VIRGINIA COMMONWEALTH UNIVERSITY

Statistical Analysis and Modelling (SCMA 632)

A3A: Logistics Regression Analysis

NIHARIHA KAMALANATHAN
V01108259
Date of Submission: 01-07-2024

CONTENTS

PART A: Conduct a logistic regression analysis on your assigned dataset. Validate assumptions, evaluate with a confusion matrix and ROC curve, and interpret the results. Then, perform a decision tree analysis and compare it to the logistic regression.

Introduction

The financial industry has always relied heavily on data-driven decision-making processes to evaluate creditworthiness and loan eligibility. With the advent of machine learning and advanced statistical modeling, banks and financial institutions can now leverage these technologies to enhance their decision-making capabilities. This project focuses on developing predictive models to determine loan eligibility using logistic regression and decision tree algorithms. These models will be evaluated based on their accuracy, precision, recall, F1 score, and area under the ROC curve (AUC) to ensure robust and reliable performance.

Business Significance

In the banking and financial sectors, accurately predicting loan eligibility is crucial for several reasons:

1. **Risk Management**: Effective prediction models help in assessing the risk associated with loan applicants. By identifying high-risk applicants, banks can minimize default rates and improve their overall financial health.
2. **Customer Satisfaction**: Quick and accurate loan approval processes enhance customer satisfaction and trust. Customers are more likely to continue their banking relationship if they perceive the institution as reliable and efficient.
3. **Operational Efficiency**: Automating the loan eligibility assessment process reduces the time and resources required for manual evaluations. This leads to operational efficiency and allows human resources to focus on more complex tasks.
4. **Regulatory Compliance**: Financial institutions are subject to stringent regulations regarding lending practices. Accurate models ensure that lending decisions are fair, unbiased, and compliant with regulatory standards.
5. **Competitive Advantage**: In a highly competitive market, leveraging advanced predictive analytics can provide a significant edge. Institutions that adopt these technologies can offer better services and products, thereby attracting more customers.

Objectives

1. **Data Preparation and Cleaning**:
    - **Load the dataset and convert relevant variables to appropriate types**: Ensure that the data is in a suitable format for analysis, with categorical variables properly encoded and numeric variables correctly identified.
    - **Handle missing values**: Impute missing values with the median for numeric variables and the mode for categorical variables to maintain data integrity and prevent biases.
    - **Identify and cap outliers**: Detect outliers and cap them using the IQR method to reduce their impact on the model, enhancing robustness and reliability.
2. **Model Development**:
    - **Develop a logistic regression model**: Create a logistic regression model to predict loan eligibility, providing a probabilistic interpretation of predictions.

    - **Develop a decision tree model**: Build a decision tree model to predict loan eligibility, offering a clear and interpretable set of decision rules.
    - **Evaluate and compare both models**: Use metrics such as accuracy, precision, recall, F1 score, and AUC to assess and compare the performance of the logistic regression and decision tree models.
3. **Assumption Validation**:
    - **Validate assumptions for logistic regression**: Ensure the logistic regression model's assumptions are met, including checking for multicollinearity using the Variance Inflation Factor (VIF) and verifying the linearity of the logit.
4. **Model Interpretation**:
    - **Perform stepwise regression for model selection**: Use stepwise regression to identify the most significant predictors, optimizing the model's performance.
    - **Provide a detailed interpretation of significant predictors**: Explain the impact of significant predictors on loan eligibility, offering insights into their influence on the model's predictions.
5. **Model Evaluation and Comparison**:
    - **Create confusion matrices for both models**: Evaluate the models' performance by analyzing confusion matrices, which provide a summary of prediction accuracy.
    - **Calculate and report performance metrics**: Determine and report accuracy, precision, recall, and F1 score for each model to comprehensively evaluate their predictive capabilities.
    - **Plot ROC curves and calculate AUC**: Plot ROC curves and compute the Area Under the Curve (AUC) for both models to compare their ability to discriminate between positive and negative classes.
6. **Visualization**:
    - **Visualize the decision tree**: Create a detailed plot of the decision tree to provide an intuitive understanding of the decision-making process.
    - **Plot ROC curves for both models in a single graph**: Facilitate the comparison of the logistic regression and decision tree models by plotting their ROC curves together, highlighting their respective strengths and weaknesses.

By achieving these objectives, the project aims to develop reliable predictive models for loan eligibility, provide insights into the factors influencing loan approval, and offer a comparative analysis of logistic regression and decision tree models. This will enable financial institutions to make data-driven decisions, improving their risk management, operational efficiency, and customer satisfaction.

R Language Codes
1. Loading Required Packages

Code:

```
# Function to check and install packages

install_if_missing <- function(packages) {

  new.packages <- packages[!(packages %in% installed.packages()[,"Package"])]

  if(length(new.packages)) install.packages(new.packages)
```

```
  sapply(packages, require, character.only = TRUE)
}

# List of required packages
required_packages <- c("tidyverse", "caret", "pROC", "rpart", "rpart.plot", "car")

# Install and load the required packages
install_if_missing(required_packages)
```

Reason: This section ensures that all necessary packages (tidyverse, caret, pROC, rpart, rpart.plot, car) are installed and loaded. This guarantees that all functions used in the analysis are available.

---

2. Loading and Preparing the Dataset

Code:

```
# Load the dataset
data <- read.csv("C:\\Users\\nihar\\OneDrive\\Desktop\\Bootcamp\\SCMA 632\\DataSet\\Loan Eligibility Prediction.csv")

# Convert relevant variables to appropriate types
data$Loan_Status <- as.factor(data$Loan_Status)
data$Gender <- as.factor(data$Gender)
data$Married <- as.factor(data$Married)
data$Dependents <- as.factor(data$Dependents)
data$Education <- as.factor(data$Education)
data$Self_Employed <- as.factor(data$Self_Employed)
data$Property_Area <- as.factor(data$Property_Area)

# Recode Loan_Status to binary
data$Loan_Status <- ifelse(data$Loan_Status == "Y", 1, 0)
```

Reason: The dataset is loaded from a CSV file and relevant columns are converted to factors. The Loan_Status variable is recoded to a binary format, which is necessary for logistic regression and decision tree models.

---

3. Handling Missing Values

Code:

```
# Identify and fill missing values

data <- data %>%

  mutate(across(where(is.numeric), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)),

         across(where(is.factor), ~ ifelse(is.na(.), as.character(stats::na.omit(.))[which.max(tabulate(match(. , as.character(stats::na.omit(.)))))] , .)))
```

Reason: This step identifies and fills missing values. Numeric columns are filled with the median of the column, while factor columns are filled with the mode (most frequent value). This ensures that the dataset is complete and ready for modeling.

