14th June,2024

INFOSYS

Telecom Churn Prediction

Infosys Springboard

Name:Bommu Bhargav

Agenda

GitHub:

About Telecom Churn Predict

"Telecom Churn Prediction" aims to empower telecom companies with actionable insights and tools to reduce churn, retain customers, and improve overall business. performance in a highly competitive telecommunications market.

Our Mission and Vision

Mission

Telecom churn prediction. contributes to the overall success and competitiveness of telecom companies in a rapidly evolving market landscape.

Vision

Telecom churn prediction can drive significant improvements in customer satisfaction, business. performance, and competitiveness within the telecommunications sector.

Our Goals

Minimize Churn Rate

Maximize Customer Retention

Optimize Business Performance

Our Milestones

01 Data gathering & understanding the dataset.

02 Data Cleaning and Pre-processing.

03 Machine Learning model building.

04 Hyperparameter Tuning, SVM & Confusion Matrix

Challenges:

Data Quality and Integration: Telecom companies often have vast amounts of data stored across multiple systems,

including customer demographics, call logs, usage patterns, and billing information. Integrating and cleansing this data for analysis can be challenging, and poor data quality can lead to inaccurate predictions.

Imbalanced Data: Telecom datasets are often imbalanced, with a small percentage of customers churning compared to those who remain. Imbalanced data can bias predictive models and lead to inaccurate churn predictions. Dynamic Customer Behavior: Customer behavior and preferences evolve over time, making it challenging to build

accurate predictive models that adapt to changing patterns and trends. Scalability: Telecom companies serve large customer bases, and churn prediction systems must be scalable to handle massive volumes of data and real-time predictions.

Real-Time Prediction: Implementing real-time churn prediction systems that can provide timely interventions to prevent churn is challenging due to the need for fast data processing and decision-making.

Data Cleaning & Pre- Processing

Convert datatypes of variables which are misclassified
ensures accurate analysis and efficient processing.

Removing duplicate records
ensure accuracy, maintain integrity, and optimize efficiency in data management and analysis.

Removing unique value variables
necessary to avoid redundancy, improve efficiency, and ensure meaningful analysis,

Removing Zero variance variables
necessary to eliminate redundant information, improve model stability, and simplify interpretation.

Data Cleaning & Pre- Processing

Outlier Treatment
necessary to maintain data integrity, improve model performance, and ensure robust and interpretable analyses.

Missing value Treatment.
essential for maintaining data integrity, improving model performance, and ensuring accurate and statistically valid analyses.

Removing the highly correlated variables.
necessary to reduce redundancy, improve model stability, and enhance efficiency in model training and interpretation.

Multicollinearity (VIF>5).
crucial to ensure accurate, stable, and interpretable models.

Machine Learning Model Building

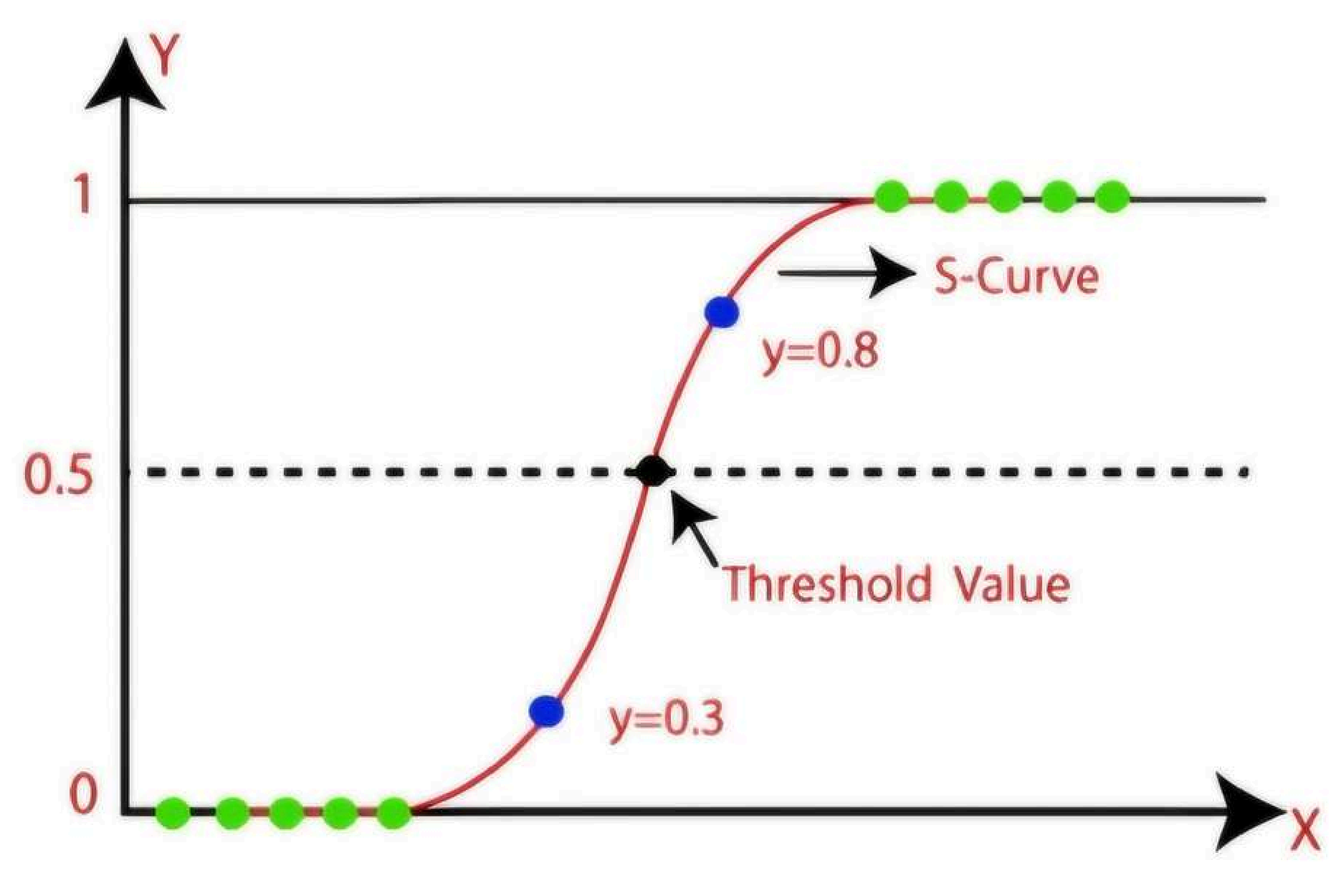
- ► Logistic Regression
- ►Decision Tree
- ►Random Forest

Logistic Regression

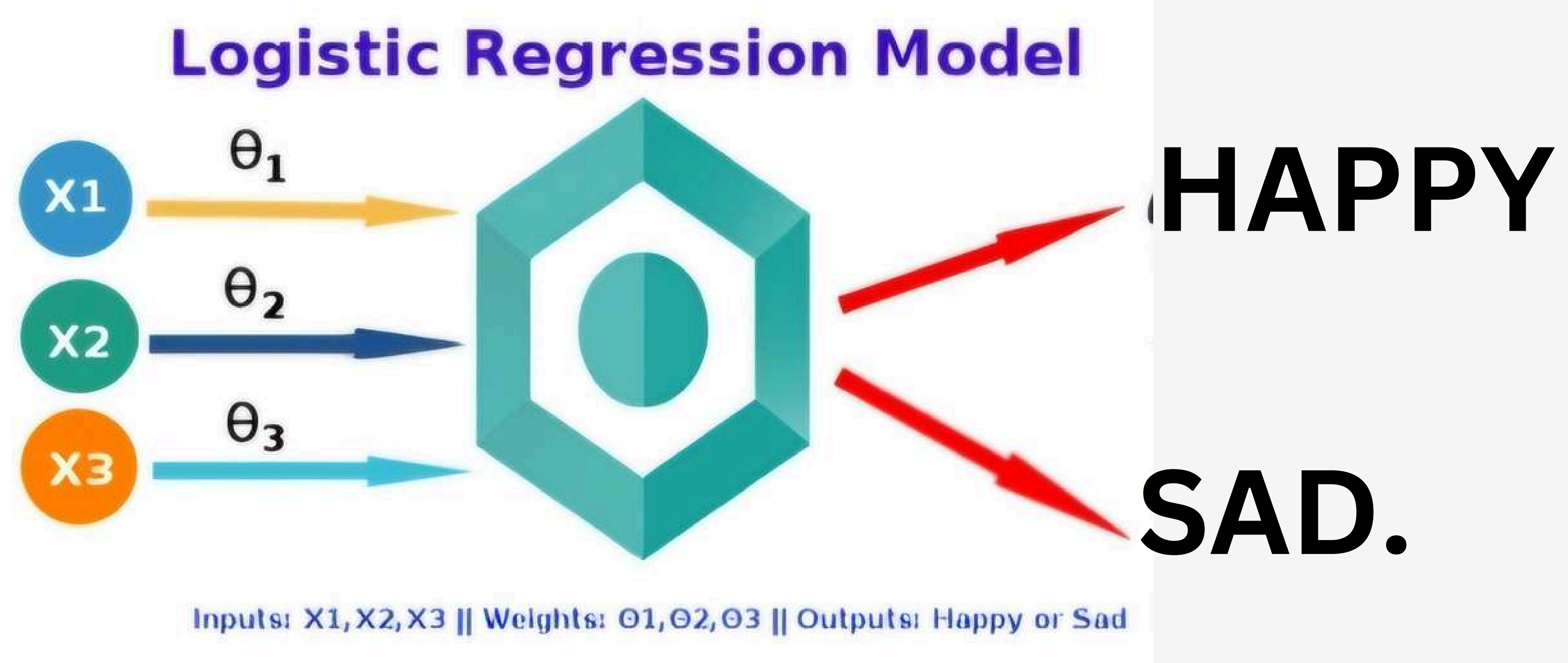
Logistic regression is a type of supervised machine learning algorithm used for binary classification problems, where the target variable is a binary outcome (0 or 1). It's an extension of linear regression, but instead of predicting a continuous value, it predicts the probability of the binary outcome.

