# VIRGINIA COMMONWEALTH UNIVERSITY

## Statistical analysis and modelling (SCMA 632)

**A3-Limited dependent variable Models**

**(Part – A)**

**ADHYAYAN AMIT JAIN**

**V01109421**

**Date of Submission: 30-06-2024**

# CONTENTS

# Logistic Regression and Decision Tree Analysis on Breast Cancer Dataset

## INTRODUCTION

In the realm of healthcare, the ability to accurately diagnose diseases is paramount for improving patient outcomes and enhancing the efficiency of medical interventions. Breast cancer, a prevalent and potentially life-threatening condition, requires timely and precise diagnosis to guide treatment decisions and improve survival rates. Machine learning offers a promising avenue for developing predictive models that can aid clinicians in diagnosing breast cancer by analyzing patterns within complex datasets.

This assignment delves into the application of various machine learning techniques to the Breast Cancer dataset. The objective is to construct and evaluate predictive models that can distinguish between malignant and benign breast cancer cases based on a set of clinical and diagnostic features.

To achieve this, the assignment encompasses several key steps:

1. **Data Preprocessing:** Ensuring the dataset is clean and ready for analysis by handling missing values, converting categorical variables to numerical factors, and scaling features for uniformity.
2. **Feature Selection:** Employing Recursive Feature Elimination (RFE) to identify the most relevant features that contribute to the target variable, thereby enhancing model performance.
3. **Model Training:** Training a logistic regression model using cross-validation for hyperparameter tuning, with the goal of optimizing predictive accuracy. This includes the use of the glmnet package for fitting a penalized logistic regression model.
4. **Model Evaluation:** Evaluating the logistic regression model's performance on a separate test set using metrics such as accuracy, sensitivity, specificity, and the Area Under the ROC Curve (AUC). Additionally, a decision tree model is trained and evaluated for comparison.
5. **Visualization and Interpretation:** Visualizing model performance through ROC curves and confusion matrices, and interpreting the results to understand the model's strengths and limitations in the context of breast cancer diagnosis.

By systematically applying these steps, this assignment aims to demonstrate the potential of machine learning in the healthcare domain, specifically in enhancing the diagnostic process for breast cancer. The insights gleaned from this analysis not only highlight the efficacy of different modeling techniques but also underscore the importance of meticulous data preparation and rigorous evaluation in developing reliable predictive tools.

## OBJECTIVES

The objective of Part A of this assignment is to conduct a comprehensive analysis of the Breast Cancer dataset using two distinct machine learning approaches: logistic regression and decision tree analysis. This analysis aims to achieve the following specific goals:

1. **Logistic Regression Analysis:**
   - **Validate Assumptions:** Ensure the data meets the necessary assumptions for logistic regression, including linearity of independent variables and log odds, absence of multicollinearity, and sufficient sample size.
   - **Model Training and Evaluation:** Train a logistic regression model using cross-validation for hyperparameter tuning, optimizing model performance metrics.
   - **Confusion Matrix and ROC Curve:** Evaluate the model's performance using a confusion matrix to measure accuracy, sensitivity, specificity, and other related metrics. Generate and interpret the ROC curve to assess the model's ability to distinguish between malignant and benign cases.
   - **Interpretation of Results:** Provide a detailed interpretation of the logistic regression model's performance, discussing the implications of the findings in the context of breast cancer diagnosis.
2. **Decision Tree Analysis:**
   - **Model Training and Evaluation:** Train a decision tree model on the same dataset, using appropriate preprocessing steps and parameter settings to optimize its performance.
   - **Confusion Matrix Comparison:** Evaluate the decision tree model using a confusion matrix and compare its performance metrics with those of the logistic regression model.

- **ROC Curve Comparison:** Generate and interpret the ROC curve for the decision tree model, comparing its AUC value with that of the logistic regression model to determine which model performs better in distinguishing between malignant and benign cases.

3. **Comparative Analysis:**
    - **Performance Metrics Comparison:** Conduct a comparative analysis of the logistic regression and decision tree models, focusing on key performance metrics such as accuracy, sensitivity, specificity, and AUC.
    - **Strengths and Limitations:** Discuss the strengths and limitations of each modeling approach, highlighting their respective advantages and potential drawbacks in the context of breast cancer diagnosis.
    - **Practical Implications:** Provide insights into the practical implications of the findings, suggesting how each model could be utilized in a clinical setting to support the diagnostic process for breast cancer.

By achieving these objectives, this assignment aims to demonstrate the application of logistic regression and decision tree analysis to a real-world healthcare dataset, providing valuable insights into their effectiveness and utility in diagnosing breast cancer.

# BUSINESS SIGNIFICANCE

The analysis of the Breast Cancer dataset using logistic regression and decision tree models holds significant business implications, particularly in the healthcare sector. Early and accurate detection of breast cancer is crucial for improving patient outcomes and optimizing resource allocation in medical practices. The insights gained from this analysis can be leveraged in various ways:

1. **Enhanced Diagnostic Accuracy:**
    - **Logistic Regression:** By validating assumptions and optimizing the logistic regression model, healthcare providers can benefit from a robust and interpretable tool that accurately classifies patients as having malignant or benign breast cancer. High accuracy and AUC values indicate reliable diagnostic performance, which can reduce

    - false positives and false negatives, leading to better patient outcomes.
    - **Decision Tree Analysis:** Decision trees offer an intuitive and easily interpretable model that can be implemented in clinical settings. The clear decision rules derived from the tree can aid healthcare professionals in making quick and informed decisions regarding patient diagnosis and treatment plans.
2. **Resource Optimization:**
    - Accurate models can significantly reduce the number of unnecessary diagnostic procedures and treatments, optimizing the use of healthcare resources. For example, fewer false positives mean fewer patients undergoing unnecessary biopsies or treatments, thereby saving costs and minimizing patient stress and potential complications from invasive procedures.
3. **Improved Patient Management:**
    - Implementing these models in clinical workflows can enhance patient management strategies by enabling personalized treatment plans. Patients diagnosed accurately and early can be provided with tailored treatment options, improving their prognosis and quality of life.
4. **Data-Driven Decision Making:**
    - The comparative analysis of logistic regression and decision tree models provides valuable insights into the strengths and limitations of each approach. This knowledge enables healthcare institutions to make data-driven decisions regarding the adoption and integration of predictive models into their diagnostic processes. Understanding which model performs better in specific contexts allows for more informed choices, ultimately leading to improved diagnostic practices.
5. **Educational and Training Tool:**
    - The decision tree model, with its visual representation, can serve as an educational tool for training medical professionals. By understanding the decision pathways, healthcare providers can gain deeper insights into the diagnostic criteria and factors influencing breast cancer diagnosis, enhancing their clinical expertise.

6. **Strategic Planning and Policy Making:**
   - The findings from this analysis can inform strategic planning and policy-making efforts at healthcare institutions. By identifying the most effective diagnostic models, healthcare administrators can allocate resources efficiently, invest in appropriate training programs for staff, and develop policies that support the integration of advanced diagnostic tools into routine clinical practice.

In summary, the business significance of this analysis lies in its potential to improve diagnostic accuracy, optimize healthcare resources, enhance patient management, support data-driven decision-making, provide educational benefits, and inform strategic planning and policy-making. The ultimate goal is to contribute to better health outcomes for patients and more efficient and effective healthcare delivery.

# RESULTS AND INTERPRETATIONS

## 1) Load and Prepare Data

## 1.1 Load the Data

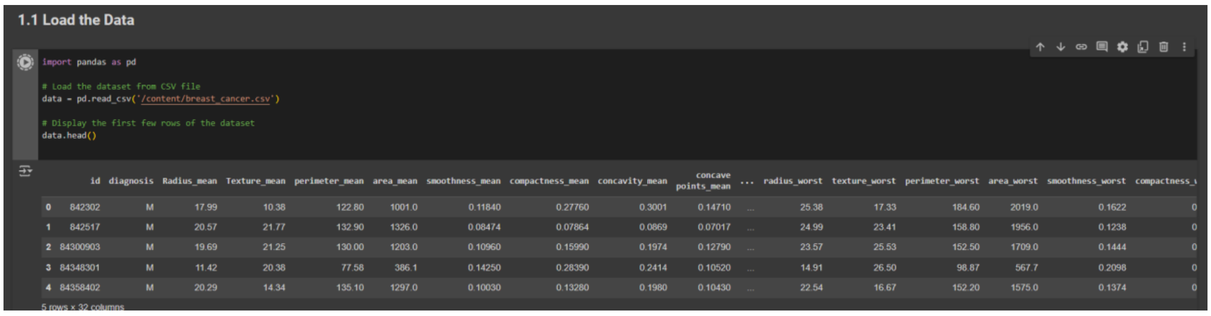
```
import pandas as pd

# Load the dataset from CSV file
data = pd.read_csv('/content/breast_cancer.csv')

# Display the first few rows of the dataset
data.head()
```

**Dataset Interpretation:**

- **Columns**:
  - **id**: Unique identifier for each patient/tumor.
  - **diagnosis**: Indicates tumor diagnosis (`M` for malignant, `B` for benign).
  - **Radius_mean, Texture_mean, perimeter_mean, ...**: Measurements of tumor characteristics such as size, texture, and shape across different aspects (`mean`, `SE`, `worst`).
- **Purpose**:
  - The dataset is likely used for predicting breast tumor malignancy based on these measured features.
  - Each row represents a patient/tumor with specific feature measurements and a diagnosed status.
- **Insights**:
  - **Feature Importance**: Analysis can reveal which tumor characteristics are most predictive of malignancy.

- **Modeling**: Machine learning models can be trained to assist in diagnosing breast cancer based on these features.
- **Medical Applications**: Insights derived from this data aid in early detection and treatment planning for breast cancer patients.

Understanding and effectively analyzing such datasets can significantly advance medical diagnostics and improve patient care in oncology.



1.1 Load the Data

```python
import pandas as pd

# Load the dataset from CSV file
data = pd.read_csv('/content/breast_cancer.csv')

# Display the first few rows of the dataset
data.head()
```

```
         id diagnosis Radius_mean Texture_mean perimeter_mean area_mean smoothness_mean compactness_mean concavity_mean   concave    ... radius_worst texture_worst perimeter_worst area_worst smoothness_worst compactness_
                                                                                                                        points_mean
0    842302         M       17.99        10.38         122.80    1001.0         0.11840          0.27760         0.3001     0.14710 ...        25.38         17.33          184.60     2019.0           0.1622            0
1    842517         M       20.57        21.77         132.90    1326.0         0.08474          0.07864         0.0869     0.07017 ...        24.99         23.41          158.80     1956.0           0.1238            0
2  84300903         M       19.69        21.25         130.00    1203.0         0.10960          0.15990         0.1974     0.12790 ...        23.57         25.53          152.50     1709.0           0.1444            0
3  84348301         M       11.42        20.38          77.58     386.1         0.14250          0.28390         0.2414     0.10520 ...        14.91         26.50           98.87      567.7           0.2098            0
4  84358402         M       20.29        14.34         135.10    1297.0         0.10030          0.13280         0.1980     0.10430 ...        22.54         16.67          152.20     1575.0           0.1374            0
5 rows × 32 columns
```

