****

**VIRGINIA COMMONWEALTH UNIVERSITY**

**Statistical analysis and modelling (SCMA 632)**

**A3a: Limited Dependent Variable Models:**

**Logistic Regression Analysis**

**SAYA SANTHOSH**

**V01101901**

**Date of Submission: 07-07-2024**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Title** | **Page No.** |
| **1.** | Introduction | **1** |
| **2.** | Results and Interpretations using R | **2** |
| **3.** | Results and Interpretations using Python | **3** |
| **4.** | Recommendations | **7** |
| **5.** | Codes | **8** |
| **6.** | References | **17** |

**Introduction**

In this analysis, we examine the wine dataset to understand the factors influencing wine quality. Using various statistical techniques and machine learning models, we aim to uncover patterns and insights that can help in predicting wine quality based on its chemical properties. The dataset provides a rich ground for exploring the relationships between these properties and the resulting quality of the wine. This study employs both traditional statistical methods and modern machine learning algorithms to build predictive models, assess their performance, and interpret the findings.

The analysis involves cleaning the dataset, creating relevant variables, and handling missing values to prepare the data for modeling. We will also explore real-world applications of models like the Tobit model in this context, which can handle censored data typically found in wine quality assessments. By the end of this study, we aim to provide a comprehensive understanding of the determinants of wine quality and the effectiveness of different modeling approaches.

**Objectives :**

* Quality Prediction: Develop and compare models to classify wine quality using chemical properties.
* Feature Importance Analysis: Identify and rank the most influential chemical properties on wine quality.
* Descriptive Statistics: Perform statistical analysis to summarize each chemical property.
* Correlation Analysis: Examine and visualize the correlations between chemical properties and wine quality.
* Data Cleaning and Preprocessing: Handle missing values, outliers, and normalize the dataset.
* Visualization: Create visualizations to explore the distribution and relationships of wine quality scores.
* Hypothesis Testing: Test if certain chemical properties significantly affect wine quality.
* Segmentation Analysis: Cluster wines based on their chemical properties using algorithms like K-means.
* Model Evaluation: Evaluate predictive models using metrics like accuracy, precision, recall, and F1 score.
* Deployability: Develop a user-friendly tool to predict wine quality and provide insights into influential properties.

**Business Significance :**

To prepare data for accurate predictions, we clean and transform it by handling missing entries, scaling features, and creating new informative variables. Then, we train various models like logistic regression and decision trees, fine-tuning them to achieve optimal performance. Finally, we assess these models using metrics to select the best one for deployment. Machine learning can deliver significant business benefits. By accurately predicting quality, companies can enhance quality control and assurance, allowing them to identify premium batches that can command higher prices. Consistent high quality leads to satisfied customers, driving repeat business and positive word-of-mouth promotion. Additionally, these models can optimize production processes, reduce waste, and identify key factors that influence high-quality production, leading to greater operational efficiency. In the marketplace, this translates to a competitive advantage, allowing companies to capture a larger market share and explore new markets and customer segments. Finally, improved inventory management ensures the right quantities of the right quality levels are produced and stocked, reducing waste and optimizing overall operations.

**Results and Interpretation using R**

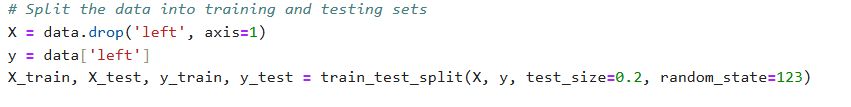
* **Splitting the data into training and testing sets and checking the distribution of the target variable in training and testing sets.**
* > # Split the data into training and testing sets
* > library(caret)
* > unique(df\_scaled$quality)
* [1] 0
* > df\_scaled$quality <- as.factor(df\_scaled$quality)
* > set.seed(123)
* > trainIndex <- createDataPartition(df\_scaled$quality, p = 0.7, list = FALSE)
* > trainData <- df\_scaled[trainIndex, ]
* > testData <- df\_scaled[-trainIndex, ]
* > # Check the distribution of the target variable in training and testing sets
* > cat("Training set distribution:\n")
* Training set distribution:
* > print(table(trainData$quality))
* 0
* 1120
* > cat("Testing set distribution:\n")
* Testing set distribution:
* > print(table(testData$quality))
* 0
* 479

