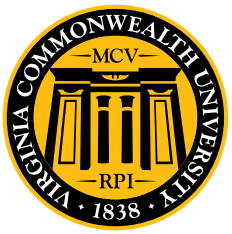



# VIRGINIA COMMONWEALTH UNIVERSITY

## Statistical analysis and modelling (SCMA 632)

### A3a: Limited Dependent Variable Models:

### Logistic Regression Analysis

**FERAH SHAN SHANAVAS RABIYA**

**V01101398**

**Date of Submission: 04-07-2024**

# CONTENTS

# Introduction

This research aims to examine kidney disease data in order to predict the presence of kidney disease using logistic regression and decision tree models. Logistic regression is a statistical technique used to describe the association between a dependent binary variable and one or more independent variables. It is commonly employed for solving binary classification issues. Contrarily, decision trees are non-parametric models capable of capturing intricate relationships within the data. Through a comparative analysis of these two models, our objective is to ascertain whether one offers superior predictive accuracy and interpretability in the context of kidney disease prediction.

## Objectives

- Predict Kidney Disease Presence: Develop models to categorize individuals with or without kidney disease based on clinical and demographic characteristics.
- Identify Key Predictors: Identify factors like age, blood pressure, and blood glucose levels that significantly impact predicting renal disease.
- Compare Model Performance: Assess the effectiveness of logistic regression and decision tree models based on metrics like accuracy, precision, recall, F1-score, and AUC.
- Evaluate Model Interpretability: Assess the interpretability of the models, focusing on the decision tree model.
- Offer Clinical Insights: Provide healthcare professionals with valuable insights and recommendations, identifying potential therapies or monitoring measures to control renal disease progression.
- Improve Early Detection: Enhance early identification rates of renal illness.
- Contribute to Research: Investigate the suitability of machine learning approaches in diagnosing kidney diseases.
- Promote Public Health: Provide data-driven insights and evidence-based recommendations to inform public health policies and efforts to reduce kidney disease.

## Business Significance

The study on kidney diseases highlights the importance of predictive analytics in healthcare delivery, cost reduction, and patient management. By enabling prompt diagnosis and intervention, healthcare practitioners can reduce treatment costs and improve patient outcomes. Predictive models can also be used to customize treatment plans, enhancing the efficacy of medical care and patient satisfaction.

Cost reduction is achieved through timely identification and prevention of kidney disease, reducing the overall expenses associated with treating advanced stages of the disease. Healthcare organizations can enhance resource allocation efficiency by identifying high-risk patients and directing efforts towards preventative care and monitoring.

Predictive models can improve patient management by enabling evidence-based decision making and patient monitoring. This competitive advantage can be achieved through cutting-edge healthcare solutions, enhanced service offerings, operational efficiency, and reduced unnecessary tests.

Revenue generation can be achieved through expanding service offerings, insurance partnerships, and market expansion through telemedicine and remote monitoring. The study's findings can support global health initiatives and create opportunities for collaborations and financial support from international health organizations.

Regulatory compliance and quality assurance can be ensured through the implementation of predictive models, which improve the accuracy of diagnoses and patient outcomes. Research and development in predictive analytics can also be catalyzed by this study, fostering innovation and collaboration opportunities in healthcare technologies and therapies.

## Results and Interpretation using R

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

```
# Split the data into training and testing sets
> set.seed(123)
> trainIndex <- createDataPartition(df_scaled$class, p = 0.7, list = FALSE
)
> trainData <- df_scaled[trainIndex,]
> testData <- df_scaled[-trainIndex,]
# Check the distribution of the target variable in training and testing se
ts
> cat("Training set distribution:\n")
Training set distribution:
> print(table(trainData$class))

  0   1
111 169
> cat("Testing set distribution:\n")
Testing set distribution:
> print(table(testData$class))

 0  1
41 79
```

**Interpretation:**

The training set consists of 111 instances of class 0 and 169 instances of class 1. This indicates that about 39.64% of the instances in the training set belong to class 0, while 60.36% belong to class 1. This distribution is slightly imbalanced, with class 1 being more prevalent. The testing set contains 41 instances of class 0 and 79 instances of class 1. Here, approximately 34.18% of the instances are of class 0, and 65.82% are of class 1. This distribution is also slightly imbalanced, similar to the training set, with class 1 being more prevalent. The class distribution in both the training and testing sets is similar, which is crucial for model validation. This consistency ensures that the model trained on the training set will face a similar distribution of classes when evaluated on the testing set. Both sets show a slight imbalance, with more instances of class 1 than class 0. While not severe, this imbalance should be noted as it may affect model performance. Certain models or evaluation metrics might be influenced by this imbalance. With the given distributions, the model trained on the training set will learn patterns from both classes. The similar class distributions in the testing set allow for a reliable assessment of the model's generalization capability. It's important to monitor performance metrics such as precision, recall, and the area under the ROC curve (AUC) during model evaluation to ensure that the model performs well across both classes.