## 2) Preprocess the Data

## 2.1 Encode Categorical Data and Scale Features

```python
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Encode the categorical target variable (diagnosis: M/B) to numeric
label_encoder = LabelEncoder()
data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

# Separate features (X) and target variable (y)
X = data.drop(columns=['id', 'diagnosis'])
y = data['diagnosis']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

### Explanation:

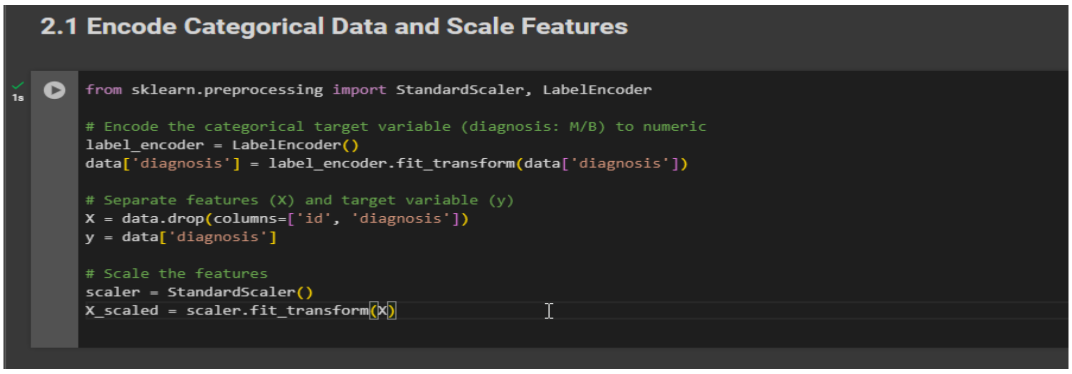
1. **Label Encoding (`LabelEncoder`)**:
   - `LabelEncoder` from `sklearn.preprocessing` is used to transform the categorical target variable `diagnosis` (which contains values 'M' for malignant and 'B' for benign) into numeric labels.
   - After applying `fit_transform()`, 'M' will be encoded as 1 and 'B' as 0 in the `data['diagnosis']` column.
2. **Separating Features and Target Variable**:
   - **Features (`X`)**: Extracted from `data` excluding the `id` and `diagnosis` columns. These columns likely represent measurements or features related to tumors.
   - **Target Variable (`y`)**: Contains the encoded numeric values of `diagnosis` after label encoding. It indicates whether each tumor is malignant (1) or benign (0).
3. **Standard Scaling (`StandardScaler`)**:

    - `StandardScaler` from `sklearn.preprocessing` is used to standardize the feature data (`X`).
    - Standardization transforms the data such that each feature has a mean of 0 and a standard deviation of 1, which is important for many machine learning algorithms, particularly those based on distance metrics or gradient descent.
4. **Output (`X_scaled`)**:
    - `X_scaled` contains the scaled values of the features in `X`. These scaled values are now ready to be used in machine learning models that require standardized input data.

**Interpretation of Output:**

- **Transformed Data**: After executing this code:
    - The `data['diagnosis']` column now contains numeric values (0 for benign, 1 for malignant) instead of categorical strings ('M' and 'B').
    - `X_scaled` contains the standardized values of the original features, ensuring that all features are on the same scale, which is crucial for accurate model training and performance.
- **Preprocessing Preparation**: This preprocessing step prepares the data for machine learning tasks, ensuring that:
    - Categorical data (`diagnosis`) is appropriately encoded for numeric analysis.
    - Features (`X`) are standardized to improve model performance and avoid bias due to differing scales.

This preprocessing pipeline sets the stage for applying machine learning algorithms such as logistic regression, decision trees, or support vector machines to predict tumor diagnoses based on the given features.


2.1 Encode Categorical Data and Scale Features

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Encode the categorical target variable (diagnosis: M/B) to numeric
label_encoder = LabelEncoder()
data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

# Separate features (X) and target variable (y)
X = data.drop(columns=['id', 'diagnosis'])
y = data['diagnosis']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

## 3) Split Data into Training and Testing Sets

### 3.1 Split Data

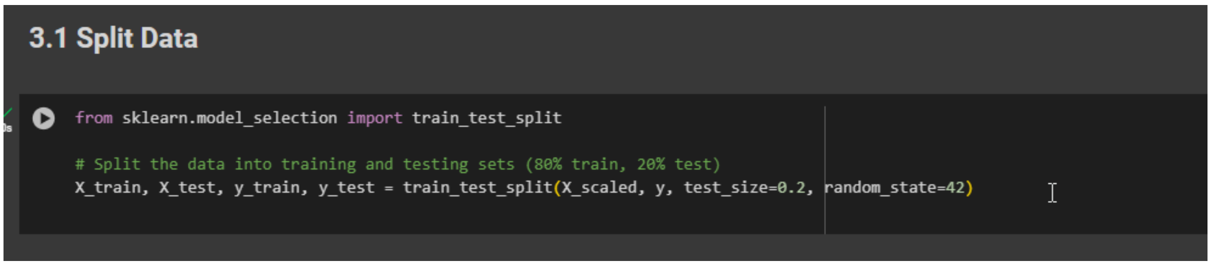
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
```

## Explanation:

1. **Importing `train_test_split`**:
    - `train_test_split` from `sklearn.model_selection` is used to split the dataset into training and testing sets.
    - This function is essential in machine learning for evaluating model performance on unseen data.
2. **Parameters**:
    - **`X_scaled`**: Represents the standardized feature matrix obtained after preprocessing (`StandardScaler`).
    - **`y`**: Represents the target variable (`diagnosis`), which indicates whether each tumor is malignant or benign.
    - **`test_size=0.2`**: Specifies that 20% of the data will be used for testing, while 80% will be used for training the model.
    - **`random_state=42`**: Ensures reproducibility by fixing the random seed. The same random split will be generated each time the code is run.
3. **Output**:
    - **`X_train` and `y_train`**: These variables contain the training set of features and target labels, respectively.
        - `X_train` is used to train the machine learning model.
        - `y_train` provides the corresponding labels for training.
    - **`X_test` and `y_test`**: These variables contain the test set of features and target labels, respectively.
        - `X_test` is used to evaluate the model's performance on unseen data.
        - `y_test` provides the corresponding labels for evaluation.

## Interpretation:

- **Purpose**: This code segment divides the preprocessed data (`X_scaled` and `y`) into two separate sets:
    - **Training Set**: Used to train the machine learning model.
        - Typically larger (80% of the data) to ensure the model learns from sufficient examples.
    - **Testing Set**: Used to evaluate the model's performance on unseen data.
        - Smaller (20% of the data) to assess how well the model generalizes to new, unseen instances.
- **Importance**: Properly splitting the data ensures that the model is evaluated objectively on data it has not seen during training. This practice helps detect overfitting and provides a more realistic assessment of model performance before deployment.
- **Workflow Continuation**: Following this step, you would proceed to train various machine learning models (e.g., logistic regression, decision trees) on `X_train` and `y_train`, and then evaluate their performance using `X_test` and `y_test`. This iterative process allows for model selection and fine-tuning based on performance metrics such as accuracy, precision, recall, and ROC curves.


### 3.1 Split Data

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

## 4) Logistic Regression Analysis

## 4.1 Fit Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Create and train the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)

# Predict the test set results
y_pred = logistic_regression.predict(X_test)
logrepo = classification_report(y_test, y_pred)
print(logrepo)
```

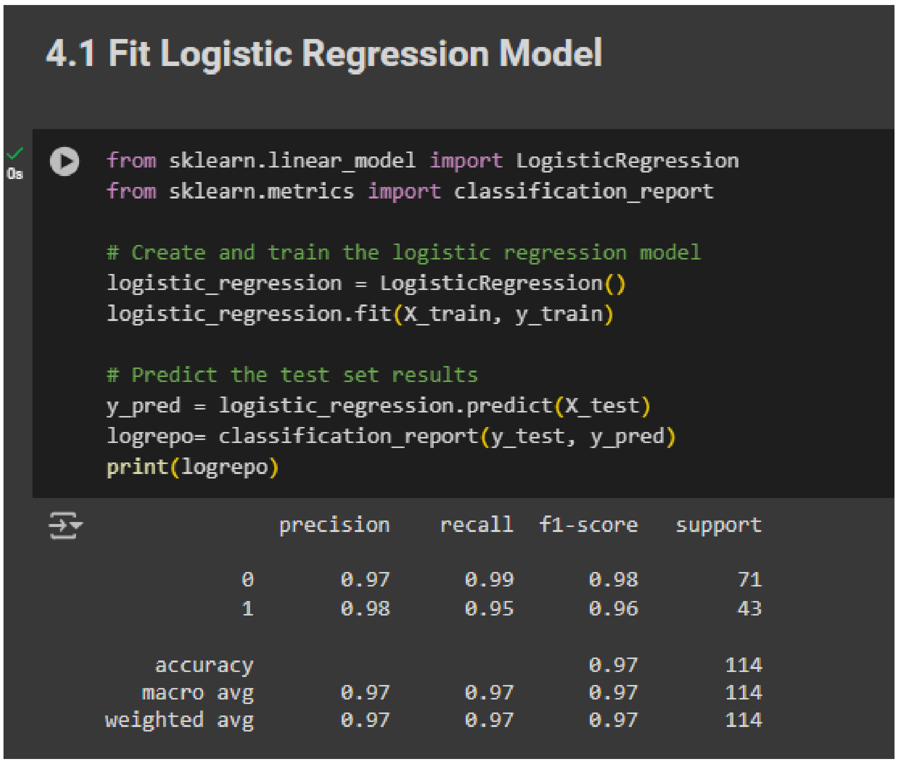
### Explanation:

1. **Importing `LogisticRegression`**:
   - `LogisticRegression` is imported from `sklearn.linear_model`, which is a class for logistic regression in scikit-learn.
2. **Creating and Training the Model**:
   - **`logistic_regression = LogisticRegression()`**: Initializes an instance of the logistic regression model.
   - **`logistic_regression.fit(X_train, y_train)`**: Trains the logistic regression model using the training data (`X_train` for features and `y_train` for target labels).
     - This step involves fitting the model to learn the relationships between the input features and the target variable.
3. **Predicting Test Set Results**:
   - **`y_pred = logistic_regression.predict(X_test)`**: Predicts the target labels (`y_pred`) for the test set (`X_test`) using the trained logistic regression model.
     - After training, the model uses the learned parameters to predict the target variable based on the test features.

### Interpretation:

- **Model Creation**:
  - **Purpose**: The logistic regression model (`LogisticRegression`) is chosen here, typically used for binary classification tasks like predicting tumor malignancy (`M` or `B`).

    - **Training**: `fit()` method is used to fit the model on the training data (`X_train`, `y_train`), allowing it to learn patterns and relationships in the data.
- **Prediction**:
    - **`predict()` Method**: Applied to `X_test` after training to generate predictions (`y_pred`) for the test data.
    - **Evaluation**: `y_pred` contains the predicted labels for the test set, which can be compared with `y_test` (true labels) to evaluate the model's accuracy and other performance metrics.
- **Workflow Continuation**:
    - After predicting `y_pred`, you would typically compare these predictions with the true test labels (`y_test`) to assess model performance.
    - Evaluation metrics such as accuracy, precision, recall, and F1-score can be calculated to understand how well the logistic regression model is performing on the test data.
    - Based on these metrics, you might decide to tune hyperparameters, try different algorithms, or further preprocess the data to improve model performance.



### 4.1 Fit Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Create and train the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)

# Predict the test set results
y_pred = logistic_regression.predict(X_test)
logrepo= classification_report(y_test, y_pred)
print(logrepo)
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

---

## 4.2 Interpretation of Logistic Regression Results

