I. EXPLORE THE DATA

Proportion of phishing sites to legitimate sites

The proportion of phishing sites to legitimate sites is calculated using the 'Class' variable in the dataset (0 for phishing and 1 for legitimate). Therefore, the data contains 1183 legitimate sites (Class = 0) and 817 phishing sites (Class = 1), resulting in a proportion of 0.4085, or approximately 40.85%, of the sites being phishing.

Descriptions of the predictor (independent) variables - Observations

The analysis of the real-valued attributes reveals several key observations:

1. **Wide Range of Values:** The real-valued attributes cover a broad spectrum of numerical values, from small integers (e.g., A23) to very large integers (e.g., A18, A23). This suggests the potential presence of outliers that should be investigated further.
2. **Missing Values:** Several attributes have a significant number of missing values (NAs), ranging from 13 to 24 missing data points. This may need to be addressed during the data preprocessing stage, either through imputation or by considering the impact of missing data on the analysis.
3. **Distributional Characteristics:** The distributions of the real-valued attributes vary, with some being more skewed or kurtotic than others (e.g., A24 and A25). This information will be crucial in determining the appropriate data transformations or modelling techniques to be applied.

Omitting Attributes

Furthermore, in the initial stage of the analysis, it is recommended that no attempt is made to omit attributes from the dataset. At this point, we are unsure of which attributes would be considered significant to the classification models, and removing attributes prematurely could potentially lead to the loss of valuable information or predictive power.

However, if attributes were to be considered for omission at a later stage, the following criteria could be used:

1. **Attributes with a high number of missing values (NA's):** Attributes with a significant amount of missing data may introduce challenges during the modelling process and could be considered for omission.
2. **Attributes with lower variance:** Attributes that exhibit low variance, i.e., little variation in their values, are generally considered less informative for the classification task and could be considered for omission.

By retaining all attributes initially, we can further investigate their relevance and potential. Removing attributes prematurely could potentially lead to the loss of valuable information or predictive power, which could negatively impact the performance of the models.

II. PRE-PROCESSING

In this data set, complete.cases() is used to remove rows with missing values in any column of the data frame, this steps make it easier in later steps when classification models are applied. In terms of if we were to remove attributes that are not relevant to the model, the reasonable threshold is removing features where variance is equal to zero, so they are actually constant; or removing attributes that have a significant amount of NA values compared to all else.

III. CLASSIFIERS EVALUATION

A. CONFUSION MATRIX - ACCURACY

DECISION TREE

	Predicted Class			Accuracy
Actual Class		Class = 0	Class = 1	$Accuracy = \frac{225 + 128}{225 + 45 + 73 + 128} = 0.7495$
	Class = 0	225	45	Therefore, the accuracy of the Decision Tree Model is approximately 74.95%
	Class = 1	73	128	

NAIVE BAYES

	Predicted Class			Accuracy
Actual Class		Class = 0	Class = 1	$Accuracy = \frac{11 + 198}{11 + 259 + 3 + 198} = 0.4437$
	Class = 0	11	259	Therefore, the accuracy of the Naïve Bayes Model is approximately 44.37%
	Class = 1	3	198	

BAGGING

	Predicted Class			Accuracy
Actual Class		Class = 0	Class = 1	$Accuracy = \frac{231 + 129}{231 + 39 + 72 + 129} = 0.7643$
	Class = 0	231	39	Therefore, the accuracy of the Bagging Model is approximately 76.43%
	Class = 1	72	129	

BOOSTING

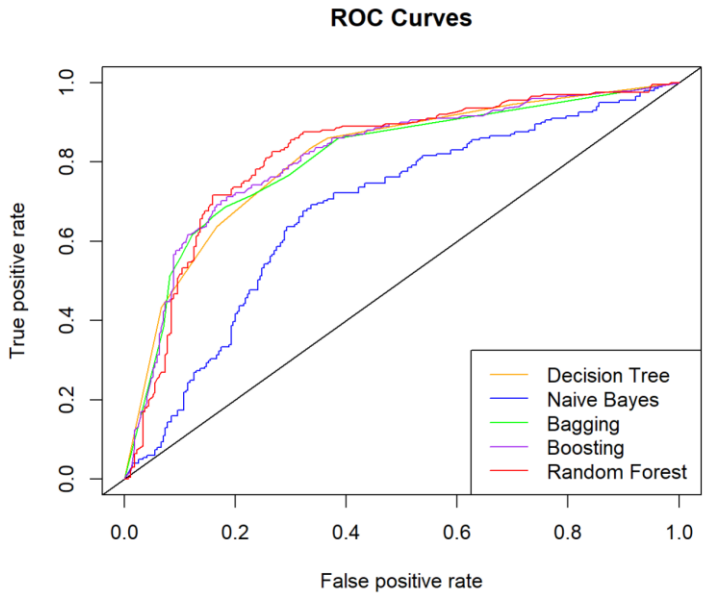
	Predicted Class			Accuracy
Actual Class		Class = 0	Class = 1	$Accuracy = \frac{227 + 135}{227 + 43 + 66 + 135} = 0.7686$
	Class = 0	227	43	

	Class = 1	66	135	Therefore, the accuracy of the Boosting Model is approximately 76.86%

