Prepared by group 19

Daimond Clarity Prediction

Machine Learning

Team Members

Y ARAVIND REDDY

VU22CSEN0500188

M V S NITHEESH

VU22CSEN0100830

Motivation

The motivation behind this diamond clarity prediction project stems from the significant role that diamond clarity plays in determining a diamond's value and appeal. By accurately predicting diamond clarity, stakeholders in the diamond industry, such as jewelers, buyers, and graders, can:

- Improve accuracy and efficiency in diamond grading.
- Boost customer trust with consistent, transparent quality assessments.
- Align pricing with predicted clarity for better valuation.
- Minimize errors and subjectivity using automated models.

Problem Statement

Given a dataset of diamonds with various attributes (carat weight, price, cut, color, depth, etc.), design a machine learning model that accurately predicts the clarity grade of each diamond. The model's performance will be evaluated using accuracy and other relevant metrics to ensure reliable assessments of diamond quality.

Datasets

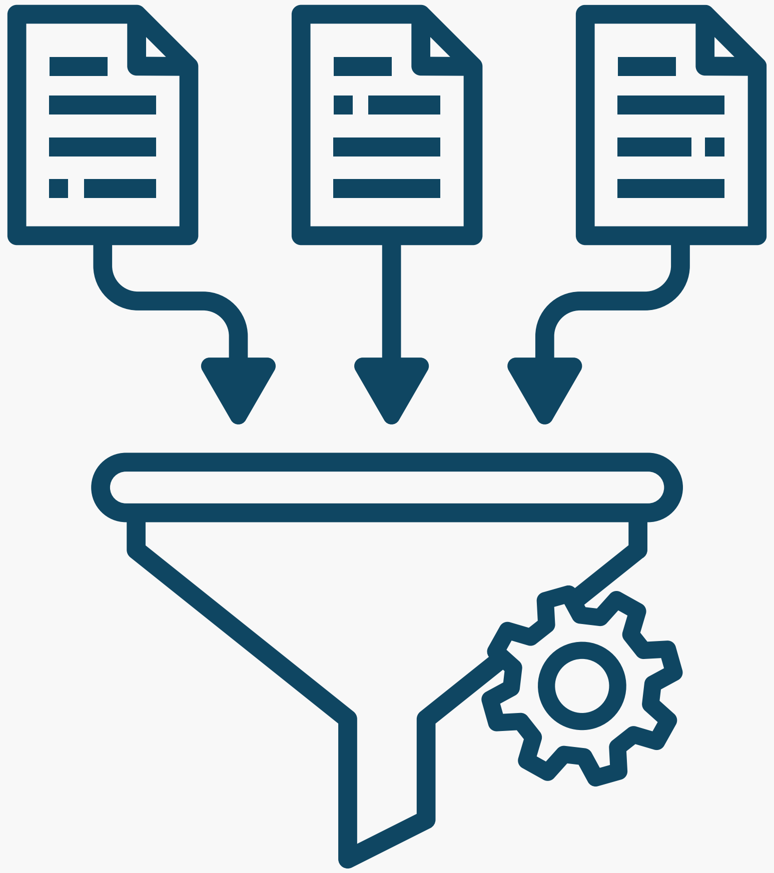
Dataset 3:

Shape: 26967 records with 11 attributes

Numerical Attributes: 7,

Catogerical Attributes: 3

No. of columns with missing data: 1



Dataset 1:

Shape: 53940 records with 10 attributes

Numerical Attributes: 8

Catogerical Attributes: 2

No. of columns with missing data: 0

Dataset 2:

Shape: 6000 records with 8 attributes

Numerical Attributes: 2

Catogerical Attributes: 6

No. of columns with missing data: 0

Models Utilized

- RANDOM FOREST CLASSIFIER
- Light Gradient Boosting Machine
- Extreme Gradient Boosting

Data Preprocessing of Dataset 1

Dataset 1 view:

	carat	cut	depth	table	price	x	y	z	Clarity	Color
0	0.23	1	61.5	55.0	326	3.95	3.98	2.43	SI2	E
1	0.21	2	59.8	61.0	326	3.89	3.84	2.31	SI1	E
2	0.23	4	56.9	65.0	327	4.05	4.07	2.31	VS1	E
3	0.29	2	62.4	58.0	334	4.20	4.23	2.63	VS2	I
4	0.31	4	63.3	58.0	335	4.34	4.35	2.75	SI2	J

	Variable Name	Description
0	Carat	Carat weight of the cubic zirconia.
1	Cut	Describe the cut quality of the cubic zirconi...
2	Color	Colour of the cubic zirconia.With D being the...
3	Clarity	cubic zirconia Clarity refers to the absence ...
4	Depth	The Height of a cubic zirconia, measured from...
5	Table	The Width of the cubic zirconia's Table expre...
6	Price	the Price of the cubic zirconia.
7	X	Length of the cubic zirconia in mm.
8	Y	Width of the cubic zirconia in mm.
9	Z	Height of the cubic zirconia in mm.

Understanding the Dataset 1

```
RangeIndex: 49472 entries, 0 to 49471
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    49472 non-null  float64
 1   cut      49472 non-null  int64
 2   depth    49472 non-null  float64
 3   table    49472 non-null  float64
 4   price    49472 non-null  int64
 5   x        49472 non-null  float64
 6   y        49472 non-null  float64
 7   z        49472 non-null  float64
 8   Clarity  49472 non-null  object
 9   Color    49471 non-null  object
dtypes: float64(6), int64(2), object(2)
memory usage: 3.8+ MB
```

Deep Dive into Dataset 1

Missing Values

carat	0
cut	0
depth	0
table	0
price	0
x	0
y	0
z	0
Clarity	0
Color	0

Duplicated Rows

Before Removing

```
[ ] df.duplicated().sum()
143
```

After Removing

```
[ ] df.duplicated().sum()
0
```

Deep Dive into Dataset 1

Categorical Values

```
categorical_cols = []
for col in df.columns:
    if df[col].nunique() <= 9:  # Consider columns with 9 or fewer unique values
        categorical_cols.append(col)

print("Categorical Columns:", categorical_cols)
print("Total number of categorical columns:", len(categorical_cols))
```

```
Categorical Columns: ['cut', 'Clarity', 'Color']
Total number of categorical columns: 3
```

Numerical Values

```
numerical_cols = []
for col in df.columns:
    if df[col].nunique() >= 9:  # Consider columns with at least 9 unique values
        numerical_cols.append(col)

print("Numerical Columns (with at least 9 unique values):", numerical_cols)
print("Total number of numerical columns (with at least 9 unique values):", len(numerical_cols))
```

```
Numerical Columns (with at least 9 unique values): ['carat', 'depth', 'table', 'price', 'x', 'y', 'z']
Total number of numerical columns (with at least 9 unique values): 7
```

Outlier Detection in Dataset 1:

```
Outlier Counts:
carat: 1873 outliers
depth: 2525 outliers
table: 604 outliers
price: 3523 outliers
x: 31 outliers
y: 28 outliers
z: 48 outliers
```

```python
# Function to calculate outliers using IQR
def calculate_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return len(outliers)

# Calculate outliers for each numerical column
outlier_counts = {}
for col in numerical_cols:
    outlier_counts[col] = calculate_outliers(df[col])

# Print outlier counts
print("Outlier Counts:")
for col, count in outlier_counts.items():
    print(f"{col}: {count} outliers")

# Plot box plots for numerical columns
plt.figure(figsize=(12, 8))
df[numerical_cols].boxplot()
plt.title('Box Plot for Numerical Attributes', fontsize=14)
plt.ylabel('Values', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()
plt.show()
```

SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) generates synthetic data points for the minority class by interpolating between existing minority class samples.

Benefits for our case:

- Improves class balance: *Helps address the class imbalance in the dataset.*
- Enhances model performance: *Increases the model's ability to correctly classify minority class samples.*
- Reduces bias: *Prevents overfitting and underperformance for minority classes, improving overall accuracy.*

SMOTE of Dataset 1:

Before

Smote

```
data_encoded['Clarity'].value_counts()
```

Clarity	count
2	13032
5	12229
3	9150
4	8156
7	5056
6	3647
1	1784
0	740

Accuracy

```
Accuracy: 0.6734826656752486
              precision    recall  f1-score   support

           0       0.84      0.67      0.74       141
           1       0.72      0.61      0.66       366
           2       0.67      0.74      0.71      2553
           3       0.76      0.77      0.77      1857
           4       0.59      0.56      0.57      1649
           5       0.65      0.67      0.66      2413
           6       0.69      0.59      0.63       741
           7       0.66      0.60      0.63      1039

    accuracy                           0.67     10759
   macro avg       0.70      0.65      0.67     10759
weighted avg       0.67      0.67      0.67     10759
```

SMOTE of Dataset 1:

After

Smote

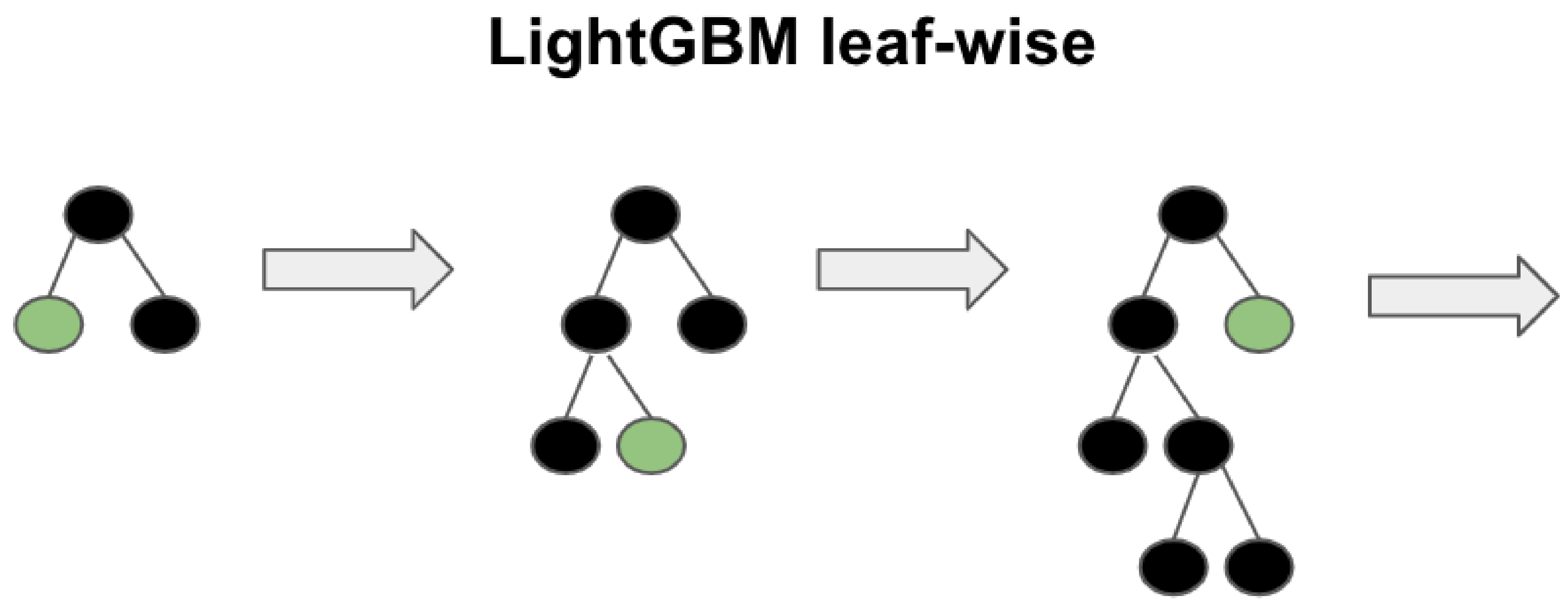
```
balanced_data['Clarity'].value_counts()
```

Clarity	count
3	13032
2	13032
4	13032
5	13032
7	13032
6	13032
0	13032
1	13032

Accuracy

```
Test Set Accuracy: 0.77
Classification Report:
              precision    recall  f1-score

           0       0.98      0.99      0.98
           1       0.88      0.90      0.89
           2       0.67      0.68      0.67
           3       0.81      0.85      0.83
           4       0.64      0.64      0.64
           5       0.64      0.62      0.63
           6       0.80      0.77      0.78
           7       0.75      0.72      0.73

    accuracy                           0.77
   macro avg       0.77      0.77      0.77
weighted avg       0.77      0.77      0.77
```

[Light Gradient Boosting Machine]