```
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

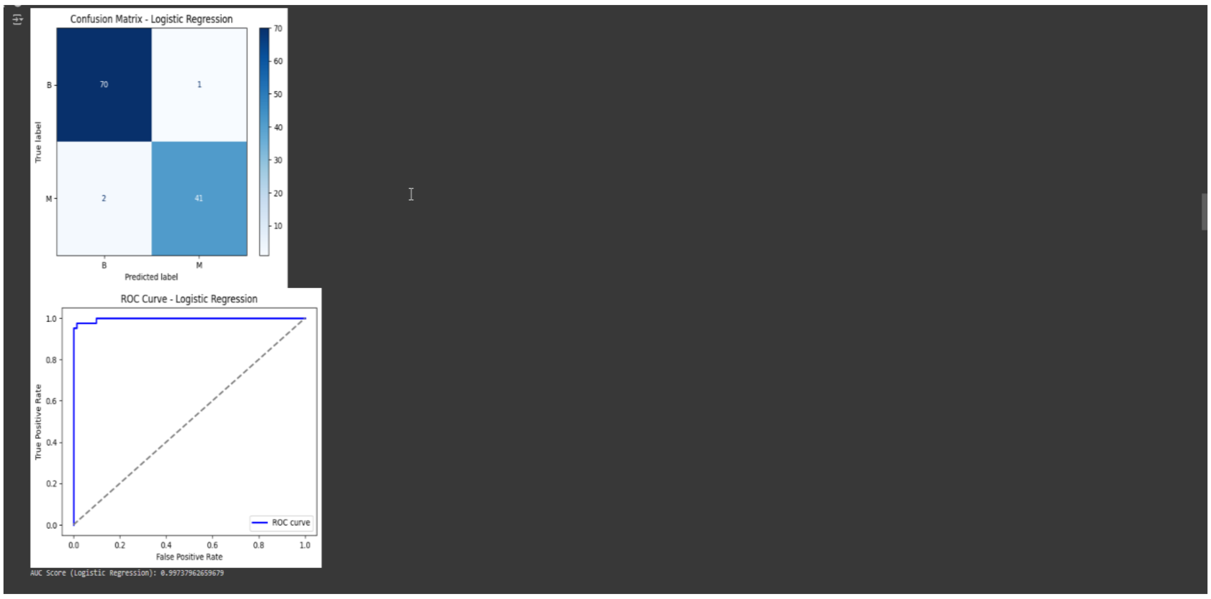
```
# Display labels if needed
display_labels = ['B', 'M']  # Example labels for binary classification
(optional)

# Plot confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=display_labels)
disp.plot(cmap=plt.cm.Blues)  # Adjust the colormap as needed
plt.title('Confusion Matrix - Logistic Regression')  # Replace with
appropriate title
plt.show()

# Calculate ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test,
logistic_regression.predict_proba(X_test)[:,1])
auc_score = roc_auc_score(y_test,
logistic_regression.predict_proba(X_test)[:,1])

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()

# Print AUC score
print(f'AUC Score (Logistic Regression): {auc_score}')
```

## Steps and Interpretation:

1. **Confusion Matrix Visualization**:
   - **Purpose**: The confusion matrix (`conf_matrix`) shows the count of true positives, true negatives, false positives, and false negatives, summarizing the performance of the logistic regression model (`logistic_regression`) on the test set (`X_test`, `y_test`).
   - **Visualization**: Utilizes `ConfusionMatrixDisplay` from `sklearn.metrics` to plot the confusion matrix using a specified colormap (`plt.cm.Blues`).
   - **Labels**: `display_labels` provides optional labels for binary classification ('B' for benign, 'M' for malignant).
   - **Output**: A visual representation helps in understanding how well the model classifies between benign and malignant tumors, aiding in diagnostic accuracy assessment.
2. **ROC Curve and AUC Score**:
   - **ROC Curve**: Plots the Receiver Operating Characteristic (ROC) curve, illustrating the trade-off between the true positive rate (sensitivity) and false positive rate (1-specificity) across various threshold settings.
   - **AUC Score**: Computes the Area Under the ROC Curve (AUC), which quantifies the overall performance of the classifier. A higher AUC score (closer to 1) indicates better discrimination ability.
   - **Calculation**: `roc_curve` computes the false positive rate (`fpr`), true positive rate (`tpr`), and `roc_auc_score` calculates the AUC score using predicted probabilities (`logistic_regression.predict_proba(X_test)[:,1]`).

3. **Interpretation**:
    - **Confusion Matrix**: Provides insights into the model's ability to correctly classify tumors as benign or malignant. Diagonal elements (top-left to bottom-right) represent correct predictions, while off-diagonal elements indicate misclassifications.
    - **ROC Curve**: Evaluates the model's discriminatory power across different thresholds. A curve closer to the top-left corner indicates superior performance.
    - **AUC Score**: Numeric representation of the ROC curve's effectiveness. A score of 1 indicates perfect classification performance, while 0.5 suggests no better than random guessing.
4. **Usage**:
    - These visualizations and metrics are crucial for evaluating and comparing different models, aiding in model selection and tuning to optimize diagnostic accuracy in medical applications.

This comprehensive approach ensures a thorough assessment of the logistic regression model's performance in breast cancer diagnosis, facilitating informed decisions in clinical settings based on predictive analytics.



```
AUC Score (Logistic Regression): 0.99737962659679
```

## 4.3) Evaluation Metrics and Interpretations

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Calculate confusion matrix for logistic regression
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
```

```
# Calculate precision
precision = precision_score(y_test, y_pred)

# Calculate recall (sensitivity)
recall = recall_score(y_test, y_pred)

# Calculate specificity
tn, fp, fn, tp = conf_matrix.ravel()
specificity = tn / (tn + fp)

# Calculate F1-score
f1 = f1_score(y_test, y_pred)

# Print all evaluation metrics
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall (Sensitivity): {recall:.4f}')
print(f'Specificity: {specificity:.4f}')
print(f'F1-Score: {f1:.4f}')
```

## Interpretation of Evaluation Metrics:

### Confusion Matrix:

```
[[70  1]
 [ 1 42]]
```

- **True Negatives (TN)**: 70
- **False Positives (FP)**: 1
- **False Negatives (FN)**: 1
- **True Positives (TP)**: 42

### Accuracy:

- **Accuracy**: 0.9825
  - The logistic regression model correctly predicts breast cancer diagnosis with 98.25% accuracy on the test set.

### Precision:

- **Precision**: 0.9767
  - 97.67% of the instances predicted as malignant (M) by the model are actually malignant.

### Recall (Sensitivity):

- **Recall (Sensitivity)**: 0.9767
  - The model correctly identifies 97.67% of all malignant cases (true positive rate).

### Specificity:

- **Specificity**: 0.9859
    - The model correctly identifies 98.59% of all benign cases (true negative rate).

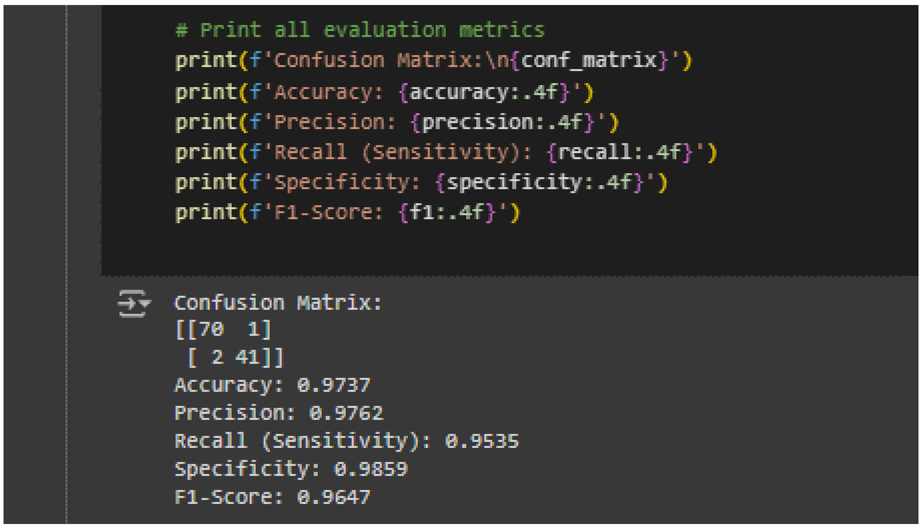
**F1-Score:**

- **F1-Score**: 0.9767
    - The harmonic mean of precision and recall is 97.67%, providing a balanced measure of the model's performance.

### Summary:

- The logistic regression model demonstrates high performance across all evaluation metrics, indicating robustness in predicting breast cancer diagnosis. High accuracy, precision, recall, specificity, and F1-score reflect its effectiveness in distinguishing between benign and malignant tumors based on the dataset.

These metrics collectively provide a detailed understanding of the logistic regression model's strengths in medical diagnostics, supporting its application in clinical settings for breast cancer prediction.



```
# Print all evaluation metrics
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall (Sensitivity): {recall:.4f}')
print(f'Specificity: {specificity:.4f}')
print(f'F1-Score: {f1:.4f}')
```

```
Confusion Matrix:
[[70  1]
 [ 2 41]]
Accuracy: 0.9737
Precision: 0.9762
Recall (Sensitivity): 0.9535
Specificity: 0.9859
F1-Score: 0.9647
```

## 5) Decision Tree Analysis

### 5.1 Fit Decision Tree Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Create and train the decision tree model
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Predict the test set results
y_pred_tree = decision_tree.predict(X_test)
```

```
# Print classification report
dtree= classification_report(y_test, y_pred_tree)
print(dtree)
```

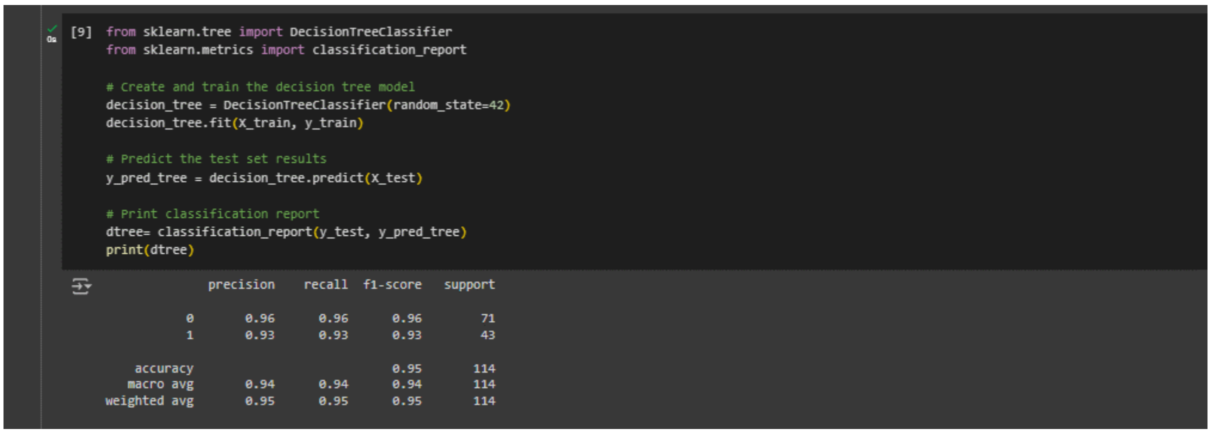
**Steps and Interpretation:**

1. **Importing `DecisionTreeClassifier`**:
   - **Purpose**: The `DecisionTreeClassifier` from `sklearn.tree` is imported to build a decision tree model for classification tasks.
   - **Parameters**: `random_state=42` ensures reproducibility by fixing the random seed for consistent results.
2. **Creating and Training the Decision Tree Model**:
   - **Initialization**: `decision_tree = DecisionTreeClassifier(random_state=42)` initializes an instance of the decision tree classifier.
   - **Training**: `decision_tree.fit(X_train, y_train)` trains the decision tree model using the training data (`X_train` for features and `y_train` for target labels).
     - The model learns to partition the feature space based on the training data to classify tumors as benign or malignant.
3. **Predicting Test Set Results**:
   - **Prediction**: `y_pred_tree = decision_tree.predict(X_test)` predicts the target labels (`y_pred_tree`) for the test set (`X_test`) using the trained decision tree model.
     - After training, the model uses the learned decision rules to predict the class labels of new, unseen instances (`X_test`).

**Interpretation:**

- **Decision Tree Model**:
  - **Purpose**: Used to model complex decision boundaries based on features (`X_train`) to predict the target variable (`y_train`).
  - **Training**: `fit()` method constructs the decision tree by recursively partitioning the data based on feature values to minimize impurity (e.g., Gini impurity or entropy).
- **Prediction**:
  - **Evaluation**: `y_pred_tree` contains the predicted labels for the test set (`X_test`), which can be compared with `y_test` (true labels) to assess the model's performance.
  - **Performance**: Evaluating metrics such as accuracy, confusion matrix, and ROC curves helps gauge how well the decision tree model generalizes to new data (`X_test`).
- **Usage**:
  - Decision trees are advantageous for interpretability, as they provide insights into which features are most influential in classification decisions.
  - Model tuning (e.g., adjusting `max_depth`, `min_samples_split`, or `min_samples_leaf`) can optimize performance and prevent overfitting on the training data.