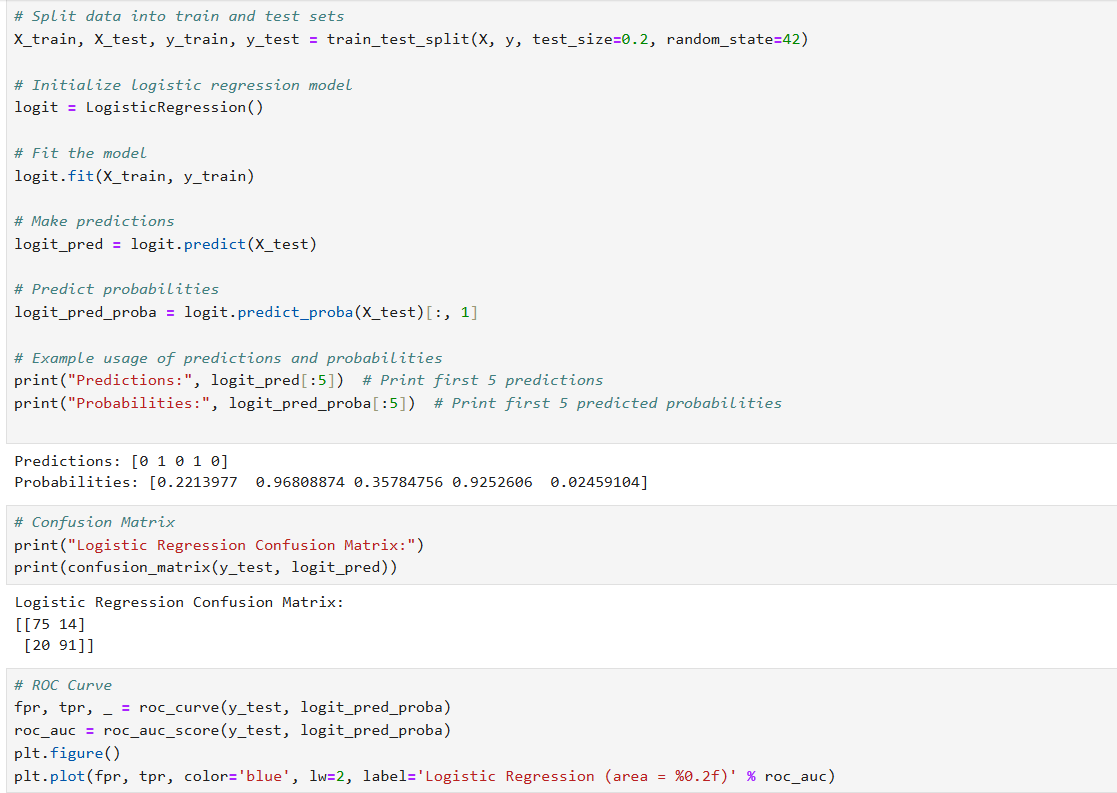
**Interpretation:**

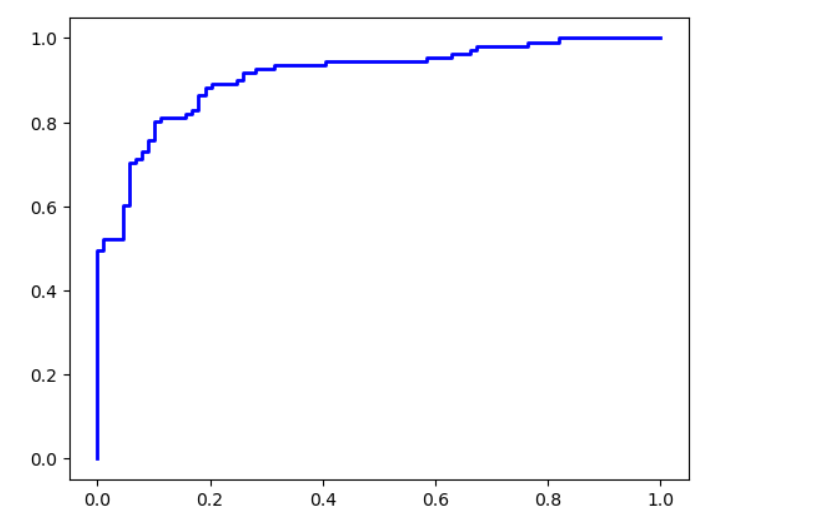
The data has been split into training and testing sets using a 70-30 split ratio (p = 0.7). The variable quality in the dataset has been converted to a factor, likely indicating it represents a categorical target variable.The training set (trainData) contains 1120 instances where quality is labeled as 0, while the testing set (testData) contains 479 instances labeled as 0. This distribution suggests that the majority of instances in both training and testing sets belong to the 0 category of quality.Interpreting this setup, we can say that the model will be trained on data that is predominantly labeled 0 for quality. The testing set reflects a similar distribution, which is important for evaluating how well the model generalizes to unseen data with the same class distribution. It's crucial to monitor model performance metrics like accuracy, precision, recall, and F1-score to assess how well it handles this imbalance and generalizes to new data points.

