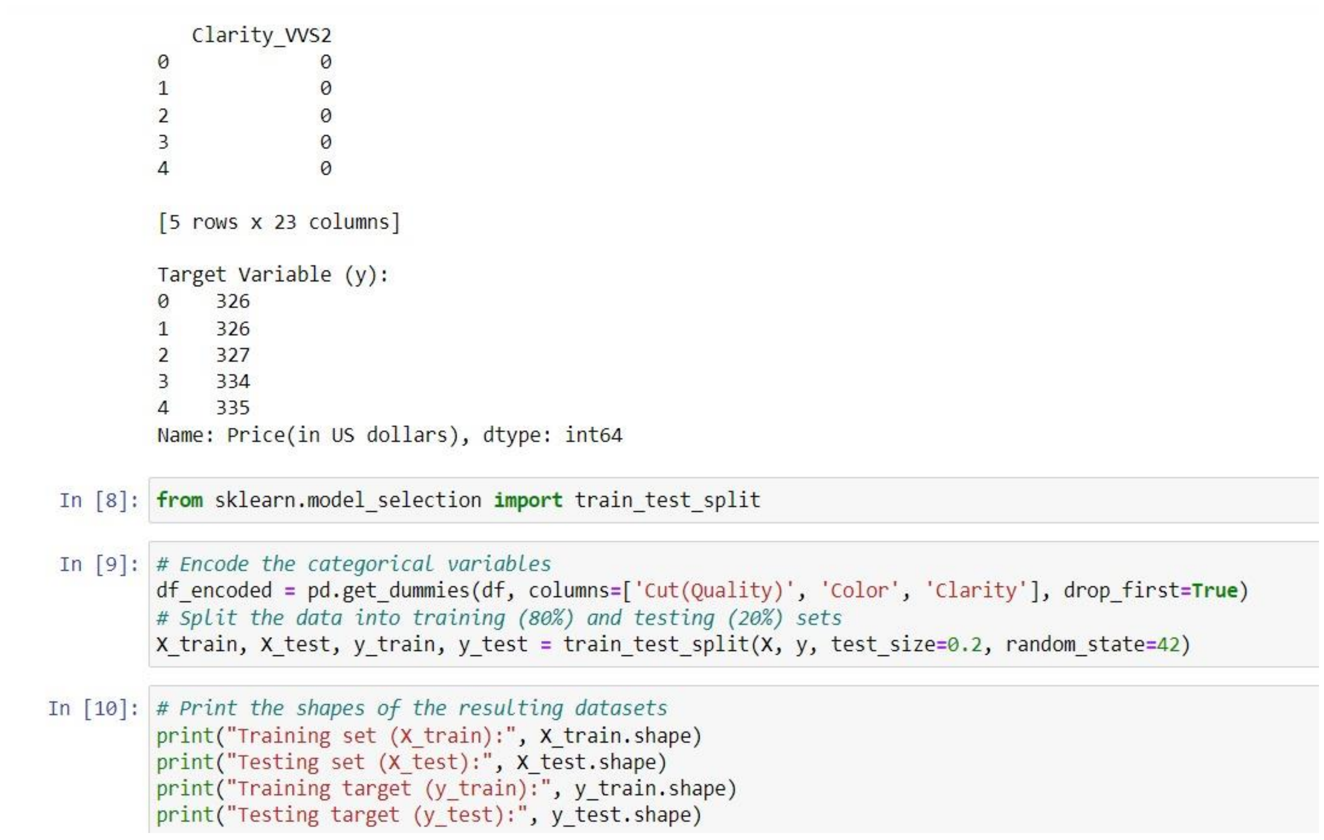
SCREENSHOTS:



```
In [1]: import pandas as pd

        # Replace 'dataset.csv' with your file path
        df = pd.read_csv('Diamond Price Prediction.csv')

In [2]: print(df.head())

           Carat(Weight of Daimond) Cut(Quality) Color Clarity  Depth  Table  \
        0                      0.23        Ideal     E     SI2   61.5   55.0
        1                      0.21      Premium     E     SI1   59.8   61.0
        2                      0.23         Good     E     VS1   56.9   65.0
        3                      0.29      Premium     I     VS2   62.4   58.0
        4                      0.31         Good     J     SI2   63.3   58.0

           Price(in US dollars)  X(length)  Y(width)  Z(Depth)
        0                   326       3.95      3.98      2.43
        1                   326       3.89      3.84      2.31
        2                   327       4.05      4.07      2.31
        3                   334       4.20      4.23      2.63
        4                   335       4.34      4.35      2.75

In [3]: df_encoded = pd.get_dummies(df, columns=['Cut(Quality)', 'Color', 'Clarity'], drop_first=True)

In [4]: # Separate independent (X) and dependent (y) variables
        X = df_encoded.drop(columns=['Price(in US dollars)'])  # Replace with your target column name
        y = df_encoded['Price(in US dollars)']  # Replace with your target column name
```



```
In [7]:
        print("Independent Variables (X):")
        print(X.head())

        print("\nTarget Variable (y):")
        print(y.head())

        Independent Variables (X):
           Carat(Weight of Daimond)  Depth  Table  X(length)  Y(width)  Z(Depth)  \
        0                      0.23   61.5   55.0       3.95      3.98      2.43
        1                      0.21   59.8   61.0       3.89      3.84      2.31
        2                      0.23   56.9   65.0       4.05      4.07      2.31
        3                      0.29   62.4   58.0       4.20      4.23      2.63
        4                      0.31   63.3   58.0       4.34      4.35      2.75

           Cut(Quality)_Good  Cut(Quality)_Ideal  Cut(Quality)_Premium  \
        0                  0                   1                     0
        1                  0                   0                     1
        2                  1                   0                     0
        3                  0                   0                     1
        4                  1                   0                     0

           Cut(Quality)_Very Good  ...  Color_H  Color_I  Color_J  Clarity_IF  \
        0                       0  ...        0        0        0           0
        1                       0  ...        0        0        0           0
        2                       0  ...        0        0        0           0
        3                       0  ...        0        1        0           0
        4                       0  ...        0        0        1           0

           Clarity_SI1  Clarity_SI2  Clarity_VS1  Clarity_VS2  Clarity_VVS1  \
        0            0            1            0            0             0
        1            1            0            0            0             0
        2            0            0            1            0             0
        3            0            0            0            1             0
        4            0            1            0            0             0
```