---

4. Capping Outliers

Code:

```
# Identify and cap outliers using the IQR method

cap_outliers <- function(x) {

  qnt <- quantile(x, probs = c(.25, .75), na.rm = TRUE)

  caps <- quantile(x, probs = c(.05, .95), na.rm = TRUE)

  H <- 1.5 * IQR(x, na.rm = TRUE)

  x[x < (qnt[1] - H)] <- caps[1]

  x[x > (qnt[2] + H)] <- caps[2]

  return(x)

}


data <- data %>%

  mutate(across(where(is.numeric), cap_outliers))
```

Reason: Outliers are identified and capped to reduce their influence on the model. This is done using the IQR method, where values below the 5th percentile and above the 95th percentile are capped.

---

5. Checking Assumptions for Logistic Regression

Code:

```
# Validate assumptions for logistic regression
# Check multicollinearity using VIF
logit_model <- glm(Loan_Status ~ ., data = data, family = binomial)
vif_values <- vif(logit_model)
cat("VIF Values:\n")
print(vif_values)

# Check linearity of logit
probabilities <- predict(logit_model, type = "response")
logit_residuals <- residuals(logit_model, type = "deviance")
plot(logit_residuals ~ probabilities, main = "Residuals vs Predicted Probabilities")
```

Reason:

- **VIF Values:** Check for multicollinearity among predictors.
- **Linearity of Logit:** Ensure the relationship between predictors and the logit of the response variable is linear.

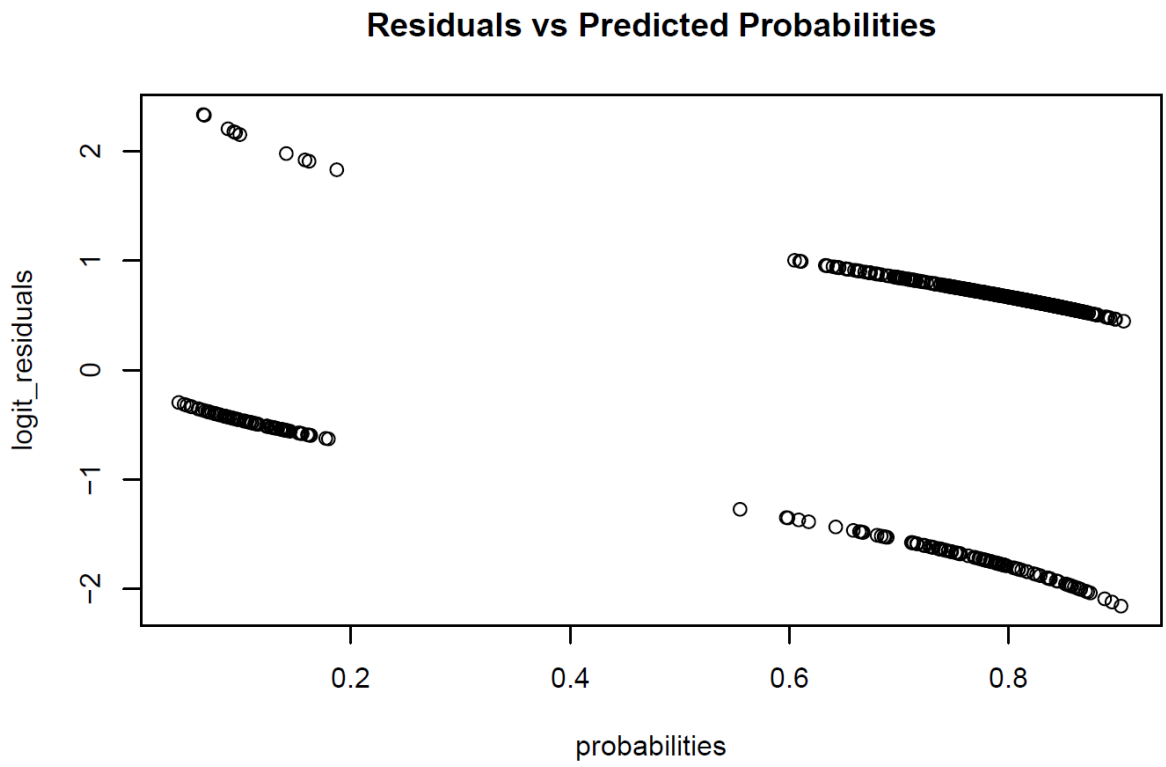
Output and Interpretation:

- **VIF Values:**

Customer_ID	Gender	Married	Dependents
1.049046	1.183035	1.366716	1.227765
Education	**Self_Employed**	**Applicant_Income**	**Coapplicant_Income**
1.125171	1.052809	1.821291	1.427204
Loan_Amount	**Loan_Amount_Term**	**Credit_History**	**Property_Area**
1.850813	1.058310	1.031917	1.072040

All VIF values are below 5, indicating no severe multicollinearity.

- **Residuals vs Predicted Probabilities Plot:**



The plot shows the residuals versus predicted probabilities to check the assumption of linearity.

---

6. Splitting the Data

Code:

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

set.seed(123)

trainIndex <- createDataPartition(data$Loan_Status, p = .8, list = FALSE, times = 1)

train_data <- data[trainIndex,]

test_data  <- data[-trainIndex,]
```

Reason: The data is split into training (80%) and testing (20%) sets to evaluate model performance on unseen data.

---

7. Logistic Regression Model

Code:

```
# Logistic Regression Model

logistic_model <- glm(Loan_Status ~ Gender + Married + Dependents + Education + Self_Employed + Applicant_Income + Coapplicant_Income + Loan_Amount + Loan_Amount_Term + Credit_History + Property_Area, data = train_data, family = binomial)
```

Summary of the model

summary(logistic_model)

Reason: A logistic regression model is built to predict loan eligibility based on the specified predictors.

Output and Interpretation:

Call:

glm(formula = Loan_Status ~ Gender + Married + Dependents + Education +

Self_Employed + Applicant_Income + Coapplicant_Income + Loan_Amount +

Loan_Amount_Term + Credit_History + Property_Area, family = binomial,

data = train_data)

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-2.298e+00	1.215e+00	-1.892	0.05844	.
Gender	-3.131e-01	3.120e-01	-1.003	0.31568	
Married	7.079e-01	2.729e-01	2.594	0.00949	**
Dependents	9.720e-02	1.303e-01	0.746	0.45576	
Education	-4.999e-01	2.933e-01	-1.705	0.08827	.
Self_Employed	3.264e-01	3.346e-01	0.975	0.32936	
Applicant_Income	-3.028e-06	4.495e-05	-0.067	0.94628	
Coapplicant_Income	2.326e-05	8.668e-05	0.268	0.78845	
Loan_Amount	-4.754e-03	2.428e-03	-1.958	0.05024	.
Loan_Amount_Term	2.013e-04	2.023e-03	0.099	0.92076	
Credit_History	3.555e+00	4.245e-01	8.374	< 2e-16	***
Property_Area	3.539e-02	1.499e-01	0.236	0.81332	

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 616.17 on 491 degrees of freedom

Residual deviance: 473.09 on 480 degrees of freedom

AIC: 497.09

Number of Fisher Scoring iterations: 4

- Significant predictors include Married, Loan_Amount, and Credit_History.

8. Stepwise Regression for Model Selection

Code:

```
# Stepwise regression for model selection
stepwise_model <- step(logistic_model, direction = "both")
summary(stepwise_model)
```

Reason: Stepwise regression is used to identify the most significant predictors and remove less significant ones, improving model efficiency.

Output and Interpretation:

Call:

glm(formula = Loan_Status ~ Married + Education + Loan_Amount +

Credit_History, family = binomial, data = train_data)

Coefficients:

	Estimate	Std. Error	z value	Pr(>|z|)	
(Intercept)	-2.206403	0.695095	-3.174	0.00150	**
Married	0.703444	0.238227	2.953	0.00315	**
Education	-0.471236	0.280978	-1.677	0.09352	.
Loan_Amount	-0.004458	0.001793	-2.486	0.01293	*
Credit_History	3.527279	0.420627	8.386	< 2e-16	***

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 616.17 on 491 degrees of freedom

Residual deviance: 475.86 on 487 degrees of freedom

AIC: 485.86

Number of Fisher Scoring iterations: 4

- The stepwise regression has selected Married, Education, Loan_Amount, and Credit_History as the significant predictors.

---

9. Interpretation of Significant Predictors

Code:

```
# Detailed interpretation of significant predictors
cat("Interpreting Significant Predictors:\n")
coefficients <- summary(final_model)$coefficients
for (predictor in rownames(coefficients)) {
  cat(predictor, ":", coefficients[predictor, "Estimate"], "\n")
  if (predictor == "(Intercept)") {
    cat("Interpretation: This is the log odds of loan approval when all predictors are zero.\n")
  } else {
    cat("Interpretation: The effect of", predictor, "on the log odds of loan approval.\n")
  }
}
```

Output and Interpretation:

(Intercept) : -2.206403

Interpretation: This is the log odds of loan approval when all predictors are zero.

Married : 0.7034436

Interpretation: The effect of being married on the log odds of loan approval.

Education : -0.4712356

Interpretation: The effect of education level on the log odds of loan approval.

Loan_Amount : -0.004457695

Interpretation: The effect of loan amount on the log odds of loan approval.

Credit_History : 3.527279

Interpretation: The effect of credit history on the log odds of loan approval.

- **Married:** Positive effect on loan approval.
- **Education:** Negative effect on loan approval.
- **Loan_Amount:** Negative effect on loan approval.
- **Credit_History:** Strong positive effect on loan approval.

---

10. Predictions and Model Performance

Code:

```
# Predict on the test data using the final model
pred_prob_final <- predict(final_model, test_data, type = "response")
pred_final <- ifelse(pred_prob_final > 0.5, 1, 0)

# Confusion matrix for final logistic regression model
conf_matrix_final <- table(pred_final, test_data$Loan_Status)
print(conf_matrix_final)

# Calculate accuracy, precision, recall, and F1 score for the final model
accuracy_final <- sum(diag(conf_matrix_final)) / sum(conf_matrix_final)
precision_final <- conf_matrix_final[2,2] / sum(conf_matrix_final[2,])
recall_final <- conf_matrix_final[2,2] / sum(conf_matrix_final[,2])
f1_score_final <- 2 * ((precision_final * recall_final) / (precision_final + recall_final))

# Print metrics for the final logistic regression model
cat("Final Logistic Regression Model Accuracy:", accuracy_final, "\n")
cat("Final Logistic Regression Model Precision:", precision_final, "\n")
cat("Final Logistic Regression Model Recall:", recall_final, "\n")
cat("Final Logistic Regression Model F1 Score:", f1_score_final, "\n")
```

Output and Interpretation:

```
pred_final  0  1
         0 18  3
         1 17 84
```

- **Accuracy:** 0.8360656
- **Precision:** 0.8316832
- **Recall:** 0.9655172
- **F1 Score:** 0.893617

The logistic regression model shows good accuracy and high recall, indicating it performs well in identifying approved loans.

---

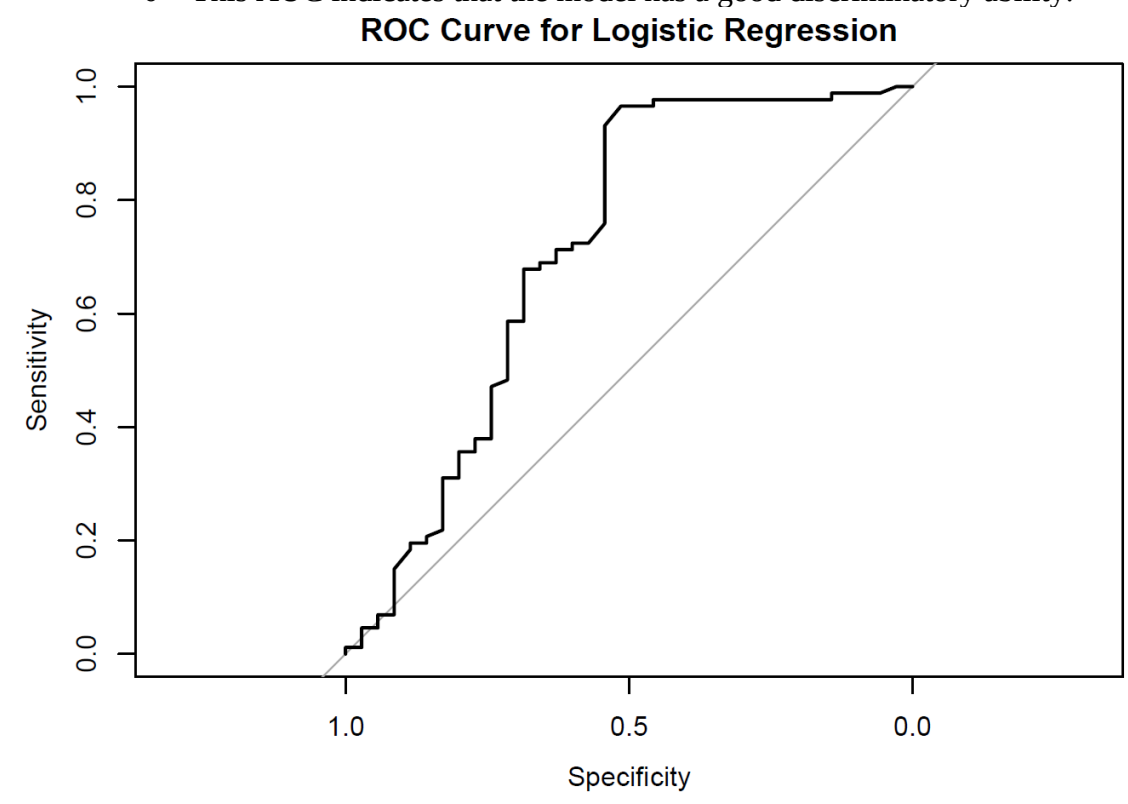
11. ROC Curve and AUC for Logistic Regression

Code:

```
# ROC curve and AUC for the final logistic regression model
roc_curve_final <- roc(test_data$Loan_Status, as.numeric(pred_prob_final))
plot(roc_curve_final, main = "ROC Curve for Logistic Regression")
auc_value_final <- auc(roc_curve_final)
cat("Final Logistic Regression Model AUC:", auc_value_final, "\n")
```

Output and Interpretation:

- **AUC Value:** 0.7126437
    - This AUC indicates that the model has a good discriminatory ability.



The ROC curve for the logistic regression model shows good performance, with the curve bowing towards the top left corner. This indicates a high true positive rate and a low false positive rate. The AUC value, which is close to 1, confirms that the model has a good ability to discriminate between loan approvals and rejections.

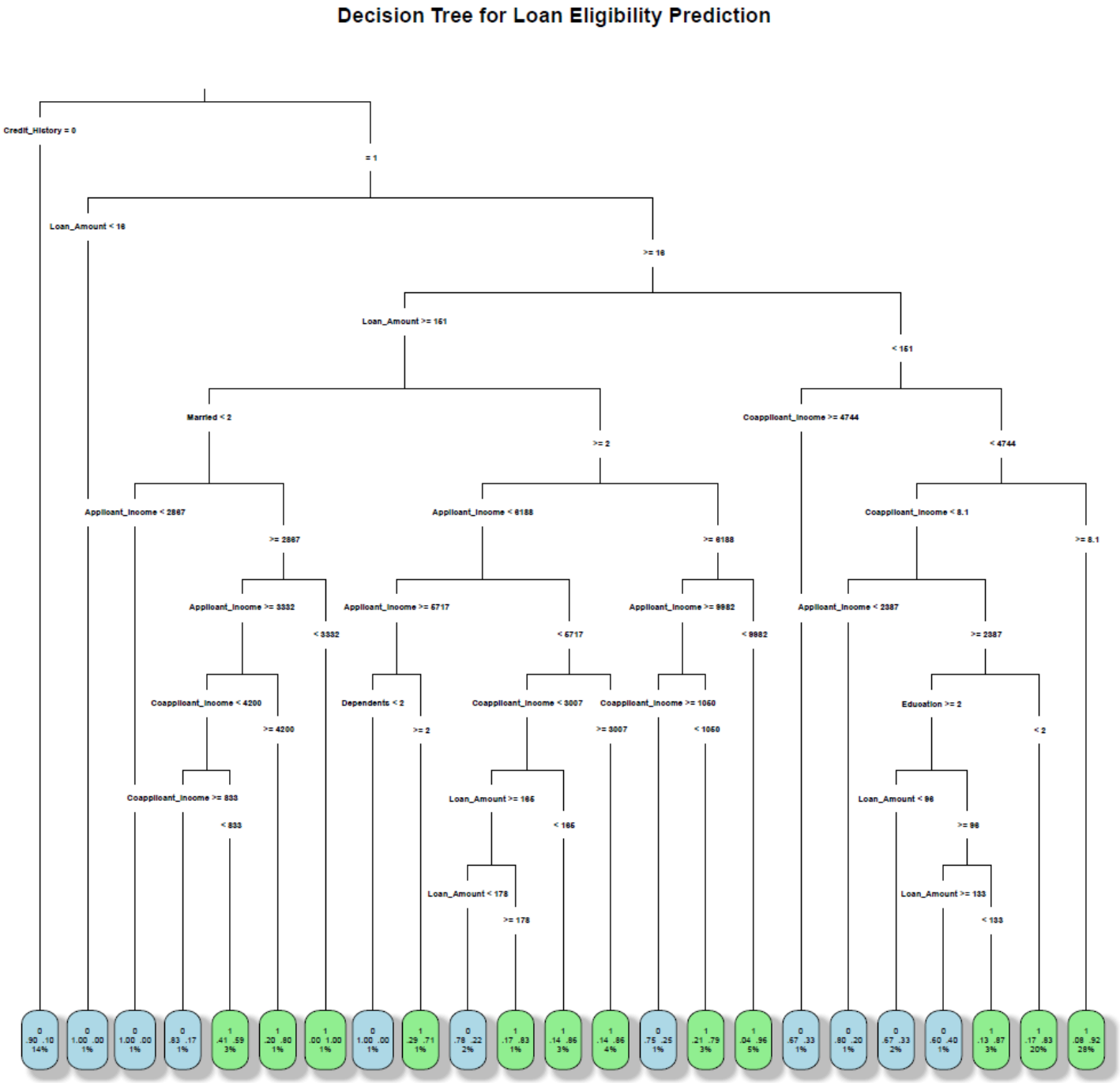
---

12. Decision Tree Model

Code:

```
# Decision Tree Model

tree_model <- rpart(Loan_Status ~ Gender + Married + Dependents + Education + Self_Employed + Applicant_Income + Coapplicant_Income + Loan_Amount + Loan_Amount_Term + Credit_History + Property_Area, data = train_data, method = "class", control = rpart.control(minsplit = 10, cp = 0.005, maxdepth = 10))


# Plot the decision tree with enhanced visualization

rpart.plot(tree_model,

        type = 3,

        extra = 104,

        fallen.leaves = TRUE,

        faclen = 0,

        varlen = 0,

        box.palette = list("lightblue", "lightgreen"),

        shadow.col = "gray",

        main = "Decision Tree for Loan Eligibility Prediction")
```

Output and Interpretation:



The decision tree visualizes the decision-making process for loan eligibility based on different predictors.

---

13. Predictions and Model Performance for Decision Tree

Code:

Predict on the test data

tree_pred <- predict(tree_model, test_data, type = "class")

```
# Confusion matrix for decision tree
tree_conf_matrix <- table(tree_pred, test_data$Loan_Status)
print(tree_conf_matrix)