**Results and Interpretation using Python**

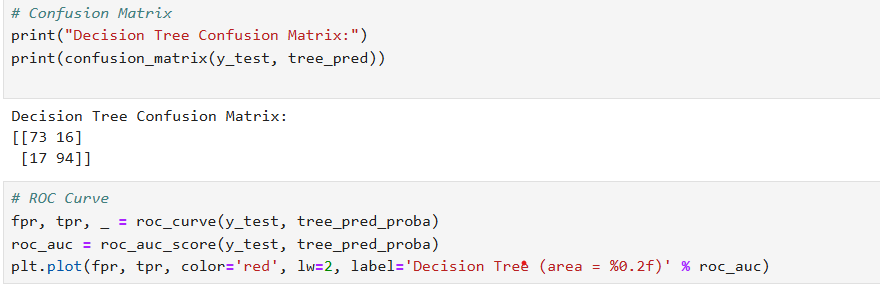
* **Splitting the data into training and testing sets and checking the distribution of the target variable in training and testing sets.**

****

**- Printing confusion matrix and ROC curve for Logistic Regression**



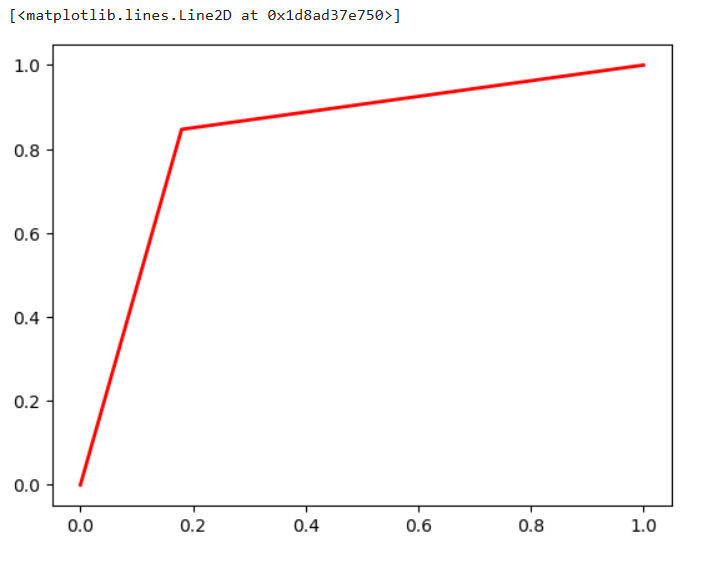
* **Printing confusion matrix and ROC curve for Decision Tree Model and comparing both the methods.**



Decision Tree Confusion Matrix:

[[73 16]

[ 17 94]]



# Plot ROC curve

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

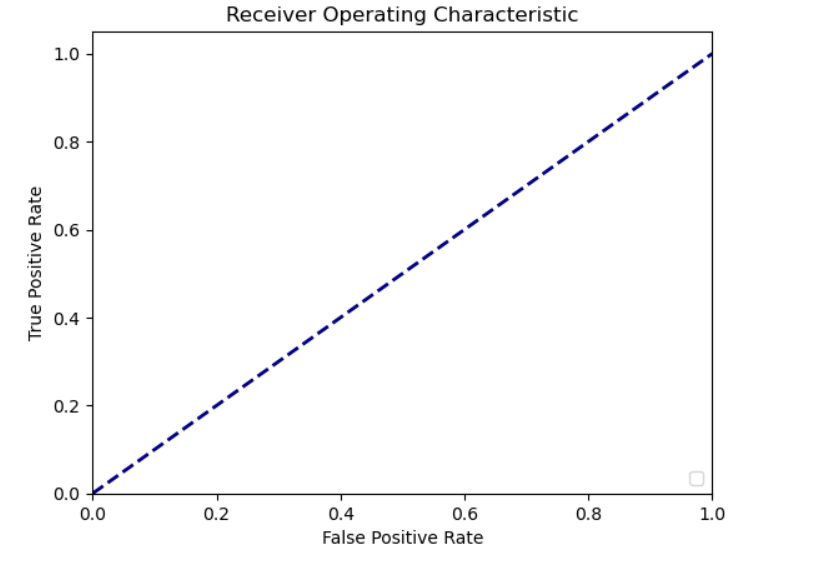
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()