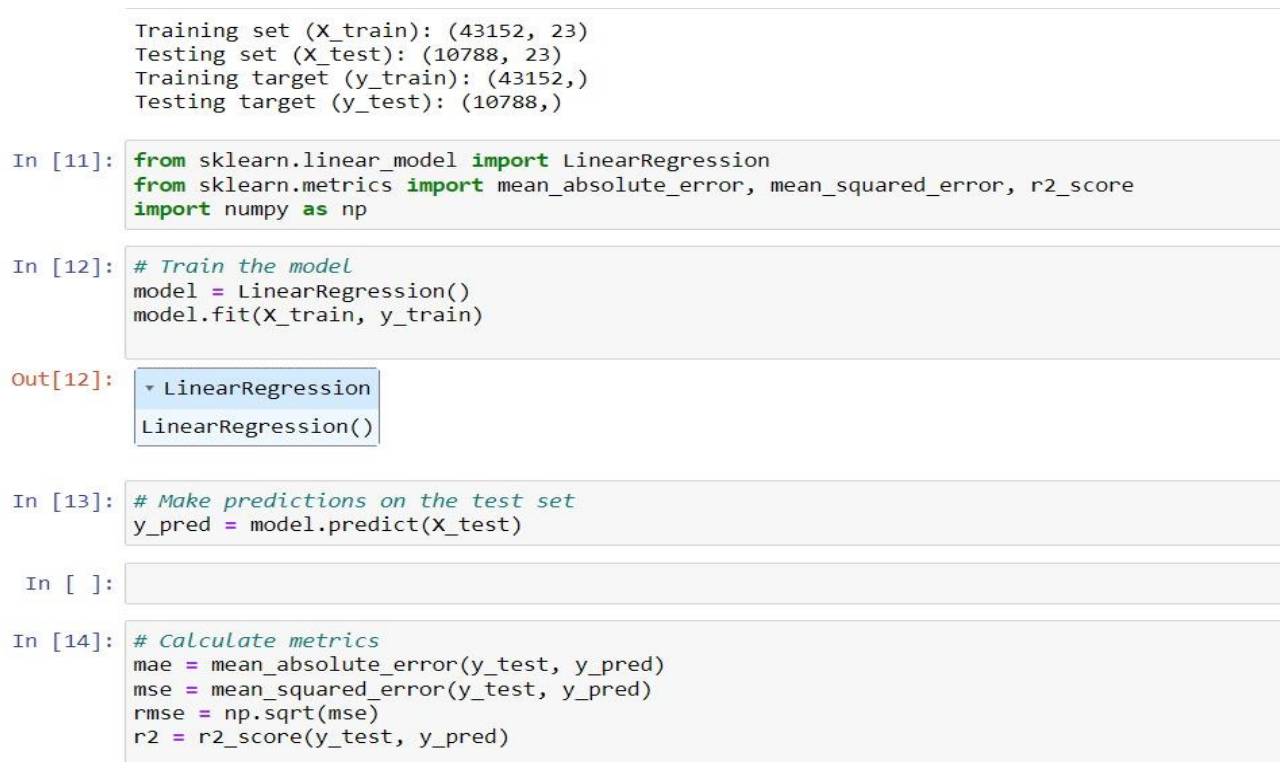


```
   Clarity_VVS2
0             0
1             0
2             0
3             0
4             0

[5 rows x 23 columns]

Target Variable (y):
0    326
1    326
2    327
3    334
4    335
Name: Price(in US dollars), dtype: int64
```

```
In [8]: from sklearn.model_selection import train_test_split
```

```
In [9]: # Encode the categorical variables
        df_encoded = pd.get_dummies(df, columns=['Cut(Quality)', 'Color', 'Clarity'], drop_first=True)
        # Split the data into training (80%) and testing (20%) sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [10]: # Print the shapes of the resulting datasets
         print("Training set (X_train):", X_train.shape)
         print("Testing set (X_test):", X_test.shape)
         print("Training target (y_train):", y_train.shape)
         print("Testing target (y_test):", y_test.shape)
```



```
Training set (X_train): (43152, 23)
Testing set (X_test): (10788, 23)
Training target (y_train): (43152,)
Testing target (y_test): (10788,)
```