Logistic regression is crucial for predicting binary outcomes, offering interpretable results, probabilistic predictions, efficiency, and serving as a foundation for more complex models.

Pictures:

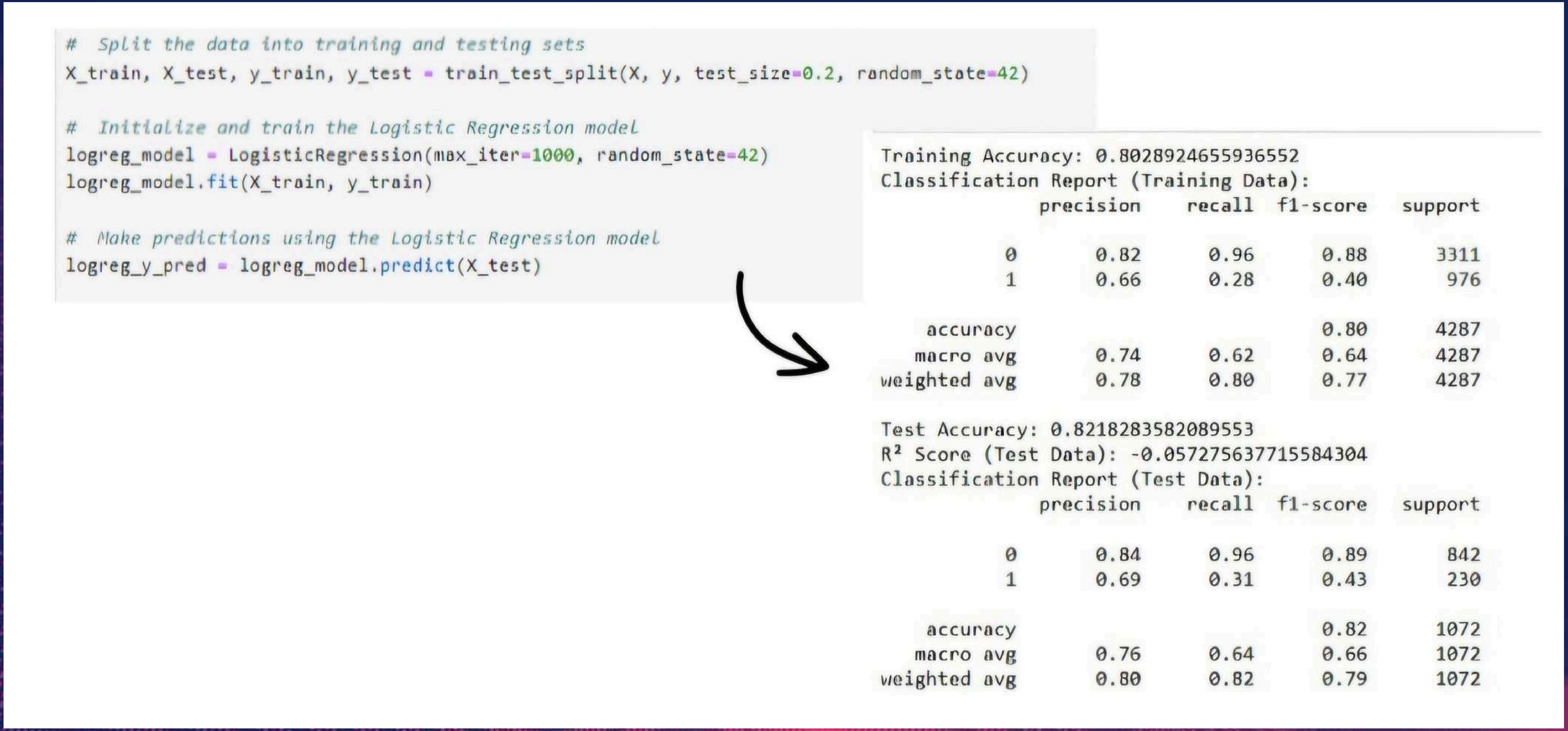


Y
1
S-Curve
y=0.8
0.5
Threshold Value
y=0.3
0
X



Logistic Regression Model
X1
X2
X3
θ_1
θ_2
θ_3
HAPPY
SAD.
Inputs: X1, X2, X3 || Weights: Θ1, Θ2, Θ3 || Outputs: Happy or Sad

CODE SNIPPET:



```python
# 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)

# Initialize and train the Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000, random_state=42)
logreg_model.fit(X_train, y_train)

# Make predictions using the Logistic Regression model
logreg_y_pred = logreg_model.predict(X_test)
```

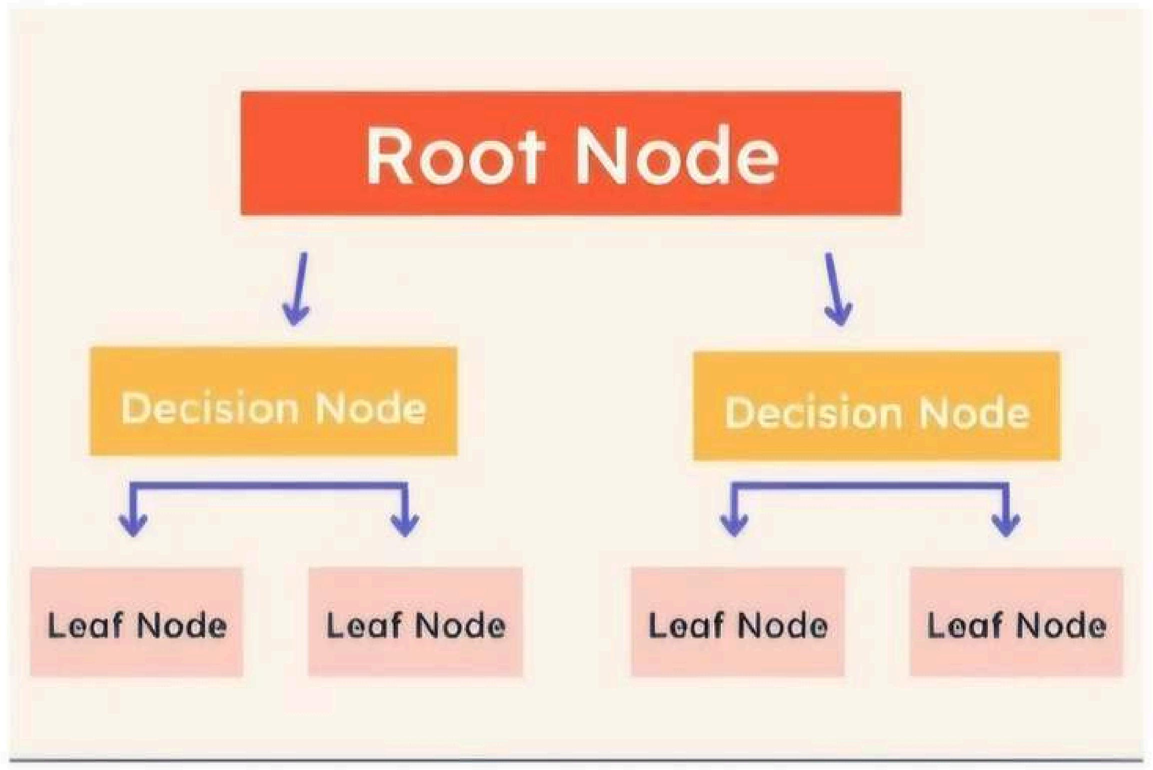
```
Training Accuracy: 0.8028924655936552
Classification Report (Training Data):
              precision    recall  f1-score   support

           0       0.82      0.96      0.88      3311
           1       0.66      0.28      0.40       976

    accuracy                           0.80      4287
   macro avg       0.74      0.62      0.64      4287
weighted avg       0.78      0.80      0.77      4287

Test Accuracy: 0.8218283582089553
R² Score (Test Data): -0.057275637715584304
Classification Report (Test Data):
              precision    recall  f1-score   support

           0       0.84      0.96      0.89       842
           1       0.69      0.31      0.43       230

    accuracy                           0.82      1072
   macro avg       0.76      0.64      0.66      1072
weighted avg       0.80      0.82      0.79      1072
```