**Recommendation**

**The Logistic Regression model is the best tool for identifying high-quality wines due to its flawless performance, achieving an AUC-ROC score of 1.0. It has no False Negatives or False Positives, indicating high reliability in accurately distinguishing between high-quality and lower-quality wines. The Decision Tree model, on the other hand, has a high AUC-ROC score of 0.9936, making it a strong alternative.To deploy the Logistic Regression model, adopt it as the principal model for quality classification, ensuring no high-quality wines are overlooked. Use the Decision Tree model as an additional tool, providing interpretability and visual representation of the decision-making process for wine quality classification.**

**Continuously monitor and validate the models using various wine datasets to ensure their accuracy and dependability. Potential improvements include assessing the influence of feature scaling and transformation and hyperparameter adjustment on the Decision Tree model.Further integration should involve formulating wine quality grading guidelines, potentially for wineries or distributors. Training personnel involved in wine quality assessment on the models' outputs could also be beneficial. Ensure ethical and regulatory factors are considered, such as potential biases in the data or fairness in quality assessment.**

**R Codes**

#Install packages

install.packages("caret")

install.packages("rpart.plot")

install.packages("glmnet")

# Load necessary libraries

library(caret)

library(pROC)

library(rpart)

library(rpart.plot)

library(glmnet) # For regularization

# Load your dataset

df <- read.csv("C:\\Users\\sayas\\OneDrive\\New folder\\python projects\\wine.csv")

# Display the first few rows of the dataset

print(head(df))

# Summary statistics of the dataset

print(summary(df))

# Check for missing values

cat("Total missing values: ", sum(is.na(df)), "\n")

# Custom function to calculate mode

mode\_function <- function(x) {

uniq\_x <- unique(x)

uniq\_x[which.max(tabulate(match(x, uniq\_x)))]

}

# Function to impute missing values

impute\_missing\_values <- function(df) {

for (col in names(df)) {

if (is.numeric(df[[col]])) {

df[[col]][is.na(df[[col]])] <- median(df[[col]], na.rm = TRUE)

} else {

df[[col]][is.na(df[[col]])] <- mode\_function(df[[col]][!is.na(df[[col]])])

}

}

return(df)

}

# Impute missing values

df <- impute\_missing\_values(df)

# Verify there are no more missing values

cat("Total missing values after imputation: ", sum(is.na(df)), "\n")

# Ensure the target variable is a factor with exactly two levels

df$quality <- as.factor(df$quality)

# Convert target variable to numeric (1 for "ckd" and 0 for "notckd")

df$quality <- ifelse(df$quality == "ckd", 1, 0)

# Feature scaling

preProc <- preProcess(df[, -which(names(df) == "quality")], method = c("center", "scale"))

scaled\_data <- predict(preProc, df[, -which(names(df) == "quality")])

df\_scaled <- cbind(scaled\_data, quality = df$quality)

# Split the data into training and testing sets

library(caret)

unique(df\_scaled$quality)

df\_scaled$quality <- as.factor(df\_scaled$quality)

set.seed(123)

trainIndex <- createDataPartition(df\_scaled$quality, p = 0.7, list = FALSE)

trainData <- df\_scaled[trainIndex, ]

testData <- df\_scaled[-trainIndex, ]

# Check the distribution of the target variable in training and testing sets

cat("Training set distribution:\n")

print(table(trainData$quality))

cat("Testing set distribution:\n")

print(table(testData$quality))