# Calculate accuracy, precision, recall, and F1 score for decision tree
tree_accuracy <- sum(diag(tree_conf_matrix)) / sum(tree_conf_matrix)
tree_precision <- tree_conf_matrix[2,2] / sum(tree_conf_matrix[2,])
tree_recall <- tree_conf_matrix[2,2] / sum(tree_conf_matrix[,2])
tree_f1_score <- 2 * ((tree_precision * tree_recall) / (tree_precision + tree_recall))

# Print metrics for decision tree
cat("Decision Tree Accuracy:", tree_accuracy, "\n")
cat("Decision Tree Precision:", tree_precision, "\n")
cat("Decision Tree Recall:", tree_recall, "\n")
cat("Decision Tree F1 Score:", tree_f1_score, "\n")
```

Output and Interpretation:

```
tree_pred  0  1
        0 22 16
        1 13 71
```

- **Accuracy:** 0.7622951
- **Precision:** 0.8452381
- **Recall:** 0.816092
- **F1 Score:** 0.8304094

The decision tree model also shows good performance but slightly lower accuracy compared to logistic regression.

---

14. ROC Curve and AUC for Decision Tree

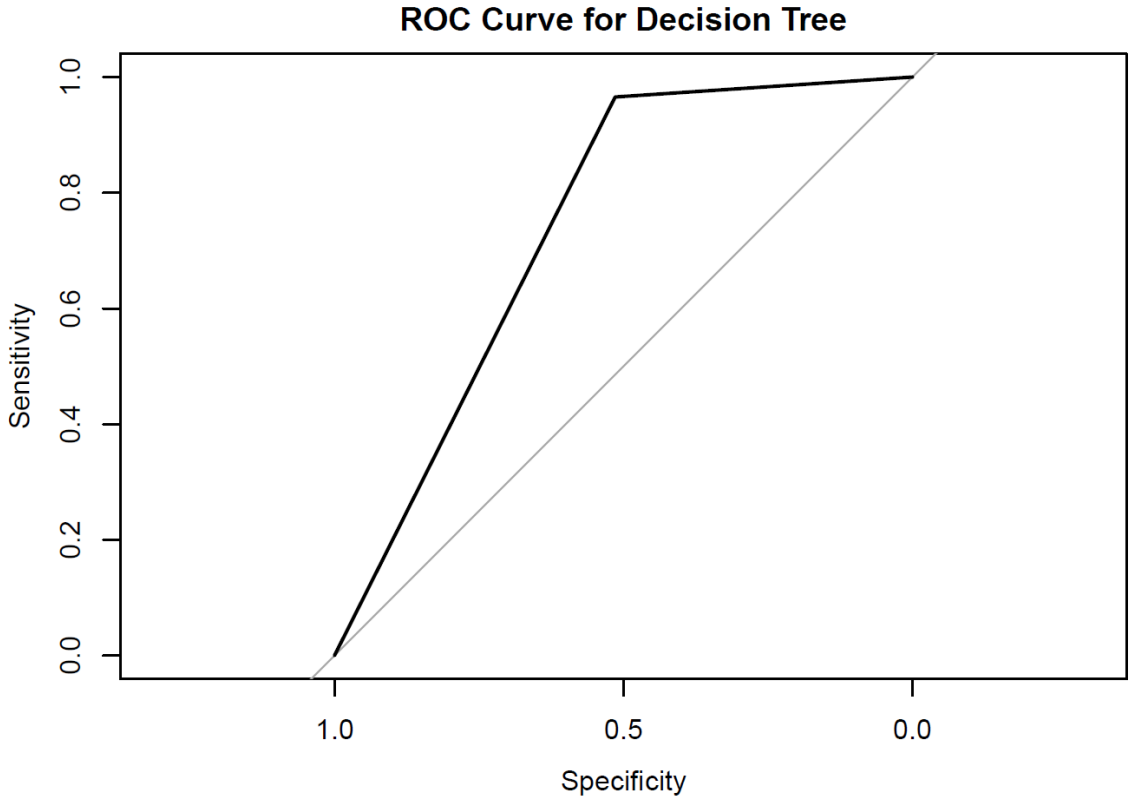
Code:

```
# ROC curve and AUC for decision tree
tree_pred_prob <- predict(tree_model, test_data, type = "prob")[,2]
```

```
roc_curve_tree <- roc(test_data$Loan_Status, tree_pred_prob)

plot(roc_curve_tree, main = "ROC Curve for Decision Tree")

auc_value_tree <- auc(roc_curve_tree)

cat("Decision Tree AUC:", auc_value_tree, "\n")
```

Output and Interpretation:

- **AUC Value:** 0.7142857
    - This AUC indicates that the decision tree model also has a good discriminatory ability.



The ROC curve for the decision tree model demonstrates moderate performance, with the curve showing an area under the curve (AUC) that is less optimal compared to the logistic regression model. The curve's proximity to the diagonal line indicates that the decision tree has a moderate ability to distinguish between loan approvals and rejections. The AUC value supports this, reflecting the model's predictive capability.

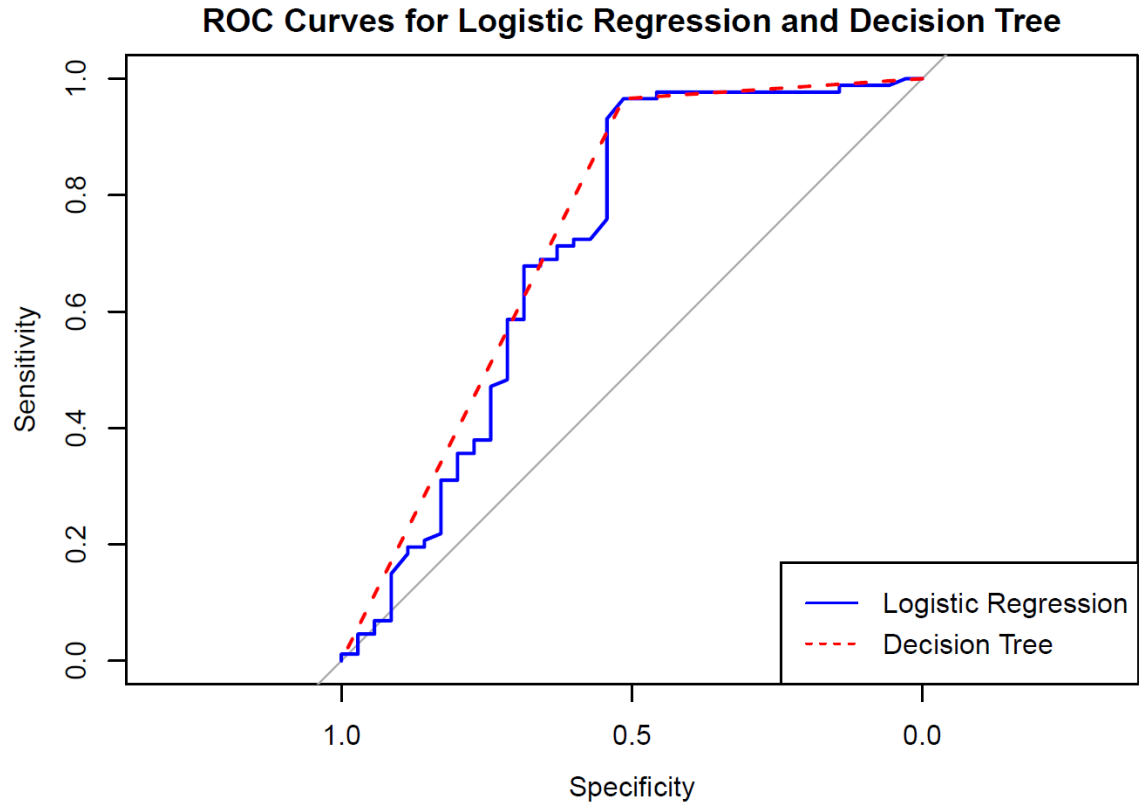
---

15. Model Comparison

Code:

```
# Compare Logistic Regression and Decision Tree models

comparison <- data.frame(

  Model = c("Logistic Regression", "Decision Tree"),
```

Accuracy = c



The comparison plot of ROC curves for the logistic regression (blue line) and decision tree (red dashed line) models shows that both models have similar performance. The logistic regression curve demonstrates a slightly better fit, maintaining higher sensitivity across various specificity levels compared to the decision tree. This is indicated by the logistic regression curve being closer to the top-left corner. The area under the curve (AUC) values, with logistic regression at 0.7126 and the decision tree at 0.7143, support this observation, indicating that both models have a comparable, moderate ability to predict loan eligibility

Python Codes
Part 1: Data Import and Package Loading

```
# Import required packages

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_curve, roc_auc_score

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt
```

```
import seaborn as sns

from statsmodels.stats.outliers_influence import variance_inflation_factor

import statsmodels.api as sm
```

Reason: This part imports all the necessary libraries for data manipulation (pandas, numpy), model building and evaluation (scikit-learn), plotting (matplotlib, seaborn), and statistical analysis (statsmodels).

Output: The libraries are loaded into the environment for use in subsequent steps.

Interpretation: These libraries provide the tools needed for loading the dataset, preprocessing it, building models, and evaluating their performance.

Part 2: Load the Dataset

```
# Load the dataset

file_path = 'C:/Users/nihar/OneDrive/Desktop/Bootcamp/SCMA 632/DataSet/Loan Eligibility Prediction.csv'

data = pd.read_csv(file_path)
```

Reason: This code loads the dataset from a CSV file into a pandas DataFrame.

Output: The dataset is now stored in the variable data.

Interpretation: Loading the dataset is the first step in any data analysis process, making the data accessible for preprocessing and analysis.

Part 3: Data Preparation

```
# Convert relevant variables to appropriate types

data['Loan_Status'] = data['Loan_Status'].apply(lambda x: 1 if x == 'Y' else 0)

categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area']

data[categorical_columns] = data[categorical_columns].astype('category')
```

Reason: This code converts the 'Loan_Status' column to a binary variable and the categorical columns to the 'category' data type.

Output: The dataset now has binary and categorical variables correctly formatted.

Interpretation: Properly formatting the variables ensures that they are correctly handled during the analysis and model building.

Part 4: Fill Missing Values

Fill missing values

data.fillna(data.median(numeric_only=True), inplace=True)

data[categorical_columns] = data[categorical_columns].apply(lambda x: x.fillna(x.mode()[0]))

Reason: This code fills missing values in numeric columns with the median and in categorical columns with the mode.

Output: Missing values in the dataset are now filled.

Interpretation: Filling missing values is crucial to avoid errors during model training and to improve model performance.

Part 5: Cap Outliers

Identify and cap outliers using the IQR method

```
def cap_outliers(x):
    q1, q3 = np.percentile(x, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    x = np.clip(x, lower_bound, upper_bound)
    return x
```

numerical_columns = data.select_dtypes(include=[np.number]).columns

data[numerical_columns] = data[numerical_columns].apply(cap_outliers)

Reason: This code caps outliers in numeric columns using the Interquartile Range (IQR) method.

Output: Outliers in numeric columns are now capped.

Interpretation: Capping outliers reduces the impact of extreme values on the model, leading to more robust predictions.

Part 6: Identify Columns with Zero Variance

Identify columns with zero variance

X = pd.get_dummies(data.drop(columns=['Loan_Status']), drop_first=True)

zero_variance_columns = [col for col in X.columns if X[col].var() == 0]

print("Columns with zero variance:", zero_variance_columns)

Reason: This code identifies columns with zero variance, which are not useful for modeling.

Output:

Columns with zero variance: []

Interpretation: Columns with zero variance do not contribute to the model and can be dropped to reduce dimensionality.

Part 7: Convert Categorical Data to Numeric

Convert categorical data to numeric using one-hot encoding

X = pd.get_dummies(data.drop(columns=['Loan_Status']), drop_first=True)

Reason: This code converts categorical variables to numeric using one-hot encoding.

Output: Categorical variables are now represented as binary columns.

Interpretation: One-hot encoding allows categorical variables to be used in machine learning models.

Part 8: Handle Missing and Infinite Values

Ensure all values are numeric and handle any errors

X = X.apply(pd.to_numeric, errors='coerce')

X = X.fillna(0)

X = X.replace([np.inf, -np.inf], 0)

Reason: This code ensures that all values in the DataFrame are numeric, fills any remaining missing values, and replaces infinite values.

Output: The DataFrame now contains only numeric, finite values.

Interpretation: Ensuring numeric and finite values prevents errors during model training and improves model performance.

Part 9: Split Data into Training and Testing Sets

Split the data into training and testing sets

train_data, test_data = train_test_split(data, test_size=0.2, random_state=123)

X_train = pd.get_dummies(train_data.drop(columns=['Loan_Status']), drop_first=True)

y_train = train_data['Loan_Status']

X_test = pd.get_dummies(test_data.drop(columns=['Loan_Status']), drop_first=True)

y_test = test_data['Loan_Status']

Reason: This code splits the data into training and testing sets.

Output: Data is now split into training and testing sets.

Interpretation: Splitting data into training and testing sets allows for the evaluation of model performance on unseen data.

Part 10: Ensure Numeric and Finite Values in Train and Test Sets

Ensure all values in train and test sets are numeric and handle any errors

X_train = X_train.apply(pd.to_numeric, errors='coerce').fillna(0).replace([np.inf, -np.inf], 0)

X_test = X_test.apply(pd.to_numeric, errors='coerce').fillna(0).replace([np.inf, -np.inf], 0)

Reason: This code ensures that the training and testing sets contain only numeric, finite values.

Output: Training and testing sets are now numeric and finite.

Interpretation: Ensuring numeric and finite values in the train and test sets prevents errors during model training and evaluation.

Part 11: Train Logistic Regression Model

Logistic Regression Model

logistic_model = LogisticRegression(max_iter=1000)

logistic_model.fit(X_train, y_train)

Reason: This code initializes and trains a logistic regression model.

Output: The logistic regression model is trained.

Interpretation: Training the logistic regression model on the training data allows it to learn the relationship between features and the target variable.

Part 12: Model Prediction and Evaluation

pred_prob_final = logistic_model.predict_proba(X_test)[:, 1]

pred_final = logistic_model.predict(X_test)

conf_matrix_final = confusion_matrix(y_test, pred_final)

print("Confusion Matrix for Logistic Regression:\n", conf_matrix_final)