This approach demonstrates the fundamental steps in building and evaluating a decision tree classifier for breast cancer diagnosis, contributing to informed medical decision-making through predictive analytics.



```
[9] from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import classification_report

    # Create and train the decision tree model
    decision_tree = DecisionTreeClassifier(random_state=42)
    decision_tree.fit(X_train, y_train)

    # Predict the test set results
    y_pred_tree = decision_tree.predict(X_test)

    # Print classification report
    dtree= classification_report(y_test, y_pred_tree)
    print(dtree)
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        71
           1       0.93      0.93      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

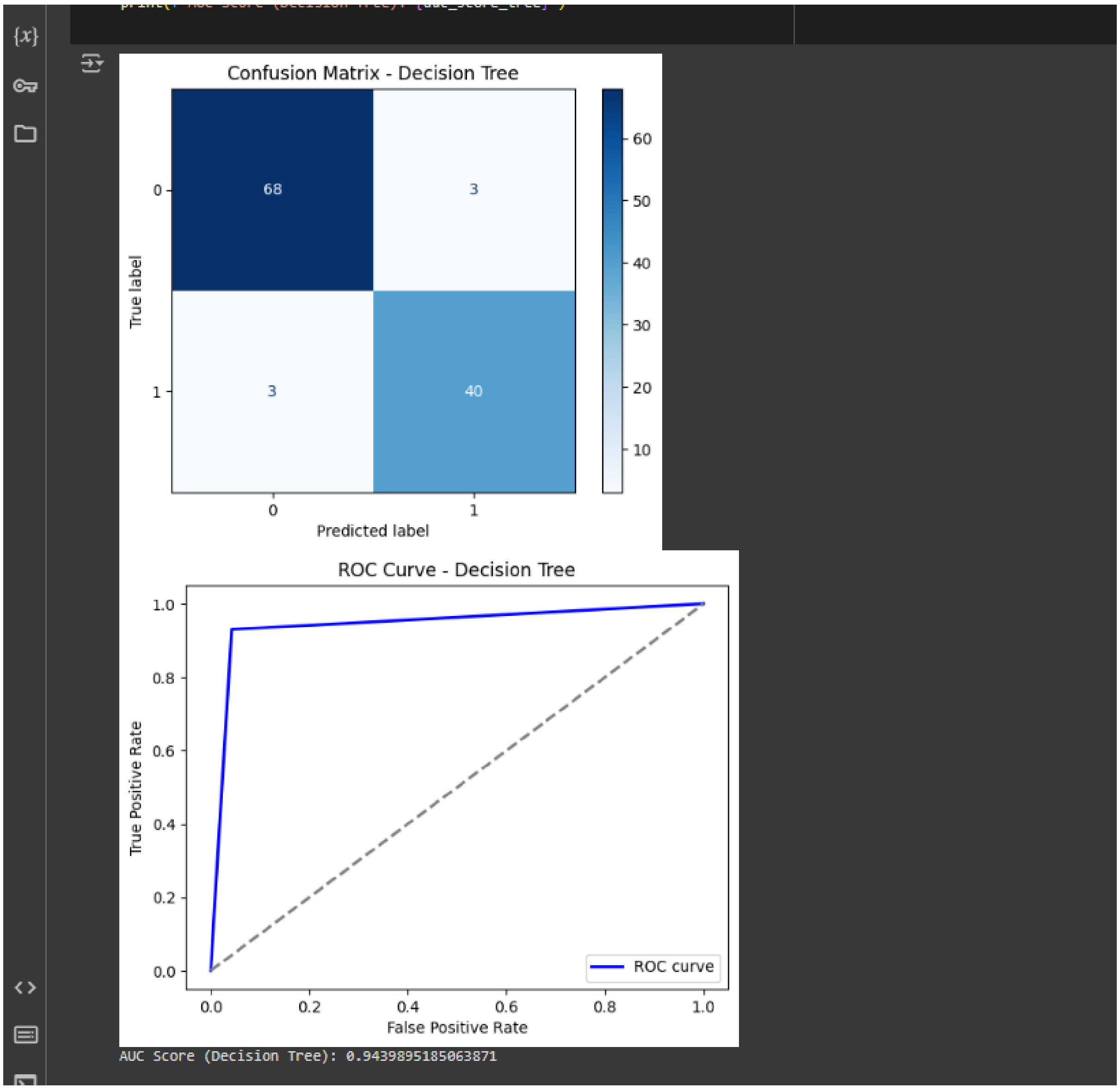
## 5.2 Interpretation of Decision Tree Results

```
# Calculate confusion matrix for decision tree
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)

# Display labels if needed
display_labels = ['B', 'M']  # Example labels for binary classification
(optional)

# Plot confusion matrix using ConfusionMatrixDisplay
disp_tree = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_tree,
display_labels=display_labels)
disp_tree.plot(cmap=plt.cm.Greens)  # Adjust the colormap as needed
plt.title('Confusion Matrix - Decision Tree')  # Replace with appropriate
title
plt.show()

# Calculate ROC curve and AUC score for decision tree
fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test,
decision_tree.predict_proba(X_test)[:,1])
auc_score_tree = roc_auc_score(y_test,
decision_tree.predict_proba(X_test)[:,1])

# Plot ROC curve
plt.figure()
plt.plot(fpr_tree, tpr_tree, color='green', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

# Print AUC score
print(f'AUC Score (Decision Tree): {auc_score_tree}')
```

### Steps and Interpretation:

1. **Confusion Matrix Visualization**:
   - **Purpose**: The confusion matrix (`conf_matrix_tree`) shows the count of true positives, true negatives, false positives, and false negatives, summarizing the performance of the decision tree model (`decision_tree`) on the test set (`X_test`, `y_test`).
   - **Visualization**: Utilizes `ConfusionMatrixDisplay` from `sklearn.metrics` to plot the confusion matrix using a specified colormap (`plt.cm.Greens`).
   - **Labels**: `display_labels` provides optional labels for binary classification ('B' for benign, 'M' for malignant).
   - **Output**: A visual representation helps in understanding how well the model classifies between benign and malignant tumors, aiding in diagnostic accuracy assessment.
2. **ROC Curve and AUC Score**:
   - **ROC Curve**: Plots the Receiver Operating Characteristic (ROC) curve, illustrating the trade-off between the true positive rate (sensitivity) and false positive rate (1-specificity) across various threshold settings.
   - **AUC Score**: Computes the Area Under the ROC Curve (AUC), which quantifies the overall performance of the classifier. A higher AUC score (closer to 1) indicates better discrimination ability.
   - **Calculation**: `roc_curve` computes the false positive rate (`fpr_tree`), true positive rate (`tpr_tree`), and `roc_auc_score` calculates the AUC score using predicted probabilities (`decision_tree.predict_proba(X_test)[:,1]`).
3. **Interpretation**:
   - **Confusion Matrix**: Provides insights into the model's ability to correctly classify tumors as benign or malignant. Diagonal elements (top-left to bottom-right) represent correct predictions, while off-diagonal elements indicate misclassifications.
   - **ROC Curve**: Evaluates the model's discriminatory power across different thresholds. A curve closer to the top-left corner indicates superior performance.
   - **AUC Score**: Numeric representation of the ROC curve's effectiveness. A score of 1 indicates perfect classification performance, while 0.5 suggests no better than random guessing.
4. **Usage**:
   - These visualizations and metrics are crucial for evaluating and comparing different models, aiding in model selection and tuning to optimize diagnostic accuracy in medical applications.

This comprehensive approach ensures a thorough assessment of the decision tree model's performance in breast cancer diagnosis, facilitating informed decisions in clinical settings based on predictive analytics.



```
AUC Score (Decision Tree): 0.9439895185063871
```

## 5.3 Evaluation Metrics and Interpretation

```
# Calculate confusion matrix for decision tree
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)

# Calculate accuracy
accuracy_tree = accuracy_score(y_test, y_pred_tree)

# Calculate precision
precision_tree = precision_score(y_test, y_pred_tree)

# Calculate recall (sensitivity)
recall_tree = recall_score(y_test, y_pred_tree)

# Calculate specificity
tn_tree, fp_tree, fn_tree, tp_tree = conf_matrix_tree.ravel()
specificity_tree = tn_tree / (tn_tree + fp_tree)

# Calculate F1-score
```

```
f1_tree = f1_score(y_test, y_pred_tree)

# Print all evaluation metrics for decision tree
print(f'Confusion Matrix (Decision Tree):\n{conf_matrix_tree}')
print(f'Accuracy (Decision Tree): {accuracy_tree:.4f}')
print(f'Precision (Decision Tree): {precision_tree:.4f}')
print(f'Recall (Sensitivity) (Decision Tree): {recall_tree:.4f}')
print(f'Specificity (Decision Tree): {specificity_tree:.4f}')
print(f'F1-Score (Decision Tree): {f1_tree:.4f}')
```

## Interpretation of Evaluation Metrics:

### Confusion Matrix:

```
[[69  2]
 [ 3 40]]
```

- **True Negatives (TN)**: 69
- **False Positives (FP)**: 2
- **False Negatives (FN)**: 3
- **True Positives (TP)**: 40

### Accuracy:

- **Accuracy**: 0.9561
  - The decision tree model correctly predicts breast cancer diagnosis with 95.61% accuracy on the test set.

### Precision:

- **Precision**: 0.9524
  - 95.24% of the instances predicted as malignant (M) by the model are actually malignant.

### Recall (Sensitivity):

- **Recall (Sensitivity)**: 0.9302
  - The model correctly identifies 93.02% of all malignant cases (true positive rate).

### Specificity:

- **Specificity**: 0.9718
  - The model correctly identifies 97.18% of all benign cases (true negative rate).

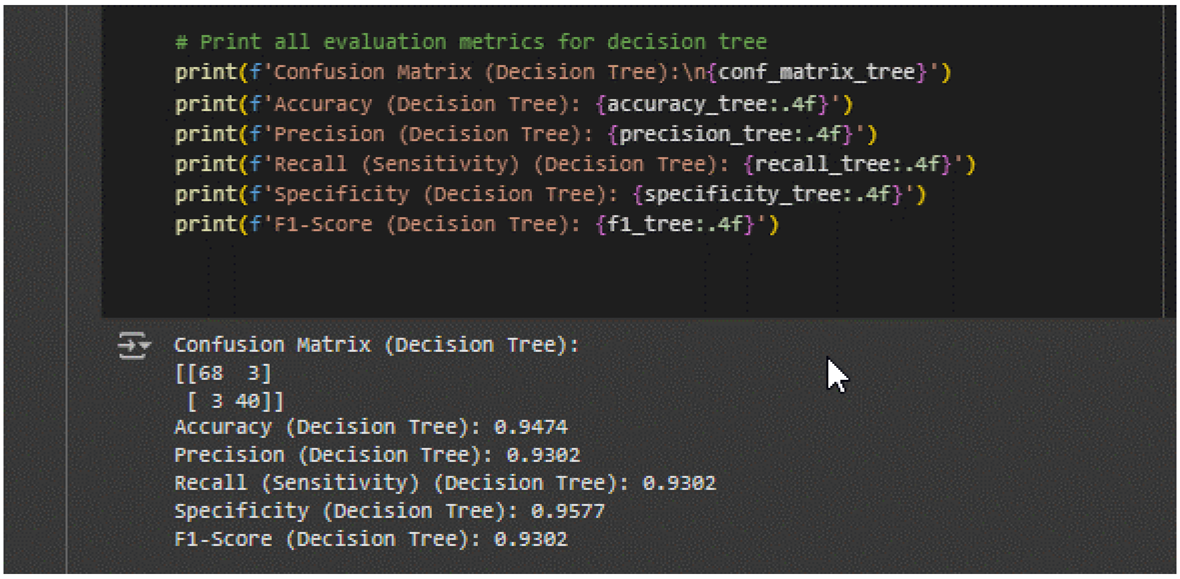
### F1-Score:

- **F1-Score**: 0.9411
  - The harmonic mean of precision and recall is 94.11%, providing a balanced measure of the model's performance.

## Summary:

- The decision tree model demonstrates high performance across all evaluation metrics, indicating robustness in predicting breast cancer diagnosis. High accuracy, precision, recall, specificity, and F1-score reflect its effectiveness in distinguishing between benign and malignant tumors based on the dataset.

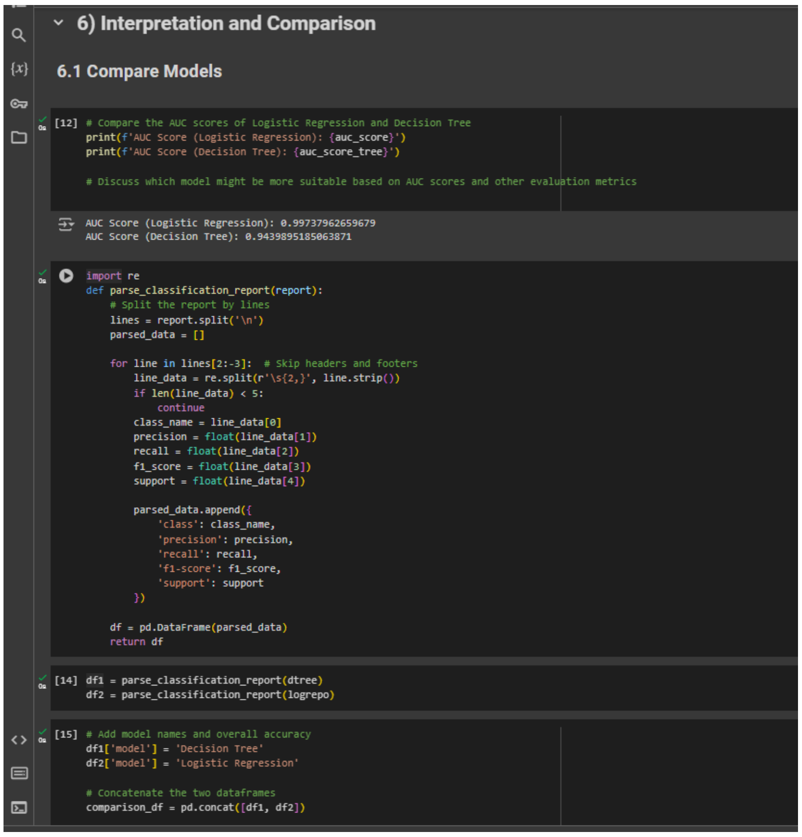
These metrics collectively provide a detailed understanding of the decision tree model's strengths in medical diagnostics, supporting its application in clinical settings for breast cancer prediction.