RANDOM FOREST

	Predicted Class			Accuracy
Actual Class		Class = 0	Class = 1	$Accuracy = \frac{229 + 138}{229 + 41 + 63 + 138} = 0.7792$
	Class = 0	229	41	Therefore, the accuracy of the Random Forest Model is approximately 77.92%
	Class = 1	63	138	

B. ROC CURVES - AUC - COMPARISONS



Graph 6.a. ROC Curves of All Original Models

	Accuracy	AUC
Decision Tree	74.95%	81.15%
Naive Bayes	44.37%	68.00%
Bagging	76.43%	80.75%
Boosting	76.86%	81.44%

Random Forest	77.92%	81.98%

From the data given in the model, it is deemed that although most classifiers have approximately similar accuracy and AUC, except for the Naive Bayes Model, the Random Forest Model yields the highest accuracy and AUC, compared to other models. Overall, it can be observed that tree-based models would considered to have higher accuracy, as well as better performance compared to the Naive Bayes Model.

IV. ATTRIBUTES ANALYSIS

Upon examining the importance scores provided by each model and the summary of the decision tree model, we can determine the most important variables in predicting whether a website is phishing or legitimate, as well as the variables that could be omitted with little effect on performance.

The most important variables from each model

1. Decision Tree: A01, A23, A22, A18
2. Bagging: A01, A22, A23
3. Boosting: A01, A22, A23, A18
4. Random Forest: A01, A18, A22, A23

Overall, the most important variables in predicting whether a website will be phishing or legitimate are A01, A18, A22, and A23, regardless of the classification models. While these same variables are identified as important across models, their relative importance varies. For example, A01 is highly prioritised in Random Forest and Decision Tree but holds slightly less significance in other models. This discrepancy highlights the importance of considering multiple modelling techniques to gain a further understanding of variable importance.

Variables that could be omitted with little effect on performance

1. Decision Tree: The model only used 4 variables (A01, A23, A22, A18), so the remaining variables could potentially be omitted with little effect on performance.
2. Bagging: Variables A02, A03, A04, A05, A07, A09, A10, A11, A13, A15, A16, A17, A19, A20, A21, A25 have an importance score of 0, indicating they could be omitted with little effect on performance.
3. Boosting: Variables A03, A05, A07, A11, A13, A16, A21, and A25 have an importance score of 0, indicating they could be omitted with little effect on performance.
4. Random Forest: Variables A03, A05, A07, A13, and A25 have a relatively low importance score (Mean Decrease Gini < 1), suggesting they could potentially be omitted with little effect on performance.

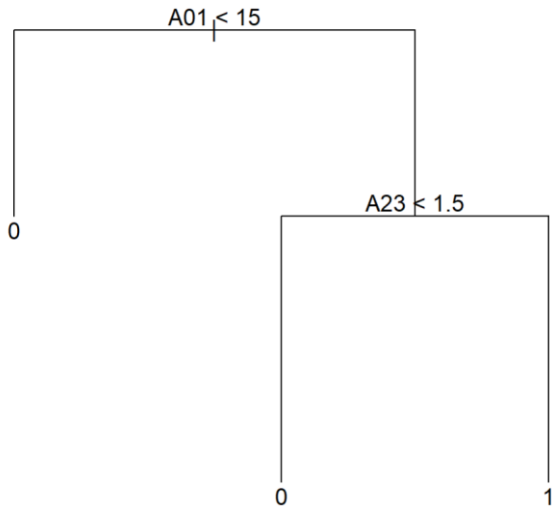
In summary, the most important variables in predicting whether a website is phishing or legitimate are A01, A18, A22, and A23. The variables that could be omitted with little effect on performance are those with low importance scores, such as A02, A03, A04, A05, A07, A09, A10, A11, A13, A15, A16, A17, A19, A20, A21, and A25. The reasons for omitting these variables are:

1. They do not contribute significantly to the model's predictive power, as their importance scores are low.
2. Removing these variables can simplify the model, potentially reducing the risk of overfitting.
3. Reducing the number of variables can also improve the model's efficiency and computational performance, especially when dealing with large datasets.

However, it's important to note that the decision to omit variables should be made with caution and after carefully evaluating the impact on the model's overall performance and the specific context of the problem being addressed.

V. SIMPLE CLASSIFIER

Firstly, cross-validation is performed on the Decision Tree model in search of the misclassifications in the original model. After the analysis, it can be observed that the deviations of the model are minimised at a tree size of 7 terminal nodes, however, when the tree size has only 3 terminal nodes, the classification performance only decreases by a slight difference (from 282 for 7 nodes to 312 for 3 nodes model). Therefore, starting with the decision tree model (PD_dt) developed in Question 4, the decision tree is pruned to create a simpler model using only 3 terminal nodes. The pruning process therefore involved considering only the A01 and A23 attributes as the basis for the classification decisions. The resulting simplified decision tree model is as follows:



The key steps in this simplified decision tree model are:

1. Check if the A01 attribute (the first attribute used in the split) is less than 15.
2. If A01 < 15, then classify the site as "legitimate" (Class = 0).
3. If A01 >= 15.5, then check if the A23 attribute is less than 1.5.
   - If A23 < 1.5, then classify the site as "legitimate" (Class = 0).
   - If A23 >= 1.5, then classify the site as "phishing" (Class = 1)

The factors that were important in the decision to create this simplified model were:

1. **Interpretability:** The pruned decision tree with only 3 terminal nodes and 2 attributes (A01 and A23) is much simpler and easier for a person to understand and apply manually compared to the original, more complex decision tree model.
2. **Performance Trade-off:** While the simplified model has a slightly lower accuracy (73.03%) and AUC (75.06%) compared to the original model, the decrease in performance is relatively small, and the improved interpretability may be worth the trade-off for simpler classification process for the user, which in this case, is done by hand.
3. **Feature Selection:** The A01 and A23 attributes were chosen because they were identified as important predictors in the original decision tree model. By focusing on these two attributes, the simplified model can still capture the key patterns in the data.