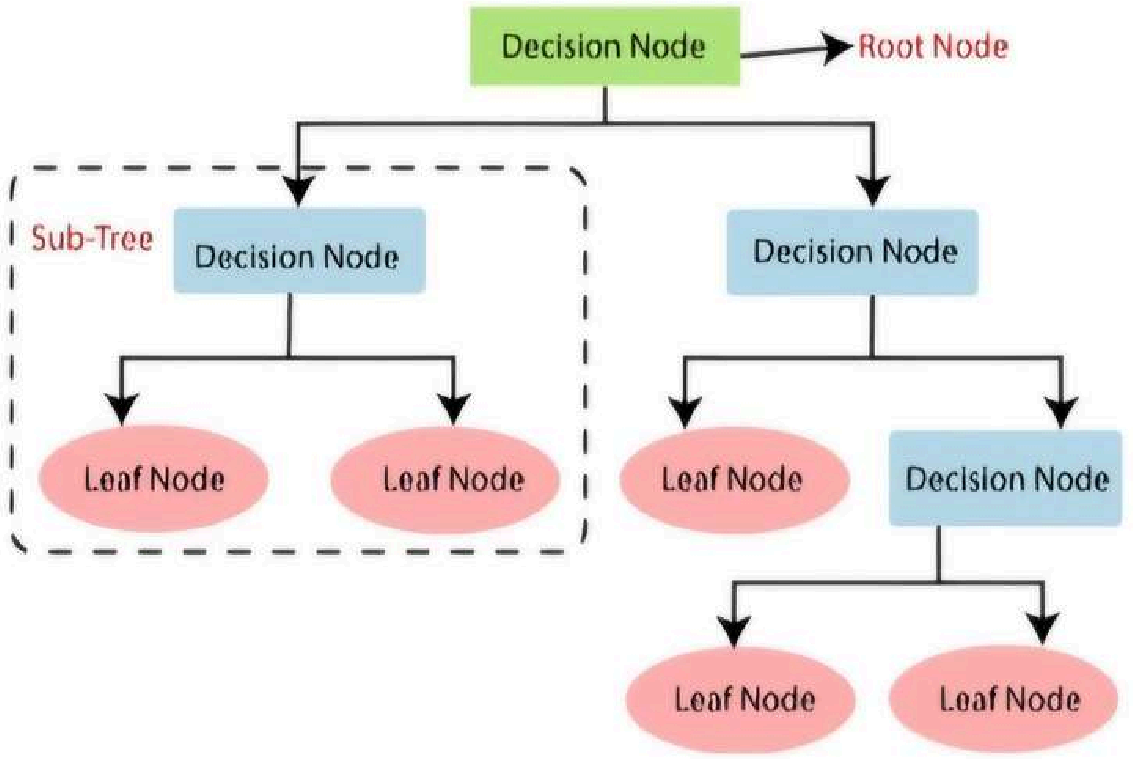
Decision Tree

Decision trees in machine learning are a supervised learning algorithm that enables developers to analyze the possible consequences of a decision and predict outcomes for future data. A decision tree is a tree-like model that starts at the root and branches out to demonstrate various. outcomes.

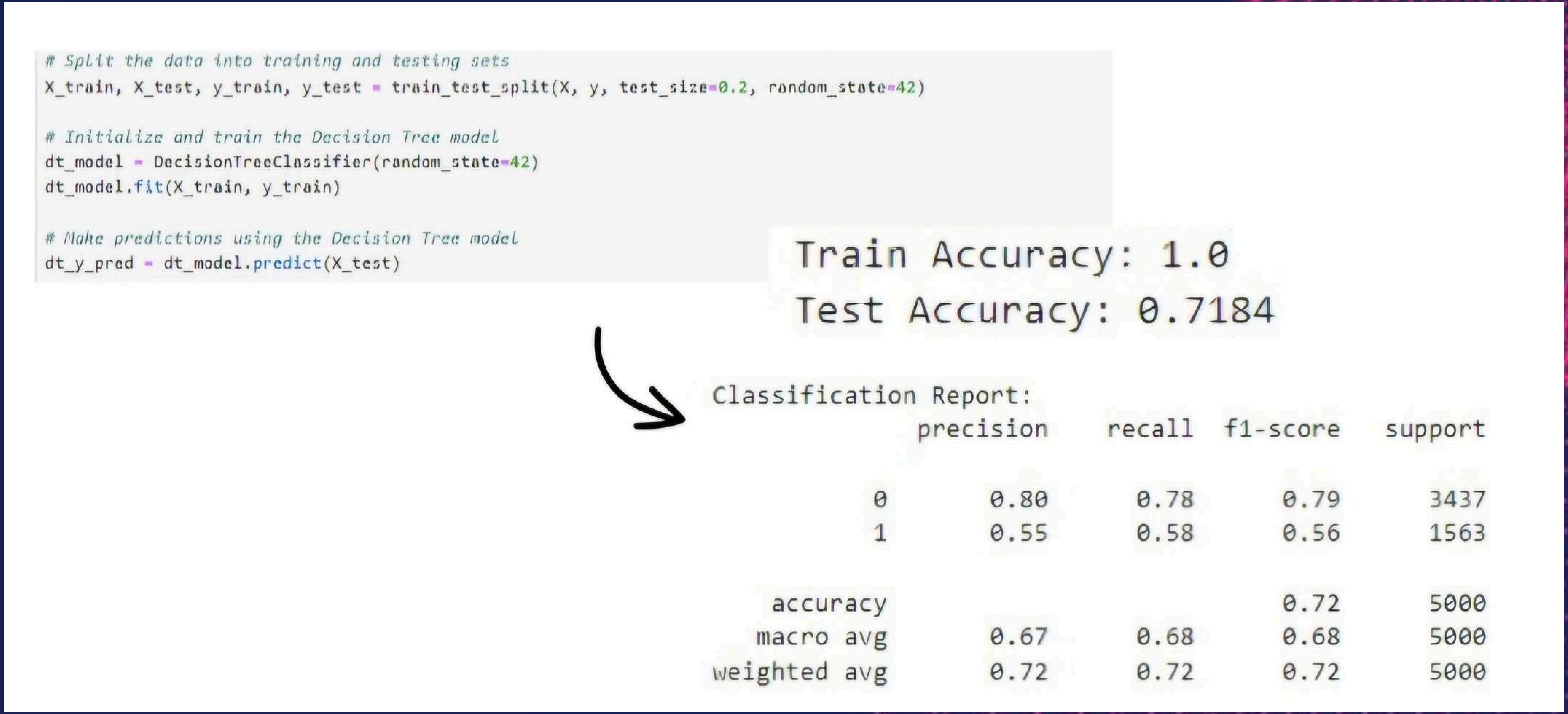
Decision trees are crucial for their simplicity, ability to handle non-linear relationships, feature importance identification, and robustness to outliers, applicable to both classification and regression tasks.

Pictures


Root Node
Decision Node
Decision Node
Leaf Node
Leaf Node
Leaf Node
Leaf Node


Decision Node
Root Node
Sub-Tree
Decision Node
Decision Node
Leaf Node
Leaf Node
Leaf Node
Decision Node
Leaf Node
Leaf Node

CODE SNIPPET:



```
# 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)

# Initialize and train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Make predictions using the Decision Tree model
dt_y_pred = dt_model.predict(X_test)
```

```
Train Accuracy: 1.0
Test Accuracy: 0.7184
```

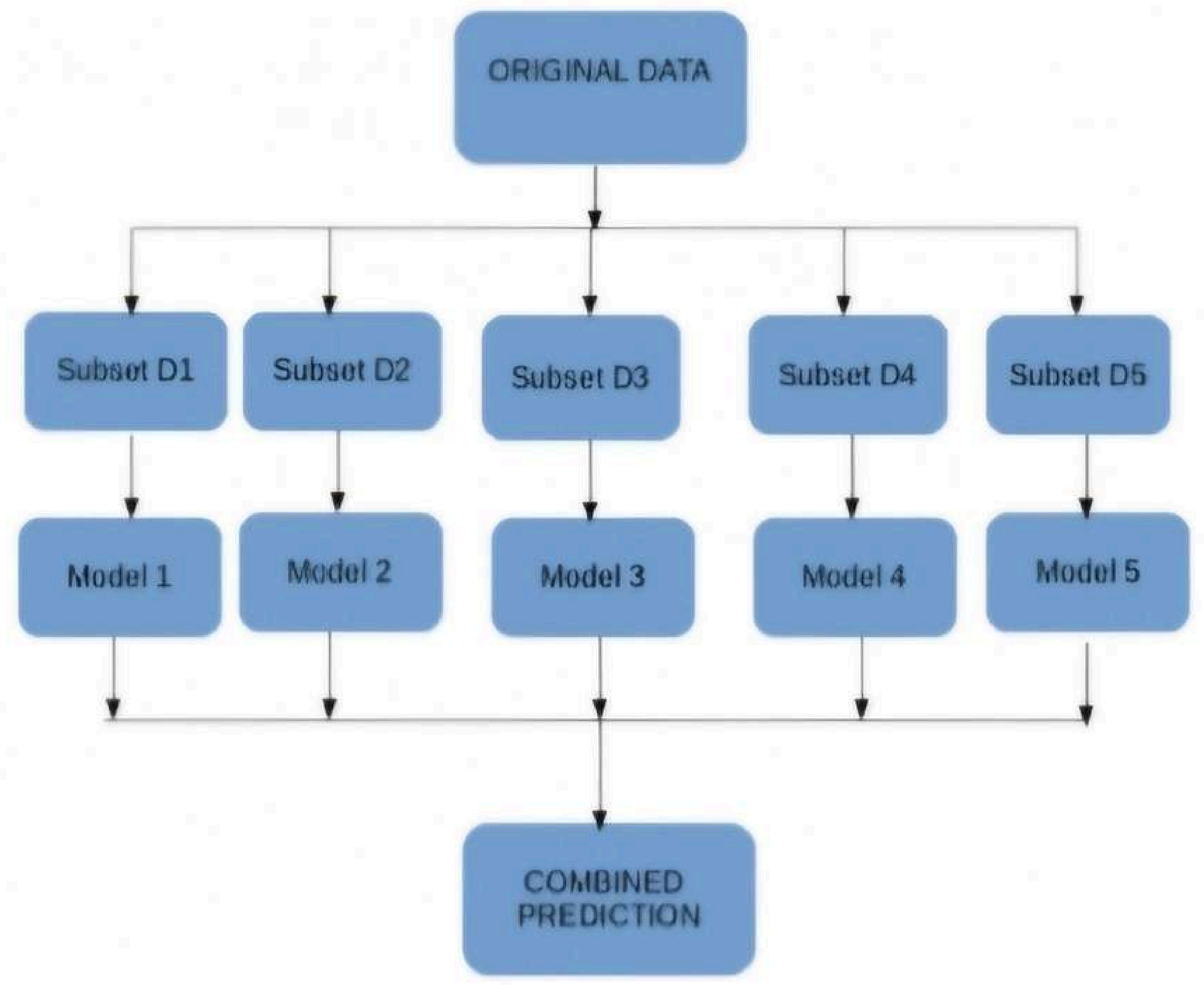
```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.78      0.79      3437
           1       0.55      0.58      0.56      1563

    accuracy                           0.72      5000
   macro avg       0.67      0.68      0.68      5000
weighted avg       0.72      0.72      0.72      5000
```

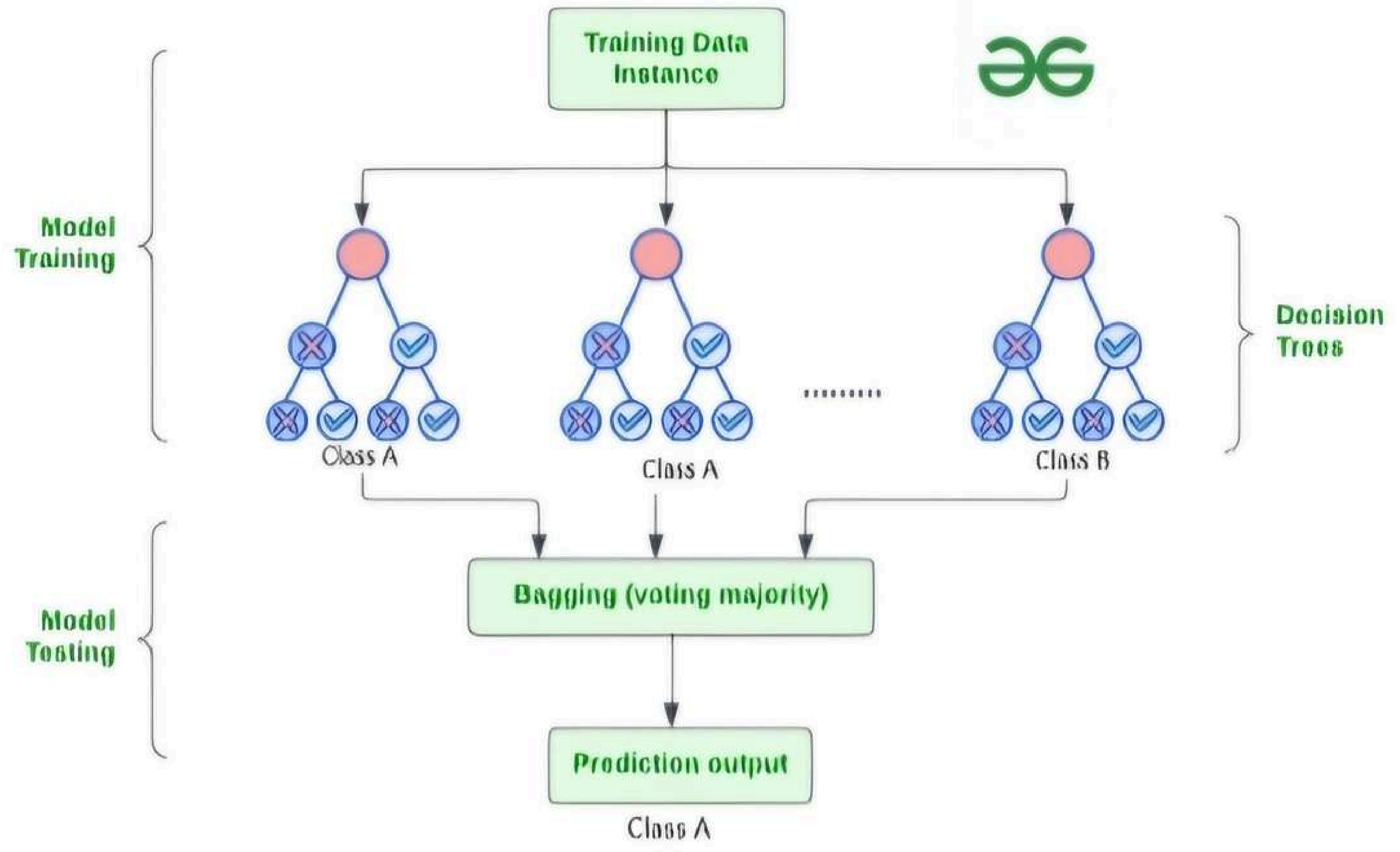
Random Forest

Random Forest is a supervised learning algorithm that combines multiple decision trees to improve the accuracy and robustness of predictions. It is a popular ensemble learning method that is widely used in classification and regression tasks.

