# | FIT3152 Data Analytics |

# | Assignment 2 |

# | Chian Yee On – 33402302 |

# Introduction

Code and output: Please refer to the appendix introduction.

I began by organising my workspace and adding the "PhishingData.csv" file to the "Phish" data frame. I used my student ID as the random seed to make sure it could be repeated. I took 10 out of the 50 rows at random to produce a subset of the data, and then I filtered "Phish" based on that selection. Next, I selected 2000 rows at random from this filtered data and stored it in "PD".

Next, I loaded several essential R packages: `tree` for decision tree models, "e1071" for support vector machines, "ROCR" for ROC curve analysis, "randomForest" for random forests, "adabag" for boosting and bagging, "rpart" and "rpart.plot" for advanced decision tree plotting, "dplyr" for data manipulation, "ggplot2" for plotting, `tidyr` for reshaping data, and "caret" for training and evaluating machine learning models.

I was ready to create and assess different machine learning models to analyse the phishing data after getting my data ready and loading my packages.

1. **Explore the data: What is the proportion of phishing sites to legitimate sites? Obtain descriptions of the predictor (independent) variables – mean, standard deviations, etc. for real-valued attributes. Is there anything noteworthy in the data? Are there any attributes you need to consider omitting from your analysis? (1 Mark)**

   Code and output: Please refer to the appendix 1.

I began by looking over the organisation and synopsis of the PD dataset in order to investigate the data and ascertain the ratio of phishing sites to legitimate websites. I made sure the Class column—which shows if a website is legitimate or phishing—was taken into consideration. The ratio of phishing sites (label 1) to legitimate sites (label 0) was then computed and printed. According to the results, 38.15% of the websites were phishing, and roughly 61.85% of them were legitimate sites.

I next computed the means and standard deviations of the predictor variables to get descriptions of them. This required taking certain numerical columns out of the dataset and computing these statistics with the sapply function. The variability and core tendency of the numerical properties were revealed by the summary statistics.

Significantly, as the summary output showed, some variables had a high percentage of missing data. For example, there were a lot of missing entries in columns A02, A03, A05, and a few more, which could have an impact on findings. Should these variables not make a substantial contribution to the model's performance—which seems likely given the high percentage of missing data in several columns—they may be candidates for removal.

I discovered 500 missing entries when I finally counted the missing values in the dataset. I was able to recognise probable factors for omission due to missing data, comprehend the distribution of the data, and pinpoint important statistics for the predictor variables thanks to this thorough study.

2. **Document any pre-processing required to make the data set suitable for the model fitting that follows. (1 Mark)**

Code and output: Please refer to the appendix 2.

I completed the required pre-processing procedures to make sure the data was clean and appropriate for analysis before preparing the dataset for model fitting. Use of the na.omit method to eliminate rows with missing values was the main pre-processing action I performed. The performance and accuracy of machine learning models can be negatively impacted by missing data, therefore this was vital. Making sure the dataset only included full cases, I removed partial entries to make it robust enough for further research.

I produced a summary of the cleaned dataset once the missing values were removed. The central tendency and distribution of each variable were shown by the summary statistics. A02, on the other hand, had a maximum value of 16 and a mean of 0.1469, whereas the variable A01 ranged from 9 to 50 with a mean of 26.11. With a high mean value of 0.8467, A08 and other variables demonstrated their probable significance in the analysis. Furthermore, the dataset included 603 phishing websites and 956 legitimate sites following cleaning, as indicated by the Class variable.

To guarantee that the dataset was free of missing values, which could otherwise introduce biases or mistakes into the model fitting process, this pre-processing phase was crucial. Through data cleaning, I established a strong basis for developing precise and dependable prediction models.

3. **Divide your data into a 70% training and 30% test set by adapting the following code (written for the iris data). Use your student ID as the random seed.**

   Code and output: Please refer to the appendix 3.

Setting the random seed to my student ID with set.seed(33402302) ensures reproducibility. With 70% of the data designated for training and 30% for testing, I subsequently divided the dataset PD into training and test sets. To pick the training rows at random, I used the sample function. To make sure the data was clean, after generating the training and test sets, I used na.omit to eliminate any rows that had missing values. In conclusion, since classifying data is critical for classification tasks, I ensured that the Class column was a factor in both the training and test sets.

4. **Implement a classification model using each of the following techniques. For this question you may use each of the R functions at their default settings if suitable. (5 Marks)**
   **• Decision Tree**
   **• Naïve Bayes**
   **• Bagging**
   **• Boosting**
   **• Random Forest**

   Code and output: Please refer to the appendix 4.

I used the tree function to compute a decision tree on the training data and then examined its summary to comprehend its structure and functionality before constructing and assessing a variety of classification models. The Naive Bayes classifier was then implemented using the naiveBayes function from the e1071 package, and its performance was examined by summarising the results.

Bagging and boosting were implemented for ensemble methods using the adabag package. I looked over the model summary and decided on 5 iterations (mfinal=5) for bagging. In order to gauge the performance of the model, I gave boosting 10 iterations (mfinal=10) and summarised the data.

Lastly, I used the randomForest package to construct a Random Forest model, making sure that no missing data were included in the process. In order to understand the Random Forest model's functionality and key parameters, I summarised it.

5. **Using the test data, classify each of the test cases as 'phishing (1)' or 'legitimate (0)'. Create a confusion matrix and report the accuracy of each model. (1 Mark)**

   Code and output: Please refer to the appendix 5.

I used the test data to compute confusion matrices and accuracy for each model, classifying each test case as either "phishing" or "legitimate" and assessing the effectiveness of various models. To classify the test data for the decision tree, I used the predict function. Then, I made a confusion matrix to compare the predicted classes with the actual classes. With an accuracy of 73.72%, the decision tree accurately identified 256 legitimate websites and 89 phishing websites.

Next, I assessed the Naive Bayes model, which had an accuracy of only 40.38% and accurately predicted 7 legitimate sites while misclassifying a significant number of legitimate sites as phishing. Naive Bayes may not be appropriate for this dataset, based on its inferior performance. I obtained a confusion matrix with 251 legitimate sites and 93 phishing sites accurately categorised, achieving an accuracy of 73.5%, using the bagging model by using the predict.bagging function.

Similarly, the boosting model had an accuracy of 72.86% when it was validated using predict.boosting, successfully classifying 238 legitimate sites and 103 phishing sites. The test data was finally predicted using the random forest model. The maximum accuracy of 76.5% was achieved by properly classifying 253 legitimate sites and 105 phishing sites, according to the confusion matrix. Based on an extensive analysis of several models, the random forest was shown to be the most accurate model. Comparatively speaking, the decision tree and bagging models fared better than the Naive Bayes model, which demonstrated noticeably lower accuracy and might not be appropriate for this specific dataset.

6. **Using the test data, calculate the confidence of predicting 'phishing' for each case and construct an ROC curve for each classifier. You should be able to plot all the curves on the same axis. Use a different colour for each classifier. Calculate the AUC for each classifier. (1 Mark)**

   Code and output: Please refer to the appendix 6.

In order to generate ROC curves for each classifier and determine the confidence in predicting "phishing" for each example, I began by making a blank plot with axes labelled "True Positive Rate" and "False Positive Rate." To represent the performance of a random classifier, I added a diagonal reference line.

To create predictions and determine the ROC curve for each classifier, I used the test data. To acquire the class probabilities, I first entered type = "vector" in the predict function to get the prediction probabilities for the decision tree. Next, using the performance function, I built a ROCR::prediction object and determined the true positive rate (TPR) and false positive rate (FPR). My area under the curve (AUC) was 0.7081 when I plotted the decision tree's ROC curve in blue.

Then, I carried out the same steps for the Naive Bayes model, obtaining the probabilities by using type = "raw". Blueviolet was used to plot the ROC curve, and 0.6871 was the area under the curve. I produced a red ROC curve with an AUC of 0.7483 for the bagging model using the predict.bagging function. The ROC curve I created for the boosting model, which has an AUC of 0.7537, is shown in green. When I finally

plotted the ROC curve in dark green and got the maximum AUC of 0.7721, I used the predict function for the random forest model to acquire class probabilities.

I labelled each classifier's ROC curve and associated AUC value by adding a legend to the plot. I was able to evaluate each classifier's performance using their ROC curves and AUC values thanks to this thorough visualisation, which showed that the random forest performed the best out of all the models that were assessed.

7. **Create a table comparing the results in Questions 5 and 6 for all classifiers. Is there a single "best" classifier? (1 Mark)**

   Code and output: Please refer to the appendix 7.

I made a comparison table so I could see the accuracy and AUC (Area Under the Curve) statistics for every classifier to see how well they performed in contrast to one another. Naive Bayes, Random Forest, Bagging, Boosting, and Decision Tree were the classifiers that were assessed. The table gave me a direct comparison of their efficacy by summarising the answers to Questions 5 and 6.

Although the accuracy ratings provide the proportion of correctly categorised cases, the AUC values give an indication of how well the classifiers can distinguish between classes. Following were the results displayed in the table:

- Decision Tree: AUC = 0.7081, Accuracy = 73.72%
- Naive Bayes: AUC = 0.6871, Accuracy = 40.38%
- Bagging: AUC = 0.7483, Accuracy = 73.50%
- Boosting: AUC = 0.7537, Accuracy = 72.86%
- Random Forest: AUC = 0.7721, Accuracy = 76.50%

With the highest accuracy of 76.50% and AUC of 0.7721, the Random Forest classifier was the top performer when measured by these parameters. The Random Forest model performs consistently well on both criteria, suggesting that it is the most suitable for correctly categorising the test data as legitimate or phishing.

8. **Examining each of the models, determine the most important variables in predicting whether a web site will be phishing or legitimate. Which variables could be omitted from the data with very little effect on performance? Give reasons. (2 Marks)**

   Code and output: Please refer to the appendix 8.

I looked at the attribute importance for each model in order to identify the key factors that influence the likelihood of a website being phished or legitimate. The summary began with the decision tree and showed that variables A18, A01, and A23 were essential to building the tree. A01, A18, A23, and A22 were the most important variables for the bagging model, as indicated by their high importance ratings. A01, A18, A23, A22, and A08 were the variables that stood out as important in the boosting model.

Based on the Mean Decrease Gini, the random forest model offered a thorough understanding of variable importance. A01, A18, A23, and A22 were the most significant variables, however A08 and A12 also demonstrated a noteworthy degree of significance. It is not possible to evaluate the significance of the variables Naive Bayes utilises because it computes the probability of each variable given the class and does not provide variable relevance ratings.

These analyses showed that variables like A01, A18, A23, and A22 consistently showed up as significant across models, demonstrating their high predictive ability. On the other hand, factors like A03, A05, A07, A13, and A25 that have low relevance scores might be left out with no effect on how well the model

performs. Removing these less crucial variables could simplify the model and increase analytical efficiency without appreciably compromising its accuracy.

9. **Starting with one of the classifiers you created in Question 4, create a classifier that is simple enough for a person to be able to classify whether a site is phishing or legitimate by hand. Describe your model with either a diagram or written explanation. What factors were important in your decision? State why you chose the attributes you used. Using the test data created in Question 3, evaluate model performance using the measures you calculated for Questions 5 and 6. How does it compare to those in Question 4? (4 Marks)**

   Code and output: Please refer to the appendix 9.

I used the decision tree model from Question 4 to start building a basic classifier for manually identifying if a website is phishing or legitimate. The interpretability of decision trees makes them appropriate for this kind of work. On the basis of the training data (PD.train), I trained a decision tree with the rpart function and then pruned it to improve its clarity and simplicity. The model becomes simpler to comprehend and operate manually when superfluous branches are removed through pruning.

After training the tree, I used the complexity parameter (cp) value that reduced cross-validation error to prune it. I set cp to 0.01. Three attributes (URL length, A01), (number of domain tokens, A18), and (URL entropy, A23) were employed in the ultimately pruned tree. The reason these characteristics were chosen was that prior analyses had shown them to be significant predictors, indicating their importance in phishing website identification.

Using rpart.plot, I was able to visualise the pruned tree and present a straightforward, manually observable decision structure. The description of the model states that it classifies webpages using A18 for the initial split and A01 and A23 for future splits.

The confusion matrix indicated an accuracy of 73.72% when assessing the pruned tree's performance on the test data (PD.test). AUC (Area Under the Curve), which I also computed to assess the model's discriminatory power, came back at 0.6954.

In terms of accuracy, the pruned tree performed similarly to the models from Question 4, but with a significantly lower AUC. A summary is given in the following table:

| Classifier | AUC | Accuracy |
|---|---|---|
| Decision Tree | 0.7081 | 73.72% |
| Naïve Bayes | 0.6871 | 40.38% |
| Bagging | 0.7652 | 73.29% |
| Boosting | 0.7867 | 74.15% |
| Random Forest | 0.7773 | 76.50% |
| Pruned Decision Tree | 0.6954 | 73.72% |

10. **Create the best tree-based classifier you can. You may do this by adjusting the parameters, and/or cross-validation of the basic models in Question 4. Show that your model is better than the others using the measures you calculated for Questions 5 and 6. Describe how you created your improved model, and why you chose that model. What factors were important in your decision? State why you chose the attributes you used. (4 Marks)**

    Code and output: Please refer to the appendix 10.

By using the Random Forest model (PD.rf) from Question 4, I tried to develop the best tree-based classifier possible, improving its evaluation by modifying its parameters and, if needed, doing cross-validation. Here is a thorough explanation:

First, I predicted the test dataset (PD.test) using the PD.rf model. By comparing the projected classes with the actual classes in a confusion matrix, the predictions were assessed. This gave a clear picture of the model's performance in terms of correctly and incorrectly classified data.

Next, using the confusion matrix, the Random Forest model's accuracy was computed. To be more precise, an accuracy of 76.5% was obtained by dividing the entire number of forecasts by the sum of the diagonal components, or accurate predictions.

Then, I computed the Random Forest model's Area Under the Curve (AUC). This was accomplished by forecasting the class probabilities for the test dataset and computing the AUC with the ROCR tool. With an AUC score of 0.7773, the Random Forest model demonstrated a remarkable capacity to discriminate between legitimate and phishing websites.

I put the Random Forest model's ROC curve alongside the ROC curves of other models for comparison in order to see how well the model performed. This was accomplished by charting each classifier's True Positive Rate (TPR) versus False Positive Rate (FPR) on the same axis. The Random Forest model (red line) outperformed other classifiers, according to the ROC curves.

To summarise the accuracy and AUC of all classifiers—Decision Tree, Naive Bayes, Bagging, Boosting, and Random Forest—a comparison table was made. The table unequivocally demonstrated that the Random Forest model was the most accurate (76.5%) and had the second-best competitive AUC (0.7773), behind only Boosting (0.7867).

There are several reasons why I chose the Random Forest model. First of all, Random Forests are renowned for their resilience and efficiency when working with big datasets. They also shed light on the significance of features, which is essential for comprehending the variables affecting predictions. To make sure that only the most pertinent aspects were included, the model's attributes were chosen based on their relevance scores from earlier assessments. This enhanced the model's functionality and rendered it easier to understand.

Because of its higher performance measures, the Random Forest model was selected as the best tree-based classifier. Its efficacy in distinguishing between phishing and legitimate websites was shown by its excellent accuracy and competitive AUC. This model is a solid option for this classification assignment since it allows me to guarantee that the predictions are dependable and grounded on the most important characteristics.

**11. Using the insights from your analysis so far, implement an Artificial Neural Network classifier and report its performance. Comment on attributes used and your data preprocessing required. How does this classifier compare with the others? Can you give any reasons? (4 Marks)**

Code and output: Please refer to the appendix 11.

To prepare the dataset for ANN modelling, I first loaded it and verified that the 'Class' variable—which denotes whether a website is legitimate (0) or phishing (1)—was a factor. Then, I changed it to a numeric variable. Clear data was ensured by removing missing values.

In order to improve performance and hasten the neural network's convergence, the data was then divided into training and test sets using a 70-30 split, and the numerical characteristics were scaled. Data centering

and scaling were accomplished using the caret package's preProcess function.

Using the neuralnet package and a straightforward network architecture with one hidden layer and three neurons, I trained the ANN model. As a classification issue, the linear.output option was set to FALSE. I visualised the network to comprehend its structure after training.

The trained ANN model was used to make predictions on the test set. By contrasting the expected classes with the actual classes, I built a confusion matrix to assess the performance. In comparison to some of the other classifiers, the ANN model's accuracy was determined to be 62.5%.

The Area Under the Curve (AUC) was computed using the ROCR software in order to evaluate the model's performance in more detail. Comparing the ANN to other models, its discriminative performance is low, as evidenced by its AUC of 0.5414.

AUC and accuracy for each classifier, including the ANN, are included in the comparison table I created at the end to provide a summary of the findings. In compared to the other models, the ANN's performance was clearly shown by this.

Following preprocessing, the characteristics utilised in the ANN model were selected based on their occurrence in the dataset. Scaling the features, resolving missing values, and transforming the 'Class' variable to numeric were all part of the data preparation procedure. In order for the ANN to properly learn from the data and produce accurate predictions, these procedures were essential.

In contrast, the ANN model did not perform as well as other models such as Random Forest, Boosting, Bagging, Decision Tree, and Naive Bayes. With an AUC of 0.5414, the ANN demonstrated 62.5% accuracy. Even with an AUC of 0.7773 and an accuracy of 76.5%, the Random Forest model surpassed all other classifiers.

A number of variables, like the intricacy of the dataset, the oversimplified network construction, or the requirement for more thorough hyperparameter tweaking, might be to blame for the ANN's inferior performance. Because of its resilience and enhanced capacity to manage the dataset's intricacy and relationships, the Random Forest model most likely demonstrated superior performance.

The comparison table indicates that the Random Forest model outperforms the other tree-based classifiers, including the ANN model, while having some predictive power of its own. The accuracy of the ANN is reduced, and its AUC suggests that it can distinguish between phishing and legitimate websites with less reliability. This may be the result of the ANN model's simplicity as well as the possible requirement for more intricate network architecture and extensive tuning.

12. **Fit a new classifier to the data, test and report its performance in the same way as for previous models. You can choose a new type of classifier not covered in the course, or a new version of any of the classifiers we have studied. Either way, you will be implementing a new R package. As a starting point, you might refer to James et al. (2021), or look online. When writing up, state the new classifier and package used. Include a web link to the package details. Give a brief description of the model type and how it works. Comment on the performance of your new model. (4 Marks)**

    Code and output: Please refer to the appendix 12.

I decided to use the R package xgboost to create an XGBoost model in order to take on the challenge of fitting a new classifier. Tree boosting may be accomplished with XGBoost, or eXtreme Gradient Boosting, a potent and scalable machine learning technique. Its effectiveness and efficiency are well known, particularly when it comes to structured data.

XGBoost is a fast and efficient implementation of gradient-boosted decision trees. It works by building a group of trees, each of which fixes mistakes produced by the preceding ones. Predictive accuracy is optimised by this iterative method. The three main parameters that are involved are the learning rate (eta), the maximum depth of trees (max_depth), and the loss function (objective).

I started by making sure the Class variable in my dataset was changed to a numeric format that XGBoost could use, with '0' standing for legitimate websites and '1' for phishing websites. 30% of the data was used for testing and 70% of the data was used for training. XGBoost's preferred data structure, xgb.DMatrix format, was used to transform the data once the features and labels were separated.Add the following: objective, which specifies the loss function; max_depth, which indicates the maximum depth of trees; and eta, which indicates the learning rate.

I specified the following settings for the XGBoost model: max_depth to 6 to regulate tree complexity, eta to 0.1 as the learning rate, objective to "binary:logistic" for binary classification, and eval_metric to "auc" for performance evaluation using the Area Under the Curve (AUC) metric. These parameters were used to train the model during a 100-round period. If the test AUC did not improve after 10 rounds, the model was stopped early.

In order to evaluate performance, predictions were made on the test set and a confusion matrix was generated. The XGBoost model's accuracy was determined to be 77.14%. The ROCR software was also used to determine the AUC, and the result was an AUC of 0.7948, indicates strong discriminative capacity.

The comparison table shown that, in terms of accuracy and AUC, the XGBoost model performed better than other classifiers including Decision Tree, Naive Bayes, Bagging, Boosting, and Random Forest. The XGBoost model performed marginally better with an accuracy of 77.14% and an AUC of 0.7948 than the Random Forest model, which had an accuracy of 76.5% and an AUC of 0.7721.

# Appendix

Introduction:

```
rm(list = ls())
Phish = read.csv("PhishingData.csv")
set.seed(33402302) # Your Student ID is the random seed
L = as.data.frame(c(1:50))
L = L[sample(nrow(L), 10, replace = FALSE),]
Phish = Phish[(Phish$A01 %in% L),]
PD = Phish[sample(nrow(Phish), 2000, replace = FALSE),] # sample of 2000 rows

#install.packages("tree")
library(tree)
#install.packages("e1071")
library(e1071)
#install.packages(("ROCR"))
library(ROCR)
```

```
#install.packages("randomForest")
library(randomForest)
#install.packages("adabag")
library(adabag)
#install.packages("rpart")
library(rpart)
#install.packages("dplyr")
library(dplyr)
#install.packages("ggplot2")
library(ggplot2)
#install.packages("tidyr")
library(tidyr)
#install.packages("rpart.plot")
library(rpart.plot)
#install.packages("caret")
library(caret)
```

```
> rm(list = ls())
> Phish = read.csv("PhishingData.csv")
> set.seed(33402302) # Your Student ID is the random seed
> L = as.data.frame(c(1:50))
> L = L[sample(nrow(L), 10, replace = FALSE),]
> Phish = Phish[(Phish$A01 %in% L),]
> PD = Phish[sample(nrow(Phish), 2000, replace = FALSE),] # sample of 2000 rows
> #install.packages("tree")
> library(tree)
> #install.packages("e1071")
> library(e1071)
> #install.packages(("ROCR"))
> library(ROCR)
> #install.packages("randomForest")
> library(randomForest)
randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.
> #install.packages("adabag")
> library(adabag)
Loading required package: rpart
Loading required package: caret
Loading required package: ggplot2
Learn more about the underlying theory at https://ggplot2-book.org/

Attaching package: 'ggplot2'

The following object is masked from 'package:randomForest':

    margin

Loading required package: lattice
Loading required package: foreach
Loading required package: doParallel
Loading required package: iterators
Loading required package: parallel
> #install.packages("rpart")
> library(rpart)
> #install.packages("dplyr")
> library(dplyr)

Attaching package: 'dplyr'

The following object is masked from 'package:randomForest':

    combine

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union

> #install.packages("ggplot2")
> library(ggplot2)
> #install.packages("tidyr")
> library(tidyr)
> #install.packages("rpart.plot")
> library(rpart.plot)
> #install.packages("caret")
> library(caret)
```

1. ***str(PD)***

   ***summary(PD)***

   ***# Ensure the 'Class' column is a factor***
   ***PD$Class = as.factor(PD$Class)***

   ***# Calculate and Print the Proportion of Phishing vs. Legitimate Sites***
   ***cat("Proportion of Phishing vs. Legitimate Sites:\n")***
   ***print(table(PD$Class))***

```
# Label 0 corresponds to a legitimate URL, label 1 to a phishing URL
class_proportion = prop.table(table(PD$Class))
print(class_proportion)

# Obtain Descriptions of the Predictor (Independent) Variables
cat("\nSummary of Predictor Variables:\n")
print(summary(PD))
# Calculate mean and standard deviation for numeric predictors
numeric_num = PD %>% select(where(is.numeric))

# Calculate the standard deviation for each numeric column
Standard_Deviation = sapply(numeric_num, sd, na.rm = TRUE)
print("Standard Deviation of Numeric Variables:")
print(Standard_Deviation)

# Calculate the mean for each numeric column
Mean = sapply(numeric_vars, mean, na.rm = TRUE)
print("Mean of Numeric Variables:")
print(Mean)

# Combine Mean and Standard Deviation into a single summary data frame
numeric_summary = data.frame(
  Variable = names(numeric_num),
  Mean = Mean,
  Standard_Deviation = Standard_Deviation)
print("Summary of Numeric Variables (Mean and Standard Deviation):")
print(numeric_summary)

# Check for Missing Values
cat("\nNumber of Missing Values:\n")
missing_values = sum(is.na(PD))
print(missing_values)
```

```
> str(PD)
'data.frame':   2000 obs. of  26 variables:
 $ A01   : int  49 38 32 32 17 14 50 49 18 22 ...
 $ A02   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A03   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A04   : int  2 2 3 3 3 3 3 3 2 2 ...
 $ A05   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A06   : int  0 0 1 0 0 0 0 0 0 0 ...
 $ A07   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A08   : num  1 1 1 1 1 1 1 1 0.7 1 ...
 $ A09   : int  0 0 0 1 0 0 0 0 0 0 ...
 $ A10   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A11   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A12   : int  403 219 232 504 232 232 NA 232 646 364 ...
 $ A13   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A14   : int  0 1 0 0 0 0 1 0 0 0 ...
 $ A15   : int  0 0 0 0 0 0 1 0 0 0 ...
 $ A16   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A17   : int  1 1 1 1 1 1 0 1 2 1 ...
 $ A18   : int  7 112 23 19 139 6 11 27 31 106 ...
 $ A19   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A20   : int  0 0 1 0 1 0 0 0 0 0 ...
 $ A21   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ A22   : num  0.0418 0.0693 0.0596 0.074 0.0546 ...
 $ A23   : int  0 126 64 108 129 100 0 0 127 38 ...
 $ A24   : num  0.000361 0.001219 0.522907 0.079963 0.522907 ...
 $ A25   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Class: int  0 1 1 1 0 1 0 0 0 1 ...
> summary(PD)
      A01              A02               A03               A04              A05
 Min.   : 9.00   Min.   :  0.000   Min.   :0.000000   Min.   :2.000   Min.   :  0.00000
 1st Qu.:14.00   1st Qu.:  0.000   1st Qu.:0.000000   1st Qu.:2.000   1st Qu.:  0.00000
 Median :18.00   Median :  0.000   Median :0.000000   Median :3.000   Median :  0.00000
 Mean   :26.27   Mean   :  0.211   Mean   :0.002527   Mean   :2.741   Mean   :  0.09673
 3rd Qu.:38.00   3rd Qu.:  0.000   3rd Qu.:0.000000   3rd Qu.:3.000   3rd Qu.:  0.00000
 Max.   :50.00   Max.   :128.000   Max.   :1.000000   Max.   :7.000   Max.   :149.00000
                 NA's   :24        NA's   :21         NA's   :21      NA's   :15
      A06               A07               A08              A09               A10
 Min.   :0.0000   Min.   :0.000000   Min.   :0.1556   Min.   :0.00000   Min.   :0.00000
 1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6667   1st Qu.:0.00000   1st Qu.:0.00000
 Median :0.0000   Median :0.000000   Median :1.0000   Median :0.00000   Median :0.00000
 Mean   :0.1223   Mean   :0.002015   Mean   :0.8430   Mean   :0.02178   Mean   :0.04841
 3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.00000
```

```
  Max.    :1.0000   Max.    :1.000000   Max.    :1.0000   Max.    :1.00000   Max.    :1.00000
  NA's    :13       NA's    :15         NA's    :19       NA's    :26        NA's    :17
       A11               A12               A13               A14               A15
  Min.    :  0.0000   Min.    :  83   Min.    :  0.0000   Min.    :0.0000   Min.    :0.0000
  1st Qu.:  0.0000   1st Qu.:232   1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.0000
  Median :  0.0000   Median :232   Median :  0.0000   Median :0.0000   Median :0.0000
  Mean   :  0.1329   Mean   :321   Mean   :  0.2482   Mean   :0.1463   Mean   :0.1322
  3rd Qu.:  0.0000   3rd Qu.:451   3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
  Max.    :163.0000   Max.    :686   Max.    :447.0000   Max.    :1.0000   Max.    :1.0000
  NA's    :14       NA's    :18   NA's    :30       NA's    :25        NA's    :26
       A16               A17               A18               A19               A20
  Min.    :0.00000   Min.    :0.000   Min.    :   4.00   Min.    :0.00000   Min.    :0.0000
  1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:  14.00   1st Qu.:0.00000   1st Qu.:0.0000
  Median :0.00000   Median :1.000   Median :  32.00   Median :0.00000   Median :0.0000
  Mean   :0.05048   Mean   :1.186   Mean   :  57.15   Mean   :0.09773   Mean   :0.2449
  3rd Qu.:0.00000   3rd Qu.:1.000   3rd Qu.:  89.00   3rd Qu.:0.00000   3rd Qu.:0.0000
  Max.    :1.00000   Max.    :5.000   Max.    :1690.00   Max.    :1.00000   Max.    :1.0000
  NA's    :19       NA's    :20       NA's    :20       NA's    :15        NA's    :24
       A21               A22               A23               A24
  Min.    :0.00000   Min.    :0.007144   Min.    :   0.00   Min.    :0.00000
  1st Qu.:0.00000   1st Qu.:0.051507   1st Qu.:   6.00   1st Qu.:0.00697
  Median :0.00000   Median :0.058155   Median :  97.00   Median :0.07996
  Mean   :0.02733   Mean   :0.055757   Mean   :  66.79   Mean   :0.26173
  3rd Qu.:0.00000   3rd Qu.:0.062861   3rd Qu.: 105.00   3rd Qu.:0.52291
  Max.    :3.00000   Max.    :0.086568   Max.    :1539.00   Max.    :0.52291
  NA's    :24       NA's    :21       NA's    :23       NA's    :22
       A25               Class
  Min.    :0.000000   Min.    :0.0000
  1st Qu.:0.000000   1st Qu.:0.0000
  Median :0.000000   Median :0.0000
  Mean   :0.000182   Mean   :0.3815
  3rd Qu.:0.000000   3rd Qu.:1.0000
  Max.    :0.158000   Max.    :1.0000
  NA's    :28
> # Ensure the 'Class' column is a factor
> PD$Class = as.factor(PD$Class)
> # Calculate and Print the Proportion of Phishing vs. Legitimate Sites
> cat("Proportion of Phishing vs. Legitimate Sites:\n")
Proportion of Phishing vs. Legitimate Sites:
> print(table(PD$Class))

   0    1
1237  763
> class_proportion = prop.table(table(PD$Class))
> print(class_proportion)

     0      1
0.6185 0.3815
> # Obtain Descriptions of the Predictor (Independent) Variables
> cat("\nSummary of Predictor Variables:\n")

Summary of Predictor Variables:
> print(summary(PD))
       A01               A02               A03               A04               A05
 Min.    : 9.00   Min.    :  0.000   Min.    :0.000000   Min.    :2.000   Min.    :  0.00000
 1st Qu.:14.00   1st Qu.:  0.000   1st Qu.:0.000000   1st Qu.:2.000   1st Qu.:  0.00000
 Median :18.00   Median :  0.000   Median :0.000000   Median :3.000   Median :  0.00000
 Mean   :26.27   Mean   :  0.211   Mean   :0.002527   Mean   :2.741   Mean   :  0.09673
 3rd Qu.:38.00   3rd Qu.:  0.000   3rd Qu.:0.000000   3rd Qu.:3.000   3rd Qu.:  0.00000
 Max.    :50.00   Max.    :128.000   Max.    :1.000000   Max.    :7.000   Max.    :149.00000
                 NA's    :24       NA's    :21       NA's    :21       NA's    :15
       A06               A07               A08               A09               A10
 Min.    :0.0000   Min.    :0.000000   Min.    :0.1556   Min.    :0.00000   Min.    :0.00000
 1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6667   1st Qu.:0.00000   1st Qu.:0.00000
 Median :0.0000   Median :0.000000   Median :1.0000   Median :0.00000   Median :0.00000
 Mean   :0.1223   Mean   :0.002015   Mean   :0.8430   Mean   :0.02178   Mean   :0.04841
 3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.00000
 Max.    :1.0000   Max.    :1.000000   Max.    :1.0000   Max.    :1.00000   Max.    :1.00000
 NA's    :13       NA's    :15       NA's    :19       NA's    :26        NA's    :17
       A11               A12               A13               A14               A15
 Min.    :  0.0000   Min.    :  83   Min.    :  0.0000   Min.    :0.0000   Min.    :0.0000
 1st Qu.:  0.0000   1st Qu.:232   1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.0000
 Median :  0.0000   Median :232   Median :  0.0000   Median :0.0000   Median :0.0000
 Mean   :  0.1329   Mean   :321   Mean   :  0.2482   Mean   :0.1463   Mean   :0.1322
 3rd Qu.:  0.0000   3rd Qu.:451   3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
 Max.    :163.0000   Max.    :686   Max.    :447.0000   Max.    :1.0000   Max.    :1.0000
 NA's    :14       NA's    :18   NA's    :30       NA's    :25        NA's    :26
       A16               A17               A18               A19               A20
 Min.    :0.00000   Min.    :0.000   Min.    :   4.00   Min.    :0.00000   Min.    :0.0000
 1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:  14.00   1st Qu.:0.00000   1st Qu.:0.0000
 Median :0.00000   Median :1.000   Median :  32.00   Median :0.00000   Median :0.0000
 Mean   :0.05048   Mean   :1.186   Mean   :  57.15   Mean   :0.09773   Mean   :0.2449
 3rd Qu.:0.00000   3rd Qu.:1.000   3rd Qu.:  89.00   3rd Qu.:0.00000   3rd Qu.:0.0000
 Max.    :1.00000   Max.    :5.000   Max.    :1690.00   Max.    :1.00000   Max.    :1.0000
 NA's    :19       NA's    :20       NA's    :20       NA's    :15        NA's    :24
       A21               A22               A23               A24
 Min.    :0.00000   Min.    :0.007144   Min.    :   0.00   Min.    :0.00000
 1st Qu.:0.00000   1st Qu.:0.051507   1st Qu.:   6.00   1st Qu.:0.00697
 Median :0.00000   Median :0.058155   Median :  97.00   Median :0.07996
 Mean   :0.02733   Mean   :0.055757   Mean   :  66.79   Mean   :0.26173
 3rd Qu.:0.00000   3rd Qu.:0.062861   3rd Qu.: 105.00   3rd Qu.:0.52291
 Max.    :3.00000   Max.    :0.086568   Max.    :1539.00   Max.    :0.52291
 NA's    :24       NA's    :21       NA's    :23       NA's    :22
       A25               Class
 Min.    :0.000000   0:1237
 1st Qu.:0.000000   1: 763
 Median :0.000000
 Mean   :0.000182
 3rd Qu.:0.000000
 Max.    :0.158000
 NA's    :28
```

```
> # Calculate mean and standard deviation for numeric predictors
> numeric_num = PD %>% select(where(is.numeric))
> # Calculate the standard deviation for each numeric column
> Standard_Deviation = sapply(numeric_num, sd, na.rm = TRUE)
> print("Standard Deviation of Numeric Variables:")
[1] "Standard Deviation of Numeric Variables:"
> print(Standard_Deviation)
      A01       A02       A03       A04       A05       A06       A07       A08       A09
 14.32993   3.02638   0.05021   0.55966   3.38175   0.32771   0.04486   0.21905   0.14601
      A10       A11       A12       A13       A14       A15       A16       A17       A18
  0.21469   3.68884 143.77314  10.09004   0.35353   0.33881   0.21899   0.60411  89.46383
      A19       A20       A21       A22       A23       A24       A25
  0.29703   0.43016   0.19163   0.01075  70.65474   0.25188   0.00456
> # Calculate the mean for each numeric column
> Mean = sapply(numeric_num, mean, na.rm = TRUE)
> print("Mean of Numeric Variables:")
[1] "Mean of Numeric Variables:"
> print(Mean)
      A01       A02       A03       A04       A05       A06       A07       A08       A09
2.627e+01 2.110e-01 2.527e-03 2.741e+00 9.673e-02 1.223e-01 2.015e-03 8.430e-01 2.178e-02
      A10       A11       A12       A13       A14       A15       A16       A17       A18
4.841e-02 1.329e-01 3.210e+02 2.482e-01 1.463e-01 1.322e-01 5.048e-02 1.186e+00 5.715e+01
      A19       A20       A21       A22       A23       A24       A25
9.773e-02 2.449e-01 2.733e-02 5.576e-02 6.679e+01 2.617e-01 1.820e-04
> # Combine Mean and Standard Deviation into a single summary data frame
> numeric_summary = data.frame(
+   Variable = names(numeric_num),
+   Mean = Mean,
+   Standard_Deviation = Standard_Deviation
+ )
> print("Summary of Numeric Variables (Mean and Standard Deviation):")
[1] "Summary of Numeric Variables (Mean and Standard Deviation):"
> print(numeric_summary)
    Variable      Mean Standard_Deviation
A01      A01 2.627e+01           14.32993
A02      A02 2.110e-01            3.02638
A03      A03 2.527e-03            0.05021
A04      A04 2.741e+00            0.55966
A05      A05 9.673e-02            3.38175
A06      A06 1.223e-01            0.32771
A07      A07 2.015e-03            0.04486
A08      A08 8.430e-01            0.21905
A09      A09 2.178e-02            0.14601
A10      A10 4.841e-02            0.21469
A11      A11 1.329e-01            3.68884
A12      A12 3.210e+02          143.77314
A13      A13 2.482e-01           10.09004
A14      A14 1.463e-01            0.35353
A15      A15 1.322e-01            0.33881
A16      A16 5.048e-02            0.21899
A17      A17 1.186e+00            0.60411
A18      A18 5.715e+01           89.46383
A19      A19 9.773e-02            0.29703
A20      A20 2.449e-01            0.43016
A21      A21 2.733e-02            0.19163
A22      A22 5.576e-02            0.01075
A23      A23 6.679e+01           70.65474
A24      A24 2.617e-01            0.25188
A25      A25 1.820e-04            0.00456
> # Check for Missing Values
> cat("\nNumber of Missing Values:\n")

Number of Missing Values:
> missing_values = sum(is.na(PD))
> print(missing_values)
[1] 500
```

2. ***PD = na.omit(PD)***
   ***summary(PD)***

```
> PD = na.omit(PD)
> summary(PD)
       A01              A02               A03               A04               A05          
 Min.   : 9.00   Min.   : 0.0000   Min.   :0.000000   Min.   :2.000   Min.   :  0.0000  
 1st Qu.:14.00   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.:2.000   1st Qu.:  0.0000  
 Median :18.00   Median : 0.0000   Median :0.000000   Median :3.000   Median :  0.0000  
 Mean   :26.11   Mean   : 0.1469   Mean   :0.002566   Mean   :2.738   Mean   :  0.1232  
 3rd Qu.:38.00   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.:3.000   3rd Qu.:  0.0000  
 Max.   :50.00   Max.   :16.0000   Max.   :1.000000   Max.   :7.000   Max.   :149.0000  
       A06               A07               A08               A09               A10          
 Min.   :0.0000   Min.   :0.000000   Min.   :0.1556   Min.   :0.00000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.:0.000000   1st Qu.:0.6842   1st Qu.:0.00000   1st Qu.:0.0000  
 Median :0.0000   Median :0.000000   Median :1.0000   Median :0.00000   Median :0.0000  
 Mean   :0.1212   Mean   :0.002566   Mean   :0.8467   Mean   :0.02117   Mean   :0.0526  
 3rd Qu.:0.0000   3rd Qu.:0.000000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.0000  
 Max.   :1.0000   Max.   :1.000000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000  
       A11               A12               A13               A14               A15          
 Min.   :  0.0000   Min.   : 83.0   Min.   :  0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:  0.0000   1st Qu.:232.0   1st Qu.:  0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
 Median :  0.0000   Median :232.0   Median :  0.0000   Median :0.0000   Median :0.0000  
 Mean   :  0.1578   Mean   :322.8   Mean   :  0.3137   Mean   :0.1443   Mean   :0.1373  
 3rd Qu.:  0.0000   3rd Qu.:451.0   3rd Qu.:  0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
 Max.   :163.0000   Max.   :686.0   Max.   :447.0000   Max.   :1.0000   Max.   :1.0000  
       A16               A17               A18               A19               A20          
 Min.   :0.00000   Min.   :0.000   Min.   :   4.00   Min.   :0.00000   Min.   :0.0000  
 1st Qu.:0.00000   1st Qu.:1.000   1st Qu.:  14.00   1st Qu.:0.00000   1st Qu.:0.0000  
 Median :0.00000   Median :1.000   Median :  32.00   Median :0.00000   Median :0.0000  
 Mean   :0.05131   Mean   :1.178   Mean   :  57.62   Mean   :0.09814   Mean   :0.2495  
 3rd Qu.:0.00000   3rd Qu.:1.000   3rd Qu.:  89.00   3rd Qu.:0.00000   3rd Qu.:0.0000  
 Max.   :1.00000   Max.   :5.000   Max.   :1690.00   Max.   :1.00000   Max.   :1.0000  
       A21               A22               A23               A24          
 Min.   :0.00000   Min.   :0.007144   Min.   :   0.00   Min.   :0.000000  
 1st Qu.:0.00000   1st Qu.:0.051475   1st Qu.:   7.00   1st Qu.:0.007505  
 Median :0.00000   Median :0.058086   Median : 100.00   Median :0.079963  
 Mean   :0.02822   Mean   :0.055801   Mean   :  67.43   Mean   :0.260638  
 3rd Qu.:0.00000   3rd Qu.:0.062896   3rd Qu.: 105.00   3rd Qu.:0.522907  
 Max.   :3.00000   Max.   :0.086568   Max.   :1539.00   Max.   :0.522907  
       A25          Class      
 Min.   :0.0000000   0:956  
 1st Qu.:0.0000000   1:603  
 Median :0.0000000          
 Mean   :0.0002303          
 3rd Qu.:0.0000000          
 Max.   :0.1580000          
```

3. ***set.seed(33402302) #Student ID as random seed***
   ***train.row = sample(1:nrow(PD), 0.7*nrow(PD))***
   ***PD.train = PD[train.row, ]***
   ***PD.test = PD[-train.row, ]***

   ***# Omitting the NA values***
   ***PD.train = na.omit(PD.train)***
   ***PD.test = na.omit(PD.test)***

   ***# Ensure Class is a factor in the training set***
   ***PD.train$Class = as.factor(PD.train$Class)***
   ***PD.test$Class = as.factor(PD.test$Class)***

```
> set.seed(33402302) #Student ID as random seed
> train.row = sample(1:nrow(PD), 0.7*nrow(PD))
> PD.train = PD[train.row, ]
> PD.test = PD[-train.row, ]
> # Omitting the NA values
> PD.train = na.omit(PD.train)
> PD.test = na.omit(PD.test)
> # Ensure Class is a factor in the training set
> PD.train$Class = as.factor(PD.train$Class)
> PD.test$Class = as.factor(PD.test$Class)
```

4. ***# Calculate a decision tree***
   ***PD.tree = tree(Class ~., data = PD.train)***
   ***summary(PD.tree)***

   ***# Calculate naive bayes***
   ***PD.bayes = naiveBayes(Class ~. , data = PD.train)***
   ***summary(PD.bayes)***

   ***# Bagging***
   ***PD.bag = bagging(Class ~. , data = PD.train, mfinal=5)***
   ***summary(PD.bag)***

   ***#Boosting***
   ***PD.boost = boosting(Class ~. , data = PD.train, mfinal=10)***
   ***summary(PD.boost)***

   ***# Random Forest***
   ***PD.rf = randomForest(Class ~. , data = PD.train, na.action = na.exclude)***
   ***summary(PD.rf)***

```
> # Calculate a decision tree
> PD.tree = tree(Class ~., data = PD.train)
> summary(PD.tree)

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A18" "A01" "A23"
Number of terminal nodes:  5
Residual mean deviance:  1.056 = 1147 / 1086
Misclassification error rate: 0.2401 = 262 / 1091
> # Calculate naive bayes
> PD.bayes = naiveBayes(Class ~. , data = PD.train)
> summary(PD.bayes)
          Length Class  Mode
apriori    2     table  numeric
tables    25     -none- list
levels     2     -none- character
isnumeric 25     -none- logical
call       4     -none- call
> # Bagging
> PD.bag = bagging(Class ~. , data = PD.train, mfinal=5)
> summary(PD.bag)
           Length Class   Mode
formula       3   formula call
trees         5   -none-  list
votes      2182   -none-  numeric
prob       2182   -none-  numeric
class      1091   -none-  character
samples    5455   -none-  numeric
importance   25   -none-  numeric
terms         3   terms   call
call          4   -none-  call
> #Boosting
> PD.boost = boosting(Class ~. , data = PD.train, mfinal=10)
> summary(PD.boost)
           Length Class   Mode
formula       3   formula call
trees        10   -none-  list
weights      10   -none-  numeric
votes      2182   -none-  numeric
prob       2182   -none-  numeric
class      1091   -none-  character
importance   25   -none-  numeric
terms         3   terms   call
call          4   -none-  call
> # Random Forest
> PD.rf = randomForest(Class ~. , data = PD.train, na.action = na.exclude)
> summary(PD.rf)
                Length Class  Mode
call               4   -none- call
type               1   -none- character
predicted       1091   factor numeric
err.rate        1500   -none- numeric
confusion          6   -none- numeric
votes           2182   matrix numeric
oob.times       1091   -none- numeric
classes            2   -none- character
importance        25   -none- numeric
importanceSD       0   -none- NULL
localImportance    0   -none- NULL
proximity          0   -none- NULL
ntree              1   -none- numeric
mtry               1   -none- numeric
forest            14   -none- list
y               1091   factor numeric
test               0   -none- NULL
inbag              0   -none- NULL
terms              3   terms  call
```

5.
```
#==============#
# Decision Tree #
#==============#
# do predictions as classes and draw a table
PD.predtree = predict(PD.tree, PD.test, type = "class")
tree_conf = table(Predicted_Class = PD.predtree, Actual_Class = PD.test$Class)
cat("\n#Decision Tree Confusion\n")
print(tree_conf)
tree_sum = sum(tree_conf)
tree_sum2 = sum(diag(tree_conf))
tree_acc = (round(tree_sum2 / tree_sum, 4)) * 100
tree_acc
#==============#
#  Naive Bayes  #
#==============#
PD.predbayes = predict(PD.bayes, PD.test)
naive_conf = table(Predicted_Class = PD.predbayes, Actual_Class = PD.test$Class)
cat("\n#NaiveBayes Confusion\n")
print(naive_conf)
nb_sum = sum(naive_conf)
nb_sum2 = sum(diag(naive_conf))
naive_acc = (round(nb_sum2 / nb_sum, 4)) * 100
naive_acc
#==============#
#    Bagging    #
#==============#
PDpred.bag = predict.bagging(PD.bag, PD.test)
bag_conf = PDpred.bag$confusion
cat("\n#Bagging Confusion\n")
print(bag_conf)
bag_sum = sum(bag_conf)
bag_sum2 = sum(diag(bag_conf))
bag_acc = (round(bag_sum2 / bag_sum, 4)) * 100
bag_acc
#==============#
#    Boosting   #
#==============#
PDpred.boost = predict.boosting(PD.boost, newdata=PD.test)
boost_conf = PDpred.boost$confusion
cat("\n#Boosting Confusion\n")
print(boost_conf)
boost_sum = sum(boost_conf)
boost_sum2 = sum(diag(boost_conf))
boost_acc = (round(boost_sum2 / boost_sum, 4)) * 100
boost_acc
#==============#
# Random Forest #
#==============#
PDpredrf = predict(PD.rf, PD.test)
random_conf = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$Class)
cat("\n#Random Forest Confusion\n")
```

***print(random_conf)***

***random_sum = sum(random_conf)***
***random_sum2 = sum(diag(random_conf))***
***random_acc = (round(random_sum2 / random_sum, 4)) * 100***

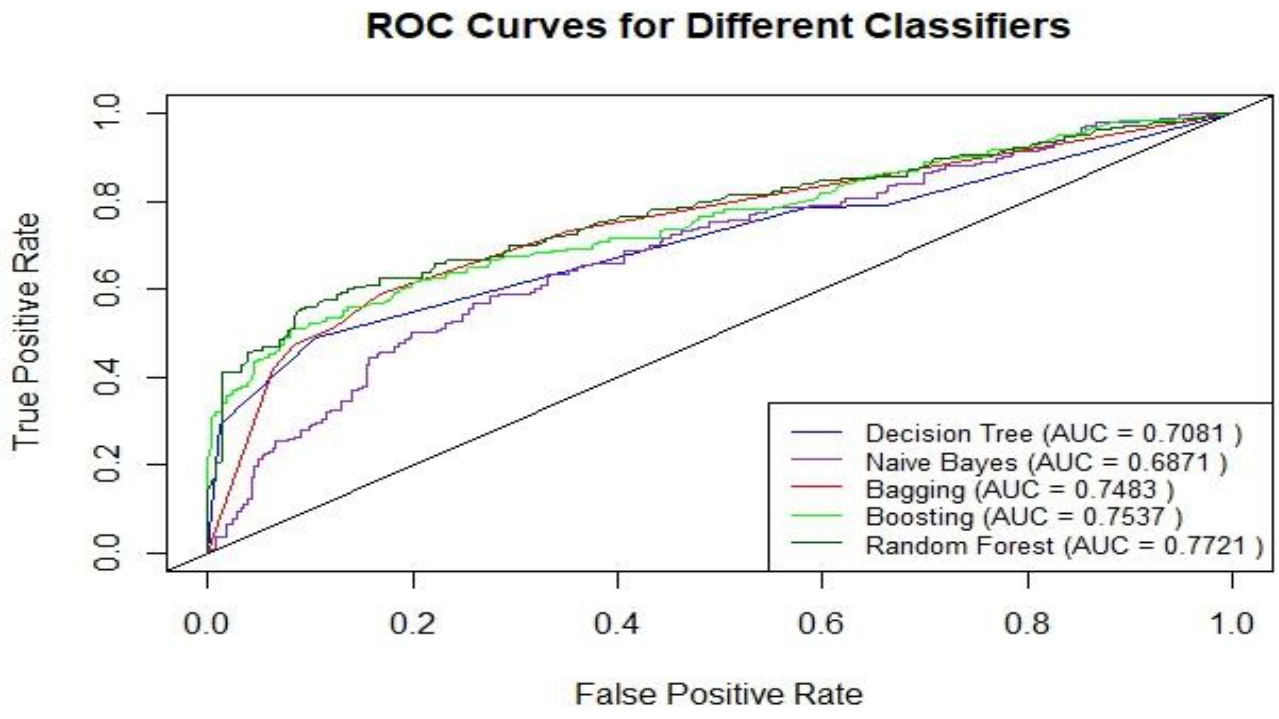
```
> # do predictions as classes and draw a table
> PD.predtree = predict(PD.tree, PD.test, type = "class")
> tree_conf = table(Predicted_Class = PD.predtree, Actual_Class = PD.test$class)
> cat("\n#Decision Tree Confusion\n")

#Decision Tree Confusion
> print(tree_conf)
               Actual_Class
Predicted_Class   0   1
              0 256  93
              1  30  89
> tree_sum = sum(tree_conf)
> tree_sum2 = sum(diag(tree_conf))
> tree_acc = (round(tree_sum2 / tree_sum, 4)) * 100
> tree_acc
[1] 73.72
> PD.predbayes = predict(PD.bayes, PD.test)
> naive_conf = table(Predicted_Class = PD.predbayes, Actual_Class = PD.test$class)
> cat("\n#NaiveBayes Confusion\n")

#NaiveBayes Confusion
> print(naive_conf)
               Actual_Class
Predicted_Class   0   1
              0   7   0
              1 279 182
> nb_sum = sum(naive_conf)
> nb_sum2 = sum(diag(naive_conf))
> naive_acc = (round(nb_sum2 / nb_sum, 4)) * 100
> naive_acc
[1] 40.38
> PDpred.bag = predict.bagging(PD.bag, PD.test)
> bag_conf = PDpred.bag$confusion
> cat("\n#Bagging Confusion\n")

#Bagging Confusion
> print(bag_conf)
               Observed Class
Predicted Class   0   1
              0 251  89
              1  35  93
> bag_sum = sum(bag_conf)
> bag_sum2 = sum(diag(bag_conf))
> bag_acc = (round(bag_sum2 / bag_sum, 4)) * 100
> bag_acc
[1] 73.5
> PDpred.boost = predict.boosting(PD.boost, newdata=PD.test)
> boost_conf = PDpred.boost$confusion
> cat("\n#Boosting Confusion\n")

#Boosting Confusion
> print(boost_conf)
               Observed Class
Predicted Class   0   1
              0 238  79
              1  48 103
> boost_sum = sum(boost_conf)
> boost_sum2 = sum(diag(boost_conf))
> boost_acc = (round(boost_sum2 / boost_sum, 4)) * 100
> boost_acc
[1] 72.86
> PDpredrf = predict(PD.rf, PD.test)
> random_conf = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$class)
> cat("\n#Random Forest Confusion\n")

#Random Forest Confusion
> print(random_conf)
               Actual_Class
Predicted_Class   0   1
              0 253  77
              1  33 105
> random_sum = sum(random_conf)
> random_sum2 = sum(diag(random_conf))
> random_acc = (round(random_sum2 / random_sum, 4)) * 100
> random_acc
[1] 76.5
```

6. ***# Create a blank plot***
***plot(0, 0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate",***
***xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for Different Classifiers")***
***abline(0, 1, col = "black")***

```
# Decision Tree
PD.pred.tree = predict(PD.tree, PD.test, type = "vector")
PDDpred = ROCR::prediction(PD.pred.tree[, 2], PD.test$Class)
PDDperf = performance(PDDpred, "tpr", "fpr")
plot(PDDperf, add = TRUE, col = "blue")
PDDauc = performance(PDDpred, "auc")@y.values[[1]]
print(paste("Decision Tree AUC:", PDDauc))

# Naive Bayes
PDpred.bayes = predict(PD.bayes, PD.test, type = "raw")
PDBpred = ROCR::prediction(PDpred.bayes[, 2], PD.test$Class)
PDBperf = performance(PDBpred, "tpr", "fpr")
plot(PDBperf, add = TRUE, col = "blueviolet")
PDBauc = performance(PDBpred, "auc")@y.values[[1]]
print(paste("Naive Bayes AUC:", PDBauc))

# Bagging
PDpred.bag = predict.bagging(PD.bag, PD.test)
PDBagpred = ROCR::prediction(PDpred.bag$prob[, 2], PD.test$Class)
PDBagperf = performance(PDBagpred, "tpr", "fpr")
plot(PDBagperf, add = TRUE, col = "red")
PDBagauc = performance(PDBagpred, "auc")@y.values[[1]]
print(paste("Bagging AUC:", PDBagauc))

# Boosting
PDpred.boost = predict.boosting(PD.boost, newdata = PD.test)
PDBoostpred = ROCR::prediction(PDpred.boost$prob[, 2], PD.test$Class)
PDBoostperf = performance(PDBoostpred, "tpr", "fpr")
plot(PDBoostperf, add = TRUE, col = "green")
PDBoostauc = performance(PDBoostpred, "auc")@y.values[[1]]
print(paste("Boosting AUC:", PDBoostauc))

# Random Forest (assuming you have a random forest model named PD.rf)
PDpred.rf = predict(PD.rf, PD.test, type = "prob")
PDFpred = ROCR::prediction(PDpred.rf[, 2], PD.test$Class)
PDFperf = performance(PDFpred, "tpr", "fpr")
plot(PDFperf, add = TRUE, col = "darkgreen")
PDFauc = performance(PDFpred, "auc")@y.values[[1]]
print(paste("Random Forest AUC:", PDFauc))

# Add legend
legend("bottomright", legend = c(
  paste("Decision Tree (AUC =", round(PDDauc, 4), ")"),
  paste("Naive Bayes (AUC =", round(PDBauc, 4), ")"),
  paste("Bagging (AUC =", round(PDBagauc, 4), ")"),
  paste("Boosting (AUC =", round(PDBoostauc, 4), ")"),
  paste("Random Forest (AUC =", round(PDFauc, 4), ")")
```

***), col = c("blue", "blueviolet", "red", "green", "darkgreen"), lty = 1, cex = 0.8)***

```
> # Create a blank plot
> plot(0, 0, type = "n", xlab = "False Positive Rate", ylab = "True Positive Rate",
+      xlim = c(0, 1), ylim = c(0, 1), main = "ROC Curves for Different Classifiers")
> abline(0, 1, col = "black")
> # Decision Tree
> PD.pred.tree = predict(PD.tree, PD.test, type = "vector")
> PDDpred = ROCR::prediction(PD.pred.tree[, 2], PD.test$class)
> PDDperf = performance(PDDpred, "tpr", "fpr")
> plot(PDDperf, add = TRUE, col = "blue")
> PDDauc = performance(PDDpred, "auc")@y.values[[1]]
> print(paste("Decision Tree AUC:", PDDauc))
[1] "Decision Tree AUC: 0.708070775378468"
> # Naive Bayes
> PDpred.bayes = predict(PD.bayes, PD.test, type = "raw")
> PDBpred = ROCR::prediction(PDpred.bayes[, 2], PD.test$class)
> PDBperf = performance(PDBpred, "tpr", "fpr")
> plot(PDBperf, add = TRUE, col = "blueviolet")
> PDBauc = performance(PDBpred, "auc")@y.values[[1]]
> print(paste("Naive Bayes AUC:", PDBauc))
[1] "Naive Bayes AUC: 0.687082148620609"
> # Bagging
> PDpred.bag = predict.bagging(PD.bag, PD.test)
> PDBagpred = ROCR::prediction(PDpred.bag$prob[, 2], PD.test$class)
> PDBagperf = performance(PDBagpred, "tpr", "fpr")
> plot(PDBagperf, add = TRUE, col = "red")
> PDBagauc = performance(PDBagpred, "auc")@y.values[[1]]
> print(paste("Bagging AUC:", PDBagauc))
[1] "Bagging AUC: 0.748328594482441"
> # Boosting
> PDpred.boost = predict.boosting(PD.boost, newdata = PD.test)
> PDBoostpred = ROCR::prediction(PDpred.boost$prob[, 2], PD.test$class)
> PDBoostperf = performance(PDBoostpred, "tpr", "fpr")
> plot(PDBoostperf, add = TRUE, col = "green")
> PDBoostauc = performance(PDBoostpred, "auc")@y.values[[1]]
> print(paste("Boosting AUC:", PDBoostauc))
[1] "Boosting AUC: 0.753746253746254"
> # Random Forest
> PDpred.rf = predict(PD.rf, PD.test, type = "prob")
> PDFpred = ROCR::prediction(PDpred.rf[, 2], PD.test$class)
> PDFperf = performance(PDFpred, "tpr", "fpr")
> plot(PDFperf, add = TRUE, col = "darkgreen")
> PDFauc = performance(PDFpred, "auc")@y.values[[1]]
> print(paste("Random Forest AUC:", PDFauc))
[1] "Random Forest AUC: 0.772083685545224"
> # Add legend
> legend("bottomright", legend = c(
+   paste("Decision Tree (AUC =", round(PDDauc, 4), ")"),
+   paste("Naive Bayes (AUC =", round(PDBauc, 4), ")"),
+   paste("Bagging (AUC =", round(PDBagauc, 4), ")"),
+   paste("Boosting (AUC =", round(PDBoostauc, 4), ")"),
+   paste("Random Forest (AUC =", round(PDFauc, 4), ")")
+ ), col = c("blue", "blueviolet", "red", "green", "darkgreen"), lty = 1, cex = 0.8)
```



7. ***# Create a comparison table***
***comparison_table = data.frame(***
***Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),***
***AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc),***
***Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc)***
***)***

***# Print the comparison table***
***print(comparison_table)***

***# Determine the best classifier based on AUC***
***best_auc = comparison_table[which.max(comparison_table$AUC), ]***
***print(paste("Best Classifier Based on AUC:", best_auc$Classifier))***

***# Determine the best classifier based on Accuracy***
***best_accuracy = comparison_table[which.max(comparison_table$Accuracy), ]***
***print(paste("Best Classifier Based on Accuracy:", best_accuracy$Classifier))***

```
> # Create a comparison table
> comparison_table = data.frame(
+   Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
+   AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc),
+   Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc)
+ )
> # Print the comparison table
> print(comparison_table)
     Classifier       AUC Accuracy
1 Decision Tree 0.7080708    73.72
2   Naive Bayes 0.6870821    40.38
3       Bagging 0.7483286    73.50
4      Boosting 0.7537463    72.86
5 Random Forest 0.7720837    76.50
> # Determine the best classifier based on AUC
> best_auc = comparison_table[which.max(comparison_table$AUC), ]
> print(paste("Best Classifier Based on AUC:", best_auc$Classifier))
[1] "Best Classifier Based on AUC: Random Forest"
> # Determine the best classifier based on Accuracy
> best_accuracy = comparison_table[which.max(comparison_table$Accuracy), ]
> print(paste("Best Classifier Based on Accuracy:", best_accuracy$Classifier))
[1] "Best Classifier Based on Accuracy: Random Forest"
```

8. ***#Attribute importance***
***cat("\n#Decision Tree Attribute Importance\n")***
***print(summary(PD.tree))***

***cat("\n#Baging Attribute Importance\n")***
***print(PD.bag$importance)***

***cat("\n#Boosting Attribute Importance\n")***
***print(PD.boost$importance)***

***cat("\n#Random Forest Attribute Importance\n")***
***print(PD.rf$importance)***

***cat("\n#Naive Bayes Attribute Importance\n")***

***print("Naive Bayes just computes the probability of each variable supplied, hence it is impossible to assess the significance of the variables it uses.")***

```
> #Attribute importance
> cat("\n#Decision Tree Attribute Importance\n")

#Decision Tree Attribute Importance
> print(summary(PD.tree))

Classification tree:
tree(formula = Class ~ ., data = PD.train)
Variables actually used in tree construction:
[1] "A18" "A01" "A23"
Number of terminal nodes:  5
Residual mean deviance:  1.056 = 1147 / 1086
Misclassification error rate: 0.2401 = 262 / 1091
> cat("\n#Baging Attribute Importance\n")

#Baging Attribute Importance
> print(PD.bag$importance)
       A01        A02        A03        A04        A05        A06        A07        A08
33.0447902  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  0.0000000  3.4850003
       A09        A10        A11        A12        A13        A14        A15        A16
 0.0000000  0.0000000  0.0000000  1.2729251  0.0000000  0.0000000  0.0000000  0.0000000
       A17        A18        A19        A20        A21        A22        A23        A24
 0.0000000 23.1535455  0.0000000  0.0000000  0.0000000 11.9765956 26.4212016  0.6459416
       A25
 0.0000000
> cat("\n#Boosting Attribute Importance\n")

#Boosting Attribute Importance
> print(PD.boost$importance)
       A01        A02        A03        A04        A05        A06        A07        A08
19.0351535  0.3756252  0.3336242  0.9165331  0.0000000  0.3441329  0.0000000  8.8838858
       A09        A10        A11        A12        A13        A14        A15        A16
 0.1448748  0.1756638  0.0000000  5.1181532  0.0000000  1.5482712  0.5191115  0.0000000
       A17        A18        A19        A20        A21        A22        A23        A24
 2.1065364 17.6186241  0.4164074  0.6577795  0.9484008 13.0633259 25.7195658  2.0743311
       A25
 0.0000000
> cat("\n#Random Forest Attribute Importance\n")

#Random Forest Attribute Importance
> print(PD.rf$importance)
    MeanDecreaseGini
A01        83.57275135
A02         6.28562974
A03         0.32916609
A04         9.22463842
A05         0.28084671
A06         5.23370034
A07         0.09748373
A08        37.59087238
A09         1.70536955
A10         2.32562515
A11         2.33891809
A12        26.76540550
A13         0.07427272
A14         9.81408101
A15         5.63760302
A16         4.25702765
A17        11.39765136
A18        73.17642206
A19         5.05078451
A20         8.77718166
A21         1.90744477
A22        66.91577750
A23        75.41781011
A24        26.43868348
A25         0.06909587
> cat("\n#Naive Bayes Attribute Importance\n")

#Naive Bayes Attribute Importance
> print("Naive Bayes just computes the probability of each variable supplied, hence it is imp
ossible to assess the significance of the variables it uses.")
[1] "Naive Bayes just computes the probability of each variable supplied, hence it is impossi
ble to assess the significance of the variables it uses."
```

9. ***# Train a simple decision tree classifier***
   ***simple_tree = rpart(Class ~ ., data = PD.train, method = "class", control = rpart.control(cp = 0.01))***

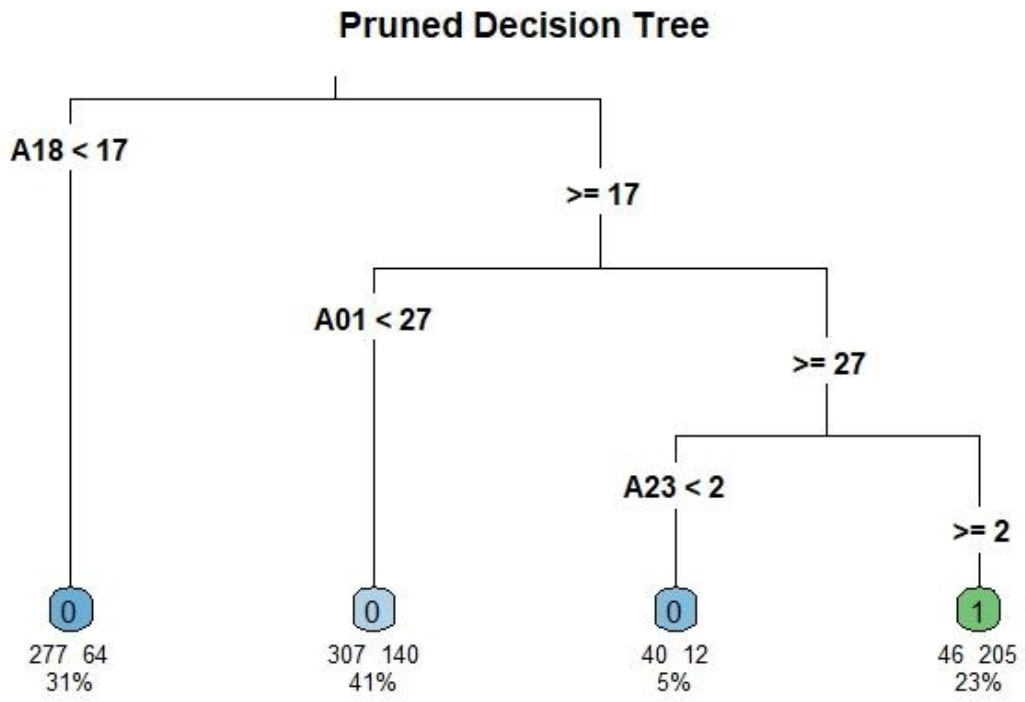
   ***# Prune the tree to simplify it***
   ***min = which.min(simple_tree$cptable[,"xerror"])***
   ***pruned_tree = prune(simple_tree, cp = simple_tree$cptable[min, "CP"])***

   ***# Plot the pruned tree***
   ***rpart.plot(pruned_tree, type = 3, extra = 101, under = TRUE, fallen.leaves = TRUE, main = "Pruned Decision Tree")***

   ***# Describe the model***

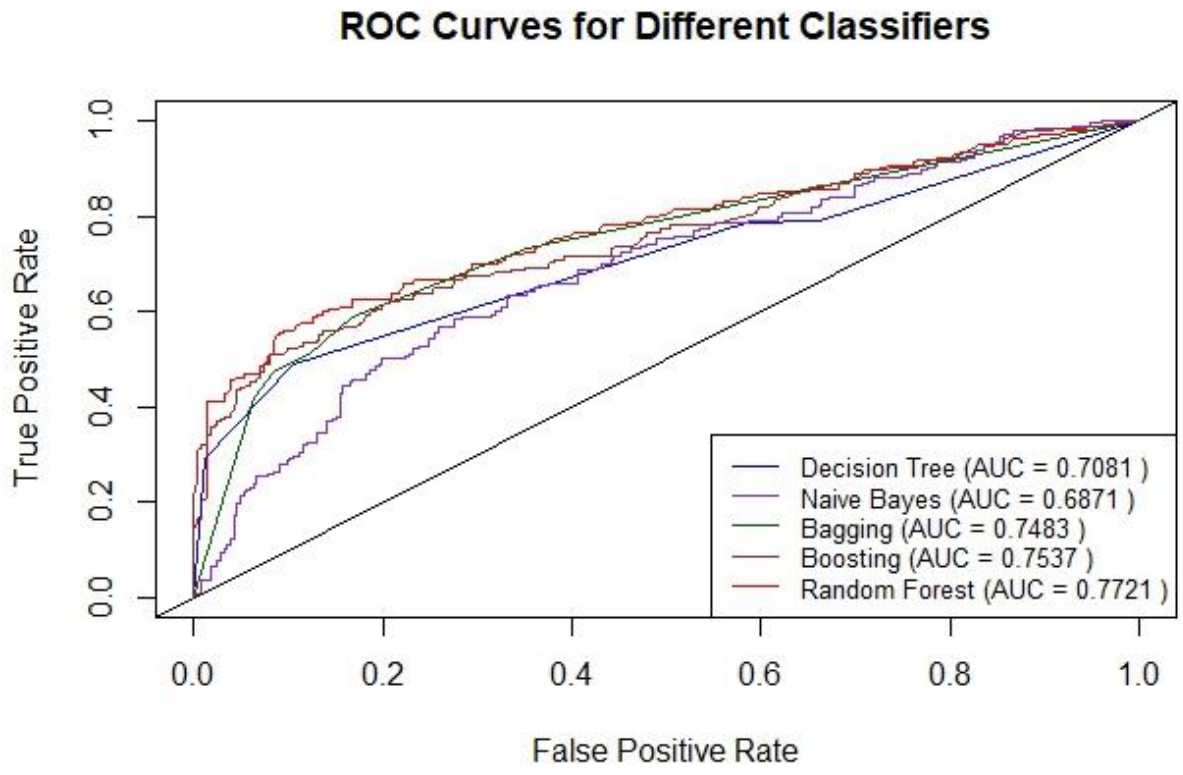
```
print("Pruned Decision Tree Description:")
print(pruned_tree)

# Evaluate model performance using test data
pred_pruned_tree = predict(pruned_tree, PD.test, type = "class")
pruned_tree_conf = table(Predicted_Class = pred_pruned_tree, Actual_Class = PD.test$Class)
cat("\n#Pruned Decision Tree Confusion Matrix\n")
print(pruned_tree_conf)

pruned_sum = sum(pruned_tree_conf)
pruned_sum2 = sum(diag(pruned_tree_conf))

pruned_tree_acc = (round(pruned_sum2 / pruned_sum, 4)) * 100
pruned_tree_acc

# Compute AUC for pruned tree
pred_pruned_tree_prob = predict(pruned_tree, PD.test, type = "prob")
pruned_tree_pred = ROCR::prediction(pred_pruned_tree_prob[, 2], PD.test$Class)
pruned_tree_perf = performance(pruned_tree_pred, "tpr", "fpr")
pruned_tree_auc = performance(pruned_tree_pred, "auc")@y.values[[1]]
pruned_tree_auc

# Comparison with previous classifiers
comparison_table = rbind(
  comparison_table,
  data.frame(Classifier = "Pruned Decision Tree", AUC = pruned_tree_auc, Accuracy =
pruned_tree_acc)
)

print("Comparison Table with Pruned Decision Tree:")
print(comparison_table)
```

```
> # Train a simple decision tree classifier
> simple_tree = rpart(Class ~ ., data = PD.train, method = "class", control = rpart.control(c
p = 0.01))
> # Prune the tree to simplify it
> min = which.min(simple_tree$cptable[,"xerror"])
> pruned_tree = prune(simple_tree, cp = simple_tree$cptable[min, "CP"])
> # Plot the pruned tree
> rpart.plot(pruned_tree, type = 3, extra = 101, under = TRUE, fallen.leaves = TRUE, main =
"Pruned Decision Tree")
> # Describe the model
> print("Pruned Decision Tree Description:")
[1] "Pruned Decision Tree Description:"
> print(pruned_tree)
n= 1091

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 1091 421 0 (0.6141155 0.3858845)
   2) A18< 16.5 341  64 0 (0.8123167 0.1876833) *
   3) A18>=16.5 750 357 0 (0.5240000 0.4760000)
     6) A01< 27 447 140 0 (0.6868009 0.3131991) *
     7) A01>=27 303  86 1 (0.2838284 0.7161716)
      14) A23< 1.5 52  12 0 (0.7692308 0.2307692) *
      15) A23>=1.5 251  46 1 (0.1832669 0.8167331) *
> # Evaluate model performance using test data
> pred_pruned_tree = predict(pruned_tree, PD.test, type = "class")
> pruned_tree_conf = table(Predicted_Class = pred_pruned_tree, Actual_Class = PD.test$Class)
> cat("\n#Pruned Decision Tree Confusion Matrix\n")

#Pruned Decision Tree Confusion Matrix
> print(pruned_tree_conf)
               Actual_Class
Predicted_Class   0   1
              0 256  93
              1  30  89
> pruned_sum = sum(pruned_tree_conf)
> pruned_sum2 = sum(diag(pruned_tree_conf))
> pruned_tree_acc = (round(pruned_sum2 / pruned_sum, 4)) * 100
> pruned_tree_acc
[1] 73.72
> # Compute AUC for pruned tree
> pred_pruned_tree_prob = predict(pruned_tree, PD.test, type = "prob")
> pruned_tree_pred = ROCR::prediction(pred_pruned_tree_prob[, 2], PD.test$Class)
> pruned_tree_perf = performance(pruned_tree_pred, "tpr", "fpr")
> pruned_tree_auc = performance(pruned_tree_pred, "auc")@y.values[[1]]
> pruned_tree_auc
[1] 0.6953623
> # Comparison with previous classifiers
> comparison_table = rbind(
+   comparison_table,
+   data.frame(Classifier = "Pruned Decision Tree", AUC = pruned_tree_auc, Accuracy = pruned_
tree_acc)
+ )
> print("Comparison Table with Pruned Decision Tree:")
[1] "Comparison Table with Pruned Decision Tree:"
> print(comparison_table)
             Classifier       AUC Accuracy
1         Decision Tree 0.7080708    73.72
2           Naive Bayes 0.6870821    40.38
3               Bagging 0.7483286    73.50
4              Boosting 0.7537463    72.86
5         Random Forest 0.7720837    76.50
6 Pruned Decision Tree 0.6953623    73.72
```



10.
```
# Predict using the existing Random Forest model on the test set
PDpredrf = predict(PD.rf, PD.test, type = "class")
random_conf = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$Class)
cat("\n# Random Forest Confusion Matrix\n")
print(random_conf)

# Calculate accuracy
random_sum = sum(random_conf)
random_sum2 = sum(diag(random_conf))
random_acc = (round(random_sum2 / random_sum, 4)) * 100
cat("Random Forest Accuracy: ", random_acc, "%\n")

# Calculate AUC for the existing Random Forest model
PDpred_rf_prob = predict(PD.rf, PD.test, type = "prob")
new_predrf = ROCR::prediction(PDpred_rf_prob[, 2], PD.test$Class)
new_prefrf = performance(new_predrf, "tpr", "fpr")
new_auc = performance(new_predrf, "auc")@y.values[[1]]
cat("Random Forest AUC: ", new_auc, "\n")

# Plot the ROC curve for Random Forest
plot(new_prefrf, col = "red", main = "ROC Curves for Different Classifiers", xlab = "False
Positive Rate", ylab = "True Positive Rate")
abline(0, 1, col = "black")

# Add ROC curves for other models
plot(PDDperf, add = TRUE, col = "blue")
plot(PDBperf, add = TRUE, col = "blueviolet")
plot(PDBagperf, add = TRUE, col = "darkgreen")
plot(PDBoostperf, add = TRUE, col = "brown")

# Add legend to the ROC plot
legend("bottomright", legend = c(
  paste("Decision Tree (AUC =", round(PDDauc, 4), ")"),
  paste("Naive Bayes (AUC =", round(PDBauc, 4), ")"),
  paste("Bagging (AUC =", round(PDBagauc, 4), ")"),
  paste("Boosting (AUC =", round(PDBoostauc, 4), ")"),
  paste("Random Forest (AUC =", round(new_auc, 4), ")")
), col = c("blue", "blueviolet", "darkgreen", "brown", "red"), lty = 1, cex = 0.8)

# Create a comparison table
comparison_table = data.frame(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
  AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, new_auc),
  Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc)
)

cat("Comparison Table with Random Forest:\n")
print(comparison_table)
```

```
> # Predict using the existing Random Forest model on the test set
> PDpredrf = predict(PD.rf, PD.test, type = "class")
> random_conf = table(Predicted_Class = PDpredrf, Actual_Class = PD.test$Class)
> cat("\n# Random Forest Confusion Matrix\n")

# Random Forest Confusion Matrix
> print(random_conf)
               Actual_Class
Predicted_Class   0   1
              0 253  77
              1  33 105
> # Calculate accuracy
> random_sum = sum(random_conf)
> random_sum2 = sum(diag(random_conf))
> random_acc = (round(random_sum2 / random_sum, 4)) * 100
> cat("Random Forest Accuracy: ", random_acc, "%\n")
Random Forest Accuracy:  76.5 %
> # Calculate AUC for the existing Random Forest model
> PDpred_rf_prob = predict(PD.rf, PD.test, type = "prob")
> new_predrf = ROCR::prediction(PDpred_rf_prob[, 2], PD.test$Class)
> new_prefrf = performance(new_predrf, "tpr", "fpr")
> new_auc = performance(new_predrf, "auc")@y.values[[1]]
> cat("Random Forest AUC: ", new_auc, "\n")
Random Forest AUC:  0.7720837
> # Plot the ROC curve for Random Forest
> plot(new_prefrf, col = "red", main = "ROC Curves for Different Classifiers", xlab = "False
Positive Rate", ylab = "True Positive Rate")
> abline(0, 1, col = "black")
> # Add ROC curves for other models
> plot(PDDperf, add = TRUE, col = "blue")
> plot(PDBperf, add = TRUE, col = "blueviolet")
> plot(PDBagperf, add = TRUE, col = "darkgreen")
> plot(PDBoostperf, add = TRUE, col = "brown")
> # Add legend to the ROC plot
> legend("bottomright", legend = c(
+   paste("Decision Tree (AUC =", round(PDDauc, 4), ")"),
+   paste("Naive Bayes (AUC =", round(PDBauc, 4), ")"),
+   paste("Bagging (AUC =", round(PDBagauc, 4), ")"),
+   paste("Boosting (AUC =", round(PDBoostauc, 4), ")"),
+   paste("Random Forest (AUC =", round(new_auc, 4), ")")
+ ), col = c("blue", "blueviolet", "darkgreen", "brown", "red"), lty = 1, cex = 0.8)
> # Create a comparison table
> comparison_table = data.frame(
+   Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest"),
+   AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, new_auc),
+   Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc)
+ )
> cat("Comparison Table with Random Forest:\n")
Comparison Table with Random Forest:
> print(comparison_table)
     Classifier       AUC Accuracy
1 Decision Tree 0.7080708    73.72
2   Naive Bayes 0.6870821    40.38
3       Bagging 0.7483286    73.50
4      Boosting 0.7537463    72.86
5 Random Forest 0.7720837    76.50
```

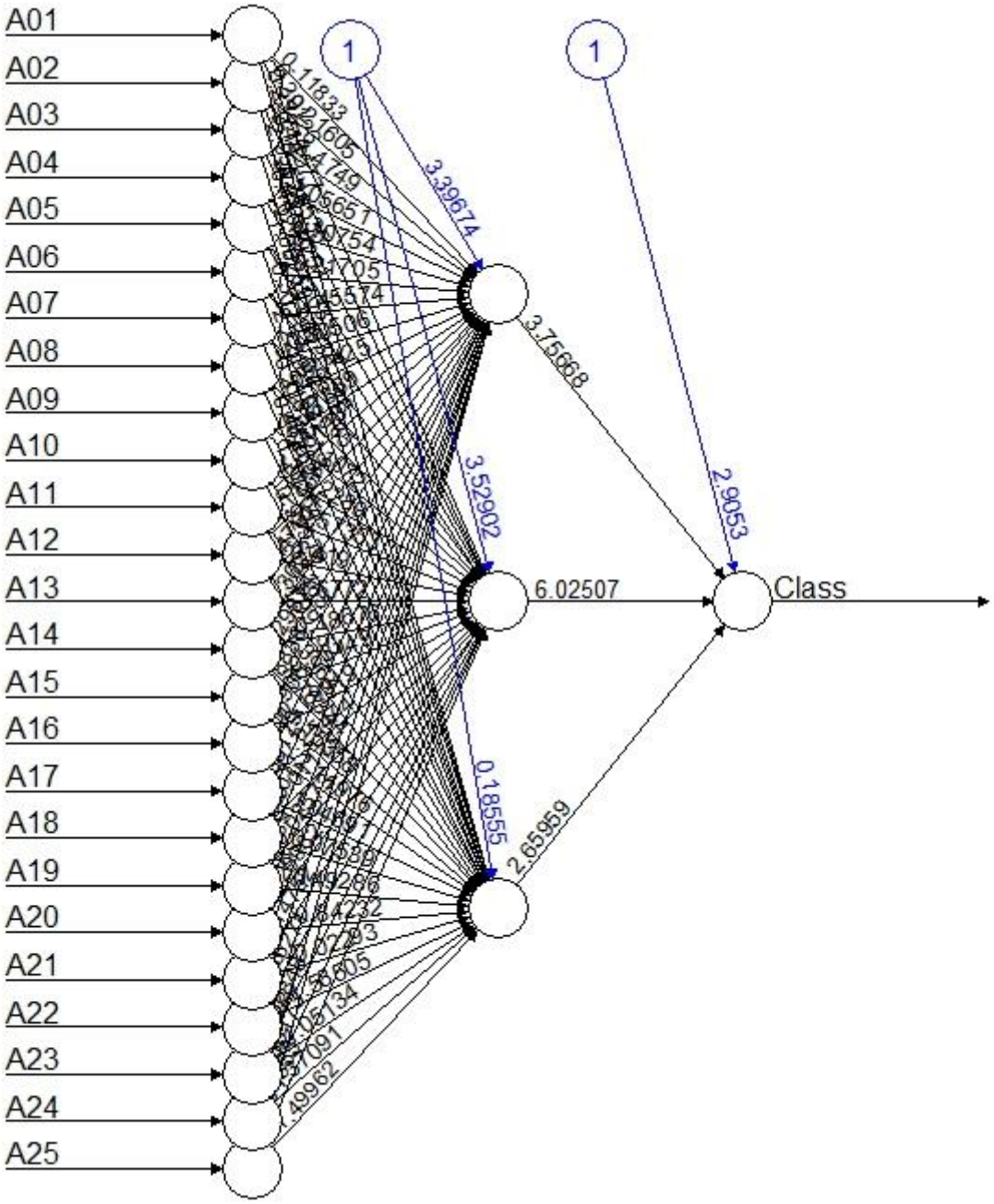


11.
```
#install.packages("neuralnet")
library(neuralnet)

options(digits = 4)

# Load the dataset
ann_PD = PD

# Ensure 'Class' is a factor and convert to numeric for ANN
ann_PD$Class = as.factor(ann_PD$Class)
ann_PD$Class = as.numeric(ann_PD$Class)

# Remove rows with missing values
ann_PD = ann_PD[complete.cases(ann_PD), ]

# Split the data into training and test sets
set.seed(33402302)
ind = sample(2, nrow(ann_PD), replace = TRUE, prob = c(0.7, 0.3))
ann_PD_train = ann_PD[ind == 1, ]
ann_PD_test = ann_PD[ind == 2, ]

# Scale the numeric features
preProc = preProcess(ann_PD_train[, -ncol(ann_PD_train)], method = c("center", "scale"))
ann_PD_train[, -ncol(ann_PD_train)] <- predict(preProc, ann_PD_train[, -ncol(ann_PD_train)])
ann_PD_test[, -ncol(ann_PD_test)] <- predict(preProc, ann_PD_test[, -ncol(ann_PD_test)])

# Train the ANN model using the neuralnet package
set.seed(33402302)
PD.nn = neuralnet(Class ~ ., data = ann_PD_train, hidden = 3, linear.output = FALSE)

# Plot the neural network
plot(PD.nn)

# Predict using the ANN model on the test set
pred_ann = compute(PD.nn, ann_PD_test[, -ncol(ann_PD_test)])
pred_ann_class = ifelse(pred_ann$net.result > 0.5, 1, 0)

# Create a confusion matrix
ann_conf = confusionMatrix(factor(pred_ann_class), factor(ann_PD_test$Class))
print(ann_conf)

# Calculate AUC
pred_ann_prob = pred_ann$net.result
ann_pred = ROCR::prediction(pred_ann_prob, ann_PD_test$Class)
ann_perf = performance(ann_pred, "tpr", "fpr")
ann_auc = performance(ann_pred, "auc")@y.values[[1]]
print(paste("AUC for ANN:", ann_auc))

# Comparison table
```

```
comparison_table = data.frame(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest",
"ANN"),
  AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc, ann_auc),
  Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc,
ann_conf$overall['Accuracy'] * 100)
)

print("Comparison Table with ANN:")
print(comparison_table)
```

```
> #install.packages("neuralnet")
> library(neuralnet)

Attaching package: 'neuralnet'

The following object is masked from 'package:dplyr':

    compute

The following object is masked from 'package:ROCR':

    prediction

> options(digits = 4)
> # Load the dataset
> ann_PD = PD
> # Ensure 'Class' is a factor and convert to numeric for ANN
> ann_PD$Class = as.factor(ann_PD$Class)
> ann_PD$Class = as.numeric(ann_PD$Class)
> # Remove rows with missing values
> ann_PD = ann_PD[complete.cases(ann_PD), ]
> # Split the data into training and test sets
> set.seed(33402302)
> ind = sample(2, nrow(ann_PD), replace = TRUE, prob = c(0.7, 0.3))
> ann_PD_train = ann_PD[ind == 1, ]
> ann_PD_test = ann_PD[ind == 2, ]
> # Scale the numeric features
> preProc = preProcess(ann_PD_train[, -ncol(ann_PD_train)], method = c("center", "scale"))
> ann_PD_train[, -ncol(ann_PD_train)] <- predict(preProc, ann_PD_train[, -ncol(ann_PD_trai
n)])
> ann_PD_test[, -ncol(ann_PD_test)] <- predict(preProc, ann_PD_test[, -ncol(ann_PD_test)])
> # Train the ANN model using the neuralnet package
> set.seed(33402302)
> PD.nn = neuralnet(Class ~ ., data = ann_PD_train, hidden = 3, linear.output = FALSE)
> # Plot the neural network
> plot(PD.nn)
> # Predict using the ANN model on the test set
> pred_ann = compute(PD.nn, ann_PD_test[, -ncol(ann_PD_test)])
> pred_ann_class = ifelse(pred_ann$net.result > 0.5, 1, 0)
> # Create a confusion matrix
> ann_conf = confusionMatrix(factor(pred_ann_class), factor(ann_PD_test$Class))
Warning message:
In confusionMatrix.default(factor(pred_ann_class), factor(ann_PD_test$Class)) :
  Levels are not in the same order for reference and data. Refactoring data to match.
> print(ann_conf)
Confusion Matrix and Statistics

          Reference
Prediction   1   2
         1 295 177
         2   0   0

               Accuracy : 0.625
                 95% CI : (0.58, 0.669)
    No Information Rate : 0.625
    P-Value [Acc > NIR] : 0.521

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.000
            Specificity : 0.000
         Pos Pred Value : 0.625
         Neg Pred Value :   NaN
             Prevalence : 0.625
         Detection Rate : 0.625
   Detection Prevalence : 1.000
      Balanced Accuracy : 0.500

       'Positive' Class : 1

> # Calculate AUC
> pred_ann_prob = pred_ann$net.result
> ann_pred = ROCR::prediction(pred_ann_prob, ann_PD_test$Class)
> ann_perf = performance(ann_pred, "tpr", "fpr")
> ann_auc = performance(ann_pred, "auc")@y.values[[1]]
> print(paste("AUC for ANN:", ann_auc))
[1] "AUC for ANN: 0.541415302116251"
> # Comparison table
> comparison_table = data.frame(
+   Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest", "A
NN"),
+   AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc, ann_auc),
+   Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc, ann_conf$overall['Accur
acy'] * 100)
```

```
+ )
> print("Comparison Table with ANN:")
[1] "Comparison Table with ANN:"
> print(comparison_table)
     Classifier    AUC Accuracy
1 Decision Tree 0.7081    73.72
2   Naive Bayes 0.6871    40.38
3       Bagging 0.7483    73.50
4      Boosting 0.7537    72.86
5 Random Forest 0.7721    76.50
6           ANN 0.5414    62.50
```



```
12. # Load necessary libraries
# install.packages("xgboost")
library(xgboost)

# Ensure 'Class' is a factor and convert to numeric for xgboost
PD$Class = as.factor(PD$Class)
PD$Class = as.numeric(PD$Class) - 1  # Convert to 0 and 1

# Split the data into training and test sets
set.seed(33402302)
train.target = sample(1:nrow(PD), 0.7 * nrow(PD))
xgb_train_set = PD[train.target, ]
xgb_test_set = PD[-train.target, ]

# Convert to matrix form, separating features and target
xgb_train_data = xgb.DMatrix(data = as.matrix(xgb_train_set[, -ncol(xgb_train_set)]), label =
xgb_train_set$Class)
xgb_test_data = xgb.DMatrix(data = as.matrix(xgb_test_set[, -ncol(xgb_test_set)]), label =
xgb_test_set$Class)

# Set parameters for XGBoost
params = list(
  objective = "binary:logistic",
  eval_metric = "auc",
  max_depth = 6,
  eta = 0.1,
  nthread = 2
)

# Train the model
xgb_model = xgb.train(
  params = params,
  data = xgb_train_data,
  nrounds = 100,
  watchlist = list(val = xgb_test_data),
  early_stopping_rounds = 10,
  verbose = 0
)

# Predict using the XGBoost model on the test set
pred_xgb = predict(xgb_model, xgb_test_data)
pred_xgb_class = ifelse(pred_xgb > 0.5, 1, 0)

# Create a confusion matrix
xgb_conf = confusionMatrix(factor(pred_xgb_class), factor(xgb_test_set$Class))
print(xgb_conf)

# Calculate AUC
xgb_pred = ROCR::prediction(pred_xgb, xgb_test_set$Class)
```

```
xgb_perf = performance(xgb_pred, "tpr", "fpr")
xgb_auc = performance(xgb_pred, "auc")@y.values[[1]]
print(paste("XGBoost AUC:", xgb_auc))

# Comparison table
comparison_table = data.frame(
  Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest",
"XGBoost"),
  AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc, xgb_auc),
  Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc, xgb_conf$overall['Accuracy'] *
100)
)

print("Comparison Table with XGBoost:")
print(comparison_table)
```

```
> # Load necessary libraries
> # install.packages("xgboost")
> library(xgboost)
> # Ensure 'Class' is a factor and convert to numeric for xgboost
> PD$Class = as.factor(PD$Class)
> PD$Class = as.numeric(PD$Class) - 1  # Convert to 0 and 1
> # Split the data into training and test sets
> set.seed(33402302)
> train.target = sample(1:nrow(PD), 0.7 * nrow(PD))
> xgb_train_set = PD[train.target, ]
> xgb_test_set = PD[-train.target, ]
> # Convert to matrix form, separating features and target
> xgb_train_data = xgb.DMatrix(data = as.matrix(xgb_train_set[, -ncol(xgb_train_set)]), label
= xgb_train_set$Class)
> xgb_test_data = xgb.DMatrix(data = as.matrix(xgb_test_set[, -ncol(xgb_test_set)]), label =
xgb_test_set$Class)
> # Set parameters for XGBoost
> params = list(
+   objective = "binary:logistic",
+   eval_metric = "auc",
+   max_depth = 6,
+   eta = 0.1,
+   nthread = 2
+ )
> # Train the model
> xgb_model = xgb.train(
+   params = params,
+   data = xgb_train_data,
+   nrounds = 100,
+   watchlist = list(val = xgb_test_data),
+   early_stopping_rounds = 10,
+   verbose = 0
+ )
> # Predict using the XGBoost model on the test set
> pred_xgb = predict(xgb_model, xgb_test_data)
> pred_xgb_class = ifelse(pred_xgb > 0.5, 1, 0)
> # Create a confusion matrix
> xgb_conf = confusionMatrix(factor(pred_xgb_class), factor(xgb_test_set$Class))
> print(xgb_conf)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
```

```
         0 258  79
         1  28 103

               Accuracy : 0.771
                 95% CI : (0.731, 0.809)
    No Information Rate : 0.611
    P-Value [Acc > NIR] : 1.21e-13

                  Kappa : 0.493

 Mcnemar's Test P-Value : 1.34e-06

            Sensitivity : 0.902
            Specificity : 0.566
         Pos Pred Value : 0.766
         Neg Pred Value : 0.786
             Prevalence : 0.611
         Detection Rate : 0.551
   Detection Prevalence : 0.720
      Balanced Accuracy : 0.734

       'Positive' Class : 0

> # Calculate AUC
> xgb_pred = ROCR::prediction(pred_xgb, xgb_test_set$Class)
> xgb_perf = performance(xgb_pred, "tpr", "fpr")
> xgb_auc = performance(xgb_pred, "auc")@y.values[[1]]
> print(paste("XGBoost AUC:", xgb_auc))
[1] "XGBoost AUC: 0.794810958272495"
> # Comparison table
> comparison_table = data.frame(
+   Classifier = c("Decision Tree", "Naive Bayes", "Bagging", "Boosting", "Random Forest", "X
GBoost"),
+   AUC = c(PDDauc, PDBauc, PDBagauc, PDBoostauc, PDFauc, xgb_auc),
+   Accuracy = c(tree_acc, naive_acc, bag_acc, boost_acc, random_acc, xgb_conf$overall['Accur
acy'] * 100)
+ )
> print("Comparison Table with XGBoost:")
[1] "Comparison Table with XGBoost:"
> print(comparison_table)
     Classifier    AUC Accuracy
1 Decision Tree 0.7081    73.72
2   Naive Bayes 0.6871    40.38
3       Bagging 0.7483    73.50
4      Boosting 0.7537    72.86
5 Random Forest 0.7721    76.50
6       XGBoost 0.7948    77.14
```