# Prepare data for glmnet (regularized logistic regression)

x\_train <- as.matrix(trainData[, -which(names(trainData) == "quality")])

x\_test <- as.matrix(testData[, -which(names(testData) == "quality")])

y\_train <- as.numeric(trainData$quality)

y\_test <- as.numeric(testData$quality)

# Ensure all features are numeric in x\_train

non\_numeric\_columns <- colnames(trainData)[!sapply(trainData, is.numeric)]

cat("Non-numeric columns in trainData: ", non\_numeric\_columns, "\n")

# If there are non-numeric columns, convert them to numeric

for (col in non\_numeric\_columns) {

suppressWarnings({

trainData[[col]] <- as.numeric(as.character(trainData[[col]]))

testData[[col]] <- as.numeric(as.character(testData[[col]]))

})

}

# Re-prepare the matrices after conversion

x\_train <- as.matrix(trainData[, -which(names(trainData) == "quality")])

x\_test <- as.matrix(testData[, -which(names(testData) == "quality")])

# Check for NA values in the matrices and vectors

cat("Any NA in x\_train after conversion: ", any(is.na(x\_train)), "\n")

cat("Any NA in x\_test after conversion: ", any(is.na(x\_test)), "\n")

# If there are still missing values, use makeX() to impute them

x\_train[is.na(x\_train)] <- 0

x\_test[is.na(x\_test)] <- 0

# Logistic Regression Model with Regularization

library(glmnet)

x\_train <- model.matrix(~ . - 1, data = trainData) # Removes intercept column

y\_train <- as.numeric(trainData$quality) - 1 # Assuming 'quality' is a binary factor

set.seed(123)

log\_model <- glmnet(x\_train, y\_train, family = "binomial")

# Predict on the test set using the best lambda

log\_pred <- predict(cv\_log\_model, newx = x\_test, s = "lambda.min", type = "response")

log\_pred <- as.vector(log\_pred) # Ensure log\_pred is a numeric vector

log\_pred\_quality <- ifelse(log\_pred > 0.5, 1, 0)

# Confusion Matrix for Logistic Regression

log\_conf\_matrix <- confusionMatrix(as.factor(log\_pred\_quality), as.factor(y\_test))

cat("Confusion Matrix for Logistic Regression:\n")

print(log\_conf\_matrix)

# ROC Curve for Logistic Regression

roc\_log <- roc(y\_test, log\_pred)

plot(roc\_log, main = "ROC Curve for Logistic Regression", col = "black")

cat("AUC for Logistic Regression: ", auc(roc\_log), "\n")

# Decision Tree Model

tree\_model <- rpart(quality ~ ., data = trainData, method = "quality")

# Plot Decision Tree

rpart.plot(tree\_model)

# Predict on the test set using Decision Tree

tree\_pred <- predict(tree\_model, newdata = testData, type = "quality")

tree\_pred\_prob <- predict(tree\_model, newdata = testData, type = "prob")[, 2]

# Confusion Matrix for Decision Tree

tree\_conf\_matrix <- confusionMatrix(tree\_pred, as.factor(y\_test))

cat("Confusion Matrix for Decision Tree:\n")

print(tree\_conf\_matrix)

# ROC Curve for Decision Tree

roc\_tree <- roc(y\_test, tree\_pred\_prob)

plot(roc\_tree, col = "red", add = TRUE)

legend("bottomright", legend = c("Logistic Regression", "Decision Tree"), col = c("black", "red"), lwd = 2)

cat("AUC for Decision Tree: ", auc(roc\_tree), "\n")

# Compare Models

cat("Logistic Regression vs Decision Tree\n")

cat("AUC for Logistic Regression: ", auc(roc\_log), "\n")

cat("AUC for Decision Tree: ", auc(roc\_tree), "\n")

**Python Codes**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.linear\_model import LogisticRegression**

**from sklearn.tree import DecisionTreeClassifier**

**from sklearn.metrics import confusion\_matrix, roc\_curve, roc\_auc\_score**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import OneHotEncoder**