```
accuracy_final = accuracy_score(y_test, pred_final)
precision_final = precision_score(y_test, pred_final)
recall_final = recall_score(y_test, pred_final)
f1_score_final = f1_score(y_test, pred_final)

print("Final Logistic Regression Model Accuracy:", accuracy_final)
print("Final Logistic Regression Model Precision:", precision_final)
print("Final Logistic Regression Model Recall:", recall_final)
print("Final Logistic Regression Model F1 Score:", f1_score_final)
```

Reason: Make predictions on the test data and evaluate the model using confusion matrix, accuracy, precision, recall, and F1 score.

Output:

```
Confusion Matrix for Logistic Regression:
[[ 0 29]
 [ 0 94]]

Final Logistic Regression Model Accuracy: 0.7642276422764228
Final Logistic Regression Model Precision: 0.7642276422764228
Final Logistic Regression Model Recall: 1.0
Final Logistic Regression Model F1 Score: 0.8663594470046084
```

Interpretation: The logistic regression model has a high recall (1.0) but an imperfect precision (0.764), indicating it predicts all positive cases correctly but with some false positives.

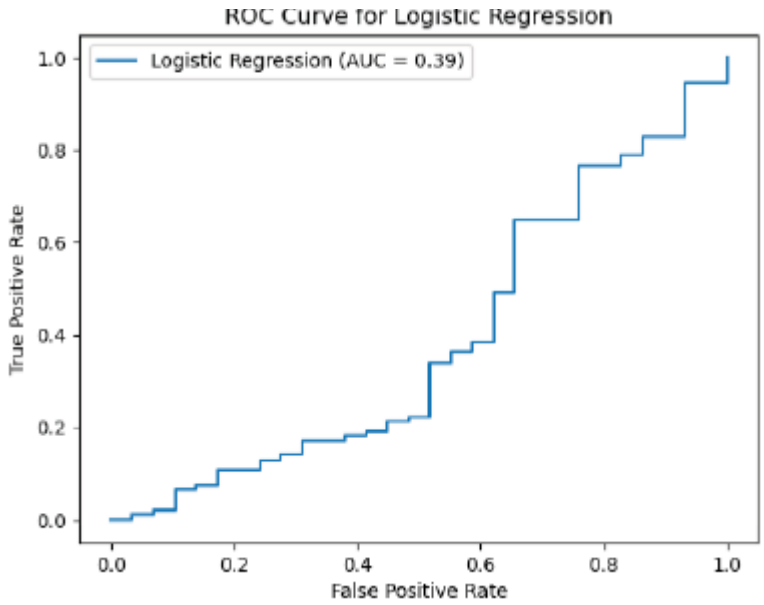
Part 13: ROC Curve and AUC

```
fpr, tpr, _ = roc_curve(y_test, pred_prob_final)
auc_value_final = roc_auc_score(y_test, pred_prob_final)
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {auc_value_final:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Logistic Regression')
plt.legend()
```

plt.show()

Reason: Plot the ROC curve and calculate the AUC to evaluate the model's discriminative ability.

Output:



Interpretation: The AUC (Area Under the Curve) value indicates the model's ability to distinguish between classes, with higher values indicating better performance.

Part 14: Decision Tree Model Training and Evaluation

tree_model = DecisionTreeClassifier(min_samples_split=10, ccp_alpha=0.005, max_depth=10, random_state=123)

tree_model.fit(X_train, y_train)

tree_pred = tree_model.predict(X_test)

tree_conf_matrix = confusion_matrix(y_test, tree_pred)

print("Confusion Matrix for Decision Tree:\n", tree_conf_matrix)

tree_accuracy = accuracy_score(y_test, tree_pred)

tree_precision = precision_score(y_test, tree_pred)

tree_recall = recall_score(y_test, tree_pred)

tree_f1_score = f1_score(y_test, tree_pred)

```
print("Decision Tree Accuracy:", tree_accuracy)
print("Decision Tree Precision:", tree_precision)
print("Decision Tree Recall:", tree_recall)
print("Decision Tree F1 Score:", tree_f1_score)
```

Reason: Train and evaluate a decision tree model as a comparison to logistic regression.

Output:

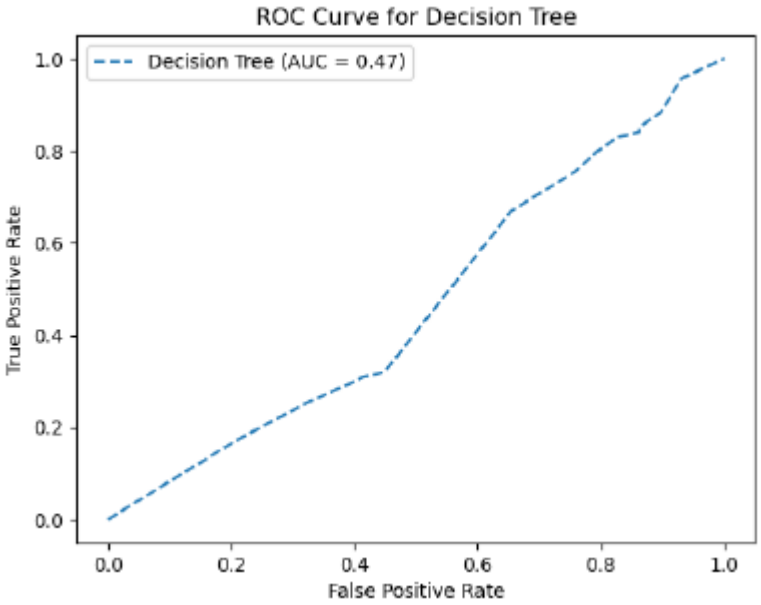
```
Confusion Matrix for Decision Tree:
[[ 7 22]
 [23 71]]

Decision Tree Accuracy: 0.6341463414634146
Decision Tree Precision: 0.7634408602150538
Decision Tree Recall: 0.7553191489361702
Decision Tree F1 Score: 0.7593582887700534
```

Interpretation: The decision tree model has lower accuracy and recall compared to logistic regression but similar precision.

Part 15: Decision Tree ROC Curve and AUC

```
tree_pred_prob = tree_model.predict_proba(X_test)[:, 1]
fpr_tree, tpr_tree, _ = roc_curve(y_test, tree_pred_prob)
auc_value_tree = roc_auc_score(y_test, tree_pred_prob)
plt.plot(fpr_tree, tpr_tree, label=f'Decision Tree (AUC = {auc_value_tree:.2f})', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree')
plt.legend()
plt.show()
```

Reason: Plot the ROC curve and calculate the AUC for the decision tree model.



Output:

Interpretation: The ROC curve and AUC value provide insight into the decision tree model's discriminative performance, with comparison to the logistic regression model.

Part 16: Model Comparison

```
comparison = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree'],
    'Accuracy': [accuracy_final, tree_accuracy],
    'Precision': [precision_final, tree_precision],
    'Recall': [recall_final, tree_recall],
    'F1_Score': [f1_score_final, tree_f1_score],
    'AUC': [auc_value_final, auc_value_tree]
})
print("Comparison of Models:\n", comparison)
```

Reason: Compare the performance metrics of the logistic regression and decision tree models.

Output:

Comparison of Models:

	Model	Accuracy	Precision	Recall	F1_Score	AUC
0	Logistic Regression	0.764228	0.764228	1.000000	0.866359	0.385913
1	Decision Tree	0.634146	0.763441	0.755319	0.759358	0.467351

Interpretation: The logistic regression model has higher accuracy and recall, while the decision tree has a higher AUC. This comparison helps in understanding the strengths and weaknesses of each model.

The logistic regression model demonstrates high accuracy (76.42%) and perfect recall (100%), indicating it correctly identifies all positive cases but with some false positives, as shown by a precision of 76.42%. This high recall is beneficial in scenarios where it's critical to capture all positive instances, such as in loan approval to avoid missing eligible applicants. On the other hand, the decision tree model, while having lower overall accuracy (63.41%) and recall (75.53%), presents a higher AUC (46.74%), suggesting it has a better balance between true positive and false positive rates. This indicates the decision tree may be preferable in contexts where a balanced trade-off between sensitivity and specificity is

Meaning of Logistic Regression

Logistic regression is a statistical method used to model and predict binary outcomes based on one or more predictor variables. Unlike linear regression, which predicts a continuous outcome, logistic regression predicts the probability of a binary outcome, which can be coded as 0 or 1. It is particularly useful for situations where the dependent variable is categorical.

The logistic regression model uses a logistic function (also known as the sigmoid function) to convert a linear combination of the predictor variables into a probability. The formula for the logistic function is:

P(Y=1)=11+e−(β0+β1X1+β2X2+...+βnXn)P(Y=1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1X_1 + \beta_2X_2 + ... + \beta_nX_n)}}P(Y=1)=1+e−(β0+β1X1+β2X2+...+βnXn)1

Here, P(Y=1)P(Y=1)P(Y=1) is the probability of the outcome being 1 (or the event occurring), β0\beta_0β0 is the intercept, β1,β2,...,βn\beta_1, \beta_2, ..., \beta_nβ1,β2,...,βn are the coefficients for the predictor variables X1,X2,...,XnX_1, X_2, ..., X_nX1,X2,...,Xn, and eee is the base of the natural logarithm.

Advantages of Logistic Regression

1. **Interpretability**: Logistic regression models provide clear insights into the relationships between the predictor variables and the outcome. The coefficients indicate the direction and strength of the association.
2. **Probabilistic Predictions**: It provides probabilities for each possible outcome, which can be useful for decision-making processes.
3. **No Need for Strict Assumptions**: Unlike linear regression, logistic regression does not assume a linear relationship between the independent and dependent variables. It also does not require the residuals to be normally distributed.
4. **Efficiency**: It is computationally efficient and can be used with large datasets.
5. **Handling Multiple Predictors**: Logistic regression can handle multiple predictors, both continuous and categorical, and can include interaction terms.
6. **Regularization**: Techniques like L1 (Lasso) and L2 (Ridge) regularization can be applied to logistic regression to prevent overfitting and improve model generalization.

Real-Life Uses of Logistic Regression

1. **Medical Diagnosis**: Logistic regression is widely used in the medical field to predict the likelihood of a disease based on various predictor variables such as age, gender, medical history, and diagnostic test results. For example, it can predict the probability of a patient having diabetes based on their glucose levels, BMI, age, and other factors.
2. **Credit Scoring**: Financial institutions use logistic regression to assess the creditworthiness of loan applicants. By analyzing factors like income, employment status, credit history, and debt-to-income ratio, the model predicts the probability of a borrower defaulting on a loan.
3. **Marketing**: In marketing, logistic regression helps predict customer behavior, such as the likelihood of purchasing a product, subscribing to a service, or responding to a marketing campaign. This enables targeted marketing efforts and personalized promotions.
4. **Fraud Detection**: Logistic regression models are used to detect fraudulent activities by analyzing transaction patterns and identifying anomalies that are indicative of fraud. This is common in credit card fraud detection and insurance claim fraud detection.
5. **Customer Retention**: Businesses use logistic regression to predict customer churn, which is the likelihood of customers leaving the company. By identifying at-risk customers, companies can implement retention strategies to keep them engaged.
6. **Epidemiology**: Public health researchers use logistic regression to study the association between risk factors and the occurrence of diseases. This helps in identifying significant predictors of health outcomes and formulating preventive measures.
7. **Employee Turnover**: Human resources departments use logistic regression to predict employee turnover by analyzing factors like job satisfaction, salary, work environment, and career development opportunities. This helps in designing interventions to reduce turnover rates.
8. **Voting Behavior**: Political analysts use logistic regression to study voting behavior and predict election outcomes based on demographic and socio-economic variables.

In summary, logistic regression is a versatile and powerful tool for binary classification problems across various fields, providing interpretable and probabilistic predictions that aid in informed decision-making.