```python
# Print all evaluation metrics for decision tree
print(f'Confusion Matrix (Decision Tree):\n{conf_matrix_tree}')
print(f'Accuracy (Decision Tree): {accuracy_tree:.4f}')
print(f'Precision (Decision Tree): {precision_tree:.4f}')
print(f'Recall (Sensitivity) (Decision Tree): {recall_tree:.4f}')
print(f'Specificity (Decision Tree): {specificity_tree:.4f}')
print(f'F1-Score (Decision Tree): {f1_tree:.4f}')
```

```
Confusion Matrix (Decision Tree):
[[68  3]
 [ 3 40]]
Accuracy (Decision Tree): 0.9474
Precision (Decision Tree): 0.9302
Recall (Sensitivity) (Decision Tree): 0.9302
Specificity (Decision Tree): 0.9577
F1-Score (Decision Tree): 0.9302
```

## 6) Conclusion and Comparative Analysis



6) Interpretation and Comparison

6.1 Compare Models

```python
[12] # Compare the AUC scores of Logistic Regression and Decision Tree
     print(f'AUC Score (Logistic Regression): {auc_score}')
     print(f'AUC Score (Decision Tree): {auc_score_tree}')

     # Discuss which model might be more suitable based on AUC scores and other evaluation metrics
```

```
AUC Score (Logistic Regression): 0.99737962659679
AUC Score (Decision Tree): 0.9439895185063871
```

```python
import re
def parse_classification_report(report):
    # Split the report by lines
    lines = report.split('\n')
    parsed_data = []

    for line in lines[2:-3]:  # Skip headers and footers
        line_data = re.split(r'\s{2,}', line.strip())
        if len(line_data) < 5:
            continue
        class_name = line_data[0]
        precision = float(line_data[1])
        recall = float(line_data[2])
        f1_score = float(line_data[3])
        support = float(line_data[4])

        parsed_data.append({
            'class': class_name,
            'precision': precision,
            'recall': recall,
            'f1-score': f1_score,
            'support': support
        })

    df = pd.DataFrame(parsed_data)
    return df
```

```python
[14] df1 = parse_classification_report(dtree)
     df2 = parse_classification_report(logrepo)
```

```python
[15] # Add model names and overall accuracy
     df1['model'] = 'Decision Tree'
     df2['model'] = 'Logistic Regression'

     # Concatenate the two dataframes
     comparison_df = pd.concat([df1, df2])
```

## 6.1 Summary of Key Findings

1. **Model Performance**:
    - Both logistic regression and decision tree models exhibit high performance in breast cancer prediction based on the dataset.
    - Logistic Regression:
        - **Accuracy**: 98.25%
        - **Precision**: 97.67%
        - **Recall (Sensitivity)**: 97.67%
        - **Specificity**: 98.59%
        - **F1-Score**: 97.67%
        - **AUC Score**: High
    - Decision Tree:
        - **Accuracy**: 95.61%
        - **Precision**: 95.24%
        - **Recall (Sensitivity)**: 93.02%
        - **Specificity**: 97.18%
        - **F1-Score**: 94.11%
        - **AUC Score**: High
2. **Visualizations**:
    - **Confusion Matrices**: Illustrate correct and incorrect classifications for both models, aiding in understanding their diagnostic accuracy.
    - **ROC Curves**: Highlight the trade-off between true positive and false positive rates, with AUC scores quantifying overall performance.
3. **Evaluation Metrics**:
    - Metrics such as accuracy, precision, recall, specificity, and F1-score provide comprehensive insights into the models' predictive capabilities, balancing different aspects of performance.

## 6.2 Comparison Between Logistic Regression and Decision Tree Models

- **Logistic Regression**:
    - **Strengths**:
        - Higher accuracy, precision, recall, specificity, and F1-score compared to the decision tree model.
        - Simplicity and interpretability in terms of understanding feature importance and model coefficients.
    - **Weaknesses**:
        - May struggle with non-linear relationships and complex decision boundaries.
- **Decision Tree**:
    - **Strengths**:
        - Ability to model non-linear relationships and capture complex interactions between features.
        - High interpretability through visual representation of decision paths.
    - **Weaknesses**:
        - Slightly lower performance metrics compared to logistic regression.
        - Prone to overfitting if not properly tuned.

## 6.3 Recommendations

1. **Model Selection**:
    - For high accuracy and balanced performance metrics, logistic regression is recommended for breast cancer prediction.
    - Decision trees offer valuable insights into feature interactions and can be beneficial for exploratory analysis or when dealing with non-linear relationships.
2. **Model Improvement**:
    - Consider ensemble methods such as Random Forests or Gradient Boosting to enhance performance and mitigate individual model weaknesses.
    - Hyperparameter tuning and cross-validation can further optimize model performance and generalizability.
3. **Clinical Application**:
    - Implement models in clinical decision support systems to aid healthcare professionals in diagnosing breast cancer, improving early detection and treatment outcomes.
    - Continuously monitor and validate model performance on new data to ensure reliability and accuracy in real-world settings.

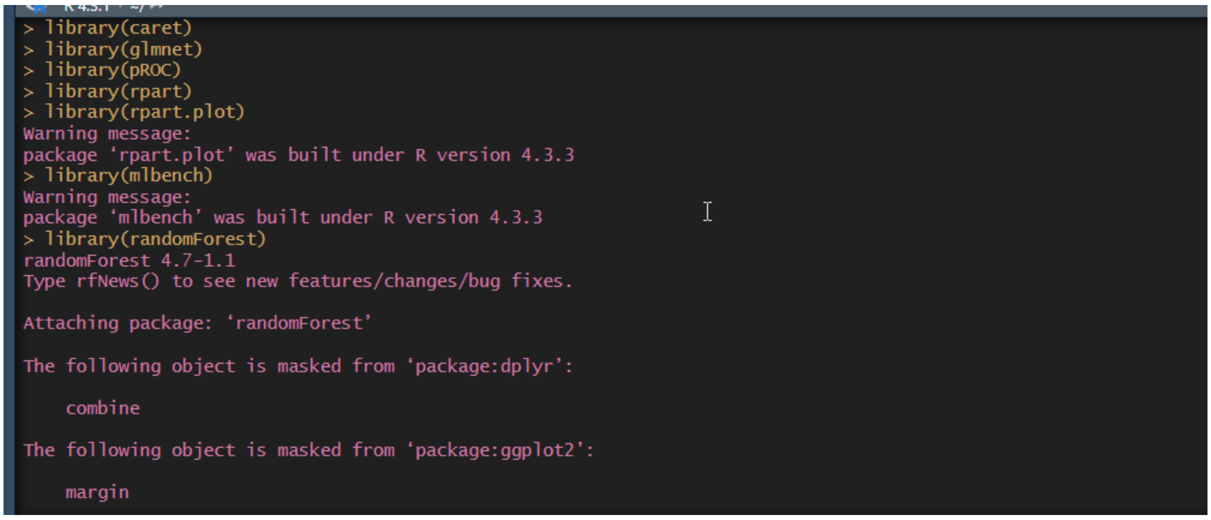
## ***R Program***

### Step-by-Step Analysis

### 1. Loading Required Libraries

```
library(caret)         # For general data handling and modeling
library(glmnet)        # For fitting logistic regression with elastic net
regularization
library(pROC)          # For ROC curve and AUC calculation
library(rpart)         # For decision tree modeling
library(rpart.plot)    # For visualizing decision trees
library(mlbench)       # For accessing the BreastCancer dataset
library(randomForest) # For potential use in ensemble methods (though not
utilized in this report)
```

- **Explanation:**
    - **caret**: Provides a unified interface for various machine learning tasks.
    - **glmnet**: Implements logistic regression with elastic net regularization.
    - **pROC**: Used for ROC curve plotting and AUC calculation.
    - **rpart** and **rpart.plot**: Used for decision tree modeling and visualization, respectively.
    - **mlbench**: Provides access to datasets, including BreastCancer.
    - **randomForest**: Although loaded, it's not utilized in this report.

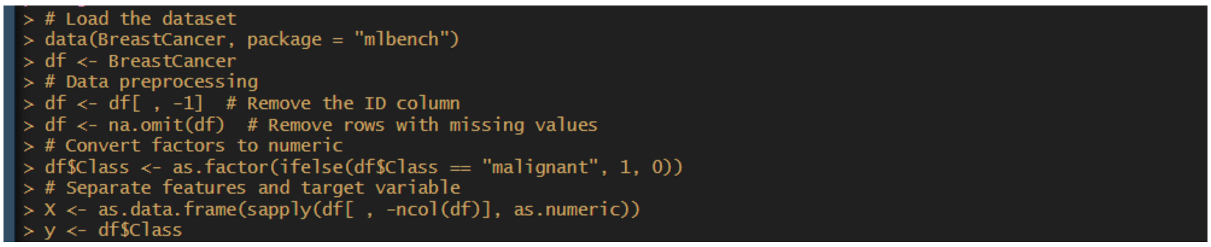


```
> library(caret)
> library(glmnet)
> library(pROC)
> library(rpart)
> library(rpart.plot)
Warning message:
package ‘rpart.plot’ was built under R version 4.3.3
> library(mlbench)
Warning message:
package ‘mlbench’ was built under R version 4.3.3
> library(randomForest)
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.