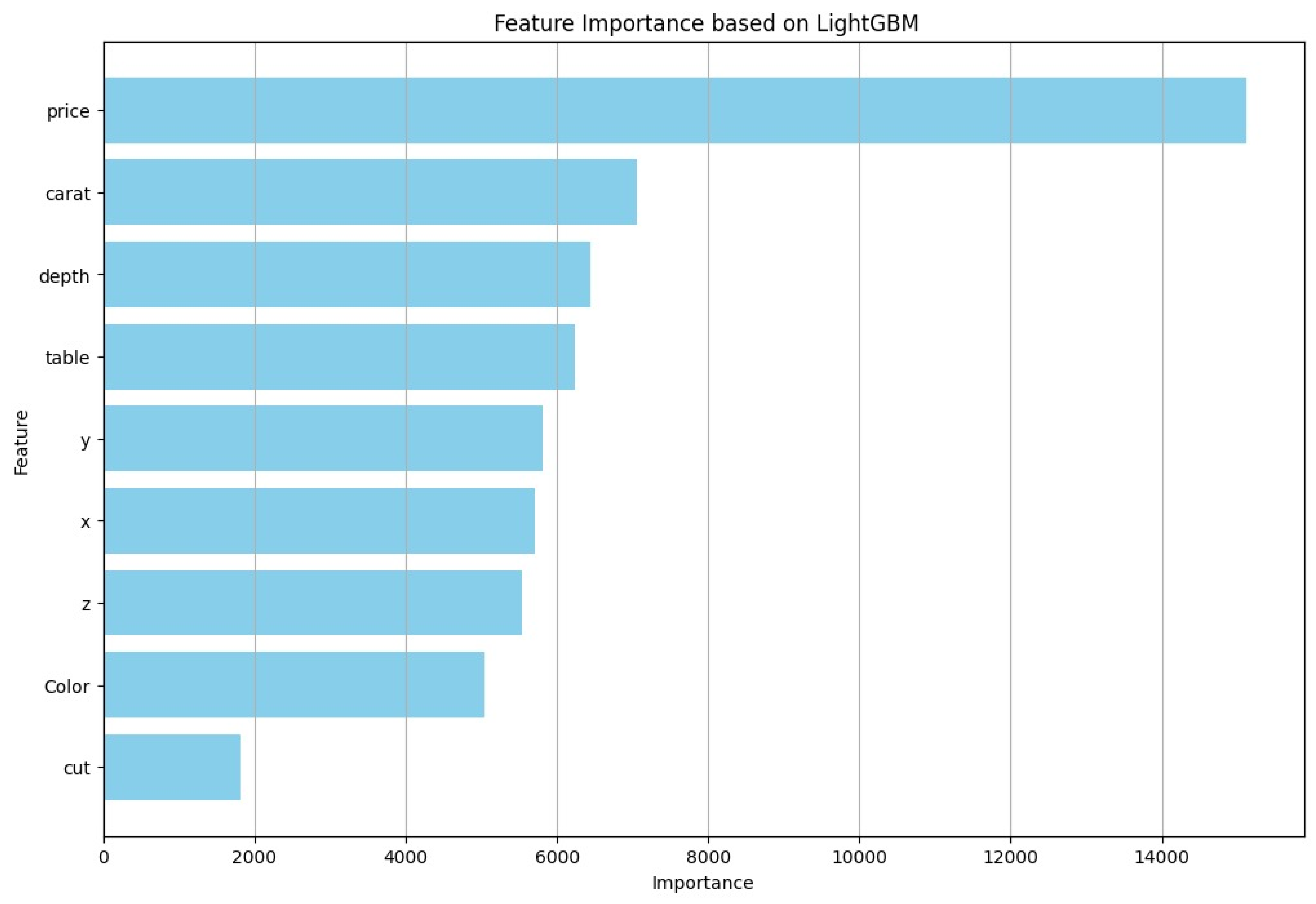


Model Training using LGBoost Dataset 1:

```
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Define the reduced parameter distribution
param_dist = {
    'n_estimators': [100, 150],
    'learning_rate': [0.1, 0.05],
    'max_depth': [6, 8]
}
# Initialize the LightGBM model
lgb = LGBMClassifier(random_state=42)
# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=lgb,
    param_distributions=param_dist,
    n_iter=10,  # Smaller number of iterations for speed
    cv=3,
    scoring='balanced_accuracy',
    random_state=42
)
# Fit RandomizedSearchCV
random_search.fit(X_train_resampled, y_train_resampled)
# Get the best parameters and model
best_params = random_search.best_params_
print("Best Parameters:", best_params)
best_lgb_model = random_search.best_estimator_
# Predictions and evaluation
y_pred = best_lgb_model.predict(X_test)
# Print evaluation metrics
print("\nBalanced Accuracy Score:")
print(balanced_accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

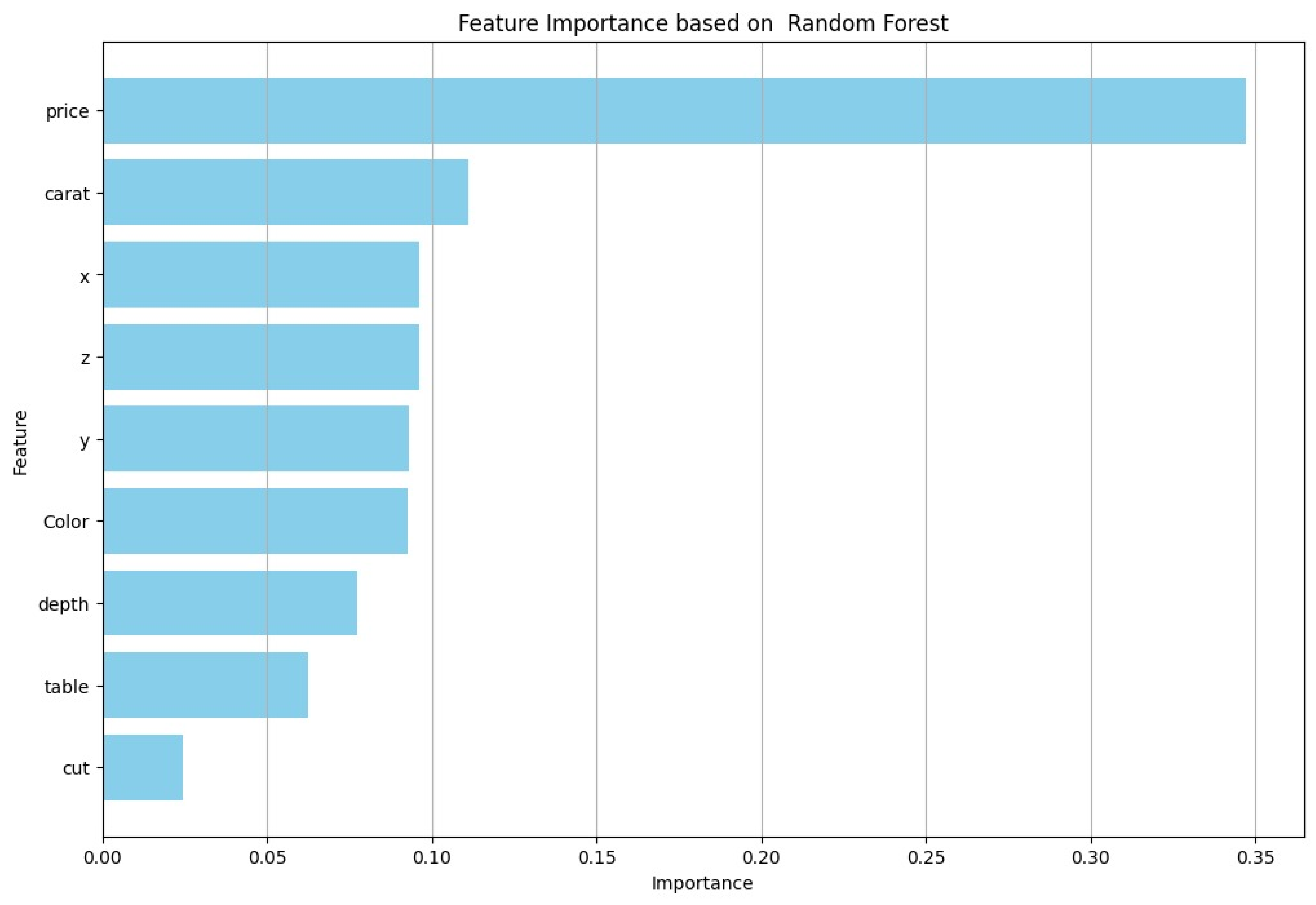
Feature Importance

Dataset 1:



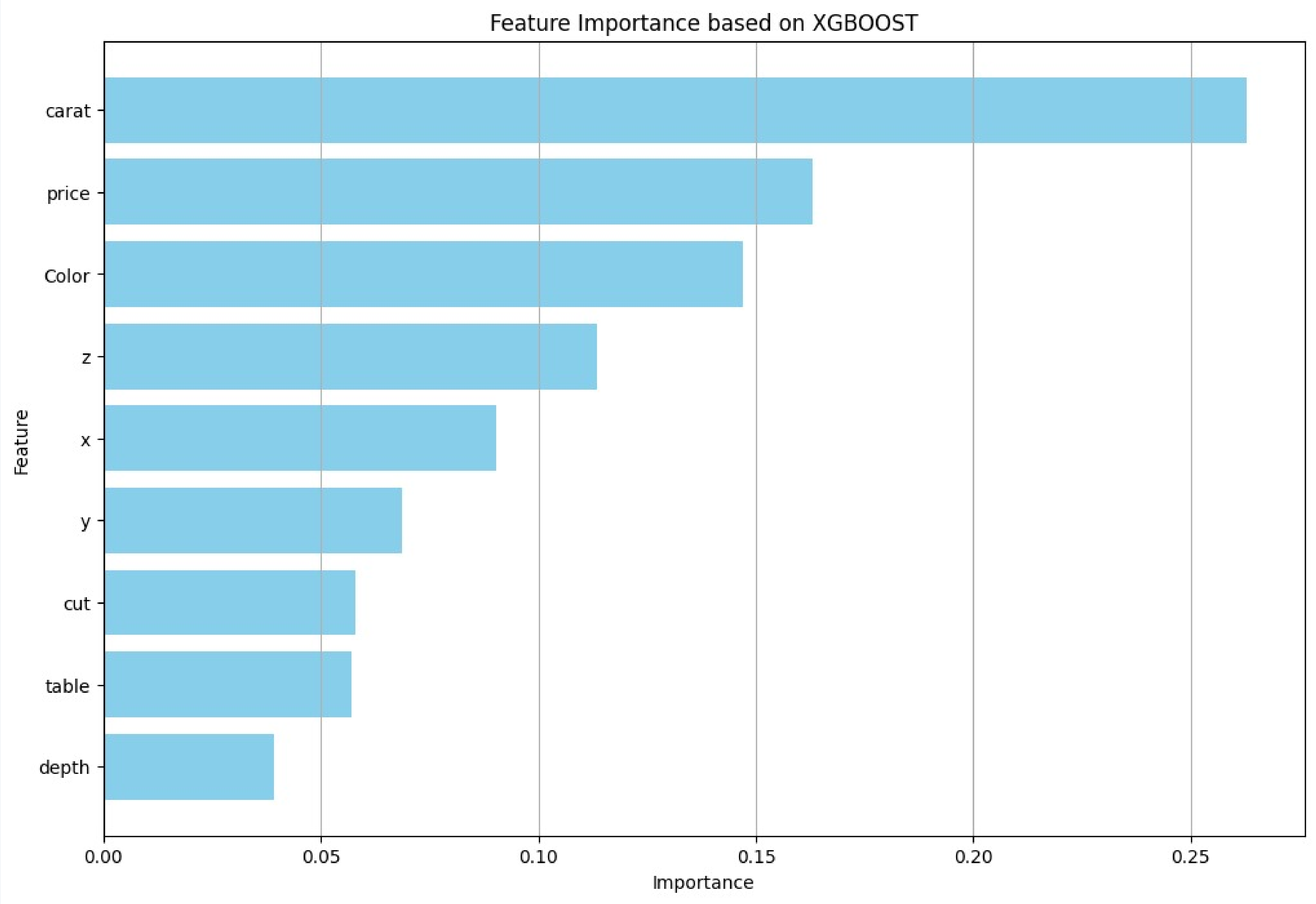
Feature Importance

Dataset 1:



Feature Importance

Dataset 1:



Dataset-1 Results :

Light GBM

```
Balanced Accuracy Score:
0.7964624018411843

Classification Report:
              precision    recall  f1-score   s

           0       1.00      1.00      1.00
           1       0.93      0.95      0.94
           2       0.82      0.86      0.84
           3       0.63      0.59      0.61
           4       0.61      0.58      0.60
           5       0.83      0.88      0.85
           6       0.73      0.72      0.73

    accuracy                           0.80
   macro avg       0.79      0.80      0.79
weighted avg       0.79      0.80      0.79
```

```
'n_estimators': 150, 'max_depth': 8, 'learning_rate': 0.1
```

Random Forest

```
Test Set Accuracy: 0.75
Classification Report:
              precision    recall  f1-score

           0       1.00      1.00      1.00
           1       0.88      0.92      0.90
           2       0.74      0.83      0.78
           3       0.58      0.54      0.56
           4       0.58      0.53      0.55
           5       0.78      0.83      0.81
           6       0.68      0.64      0.66

    accuracy                           0.75
   macro avg       0.75      0.75      0.75
weighted avg       0.75      0.75      0.75
```

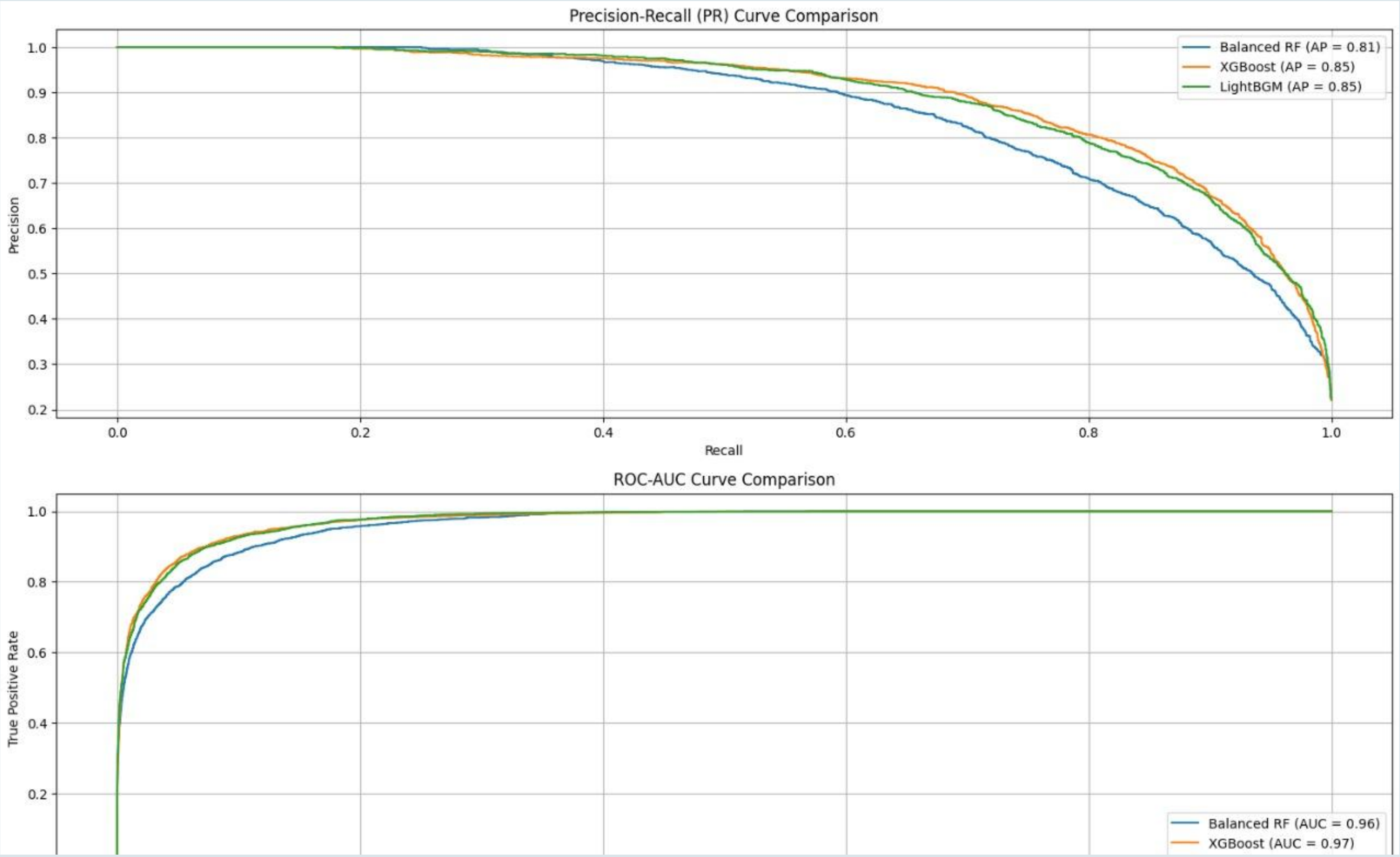
```
{'n_estimators': 150, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 20, 'bootstrap': True}
```

XGBoost

```
Accuracy: 0.8012486992715921
Classification Report:
              precision    recall  f1-score

           0       1.00      1.00      1.00
           1       0.91      0.93      0.92
           2       0.85      0.87      0.86
           3       0.62      0.62      0.62
           4       0.64      0.62      0.63
           5       0.81      0.86      0.84
           6       0.76      0.71      0.74

    accuracy                           0.80
   macro avg       0.80      0.80      0.80
weighted avg       0.80      0.80      0.80
```

```
'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 200
```

Dataset 1: PR & ROC Curve Comparision



Data Preprocessing of Dataset 2

Dataset 2 view:

	Carat Weight	Cut	Color	Clarity	Polish	Symmetry	Report	Price
0	1.10	Ideal	H	SI1	VG	EX	GIA	5169
1	0.83	Ideal	H	VS1	ID	ID	AGSL	3470
2	0.85	Ideal	H	SI1	EX	EX	GIA	3183
3	0.91	Ideal	E	SI1	VG	VG	GIA	4370
4	0.83	Ideal	G	SI1	EX	EX	GIA	3171

Understanding the Dataset 2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Carat Weight   6000 non-null   float64
 1   Cut            6000 non-null   object
 2   Color          6000 non-null   object
 3   Clarity        6000 non-null   object
 4   Polish         6000 non-null   object
 5   Symmetry       6000 non-null   object
 6   Report         6000 non-null   object
 7   Price          6000 non-null   int64
dtypes: float64(1), int64(1), object(6)
memory usage: 375.1+ KB
```

Deep Dive into Dataset 2

Missing Values

	0
Carat Weight	0
Cut	0
Color	0
Clarity	0
Polish	0
Symmetry	0
Report	0
Price	0

dtype: int64

Duplicated Rows

Before Removing

```
[35] df.duplicated().sum()
83
```

After Removing

```
[ ] df.duplicated().sum()
0
```

Deep Dive into Dataset 2

Categorial Values

```
categorical_cols = []
for col in df.columns:
    if df[col].nunique() <= 9:  # Consider columns with 9 or fewer unique values
        categorical_cols.append(col)

print("Categorical Columns:", categorical_cols)
print("Total number of categorical columns:", len(categorical_cols))
```

```
Categorical Columns: ['Cut', 'Color', 'Clarity', 'Polish', 'Symmetry', 'Report']
Total number of categorical columns: 6
```

Numerical Values

```
[37] numerical_cols = []
for col in df.columns:
    if df[col].nunique() >= 9:  # Consider columns with at least 9 unique values
        numerical_cols.append(col)

print("Numerical Columns (with at least 9 unique values):", numerical_cols)
print("Total number of numerical columns (with at least 9 unique values):", len(numerical_cols))
```

```
Numerical Columns (with at least 9 unique values): ['Carat Weight', 'Price']
Total number of numerical columns (with at least 9 unique values): 2
```

Categorial Values

```
for col in categorical_cols:
    print(f"\nUnique values for {col}:")
    print(df[col].unique())
```

```
Unique values for Cut:
['Ideal' 'Very Good' 'Fair' 'Good' 'Signature-Ideal']

Unique values for Color:
['H' 'E' 'G' 'D' 'F' 'I']

Unique values for Clarity:
['SI1' 'VS1' 'VS2' 'VVS2' 'VVS1' 'IF' 'FL']

Unique values for Polish:
['VG' 'ID' 'EX' 'G']

Unique values for Symmetry:
['EX' 'ID' 'VG' 'G']

Unique values for Report:
['GIA' 'AGSL']
```

Outlier Detection in Dataset 2:

```
Outlier Counts:
Carat Weight: 93 outliers
Price: 379 outliers
```

```python
# Function to calculate outliers using IQR
def calculate_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return len(outliers)

# Calculate outliers for each numerical column
outlier_counts = {}
for col in numerical_cols:
    outlier_counts[col] = calculate_outliers(df[col])

# Print outlier counts
print("Outlier Counts:")
for col, count in outlier_counts.items():
    print(f"{col}: {count} outliers")

# Plot box plots for numerical columns
plt.figure(figsize=(12, 8))
df[numerical_cols].boxplot()
plt.title('Box Plot for Numerical Attributes', fontsize=14)
plt.ylabel('Values', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()
plt.show()
```

```python
# Initialize and train a Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
              precision    recall  f1-score   support

           1       0.52      0.46      0.49        37
           2       0.75      0.84      0.79       402
           3       0.41      0.39      0.40       242
           4       0.52      0.54      0.53       312
           5       0.26      0.19      0.22        54
           6       0.48      0.41      0.44       153

    accuracy                           0.57      1200
   macro avg       0.49      0.47      0.48      1200
weighted avg       0.56      0.57      0.57      1200

Accuracy: 0.5741666666666667
```

SMOTE of Dataset 2:

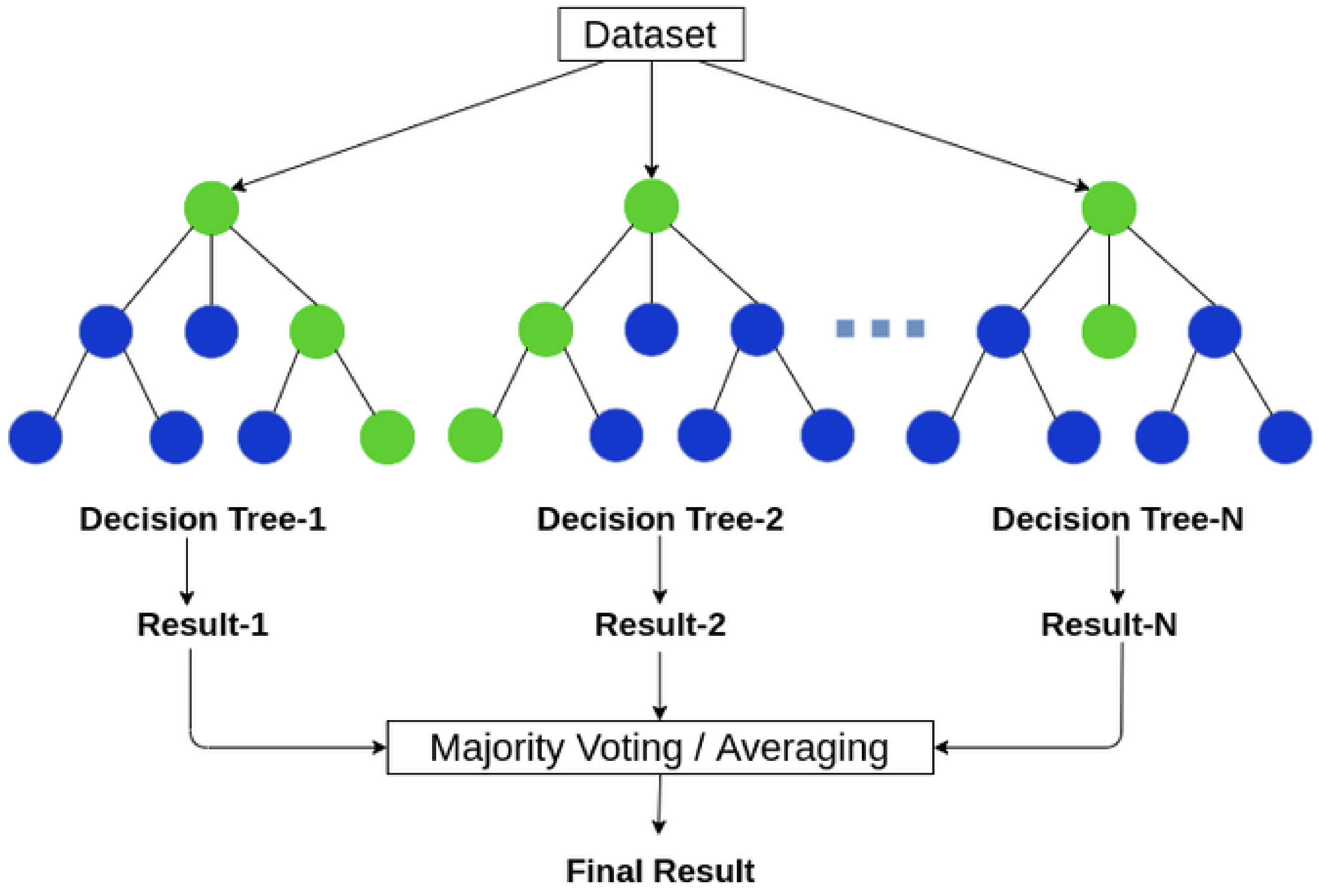
Before Smote

Clarity	count
2	2059
4	1575
3	1192
6	666
5	285
1	219
0	4

After Smote

Clarity	count
2	2059
3	2059
4	2059
6	2059
5	2059
1	2059
0	2059

Random Forest



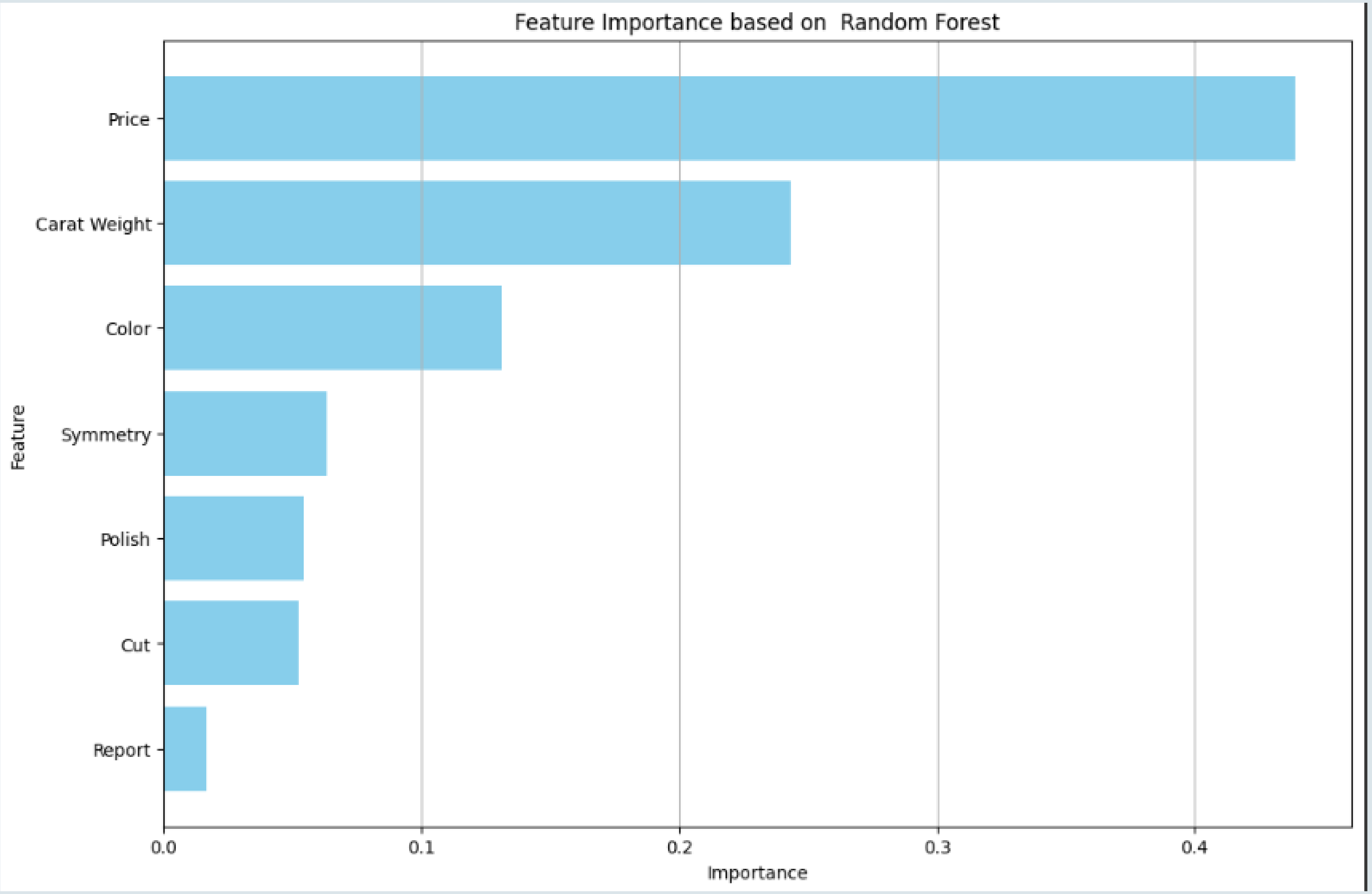
Model Training using Random Forest Dataset 2:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, accuracy_score

# Define a reduced parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize a Random Forest Classifier
rf = RandomForestClassifier(random_state=42)

# Set up RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_grid,
                                   n_iter=50, cv=3, n_jobs=-1, verbose=2, scoring='accuracy', random_state=42)

# Fit RandomizedSearchCV on training data
random_search.fit(X_train, y_train)

# Get the best parameters and best model
best_rf = random_search.best_estimator_
print("Best Parameters:", random_search.best_params_)

# Evaluate the best model
y_pred = best_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print results
print(f"Test Set Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

Best Parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': None, 'bootstrap': False}

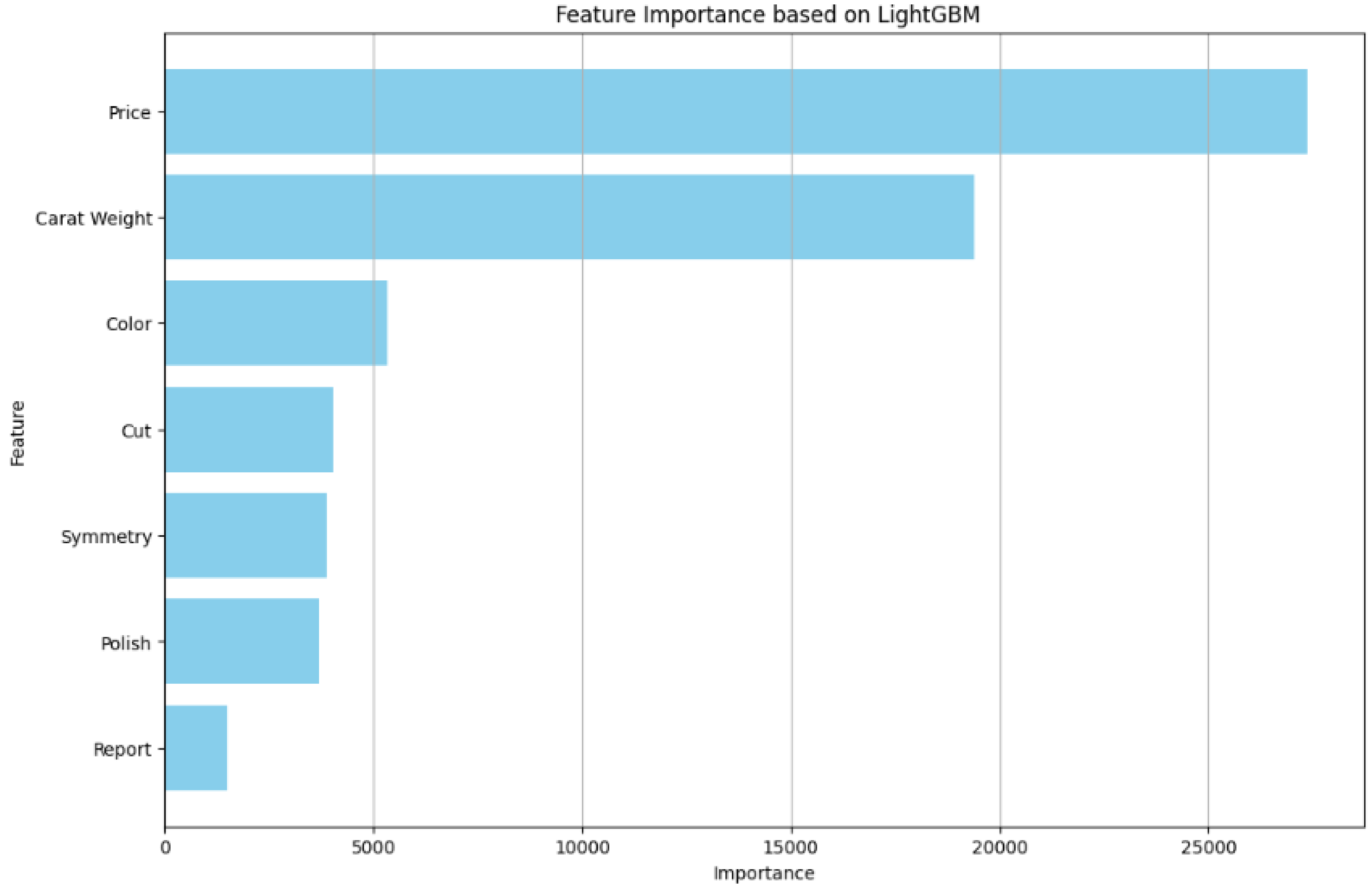
Dataset 2:

Feature Importance



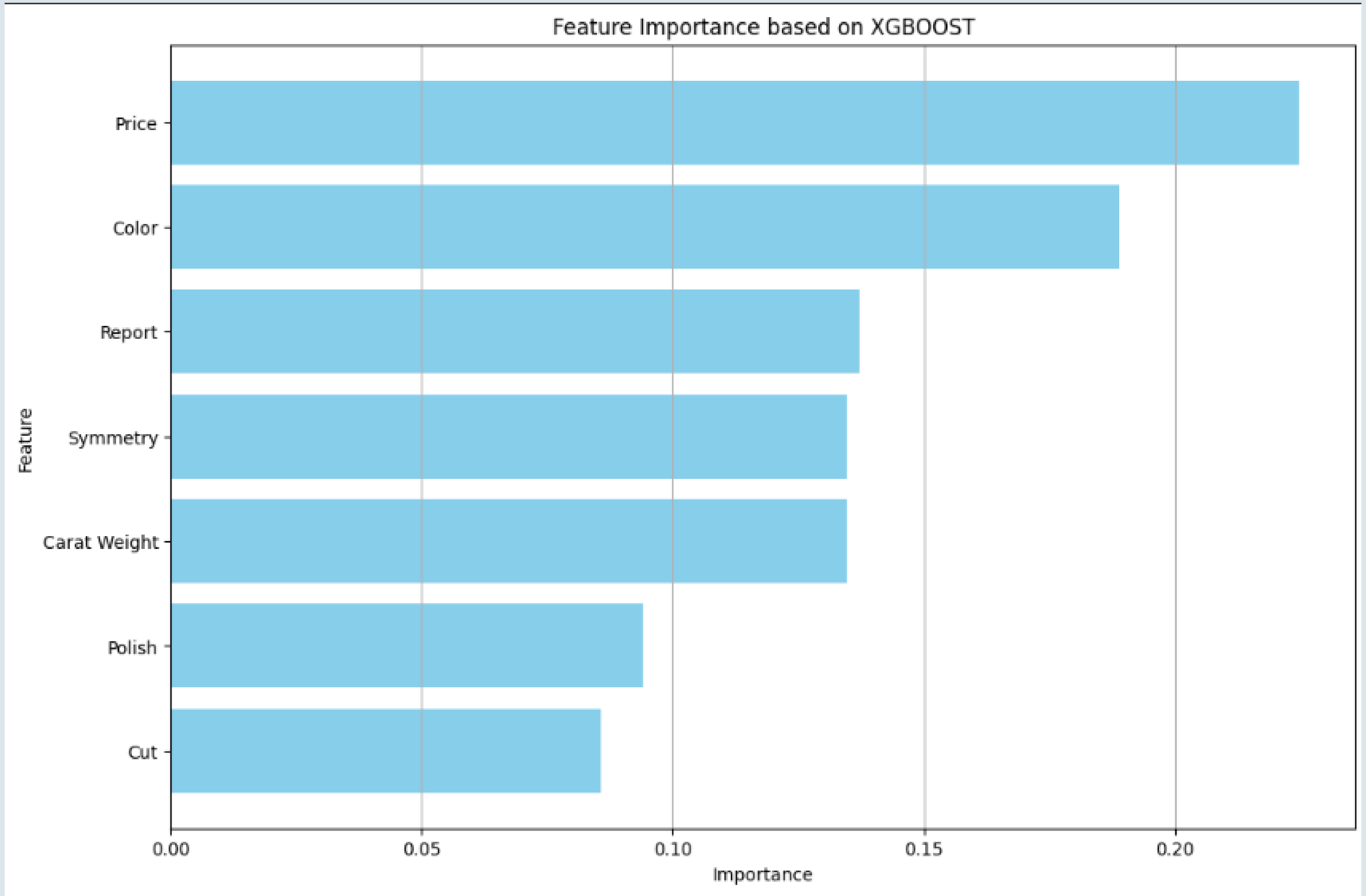
Dataset 2:

Feature Importance



Feature Importance

Dataset 2:



Dataset-2 Results :

Random Forest

Test Set Accuracy: 0.75

Classification Report:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	0.88	0.92	0.90
2	0.74	0.83	0.78
3	0.58	0.54	0.56
4	0.58	0.53	0.55
5	0.78	0.83	0.81
6	0.68	0.64	0.66
accuracy			0.75
macro avg	0.75	0.75	0.75
weighted avg	0.75	0.75	0.75

Best Parameters: {'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_depth': None, 'bootstrap': False}

Light GBM

Balanced Accuracy Score:
0.7964624018411843

Classification Report:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	0.93	0.95	0.94
2	0.82	0.86	0.84
3	0.63	0.59	0.61
4	0.61	0.58	0.60
5	0.83	0.88	0.85
6	0.73	0.72	0.73
accuracy			0.80
macro avg	0.79	0.80	0.79
weighted avg	0.79	0.80	0.79

n_estimators': 200, 'max_depth': 8, 'learning_rate': 0.1

XGBoost

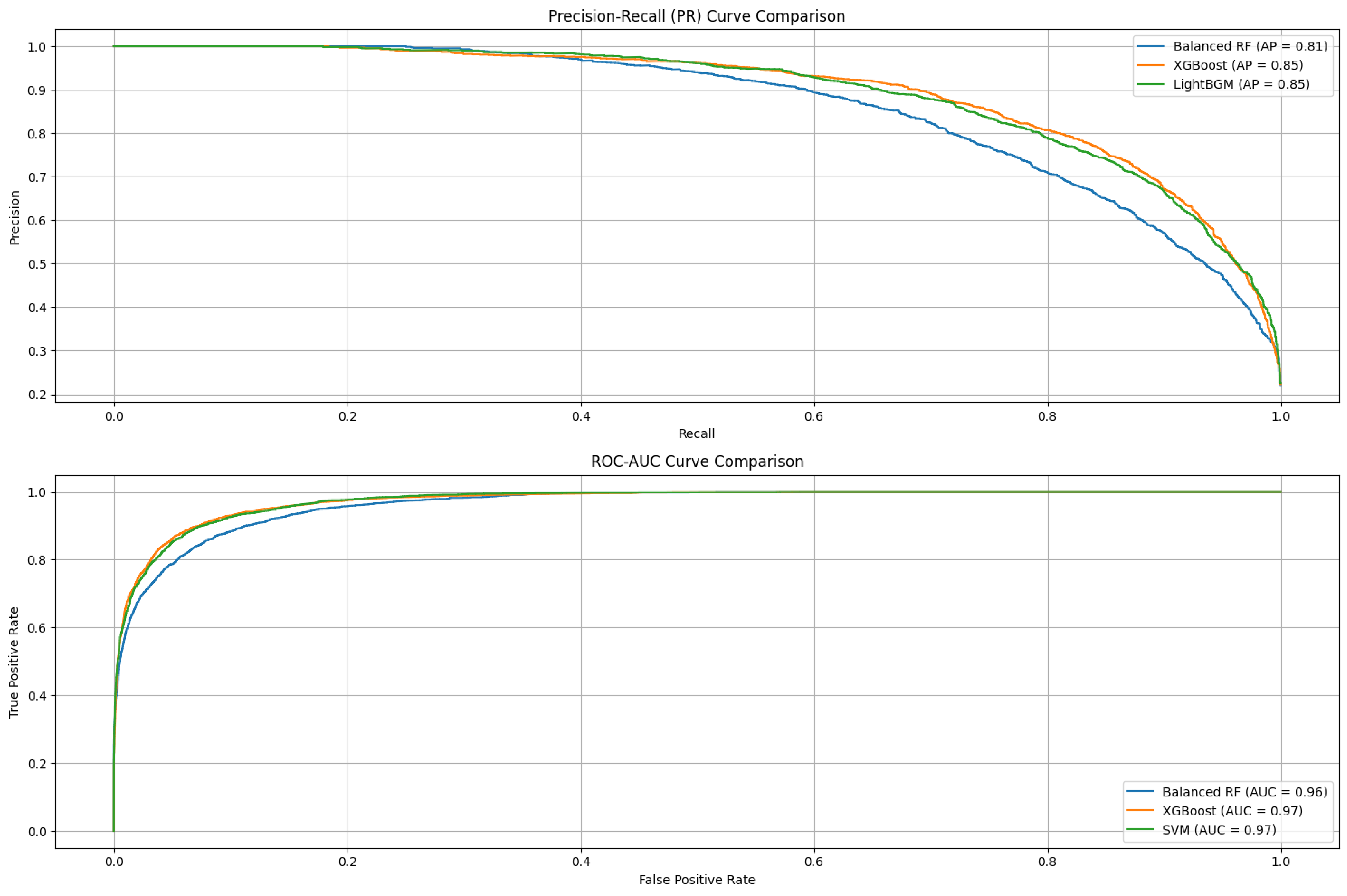
Accuracy: 0.8012486992715921

Classification Report:

	precision	recall	f1-score
0	1.00	1.00	1.00
1	0.91	0.93	0.92
2	0.85	0.87	0.86
3	0.62	0.62	0.62
4	0.64	0.62	0.63
5	0.81	0.86	0.84
6	0.76	0.71	0.74
accuracy			0.80
macro avg	0.80	0.80	0.80
weighted avg	0.80	0.80	0.80

n_estimators': 200, 'max_depth': 8, 'learning_rate': 0.

Dataset 2: PR & ROC Curve Comparision



Data Preprocessing of Dataset 3

Dataset 3 view:

	S.No	carat	cut	color	clarity	depth	table	x	y	z	price
0	1	0.30	Ideal	E	SI1	62.1	58.0	4.27	4.29	2.66	499
1	2	0.33	Premium	G	IF	60.8	58.0	4.42	4.46	2.70	984
2	3	0.90	Very Good	E	VVS2	62.2	60.0	6.04	6.12	3.78	6289
3	4	0.42	Ideal	F	VS1	61.6	56.0	4.82	4.80	2.96	1082
4	5	0.31	Ideal	F	VVS1	60.4	59.0	4.35	4.43	2.65	779

Understanding the Dataset 3

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26967 entries, 0 to 26966
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   S.No      26967 non-null  int64
 1   carat     26967 non-null  float64
 2   cut       26967 non-null  object
 3   color     26967 non-null  object
 4   clarity   26967 non-null  object
 5   depth     26270 non-null  float64
 6   table     26967 non-null  float64
 7   x         26967 non-null  float64
 8   y         26967 non-null  float64
 9   z         26967 non-null  float64
 10  price     26967 non-null  int64
dtypes: float64(6), int64(2), object(3)
memory usage: 2.3+ MB
```

Deep Dive into Dataset 3

Missing Values

Before removing

S.No	0
carat	0
cut	0
color	0
clarity	0
depth	697
table	0
x	0
y	0
z	0
price	0

dtype: int64

After removing

S.No	0
carat	0
cut	0
color	0
clarity	0
depth	0
table	0
x	0
y	0
z	0
price	0

dtype: int64

Duplicated Rows

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

0
```

Deep Dive into Dataset 3

Categorial Values

```
categorical_cols = []
for col in df.columns:
    if df[col].nunique() <= 9:  # Consider columns with 9 or fewer unique values
        categorical_cols.append(col)

print("Categorical Columns:", categorical_cols)
print("Total number of categorical columns:", len(categorical_cols))
```

```
Categorical Columns: ['cut', 'color', 'clarity']
Total number of categorical columns: 3
```

Numerical Values

```
numerical_cols = []
for col in df.columns:
    if df[col].nunique() >= 9:  # Consider columns with at least 9 unique values
        numerical_cols.append(col)

print("Numerical Columns (with at least 9 unique values):", numerical_cols)
print("Total number of numerical columns (with at least 9 unique values):", len(numerical_cols))
```

```
Numerical Columns (with at least 9 unique values): ['S.No', 'carat', 'depth', 'table', 'x', 'y', 'z', 'price']
Total number of numerical columns (with at least 9 unique values): 8
```

Outlier Detection in Dataset 3:

```
Outlier Counts:
S.No: 0 outliers
carat: 662 outliers
depth: 1419 outliers
table: 318 outliers
x: 15 outliers
y: 15 outliers
z: 23 outliers
price: 1779 outliers
```

```python
# Function to calculate outliers using IQR
def calculate_outliers(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return len(outliers)

# Calculate outliers for each numerical column
outlier_counts = {}
for col in numerical_cols:
    outlier_counts[col] = calculate_outliers(df[col])

# Print outlier counts
print("Outlier Counts:")
for col, count in outlier_counts.items():
    print(f"{col}: {count} outliers")

# Plot box plots for numerical columns
plt.figure(figsize=(12, 8))
df[numerical_cols].boxplot()
plt.title('Box Plot for Numerical Attributes', fontsize=14)
plt.ylabel('Values', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()
plt.show()
```

SMOTE of Dataset 3:

Before Smote

clarity	
2	6571
5	6099
3	4575
4	4093
7	2531
6	1839
1	894
0	365

dtype: int64

After Smote

clarity	
2	6571
1	6571
7	6571
4	6571
6	6571
5	6571
3	6571
0	6571

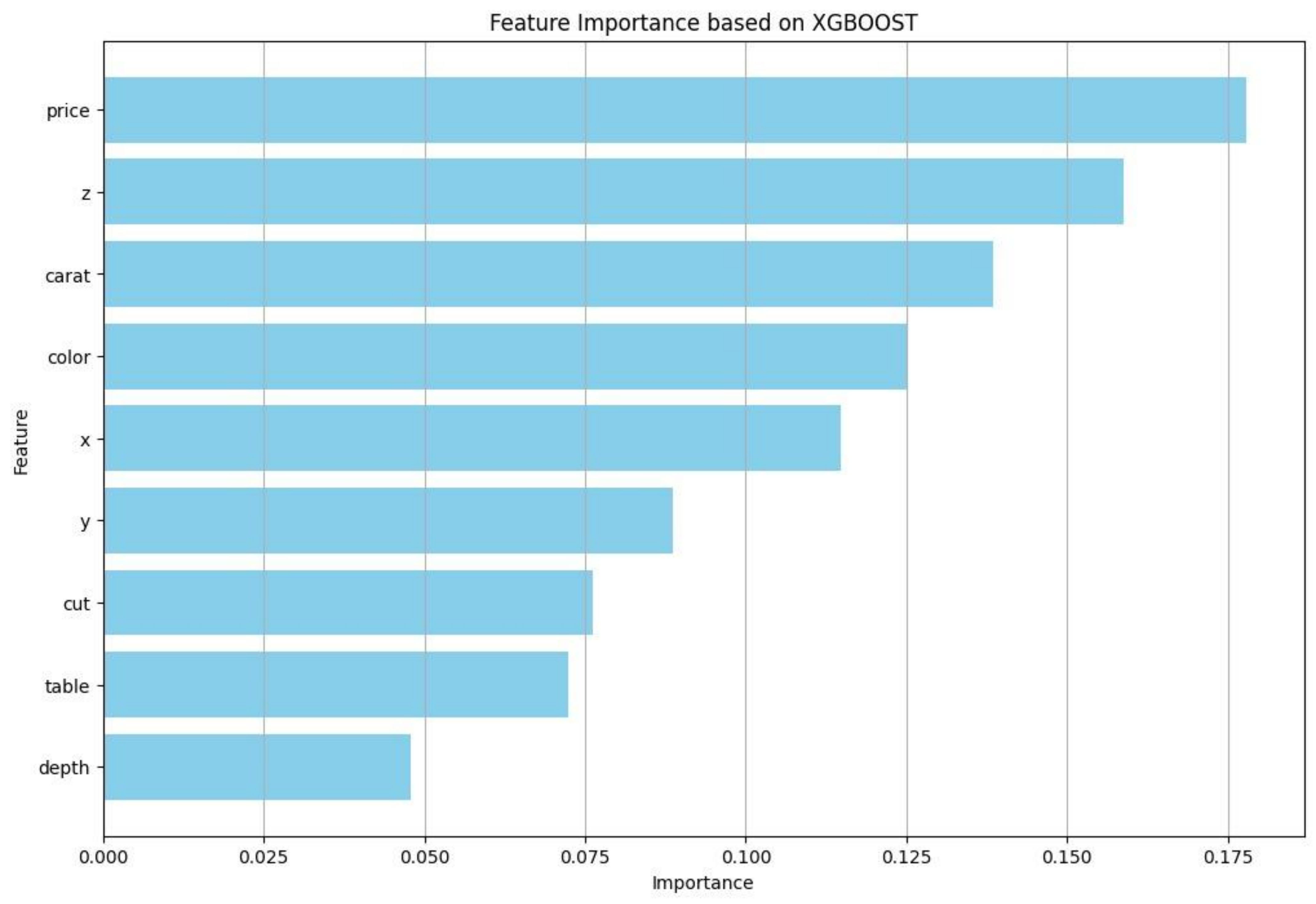
Extreme Gradient Boost

Model Training using XGBoost Dataset 3:

```
# Calculate class weights (for multi-class classification)
class_weights = {class_label: len(y_train) / (len(np.unique(y_train))) for class_label in np
    .unique(y_train)}
# Define parameter grid for GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [9],
    'learning_rate': [0.1, 0.05, 0.01]
}
xgb_classifier = XGBClassifier(
    random_state=42,
    objective='multi:softmax',
    eval_metric="mlogloss"
)
# Perform GridSearchCV
grid_search = GridSearchCV(
    estimator=xgb_classifier,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    verbose=2
)
grid_search.fit(X_train, y_train)
# Best Parameters
best_params = grid_search.best_params_
print(f"Best Parameters: {best_params}")
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train, sample_weight=[class_weights[y] for y in y_train])
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print("Classification Report:\n", classification_rep)
```

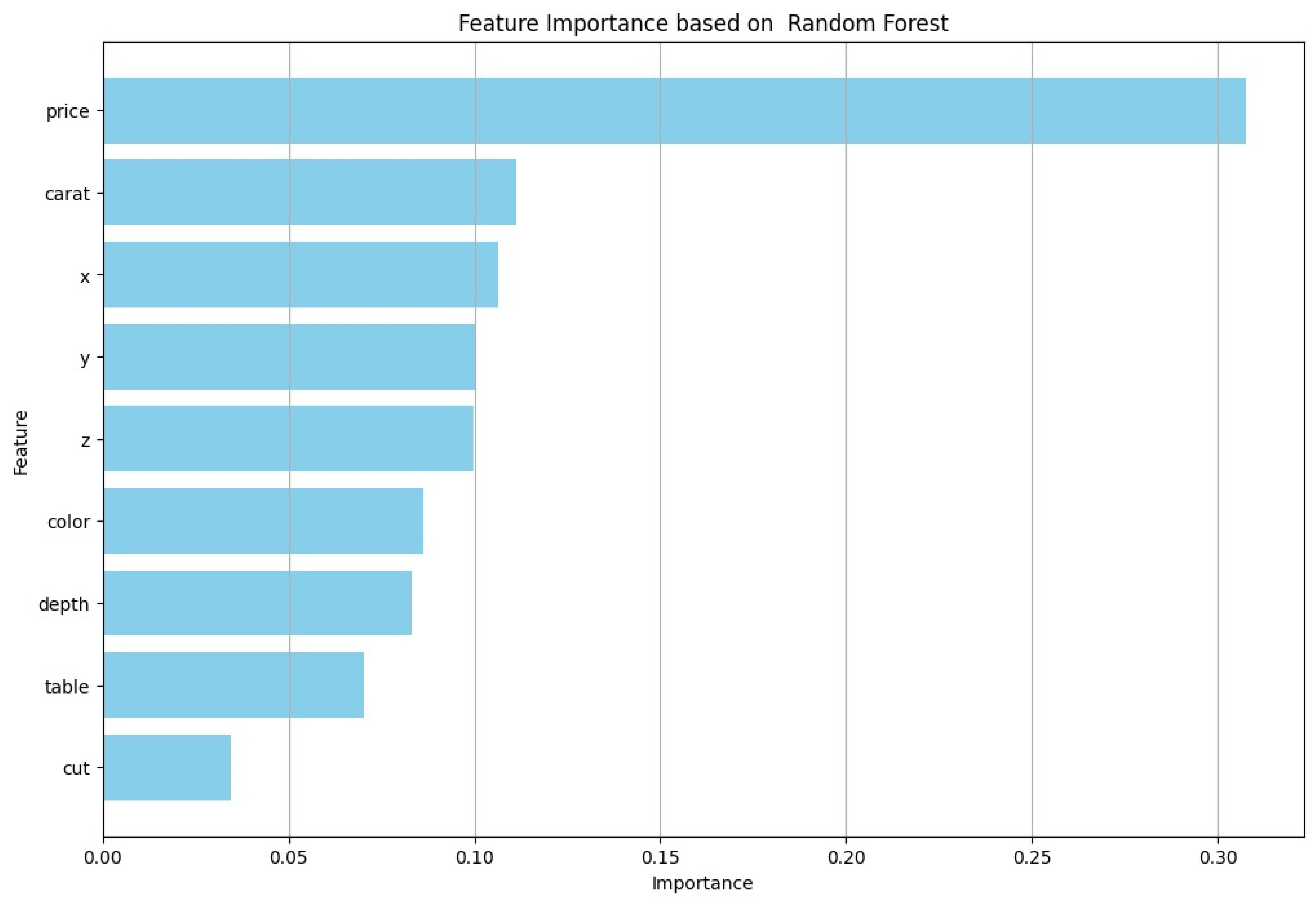
Dataset 3:

Feature Importance



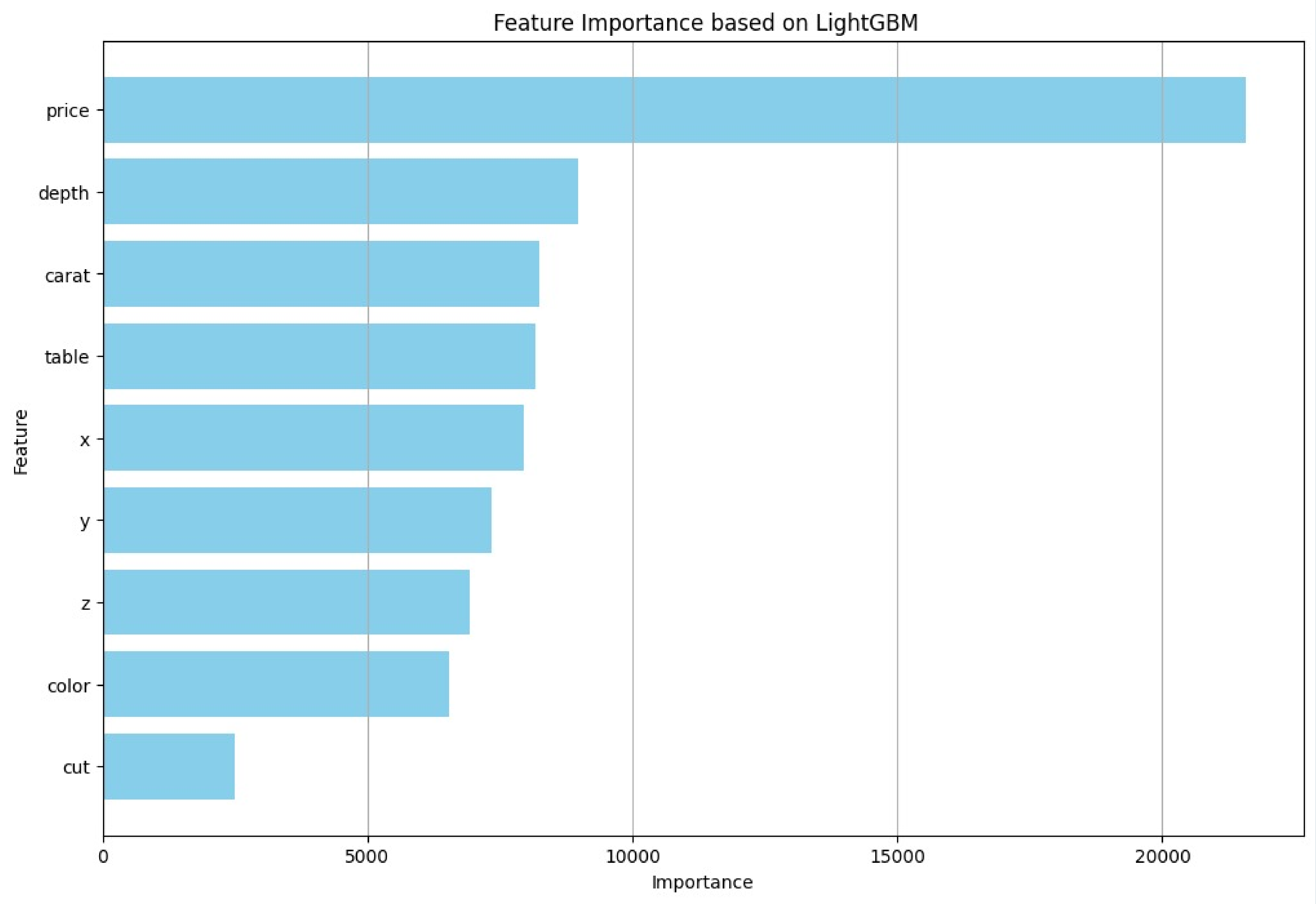
Dataset 3:

Feature Importance



Dataset 3:

Feature Importance



Dataset-3 Results :

Light GBM

Balanced Accuracy Score:
0.7615467530137565

Classification Report:

	precision	recall	f1-score
0	0.99	0.99	0.99
1	0.89	0.92	0.91
2	0.65	0.70	0.67
3	0.83	0.85	0.84
4	0.63	0.55	0.59
5	0.61	0.59	0.60
6	0.76	0.77	0.77
7	0.72	0.71	0.71
accuracy			0.76
macro avg	0.76	0.76	0.76
weighted avg	0.76	0.76	0.76

'n_estimators': 200, 'max_depth': 8, 'learning_rate': 0.1

Random Forest

Accuracy: 0.78
Classification Report:

	precision	recall	f1-score
I1	0.98	0.99	0.99
IF	0.91	0.93	0.92
SI1	0.64	0.67	0.66
SI2	0.81	0.84	0.82
VS1	0.66	0.63	0.64
VS2	0.64	0.60	0.62
VVS1	0.81	0.82	0.81
VVS2	0.77	0.76	0.76
accuracy			0.78
macro avg	0.78	0.78	0.78
weighted avg	0.78	0.78	0.78

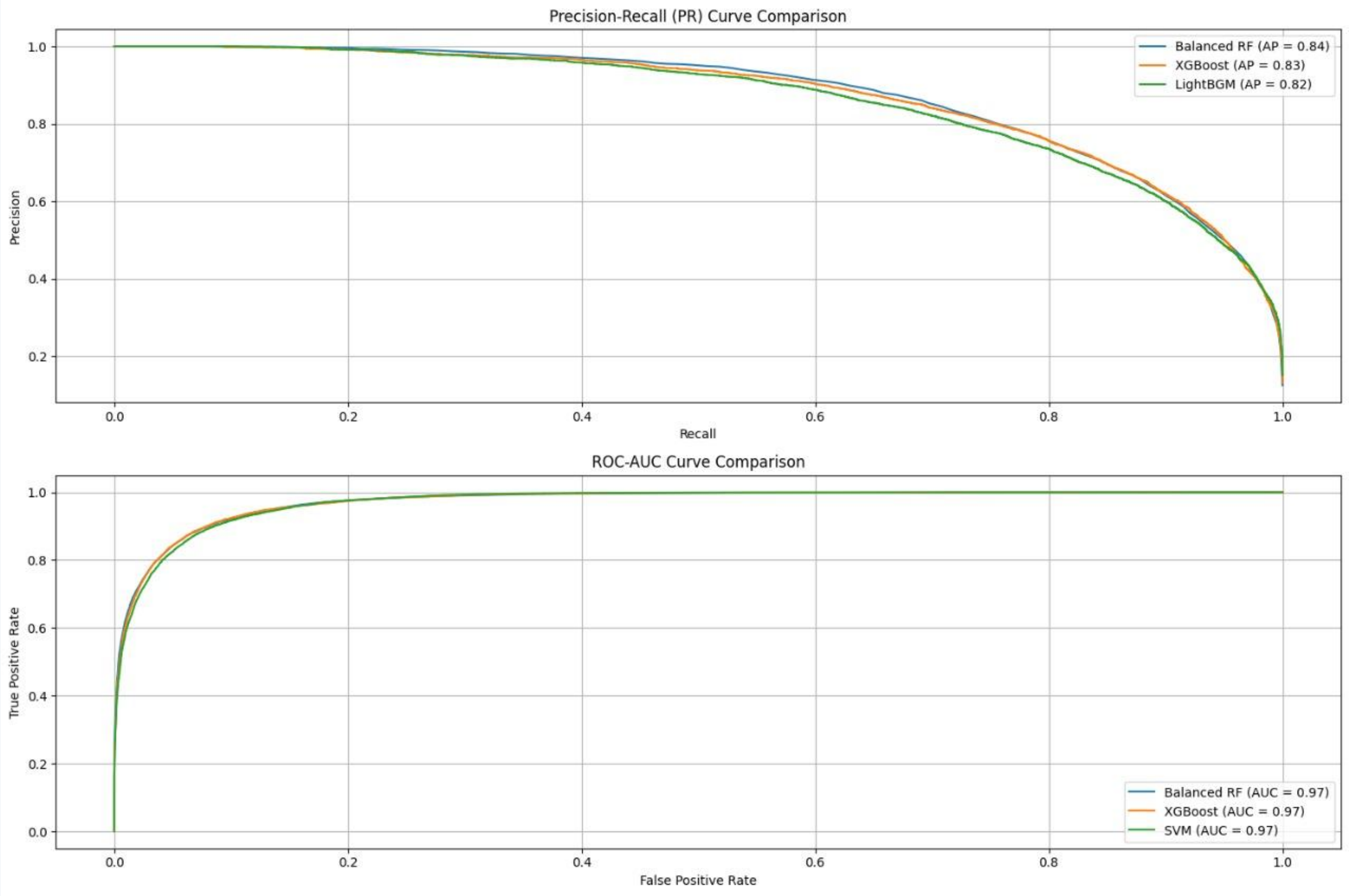
XGBoost

Accuracy: 0.7540422294084078
Classification Report:

	precision	recall	f1-score
0	0.98	0.98	0.98
1	0.89	0.90	0.89
2	0.63	0.68	0.65
3	0.82	0.83	0.82
4	0.62	0.59	0.60
5	0.61	0.58	0.59
6	0.76	0.77	0.77
7	0.72	0.70	0.71
accuracy			0.75
macro avg	0.75	0.75	0.75
weighted avg	0.75	0.75	0.75

{'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 200}

Dataset 3: PR & ROC Curve Comparision



Thank you