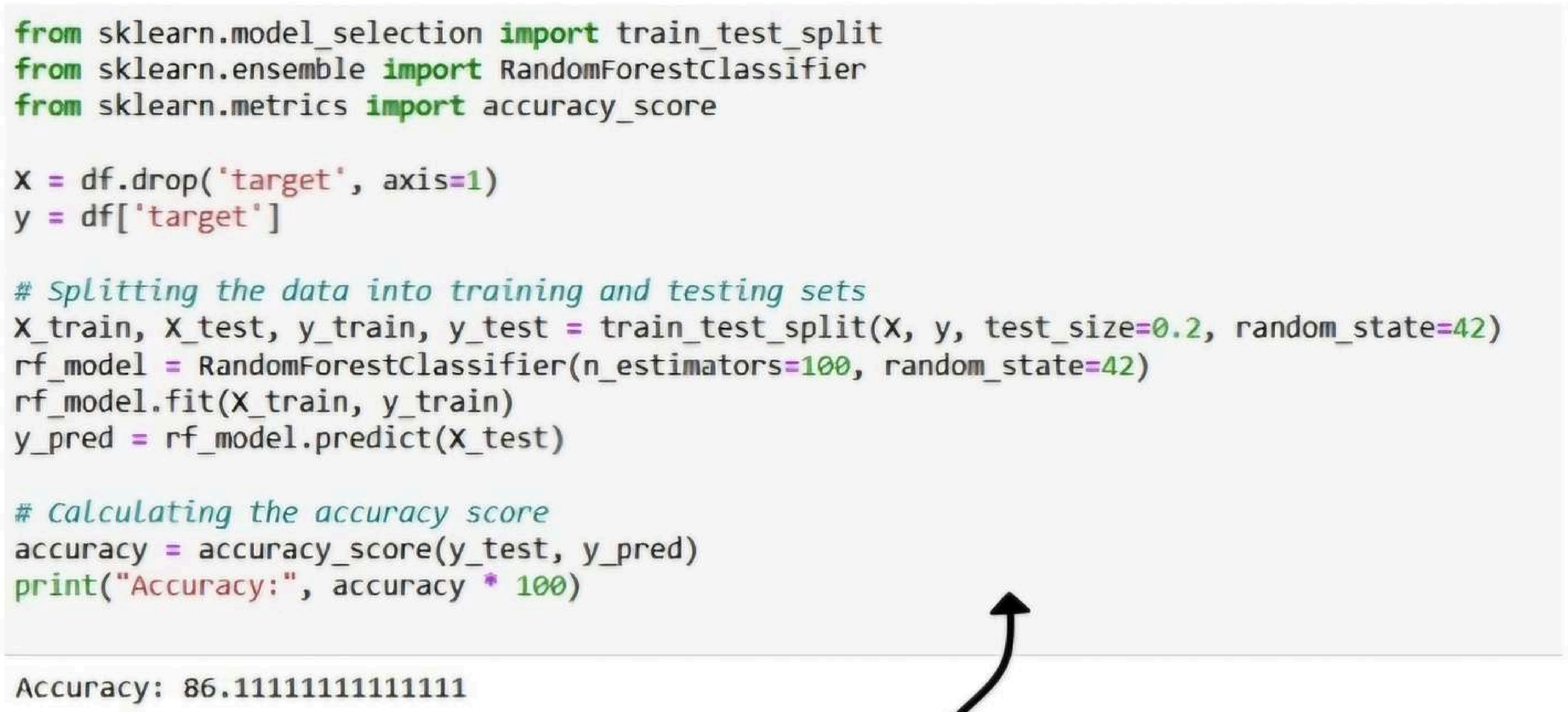
Random Forests enhance accuracy and robustness through ensemble learning, resist overfitting, highlight feature importance, handle missing values, scale well with large datasets, and are versatile for both classification and regression.

Pictures:


ORIGINAL DATA
Subset D1
Subset D2
Subset D3
Subset D4
Subset D5
Model 1
Model 2
Model 3
Model 4
Model 5
COMBINED PREDICTION

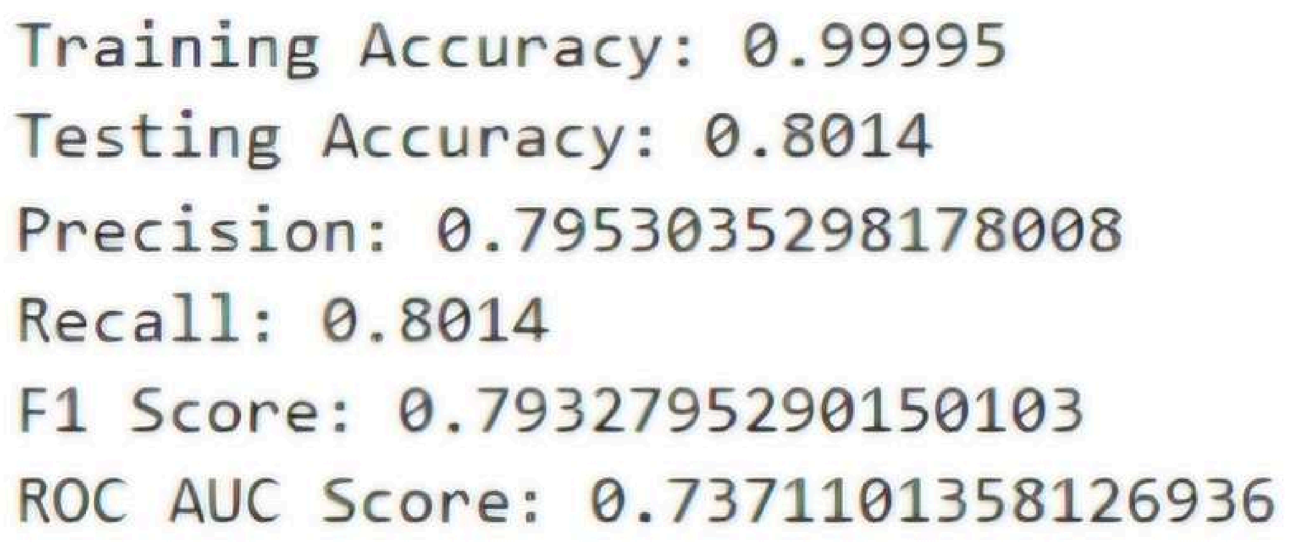

Training Data Instance
Model Training
Decision Trees
Class A
Class A
Class B
Bagging (voting majority)
Model Testing
Prediction output
Class A

CODE SNIPPET



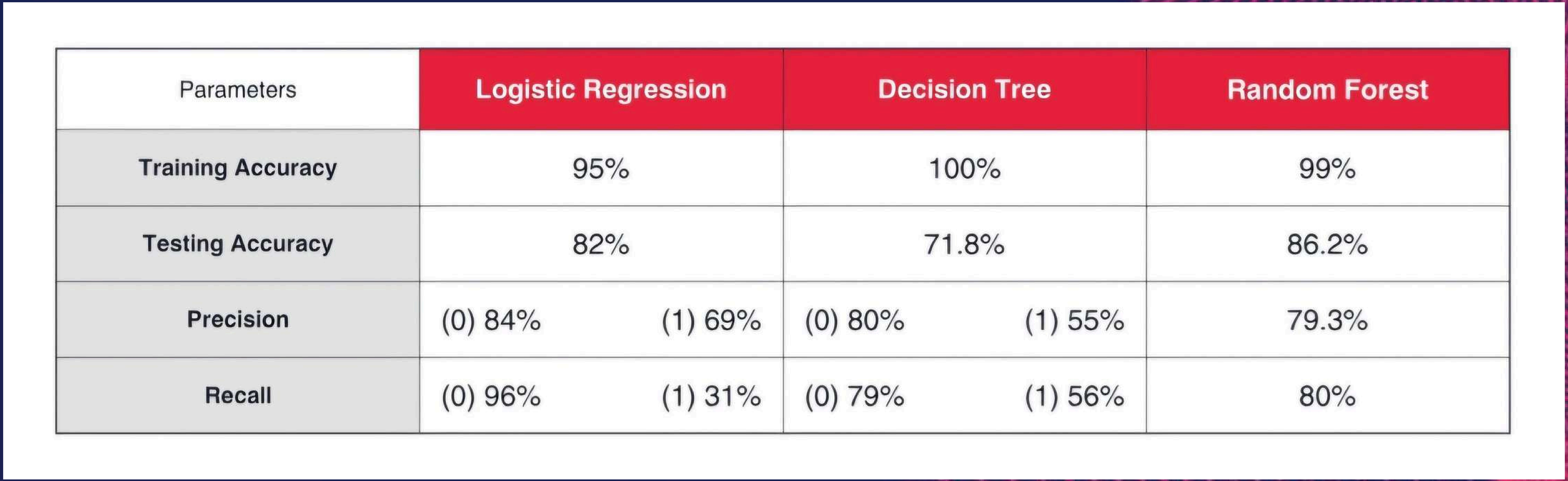
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X = df.drop('target', axis=1)
y = df['target']