Using the test data created in Question 3, the performance of the simplified decision tree model has an accuracy of 73.03% and an AUC of 75.06%.

	Accuracy	**AUC**

Decision Tree	74.95%	81.15%
Naive Bayes	44.37%	68.00%
Bagging	76.43%	80.75%
Boosting	76.86%	81.44%
Random Forest	77.92%	81.98%
Pruned Decision Tree	73.03%	75.06%

As expected, the simplified decision tree model has a lower accuracy and AUC compared to the more complex models. However, the decrease in performance is relatively small, and the simplified model is much easier for a person to understand and apply by hand. The simplified decision tree model created is a good trade-off between interpretability and performance. It is simple enough for a person to classify sites as phishing or legitimate by hand, while still maintaining reasonable accuracy and AUC. The choice of the A01 and A23 attributes was based on their importance in the original decision tree model, and the pruning process was used to create a model with only 3 terminal nodes.

VI. BEST TREE-BASED CLASSIFIER

From prior analysis and calculations, it can be observed that the Random Forest classification model is currently having the best performance in terms of both overall accuracy and classification power (AUC and ROC). Therefore, we will be using this model to perform further tunning and cross-validation to create the best tree-based classifier to differentiate our test cases. These steps are performed to improve our current Random Forest model:

1. **Cross-Validation for Random Forest:**
   - Performing cross-validation on the Random Forest model using the rfcv() function.
   - This allows us to determine the optimal number of variables to include in the model (n.var and error.cv), based on the lowest test error.
   - The results show that the test error would be lowest when considering 3 to 6 variables, therefore, we would try to fit the classification model with different random variables varying from 3 to 6.
2. **Tuning the Random Forest Model:**
   - Defining a range of values for the mtry (number of variables randomly sampled as candidates at each split) and ntree (number of trees to grow) parameters to try and find the optimal hyperparameters for a model, as they can impact the model's performance. By looping through different combinations of mtry and ntree values, we are systematically exploring the parameter space to find the best-performing model.
3. **Evaluation Metrics:**
   - Using two key performance metrics to evaluate the models: accuracy and AUC (Area Under the Curve). Accuracy is a straightforward measure of the model's overall classification performance. At the same time, AUC provides a more comprehensive assessment of the model's ability to distinguish between the two classes (phishing and legitimate).
4. **Attribute Selection:**

- In the final Random Forest model, only a subset of the available features is used: A01, A02, A04, A06, A08, A09, A10, A11, A12, A14, A15, A16, A17, A18, A19, A20, A21, A22, A23, A24, as these features are considered as important features to the random forest model. Choosing these attributes can help improve the model's interpretability and potentially enhance its performance.

5. **Selecting the Best Model:**
   - Updating the "best" model parameters (mtry, ntree, accuracy, and AUC) whenever a model with a higher AUC is found.

By leveraging cross-validation, parameter tuning, and feature selection, a robust and effective model for predicting phishing websites is developed. The factors considered, such as the optimal number of split variables, the selection of the most important attributes, and the focus on maximising the AUC, are all crucial elements that contribute to the success of the model. After cross-validation and tuning the parameters of the Random Forest model with the objective of yielding the highest AUC (mtry = 6 and ntree = 200), we create a new model which has an Accuracy of 77.07% and an AUC of 82.22%

	Accuracy	**AUC**
Decision Tree	74.95%	81.15%
Naive Bayes	44.37%	68.00%
Bagging	76.43%	80.75%
Boosting	76.86%	81.44%
Random Forest	77.92%	81.98%
Pruned Decision Tree	73.03%	75.06%
Best Classifier (Random Forest)	77.07%	82.22%

The trade-off between accuracy and AUC is an important consideration when selecting the best model. Based on these findings, it seems that when trying to balance the accuracy and AUC, the accuracy improves, but the AUC remains relatively the same as the original model. On the other hand, when tuning the model to maximise the AUC, the accuracy slightly decreases. In this report, we will focus on optimising for the highest AUC, as the AUC is a more comprehensive metric that captures the model's ability to distinguish between the phishing and legitimate classes. Here's the rationale and factors that are taken into account:

1. **AUC as the Primary Metric:** The AUC (Area Under the Curve) provides a more holistic assessment of the model's performance, as it measures the model's ability to correctly rank the positive (phishing) and negative (legitimate) instances. This is particularly important in the context of phishing detection, where accurately identifying the true positives (phishing websites) is crucial.

2. **Accuracy vs. AUC Trade-off:** the model may be making more conservative predictions to improve its ability to distinguish between the two classes as we observed a decrease in accuracy when optimising for the highest AUC.
3. **Practical Implications:** In the context of phishing detection, it's generally more important to have a model that can reliably identify phishing websites (high AUC) than to have a model that maximises overall accuracy. Falsely classifying a legitimate website as phishing (false positive) may be less detrimental than failing to detect a phishing website (false negative).
4. **The AUC (Area Under the Curve) is a more comprehensive metric:** AUC captures the overall performance of the model, whereas accuracy is a point estimator that only provides a single performance measure. Accuracy is calculated as the proportion of correctly classified instances, which means it only considers the model's performance at a single decision threshold. In contrast, the AUC measures the model's ability to rank the positive (phishing) and negative (legitimate) instances correctly, across all possible decision thresholds.

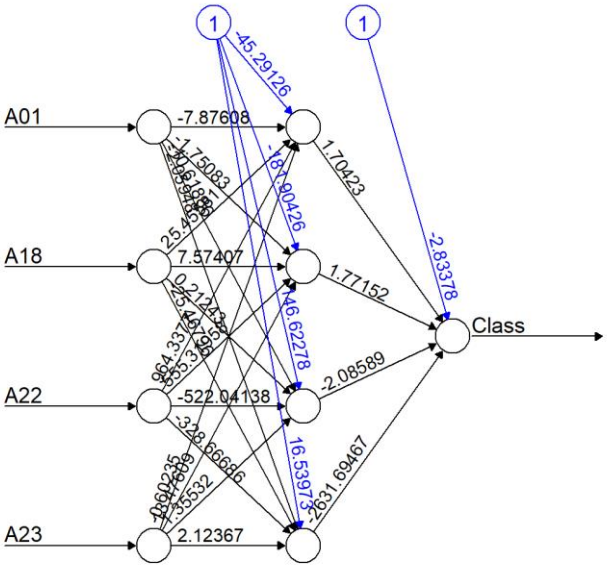
Therefore, the focus should be on the model that yields the highest AUC, even if it comes at the cost of a slight decrease in overall accuracy. This approach will ensure that the phishing detection system is optimised to accurately classify the true phishing websites, which is the primary goal in this context.

VII. ARTIFICIAL NEURAL NETWORK (ANN) CLASSIFIER

Data Preprocessing and Model Implementation

The data preprocessing steps for the ANN model are similar to the previous models. Firstly, the dataset is processed by removing any rows with missing values using the `complete.cases()` function. We then split the data into training and testing sets, with a 70:30 ratio, respectively.

For the ANN model, a subset of the most significant attributes is selected based on the insights gained from the previous analyses. Specifically, we chose the attributes A01, A18, A22, and A23 as the input variables for the neural network. These attributes were identified as important predictors in the earlier models, such as the Decision Tree and Random Forest. The ANN model is then implemented using the `neuralnet` library in R. The model was trained on the training set, with the number of hidden layers set to 4 and the `linear.output` parameter set to `FALSE` to perform a binary classification task.



Model Evaluation and Comparison

To evaluate the performance of the ANN model, the accuracy and the AUC is calculated on the test set. The confusion matrix for the ANN model showed an accuracy of 0.7325, which is comparable to the performance of the other models we have explored. This suggests that the ANN model is able to capture the underlying patterns in the data and make accurate predictions.

To further assess the model's performance, we calculated the AUC, which is a widely used metric for evaluating the overall classification performance of a model. The AUC for the ANN model was 0.7536, indicating a good level of discrimination between the positive and negative classes.

	Accuracy	**AUC**
Decision Tree	74.95%	81.15%
Naive Bayes	44.37%	68.00%
Bagging	76.43%	80.75%
Boosting	76.86%	81.44%
Random Forest	77.92%	81.98%
Pruned Decision Tree	73.03%	75.06%
Best Classifier (Random Forest)	77.07%	82.22%
ANN	73.25%	75.36%

When comparing the ANN model to the other classifiers we have implemented, we can observe the following:

1. **Naive Bayes:** The ANN model only outperformed the Naive Bayes classifier, which is a simpler and more assumptions-based model. The ANN's flexibility in modelling the underlying data distribution likely contributed to its superior performance.
2. **Decision Tree, Bagging, Boosting, and Random Forest:** The Decision Tree model and ensemble methods, such as Bagging, Boosting, and Random Forest, generally performed better than the ANN model in terms of both accuracy and AUC. These ensemble techniques are known for their ability to capture complex patterns and reduce overfitting, which may give them an advantage over the ANN model in this specific problem.

Reasons for the ANN Model's Performance

The slightly lower performance of the ANN model can be attributed to several factors:

1. **Model Complexity:** The ANN model's ability to capture complex non-linear relationships in the data may have given it an advantage over simpler models, such as Naive Bayes. The four hidden layers in the ANN

architecture allowed the model to learn intricate patterns in the data. However, the multiple hidden layers and the non-linear activation functions in the ANN make it challenging to interpret the model's internal workings and understand the specific relationships between the input features and the target variable. This lack of interpretability can be a disadvantage, especially when dealing with a relatively small dataset like the one in this problem.

2. **Overfitting:** The ANN model, with its high model complexity, may be more prone to overfitting, especially when the dataset is relatively small. The large number of parameters in the ANN can lead to the model memorising the training data rather than learning the true underlying patterns, which can result in poor generalisation of the test set.
3. **Data Characteristics:** In this specific problem, the ensemble methods, particularly Random Forest, may have been better suited to handle the high-dimensional feature space and identify the most relevant features for the classification task. The tree-based structure of these models allows them to effectively capture the complex interactions between the input variables, which may be more challenging for the ANN model to learn.

While the ANN model outperformed the simpler model, such as Naive Bayes, the ensemble methods, including Bagging, Boosting, and Random Forest, were able to achieve better results. This suggests that the ensemble techniques may be better suited for this specific problem, as they can effectively capture the complex patterns in the data and reduce overfitting. Overall, implementing the ANN model has provided valuable insights into the performance of different classifiers and the importance of feature selection and data preprocessing in the context of phishing email classification.

VIII. XGBOOST CLASSIFIER

XGBoost Model

For this task, the XGBoost[1] classifier is chosen to be implemented, using the 'xgboost' package in R. XGBoost is a tree-based ensemble learning algorithm that combines multiple weak decision trees to create a strong predictive model. The algorithm works by building new trees that aim to correct the errors made by the previous trees, resulting in a powerful and robust classifier. To implement the XGBoost model, the Class variable in the PD_cleaned dataset is recoded to be 0 and 1, representing the legitimate and phishing classes, respectively. The data is then split into training and test sets, using the same 70/30 split as in the previous models.

The XGBoost model was trained using the following hyperparameters:

1. max_depth = 3: The maximum depth of the trees, which controls the complexity of the model.
2. eta = 0.1: The learning rate, which determines the step size of the gradient descent optimisation.
3. nround = 100: The number of boosting iterations, which controls the number of trees in the ensemble.

XGBoost Model Details

The key features of XGBoost that make it a suitable choice for the phishing detection problem are:

1. **Handling of Heterogeneous Data:** XGBoost can handle a wide range of data types, including numerical, categorical, and sparse features, without the need for extensive feature engineering.
2. **Robustness to Overfitting:** XGBoost employs several regularisation techniques, such as L1/L2 regularisation and tree pruning, to prevent overfitting and improve the model's generalisation performance.
3. **Efficient Computation:** XGBoost is [1]designed to be highly efficient, with fast training times and low memory usage, making it suitable for large-scale datasets.

[1] CRAN.R, *xgboost : Extreme Gradient Boosting*, https://CRAN.R-project.org/package=xgboost

The XGBoost algorithm works by iteratively building an ensemble of decision trees, where each new tree is trained to correct the errors made by the previous trees. This process continues until the desired level of performance is achieved or the maximum number of trees is reached.

Performance Evaluation

Using the test data, the performance of the XGBoost model is as follows:

	Accuracy	AUC
Decision Tree	74.95%	81.15%
Naive Bayes	44.37%	68.00%
Bagging	76.43%	80.75%
Boosting	76.86%	81.44%
Random Forest	77.92%	81.98%
Pruned Decision Tree	73.03%	75.06%
Best Classifier (Random Forest)	76.86%	82.22%
ANN	73.25%	75.36%
XGBoost	77.71%	82.89%

Compared to the other models evaluated in the previous questions, the XGBoost model demonstrates a strong performance, with an accuracy of 77.71% and an AUC of 82.89%. This places the XGBoost model among the top-performing classifiers in the comparison, suggesting that it is a suitable choice for the website classification task. The high AUC value indicates that the XGBoost model is effective at distinguishing between phishing and legitimate websites, which is a crucial requirement for practical phishing detection applications.

In this report, the implementation and evaluation of the XGBoost classifier for the task of phishing website detection is presented. XGBoost is a powerful machine learning algorithm that has shown excellent performance on a wide range of classification problems, and it has proven to be a strong contender in the context of this phishing detection task. The XGBoost model has outperformed several of the other classifiers evaluated earlier. This suggests that XGBoost is a suitable choice for this problem and could be a valuable addition to website detection tools.

APPENDIX

R CODES

```
setwd("~/MONASH/SEM 01-2024/FIT3152/Assignment 2")

rm(list = ls())
Phish <- read.csv("PhishingData.csv")
set.seed(33295379)

L <- as.data.frame(c(1:50))
L <- L[sample(nrow(L), 10, replace = FALSE),]
Phish <- Phish[(Phish$A01 %in% L),]
PD <- Phish[sample(nrow(Phish), 2000, replace = FALSE),]
PD$Class <- as.factor(PD$Class)
unique(PD$Class)


# QUESTIONS
# (1) Exploring the data set
library(dplyr)
glimpse(PD)
table(PD$Class)
summary(PD)



# (2) Pre-processing
# Omitting the NA values from the data set with complete.cases
PD_cleaned <- PD[complete.cases(PD), ]



# (3) Splitting the data
set.seed(33295379)
train.row <-  sample(1:nrow(PD_cleaned), 0.7*nrow(PD_cleaned))
PD.train <-  PD_cleaned[train.row,]
PD.test <-  PD_cleaned[-train.row,]




# (4) Implement classification models
# Decision Tree
library(tree)
PD_dt <- tree(Class ~ ., data=PD.train)
summary(PD_dt)
PD_dt_predict <- predict(PD_dt, PD.test, type = "class")
# Plot
plot(PD_dt, main="Decision Tree Model")
text(PD_dt, pretty = 0)

#Naïve Bayes
library(e1071)
PD_nb <- naiveBayes(Class ~ ., data = PD.train)
PD_nb_predict <- predict(PD_nb, PD.test, type = "class")

# Bagging
library(adabag)
```

```
library(rpart)
PD_bag <- bagging(Class ~ ., data = PD.train, mfinal = 10)
PD_bag_predict <- predict.bagging(PD_bag, newdata = PD.test)
# Plot
plot(PD_bag$trees[[1]])
text(PD_bag$trees[[1]], pretty = 0)

# Boosting
library(rpart)
PD_boost <- boosting(Class ~ ., data = PD.train, mfinal = 10)
PD_boost_predict <- predict.boosting(PD_boost, newdata = PD.test)

# Random Forest
library(randomForest)
PD_rf <- randomForest(Class ~ ., data = PD.train, na.action = na.exclude)
PD_rf_predict <- predict(PD_rf, PD.test)
plot(PD_rf)
```

```
# (5) Evaluating the models (Confusion Matrix - Accuracy)
# Decision Tree
# Confusion Matrix
t_dt <- table(actual = PD.test$Class, predicted = PD_dt_predict)
t_dt_df <- as.data.frame.matrix(t_dt)
print(t_dt_df)
# Accuracy
accuracy_dt <- sum(diag(as.matrix(t_dt))) / nrow(PD.test)
print(paste("Decision Tree Accuracy:", accuracy_dt))

# Naïve Bayes
# Confusion Matrix
t_nb <- table(actual = PD.test$Class, predicted = PD_nb_predict)
t_nb_df <- as.data.frame.matrix(t_nb)
print(t_nb_df)
# Accuracy
accuracy_nb <- sum(diag(as.matrix(t_nb))) / nrow(PD.test)
print(paste("Naïve Bayes Accuracy:", accuracy_nb))

# Bagging
# Confusion Matrix
t_bag <- as.matrix(PD_bag_predict$confusion)
t_bag_df <- as.data.frame.matrix(t_bag)
print(t_bag_df)
# Accuracy
accuracy_bag <- sum(diag(t_bag)) / nrow(PD.test)
print(paste("Bagging Accuracy:", accuracy_bag))

# Boosting
# Confusion Matrix
t_boost <- as.matrix(PD_boost_predict$confusion)
t_boost_df <- as.data.frame.matrix(t_boost)
print(t_boost_df)
# Accuracy
accuracy_boost <- sum(diag(t_boost)) / nrow(PD.test)
print(paste("Boosting Accuracy:", accuracy_boost))
```

```
# Random Forest
# Confusion Matrix
t_rf <- table(observed = PD.test$Class, predicted = PD_rf_predict)
t_rf_df <- as.data.frame.matrix(t_rf)
print(t_rf_df)
# Accuracy
accuracy_rf <- sum(diag(as.matrix(t_rf))) / nrow(PD.test)
print(paste("Random Forest Accuracy:", accuracy_rf))



# (6) Constructing ROC curves - AUC
library(ROCR)
# Decision Tree
# ROC
PD_dt_probs <- predict(PD_dt, PD.test, type = "vector")[,"1"]
PD_dt_pred <- prediction(PD_dt_probs, PD.test$Class)
PD_dt_perf <- performance(PD_dt_pred, "tpr", "fpr")
plot(PD_dt_perf, main="ROC Curves", col="orange")
abline(0,1)
# AUC
PD_dt_auc <- performance(PD_dt_pred, "auc")
print(as.numeric(PD_dt_auc@y.values))


# Naïve Bayes
# ROC
PD_nb_probs <- predict(PD_nb, PD.test, type = "raw")[,"1"]
PD_nb_pred <- prediction(PD_nb_probs, PD.test$Class)
PD_nb_perf <- performance(PD_nb_pred, "tpr", "fpr")
plot(PD_nb_perf, add=TRUE, col="blue")
# AUC
PD_nb_auc <- performance(PD_nb_pred, "auc")
print(as.numeric(PD_nb_auc@y.values))


#Bagging
# ROC
PD_bag_pred <- prediction(PD_bag_predict$prob[,2], PD.test$Class)
PD_bag_perf <- performance(PD_bag_pred, "tpr", "fpr")
# Plot ROC curve
plot(PD_bag_perf, add=TRUE, col="green")
# AUC
PD_bag_auc <- performance(PD_bag_pred, "auc")
print(as.numeric(PD_bag_auc@y.values))


# Boosting
# ROC
PD_boost_pred <- prediction(PD_boost_predict$prob[,2], PD.test$Class)
PD_boost_perf <- performance(PD_boost_pred, "tpr", "fpr")
# Plot ROC curve
plot(PD_boost_perf, add=TRUE, col="purple")
# AUC
PD_boost_auc <- performance(PD_boost_pred, "auc")
print(as.numeric(PD_boost_auc@y.values))


# Random Forest
```

```
# ROC
PD_rf_probs <- predict(PD_rf, PD.test, type = "prob")[,"1"]
PD_rf_pred <- prediction(PD_rf_probs, PD.test$Class)
PD_rf_perf <- performance(PD_rf_pred, "tpr", "fpr")
plot(PD_rf_perf, add=TRUE, col="red")
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging",
"Boosting", "Random Forest"), col = c("orange", "blue", "green", "purple",
"red"), lty = c(1,1))
# AUC
PD_rf_auc <- performance(PD_rf_pred, "auc")
print(as.numeric(PD_rf_auc@y.values))



#(8) Attributes' performance evaluation
# Decision Tree
print(summary(PD_dt))
# Bagging
print(PD_bag$importance)
# Boosting
print(PD_boost$importance)
# Random Forest
print(PD_rf$importance)



# (9) Simplify a Classification Model
# Cross validation to check for misclassifications
cvdttest <- cv.tree(PD_dt, FUN = prune.misclass)
cvdttest
# Pruning the decision tree to 3 terminal nodes
prune_PD_dt <- prune.misclass(PD_dt, best = 3)
plot(prune_PD_dt)
text(prune_PD_dt, pretty = 0)

summary(prune_PD_dt)

PD_prune_dt_predict <- predict(prune_PD_dt, PD.test, type = "class")
t_prune <- table(actual = PD.test$Class, predicted = PD_prune_dt_predict)
# Accuracy
sum(diag(as.matrix(t_prune)))/nrow(PD.test)
# AUC
PD_dt_prune_prob <- predict(prune_PD_dt, PD.test, type = "vector")[,"1"]
PD_dt_prune_pred <- prediction(PD_dt_prune_prob, PD.test$Class)
PD_dt_prune_auc <- performance(PD_dt_prune_pred, "auc")
print(as.numeric(PD_dt_prune_auc@y.values))



# (10) Create the best tree-based model
# Cross-validation for Random Forest
PD_rf_cv_test   <-   rfcv(as.matrix(PD.train[,   -which(names(PD.train)   ==
"Class")]),
                     PD.train$Class,
                     ntreeTry = 500,
                     trace = FALSE)
```

```
PD_rf_cv_test$n.var
PD_rf_cv_test$error.cv
# Through cross-validation, it can be obsereved that the test error would be
# lowest when considering from 3 to 6 variables (the test error for 12 to 25
is low
# but would cause overfitting when applied to the model)


# Define the range of mtry (number of variables randomly sampled as
# candidates at each split) and ntree (number of trees) values to try
mtry_values <- seq(2, 6, by = 1)
ntree_values <- seq(200, 500, by = 50)

# Initialize variables to store the best accuracy, AUC and its corresponding
mtry value
best_accuracy <- 0
best_mtry <- NULL
best_auc <- 0
best_ntree <- NULL

# Loop over each mtry value and ntree value
for (mtry in mtry_values) {
  for (ntree in ntree_values) {
    set.seed(33295379)

    # Train Random Forest model with cross-validation
    rf_model <- randomForest(Class ~ A01 + A02 + A04 + A06 + A08 + A09 + A10 +
A11 +
                               A12 + A14 + A15 + A16 + A17 + A18 + A19 + A20 +
A21 +
                               A22 + A23 + A24, data = PD.train, mtry = mtry,
ntree = ntree, importance = TRUE)
    rf_predictions <- predict(rf_model, PD.test)
    conf_matrix_rf   <-   table(observed   =   PD.test$Class,   predicted   =
rf_predictions)

    # Calculate accuracy
    accuracy <- sum(diag(conf_matrix_rf)) / nrow(PD.test)

    # Calculate AUC
    rf_probs <- predict(rf_model, PD.test, type = "prob")[, 2]
    rf_pred <- prediction(rf_probs, PD.test$Class)
    rf_aucs <- performance(rf_pred, "auc")
    auc <- as.numeric(rf_aucs@y.values)

    # Update best accuracy, mtry, ntree, and AUC if current model performs
better
    #(taken AUC into consideration)
    if (auc > best_auc) {
      best_accuracy <- accuracy
      best_mtry <- mtry
      best_ntree <- ntree
      best_auc <- auc
    }
  }
}

# Print the best mtry, ntree values, and corresponding accuracy
cat("Best mtry value:", best_mtry, "\n")
cat("Best ntree value:", best_ntree, "\n")
```

```
cat("Corresponding highest accuracy:", best_accuracy, "\n")
cat("Corresponding highest AUC:", best_auc, "\n")

# Best Classification Model - Random Forest
# With mtry = 4 and ntree = 350 (Highest Accuracy AND AUC)
set.seed(33295379)
final_PD_rf <- randomForest(Class ~ A01 + A02 + A04 + A06 + A08 + A09 + A10 +
A11 +
                              A12 + A14 + A15 + A16 + A17 + A18 + A19 + A20 +
A21 +
                              A22 + A23 + A24, data = PD.train, mtry = 6, ntree
= 350, importance = TRUE)
final_rf_pred <- predict(final_PD_rf, PD.test)
conf_matrix_final <- table(observed = PD.test$Class, predicted = final_rf_pred)
# Calculate accuracy
accuracy <- sum(diag(conf_matrix_final)) / nrow(PD.test)
accuracy
#AUC
final_rf_probs <- predict(final_PD_rf, PD.test, type = "prob")[, 2]
final_rf_pred <- prediction(final_rf_probs, PD.test$Class)
final_rf_aucs <- performance(final_rf_pred, "auc")
auc <- as.numeric(final_rf_aucs@y.values)
auc

# With mtry = 6 and ntree = 200 (Highest AUC)
set.seed(33295379)
final_PD_rf_bestAUC <- randomForest(Class ~ A01 + A02 + A04 + A06 + A08 + A09
+ A10 + A11 +
                              A12 + A14 + A15 + A16 + A17 + A18 + A19 + A20 +
A21 +
                              A22 + A23 + A24, data = PD.train, mtry = 6, ntree
= 200, importance = TRUE)
final_rf_pred_bestAUC <- predict(final_PD_rf_bestAUC, PD.test)
conf_matrix_final_bestAUC <- table(observed = PD.test$Class, predicted =
final_rf_pred_bestAUC)
# Calculate accuracy
accuracy <- sum(diag(conf_matrix_final_bestAUC)) / nrow(PD.test)
accuracy
#AUC
final_rf_probs_bestAUC <- predict(final_PD_rf_bestAUC, PD.test, type =
"prob")[, 2]
final_rf_pred_bestAUC <- prediction(final_rf_probs_bestAUC, PD.test$Class)
final_rf_aucs_bestAUC <- performance(final_rf_pred_bestAUC, "auc")
auc <- as.numeric(final_rf_aucs_bestAUC@y.values)
auc




#(11)
library(neuralnet)
library(ROCR)
set.seed(33295379)
options (digits = 4)
PD_cleaned <- PD[complete.cases(PD),]
PD_cleaned$Class <- as.numeric(PD_cleaned$Class) - 1
unique(PD_cleaned$Class)

train.row <- sample(1:nrow(PD_cleaned), 0.7*nrow(PD_cleaned))
```

```
PD.train <-  PD_cleaned[train.row,]
PD.test <-  PD_cleaned[-train.row,]

# Build the neural net using the significant attributes (A01, A18, A22, A23)
PD_nn <- neuralnet(Class ~ A01 + A18 + A22 + A23, PD.train, hidden = 4,
linear.output = FALSE)

# Test the neural net
PD.pred.nn <- compute(PD_nn, PD.test[c(1, 18, 22, 23)])
PD.pred.nn_probs <- PD.pred.nn$net.result
# Convert the probabilities to binary predictions
PD.pred.nn_class <- as.data.frame(round(PD.pred.nn_probs, 0))

# Confusion Matrix - Accuracy
t_ANN <- table(observed = PD.test$Class, predicted = PD.pred.nn_class$V1)
t_ANN_df <- as.data.frame.matrix(t_ANN)
print(t_ANN_df)
# Accuracy
accuracy_ANN <- sum(diag(as.matrix(t_ANN)))/nrow(PD.test)
print(paste("ANN Accuracy:", accuracy_ANN))

# Calculate the AUC
PD_nn_pred <- prediction(PD.pred.nn_probs, PD.test$Class)
PD_nn_perf <- performance(PD_nn_pred, "tpr", "fpr")
PD_nn_auc <- performance(PD_nn_pred, "auc")
PD_nn_auc <- as.numeric(PD_nn_auc@y.values)
cat("ANN AUC:", PD_nn_auc, "\n")

# ANN Model Plot
plot(PD_nn)



# (12)
# Choose a new type of classifier or a new version of a studied classifier
# Install and load the required package(s)
library(xgboost)
library(ROCR)
# Split the data again
set.seed(33295379)
PD_cleaned <- PD[complete.cases(PD),]
PD_cleaned$Class <- as.numeric(PD_cleaned$Class) - 1
unique(PD_cleaned$Class)
train.row <- sample(1:nrow(PD_cleaned), 0.7*nrow(PD_cleaned))

PD.train <- PD_cleaned[train.row,]
PD.test <- PD_cleaned[-train.row,]

# Implement the new classifier and evaluate its performance
xgb_model <- xgboost(data = as.matrix(PD.train[, -which(names(PD.train) ==
"Class")]),
                     label = PD.train$Class,
                     max.depth = 3,
                     eta = 0.1,
                     nround = 100,
                     objective = "binary:logistic")

xgb_pred <- predict(xgb_model, as.matrix(PD.test[, -which(names(PD.test) ==
"Class")]))
```

```
xgb_pred_class <- as.integer(xgb_pred > 0.5)

# Create the confusion matrix
xgb_conf_matrix <- table(actual = PD.test$Class, predicted = xgb_pred_class)
# Accuracy
xgb_acc <- sum(diag(xgb_conf_matrix)) / nrow(PD.test)

# Calculate the AUC and ROC
xgb_pred_obj <- prediction(xgb_pred, PD.test$Class)
xgb_perf <- performance(xgb_pred_obj, "tpr", "fpr")
plot(xgb_perf, add = TRUE, col="hotpink")
legend("bottomright", legend = c("Decision Tree", "Naive Bayes", "Bagging",
"Boosting", "Random Forest", "XGBoost"), col = c("orange", "blue", "green",
"purple", "red","hotpink"), lty = c(1,1))
xgb_auc <- performance(xgb_pred_obj, "auc")
xgb_auc <- as.numeric(xgb_auc@y.values)

cat("XGBoost Accuracy:", xgb_acc, "\n")
cat("XGBoost AUC:", xgb_auc, "\n")
print(xgb_conf_matrix)
```