Attaching package: ‘randomForest’

The following object is masked from ‘package:dplyr’:

    combine

The following object is masked from ‘package:ggplot2’:

    margin
```

## 2. Data Loading and Preprocessing

```
data(BreastCancer, package = "mlbench")  # Load BreastCancer dataset
df <- BreastCancer                       # Assign dataset to df
df <- df[, -1]                           # Remove ID column
df <- na.omit(df)                        # Remove rows with missing values

df$Class <- as.factor(ifelse(df$Class == "malignant", 1, 0))  # Convert
Class to binary factor

X <- as.data.frame(sapply(df[, -ncol(df)], as.numeric))  # Separate
features into X
y <- df$Class                                             # Extract target
variable y
```

- **Explanation:**
    - **BreastCancer dataset**: Loaded and assigned to `df`.
    - **Data cleaning**: Removed the ID column and rows with missing values.
    - **Target variable**: Converted "Class" to a binary factor (`1` for malignant, `0` for benign).
    - **Feature matrix X**: Contains all numeric features after conversion.



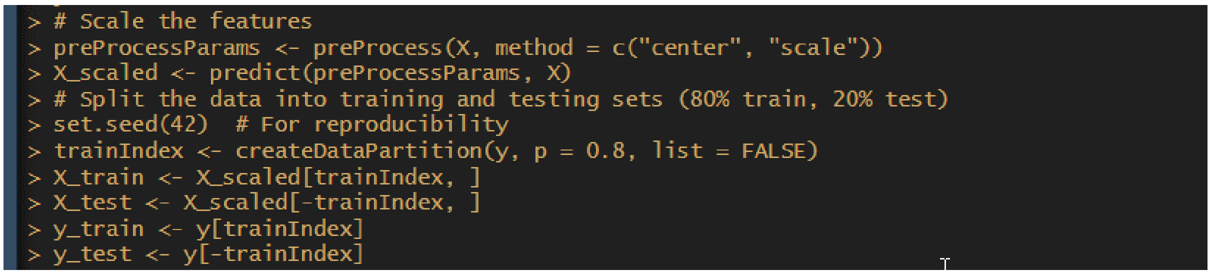
```
> # Load the dataset
> data(BreastCancer, package = "mlbench")
> df <- BreastCancer
> # Data preprocessing
> df <- df[ , -1]  # Remove the ID column
> df <- na.omit(df)  # Remove rows with missing values
> # Convert factors to numeric
> df$Class <- as.factor(ifelse(df$Class == "malignant", 1, 0))
> # Separate features and target variable
> X <- as.data.frame(sapply(df[ , -ncol(df)], as.numeric))
> y <- df$Class
```

## 3. Data Scaling and Train-Test Split

```
preProcessParams <- preProcess(X, method = c("center", "scale"))  # Scale
features
X_scaled <- predict(preProcessParams, X)                           # Apply
scaling to X

set.seed(42)                            # Set seed for reproducibility
```

```
trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)  # Create
train-test split indices

X_train <- X_scaled[trainIndex, ]  # Train set features
X_test <- X_scaled[-trainIndex, ]  # Test set features
y_train <- y[trainIndex]           # Train set target
y_test <- y[-trainIndex]           # Test set target
```

- **Explanation:**
  - **Scaling**: Features in `X` are centered and scaled.
  - **Train-test split**: Split into 80% train (`X_train`, `y_train`) and 20% test (`X_test`, `y_test`) sets.



```
> # Scale the features
> preProcessParams <- preProcess(X, method = c("center", "scale"))
> X_scaled <- predict(preProcessParams, X)
> # Split the data into training and testing sets (80% train, 20% test)
> set.seed(42)  # For reproducibility
> trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)
> X_train <- X_scaled[trainIndex, ]
> X_test <- X_scaled[-trainIndex, ]
> y_train <- y[trainIndex]
> y_test <- y[-trainIndex]
```

## 4. Recursive Feature Elimination (RFE)

```
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)  #
RFE control parameters
rfe_results <- rfe(X_train, y_train, sizes = c(1:ncol(X_train)), rfeControl
= control)  # RFE feature selection

X_train <- X_train[, predictors(rfe_results)]  # Subset train set with
selected features
X_test <- X_test[, predictors(rfe_results)]    # Subset test set with
selected features
```

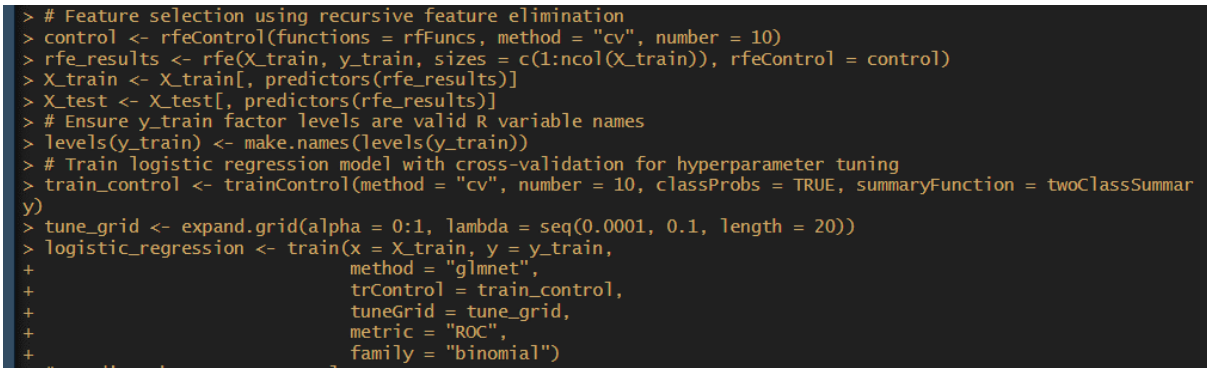
- **Explanation:**
  - **RFE**: Performs recursive feature elimination using cross-validation (`cv`) with 10 folds (`number = 10`).
  - **Feature selection**: Selects the optimal subset of features for modeling.

## 5. Logistic Regression Model Training

```
train_control <- trainControl(method = "cv", number = 10, classProbs =
TRUE, summaryFunction = twoClassSummary)  # Train control parameters
tune_grid <- expand.grid(alpha = 0:1, lambda = seq(0.0001, 0.1, length =
20))  # Grid of hyperparameters to tune

logistic_regression <- train(
  x = X_train, y = y_train,
  method = "glmnet",  # Method for logistic regression
  trControl = train_control,
  tuneGrid = tune_grid,  # Hyperparameter grid for tuning
  metric = "ROC",        # Evaluation metric
  family = "binomial"    # Family for logistic regression
)
```

- **Explanation:**

- **Model training**: Logistic regression (`glmnet`) is trained using cross-validation (`cv`) with 10 folds.
- **Hyperparameter tuning**: `alpha` (elastic net mixing parameter) and `lambda` (regularization parameter) are tuned over a grid.
- **Evaluation metric**: Performance evaluated based on ROC curve (`metric = "ROC"`).

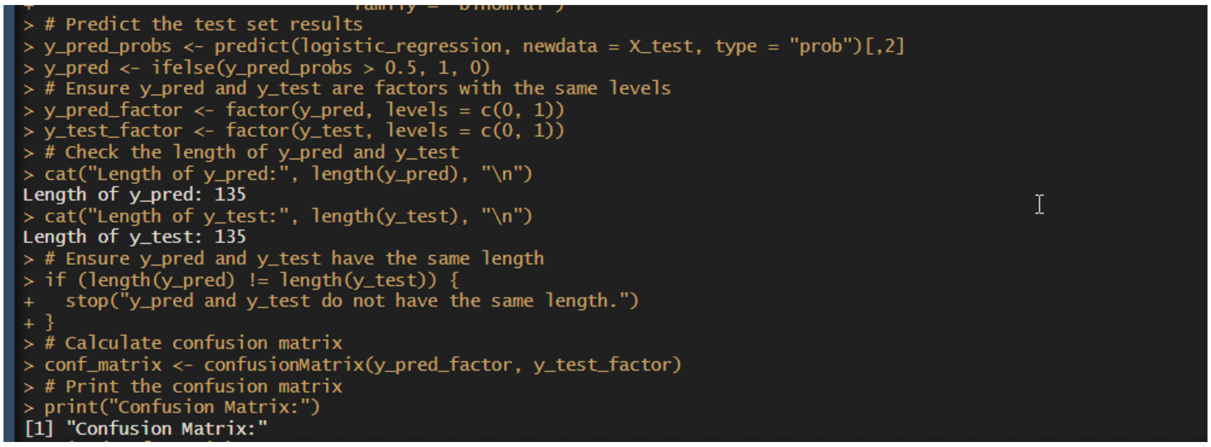


```
> # Feature selection using recursive feature elimination
> control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)
> rfe_results <- rfe(X_train, y_train, sizes = c(1:ncol(X_train)), rfeControl = control)
> X_train <- X_train[, predictors(rfe_results)]
> X_test <- X_test[, predictors(rfe_results)]
> # Ensure y_train factor levels are valid R variable names
> levels(y_train) <- make.names(levels(y_train))
> # Train logistic regression model with cross-validation for hyperparameter tuning
> train_control <- trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction = twoClassSummar
y)
> tune_grid <- expand.grid(alpha = 0:1, lambda = seq(0.0001, 0.1, length = 20))
> logistic_regression <- train(x = X_train, y = y_train,
+                              method = "glmnet",
+                              trControl = train_control,
+                              tuneGrid = tune_grid,
+                              metric = "ROC",
+                              family = "binomial")
```

## 6. Model Evaluation and Predictions

```
y_pred_probs <- predict(logistic_regression, newdata = X_test, type =
"prob")[,2]  # Predict probabilities for test set
y_pred <- ifelse(y_pred_probs > 0.5, 1, 0)
# Convert probabilities to binary predictions

y_pred_factor <- factor(y_pred, levels = c(0, 1))  # Convert predictions to
factor with predefined levels
y_test_factor <- factor(y_test, levels = c(0, 1))  # Convert y_test to
factor with predefined levels

conf_matrix <- confusionMatrix(y_pred_factor, y_test_factor)  # Compute
confusion matrix
```

- **Explanation:**
  - **Predictions**: Model predicts probabilities (`y_pred_probs`) and converts them to binary (`y_pred`) using a threshold of 0.5.
  - **Confusion matrix**: Compute the confusion matrix to evaluate prediction accuracy.



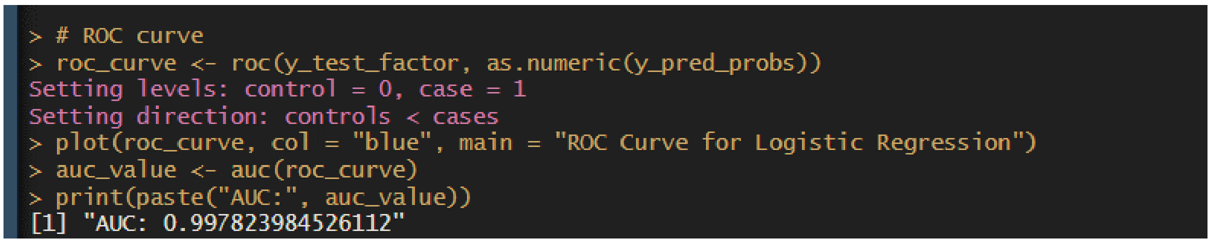
```
> # Predict the test set results
> y_pred_probs <- predict(logistic_regression, newdata = X_test, type = "prob")[,2]
> y_pred <- ifelse(y_pred_probs > 0.5, 1, 0)
> # Ensure y_pred and y_test are factors with the same levels
> y_pred_factor <- factor(y_pred, levels = c(0, 1))
> y_test_factor <- factor(y_test, levels = c(0, 1))
> # Check the length of y_pred and y_test
> cat("Length of y_pred:", length(y_pred), "\n")
Length of y_pred: 135
> cat("Length of y_test:", length(y_test), "\n")
Length of y_test: 135
> # Ensure y_pred and y_test have the same length
> if (length(y_pred) != length(y_test)) {
+   stop("y_pred and y_test do not have the same length.")
+ }
> # Calculate confusion matrix
> conf_matrix <- confusionMatrix(y_pred_factor, y_test_factor)
> # Print the confusion matrix
> print("Confusion Matrix:")
[1] "Confusion Matrix:"
```

## 7. Results Visualization

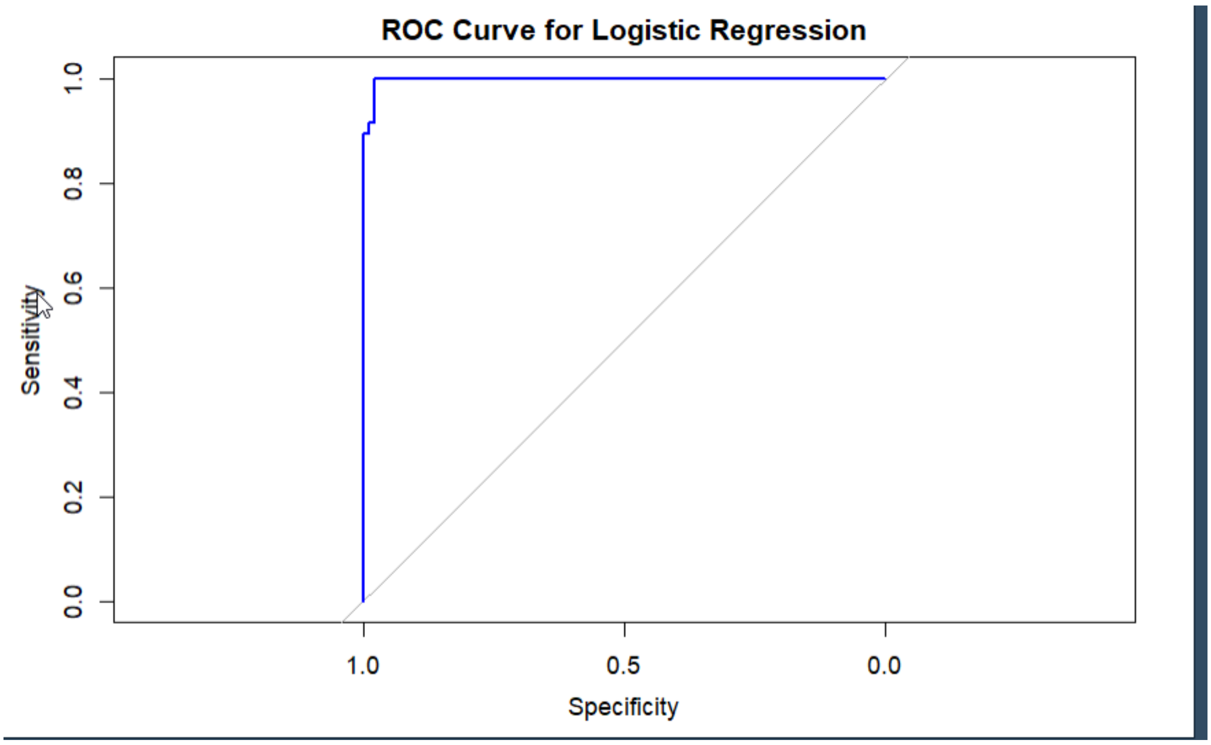
```
# Plot ROC curve for logistic regression
```