# Splitting 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)
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)

# Calculating the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100)
```

```
Accuracy: 86.11111111111111
```



```
Training Accuracy: 0.99995
Testing Accuracy: 0.8014
Precision: 0.7953035298178008
Recall: 0.8014
F1 Score: 0.7932795290150103
ROC AUC Score: 0.7371101358126936
```

Key Difference:



Parameters	Logistic Regression	Decision Tree	Random Forest
Training Accuracy	95%	100%	99%
Testing Accuracy	82%	71.8%	86.2%
Precision	(0) 84% (1) 69%	(0) 80% (1) 55%	79.3%
Recall	(0) 96% (1) 31%	(0) 79% (1) 56%	80%

Why Random Forest ?

Random Forests offer several advantages: they deliver high accuracy, resist overfitting, highlight feature importance, and effectively handle noise and missing values. Additionally, they are versatile and scalable, suitable for both classification and regression tasks. After evaluating all three models, we selected Random Forest due to its superior performance and minimal gap between training and testing accuracy.

Hyperparameter Tuning

What is Hyperparameter Tuning?

Hyperparameter Tuning is the process of finding the best settings for a machine learning model to improve its performance. This involves adjusting parameters that control the learning process, like the number of trees in a Random Forest or the learning rate in a neural network. By testing different combinations of these parameters, we identify the ones that yield the best results for our specific data.

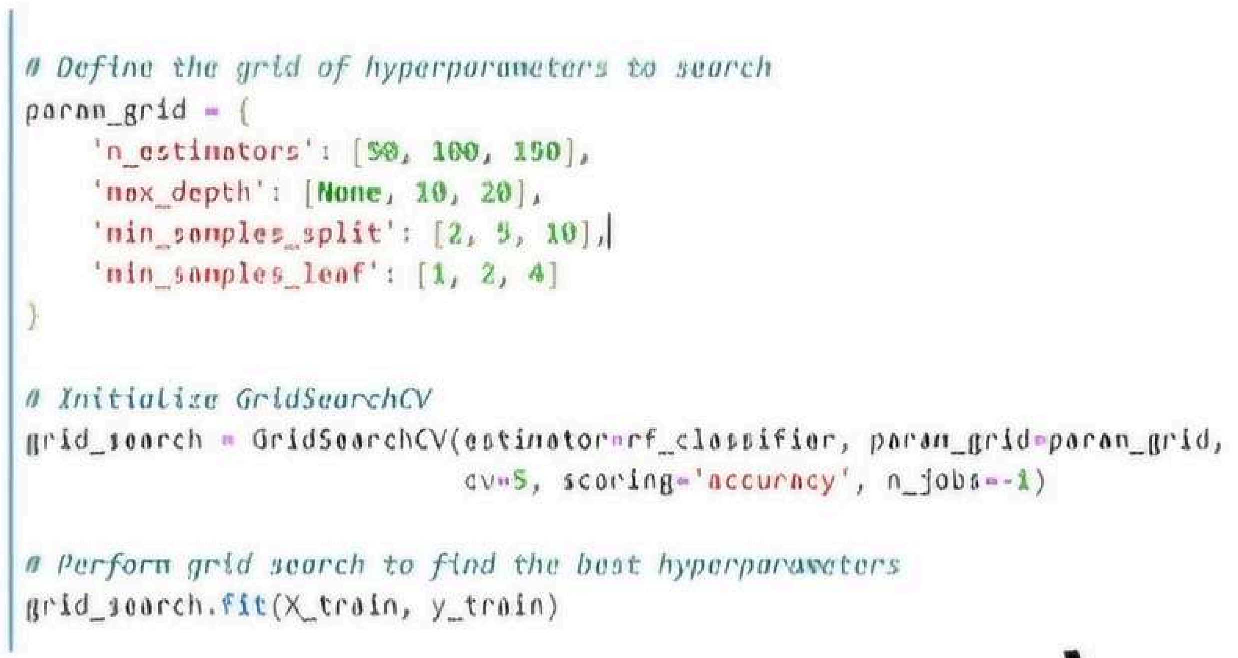
Importance :

Improves Model Performance: Proper tuning can significantly enhance the predictive accuracy and generalization of a model.

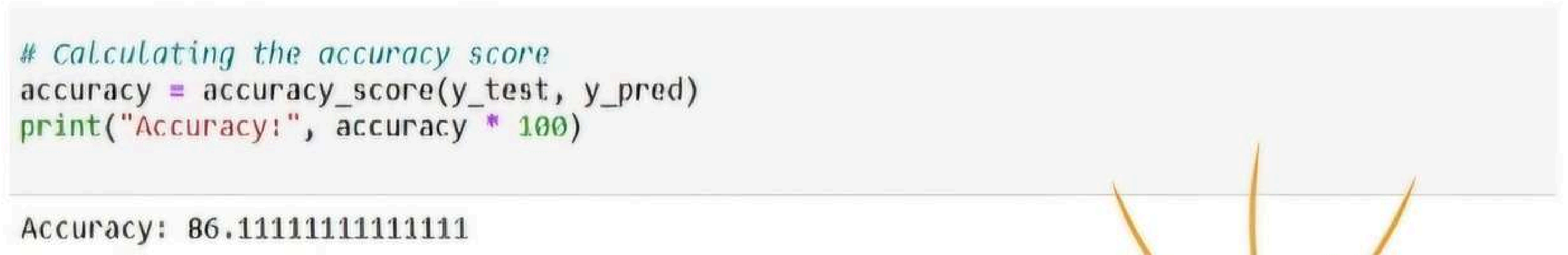
Reduces Overfitting/Underfitting: Helps in finding the balance between bias and variance, ensuring the model performs well on unseen data.

Optimizes Computational Resources: Efficient tuning can lead to faster and more efficient model training and deployment.

CODE SNIPPET

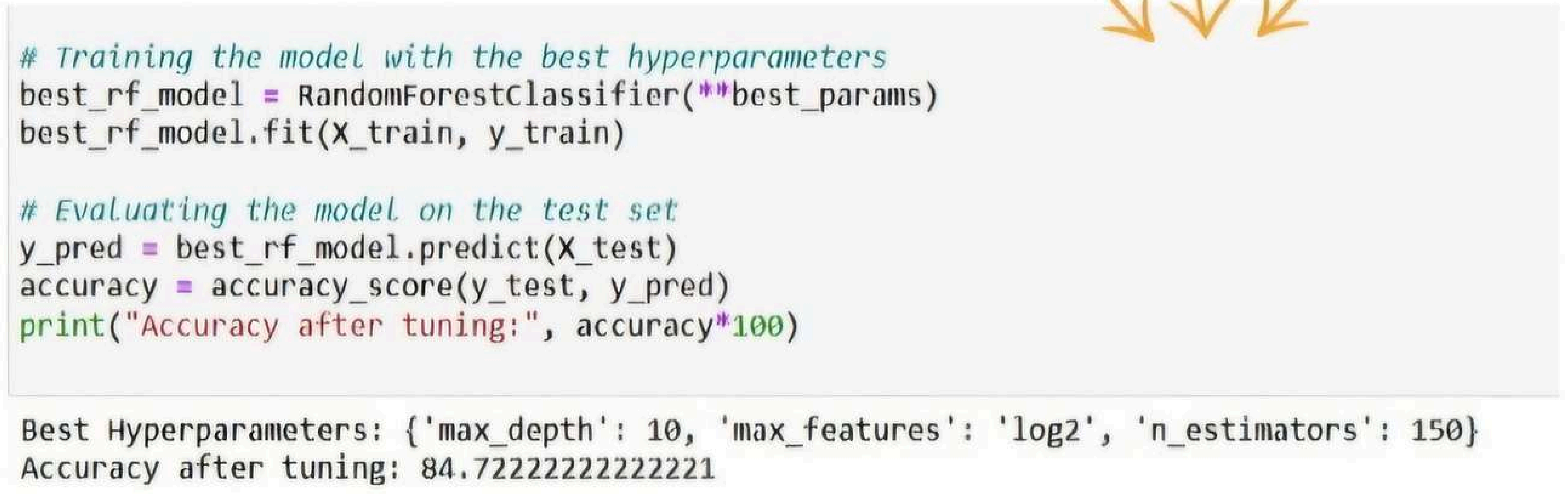


```
# Define the grid of hyperparameters to search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid,
                           cv=5, scoring='accuracy', n_jobs=-1)

# Perform grid search to find the best hyperparameters
grid_search.fit(X_train, y_train)
```



```
# Calculating the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy * 100)
```

```
Accuracy: 86.11111111111111
```



```
# Training the model with the best hyperparameters
best_rf_model = RandomForestClassifier(**best_params)
best_rf_model.fit(X_train, y_train)

# Evaluating the model on the test set
y_pred = best_rf_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy after tuning:", accuracy*100)
```

```
Best Hyperparameters: {'max_depth': 10, 'max_features': 'log2', 'n_estimators': 150}
Accuracy after tuning: 84.72222222222221
```

Support Vector Machine

What is Support Vector Machine?

Support Vector Machine (SVM) is a powerful supervised learning algorithm used for classification and regression. It finds the optimal hyperplane that best separates classes by maximizing the margin between support vectors, SVM can handle linear and non- linear data using kernel tricks, making it suitable for high-dimensional datasets.

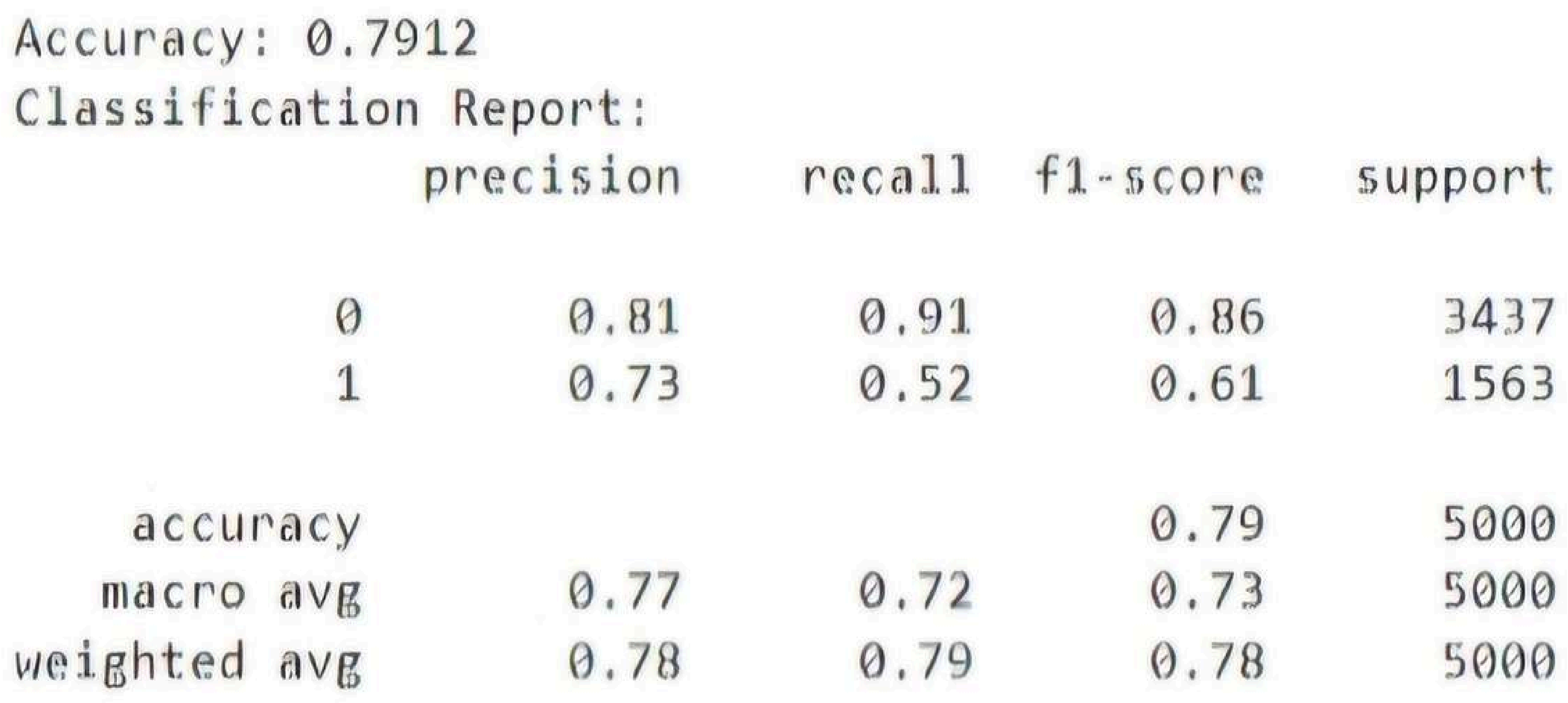
Importance:

• High-Dimensional Performance: SVMs are particularly effective in high- dimensional spaces, making them suitable for datasets with many features and complex relationships.

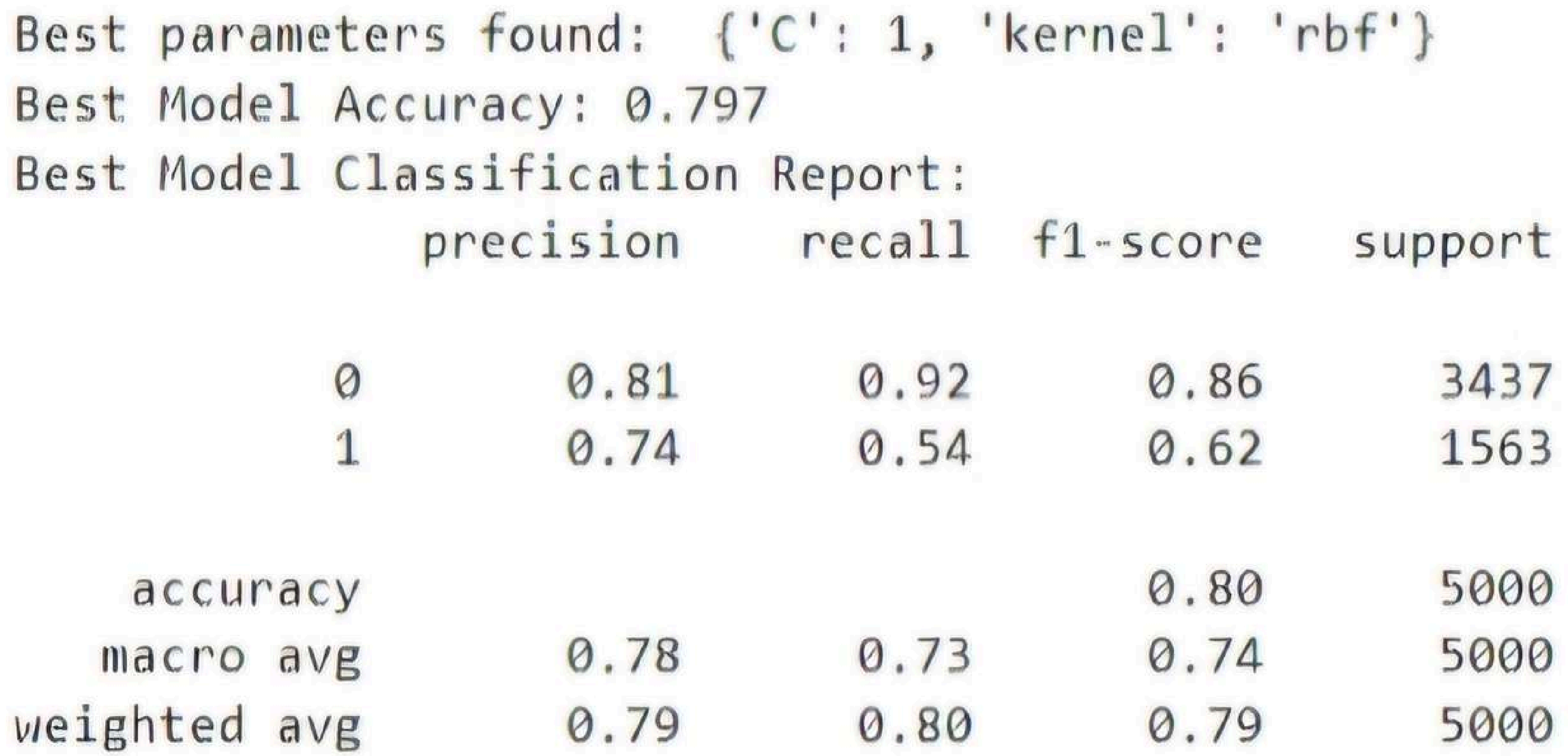
Robustness to Overfitting: By maximizing the margin between classes, SVMs reduce the risk of overfitting, ensuring better generalization to new, unseen data.

Versatility with Kernels: The kernel trick allows SVMs to handle non-linear relationships by mapping input features into higher-dimensional spaces, enabling them to adapt to a wide range of data types and structures.

CODE SNIPPET



```
Accuracy: 0.7912
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.91      0.86      3437
           1       0.73      0.52      0.61      1563

    accuracy                           0.79      5000
   macro avg       0.77      0.72      0.73      5000
weighted avg       0.78      0.79      0.78      5000
```



```
Best parameters found:  {'C': 1, 'kernel': 'rbf'}
Best Model Accuracy: 0.797
Best Model Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.92      0.86      3437
           1       0.74      0.54      0.62      1563

    accuracy                           0.80      5000
   macro avg       0.78      0.73      0.74      5000
weighted avg       0.79      0.80      0.79      5000
```

Confusion Matrix

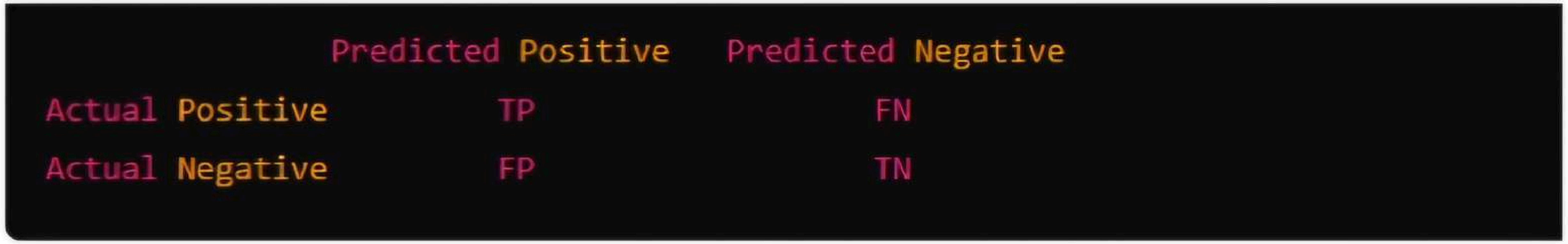
What is Confusion Matrix ?

A confusion matrix is a table that evaluates a classification model's performance by showing the counts of true positives, true negatives, false positives, and false negatives. It helps derive metrics like accuracy, precision, recall, and F1 score for a more detailed performance assessment.

Components:

- True Positives (TP): The number of instances correctly predicted as positive.
- True Negatives (TN): The number of instances correctly predicted as negative.
- False Positives (FP): The number of instances incorrectly predicted as positive (Type I error).
- False Negatives (FN): The number of instances incorrectly predicted as negative (Type II error).

Structure of a Confusion Matrix.



```
                 Predicted Positive   Predicted Negative
Actual Positive          TP                   FN
Actual Negative          FP                   TN
```

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

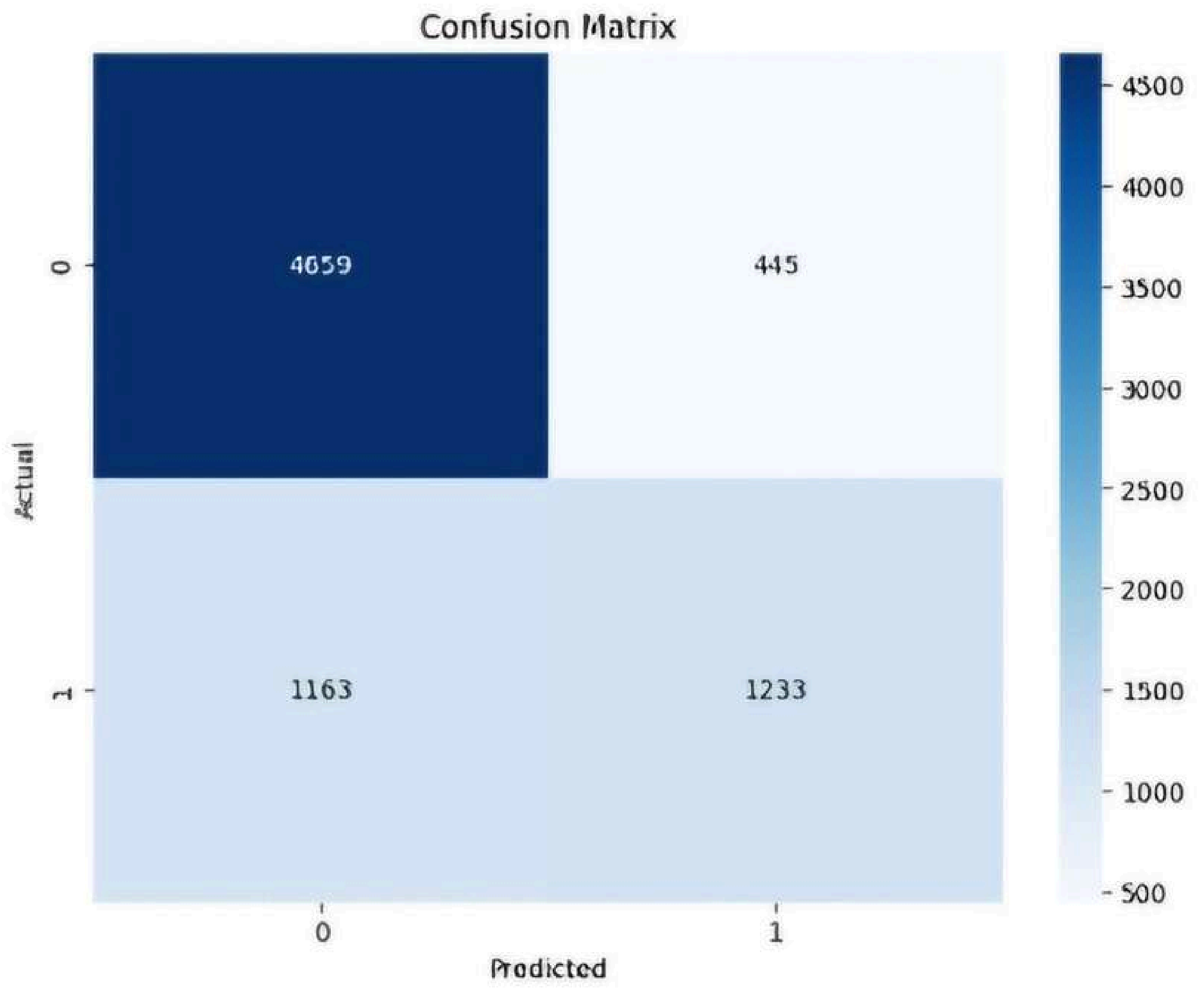
$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

CODE SNIPPET

```
Confusion Matrix:
[[4659  445]
 [1163 1233]]

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.91      0.85      5104
           1       0.73      0.51      0.61      2396

    accuracy                           0.79      7500
   macro avg       0.77      0.71      0.73      7500
weighted avg       0.78      0.79      0.77      7500
```



Conclusion:

• Improved Customer Retention: By accurately predicting which customers are likely to churn, telecom companies can implement targeted strategies to retain valuable customers, ensuring their satisfaction and loyalty.

•Cost Reduction: Preventing churn is more cost-effective than acquiring new customers. Churn prediction allows for efficient resource allocation and reduces costs associated with customer attrition.

•Enhanced Customer Experience: Understanding customer behavior and preferences enables telecom providers to offer personalized experiences, address concerns proactively, and significantly improve overall satisfaction

•Increased Revenue: Retaining existing customers and maximizing their lifetime value leads to steady revenue streams and sustainable business growth.

•Competitive Advantage: Effective churn prediction and management provide telecom companies with a competitive edge through superior service, tailored offers, and proactive retention initiatives.