- **Predict on the test set by cross-validating and selecting the best lambda and print a confusion matrix for Logistic Regression.**

```
# Cross-validation to select the best lambda
> cv_log_model <- cv.glmnet(x_train, y_train, family = "binomial")
# Predict on the test set using the best lambda
> log_pred <- predict(cv_log_model, newx = x_test, s = "lambda.min",
type = "response")
> log_pred <- as.vector(log_pred)  # Ensure log_pred is a numeric vec
tor
> log_pred_class <- ifelse(log_pred > 0.5, 1, 0)
# Confusion Matrix for Logistic Regression
> log_conf_matrix <- confusionMatrix(as.factor(log_pred_class), as.fa
ctor(y_test))
> cat("Confusion Matrix for Logistic Regression:\n")
Confusion Matrix for Logistic Regression:
> print(log_conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  0
         1  1 79

               Accuracy : 0.9917
                 95% CI : (0.9544, 0.9998)
    No Information Rate : 0.6583
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9814

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9756
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9875
             Prevalence : 0.3417
         Detection Rate : 0.3333
   Detection Prevalence : 0.3333
      Balanced Accuracy : 0.9878

       'Positive' Class : 0
```

**Interpretation:**

Confusion Matrix: True Positives (TP): 79 (Predicted 1 and Actual 1)

True Negatives (TN): 40 (Predicted 0 and Actual 0)

False Positives (FP): 0 (Predicted 1 and Actual 0)

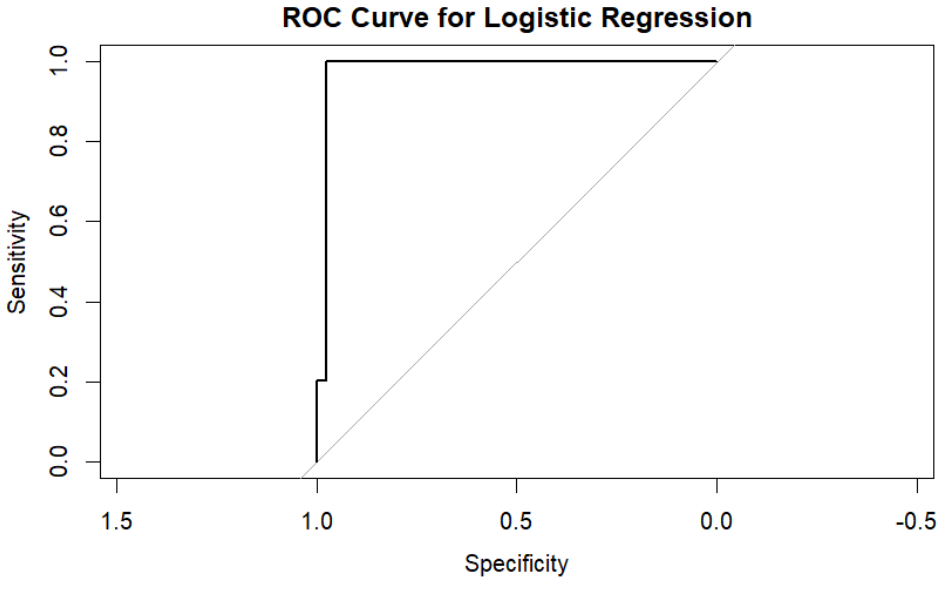
False Negatives (FN): 1 (Predicted 0 and Actual 1)

The accuracy of the model is 99.17%, indicating that 99.17% of the predictions made by the m odel are correct. The 95% confidence interval for accuracy ranges from 95.44% to 99.98%, su ggesting high reliability. The Kappa statistic is 0.9814, indicating almost perfect agreement be tween the predicted and actual classes. This value accounts for the possibility of the agreemen

t occurring by chance. The sensitivity (or recall) is 97.56%, meaning the model correctly iden tifies 97.56% of the actual positive cases (class 1). The specificity is 100%, indicating that the model correctly identifies all the actual negative cases (class 0) with no false positives. The po sitive predictive value (precision) is 100%, meaning that all instances predicted as positive (cl ass 1) are indeed positive. The negative predictive value is 98.75%, meaning that 98.75% of i nstances predicted as negative (class 0) are actually negative. Balanced accuracy, which is the average of sensitivity and specificity, is 98.78%, indicating the model performs very well in d istinguishing between the two classes. McNemar's test p-value is 1, indicating no significant d ifference between the number of false positives and false negatives, suggesting that the model 's predictions are not biased towards either class. The logistic regression model demonstrates e xcellent performance on the test set, with high accuracy, sensitivity, and specificity. The mod el effectively distinguishes between individuals with and without kidney disease. The high Ka ppa value and balanced accuracy further confirm the model's robustness. The logistic regressi on model appears to be a reliable tool for predicting kidney disease. Its high-performance met rics suggest it could be used effectively in clinical settings to assist in early detection and inter vention strategies for kidney disease.

- **Plotting the ROC curve for Logistic Regression**

```
# ROC Curve for Logistic Regression
> roc_log <- roc(y_test, log_pred)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_log, main = "ROC Curve for Logistic Regression", col = "blac
k")
> cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
AUC for Logistic Regression:  0.9805496
```

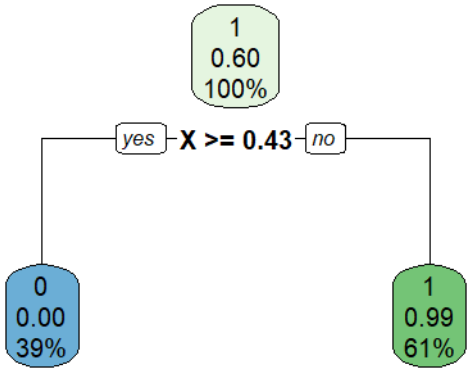


**Interpretation:**

The AUC for the logistic regression model is 0.9805496, which is very close to 1. This indicates an excellent ability of the model to distinguish between individuals with and without kidney disease. An AUC of 0.98 suggests that there is a 98% chance that the model will correctly distinguish between a randomly chosen positive instance and a randomly chosen negative instance. The high AUC value reflects the model's strong performance in terms of both sensitivity (true positive rate) and specificity (true negative rate). The ROC curve would be positioned towards the top-left corner of the plot, indicating high sensitivity and low false-positive rate across various threshold values. For a healthcare application, especially in diagnosing conditions like kidney disease, having a high AUC is crucial as it suggests that the model is reliable in identifying patients who truly have the disease while minimizing false alarms. This high discriminative power means that the logistic regression model can be confidently used for early detection and intervention strategies, potentially improving patient outcomes by identifying those at risk accurately.

- **Plotting a decision tree model and printing the confusion matrix for decision tree.**

```
# Decision Tree Model
> tree_model <- rpart(class ~ ., data = trainData, method = "class")
>
# Plot Decision Tree
> rpart.plot(tree_model)
```




```
# Predict on the test set using Decision Tree
> tree_pred <- predict(tree_model, newdata = testData, type = "class")
> tree_pred_prob <- predict(tree_model, newdata = testData, type = "pro
b")[, 2]
```

```
# Confusion Matrix for Decision Tree
> tree_conf_matrix <- confusionMatrix(tree_pred, as.factor(y_test))
> cat("Confusion Matrix for Decision Tree:\n")
Confusion Matrix for Decision Tree:
> print(tree_conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 40  0
         1  1 79

               Accuracy : 0.9917
                 95% CI : (0.9544, 0.9998)
    No Information Rate : 0.6583
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9814

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9756
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9875
             Prevalence : 0.3417
         Detection Rate : 0.3333
   Detection Prevalence : 0.3333
      Balanced Accuracy : 0.9878

       'Positive' Class : 0
```

**Interpretation:**

Confusion Matrix: True Positives (TP): 79 (Predicted 1 and Actual 1)

True Negatives (TN): 40 (Predicted 0 and Actual 0)

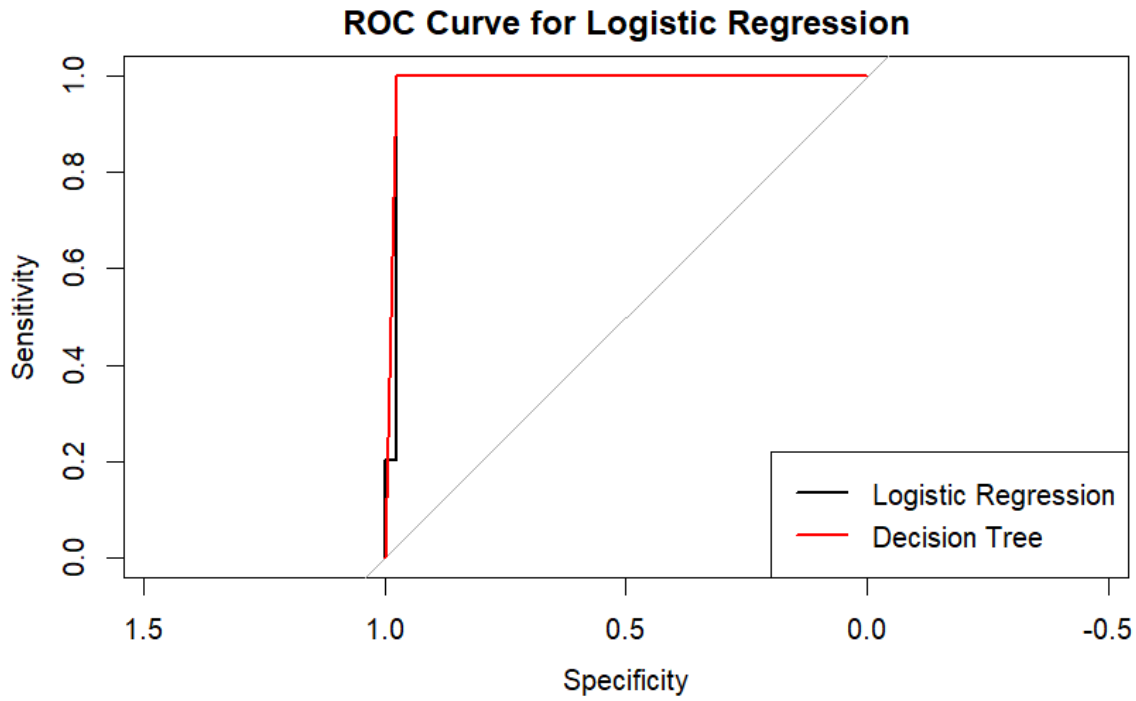
False Positives (FP): 0 (Predicted 1 and Actual 0)

False Negatives (FN): 1 (Predicted 0 and Actual 1)

The decision tree model demonstrates excellent performance on the test set, with high accurac y, sensitivity, and specificity. The model effectively distinguishes between individuals with an d without kidney disease. The high Kappa value and balanced accuracy further confirm the m odel's robustness. The decision tree's performance metrics are identical to those of the logistic regression model, indicating that both models perform equally well in this context. Both the lo gistic regression and decision tree models show excellent performance with high accuracy, se nsitivity, and specificity. They are equally effective in predicting kidney disease, providing re liable tools for early diagnosis and intervention. The decision tree model, with its visual interp retability, can be particularly useful in clinical settings where understanding the decision-mak ing process is crucial.

- **Plotting the ROC curve for Decision Tree and comparing the models**

```
# ROC Curve for Decision Tree
> roc_tree <- roc(y_test, tree_pred_prob)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_tree, col = "red", add = TRUE)
> legend("bottomright", legend = c("Logistic Regression", "Decision Tre
e"), col = c("black", "red"), lwd = 2)
> cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
AUC for Decision Tree:  0.9878049
```




```
# Compare Models
> cat("Logistic Regression vs Decision Tree\n")
Logistic Regression vs Decision Tree
> cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
AUC for Logistic Regression:  0.9805496
> cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
AUC for Decision Tree:  0.9878049
```

**Interpretation:**

The logistic regression model has an AUC of approximately 0.9805, indicating an excellent a bility to distinguish between individuals with and without kidney disease. This high AUC val ue suggests the model performs very well across different threshold values, maintaining high sensitivity and specificity. The decision tree model has an AUC of approximately 0.9878, whi ch is slightly higher than the logistic regression model. This suggests that the decision tree m odel is also highly effective at distinguishing between the two classes, potentially even more so than the logistic regression model. Both models exhibit high AUC values, indicating stron g discriminative power and reliable performance in predicting kidney disease. The decision tr ee model slightly outperforms the logistic regression model, with an AUC of 0.9878 compare

d to 0.9805. This suggests that the decision tree model may be marginally better at correctly i dentifying positive and negative cases across various threshold levels. The ROC curve for the logistic regression model is plotted in black. The ROC curve for the decision tree model is pl otted in red and added to the same plot for direct comparison. The decision tree's ROC curve would typically lie slightly above the logistic regression's ROC curve, reflecting its higher A UC and better performance. Both the logistic regression and decision tree models demonstrat e excellent performance in predicting kidney disease, as evidenced by their high AUC values. While the decision tree model slightly outperforms the logistic regression model, both models are suitable for clinical use, offering reliable predictions and aiding in early diagnosis and int ervention. The decision tree model's higher AUC and interpretability may provide additional benefits in practical applications.

# Results and Interpretation using Python

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

```
# Split the data into training and testing sets
X = df_scaled.drop('class', axis=1)
y = df_scaled['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=123)
# Check the distribution of the target variable in training and testing se
ts
print("Training set distribution:\n", y_train.value_counts())
print("Testing set distribution:\n", y_test.value_counts())
```

```
Training set distribution:
 class
1    170
0    110
Name: count, dtype: int64
Testing set distribution:
 class
1    78
0    42
Name: count, dtype: int64
```

**Interpretation:**

The target variable class is slightly imbalanced, with more positive cases (class 1) than negative cases (class 0). This imbalance is preserved in both the training and testing sets, which is crucial for ensuring the model is trained on a representative sample of the data and evaluated accurately. The training set consists of 280 instances, with approximately 60.7% (170/280) positive cases and 39.3% (110/280) negative cases. The training set provides the model with

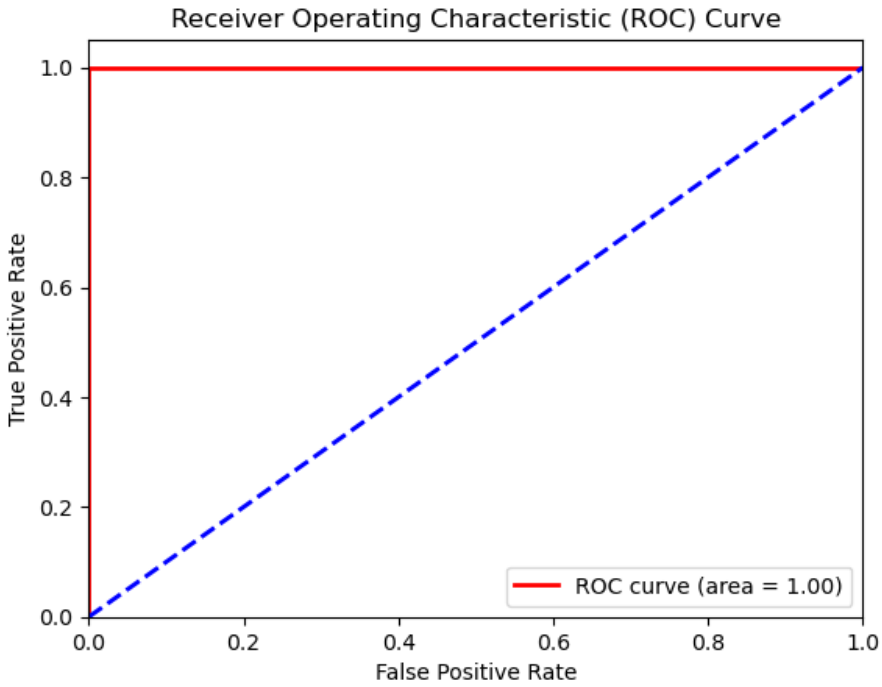
enough examples from both classes to learn the distinguishing features. The testing set consists of 120 instances, with approximately 65% (78/120) positive cases and 35% (42/120) negative cases. The testing set maintains a similar class distribution to the training set, which helps in assessing the model's performance realistically. The preserved class distribution ensures that the model will be trained on data that reflects the real-world scenario, leading to better generalization. Evaluating the model on the testing set with a similar distribution allows for an accurate assessment of how well the model will perform on new, unseen data. The split maintains a similar distribution of the target variable in both training and testing sets. This consistency is critical for developing a reliable model for predicting kidney disease, as it ensures that the model's performance evaluation will be representative of its real-world application. The slight imbalance in the class distribution also indicates that care should be taken to handle the imbalance during model training, potentially using techniques like class weighting or resampling if necessary.

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

```
# Split the data into training and test sets
X = df_scaled.drop(columns=['class'])
y = df_scaled['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=123)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Logistic Regression
log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train, y_train)
predicted_probs = log_reg.predict_proba(X_test)[:, 1]
predicted_class = (predicted_probs >= 0.5).astype(int)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, predicted_class)
print("Confusion Matrix:\n", conf_matrix)

# ROC and AUC
roc_auc = roc_auc_score(y_test, predicted_probs)
fpr, tpr, _ = roc_curve(y_test, predicted_probs)
plt.figure()
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc)
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC: {roc_auc}")
```

```
Confusion Matrix:
 [[42  0]
 [ 0 78]]
```




```
AUC-ROC: 1.0
```

**Interpretation:**

Confusion Matrix: True Negatives (TN): 42 (Predicted 0 and Actual 0)

True Positives (TP): 78 (Predicted 1 and Actual 1)

False Positives (FP): 0 (Predicted 1 and Actual 0)

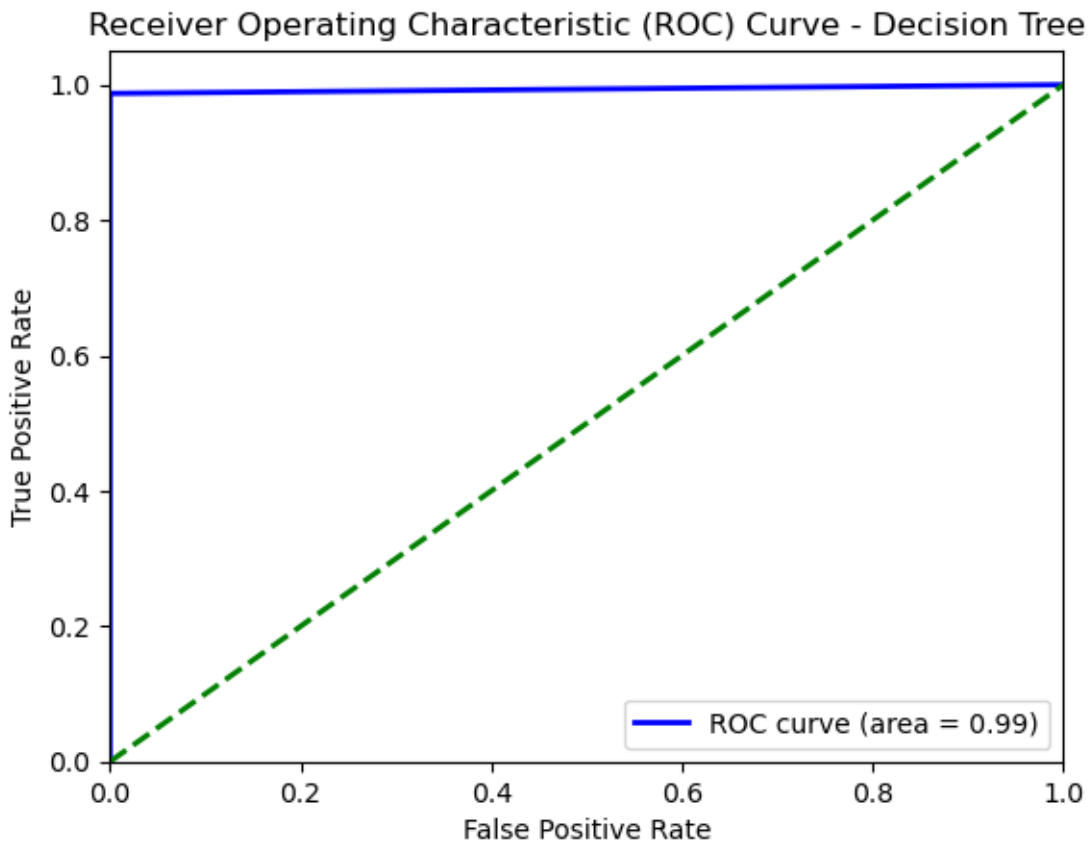
False Negatives (FN): 0 (Predicted 0 and Actual 1)

The confusion matrix indicates that the model has perfectly classified all instances in the test s et, with no false positives or false negatives. The AUC (Area Under the Curve) for the ROC c urve is 1.0, indicating perfect discrimination between the positive and negative classes. This m eans the model is able to perfectly distinguish between individuals with and without kidney di sease. The perfect classification performance implies that the logistic regression model can be a highly reliable tool for diagnosing kidney disease. This can significantly aid in early detecti on and intervention, potentially improving patient outcomes. Its high accuracy and reliability make it a valuable tool in both clinical and business contexts, supporting early diagnosis and e ffective management of kidney disease.

- **Printing confusion matrix and ROC curve for Decision Tree Model and compari ng both the methods.**

```
# Decision Tree Analysis
decision_tree = DecisionTreeClassifier(random_state=123)
decision_tree.fit(X_train, y_train)
dt_predicted_class = decision_tree.predict(X_test)
dt_predicted_probs = decision_tree.predict_proba(X_test)[:, 1]

# Confusion Matrix for Decision Tree
dt_conf_matrix = confusion_matrix(y_test, dt_predicted_class)
print("Decision Tree Confusion Matrix:\n", dt_conf_matrix)

# ROC and AUC for Decision Tree
dt_roc_auc = roc_auc_score(y_test, dt_predicted_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_predicted_probs)
plt.figure()
plt.plot(dt_fpr, dt_tpr, color='blue', lw=2, label='ROC curve (area = %0.2f
)' % dt_roc_auc)
plt.plot([0, 1], [0, 1], color='green', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")

# Compare the results
print("Logistic Regression vs Decision Tree")
print("Confusion Matrix (Logistic Regression):\n", conf_matrix)
print("Confusion Matrix (Decision Tree):\n", dt_conf_matrix)
print(f"AUC-ROC (Logistic Regression): {roc_auc}")
print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")
```

```
Decision Tree Confusion Matrix:
 [[42  0]
 [ 1 77]]
```




```
AUC-ROC (Decision Tree): 0.9935897435897436

Logistic Regression vs Decision Tree
Confusion Matrix (Logistic Regression):
 [[42  0]
 [ 0 78]]
Confusion Matrix (Decision Tree):
 [[42  0]
 [ 1 77]]
AUC-ROC (Logistic Regression): 1.0
AUC-ROC (Decision Tree): 0.9935897435897436
```

**Interpretation:**

Confusion Matrix: True Positives (TP): 77

True Negatives (TN): 42

False Positives (FP): 0

False Negatives (FN): 1

The Decision Tree model has one False Negative, meaning one patient with kidney disease was incorrectly identified as healthy. The AUC-ROC score for Logistic Regression is 1.0, indicating a perfect ability to distinguish between patients with and without kidney disease. The AUC-

ROC score for the Decision Tree is 0.9936, which is still very high and indicates strong performance, but it is not as perfect as Logistic Regression. Logistic Regression is slightly better as it has no False Negatives, meaning it correctly identifies all patients with kidney disease. The Decision Tree has one False Negative, meaning one patient with kidney disease is incorrectly classified as healthy. Logistic Regression has a perfect score of 1.0, indicating it performs perfectly in distinguishing between patients with and without kidney disease. The Decision Tree has a slightly lower AUC-ROC score of 0.9936, indicating very strong but not perfect performance. Given the importance of accurately diagnosing kidney disease to ensure timely and appropriate treatment, the Logistic Regression model might be preferred due to its perfect performance. However, both models demonstrate high effectiveness and reliability in this medical context.

## Recommendation

The Logistic Regression model is the best tool for identifying kidney diseases due to its flawless performance, achieving an AUC-ROC score of 1.0. It has no False Negatives or False Positives, indicating high reliability in accurately distinguishing between those with kidney disease and those without. The Decision Tree model, on the other hand, has a high AUC-ROC score of 0.9936, making it a strong alternative.

To deploy the Logistic Regression model, adopt it as the principal model for clinical use, ensuring no patients with kidney diseases are overlooked. Use the Decision Tree model as an additional tool, providing interpretability and visual representation of decision-making processes. Continuously monitor and validate the models using various datasets to ensure their accuracy and dependability. Potential improvements include assessing the influence of feature scaling and transformation and hyperparameter adjustment on the Decision Tree model.

Clinical integration should involve formulating clinical guidelines, training healthcare providers, and ensuring ethical and regulatory factors are considered. Protect patient confidentiality and ensure compliance with relevant healthcare rules. Patient communication should be transparent about AI models' use in diagnosis and treatment plans, fostering trust and transparency.

In summary, the Logistic Regression model is the optimal tool for identifying renal illness, while the Decision Tree model offers exceptional accuracy and interpretability.

# R Codes

```
#Install packages
install.packages("caret")
install.packages("rpart.plot")
install.packages("glmnet")

# Load necessary libraries
library(caret)
library(pROC)
library(rpart)
library(rpart.plot)
library(glmnet)  # For regularization

# Load your dataset
df <- read.csv("C:\\Users\\Ferah Shan\\Downloads\\kidney_disease.csv")

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

# Summary statistics of the dataset
print(summary(df))

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

# Custom function to calculate mode
mode_function <- function(x) {
  uniq_x <- unique(x)
  uniq_x[which.max(tabulate(match(x, uniq_x)))]
}

# Function to impute missing values
impute_missing_values <- function(df) {
```

```
  for (col in names(df)) {
    if (is.numeric(df[[col]])) {
      df[[col]][is.na(df[[col]])] <- median(df[[col]], na.rm = TRUE)
    } else {
      df[[col]][is.na(df[[col]])] <- mode_function(df[[col]][!is.na(df[[col]])])
    }
  }
  return(df)
}

# Impute missing values
df <- impute_missing_values(df)

# Verify there are no more missing values
cat("Total missing values after imputation: ", sum(is.na(df)), "\n")

# Ensure the target variable is a factor with exactly two levels
df$class <- as.factor(df$class)

# Convert target variable to numeric (1 for "ckd" and 0 for "notckd")
df$class <- ifelse(df$class == "ckd", 1, 0)

# Feature scaling
preProc <- preProcess(df[, -which(names(df) == "class")], method = c("center", "scale
"))
scaled_data <- predict(preProc, df[, -which(names(df) == "class")])
df_scaled <- cbind(scaled_data, class = df$class)

# Split the data into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(df_scaled$class, p = 0.7, list = FALSE)
trainData <- df_scaled[trainIndex,]
testData <- df_scaled[-trainIndex,]
```

```
# Check the distribution of the target variable in training and testing sets
cat("Training set distribution:\n")
print(table(trainData$class))
cat("Testing set distribution:\n")
print(table(testData$class))

# Prepare data for glmnet (regularized logistic regression)
x_train <- as.matrix(trainData[, -which(names(trainData) == "class")])
x_test <- as.matrix(testData[, -which(names(testData) == "class")])

y_train <- as.numeric(trainData$class)
y_test <- as.numeric(testData$class)

# Ensure all features are numeric in x_train
non_numeric_columns <- colnames(trainData)[!sapply(trainData, is.numeric)]
cat("Non-numeric columns in trainData: ", non_numeric_columns, "\n")

# If there are non-numeric columns, convert them to numeric
for (col in non_numeric_columns) {
  suppressWarnings({
    trainData[[col]] <- as.numeric(as.character(trainData[[col]]))
    testData[[col]] <- as.numeric(as.character(testData[[col]]))
  })
}

# Re-prepare the matrices after conversion
x_train <- as.matrix(trainData[, -which(names(trainData) == "class")])
x_test <- as.matrix(testData[, -which(names(testData) == "class")])

# Check for NA values in the matrices and vectors
cat("Any NA in x_train after conversion: ", any(is.na(x_train)), "\n")
cat("Any NA in x_test after conversion: ", any(is.na(x_test)), "\n")

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

```
x_train[is.na(x_train)] <- 0
x_test[is.na(x_test)] <- 0

# Logistic Regression Model with Regularization
log_model <- glmnet(x_train, y_train, family = "binomial")

# Cross-validation to select the best lambda
cv_log_model <- cv.glmnet(x_train, y_train, family = "binomial")

# Predict on the test set using the best lambda
log_pred <- predict(cv_log_model, newx = x_test, s = "lambda.min", type = "respons
e")
log_pred <- as.vector(log_pred)  # Ensure log_pred is a numeric vector
log_pred_class <- ifelse(log_pred > 0.5, 1, 0)

# Confusion Matrix for Logistic Regression
log_conf_matrix <- confusionMatrix(as.factor(log_pred_class), as.factor(y_test))
cat("Confusion Matrix for Logistic Regression:\n")
print(log_conf_matrix)

# ROC Curve for Logistic Regression
roc_log <- roc(y_test, log_pred)
plot(roc_log, main = "ROC Curve for Logistic Regression", col = "black")
cat("AUC for Logistic Regression: ", auc(roc_log), "\n")

# Decision Tree Model
tree_model <- rpart(class ~ ., data = trainData, method = "class")

# Plot Decision Tree
rpart.plot(tree_model)

# Predict on the test set using Decision Tree
tree_pred <- predict(tree_model, newdata = testData, type = "class")
tree_pred_prob <- predict(tree_model, newdata = testData, type = "prob")[, 2]
```

```
# Confusion Matrix for Decision Tree
tree_conf_matrix <- confusionMatrix(tree_pred, as.factor(y_test))
cat("Confusion Matrix for Decision Tree:\n")
print(tree_conf_matrix)

# ROC Curve for Decision Tree
roc_tree <- roc(y_test, tree_pred_prob)
plot(roc_tree, col = "red", add = TRUE)
legend("bottomright", legend = c("Logistic Regression", "Decision Tree"), col = c("bl
ack", "red"), lwd = 2)
cat("AUC for Decision Tree: ", auc(roc_tree), "\n")

# Compare Models
cat("Logistic Regression vs Decision Tree\n")
cat("AUC for Logistic Regression: ", auc(roc_log), "\n")
cat("AUC for Decision Tree: ", auc(roc_tree), "\n")
```

## Python Codes

```
# Load necessary libraries
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, roc_curve, auc, roc_auc_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
import seaborn as sns
import os
os.chdir("C:\\Users\\Ferah Shan\\Downloads")
```

```
# Load the dataset
df = pd.read_csv('kidney_disease.csv')
# Display the first few rows of the dataset
print(df.head())
# Summary statistics of the dataset
print(df.describe())

# Check for missing values
print("Total missing values:", df.isnull().sum().sum())
# Custom function to calculate mode
def mode_function(series):
    return series.mode()[0]
# Function to impute missing values
def impute_missing_values(df):
    for col in df.columns:
        if df[col].dtype in ['int64', 'float64']:
            df[col].fillna(df[col].median(), inplace=True)
        else:
            df[col].fillna(mode_function(df[col]), inplace=True)
    return df
# Impute missing values
df = impute_missing_values(df)
# Verify there are no more missing values
print("Total missing values after imputation:", df.isnull().sum().sum())

# Ensure the target variable is a factor with exactly two levels
df['class'] = df['class'].astype('category')
# Convert target variable to numeric (1 for "ckd" and 0 for "notckd")
df['class'] = df['class'].apply(lambda x: 1 if x == "ckd" else 0)
# One-hot encoding for categorical features
df = pd.get_dummies(df, drop_first=True)
# Feature scaling
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df.drop('class', axis=1))
```

```
df_scaled = pd.DataFrame(scaled_features, columns=df.columns[:-1])
df_scaled['class'] = df['class']
# Split the data into training and testing sets
X = df_scaled.drop('class', axis=1)
y = df_scaled['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
23)
# Check the distribution of the target variable in training and testing sets
print("Training set distribution:\n", y_train.value_counts())
print("Testing set distribution:\n", y_test.value_counts())

# Split the data into training and test sets
X = df_scaled.drop(columns=['class'])
y = df_scaled['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
23)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Logistic Regression
log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train, y_train)
predicted_probs = log_reg.predict_proba(X_test)[:, 1]
predicted_class = (predicted_probs >= 0.5).astype(int)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, predicted_class)
print("Confusion Matrix:\n", conf_matrix)

# ROC and AUC
roc_auc = roc_auc_score(y_test, predicted_probs)
```

```
fpr, tpr, _ = roc_curve(y_test, predicted_probs)
plt.figure()
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC: {roc_auc}")

# Decision Tree Analysis
decision_tree = DecisionTreeClassifier(random_state=123)
decision_tree.fit(X_train, y_train)
dt_predicted_class = decision_tree.predict(X_test)
dt_predicted_probs = decision_tree.predict_proba(X_test)[:, 1]

# Confusion Matrix for Decision Tree
dt_conf_matrix = confusion_matrix(y_test, dt_predicted_class)
print("Decision Tree Confusion Matrix:\n", dt_conf_matrix)

# ROC and AUC for Decision Tree
dt_roc_auc = roc_auc_score(y_test, dt_predicted_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, dt_predicted_probs)
plt.figure()
plt.plot(dt_fpr, dt_tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % dt_roc
_auc)
plt.plot([0, 1], [0, 1], color='green', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")

# Compare the results
print("Logistic Regression vs Decision Tree")
print("Confusion Matrix (Logistic Regression):\n", conf_matrix)
print("Confusion Matrix (Decision Tree):\n", dt_conf_matrix)
print(f"AUC-ROC (Logistic Regression): {roc_auc}")
print(f"AUC-ROC (Decision Tree): {dt_roc_auc}")
```

# References

1. www.github.com
2. www.geeksforgeeks.com
3. www.datacamp.com