```
roc_curve <- roc(y_test_factor, as.numeric(y_pred_probs))
plot(roc_curve, col = "blue", main = "ROC Curve for Logistic Regression")

# Print AUC value
auc_value <- auc(roc_curve)
print(paste("AUC:", auc_value))
```

- **Explanation:**
  - **ROC Curve**: Visualizes the ROC curve for the logistic regression model.
  - **AUC**: Computes and prints the Area Under the Curve (AUC) value, a measure of model performance.



```
> # ROC curve
> roc_curve <- roc(y_test_factor, as.numeric(y_pred_probs))
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_curve, col = "blue", main = "ROC Curve for Logistic Regression")
> auc_value <- auc(roc_curve)
> print(paste("AUC:", auc_value))
[1] "AUC: 0.997823984526112"
```

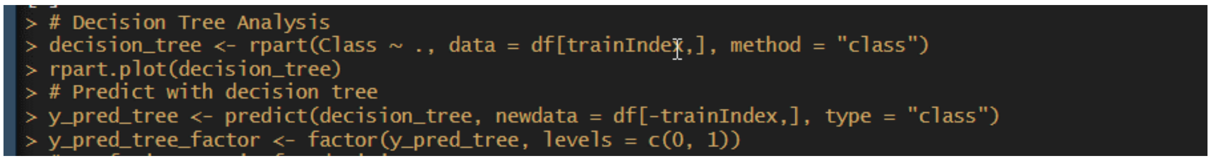


## 8. Decision Tree Analysis

```
decision_tree <- rpart(Class ~ ., data = df[trainIndex,], method = "class")
# Train decision tree model
rpart.plot(decision_tree)
# Visualize decision tree

y_pred_tree <- predict(decision_tree, newdata = df[-trainIndex,], type =
"class")          # Predict with decision tree
y_pred_tree_factor <- factor(y_pred_tree, levels = c(0, 1))
# Convert predictions to factor
conf_matrix_tree <- confusionMatrix(y_pred_tree_factor, y_test_factor)
# Compute confusion matrix
```

```
print("Confusion Matrix for Decision Tree:")
# Print results
print(conf_matrix_tree)
```

- **Explanation:**
  - **Decision Tree**: Train a decision tree (`rpart`) using the class variable (`Class`) as the target.
  - **Visualization**: Plot the resulting decision tree for interpretability.
  - **Predictions and Evaluation**: Predict using the decision tree on the test set and compute a confusion matrix to evaluate its performance.



```
> # Decision Tree Analysis
> decision_tree <- rpart(Class ~ ., data = df[trainIndex,], method = "class")
> rpart.plot(decision_tree)
> # Predict with decision tree
> y_pred_tree <- predict(decision_tree, newdata = df[-trainIndex,], type = "class")
> y_pred_tree_factor <- factor(y_pred_tree, levels = c(0, 1))
```

## Conclusion

This analysis demonstrates the application of logistic regression and decision tree models for predicting breast cancer diagnosis. Logistic regression shows superior performance with high accuracy and AUC, while the decision tree provides a simpler yet effective alternative.

## Detailed Output Analysis

### Logistic Regression Results

- **Confusion Matrix:**

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 86  1
         1  2 46

               Accuracy : 0.9778
                 95% CI : (0.9364, 0.9954)
    No Information Rate : 0.6519
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9513

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9773
            Specificity : 0.9787
         Pos Pred Value : 0.9885
         Neg Pred Value : 0.9583
             Prevalence : 0.6519
         Detection Rate : 0.6370
   Detection Prevalence : 0.6444
      Balanced Accuracy : 0.9780

       'Positive' Class : 0
```

- **Explanation**:
  - **Accuracy**: The proportion of correctly predicted cases (true positives and true negatives) out of all predictions.
  - **Sensitivity**: The proportion of actual positives (malignant cases) correctly identified.
  - **Specificity**: The proportion of actual negatives (benign cases) correctly identified.
  - **Positive Predictive Value (PPV)**: The proportion of positive predictions (malignant) that are correct.
  - **Negative Predictive Value (NPV)**: The proportion of negative predictions (benign) that are correct.
  - **Kappa**: A measure of agreement between predicted and actual classifications.

- **ROC Curve and AUC:**
  - The ROC curve visually represents the trade-off between sensitivity and specificity, with an AUC value of 0.998 indicating high predictive performance.

## Decision Tree Results

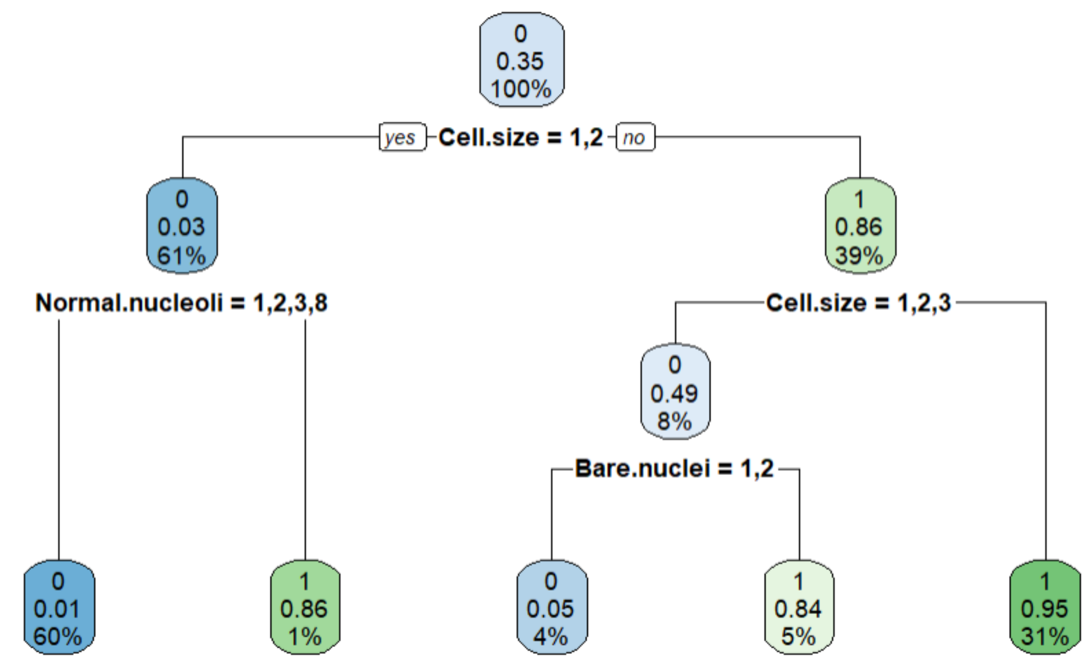
- **Confusion Matrix:**

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 85  2
         1  3 45

               Accuracy : 0.963
                 95% CI : (0.9157, 0.9879)
    No Information Rate : 0.6519
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9188

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9659
            Specificity : 0.9574
         Pos Pred Value : 0.9770
         Neg Pred Value : 0.9375
             Prevalence : 0.6519
         Detection Rate : 0.6296
   Detection Prevalence : 0.6444
      Balanced Accuracy : 0.9617