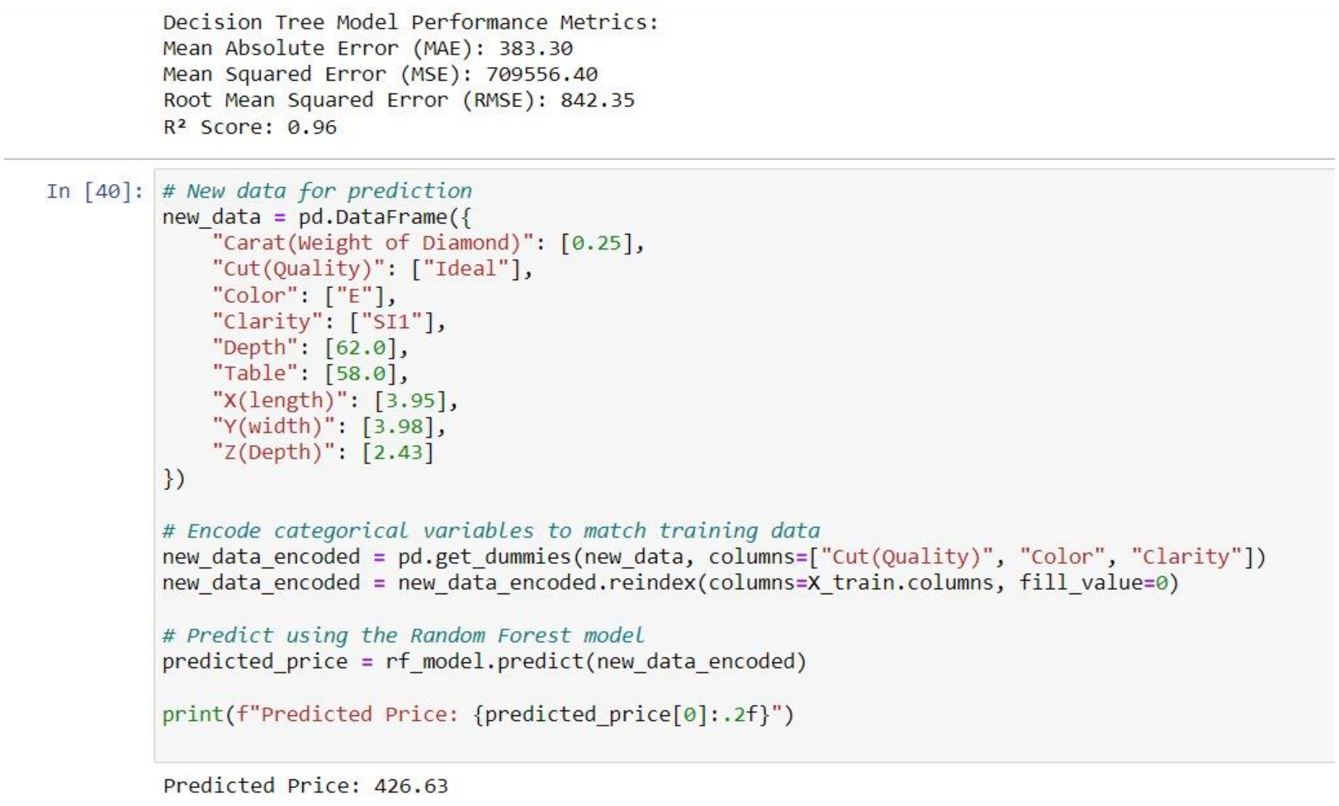
```
In [11]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
         import numpy as np
```

```
In [12]: # Train the model
         model = LinearRegression()
         model.fit(X_train, y_train)

Out[12]: ▾ LinearRegression
         LinearRegression()
```

```
In [13]: # Make predictions on the test set
         y_pred = model.predict(X_test)
```

```
In [ ]:
```

```
In [14]: # Calculate metrics
         mae = mean_absolute_error(y_test, y_pred)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)
         r2 = r2_score(y_test, y_pred)
```



```
Decision Tree Model Performance Metrics:
Mean Absolute Error (MAE): 383.30
Mean Squared Error (MSE): 709556.40
Root Mean Squared Error (RMSE): 842.35
R² Score: 0.96
```

```
In [40]: # New data for prediction
         new_data = pd.DataFrame({
             "Carat(Weight of Diamond)": [0.25],
             "Cut(Quality)": ["Ideal"],
             "Color": ["E"],
             "Clarity": ["SI1"],
             "Depth": [62.0],
             "Table": [58.0],
             "X(length)": [3.95],
             "Y(width)": [3.98],
             "Z(Depth)": [2.43]
         })

         # Encode categorical variables to match training data
         new_data_encoded = pd.get_dummies(new_data, columns=["Cut(Quality)", "Color", "Clarity"])
         new_data_encoded = new_data_encoded.reindex(columns=X_train.columns, fill_value=0)

         # Predict using the Random Forest model
         predicted_price = rf_model.predict(new_data_encoded)

         print(f"Predicted Price: {predicted_price[0]:.2f}")
```

```
Predicted Price: 426.63
```

CHAPTER 6

CONCLUSION

The application of machine learning to forecast diamond market trends offers a significant opportunity to enhance the accuracy and efficiency of pricing models in the gemstone industry. By analyzing complex datasets with features such as carat, cut, color, clarity, and other physical attributes, machine learning algorithms can uncover intricate patterns and relationships that traditional methods often overlook.

This project demonstrates that regression models like Linear Regression, Decision Tree Regressor, and Random Forest Regressor can effectively predict diamond prices. Among these, the Random Forest Regressor stands out, achieving higher accuracy and robustness due to its ability to handle non-linear relationships and interactions between variables. Metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R2R^2 scores validate the performance of these models, with Random Forest delivering the most precise predictions.

Machine learning-based forecasting empowers stakeholders, including jewelers, investors, and customers, by providing data-driven insights into diamond valuation. It reduces human bias, enhances market transparency, and aids in strategic decision-making, such as inventory management, pricing strategies, and customer engagement.

Looking ahead, integrating external market factors—such as global economic conditions, exchange rates, and consumer preferences—into the models could further improve prediction accuracy. Additionally, adopting advanced machine learning techniques like ensemble learning, deep learning, or explainable AI could enhance interpretability and predictive power.

In conclusion, harnessing machine learning for forecasting diamond market trends not only aligns with the evolving demands of a data-driven world but also lays the foundation for innovation and growth within the diamond industry.

CHAPTER 7

REFERENCES

[1] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

[2] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning* (2nd ed.). Springer.

[3] Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5-32.

[4] Quinlan, J. R. (1986). *Induction of Decision Trees*. Machine Learning, 1, 81–106.

[5] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.).

[6] Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling*. Springer.

[7] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.

[8] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.