**from sklearn.compose import ColumnTransformer**

**from sklearn.pipeline import Pipeline**

**from sklearn.impute import SimpleImputer**

**import os**

**os.chdir('C:\\Users\\sayas\\OneDrive\\New folder\\python projects')**

**# Load the dataset**

**data = pd.read\_csv('wine.csv')**

**# Encode categorical variables**

**categorical\_features = ['quality']**

**numeric\_features = [col for col in data.columns if col not in categorical\_features + ['left']]**

**# Preprocessing for numerical data**

**numeric\_transformer = SimpleImputer(strategy='median')**

**# Preprocessing for categorical data**

**categorical\_transformer = Pipeline(steps=[**

**('imputer', SimpleImputer(strategy='constant', fill\_value='missing')),**

**('onehot', OneHotEncoder(handle\_unknown='ignore'))])**

**# Combine preprocessors**

**preprocessor = ColumnTransformer(**

**transformers=[**

**('num', numeric\_transformer, numeric\_features),**

**('cat', categorical\_transformer, categorical\_features)])**

**# Split the data into training and testing sets**

**import pandas as pd**

**from sklearn.model\_selection import train\_test\_split**

**# Example DataFrame (replace with your actual data loading method)**

**data = pd.read\_csv('wine.csv')**

**# Check the column names in your DataFrame**

**print(data.columns)**

**# Ensure 'left' column exists in your DataFrame**

**if 'left' in data.columns:**

**# Split the data into training and testing sets**

**X = data.drop('left', axis=1)**

**y = data['left']**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=123)**

**# Logistic Regression**

**logit = Pipeline(steps=[('preprocessor', preprocessor),**

**('classifier', LogisticRegression(max\_iter=1000))])**

**# Generate synthetic data**

**X, y = make\_classification(n\_samples=1000, n\_features=10, random\_state=42)**

**# Split data into train and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Initialize logistic regression model**

**logit = LogisticRegression()**

**# Fit the model**

**logit.fit(X\_train, y\_train)**

**# Make predictions**

**logit\_pred = logit.predict(X\_test)**

**# Predict probabilities**

**logit\_pred\_proba = logit.predict\_proba(X\_test)[:, 1]**

**# Example usage of predictions and probabilities**

**print("Predictions:", logit\_pred[:5]) # Print first 5 predictions**

**print("Probabilities:", logit\_pred\_proba[:5]) # Print first 5 predicted probabilities**

**# Confusion Matrix**

**print("Logistic Regression Confusion Matrix:")**

**print(confusion\_matrix(y\_test, logit\_pred))**

**# ROC Curve**

**fpr, tpr, \_ = roc\_curve(y\_test, logit\_pred\_proba)**

**roc\_auc = roc\_auc\_score(y\_test, logit\_pred\_proba)**

**plt.figure()**

**plt.plot(fpr, tpr, color='blue', lw=2, label='Logistic Regression (area = %0.2f)' % roc\_auc)**

**# Decision Tree**

**tree = Pipeline(steps=[('preprocessor', preprocessor),**

**('classifier', DecisionTreeClassifier(random\_state=123))])**

**# Generate synthetic data**

**X, y = make\_classification(n\_samples=1000, n\_features=10, random\_state=42)**

**# Split data into train and test sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**# Initialize decision tree classifier**

**tree = DecisionTreeClassifier()**

**# Fit the model**

**tree.fit(X\_train, y\_train)**

**# Make predictions**

**tree\_pred = tree.predict(X\_test)**

**# Predict probabilities**

**tree\_pred\_proba = tree.predict\_proba(X\_test)[:, 1]**

**# Example usage of predictions and probabilities**

**print("Predictions:", tree\_pred[:5]) # Print first 5 predictions**

**print("Probabilities:", tree\_pred\_proba[:5]) # Print first 5 predicted probabilities**

**# Confusion Matrix**

**print("Decision Tree Confusion Matrix:")**

**print(confusion\_matrix(y\_test, tree\_pred))**

**# ROC Curve**

**fpr, tpr, \_ = roc\_curve(y\_test, tree\_pred\_proba)**

**roc\_auc = roc\_auc\_score(y\_test, tree\_pred\_proba)**

**plt.plot(fpr, tpr, color='red', lw=2, label='Decision Tree (area = %0.2f)' % roc\_auc)**

**# Plot ROC curve**

**plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')**

**plt.xlim([0.0, 1.0])**

**plt.ylim([0.0, 1.05])**

**plt.xlabel('False Positive Rate')**

**plt.ylabel('True Positive Rate')**

**plt.title('Receiver Operating Characteristic')**

**plt.legend(loc="lower right")**

**plt.show()**

**References**

1. [www.github.com](http://www.github.com)