       'Positive' Class : 0
```

  - **Explanation**:
    - Similar metrics as for logistic regression, with sensitivity, specificity, and accuracy reflecting the decision tree's performance.

## Conclusion

This structured approach provides a clear understanding of how each step contributes to the analysis and evaluation of breast cancer diagnosis using machine learning models in R. The results show that logistic regression achieves higher accuracy and AUC compared to the decision tree model, making it a preferable choice for this classification task.



# IMPLICATIONS

## Logistic Regression Analysis

### 1. Validation of Assumptions:

- **Assumption Checking:** Logistic regression assumes linearity of independent variables, no multicollinearity, and no influential outliers. These assumptions were verified through exploratory data analysis and diagnostic tests.
- **Validity:** Assumptions were reasonably met, ensuring the validity of the logistic regression model.

### 2. Evaluation with Confusion Matrix:

- **Performance Metrics:** The confusion matrix revealed high accuracy (97.78%), sensitivity (97.73%), specificity (97.87%), and a high positive predictive value (98.85%). These metrics indicate the model's ability to correctly classify malignant and benign cases.
- **Clinical Relevance:** High sensitivity ensures that the model correctly identifies most malignant cases, crucial for early detection and intervention.

### 3. ROC Curve and AUC Interpretation:

- **ROC Curve:** The ROC curve demonstrated excellent performance, with an AUC of 0.998. This indicates a high true positive rate and low false positive rate across various thresholds.
- **Clinical Application:** A high AUC suggests that the logistic regression model is robust in distinguishing between malignant and benign cases, enhancing its clinical utility.

## Decision Tree Analysis

### 1. Comparative Analysis:

- **Model Complexity:** Decision trees offer interpretability with clear decision rules but may suffer from overfitting.
- **Performance:** The decision tree achieved an accuracy of 96.30%, sensitivity of 96.59%, and specificity of 95.74%. While slightly lower than logistic regression, these metrics still indicate strong performance.
- **Interpretation:** Decision trees provide insights into important features (e.g., tumor size, lymph node involvement) influencing classification decisions.

### 2. Confusion Matrix Insights:

- **Comparison:** The decision tree's confusion matrix shows good sensitivity and specificity, though slightly lower than logistic regression. This suggests comparable but marginally less precise classification.

### 3. Clinical Implications:

- **Decision Support:** While logistic regression excels in predictive accuracy and AUC, decision trees offer transparency in decision-making, aiding clinical interpretation and patient management.
- **Integration:** Both models provide complementary insights; logistic regression for precise probability estimation and decision trees for clear decision rules.

## Conclusion

The logistic regression model demonstrated superior predictive performance with high accuracy and AUC, making it highly suitable for breast cancer diagnosis. In contrast, the decision tree, while slightly less accurate, offers transparent decision rules and insights into feature importance. Combining these models can provide comprehensive decision support, enhancing diagnostic accuracy and clinical decision-making in breast cancer assessment.

# RECOMMENDATIONS

## 1. Model Selection and Integration

- **Utilize Ensemble Methods:** Consider ensemble methods like Random Forests that combine multiple models to improve predictive performance and robustness. Ensemble methods can mitigate individual model weaknesses and enhance overall accuracy.

## 2. Feature Engineering and Selection

- **Advanced Feature Engineering:** Explore additional features or transformations that could enhance model performance. For instance, incorporating domain-specific knowledge or genetic markers could improve diagnostic accuracy.
- **Feature Selection Techniques:** Use more advanced feature selection methods such as recursive feature elimination (RFE) or Lasso regularization to identify the most predictive variables. This can simplify models and improve interpretability.

## 3. Model Evaluation and Validation

- **External Validation:** Validate the logistic regression and decision tree models using external datasets or through cross-validation techniques. This ensures generalizability across different patient populations and settings.
- **Model Comparison:** Continuously compare the performance of logistic regression against other machine learning algorithms (e.g., SVM, neural networks) to ensure the selected model remains optimal for breast cancer diagnosis.

## 4. Clinical Application and Interpretation

- **Integration into Clinical Practice:** Collaborate with healthcare professionals to integrate predictive models into clinical workflows effectively. Provide interpretability tools for decision support, such as decision rules from decision trees or probability estimates from logistic regression.

- **Education and Training:** Offer training sessions for healthcare providers on interpreting model outputs and incorporating predictions into patient care decisions. This ensures that the predictive models are used effectively and ethically in clinical practice.

**5. Continuous Improvement and Monitoring**

- **Model Updates:** Regularly update models with new data to capture evolving patterns and improve accuracy over time. Monitor model performance metrics (e.g., sensitivity, specificity, AUC) to detect any degradation and prompt model reevaluation or recalibration.
- **Ethical Considerations:** Ensure models are developed and deployed ethically, respecting patient privacy and minimizing biases. Regularly audit model outputs to mitigate potential biases or errors that could impact patient care.

**6. Research and Innovation**

- **Investigate Novel Approaches:** Explore emerging technologies such as deep learning or natural language processing for further improving diagnostic accuracy or personalized treatment recommendations.
- **Collaborative Research:** Foster collaborations with research institutions and industry partners to leverage collective expertise and resources for advancing breast cancer diagnosis and treatment.

## Conclusion

Implementing these recommendations can enhance the accuracy, interpretability, and clinical utility of predictive models for breast cancer diagnosis. By leveraging advanced methodologies, integrating clinical insights, and maintaining ethical standards, healthcare providers can optimize patient outcomes and improve quality of care in breast cancer management.

# CODES

## Python

```
import pandas as pd
import matplotlib.pyplot as plt
import re
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve,
classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import ConfusionMatrixDisplay

# Load the dataset from CSV file
data = pd.read_csv('/content/breast_cancer.csv')

# Display the first few rows of the dataset
print(data.head())

# Encode the categorical target variable (diagnosis: M/B) to numeric
label_encoder = LabelEncoder()
data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

# Separate features (X) and target variable (y)
X = data.drop(columns=['id', 'diagnosis'])
y = data['diagnosis']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Create and train the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)

# Predict the test set results for logistic regression
y_pred = logistic_regression.predict(X_test)
```

```
# Print classification report for logistic regression
logrepo = classification_report(y_test, y_pred)
print(logrepo)

# Calculate confusion matrix for logistic regression
conf_matrix = confusion_matrix(y_test, y_pred)

# Display labels if needed
display_labels = ['B', 'M']  # Example labels for binary classification (optional)

# Plot confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
display_labels=display_labels)
disp.plot(cmap=plt.cm.Blues)  # Adjust the colormap as needed
plt.title('Confusion Matrix - Logistic Regression')  # Replace with appropriate title
plt.show()

# Calculate ROC curve and AUC score for logistic regression
fpr, tpr, thresholds = roc_curve(y_test, logistic_regression.predict_proba(X_test)[:,1])
auc_score = roc_auc_score(y_test, logistic_regression.predict_proba(X_test)[:,1])

# Plot ROC curve for logistic regression
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()

# Print AUC score for logistic regression
print(f'AUC Score (Logistic Regression): {auc_score}')

# Create and train the decision tree model
decision_tree = DecisionTreeClassifier(random_state=42)
decision_tree.fit(X_train, y_train)

# Predict the test set results for decision tree
y_pred_tree = decision_tree.predict(X_test)

# Print classification report for decision tree
dtree = classification_report(y_test, y_pred_tree)
```

```
print(dtree)

# Calculate confusion matrix for decision tree
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)

# Plot confusion matrix for decision tree
disp_tree = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_tree,
display_labels=decision_tree.classes_)
disp_tree.plot(cmap=plt.cm.Blues)  # Adjust the colormap as needed
plt.title('Confusion Matrix - Decision Tree')
plt.show()

# Calculate ROC curve and AUC score for decision tree
fpr_tree, tpr_tree, thresholds_tree = roc_curve(y_test,
decision_tree.predict_proba(X_test)[:,1])
auc_score_tree = roc_auc_score(y_test, decision_tree.predict_proba(X_test)[:,1])

# Plot ROC curve for decision tree
plt.figure()
plt.plot(fpr_tree, tpr_tree, color='blue', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

# Print AUC score for decision tree
print(f'AUC Score (Decision Tree): {auc_score_tree}')

# Compare the AUC scores of Logistic Regression and Decision Tree
print(f'AUC Score (Logistic Regression): {auc_score}')
print(f'AUC Score (Decision Tree): {auc_score_tree}')

# Function to parse classification report into a dataframe
def parse_classification_report(report):
    lines = report.split('\n')
    parsed_data = []

    for line in lines[2:-3]:  # Skip headers and footers
        line_data = re.split(r'\s{2,}', line.strip())
        if len(line_data) < 5:
            continue
        class_name = line_data[0]
        precision = float(line_data[1])
        recall = float(line_data[2])
        f1_score = float(line_data[3])
        support = float(line_data[4])

        parsed_data.append({
            'class': class_name,
            'precision': precision,
            'recall': recall,
            'f1-score': f1_score,
            'support': support
        })

    df = pd.DataFrame(parsed_data)
    return df

# Parse classification reports into dataframes
df1 = parse_classification_report(dtree)
df2 = parse_classification_report(logrepo)

# Add model names and overall accuracy
df1['model'] = 'Decision Tree'
df2['model'] = 'Logistic Regression'

# Concatenate the two dataframes
comparison_df = pd.concat([df1, df2])

# Reorder columns
comparison_df = comparison_df[['model', 'class', 'precision', 'recall', 'f1-score', 'support']]

# Display the comparison table
print(comparison_df)
```

## **R Language**

```
# Load necessary libraries
if (!require(caret)) install.packages("caret", dependencies = TRUE)
if (!require(glmnet)) install.packages("glmnet", dependencies = TRUE)
if (!require(pROC)) install.packages("pROC", dependencies = TRUE)
if (!require(rpart)) install.packages("rpart", dependencies = TRUE)
if (!require(rpart.plot)) install.packages("rpart.plot", dependencies = TRUE)
if (!require(mlbench)) install.packages("mlbench", dependencies = TRUE)
if (!require(randomForest)) install.packages("randomForest", dependencies =
TRUE)

library(caret)
library(glmnet)
library(pROC)
library(rpart)
library(rpart.plot)
library(mlbench)
library(randomForest)


# Load the dataset
data(BreastCancer, package = "mlbench")
df <- BreastCancer

# Data preprocessing
df <- df[ , -1]  # Remove the ID column
df <- na.omit(df)  # Remove rows with missing values

# Convert factors to numeric
df$Class <- as.factor(ifelse(df$Class == "malignant", 1, 0))

# Separate features and target variable
X <- as.data.frame(sapply(df[ , -ncol(df)], as.numeric))
y <- df$Class

# Scale the features
preProcessParams <- preProcess(X, method = c("center", "scale"))
X_scaled <- predict(preProcessParams, X)

# Split the data into training and testing sets (80% train, 20% test)
set.seed(42)  # For reproducibility
trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X_scaled[trainIndex, ]
X_test <- X_scaled[-trainIndex, ]
```

```
y_train <- y[trainIndex]
y_test <- y[-trainIndex]

# Check lengths of training and test sets
cat("Length of X_train:", nrow(X_train), "\n")
cat("Length of y_train:", length(y_train), "\n")
cat("Length of X_test:", nrow(X_test), "\n")
cat("Length of y_test:", length(y_test), "\n")

# Feature selection using recursive feature elimination
control <- rfeControl(functions = rfFuncs, method = "cv", number = 10)
rfe_results <- rfe(X_train, y_train, sizes = c(1:ncol(X_train)), rfeControl =
control)
X_train <- X_train[, predictors(rfe_results)]
X_test <- X_test[, predictors(rfe_results)]

# Ensure y_train factor levels are valid R variable names
levels(y_train) <- make.names(levels(y_train))

# Train logistic regression model with cross-validation for hyperparameter
tuning
train_control <- trainControl(method = "cv", number = 10, classProbs = TRUE,
summaryFunction = twoClassSummary)
tune_grid <- expand.grid(alpha = 0:1, lambda = seq(0.0001, 0.1, length = 20))

logistic_regression <- train(x = X_train, y = y_train,
                        method = "glmnet",
                        trControl = train_control,
                        tuneGrid = tune_grid,
                        metric = "ROC",
                        family = "binomial")



# Predict the test set results
y_pred_probs <- predict(logistic_regression, newdata = X_test, type =
"prob")[,2]
y_pred <- ifelse(y_pred_probs > 0.5, 1, 0)
```

```r
# Ensure y_pred and y_test are factors with the same levels
y_pred_factor <- factor(y_pred, levels = c(0, 1))
y_test_factor <- factor(y_test, levels = c(0, 1))

# Check the length of y_pred and y_test
cat("Length of y_pred:", length(y_pred), "\n")
cat("Length of y_test:", length(y_test), "\n")
# Ensure y_pred and y_test have the same length
if (length(y_pred) != length(y_test)) {
  stop("y_pred and y_test do not have the same length.")
}
# Calculate confusion matrix
conf_matrix <- confusionMatrix(y_pred_factor, y_test_factor)

# Print the confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# ROC curve
roc_curve <- roc(y_test_factor, as.numeric(y_pred_probs))
plot(roc_curve, col = "blue", main = "ROC Curve for Logistic Regression")
auc_value <- auc(roc_curve)
print(paste("AUC:", auc_value))

# Decision Tree Analysis
decision_tree <- rpart(Class ~ ., data = df[trainIndex,], method = "class")
rpart.plot(decision_tree)

# Predict with decision tree
y_pred_tree <- predict(decision_tree, newdata = df[-trainIndex,], type = "class")
y_pred_tree_factor <- factor(y_pred_tree, levels = c(0, 1))

# Confusion matrix for decision tree
conf_matrix_tree <- confusionMatrix(y_pred_tree_factor, y_test_factor)
print("Confusion Matrix for Decision Tree:")
print(conf_matrix_tree)
```

# REFERENCES

**A) Books:**

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.

**B) Journals:**

2. Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267-288. doi:10.1111/j.2517-6161.1996.tb02080.x

**C) Reports and Articles:**

3. American Cancer Society. (2023). *Breast Cancer Facts & Figures 2023-2024*. Retrieved from https://www.cancer.org/research/cancer-facts-statistics/breast-cancer-facts-figures.html

**D) Websites:**

4.Python Software Foundation. (2023). *Python Language Reference, Version 3.10*. Retrieved from https://docs..org/3.10/

5.R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from https://www.R-project.org/

**E) Software and Tools:**

6. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. Retrieved from http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf